参考：<https://www.iteblog.com/archives/1008.html>

# [Apache Avro使用入门指南](https://www.iteblog.com/archives/1008.html)

[Hadoop](https://www.iteblog.com/archives/category/hadoop/)  2014-04-09 00:04:03 35692  [4评论](https://www.iteblog.com/archives/1008.html#comments) [下载为PDF](https://www.iteblog.com/pdf/1008.pdf)

　　Avro有C, C++, C#, Java, PHP, Python, and Ruby等语言的实现，本文只简单介绍如何在Java中使用Avro进行数据的序列化（data serialization）。本文使用的是Avro 1.7.4，这是写这篇文章时最新版的Avro。读完本文，你将会学到如何使用Avro编译模式、如何用Avro序列化和反序列化数据。

**文章目录**

* [1 一、准备项目需要的jar包](https://www.iteblog.com/archives/1008.html#jar)
* [2 二、定义模式(Schema)](https://www.iteblog.com/archives/1008.html#Schema)
* [3 三、编译模式](https://www.iteblog.com/archives/1008.html#i)
* [4 四、如何使用](https://www.iteblog.com/archives/1008.html#i-2)
* [5 五、一个完整的例子](https://www.iteblog.com/archives/1008.html#i-3)

#### 一、准备项目需要的jar包

本文的例子需要用到的Jar包有这四个：

avro-1.7.1.jar

avro-tools-1.7.4.jar

jackson-core-asl-1.8.8.jar

jackson-mapper-asl-1.8.8.jar

请先将这几个jar包下载好，并存放在一个地方（本文是存放在$HIVE\_HOME/lib目录中）。如果你是用Maven，你可以在你项目的pom.xml文件中加入以下依赖：

<dependency>

<groupId>org.apache.avro</groupId>

<artifactId>avro</artifactId>

<version>1.7.4</version>

</dependency>

<plugin>

<groupId>org.apache.avro</groupId>

<artifactId>avro-maven-plugin</artifactId>

<version>1.7.4</version>

<executions>

<execution>

<phase>generate-sources</phase>

<goals>

<goal>schema</goal>

</goals>

<configuration>

<sourceDirectory>${project.basedir}/src/main/avro/</sourceDirectory>

<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

　　当然，如果你需要，你也可以在Avro源码中进行编译，获取avro-1.7.1.jar和avro-tools-1.7.4.jar。关于如何编译avro已经超出本文的范围。

#### 二、定义模式(Schema)

　　在avro中，它是用Json格式来定义模式（schema）的。模式可以由基础类型（null, boolean, int, long, float, double, bytes, and string）和复制类型(record, enum, array, map, union, and fixed)的数据组成。本文只是定义了一个简单的模式user.avsc：

{

"namespace": "example.avro",

"name": "User",

"type": "record",

"fields": [

{

"name": "name",

"type": "string"

},

{

"name": "favorite\_number",

"type": [

"int",

"null"

]

},

{

"name": "favorite\_color",

"type": [

"string",

"null"

]

}

]

}

　　上面的模式是定义了一个用户的记录，在模式定义中，必须包含它的类型("type": "record")、一个名字("name": "User")以及fields。在本例中fields包括了name, favorite\_number和favorite\_color，上面的模式我们还定义了一个命名空间 ("namespace": "example.avro"),namespace可与名字一起使用，从而组成模式的全名（本例为example.avro.User）。

#### 三、编译模式

　　Avro可以允许我们根据模式的定义而生成相应的类，一旦我们定义好相关的类，我们程序中就不需要直接使用模式了。可以用avro-tools.jar包来生成代码，语法如下：

java -jar $HIVE\_HOME/lib/avro-tools-1.7.4.jar

compile schema

<schema file> <destination>

所以，在本例中我们可以这样来使用

java -jar $HIVE\_HOME/lib/avro-tools-1.7.4.jar compile schema user.avsc .

　　这时候，在当前目录下会生成example/avro/User.java类，细心的读者可能会发现example/avro不就是模式定义中的namespace么？的确是的(me：即包名)。

　　如果你直接用Avro Maven plugin，那么你就不需要手动的编译模式，因为Avro Maven plugin会自动给你编译好。

　　现在我们已经生成好了一个User.java类，我们就可以用代码生成User对象，并用avro将它序列化存放到本地文件中，最后我们再将其反序列化。

#### 四、如何使用

我们可以用下面的代码生成几个User:

User user1 = new User();

user1.setName("Alyssa");

user1.setFavoriteNumber(256);

// Leave favorite color null

// Alternate constructor

User user2 = new User("Ben", 7, "red");

// Construct via builder

User user3 = User.newBuilder()

.setName("Charlie")

.setFavoriteColor("blue")

.setFavoriteNumber(null)

.build();

从上面的列子中，我们可以看出，我们可以调用User的构造函数或者是builder来获取一个User实例。下面对上述的几个User进行序列化操作，并将序列化的数据存放到users.avro文件中:

// Serialize user1 and user2 to disk

File file = new File("users.avro");

DatumWriter<User> userDatumWriter = new SpecificDatumWriter<User>(User.class);

DataFileWriter<User> dataFileWriter = new DataFileWriter<User>(userDatumWriter);

dataFileWriter.create(user1.getSchema(), new File("users.avro"));

dataFileWriter.append(user1);

dataFileWriter.append(user2);

dataFileWriter.append(user3);

dataFileWriter.close();

运行完这个代码之后，将会在磁盘产生users.avro文件，里面是用avro序列化user的数据。我们可以对其进行反序列化，获取到原来的数据：

// Deserialize Users from disk

DatumReader<User> userDatumReader = new SpecificDatumReader<User>(User.class);

DataFileReader<User> dataFileReader =

new DataFileReader<User>(file, userDatumReader);

User user = null;

while (dataFileReader.hasNext()) {

// Reuse user object by passing it to next(). This saves us from

// allocating and garbage collecting many objects for files with

// many items.

user = dataFileReader.next(user);

System.out.println(user);

}

这段代码将会产生成以下的结果（me：json格式）：

{"name": "Alyssa", "favorite\_number": 256, "favorite\_color": null}

{"name": "Ben", "favorite\_number": 7, "favorite\_color": "red"}

{"name": "Charlie", "favorite\_number": null, "favorite\_color": "blue"}

#### 五、一个完整的例子

import java.io.\*;

import java.lang.\*;

import org.apache.avro.io.DatumWriter;

import org.apache.avro.io.DatumReader;

import org.apache.avro.specific.SpecificDatumWriter;

import org.apache.avro.specific.SpecificDatumReader;

import org.apache.avro.file.DataFileWriter;

import org.apache.avro.file.DataFileReader;

import example.avro.User;

public class Test {

public static void main(String args[]) {

User user1 = new User();

user1.setName("Alyssa");

user1.setFavoriteNumber(256);

// Leave favorite color null

// Alternate constructor

User user2 = new User("Ben", 7, "red");

// Construct via builder

User user3 = User.newBuilder()

.setName("Charlie")

.setFavoriteColor("blue")

.setFavoriteNumber(null)

.build();

//Serialize user1, user2 and user3 to disk

File file = new File("users.avro");

DatumWriter<User> userDatumWriter =

new SpecificDatumWriter<User>(User.class);

DataFileWriter<User> dataFileWriter =

new DataFileWriter<User>(userDatumWriter);

try {

dataFileWriter.create(user1.getSchema(), new File("users.avro"));

dataFileWriter.append(user1);

dataFileWriter.append(user2);

dataFileWriter.append(user3);

dataFileWriter.close();

} catch (IOException e) {

}

//Deserialize Users from dist

DatumReader<User> userDatumReader =

new SpecificDatumReader<User>(User.class);

DataFileReader<User> dataFileReader = null;

try {

dataFileReader = new DataFileReader<User>(file, userDatumReader);

} catch (IOException e) {

}

User user = null;

try {

while (dataFileReader.hasNext()) {

// Reuse user object by passing it to next(). This saves

// us from allocating and garbage collecting many objects for

// files with many items.

user = dataFileReader.next(user);

System.out.println(user);

}

} catch (IOException e) {

}

}

}

编译上述代码：

javac -classpath /home/q/hive-0.11.0/lib/avro-1.7.1.jar

:/home/q/hive-0.11.0/lib/avro-tools-1.7.4.jar

:/home/q/hive-0.11.0/lib/jackson-core-asl-1.8.8.jar

:/home/q/hive-0.11.0/lib/jackson-mapper-asl-1.8.8.jar

example/avro/User.java Test.java

运行上述代码：

java -classpath /home/q/hive-0.11.0/lib/avro-1.7.1.jar

:/home/q/hive-0.11.0/lib/avro-tools-1.7.4.jar

:/home/q/hive-0.11.0/lib/jackson-core-asl-1.8.8.jar

:/home/q/hive-0.11.0/lib/jackson-mapper-asl-1.8.8.jar:User.jar:.

Test