FLINK FORWARD # ASIA 2022 ONLINE | 实时即未来

Flink CEP 新特性进展与在实时风控场景的落地

耿飙 | 阿里云开发工程师
胡俊涛 | 阿里云开发工程师

01 Flink CEP介绍&新功能解读

02 动态多规则支持与Demo

03 Flink CEP SQL语法增强

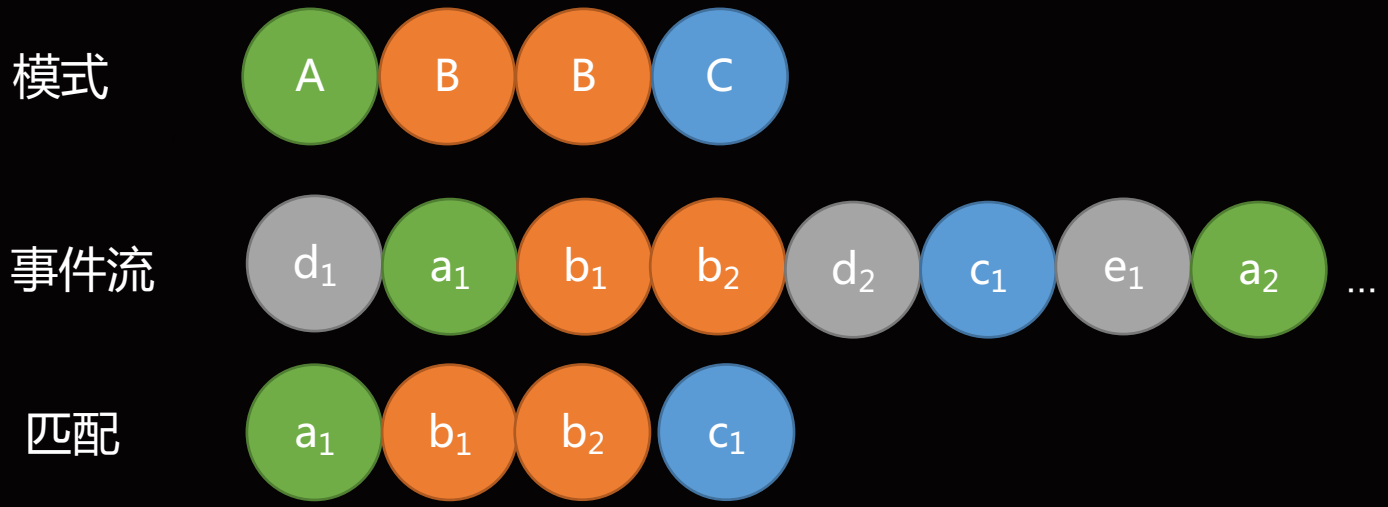
04 未来规划

01 Flink CEP介绍&新功能解读

什么是Flink CEP

CEP: 复杂事件处理(Complex Event Processing)

Flink CEP: 基于Flink实现的复杂事件处理库，它可以识别出数据流中符合特定模式(Pattern)的事件序列，并允许用户作出针对性处理



Flink CEP应用场景

实时风控

风险用户检测：5分钟内转账次数超过10次且金额大于10000

实时营销

营销策略优化：大促期间在购物车中添加了超过3次商品后，但最后没有结账付款的用户

物联网

异常状态告警：共享单车被骑出指定区域且15分钟内没有回到指定区域后发出告警

Flink CEP在1.16的改进

- FLINK-27392：支持在Pattern内的相邻事件之间定义时间窗口
- FLINK-26941：支持在带有窗口的Patten中以notFollowedBy结尾
- FLINK-24865：支持批模式下使用MATCH_RECOGNIZE
- FLINK-23890：优化Timer创建策略

Flink CEP在1.16的改进

营销场景：寻找大促当天在领取优惠券后的5分钟内在购物车中添加了商品，但最后没有结账付款的用户

```
Pattern<Event, Event> pattern =
        Pattern.<Event>begin("acceptCoupon")
                .where(new StartCondition())
                .followedBy("addItem")
                .where(new MiddleCondition())
                .within(Time.minutes(5), WithinType.PREVIOUS_AND_CURRENT)
                .notFollowedBy("pay")
                .where(new EndCondition())
                .within(Time.days(1));
```

02 动态多规则支持与Demo

动态规则支持: 背景

为什么需要支持动态规则更新？

- 频繁变化的实际场景要求对初始规则的内容进行调整或者添加新的规则，而重启Flink作业来使变化后的规则生效的方式时间成本高、影响范围大

关键问题：
- 如何让Flink作业不停机加载新规则？
- 如何解决Pattern的(de)serialization?

现有方案：
- 修改CepOperator添加注入规则的接口
- 基于Groovy引擎动态生成Pattern对象

例子：5分钟内通过广告链接访问某商品超过5次但最终没有购买

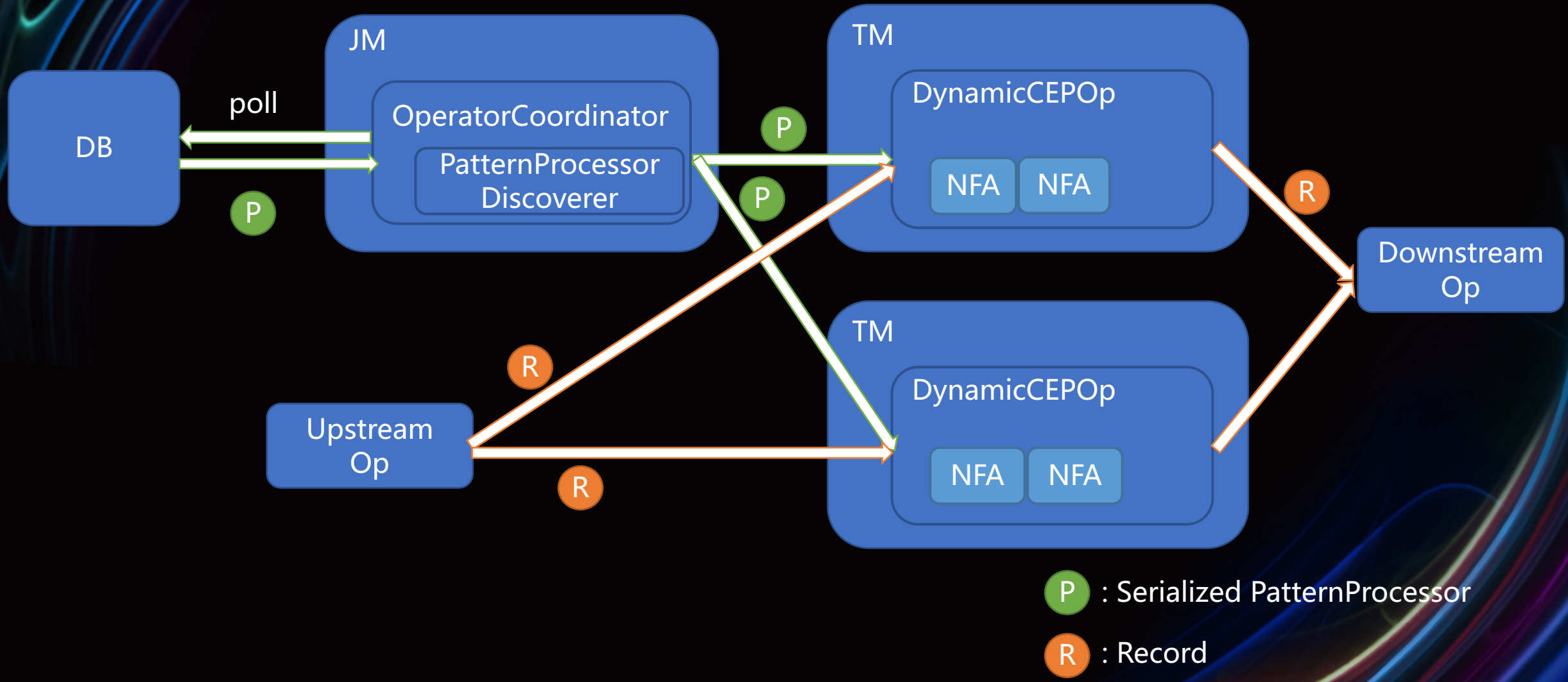
动态规则支持: 设计

新增接口（FLIP-200）

- PatternProcessor
  - id
  - version
  - timestamp
  - pattern
  - patternProcessorFunction
- PatternProcessorDiscoverer
- PatternProcessorManager

```java
public interface PatternProcessor<IN> extends Serializable, Versioned {

    /**
     * Returns the ID of the pattern processor.
     */
    String getId();

    /**
     * Returns the scheduled time at which the pattern processor should take effective.
     */
    default Long getTimestamp() { return Long.MIN_VALUE; }

    /**
     * Returns the {@link Pattern} to be matched.
     */
    Pattern<IN, ?> getPattern(ClassLoader classLoader);

    /**
     * Returns the {@link PatternProcessFunction} to process the found matches for the pattern.
     */
    PatternProcessFunction<IN, ?> getPatternProcessFunction();
}
```

动态规则支持: 设计


JM
DB
poll
OperatorCoordinator
PatternProcessor Discoverer
P
P
P
TM
DynamicCEPOp
NFA
NFA
R
Downstream Op
TM
DynamicCEPOp
NFA
NFA
R
Upstream Op
R
P : Serialized PatternProcessor
R : Record

动态规则支持：(de)serialization

Pattern的抽象：

- NFA <->状态转换图
- 节点：子Pattern
- 边：事件选择策略

Graph

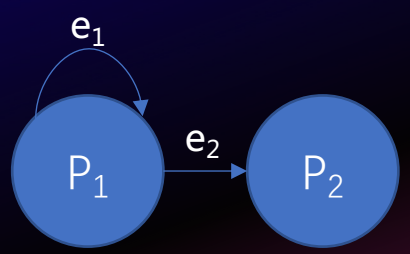
规则的(de)serialization 格式设计原则：

- 表达能力完整
- 方便序列化反序列化
- 易于拓展， 方便集成
- 可读可编辑

JSON

动态规则支持：(de)serialization

```
Pattern<Event, Event> pattern =
        Pattern.<Event>begin("start",
AfterMatchSkipStrategy.skipPastLastEvent())
                .where(new StartCondition("action ==
0"))
                .timesOrMore(3)
                .followedBy("end")
                .where(new EndCondition());
```




```
{
  "nodes": [
    {
      "name": "start",
      "quantifier": {
        "consumingStrategy": "SKIP_TILL_NEXT",
        "properties": [
          "LOOPING"
        ],
        "times": {
          "from": 3,
          "to": 3
        }
      },
      "condition": {
        "expression": "action == 0",
        "type": "AVIATOR"
      },
      "type": "ATOMIC"
    },
    ...
  ],
  "edges": [
    {
      "source": "start",
      "target": "end",
      "type": "SKIP_TILL_NEXT"
    }
  ],
  "afterMatchStrategy": {
    "type": "SKIP_PAST_LAST_EVENT"
  },
  "type": "COMPOSITE",
  ...
}
```

动态规则支持：拓展Condition

AviatorCondition:

- 结合Java反射机制，使用Aviator引擎解析表达式字符串

- 原理
  - Compile expression in constructor
  - Execute expression with variables in filter()

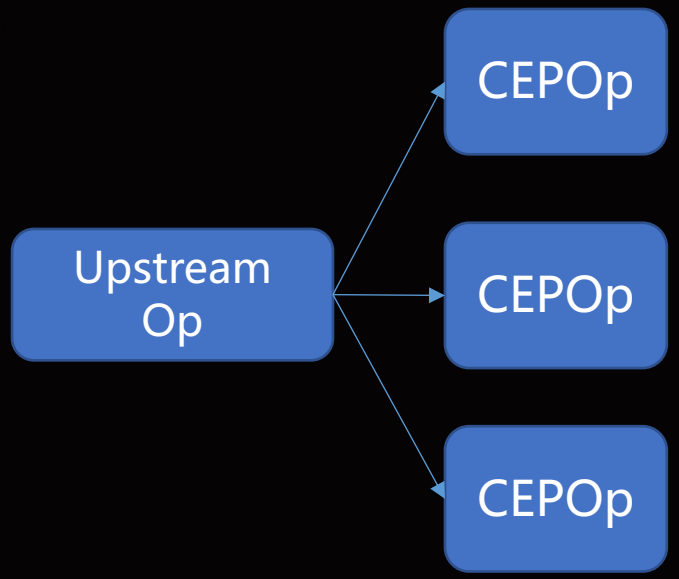
- 示例
  - AviatorCondition（‘action ==1 && price > 20’）
  - AviatorCondition（‘action ==0 && price > 50’）

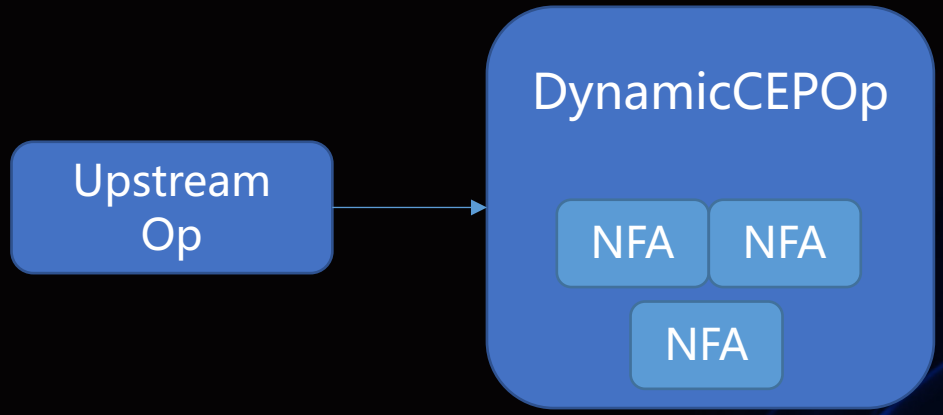
```
public class Event {
    private final int id;
    private final String name;

    private final double price;
    private final int action;
    private final long eventTime;
}
```

多规则支持

如何在同一输入流应用多条规则?
多个PatternStream, 多个CEPOperator，多个NFA -> 数据要复制多次

一个PatternStream, 一个CEPOperator，多个NFA -> 数据只需传递一次





动态CEP Demo

场景：广告投放中的实时反作弊

更多产品
实时计算Flink版
在目录中筛选
动态与公告
产品简介
产品计费
快速入门
Flink SQL作业快速入门
Flink JAR作业快速入门
数据库实时入仓快速入门
日志实时入仓快速入门
Flink动态CEP快速入门
工作空间管理
权限管理
作业开发
作业运维
作业高级配置
开发参考
产品迁移
API参考
SDK参考

Flink动态CEP快速入门

更新时间：2022-10-28 10:01 | 产品详情 相关技术圈

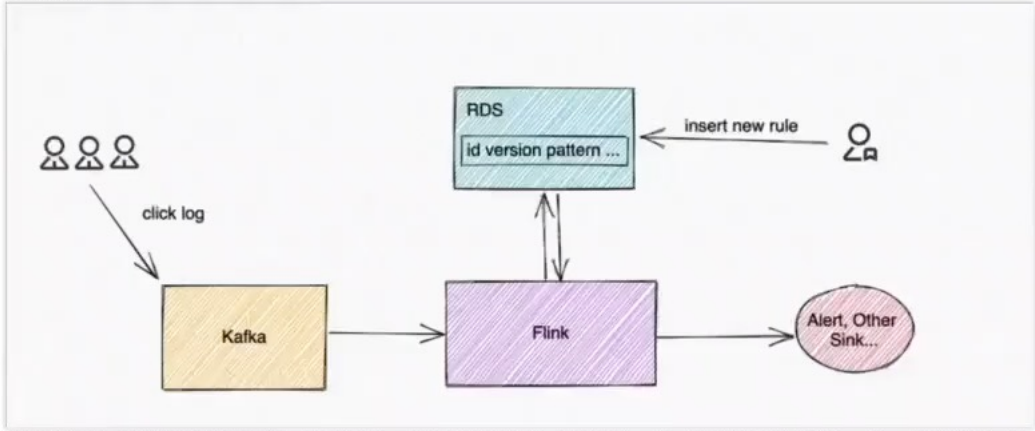
实时计算Flink版支持通过DataStream作业的方式运行支持规则动态更新的Flink CEP作业。本文结合实时营销中的反作弊场景，为您介绍如何基于Flink全托管快速构建一个动态加载最新规则来处理上游Kafka数据的Flink CEP作业。

背景信息

在电商平台投放广告时，广告主通常有预算限制。例如对于按点击次数计算费用的广告而言，如果有黑灰产构造虚假流量，攻击广告主，则会很快消耗掉正常广告主的预算，使得广告内容被提前下架。在这种情况下，广告主的利益受到了损害，容易导致后续的投诉与纠纷。

为了应对上述作弊场景，我们需要快速辨识出恶意流量，采取针对性措施（例如限制恶意用户、向广告主发送告警等）来保护用户权益。同时考虑到可能有意外因素（例如达人推荐、热点事件引流）导致流量骤变，我们也需要动态调整用于识别恶意流量的规则，避免损害正常用户的利益。

本文为您演示如何使用Flink动态CEP解决上述问题。我们假设客户的行为日志会被存放入消息队列Kafka中，Flink CEP作业会消费Kafka数据，同时会去轮询RDS数据库中的规则表，拉取策略人员添加到数据库的最新规则，并用最新规则去匹配事件。针对匹配到的事件，Flink CEP作业会发出告警或将相关信息写入其他数据存储中。示例中整体数据链路如下图所示。



实际演示中，我们会先启动Flink CEP作业，然后插入规则1：连续3条action为0的事件发生后，下一条事件的action仍非1，其业务含义为连续3次访问该产品后最后没有购买。在匹配到相应事件并进行处理后，我们会动态更新规则1内容为连续5条action为0的事件发生后，下一条事件的action仍非1，来应对流量整体增加的场景，同时插入一条规则2，它将和规则1的初始规则一样，用于辅助展示多规则支持等功能。当然，您也可以添加一个全新规则。

前提条件

- 已准备阿里云账号及账户余额。

本页导读

03 Flink CEP SQL语法增强

Flink CEP SQL示例

选择事件表

定义逻辑分区

定义事件顺序

定义CEP输出

定义序列模式

定义变量条件

```
SELECT T.user_name, T.first_a, T.last_a, T.count_a, T.b
FROM
  csv_source
  MATCH_RECOGNIZE (
    PARTITION BY user_name
    ORDER BY rowtime
    MEASURES
      FIRST (A.event_id) AS first_a,
      LAST (A.event_id) AS last_a,
      COUNT (A.event_id) AS count_a,
      B.event_id AS b
    PATTERN (A + B)
    DEFINE
      A AS A.event_type = 'A', B AS B.event_type = 'B'
  ) AS T;
```

Flink CEP SQL示例

源表

event_id	user_name	event_type	rowtime
0	Alice	A	1970-01-01 08:00:00.000
1	Alice	A	1970-01-01 08:00:01.000
2	Bob	A	1970-01-01 08:00:02.000
3	Alice	A	1970-01-01 08:00:03.000
4	Alice	B	1970-01-01 08:00:04.000
5	Alice	A	1970-01-01 08:00:05.000
6	Bob	B	1970-01-01 08:00:06.000

结果表

user_name	first_a	last_a	count_a	b
Alice	0	3	3	4
Alice	1	3	2	4
Alice	3	3	1	4
Bob	2	2	1	6

Flink CEP SQL语法增强

01 输出带时间约束模式的匹配超时序列

02 定义事件之间的连续性

03 定义循环模式中的连续性和贪婪性

01 输出带时间约束模式的匹配超时序列

案例场景：用户行为模式识别

用户从流量入口进入产品边界，执行一系列的操作后最终完成价值转化。识别整体流程周期在10分钟之内的高潜用户。

```
SELECT *
FROM
  user_action_table
  MATCH_RECOGNIZE (
    PARTITION BY user_name
    ORDER BY rowtime
    MEASURES
      A.rowtime AS action_a_time,
      B.rowtime AS action_b_time,
      C.rowtime AS action_c_time
    PATTERN (A B C) WITHIN INTERVAL '10' MINUTES
    DEFINE
      A AS A.action = 'actionA',
      B AS B.action = 'actionB',
      C AS C.action = 'actionC'
  ) AS user_action_seq_table;
```

≤10min

01 输出带时间约束模式的匹配超时序列

源表

event_id	user_name	action	rowtime
1	Alice	actionA	1970-01-01 08:00:00.000
2	Alice	actionB	1970-01-01 08:01:00.000
3	Bob	actionA	1970-01-01 08:02:00.000
4	Alice	actionC	1970-01-01 08:05:00.000
5	Bob	actionB	1970-01-01 08:10:00.000
6	Bob	actionC	1970-01-01 08:15:00.000

结果表

user_name	action_a_time	action_b_time	action_c_time
Alice	1970-01-01 08:00:00.000	1970-01-01 08:01:00.000	1970-01-01 08:05:00.000

01 输出带时间约束模式的匹配超时序列

```
SELECT *
FROM
  user_action_table
  MATCH_RECOGNIZE (
    PARTITION BY user_name
    ORDER BY rowtime
    MEASURES
      A.rowtime AS action_a_time,
      B.rowtime AS action_b_time,
      C.rowtime AS action_c_time
    ONE ROW PER MATCH SHOW TIMEOUT MATCHES
    PATTERN (A B C) WITHIN INTERVAL '10' MINUTES
    DEFINE
      A AS A.action = 'actionA',
      B AS B.action = 'actionB',
      C AS C.action = 'actionC'
  ) AS user_action_seq_table;
```

结果表

user_name	action_a_time	action_b_time	action_c_time
Alice	1970-01-01 08:00:00.000	1970-01-01 08:01:00.000	1970-01-01 08:05:00.000
Bob	1970-01-01 08:02:00.000	1970-01-01 08:10:00.000	<NULL>

02 定义事件之间的连续性

FLINK FORWARD # ASIA 2022 ONLINE | 实时即未来

源表

type	content
A	a1
B	b1
A	a2
C	c1
A	a3
B	b2
B	b3

条件变量

```
AFTER MATCH SKIP TO NEXT ROW
PATTERN (A B)
DEFINE
  A AS A.type = 'A',
  B AS B.type = 'B'
```

Java API	SQL	策略	样例匹配序列
A.next(B)	(A B)	严格连续：期望所有匹配事件严格的一个接一个出现，中间没有任何不匹配的事件。	{ a1 b1 } { a3 b2 }
A.followedBy(B)	(A {- X*? -} B) X为未在DEFINE中定义的变量，下同	松散连续：忽略匹配事件之间的不匹配事件。	{ a1 b1 } { a2 b2 } { a3 b2 }
A.followedByAny(B)	(A {- X* -} B)	非确定性松散连续：更进一步的松散连续，允许忽略掉一些匹配事件。	{ a1 b1 } { a2 b2 } { a3 b2 }
A.notNext(B)	(A [^B])	严格非连续：期望事件之后不紧接出现另一事件。	{ a2 }
A.notFollowedBy(B)	(A {- X*? -} [^B] A')	松散非连续：期望一个事件不出现在两个事件之间的任何地方。	{ a2 a3 }

03 定义循环模式中的连续性和贪婪性

FLINK FORWARD # ASIA 2022 ONLINE 实时即未来

源表

type	content
A	a1
B	b1
A	a2
A	a3
C	c1

条件变量

```
AFTER MATCH SKIP TO NEXT ROW
PATTERN (A+ C)
DEFINE
  A AS A.type = 'A',
  C AS C.type = 'A' OR C.type = 'C'
```

标识符	连续性	贪婪性	示例模式	等效Java API	样例匹配序列
无	严格连续	贪婪	(A+ C)	A.oneOrMore().consecutive() .greedy().next(C)	{a2 a3 c1} {a3 c1}
?	严格连续	非贪婪	(A+? C)	A.oneOrMore().consecutive() .next(C)	{a2 a3} {a3 c1}
??	松散连续	贪婪	(A+?? C)	A.oneOrMore().greedy().next(C)	{a1 a2 a3 c1} {a2 a3 c1} {a3 c1}
???	松散连续	非贪婪	(A+??? C)	A.oneOrMore().next(C)	{a1 a2 a3} {a2 a3} {a3 c1}

04 Flink CEP 未来规划

01 扩展动态CEP多规则能力到静态场景

02 动态CEP的JSON格式规则描述支持定义参数化的Condition

03 CEP SQL表达能力对齐Java API，增强对正则语法的支持

04 CEP SQL支持动态更新条件和模式定义

FLINK FORWARD # ASIA 2022 ONLINE | 实时即未来

THANK YOU

谢 谢 观 看