avro的使用

为了说明hadoop中rpc的实现方法，需要先临时插一下avro。因为hadoop的rpc就是通过avro来实现的，所以如果不了解avro的话，对理解hadoop的rpc来讲比较困难。因为很难去识别那些是hadoop的代码，那些是avro的代码。所以先介绍一下avro。这个还是很有用的一个开源库，可以实现java、python、js等之前的数据交换。比dubbo要完善一点，用起来也差不多。Java的dubbo很强，但是跨语言还比较弱，不过Dubbo好像也有python了。

下面开始，我们学习一下avro吧。还是通过例子的方法学习。这样比较直观。先用现学一直是我觉得开源代码的最好学习方式。

我们先创建一个maven项目，随便java的就可以了。然后添加avro用到的相关包：

<dependency>

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

<artifactId>avro</artifactId>

<version>1.8.2</version>

</dependency>

下面是用来自动生成代码用的【performingcodegeneration】

<plugin>

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

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

<version>1.8.2</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>

通过上面的两步，avro的开发环境就算搭建好了。然后我们一步一步的开始使用avro。这个库其实核心就是个类似json的一个序列化读写库。

**1.Definingaschema**

首先，我们需要定义一个schema文件，这个文件中记录了avro要处理的数据结构。它可以是简单的基本类型，例如：boolean,int,long,string等等，也可以是一些复杂的类型，像record、enum、map等，record就是pojo类。下面看示例代码中定义的那个说明文件user.avsc，如下所示：

{"namespace":"example.avro",

"type":"record",

"name":"User",

"fields":[

{"name":"name","type":"string"},

{"name":"favorite\_number","type":["int","null"]},

{"name":"favorite\_color","type":["string","null"]}

]

}

为了方便mvn来直接编译生成代码，我们将它放在${project.basedir}/src/main/avro/这个路径下。user.avsc中定义了一个record，它的名子叫User，包名为：example.avro,类型为record，然后它有3个属性字段，分别如下所示

1. name属性，类型为string;
2. favorite\_number属性，类型为int或者null;
3. favorite\_color属性，类型为string或者null;

通过以上信息，我们就定义了schema文件，有了它就可以使用avro来进行序列化和反序列化了。Avro有2种方式来进行以上两个操作，分别是采用自动代码生成器，生成schema对应的类文件，然后通过类进行调用的方式。另一种方式是直接解释schema文件。其实两种方式类拟。都是通过schema文件来定义数据结构，并用来管理数据。其实我更喜欢方法1生成文件的方式。看着比较直接。而且使用的时候，代码更精简。下面就先说这种方式吧。

方式1：生成代码方式

生成代码有两种方法。一个是用avro自带的jar包。使用如下命令来生成代码。

java-jar/path/to/avro-tools-1.8.2.jarcompileschemauser.avsc

别一种方式更简单，之前我们不是已经增加了maven-plugin么。通过这个插件可以更方便的生成代码。在plugin的配置中有如下参数，即指定了schema文件所在的路径，只要将schema文件放在该目录下。当使用mvncompile命令进行编译的时候，自动的就会生成schema对应的代码。

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

User类就是例子里自动生成了，我们简单看一下。首先此类是一个SpecificRecord接口的实现类。SpecificRecord接口的派生关系为GenericContainer->IndexedRecord->SpecificRecord。

主要需要实现以下三个方法。其中put和get是一对，用于置设User类的属性值。getSchema用来生成schema信息。可以理解为消息头。

voidput(inti,Objectv);

Objectget(inti);

SchemagetSchema();

除此以外User类还继承了SpecificRecordBase类。这个是一个实现SpecificRecord接口的抽象类。里面有最基础的get和put方法的实现。另外这个类还有两个重要的方法，用来进行正反序列化。如下所示：

publicvoidwriteExternal(ObjectOutputout)

publicvoidreadExternal(ObjectInputin)

好了这个就简单说到这了，代码大家有兴趣可以看看。

好了，有了User类就完成了avro的所有准备工作。下面就可以使用avro库进行数据的生成、序列化和反序列化了。首先生成avro数据，这个很简单就象其它pojo类一样，正常创建就可以了。如下所示：

Useruser1=newUser();

user1.setName("Alyssa");

user1.setFavoriteNumber(256);

下面进行序列化，首先需要一个DatumWriter用来转换User类。这个是说明用什么协议来表示数据。比如json、thrift、Protobuf等等。

然后再创建一个DataFileWriter对象，这个是用来读写文件的。因为avro不只可以序列化到文件。有了dataFileWriter，就可以像写文件一样写User对象了非常好用。代码如下：

DatumWriter<User>userDatumWriter=new SpecificDatumWriter<User>(

User.class);

DataFileWriter<User>dataFileWriter=new DataFileWriter<User>(

userDatumWriter);

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

dataFileWriter.append(user1);

dataFileWriter.append(user2);

dataFileWriter.append(user3);

dataFileWriter.close();

反序列化，方法类似使用DatumReader和DataFileReader类进行数据的读取。也是很方便的。代码如下所示：

DatumReader<User> userDatumReader = new SpecificDatumReader<User>(

User.class);

DataFileReader<User> dataFileReader = new DataFileReader<User>(

new File("users.avro"), userDatumReader);

User user = null;

while (dataFileReader.hasNext()) {

user = dataFileReader.next(user);

System.out.println(user);

}

通过以上几步，用代码来管理avro的方式就介绍完了。下面简单说一下另一种方式。

**方式2：不用代码方式**

这种方法只是没有了实体的类，但是操作的方式其实是一样的，所以我只简单介绍一下不同的地方就可以了。

首先需要生成一个schema对象。通过以下方法：

Schema schema = new Schema.Parser().parse(new File("src/main/avro/user.avsc"))

之后再借助GenericRecord类来生成对象，GenericRecord的put方法可以根据key和value的方法生成序列化对象对应的属性值。方法如下：

GenericRecord user1 = new GenericData.Record(schema);

user1.put("name", "Alyssa");

正反序列化也是类似的，只是使用GenericDatumWriter/GenericDatumReader来生成对应的DatumWriter/DatumReader对象了。并通过GenericRecord来处理序列化实体。相当于一套通用的Record代码。但是感觉反而不如有代码方式直观了。而且用起来也比较啰嗦。所以不推荐。