使用Flink读取Kafka中的消息

本文将介绍如何通过Flink读取Kafka中Topic的数据。

和Spark一样，Flink内置提供了读/写Kafka Topic的Kafka连接器(Kafka Connectors)。Flink Kafka Consumer和Flink的Checkpint机制进行了整合，以此提供了exactly-once处理语义。为了实现这个语义，Flink不仅仅依赖于追踪Kafka的消费者group偏移量，而且将这些偏移量存储在其内部用于追踪。

和Spark一样，Flink和Kafka整合的相关API也没有打包进Flink包中，而是单独进行了打包；所以如果我们需要在Flink中使用到Kafka，需要将这个包引入到我们的pom.xml文件中。本文以Flink 1.0.0和Scala 2.10.x为例进行说明，我们将下面的依赖引入到pom.xml文件中：

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-kafka-0.8_2.10</artifactId>
  <version>1.0.0</version>
</dependency>
```

当然，在读取Kafka中的数据之前，需要确保你已经安装和部署好Kafka集群了，这里我就不介绍如何部署Kafka集群，可以参见：《Kafka分布式集群部署手册(一)》、《Kafka分布式集群部署手册(二)》

Kafka Consumer

我们需要编写一个Kafka Consumer，通过Flink计算引擎从Kafka相应的Topic中读取数据。在Flink中，我们可以通过FlinkKafkaConsumer08来实现，这个类提供了读取一个或者多个Kafka Topic的机制。它的构造函数接收以下几个参数：

1、topic的名字，可以是String(用于读取一个Topic)List(用于读取多个Topic)；

2、可以提供一个DeserializationSchema / KeyedDeserializationSchema用于反系列化Kafka中的字节数组；

3、Kafka consumer的一些配置信息，而且我们必须指定bootstrap.servers、zookeeper.connect(这个属性仅仅在Kafka 0.8中需要)和group.id属性。

好了，我们来使用FlinkKafkaConsumer08类吧，初始化如下：

```
val properties = new Properties();
properties.setProperty("bootstrap.servers", "www.iteblog.com:9092");
// only required for Kafka 0.8
properties.setProperty("zookeeper.connect", "www.iteblog.com:2181");
```

```
properties.setProperty("group.id", "iteblog");
val stream = env.addSource(new FlinkKafkaConsumer08[String]("iteblog",
      new SimpleStringSchema(), properties))
stream.print()
```

上面的例子中使用到SimpleStringSchema来反系列化message，这个类是实现了DeserializationSchema接口，并重写了T deserialize(byte[] message)函数，DeserializationSchema接口仅提供了反系列化data的接口，所以如果我们需要反系列化key，我们需要使用KeyedDeserializationSchema的子类。KeyedDeserializationSchema接口提供了T deserialize(byte[] messageKey, byte[] message, String topic, int partition, long offset)方法，可以饭系列化kafka消息的data和key。

为了方便使用，Flink内部提供了一序列的schemas：TypeInformationSerializationSchema和TypeInformationKeyValueSerializationSchema，它可以根据Flink的TypeInformation信息来推断出需要选择的schemas。

Kafka Consumers和Fault Tolerance

如果我们启用了Flink的Checkpint机制，那么Flink Kafka Consumer将会从指定的Topic中消费消息，然后定期地将Kafka offsets信息、状态信息以及其他的操作信息进行Checkpint。所以，如果Flink作业出故障了，Flink将会从最新的Checkpint中恢复，并且从上一次偏移量开始读取Kafka中消费消息。

我们需要在执行环境下启用Flink Checkpint机制，如下：

```
val env = StreamExecutionEnvironment.getExecutionEnvironment()
env.enableCheckpointing(5000) // checkpoint every 5000 msecs
```

需要主要的是，Flink仅仅支持在拥有足够的处理slots情况下才能够从Checkpint恢复。Flink on YARN模式下支持动态地重启丢失的YARN containers。

如果我们没有启用Checkpint，那么Flink Kafka consumer将会定期地向Zookeeper commit偏移量。

完整的代码

```
package com.iteblog

import java.util.Properties
```

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend
import org.apache.flink.streaming.api.scala.{StreamExecutionEnvironment, _}
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer08
import org.apache.flink.streaming.util.serialization.SimpleStringSchema

/**
 * Created by  on 2016/5/3.
 */
object FlinkKafkaStreaming {
  def main(args: Array[String]) {
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.enableCheckpointing(5000)
    val properties = new Properties()
    properties.setProperty("bootstrap.servers", "www.iteblog.com:9092")
    // only required for Kafka 0.8
    properties.setProperty("zookeeper.connect", "www.iteblog.com:2181")
    properties.setProperty("group.id", "iteblog")


    val stream = env.addSource(new FlinkKafkaConsumer08[String]("iteblog",
       new SimpleStringSchema(), properties))
    stream.setParallelism(4).writeAsText("hdfs:///tmp/iteblog/data")

    env.execute("IteblogFlinkKafkaStreaming")
  }
}
```

运行上面的程序，将会在hdfs:///tmp/iteblog/目录下创建data文件，并将数据写入到data文件中。