# Avro序列化

## 1 特点：

1. 支持多种语言，java，c，python，ruby
2. 支持鞥福的数据格式，8中基本数据类型和6中复杂的数据类型
3. 有一套快速可压缩的二进制编码形式

压缩算法：Huaffman coding哈弗曼编码

1. 支持rpc协议
2. 依赖于模式（schema）。即利用avro实现序列化或rpc。

## 2 实现

### 2.1创建maven

|  |
| --- |
| </**dependencies**> <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.avro</**groupId**>  <**artifactId**>avro-maven-plugin</**artifactId**>  <**version**>1.8.0</**version**>  <**executions**>  <**execution**>  <**phase**>generate-sources</**phase**>  <**goals**>  <**goal**>schema</**goal**>  <**goal**>protocol</**goal**>  <**goal**>idl-protocol</**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**>  </**plugins**> </**build**> |

### 2.2 编写avro的schema

Avro scheme是通过JSON形式来定义的，一般以.avsc结尾（maven插件会去指定目录下获取.avsc结尾的文件并生成成**[Java](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "http://blog.csdn.net/hua245942641/article/details/_blank)**文件）

在src/main/avro目录下新建一个文件，并保存为user.avsc。文件内容如下：

|  |
| --- |
| {"namespace": "cn.md31.avro.test.bean",  "type": "record",  "name": "User",  "fields": [ {"name": "name", "type": "string"},  {"name": "favorite\_number", "type": ["int", "null"]},  {"name": "favorite\_color", "type": ["string", "null"]}  ] } |

namespace在java项目中翻译成包名   
name是类名   
fields就是配置的属性   
注意：必须配置type为record

### 2.3 生成User的java文件

avro提供了一个avro-tools包来生成java文件，可以通过下面命令：

java -jar /path/to/avro-tools-1.8.0.jar compile schema <schema file> <destination>

或者可以使用maven的compile命令直接编译生成

### 2.4 用varo创建对象

User u2=**new** User(**"fengfeng"**,**"nan"**,23);  
  
User u3=User.*newBuilder*().setName(**"jetty"**).setGender(**"girl"**).setAge(20).build();  
User u4=User.*newBuilder*(u2).setAge(50).build();

### 2.5 序列化

User u3=User.*newBuilder*().setName(**"jetty"**).setGender(**"girl"**).setAge(20).build();  
User u4=User.*newBuilder*(u2).setAge(50).build();  
  
DatumWriter<User> dw=**new** SpecificDatumWriter<User>(User.**class**);  
DataFileWriter<User> dfw=**new** DataFileWriter<User>(dw);  
*//schema出入的是要序列化的对象的schema*dfw.create(u1.getSchema(),**new** File(**"1.txt"**));  
dfw.append(u2);  
dfw.append(u3);

### 2.6 反序列化

|  |
| --- |
| **public void** read() **throws** IOException {  DatumReader<User> reader=**new** SpecificDatumReader<User>(User.**class**);  DataFileReader<User> fileReader=**new** DataFileReader<User>(**new** File(**"1.txt"**),reader);  **while**(fileReader.hasNext()){  System.***out***.println(fileReader.next());  }  fileReader.close(); } |

# 2 rpc 远程过程调用

Remote procedure protocol

## 2.1 通过Avro实现1+1

### 2.1.1创建两个maven工程

一个是client，一个是server

1. 引入avro的pom文件

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>org.apache.avro</**groupId**>  <**artifactId**>avro</**artifactId**>  <**version**>1.8.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.avro</**groupId**>  <**artifactId**>avro-ipc</**artifactId**>  <**version**>1.8.0</**version**>  </**dependency**> </**dependencies**> <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.avro</**groupId**>  <**artifactId**>avro-maven-plugin</**artifactId**>  <**version**>1.8.0</**version**>  <**executions**>  <**execution**>  <**phase**>generate-sources</**phase**>  <**goals**>  <**goal**>schema</**goal**>  <**goal**>protocol</**goal**>  <**goal**>idl-protocol</**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**>  </**plugins**> </**build**> |

### 2.1.2编写协议文件

这个要根据avro的根式要求去写

在两个工程的avro目录下分别添加下面的rpc协议和Avro的schema

AddService.avdl:

|  |
| --- |
| @namespace("avro.protocol") protocol AddService{ import schema "user.avsc"; int add(int x,int y); void sendUser(cn.gao.User user); } |

user.avsc:

|  |
| --- |
| {"namespace":"cn.gao", "type":"record", "name":"User", "fields":[  {"name":"name","type":"string"},  {"name":"gender","type":["string","null"]},  {"name":"age","type":["int","null"]}  ] } |

### 2.1.3在根据arvo的API实现RPC

分别编译两个maven工程，会在工程的目录下面生成对应的接口类

服务端实现接口类：

|  |
| --- |
| **public class** AddServiceImpl **implements** AddService{   @Override  **public int** add(**int** x, **int** y) **throws** AvroRemoteException {  **return** x+y;  }   @Override  **public** Void sendUser(User user) **throws** AvroRemoteException {  System.***out***.println(user);  **return null**;  } } |

开启服务端：

|  |
| --- |
| **public class** AvroServer {  **public static void** main(String[] args) {  NettyServer server=**new** NettyServer(**new** SpecificResponder(AddService.**class**,**new** AddServiceImpl()),  **new** InetSocketAddress(8888));  **while**(**true**);  } } |

开启客户端：

|  |
| --- |
| **public class** AvroClient {  **public static void** main(String[] args) **throws** IOException {  NettyTransceiver client=**new** NettyTransceiver(**new** InetSocketAddress(**"127.0.0.1"**,8888));  AddService protocol= SpecificRequestor.*getClient*(AddService.**class**,client);  **int** result=protocol.add(2,3);  System.***out***.println(result);  protocol.sendUser(**new** User(**"gao"**,**"feng"**,22));  } } |