Apache Kafka编程入门指南：Producer篇

《Apache Kafka编程入门指南：Producer篇》
《Apache Kafka编程入门指南：设置分区数和复制因子》
Apache Kafka编程入门指南：Consumer篇

Kafka最初由Linkedin公司开发的分布式、分区的、多副本的、多订阅者的消息系统。它提供了类似于JMS的特性，但是在设计实现上完全不同，此外它并不是JMS规范的实现。kafka对消息保存时根据Topic进行归类，发送消息者成为Producer；消息接受者成为Consumer；此外kafka集群有多个kafka实例组成，每个实例(server)成为broker。无论是kafka集群，还是producer和consumer都依赖于zookeeper来保证系统可用性集群保存一些meta信息。本文不打算对Apache Kafka的原理和实现进行介绍，而在编程的角度上介绍如何使用Apache Kafka。我们分别介绍如何编写Producer、Consumer以及Partitioner等。

引入依赖

在编程之前，我们首先引入kafka的依赖，在你的pom.xml文件加入以下依赖：

```
<dependency>
 <groupId>org.apache.kafka</groupId>
 <artifactId>kafka_2.10</artifactId>
 <version>0.8.2.1</version>
</dependency>
```

或者在你的build.sbt文件加入以下依赖：

```
libraryDependencies += "org.apache.kafka" % "kafka_2.10" % "0.8.2.1"
```

配置属性

为了使得我们能够连上Kafka集群、如何系列化消息以及其他的属性，我们需要在程序里面进行相关的配置，如下：

```
val props = new Properties()
props.put("metadata.broker.list", "www.iteblog.com:9092,www.iteblog.com:9091")
props.put("serializer.class", "kafka.serializer.StringEncoder")
props.put("key.serializer.class", "kafka.serializer.StringEncoder")
```

```
props.put("partitioner.class", "com.iteblog.kafka.IteblogPartitioner")
props.put("request.required.acks", "1")
```

1、metadata.broker.list参数指定了Kafka broker的地址，这个参数使得Producer可以找到每个Topic的Leader。我们不需要将Kafka broker所有的地址都列在这里，而只需要列出其中的两个，之所以列出两个是因为预防其中一个连不上，有了这两个Kafka broker。我们的Producer足以找到Topic的 meta data。

2、serializer.class这个参数指定了如何系列化发送的消息，这个参数的默认值是kafka.serializer.DefaultEncoder，它将Array[Byte]转换成Array[Byte]，也就是说什么都没做，如果你发送的消息不是Array[Byte]的，那么我们需要指定。当然，我们可以自定义消息系列化类，只需要继承kafka.serializer.Encoder即可，实现其中的toBytes方法即可（详情可以参见《用Spark往Kafka里面写对象设计与实现》）。

3、key.serializer.class参数指定了如何系列化消息的key，默认值同serializer.class，也是可以自定义的。partitioner.class参数指定了如何根据消息计算其分区号，必须是继承kafka.producer.Partitioner的类。默认值是kafka.producer.DefaultPartitioner，他是根据key的哈希值模上总分区数的。如果没有指定key，那么Producer将为这条消息随机选择一个分区。

4、request.required.acks参数指定了Producer发送消息之后是否需要来自Broker的acknowledgement，默认值为0，也就是不需要acknowledgement，这种情况下Producer只管发送消息，可能会导致消息丢失，但是因为不需要确认，所以发送速度也是很快的。

Producer的参数有很多个，我们可以根据自己需求来设定。

创建Producer

设置好配置属性之后，我们可以创建Producer，如下：

```
val config = new ProducerConfig(props)

val producer = new Producer[String, String](config)
```

发送消息

producer有个send的函数，可以发送消息。我们可以将消息封装成KeyedMessage，指定其中的topic，key以及value即可

```
val message = new KeyedMessage[String, String]("iteblog", msg, msg)
producer.send(message)
```

上面的iteblog是topic的名字，如果这个topic不存在，producer可以创建，但是我们需要保证auto.create.topics.enable参数设置成ture了（默认值就是true）。这样简单地往Kafka里面发送消息的程序就完成了。

定义分区

作为例子，我实现了自定义的分区类，其核心和默认的分区类思路类似：

```
package com.iteblog.kafka

import kafka.producer.Partitioner
import kafka.utils.VerifiableProperties

/**
 * User: 过往记忆
 * Date: 2016-02-05
 * Time: 下午07:46
 * bolg:
 * 本文地址：/archives/1580
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */

class IteblogPartitioner(props: VerifiableProperties) extends Partitioner {
  override def partition(key: Any, numPartitions: Int): Int = {
    key match {
      case msg: String => Math.abs(msg.hashCode) % numPartitions
      case _ => Math.abs(key.hashCode()) % numPartitions
    }
  }
}
```

完整代码

```
package com.iteblog.kafka

import java.util.{Date, Properties, UUID}
```

```
import kafka.producer.{KeyedMessage, Producer, ProducerConfig}

/**
 * User: 过往记忆
 * Date: 2016-02-05
 * Time: 下午07:46
 * bolg:
 * 本文地址：/archives/1580
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */

object IteblogProducer {
  def main(args: Array[String]) {
    val events = 100

    // 设置配置属性
    val props = new Properties()
    props.put("metadata.broker.list", "www.iteblog.com:9092,www.iteblog.com:9091")
    props.put("serializer.class", "kafka.serializer.StringEncoder")
    props.put("partitioner.class", "com.iteblog.kafka.IteblogPartitioner")
    props.put("key.serializer.class", "kafka.serializer.StringEncoder")
    props.put("request.required.acks", "1")


    val config = new ProducerConfig(props)
    // 创建producer
    lazy val producer = new Producer[String, String](config)

    // 产生并发送消息
    for (i <- 0 until events) {
      val runtime = new Date().getTime
      val msg = s"$runtime,${UUID.randomUUID()}"
      val message = new KeyedMessage[String, String]("iteblog", msg, msg)
      producer.send(message)
    }
    // 关闭producer
    producer.close()
  }
}
```

运行和测试

本程序的运行方式很简单，直接使用java命令即可运行：

```
[iteblog@www.iteblog.com ~]$ java -cp ./iteblog.jar com.iteblog.kafka.IteblogProducer
```

这样程序就往kafka里面发送消息了，为了检测消息是否真实发送到Kafka集群，我们可以使用Kafka自带的kafka-console-consumer.sh脚本来消费其中的消息：

```
[iteblog@www.iteblog.com ~]$ bin/kafka-console-
consumer.sh --zookeeper  www.iteblog.com:2181 --topic iteblog
1454662956377,1dd415f3-23de-46d3-abdb-0bf1e244904f
1454662956381,1c75ae7a-d3c5-4205-9dab-df437b7cd071
1454662956386,730b467d-2fc9-4cdd-a1da-184b19ae1a98
1454662956395,ea9fd195-cd63-4320-8871-90474d4b33b6
1454662956405,32f51f64-9b49-409c-9994-78b0a3a1e867
1454662956415,232a1b20-c7c3-4383-8984-c2b62e941830
1454662956424,96761614-1b61-450f-85d5-5e9602ce28bc
1454662956432,d44314e9-0888-48bb-9daf-ccbb1000fb4c
1454662956434,06935478-cbfd-4d5e-8a09-7a879ecb01a3
1454662956437,245ff5f3-e7bf-4e0e-95fe-bd060780d2a7
1454662956445,19949b6c-8be2-4fb8-b376-a721206a36e4
1454662956411,b5977884-96b5-4b89-a65b-45642f5eba10
1454662956439,989c958a-42af-498b-85f0-7364364a2dbc
1454662956442,cfdea9b5-f75a-49fa-89b5-26de1ee00bff
1454662956448,a3fc7a74-bc13-4cf4-bd14-df7ab4ec9ac2
1454662956451,196cd6bc-8dbf-45b1-9cf5-36036dc49832
1454662956454,5a85b439-a716-42a3-89f6-c3163bcc6e9d
1454662956457,c76bdf54-fc48-4b4e-826a-cb9c2a8eea77
1454662956463,e0a7fa00-ef26-403b-b18f-33f9400b7169
1454662956467,cab61107-5d44-4bf4-8759-a71f95e7c81e
1454662956471,aff16faf-8b25-41e2-a9f8-65c4f8aa51b7
1454662956475,bd723444-482c-4250-9bd1-e1799e10bf30
1454662956478,cbfacf9d-d436-4171-af1b-6330d9daa2c8
1454662956481,16136df1-3fb2-46eb-b12f-6f91636a73ad
1454662956485,8be988ab-30e3-4b86-a67f-f12715289a2a
1454662956488,efc239bd-fded-4582-b305-7d3ca0d06a4a
1454662956491,fd47493d-7f86-4f9c-bd2f-058c24ba4472
.......
```

大功告成！