在Kafka中使用Avro编码消息：Producter篇

本文将介绍如何在 Kafka 中使用 Avro 来序列化消息，并提供完整的 Producter 代码共大家使用。

Avro

Avro 是一个数据序列化的系统，它可以将数据结构或对象转化成便于存储或传输的格式。Avro设计之初就用来支持数据密集型应用，适合于远程或本地大规模数据的存储和交换。因为本文并不是专门介绍 Avro 的文章，如需要更加详细地了解，请参见《Apache Avro使用入门指南》

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

在使用 Avro 之前，我们需要先定义模式（schemas）。模式通常使用 JSON 来编写，我们不需要再定义相关的类，这篇文章中，我们将使用如下的模式：

```
{
   "fields": [
      { "name": "str1", "type": "string" },
      { "name": "str2", "type": "string" },
      { "name": "int1", "type": "int" }
   ],
   "name": "Iteblog",
   "type": "record"
}
```

上面的模式中，我们定义了一种 record 类型的对象，名字为 Iteblog，这个对象包含了两个字符串和一个 int 类型的fields。定义好模式之后，我们可以使用 avro 提供的相应方法来解析这个模式：

```
Schema.Parser parser = new Schema.Parser();
Schema schema = parser.parse(USER_SCHEMA);
```

这里的 USER_SCHEMA 变量存储的就是上面定义好的模式。

解析好模式定义的对象之后，我们需要将这个对象序列化成字节数组，或者将字节数组转换成对象。Avro 提供的 API 不太易于使用，所以本文使用 twitter 开源的 Bijection 库来方便地实现这些操作。我们先创建 Injection 对象来讲对象转换成字节数组：

```
Injection<GenericRecord, byte[]> recordInjection = GenericAvroCodecs.toBinary(schema);
```

现在我们可以根据之前定义好的模式来创建相关的 Record，并使用 recordInjection 来序列化这个 Record ：

```
GenericData.Record record = new GenericData.Record(schema);
avroRecord.put("str1", "My first string");
avroRecord.put("str2", "My second string");
avroRecord.put("int1", 42);

byte[] bytes = recordInjection.apply(record);
```

Producter实现

有了上面的介绍之后，我们现在就可以在 Kafka 中使用 Avro 来序列化我们需要发送的消息了：

```
package com.iteblog.avro;

import com.twitter.bijection.Injection;
import com.twitter.bijection.avro.GenericAvroCodecs;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
```

```
import org.apache.kafka.clients.producer.ProducerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Properties;

/**
 * Created by yangping.wu on 2017-07-20.
 */
public class AvroKafkaProducter {
    Logger logger = LoggerFactory.getLogger("AvroKafkaProducter");
    public static final String USER_SCHEMA = "{"
            + "₩"type₩":₩"record₩","
            + "₩"name₩":₩"Iteblog₩","
            + "₩"fields₩":["
            + "  { ₩"name₩":₩"str1₩", ₩"type₩":₩"string₩" },"
            + "  { ₩"name₩":₩"str2₩", ₩"type₩":₩"string₩" },"
            + "  { ₩"name₩":₩"int1₩", ₩"type₩":₩"int₩" }"
            + "]}";

    public static void main(String[] args) throws InterruptedException {
        Properties props = new Properties();
        props.put("bootstrap.servers", "www.iteblog.com:9092");
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer")
;

        Schema.Parser parser = new Schema.Parser();
        Schema schema = parser.parse(USER_SCHEMA);
        Injection<GenericRecord, byte[]> recordInjection = GenericAvroCodecs.toBinary(schema);

        KafkaProducer<String, byte[]> producer = new KafkaProducer<>(props);

        for (int i = 0; i < 1000; i++) {
            GenericData.Record avroRecord = new GenericData.Record(schema);
            avroRecord.put("str1", "Str 1-" + i);
            avroRecord.put("str2", "Str 2-" + i);
            avroRecord.put("int1", i);

            byte[] bytes = recordInjection.apply(avroRecord);

            ProducerRecord<String, byte[]> record = new ProducerRecord<>("iteblog", "" + i, bytes);
            producer.send(record);
            Thread.sleep(250);

        }
```

```
        producer.close();
    }
}
```

因为我们使用到 Avro 和 Bijection 类库，所有我们需要在 pom.xml 文件里面引入以下依赖：

```
<dependency>
  <groupId>org.apache.avro</groupId>
  <artifactId>avro</artifactId>
  <version>1.8.0</version>
</dependency>

<dependency>
  <groupId>com.twitter</groupId>
  <artifactId>bijection-avro_2.10</artifactId>
  <version>0.9.2</version>
</dependency>
```

运行

现在一切准备就绪，我们可以使用下面的命令来运行这个消息发送者了。运行这个程序我们需要准备好 avro-1.8.1.jar，slf4j-api-1.7.21.jar，log4j-1.2.17.jar，slf4j-log4j12-1.7.7.jar 以及 scala-library.jar等相关Jar包，为了方便我将这些 jar 包放到 lib
目录下，然后我们如下编写运行的脚本：

```
CLASSPATH=$CLASSPATH:

for i in /home/iteblog/lib/*.jar ; do
    CLASSPATH=$CLASSPATH:$i
done

java -cp $CLASSPATH:flink-kafka-1.0-SNAPSHOT.jar com.iteblog.avro.AvroKafkaProducter
```

当然，我们也可以将所有这些依赖全部打包进 flink-kafka-1.0-SNAPSHOT.jar 里面，成为一个 fat 包，这时候我们就不需要单独添加其他的依赖了。