FLINK实战-使用CEP进行网站状态监控报警和报警恢复

flink CEP 简介

flink CEP（Complex event processing），是在Flink之上实现的复杂事件处理库，可以允许我们在不断的流式数据中通过我们自己定义的模式（Pattern）检测和获取出我们想要的数据，然后对这些数据进行下一步的处理。 通过各种pattern的组合，我们可以定义出非常复杂的模式来匹配我们的数据。

网上讲CEP原理和用法的文章很多，大家可以参考下
https://juejin.im/post/5de1f32af265da05cc3190f9#heading-9

简单来说一下，其实我们可以把使用flink cep当做我们平时用的正则表达式，cep中的Pattern就是我们定义的正则表达式，flink中的DataStream就是正则表达式中待匹配的字符串，flink 通过DataStream 和 自定义的Pattern进行匹配，生成一个经过过滤之后的DataStream .

基于自定义的pattern，我们可以做很多工作，比如监控报警、风控、反爬等等，接下来我们基于一个简单的报警小例子来讲解一些FLINK cep的实际应用。

案例详解

我们基于flink CEP做一个简单的报警，首先我们简化一下报警的需求
1.统计出来每秒钟http状态码为非200的数量所占比例。大于0.7的时候触发报警。
2.统计结果连续发生三大于阈值（0.7，这个数字是我自己写的，为了测试用，真实环境需要根据实际经验来设置）发送报警通知。
3.统计结果小于等于阈值触发报警恢复通知。

实际应用中我们一般会去消费kafka的数据来作为source、这里我们为了简化，通过自定义source生成一些模拟的数据。

```
 public static class MySource implements SourceFunction<Tuple4<String,Long,Integer,Integer>>
{

  static int status[] = {200, 404, 500, 501, 301};

  @Override
```

```
  public void run(SourceContext<Tuple4<String,Long,Integer,Integer>> sourceContext) throws E
xception{
   while (true){
    Thread.sleep((int) (Math.random() * 100));
    // traceid,timestamp,status,response time

    Tuple4 log = Tuple4.of(
      UUID.randomUUID().toString(),
      System.currentTimeMillis(),
      status[(int) (Math.random() * 4)],
      (int) (Math.random() * 100));

    sourceContext.collect(log);
   }
  }

  @Override
  public void cancel(){

  }
 }
```

复制代码

接下来我们定义一个sql，用来计算我们的需求中的第一个要求。

```
  String sql = "select pv,errorcount,round(CAST(errorcount AS DOUBLE)/pv,2) as errorRate," +
               "(starttime + interval '8' hour ) as stime," +
               "(endtime + interval '8' hour ) as etime  " +
               "from (select count(*) as pv," +
               "sum(case when status = 200 then 0 else 1 end) as errorcount, " +
               "TUMBLE_START(proctime,INTERVAL '1' SECOND)  as starttime," +
               "TUMBLE_END(proctime,INTERVAL '1' SECOND)  as endtime  " +
               "from log  group by TUMBLE(proctime,INTERVAL '1' SECOND) )";
```

复制代码

通过执行sql，我们获取到了一个Result对象的DataStream,

```
  Table table = tenv.sqlQuery(sql);
  DataStream<Result> ds1 = tenv.toAppendStream(table, Result.class);
```

复制代码

接下来我们到了最核心的地方，我们需要定一个Pattern。

```
```

```
Pattern pattern = Pattern.<Result>begin("alert").where(new IterativeCondition<Result>(){
 @Override
 public boolean filter(
   Result i, Context<Result> context) throws Exception{
  return i.getErrorRate() > 0.7D;
 }
}).times(3).consecutive().followedBy("recovery").where(new IterativeCondition<Result>(){
 @Override
 public boolean filter(
   Result i,
   Context<Result> context) throws Exception{
  return i.getErrorRate() <= 0.7D;
 }
}).optional();
```

复制代码

来详细解释一下这个Pattern

1. 首先定义一个名为alert的Pattern，该Pattern的作用就是过滤出错误率大于0.7的数据，
2. times(3) ,表示要匹配三次，也就是要三次大于0.7.
3. consecutive 表示上述的三次匹配要是连续的，比如0.75、0.8、0.78，只有类似这样的数据才能被匹配到，中间不能有不符合的数据出现。
4. followedBy表示该alert pattern的下面要跟着一个recovery pattern，而followedBy是宽松匹配，也就是两个模式之间可以有其他的数据，如果要采用严格匹配，是使用next.
5. 最后recovery pattern加上一个optional 是我为了区分报警，和报警恢复想的的一个方案，这样的话，如果是只匹配到了alert pattern，输出的就是报警，如果recovery pattern也匹配到了，那么就是报警恢复。

在我们获得了相应的报警和恢复之后，接下来就是调用报警接口进行处理了，我们这只是简单的打印出来信息。

```
DataStream<Map<String,List<Result>>> alertStream = org.apache.flink.cep.CEP.pattern(
  ds1,
  pattern).select(new PatternSelectFunction<Result,Map<String,List<Result>>>(){
 @Override
 public Map<String,List<Result>> select(Map<String,List<Result>> map) throws Exception{
  List<Result> alertList = map.get("alert");
  List<Result> recoveryList = map.get("recovery");

  if (recoveryList != null){
   System.out.print("接受到了报警恢复的信息，报警信息如下：");
   System.out.print(alertList);
```

```
      System.out.print("  对应的恢复信息: ");
      System.out.println(recoveryList);
     } else {
      System.out.print("收到了报警信息 ");
      System.out.print(alertList);
     }

     return map;
    }
   });
```

复制代码

完整的代码请参考
https://github.com/zhangjun0x01/bigdata-examples/blob/master/flink/src/main/java/cep/WebMonitorAlert.java