Apache Avro

Apache Avro是一个数据序列化系统，完成对象与二进制之间的转换，在传输数据前将对象转换成二进制流，到达目的地址后，Avro再将二进制刘转换成对象。其提供：

1. 丰富的数据结构
2. 紧凑、快速的二进制数据格式
3. 远程过程调用
4. 简单的动态语言集成
5. 代码的生成不需要读写数据文件，也不需要使用或者实现RPC协议，代码的生成仅是一个可选的优化，但是只对静态语言有意义

# Avro Schema

在对象序列化技术中，使用JSON序列化是一种轻量级的数据传输格式，但是JSON的格式是<key,value>类型，对存储空间浪费，而且会增加数据传输的压力。使用Avro可以解决这个问题，其序列化的文件由schema和真实数据组成：schema只是数据的元数据，相当于JSON数据的key信息，单独保存在一个JSON文件中，只需要保存一次。

Schama是描述对象的文件，通过模式定义各种数据结构，在数据进行序列化和反序列化之前都要先确定模式的结构，Avro的schema是由JSON定义的。当Avro用于RPC时，客户端和服务端在连接握手阶段进行Schema的交换，示例如下：

*{*

*"namespace": "example.avro",*

*"type": "record",*

*"name": "User",*

*"fields": [*

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

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

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

*]*

*}*

Schema由下列JSON对象定义：

1. JSON字符串，命名
2. JSON对象：{“type”:”typeName”,...attributes}
3. JSON数组，Avro中Union的定义

## **1.1 Schema数据类型**

Schema定义了简单数据类型和复杂数据类型，其中复杂数据类型包含不同属性，用户通过各种不同数据类型自定义复杂的数据结构。

1. 简单数据类型

|  |  |  |
| --- | --- | --- |
| 类型 | Encoding | Example |
| null | Zero bytes | Null |
| boolean | A single byte | {true:1, false:0} |
| int/long | variable-length zig-zag coding |  |
| float | 4 bytes | Java's floatToIntBits |
| double | 8 bytes | Java's doubleToLongBits |
| bytes | a long followed by that many bytes of data |  |
| string | A long followed by that many  Bytes of UTF-8 | “foo”:{3,f,o,o}  06 66 6f 6f |

1. 复杂数据类型

|  |  |
| --- | --- |
| Type | encoding |
| Records | encoded just the concatenation of the encodings of its fields |
| Enums | a int representing the zero-based position of the symbol in the schema |
| Arrays | encoded as series of blocks. A block with count 0 indicates the end of the array. block:{long,items} |
| Maps | encoded as series of blocks. A block with count 0 indicates the end of the map. block:{long,key/value pairs}. |
| Unions | encoded by first writing a long value indicating the zero-based position within the union of the schema of its value. The value is then encoded per the indicated schema within the union. |
| fixed | encoded using number of bytes declared in the schema |

## **1.2 Avro排序**

Avro为数据定义了标准的排列顺序，用于对象之间的操作，标准定义可以进行方便有效的比较和排序。同时标准的定义可以方便对Avro的二进制编码数据直接进行排序，而不需要反序列化。只有当数据项包含相同的Schema的时候，数据之间的比较才有意义，数据的比较按照Schema深度优先，从左到右的顺序递归的进行。比较方式如下表：

|  |  |
| --- | --- |
| 类型 | 比较规则 |
| null | 总是相等 |
| int,long,float | 按照数值大小比较 |
| boolean | false在true之前 |
| string | 按照字典序进行比较 |
| bytes，fixed | 按照byte的字典序进行比较 |
| array | 按照元素的字典序进行比较 |
| enum | 按照符号在枚举中的位置比较 |
| record | 按照域的字典序排序，如果指定了以下属性 |
| ascending | 域值的顺序不变 |
| descending | 域值的顺序颠倒 |
| ignore | 排序的时候忽略域值 |
| map | 不能进行比较 |

## **1.3 对象容器文件**

Avro定义了简单的对象容器文件格式，一个文件对应一个模式。所有存储在文件中对象都是根据模式写入的，对象按照块进行存储，对块可以采用压缩的方式存储。为了进行分布式处理，在块之间采用同步记号，一个文件可以包含任意用户定义的元数据。文件由两部分组成：文件头和文件数据块组成

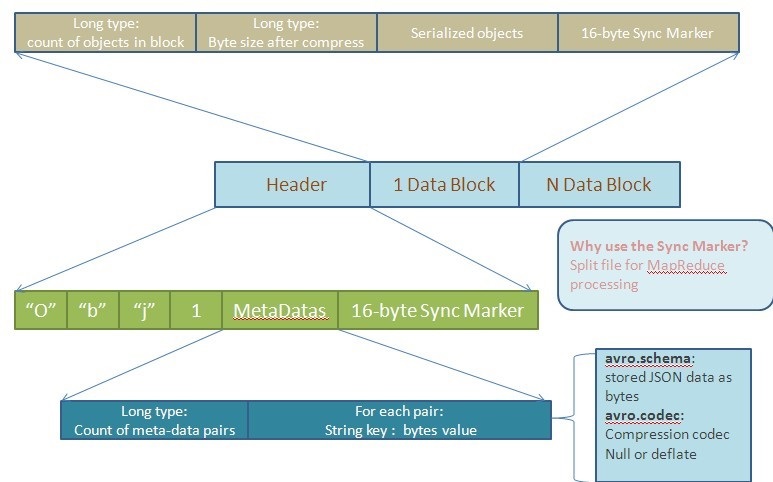
1. 文件头

* 4字节前缀，ASCII‘O’，‘b’，‘j’,1
* 文件元数据，描述schema
* 文件同步记号， 16字节同步标记符。

1. 文件数据块，包括以下：

* 文件中块中对象数目的长整型
* 块中数据序列化后的字节数长度的长整型
* 序列化的对象
* 16字节的文件同步记号

当数据块的长度为0时即为文件数据块的最后一个数据，此后的所有数据被自动忽略。对象容器文件的结构如下：



1. **Avro RPC**

Avro RPC是一个跨语言实现的RPC服务框架，轻量级、实现简介、使用方便同时支持使用者进行二次开发，主要特点：

* 客户端传输层与应用层逻辑分离，传输层主要职责包括创建连接，连接查找与复用，传输数据，接收服务端回复后回调应用层
* 客户端支持同步和异步调用。服务异步化能很好提高系统吞吐量，建议使用异步调用。为了防止异步发送请求过快，客户端增加请求流量限制功能
* 服务端有一个协议注册工厂和序列化注册工厂。针对不同应用场景来定制服务方式，RPC只是服务方式的一种。在分布式的系统架构中，分布式节点之间的通信会存在多种方式，比如MQ的TOP消息，一个消息可以有多个订阅者
* Avro序列化是Hadoop下的一个子项目，其特点是数据序列化不带标签，因此序列化数据量非常小

Avro RPC在逻辑上的框架分为两层：

* 网络传输层，使用Netty的NIO实现
* 可扩展协议层，目前支持的数据序列化方式有：Avro、Protocol Buffers、Json、Hessian及Java序列化。使用者可以注册自己的协议格式及序列化方式

1. 在pom.xml中，添加avro-rpc依赖，同时修改插件的配置

*<dependency>*

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

*<artifactId>avro-ipc</artifactId>*

*<version>1.8.1</version>*

*</dependency>*

添加插件配置：

*<plugin>*

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

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

*<version>1.8.1</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>*

1. 声明protocol Schema

*{*

*"namespace": "com.fys.avro.rpc",*

*"protocol": "Mail",*

*"types": [*

*{*

*"name": "Message",*

*"type": "record",*

*"fields": [*

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

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

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

*]*

*}*

*],*

*"messages": {*

*"send": {*

*"request" : [{ "name" : "message", "type":"Message"}],*

*"response": "string"*

*}*

*}*

*}*

执行maven compile即可生成protocol所需要的Java代码，包括接口Mail及Message类

1. 声明业务逻辑类MailImpl

*public class MailImpl implements Mail {*

*@Override*

*public CharSequence send(Message message) throws AvroRemoteException {*

*System.out.println("Message Received:" + message);*

*return "Received your message: " + message.getFrom().toString()*

*+" with body " + message.getBody().toString();*

*}*

*}*

1. 定义Avro Server类

*public class AvroServer {*

*private static Server server;*

*public static void main(String[] args) throws Exception {*

*System.out.println("Starting server");*

*startServer();*

*Thread.sleep(1000);*

*System.out.println("Server Started");*

*}*

*private static void startServer() throws IOException {*

*server = new NettyServer(new SpecificResponder(Mail.class, new MailImpl()),*

*new InetSocketAddress(65111));*

*}*

*}*

1. 定义Avro Client类

*public class AvroClient {*

*public static void main(String[] args) throws Exception {*

*NettyTransceiver client = new NettyTransceiver(*

*new InetSocketAddress(65111)*

*);*

*Mail proxy = SpecificRequestor.getClient(Mail.class, client);*

*System.out.println("Client of Mail Proxy is built");*

*args = new String[]{"to:Tom", "from:Jack", "body:How are you"};*

*Message message = new Message();*

*message.setTo(args[0]);*

*message.setFrom(args[1]);*

*message.setBody(args[2]);*

*System.out.println("Rpc call with message: " + message.toString());*

*System.out.println("Result: " + proxy.send(message));*

*client.close();*

*}*

*}*

1. 执行，在Server端输出如下：

*Server Started*

*Message Received:{"to": "to:Tom", "from": "from:Jack", "body": "body:How are you"}*

<http://shift-alt-ctrl.iteye.com/blog/2217425>

http://avro.apache.org/docs/current/spec.html#order