

# Flink CEP 在实时风控场景的落地与优化

演讲人-耿飙-阿里云计算平台事业部-开发工程师

DataFunSummit # 2023

云 & 科技  
Cloud & Technology



阿里云

DataFun.

# 目录 CONTENT

**01** Flink CEP介绍

**03** CEP SQL语法增强 & 性能优化

**02** 动态多规则支持

**04** 风控场景实际案例



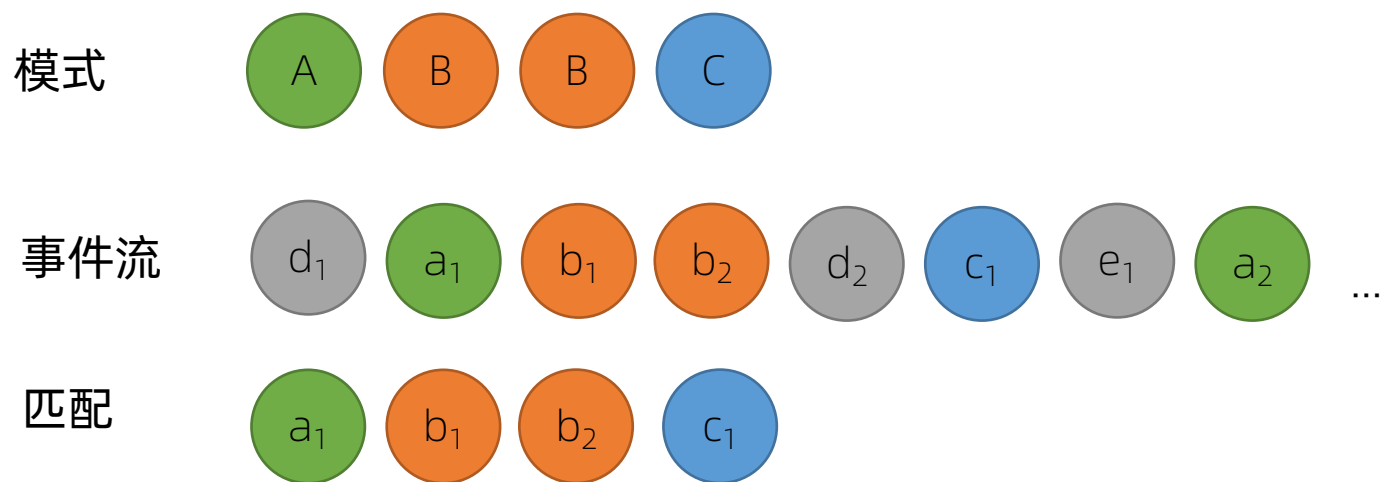
# 01

## Flink CEP介绍

DataFunSummit # 2023

# 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



# 02

## 动态多规则支持

DataFunSummit # 2023



# 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  
→ 3分钟内访问超过7次最终没有购买

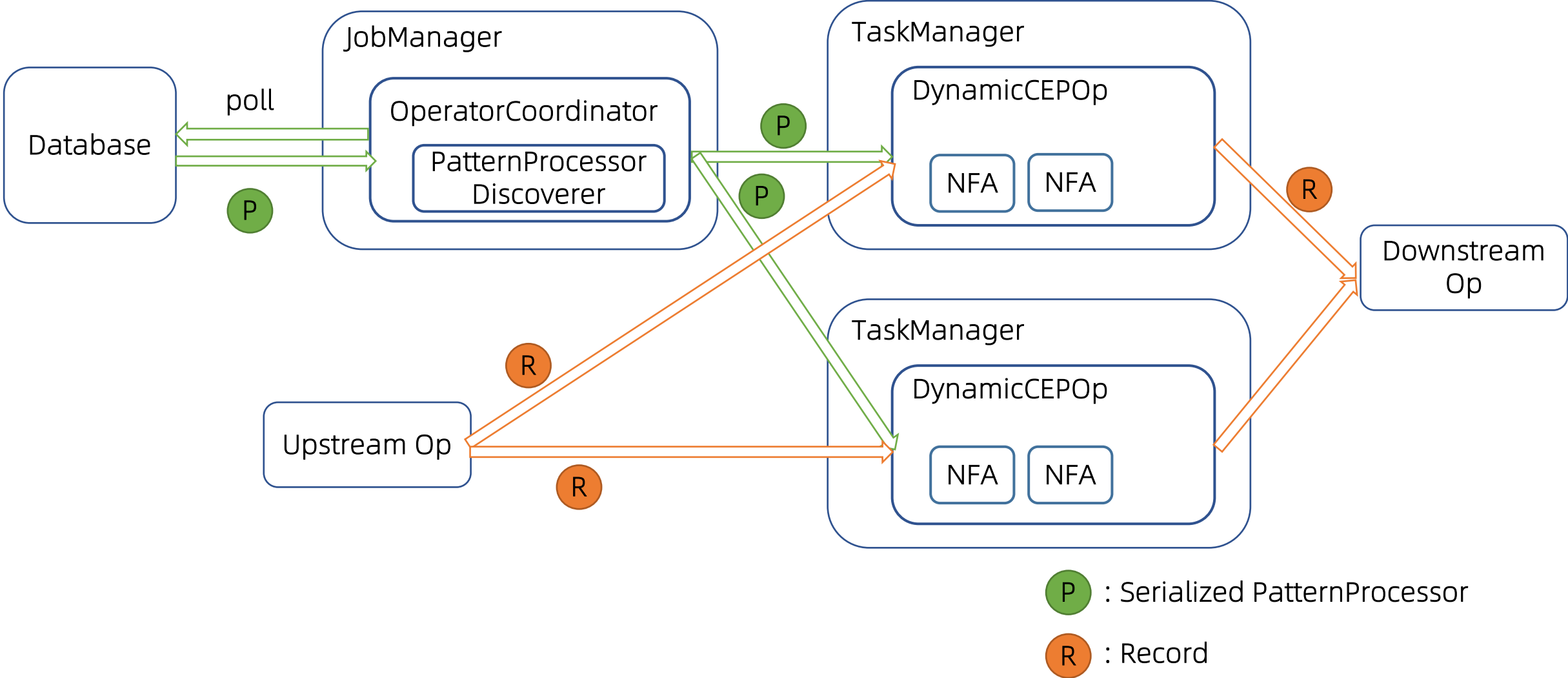
# 动态规则支持：设计

- 新增接口 (FLIP-200)
- PatternProcessor
  - id
  - version
  - timestamp
  - pattern
  - patternProcessorFunction
- PatternProcessorDiscoverer
- PatternProcessorManager

```
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



# 动态规则支持：(de)serialization



## Pattern的抽象：

- NFA  $\leftrightarrow$  状态转换图
- 子Pattern：节点
- 事件选择策略：边

*Graph*

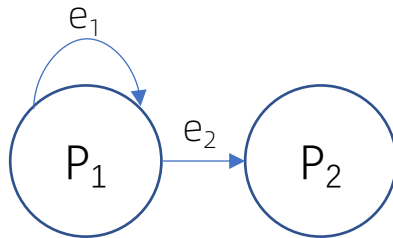
## 规则的(de)serialization 格式设计原则：

- 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;  
}
```

**GroovyCondition:** 支持Groovy语法，将Groovy表达式作为参数

**CustomArgsCondition:** 自定义参数

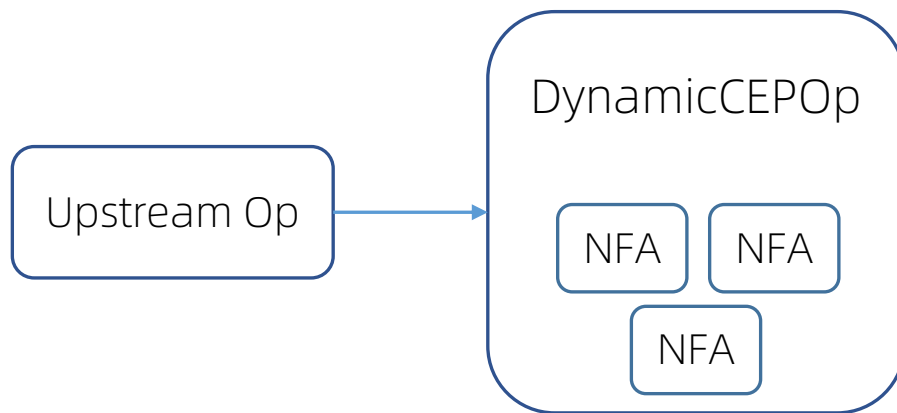
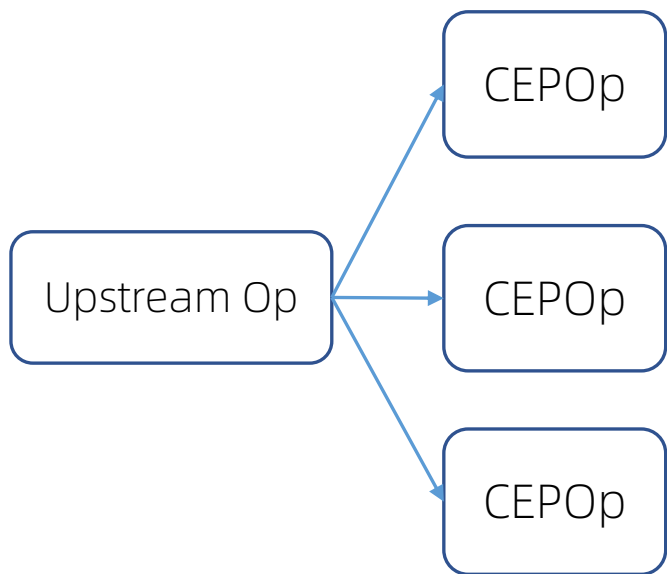


# 动态规则支持：多规则支持

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



# 动态规则支持：Demo



场景：广告投放中的实时反作弊

[https://help.aliyun.com/document\\_detail/459880.html](https://help.aliyun.com/document_detail/459880.html)

<https://github.com/RealtimeCompute/ververica-cep-demo>



03

# CEP SQL语法增强&性能优化

DataFunSummit # 2023

# Flink CEP SQL: 介绍

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

### 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分钟之内的高质量用户。

```
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;
```

# Flink CEP SQL：语法增强



## 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

# Flink CEP SQL: 语法增强

## 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

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])	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。	无匹配

## 03 定义循环模式中的连续性和贪婪性

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 循环模式指定停止条件 (Until)

源表

type	content
A	a1
D	d1
A	a2
B	b1
A	a3
C	c1

条件变量

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

模式	等效Java API	样例匹配序列	说明
(A+ C)	A.oneOrMore().consecutive().greedy().next(C)	{a2 b1 a3 c1} {b1 a3 c1} {a3 c1}	以a或b开头的事件都能匹配A模式，A模式内部和AC之间为严格连续。由于a1、a2之间存在d1，无法从a1开始匹配
(A+{B}C)	A.oneOrMore().consecutive().greedy().until(B).next(C)	{a3 c1}	A循环模式增加了until B条件，AC之间仍为严格连续。由于a2开始的循环模式需要在b1处结束，无法满足与c1之间的严格连续要求
(A+{B}{-X*?-*}C)	A.oneOrMore().consecutive().greedy().until(B).followedBy(C)	{a2 c1} {a3 c1}	AC之间为松散连续，以a2开始的循环模式在b1处结束，并跳过b1、a3匹配c1
(A+??{B}{-X*?-*}C)	A.oneOrMore().greedy().until(B).followedBy(C)	{a1 a2 c1} {a2 c1} {a3 c1}	循环模式A内部为松散连续，可跳过d1并结束于b1，匹配a1、a2

# Flink CEP SQL: 语法增强



## 05 组合模式(Group Pattern)

组合模式 (group pattern)：将多个模式组合为一个整体用在next()、followedBy()和followedByAny()函数中，并支持整体的循环。

在阿里云实时计算Flink版的SQL作业中使用SQL标准中的括号语法(...)来定义组合模式，支持使用循环量词如+、\*、{3, }等。

PATTERN (A (B C\*)+? D)

```
Pattern.<String>begin("A").where(...)  
.next( Pattern.<String>begin("B").where(...)  
    .next("C").where(...).oneOrMore().optional().greedy().consecutive())  
.oneOrMore().consecutive()  
.next("D").where(...)
```

$a_1 b_1 b_2 c_1 b_3 c_2 c_3 d_1$   
 $\rightarrow a_1 b_1 b_2 c_1 c_3 d_1$

```
SELECT *  
FROM MyTable MATCH_RECOGNIZE (  
  ORDER BY rowtime  
  MEASURES  
    A.id AS aid,  
    FIRST(B.id) AS b1_id,  
    FIRST(B.id, 1) AS b2_id,  
    FIRST(C.id) AS c1_id,  
    LAST(C.id) AS c3_id,  
    D.id AS did  
  PATTERN (A (B C*)+? D)  
  DEFINE  
    A AS type = 'A',  
    B AS type = 'B',  
    C AS type = 'C',  
    D AS type = 'D'  
) AS T
```



## 06 AFTER MATCH NO SKIP策略

SKIP\_TO\_NEXT\_ROW：丢弃以相同事件开始的所有部分匹配（CEP SQL默认策略）

NO\_SKIP：每个成功的匹配都会被输出（Java API默认策略）

模式：a b+，输入：a<sub>1</sub> b<sub>1</sub> b<sub>2</sub> b<sub>3</sub>：

跳过策略	结果	描述
NO_SKIP	a b1 a b1 b2 a b1 b2 b3	找到匹配a b1之后，不会丢弃任何结果。
SKIP_TO_NEXT	a b1	找到匹配a b1之后，会丢弃所有以a开始的部分匹配。这意味着不会产生a b1 b2和a b1 b2 b3了。

阿里云实时计算Flink版扩展了SQL标准中的AFTER MATCH语句，可通过AFTER MATCH NO SKIP语句来声明NO\_SKIP策略，NO\_SKIP策略在完成一条序列的匹配时，不会终止或丢弃其他已经开始的匹配过程。

- 减少State访问
  - 增加Cache、优化onEvent/ProcessingTime()实现
- 修复State泄漏
  - <https://issues.apache.org/jira/browse/FLINK-23314>
  - 对于部分生命周期较短的key，和其相关的computationStates没有及时清理，导致State不断增大。当key包含timestamp或随机ID时，容易出现该问题
  - 及时清除该key
- Tip：使用Flink1.16及之上的Flink CEP版本，减少Timer的注册，大大减少作业的CPU消耗（10x）
  - <https://issues.apache.org/jira/browse/FLINK-23890>

# 04

## 风控场景典型应用

DataFunSummit # 2023

- 交易风控
  - 一段时间内某个IP退款超过一定金额，触发熔断
  - 一段时间内某个IP退款超过一定次数，触发熔断
- 内容风控
  - 某用户在X分钟内发布了超过Y条帖子，则进行账号禁言或其他处理
- 物联网
  - 设备上报埋点到日志存储上，有成功和失败信息。如果某个设备连续发生10次以上的某类异常，并且超过15分钟未恢复则告警
- 网络安全
  - 检测到某台电脑的行为满足“点击钓鱼邮件”、“下载异常文件”、“执行远程代”等模式后，触发报警

# 风控场景典型应用



获取该子Pattern之前匹配的事件：

context.getEventsForPattern()

```
new IterativeCondition<Event>() {
    @Override
    public boolean filter(Event value,
        Context<Event> ctx) throws Exception {
        double amount = 0;
        for (Event evnt :
            ctx.getEventsForPattern("Refund")) {
            amount += event.getPrice();
        }
        return amount >= 1000;
    }
}
```

支持定义相邻事件之间的时间间隔：

WithinType.PREVIOUS\_AND\_CURRENT

```
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));
```





感谢观看