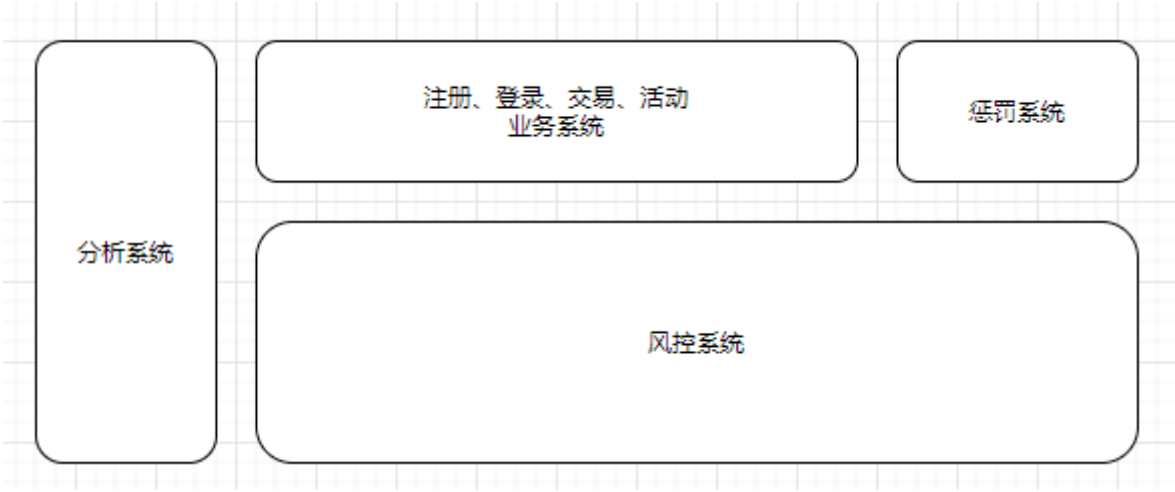
企业级风控平台架构建设

1. 风控背景

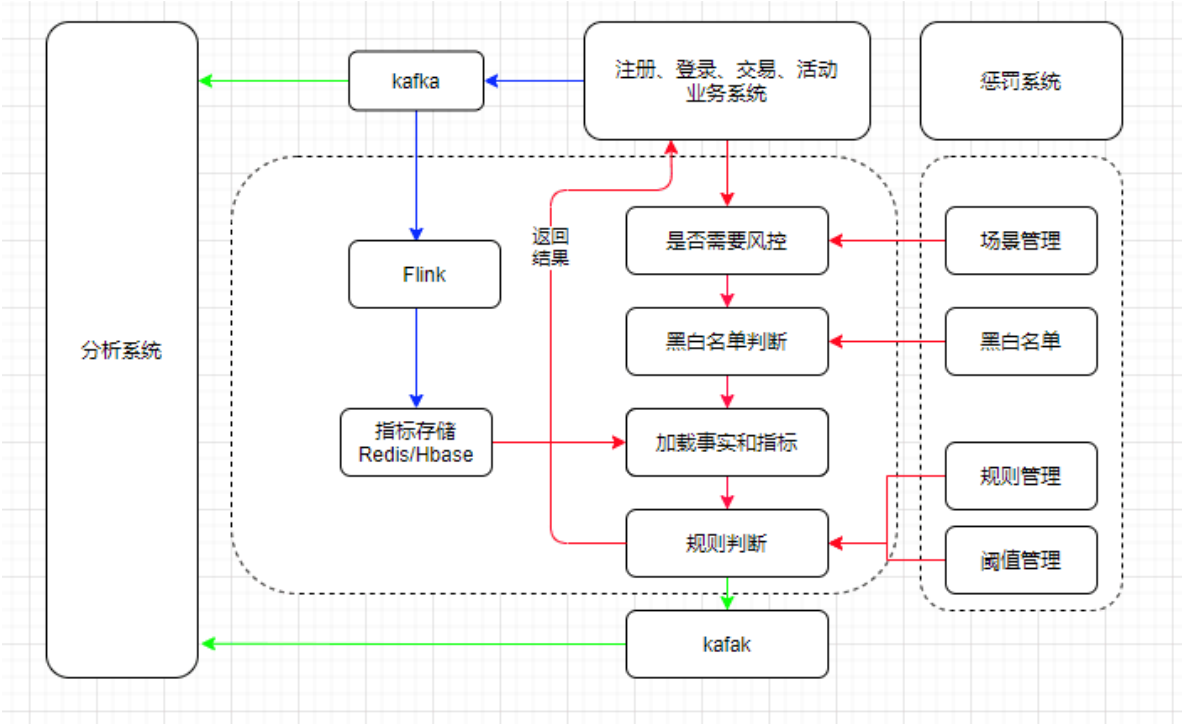
- 互联网场景中，典型的风控场景包括：注册风控、登陆风控、交易风控、活动风控等，而风控的最佳效果是防患于未然，所以事前事中和事后三种实现方案中，又以事前预警和事中控制最好。
- 这要求风控系统一定要有实时性。我们将实时风控架构作为重点讲解

2. 总体架构

- 风控是业务场景的产物，风控系统直接服务于业务系统，与之相关的还有惩罚系统和分析系统，各系统关系与角色如下：



- 风控系统有规则和模型两种技术路线，规则的优点是简单直观、可解释性强、灵活，所以长期活跃在风控系统之中，但缺点是容易被攻破，一但被黑产猜中就会失效，于是在实际的风控系统中，往往需要再结合上基于模型的风控环节来增加健壮性。
- 规则就是针对事物的条件判断，我们针对注册、登陆、交易、活动分别假设几条规则，比如：
    - 用户名与身份证姓名不一致；
    - 某 IP 最近 1 小时注册账号数超过 10 个；
    - 某账号最近 3 分钟登陆次数大于 5 次；
    - 某账号群体最近 1 小时购买优惠商品超过 100 件；
    - 某账号最近 3 分钟领券超过 3 张；
- 规则可以组合成规则组
    - 事实：即被判断的主体和属性，如上面规则的账号及登陆次数、IP 和注册次数等；
    - 条件：判断的逻辑，如某事实的某属性大于某个指标；
    - 指标阈值：判断的依据，比如登陆次数的临界阈值，注册账号数的临界阈值等；
    - 规则可由运营专家凭经验填写，也可由数据分析师根据历史数据发掘，但因为规则在与黑产的攻防之中会被猜中导致失效，所以无一例外都需要动态调整。
- 基于上边的讨论，我们设计一个风控系统方案如下：

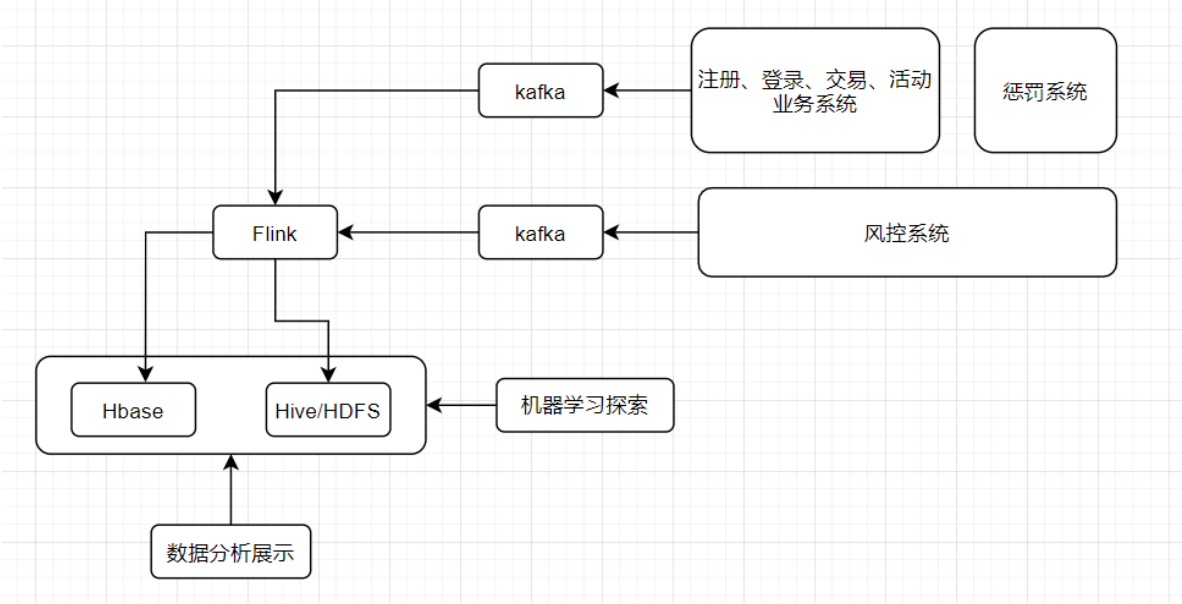


- 该系统有三条数据流向：
  - 实时风控数据流：由红线标识，同步调用，为风控调用的核心链路
  - 准实时指标数据流：由蓝线标识，异步写入，为实时风控部分准备指标数据
  - 准实时/离线分析数据流：由绿线标识，异步写入，为分析系统的提供数据
- **实时风控**
  - 实时风控是整个系统的核心，被业务系统同步调用，完成对应的风控判断。
  - 前面提到规则往往由人编写并且需要动态调整，所以我们会把风控判断部分与规则管理部分拆开。规则管理后台为运营服务，由运营人员去进行相关操作：
  - **场景管理**：决定某个场景是否实施风控，比如活动场景，在活动结束后可以关闭该场景；
  - **黑白名单**：人工/程序找到系统的黑白名单，直接过滤
  - **规则管理**：管理规则，包括增删或修改，比如登陆新增 IP 地址判断，比如下单新增频率校验等；
  - **阈值管理**：管理指标的阈值，比如规则为某 IP 最近 1 小时注册账号数不能超过 10 个，那 1 和 10 都属于阈值；
  - 基于管理后台，那规则判断部分的逻辑也就十分清晰了，分别包括前置过滤、事实数据准备、规则判断三个环节。
- **前置过滤**
  - 业务系统在特定事件（如注册、登陆、下单、参加活动等）被触发后同步调用风控系统，附带相关上下文，比如 IP 地址，事件标识等，规则判断部分会根据管理后台的配置决定是否进行判断，如果是，接着进行黑白名单过滤，都通过后进入下一个环节。
- **实时数据准备**
  - 在进行判断之前，系统必须要准备一些事实数据，比如：
    - 注册场景，假如规则为单一 IP 最近 1 小时注册账号数不超过 10 个，那系统需要根据 IP 地址去 Redis/Hbase 找到该 IP 最近 1 小时注册账号的数目，比如 15；
    - 登陆场景，假如规则为单一账号最近 3 分钟登陆次数不超过 5 次，那系统需要根据账号去 Redis/Hbase 找到该账号最近 3 分钟登陆的次数，比如 8；
- **规则判断**
  - 在得到事实数据之后，系统会根据规则和阈值进行判断，然后返回结果，整个过程便结束了。
  - 整个过程逻辑上是清晰的，我们常说的规则引擎主要在这部分起作用，一般来说这个过程有两种实现方式：

        - 借助成熟的规则引擎，比如 Drools，Drools 和 Java 环境结合的非常好，本身也非常完善，支持很多特性。
        - 基于 Groovy 等动态语言完成
    - 这两种方案都支持规则的动态更新

- **准实时数据流**
    - 这部分属于后台逻辑，为风控系统服务，准备事实数据。
    - 把数据准备与逻辑判断拆分，是出于系统的性能/可扩展性的角度考虑的。前边提到，做规则判断需要事实的相关指标，比如最近一小时登陆次数，最近一小时注册账号数等等，这些指标通常有一段时间跨度，是某种状态或聚合，很难在实时风控过程中根据原始数据进行计算，因为风控的规则引擎往往是无状态的，不会记录前面的结果。
    - 同时，这部分原始数据量很大，因为用户活动的原始数据都要传过来进行计算，所以这部分往往由一个流式大数据系统来完成。
    - 业务系统把埋点数据发送到 Kafka；
        - Flink 订阅 Kafka，完成原子粒度的聚合；
    - Flink 仅完成原子粒度的聚合是和规则的动态变更逻辑相关的。举例来说，在注册场景中，运营同学会根据效果一会要判断某 IP 最近 1 小时的注册账号数，一会要判断最近 3 小时的注册账号数，一会又要判断最近 5 小时的注册账号数……也就是说这个最近 N 小时的 N 是动态调整的。那 Flink 在计算时只应该计算 1 小时的账号数，在判断过程中根据规则来读取最近 3 个 1 小时还是 5 个 1 小时，然后聚合后进行判断。因为在 Flink 的运行机制中，作业提交后会持续运行，如果调整逻辑需要停止作业，修改代码，然后重启，相当麻烦；同时因为 Flink 中间状态的问题，重启还面临着中间状态能否复用的问题。所以假如直接由 Flink 完成 N 小时的聚合的话，每次 N 的变动都需要重复上面的操作，有时还需要追数据，非常繁琐。
        - Flink 把汇总的指标结果写入 Redis 或 Hbase，供实时风控系统查询。两者问题都不大，根据场景选择即可。
        - 通过把数据计算和逻辑判断拆分开来并引入 Flink，我们的风控系统可以应对极大的用户规模。
- **分析系统**
    - 前面的东西静态来看是一个完整的风控系统，但动态来看就有缺失了，这种缺失不体现在功能性上，而是体现在演进上。即如果从动态的角度来看一个风控系统的话，我们至少还需要两部分，一是衡量系统的整体效果，一是为系统提供规则/逻辑升级的依据。
    - 在衡量整体效果方面，我们需要：
        - 判断规则是否失效，比如拦截率的突然降低；
        - 判断规则是否多余，比如某规则从来没拦截过任何事件；
        - 判断规则是否有漏洞，比如在举办某个促销活动或发放代金券后，福利被领完了，但没有达到预期效果；
    - 在为系统提供规则/逻辑升级依据方面，我们需要：
        - **发现全局规则**：比如某人在电子产品的花费突然增长了 100 倍，单独来看是有问题的，但整体来看，可能很多人都出现了这个现象，原来是苹果发新品了。
        - **识别某种行为的组合**：单次行为是正常的，但组合是异常的，比如用户买菜刀是正常的，买车票是正常的，买绳子也是正常的，去加油站加油也是正常的，但短时间内同时做这些事情就不是正常的。
        - **群体识别**：比如通过图分析技术，发现某个群体，然后给给这个群体的所有账号都打上群体标签，防止出现那种每个账号表现都正常，但整个群体却在集中薅羊毛的情况。
    - 这便是分析系统的角色定位，在他的工作中有部分是确定性的，也有部分是探索性的，为了完成这种工作，该系统需要尽可能多的数据支持，如：
        - 业务系统的数据，业务的埋点数据，记录详细的用户、交易或活动数据；
        - 风控拦截数据，风控系统的埋点数据，比如某个用户在具有某些特征的状态下因为某条规则而被拦截，这条拦截数据本身就是一个事件数据；

- 这是一个典型的大数据分析场景，架构也比较灵活



- 相对来说这个系统是最开放的，既有固定的指标分析，也可以使用机器学习/数据分析技术发现更多新的规则或模式。

3.第一版需求开发

- 数据源介绍：

```
数据源：
public class ClientLog {
    private String userNo;  --用户ID
    private String userName;    --用户名
    private String appId;   --app的编号
    private String appVersion;  --APP的版本
    private String carrier;     --运营商
    private String imei;    --设备编号
    private String deviceType;  --设备类型
    private String ip;          --客户端IP
    private String netType;     --网络类型，WIFI,4G,5G
    private String osName;      --操作系统类型
    private String osVersion;   --操作系统版本
    private String sessionId;   --会话ID
    private String detailTime;  --创建详细时间
    private String eventId;     --事件编号
    private String createTime;  --创建日期
    private String eventType;   --事件类型
    private String gps;         --经纬度信息
    private String addr;
    private Map<String,String> properties;  --事件详细属性
}

用生成用户行为数据写入kafka
#创建topic
kafka-topics.sh --zookeeper 192.168.0.82:2181 --create --topic ys_client_log --
partitions 1 --replication-factor 1

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server 192.168.0.82:9092 --topic
ys_client_log --from-beginning
```

```
#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list 192.168.0.82:9092 --
topic ys_client_log -time -1 --offsets 1

假设公司已有的用户画像库，存储在hbase中，表结构为如下
create_namespace 'events_db'
create 'events_db:users', 'profile', 'region', 'registration'

企业实际业务数据：
{"before":null,"after":{"id":1,"name":"张
三","pwd":"123","cid":1,"age":23,"addr":"北京","time":1610582400000},"source":
{"version":"1.5.4.Final","connector":"mysql","name":"mysql_binlog_source","ts_ms
":512312312,"snapshot":"false","db":"user","sequence":null,"table":"user","serve
r_id":0,"gtid":null,"file":"","pos":0,"row":0,"thread":null,"query":null},"op":"
r","ts_ms":1637659598816,"transaction":null}
{"before":null,"after":{"id":1367365308891332609,"name":"后裔
3","pwd":"999","cid":2,"age":18,"addr":"武汉","time":1614869069000},"source":
{"version":"1.5.4.Final","connector":"mysql","name":"mysql_binlog_source","ts_ms
":1637659598828,"snapshot":"false","db":"user","sequence":null,"table":"user","s
erver_id":0,"gtid":null,"file":"","pos":0,"row":0,"thread":null,"query":null},"o
p":"r","ts_ms":1637659598828,"transaction":null}
```

- 工具类介绍
- 生成日志数据
- 生成画像系统
- 是否异地登录，是否常用地登录，修改手机号，5分钟登录限制，后期动态配置指标，完成第一版本需求开发
- 优化数据传输效率（protobuf）
    - protobuf的应用

1.1 Protobuf是什么?

- Protocol Buffers，是Google公司开发的一种数据描述语言，类似于XML能够将结构化数据序列化，可用于数据传输、通信协议等方面。数据传输相比于json，Protobuf有更高的转化效率，时间效率和空间效率都是JSON的3-5倍。
- XML、JSON、Protobuf 都具有数据结构化和数据序列化的能力
- XML、JSON 更注重 **数据结构化**，关注人类可读性和语义表达能力。Protobuf 更注重 **数据序列化**，关注效率、空间、速度，可读性差，语义表达能力不足
- Protobuf 的应用场景更为明确，XML、JSON 的应用场景更为丰富

1.2 Protobuf有什么用?

- Xml、Json是目前常用的数据交换格式，它们直接使用字段名称维护序列化后类实例中字段与数据之间的映射关系，一般用字符串的形式保存在序列化后的字节流中。消息和消息的定义相对独立，可读性较好。但序列化后的数据字节很大，序列化和

  反序列化的时间较长，数据传输效率不高。

- Protobuf和Xml、Json序列化的方式不同，采用了二进制字节的序列化方式，用字段索引和字段类型通过算法计算得到字段之前的关系映射，从而达到更高的时间效率和空间效率，特别适合对数据大小和传输速率比较敏感的场合使用。

	XML	JSON	Protobuf
数据结构	结构一般复杂	结构简单	结构比较复杂
数据存储方式	文本	文本	二进制
数据存储大小	大	一般	小
解析效率	慢	一般	快
跨语言支持	非常多	多	一般
开发成本	比较繁琐	非常简单	一般
学习成本	一般	低	一般

1.4 Protobuf的优势和弊端

- 优点：protobuf的空间效率是JSON的3-5倍，时间效率要高，对于数据大小敏感，传输效率高的模块可以采用protobuf库。
- 缺点：消息结构可读性不高，序列化后的字节序列为二进制序列不能简单的分析有效性。

1.5 Protobuf 语法简介

- 关键字 message
  - 代表了实体结构，由多个消息字段(field)组成
- 消息字段(field):
  - 包括数据类型、字段名、字段规则、字段唯一标识、默认值
- 枚举的第一个常量名的编号必须为 0
  - 同一个 proto 文件中，多个枚举之间不允许定义相同的常量名
- 数据类型:

.proto Type	C++ Type	Java Type	Python Type[2]	Go Type	Ruby Type	C# Type	PHP Type	Dart Type
double	double	double	float	float64	Float	double	float	double
float	float	float	float	float32	Float	float	float	double
int32	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
int64	int64	long	int/long[3]	int64	Bignum	long	integer/string[5]	Int64
uint32	uint32	int[1]	int/long[3]	uint32	Fixnum or Bignum (as required)	uint	integer	int
uint64	uint64	long[1]	int/long[3]	uint64	Bignum	ulong	integer/string[5]	Int64
sint32	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
sint64	int64	long	int/long[3]	int64	Bignum	long	integer/string[5]	Int64
fixed32	uint32	int[1]	int/long[3]	uint32	Fixnum or Bignum (as required)	uint	integer	int
fixed64	uint64	long[1]	int/long[3]	uint64	Bignum	ulong	integer/string[5]	Int64
sfixed32	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
sfixed64	int64	long	int/long[3]	int64	Bignum	long	integer/string[5]	Int64
bool	bool	boolean	bool	bool	TrueClass/FalseClass	bool	boolean	bool
string	string	String	str/unicode[4]	string	String (UTF-8)	string	string	String
bytes	string	ByteString	str	[]byte	String (ASCII-8BIT)	ByteString	string	List<int>

- proto3 中，数据的默认值不再支持自定义，而是由程序自行推倒：
  - string：默认值为空
  - bytes：默认值为空
  - bools：默认值为 false
  - 数字类型：默认值为 0
  - 枚举类型： 默认为定义的第一个元素，并且编号必须为 0
  - message 类型：默认值为 DEFAULT_INSTANCE，其值相当于空的 message
- 字段规则：
  - required：必须初始化字段，如果没有赋值，在数据序列化时会抛出异常
  - optional：可选字段，可以不必初始化。
  - repeated：数据可以重复(相当于java中的Array或List)
  - 字段唯一标识：序列化和反序列化将会使用到

Protobuf（Protocol Buffers）是一种由Google开发的灵活、高效的自动化机制，用于序列化结构化数据，类似于XML但更小、更快、更简单。它是一种语言无关、平台无关的序列化格式，广泛用于数据存储、通信协议等方面。

特点：

1. **跨平台**：支持多种编程语言，如C++、Java、Python等。
2. **跨语言**：定义一次数据结构，可以在多种编程语言中使用。
3. **高效**：序列化后的数据体积小，解析速度快。
4. **可扩展**：可以向后兼容，添加或删除字段不会影响已有数据结构。

使用步骤：

1. **定义数据结构**：使用`.proto`文件定义数据结构。
2. **编译**：使用Protobuf编译器（`protoc`）生成对应语言的数据访问类。
3. **使用**：在代码中使用生成的类进行数据的序列化和反序列化。

```
        <dependency>
            <groupId>com.google.protobuf</groupId>
            <artifactId>protobuf-java</artifactId>
            <version>3.19.4</version>
        </dependency>
        <dependency>
            <groupId>com.googlecode.protobuf-java-format</groupId>
            <artifactId>protobuf-java-format</artifactId>
            <version>1.2</version>
        </dependency>



<build>
        <extensions>
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId>
                <version>1.6.2</version>
            </extension>
        </extensions>

        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <!--添加编译proto文件的编译程序和对应的编译插件-->
            <plugin>
                <groupId>org.xolstice.maven.plugins</groupId>
                <artifactId>protobuf-maven-plugin</artifactId>
                <version>0.5.0</version>
                <extensions>true</extensions>
                <configuration>
                    <!-- 工具版本 -->

 <protocArtifact>com.google.protobuf:protoc:3.1.0:exe:${os.detected.classifier}
</protocArtifact>
                    <!--默认值，proto源文件路径-->

 <protoSourceRoot>${project.basedir}/src/main/java/com/yushu/proto</protoSourceR
oot>
                    <!--默认值，proto目标java文件路径-->
```

```
 <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                    <!--设置是否在生成java文件之前清空outputDirectory的文件，默认值为true，设置为false时也会覆盖同名文件-->
                    <clearOutputDirectory>false</clearOutputDirectory>

                </configuration>
                <executions>
                    <execution>
                        <!--在执行mvn compile的时候会执行以下操作-->
                        <phase>compile</phase>
                        <goals>
                            <!--生成OuterClass类-->
                            <goal>compile</goal>
                            <!--生成Grpc类-->
                            <!--goal>compile-custom</goal-->
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

示例：

新建有一个`.proto`文件定义

```
syntax = "proto3"; //定义版本

option java_package = "com.yushu.stream"; // 生成类的包名，注意：会在指定路径下按照该包名的定义来生成文件夹
option java_outer_classname="DemoProtos"; // 生成类的类名，注意：下划线的命名会在编译的时候被自动改为驼峰命名

// 定义一个 message
message User {
  int32 id = 1; // int 类型
  string name = 2; // string 类型
  string email = 3;
  Gender gender = 4; // 引用下面定义的 Gender 枚举类型
  repeated PhoneNumber phone = 5; // 引用下面定义的 PhoneNumber 类型的 message
  map<string, string> tags = 6; // map 类型

    // 定义一个枚举类型
    enum Gender {
      DEFAULT = 0;
      MALE = 1;
      FEMALE = 2; // 枚举值通常大写
    }

    // 定义一个 message 用于电话号码
    message PhoneNumber {
        string number = 1;
```

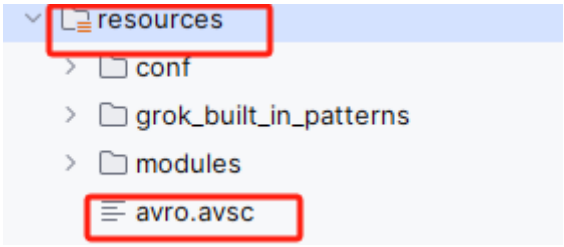
```
        PhoneType type = 2;

        enum PhoneType {
            MOBILE = 0;
            HOME = 1;
            WORK = 2;
        }
    }
}
```

```
public static void main(String[] args) {
        try {
            /*
              第一步：生成 personTest 对象
              */
            // personTest 构造器
            DemoProtos.User.Builder userBuilder = DemoProtos.User.newBuilder();
            // personTest 赋值
            userBuilder.setName("zmm");
            userBuilder.setEmail("1352087xxxx@163.com");
            userBuilder.setGender(DemoProtos.User.Gender.MALE);
            // 内部的 PhoneNumber 构造器
            DemoProtos.User.PhoneNumber.Builder phoneNumberBuilder =
DemoProtos.User.PhoneNumber.newBuilder();
            // PhoneNumber 赋值

 phoneNumberBuilder.setType(DemoProtos.User.PhoneNumber.PhoneType.MOBILE);
            phoneNumberBuilder.setNumber("1352087xxxx");
            // personTest 设置 PhoneNumber
            userBuilder.addPhone(phoneNumberBuilder);
            // 生成 personTest 对象
            DemoProtos.User buildUser = userBuilder.build();
            /*
              Step2：序列化和反序列化
              */
            // 方式一 byte[]：
            // 序列化
            byte[] bytes = buildUser.toByteArray();
            // 反序列化
            DemoProtos.User user = DemoProtos.User.parseFrom(bytes);
            System.out.printf("反序列化得到的信息，姓名：%s，性别：%d，手机号： %s%n",
user.getName(), user.getGenderValue(), user.getPhone(0).getNumber());
            // 方式二 ByteString：
            // 序列化
            ByteString bytes1 = buildUser.toByteString();
            System.out.println("序列化得到的信息:" + bytes1.toString());
            // 反序列化
            DemoProtos.User user1 = DemoProtos.User.parseFrom(bytes1);
            System.out.printf("反序列化得到的信息，姓名：%s，性别：%d，手机号： %s%n",
user1.getName(), user1.getGenderValue(), user1.getPhone(0).getNumber());
            // 方式三 InputStream
            // 粘包,将一个或者多个protobuf 对象字节写入 stream
            // 序列化
            ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
```

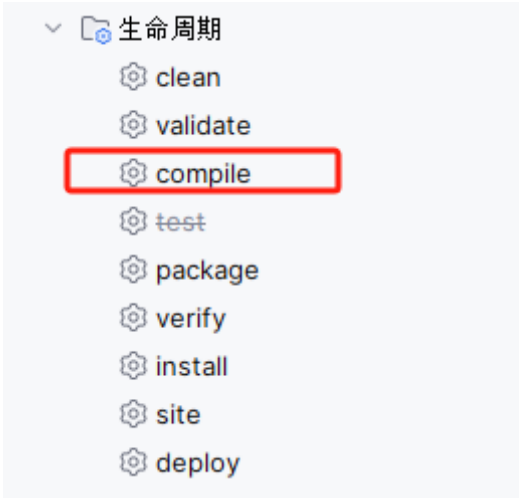
```
            buildUser.writeDelimitedTo(byteArrayOutputStream);
            // 反序列化，从 steam 中读取一个或者多个 protobuf 字节对象
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(byteArrayOutputStream.toByteArray());
            DemoProtos.User user3 =
DemoProtos.User.parseDelimitedFrom(byteArrayInputStream);
            System.out.printf("反序列化得到的信息，姓名：%s，性别：%d，手机号： %s%n",
user3.getName(), user3.getGenderValue(), user3.getPhone(0).getNumber());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

扩展一种avro 的

```
        <plugin>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-maven-plugin</artifactId>
            <version>1.10.0</version>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>schema</goal>
                    </goals>
                    <configuration>

<sourceDirectory>${project.basedir}/src/main/resources/</sourceDirectory>

<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
```



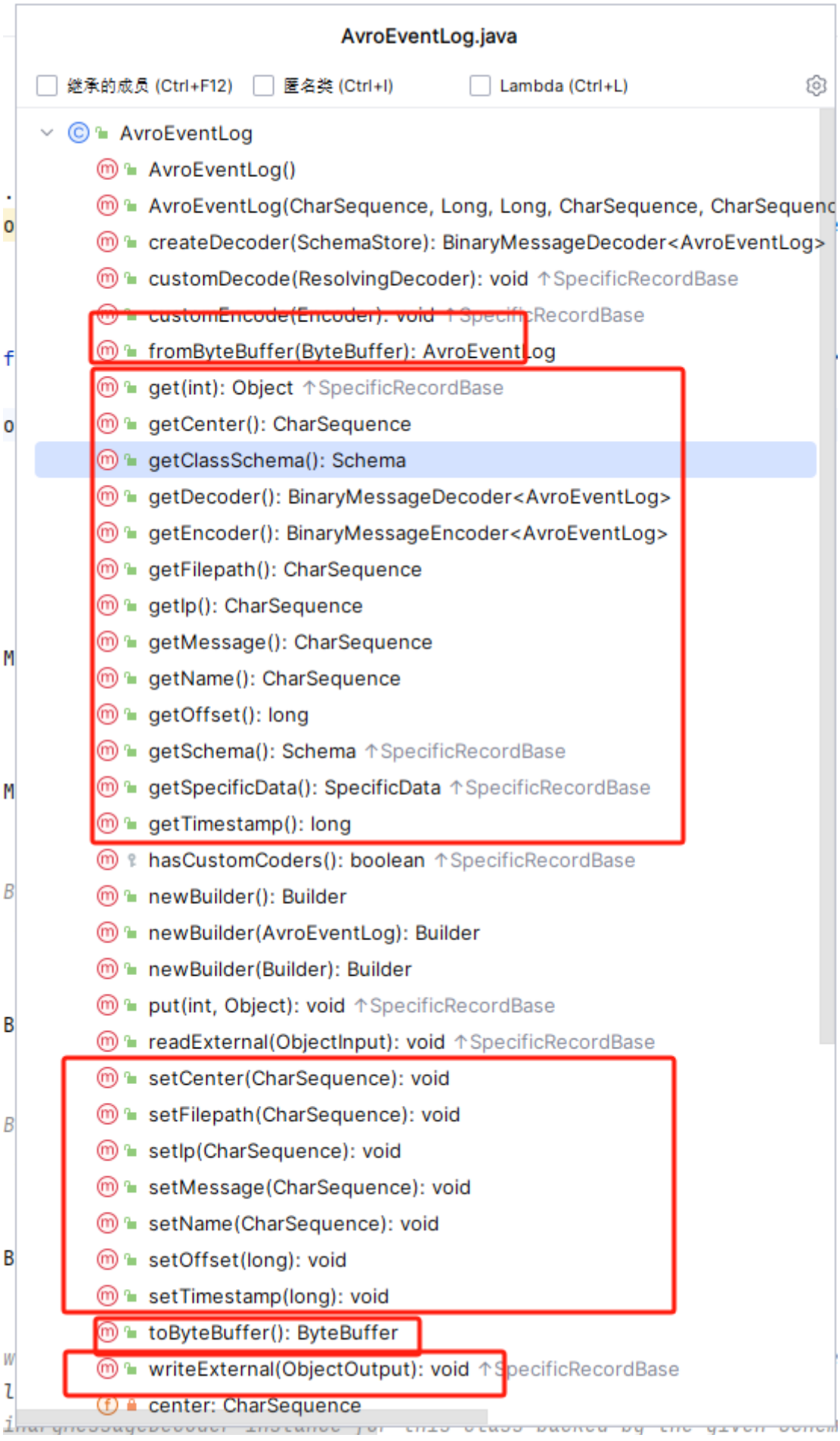
```
{"namespace": "com.[illegible].avro",
 "type": "record",
 "name": "AvroEventLog",
 "fields": [
     {"name": "message",  "type": "string"},
     {"name": "timestamp",  "type": "long"},
     {"name": "offset",  "type": "long"},
     {"name": "ip",  "type": ["string","null"]},
     {"name": "center",  "type": ["string","null"]},
     {"name": "filepath",  "type": ["string","null"]},
     {"name": "name",  "type": ["string","null"]}
 ]
}
```



AvroEventLog.java

```
import ...

45个用法
@org.apache.avro.specific.AvroGenerated
public class AvroEventLog extends org.apache.avro.specific.SpecificRecordBase implements org.apache.avro.specific.SpecificRecord {
  0个用法
  private static final long serialVersionUID = -4161892450085431044L;
  9个用法
  public static final org.apache.avro.Schema SCHEMA$ = new org.apache.avro.Schema.Parser().parse( s: "{\"type\":\"record\",\"name\":\"AvroEventLog\",\"namespac
  0个用法
  public static org.apache.avro.Schema getClassSchema() { return SCHEMA$; }

  6个用法
  private static SpecificData MODEL$ = new SpecificData();

  2个用法
  private static final BinaryMessageEncoder<AvroEventLog> ENCODER =
      new BinaryMessageEncoder<>(MODEL$, SCHEMA$);

  2个用法
  private static final BinaryMessageDecoder<AvroEventLog> DECODER =
      new BinaryMessageDecoder<>(MODEL$, SCHEMA$);

  /**
   * Return the BinaryMessageEncoder instance used by this class.
   * @return the message encoder used by this class
   */
  0个用法
  public static BinaryMessageEncoder<AvroEventLog> getEncoder() { return ENCODER; }

  /**
   * Return the BinaryMessageDecoder instance used by this class.
   * @return the message decoder used by this class
   */
  0个用法
  public static BinaryMessageDecoder<AvroEventLog> getDecoder() { return DECODER; }

  /**
   * Create a new BinaryMessageDecoder instance for this class that uses the specified {@link SchemaStore}.
   * @param resolver a {@link SchemaStore} used to find schemas by fingerprint
   * @return a BinaryMessageDecoder instance for this class backed by the given SchemaStore
```

AvroEventLog 16:14 LF UTF-8 2 个空格



4.第一套动态规则体系开发

- 动态规则定义

- 第一套动态规则开发代码实现
  - 适合对用户历史状态属性方面的规则捕获
    - 比如：用户过去做过什么操作
  - BaseRuleService：实现基本规则条件匹配
  - CombinationRuleService：组合规则的条件匹配
  - SequenceRuleService：组合顺序条件匹配
  - **使用Clickhouse加载历史行为数据**
    - 解决ListState存储时间跨度太长的问题.
    - 使用物化视图创建**实时线程**更实用。您可以这样做：

      1. 使用引擎创建一个 Kafka 消费者并作为一条数据流。
      2. 创建一个结构表。
      3. 创建物化视图，该视图会在后台**转换引擎中的数据**并将其放入之前创建的表中。

      当 `MATERIALIZED VIEW` 添加至引擎，它将会在**后台收集数据**。可以**持续不断地**从 Kafka 收集数据并通过 `SELECT` 将数据转换为所需要的格式。

    - 示例：

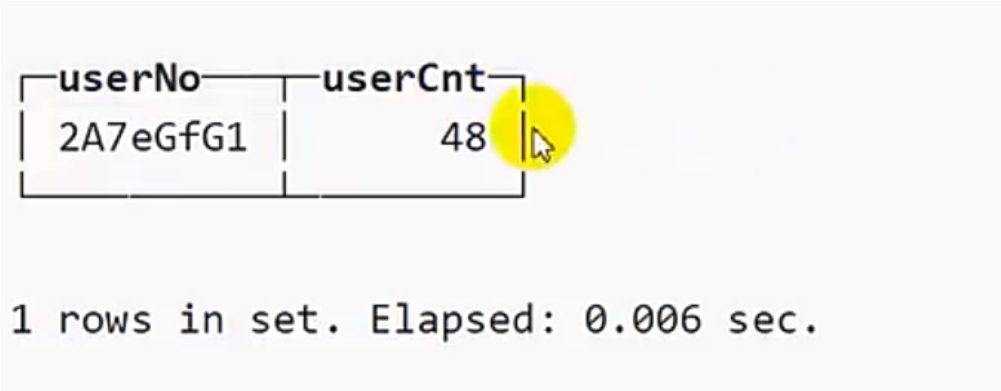
```
CREATE TABLE queue (
  timestamp UInt64,
  level String,
  message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');

CREATE TABLE daily (
  day Date,
  level String,
  total UInt64
) ENGINE = SummingMergeTree(day, (day, level), 8192);

CREATE MATERIALIZED VIEW consumer TO daily
  AS SELECT toDate(toDateTime(timestamp)) AS day, level, count() as total
  FROM queue GROUP BY day, level;

SELECT level, sum(total) FROM daily GROUP BY level;
```

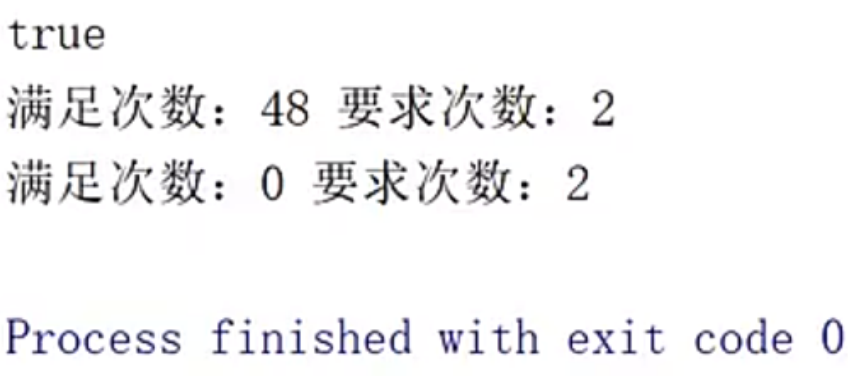
    - 官方文档：https://clickhouse.com/docs/zh/engines/table-engines/integrations/kafka/

```
create table streamRuleDB.clientLogDetail (
userNo String,
userName String,
appId String,
detailTime Int64,
appVersion String,
carrier String,
imei String,
deviceType String,
ip String,
netType String,
osName String,
osVersion String,
sessionId String,
eventId String,
eventType String,
createTime String,
gps String,
addr String,
properties Map(String,String),
INDEX u (userNo) TYPE minmax GRANULARITY 3,
INDEX t (detailTime) TYPE minmax GRANULARITY 3
)
```

```
ENGINE = MergeTree()
ORDER BY (userNo,detailTime)


create table streamRuleDB.clientLogDetail_ingestionKafka (
userNo String,
userName String,
appId String,
appVersion String,
carrier String,
imei String,
deviceType String,
ip String,
netType String,
osName String,
osVersion String,
sessionId String,
detailTime Int64,
eventId String,
eventType String,
createTime String,
gps String,
addr String,
properties Map(String,String)
) ENGINE =
Kafka('localhost:9092','client_log','consumer_group','JSONEachRow')
;


create MATERIALIZED VIEW view_clientLogDetail TO clientLogDetail
as
select
userNo,
userName,
appId,
appVersion,
carrier,
imei,
deviceType,
ip,
netType,
osName,
osVersion,
sessionId,
eventId,
eventType,
createTime,
gps,
addr,
properties
from clientLogDetail_ingestionKafka
```

-



- 

- https://clickhouse.com/docs/zh/sql-reference/aggregate-functions/parametric-functions/

- **sequenceMatch(pattern)(timestamp, cond1, cond2, ...)**

  检查序列是否包含与模式匹配的事件链。

  ```
  sequenceMatch(pattern)(timestamp, cond1, cond2, ...)
  ```

  !!! warning "警告" 在同一秒钟发生的事件可能以未定义的顺序排列在序列中，影响结果。

  参数

  - `pattern` — 模式字符串。 参考 模式语法.
  - `timestamp` — 包含时间的列。典型的时间类型是： `Date` 和 `DateTime`。您还可以使用任何支持的 UInt 数据类型。
  - `cond1`, `cond2` — 事件链的约束条件。 数据类型是： `UInt8`。 最多可以传递32个条件参数。 该函数只考虑这些条件中描述的事件。 如果序列包含未在条件中描述的数据，则函数将跳过这些数据。

  返回值

  - 1，如果模式匹配。
  - 0，如果模式不匹配。

  类型: `UInt8`.

  模式语法

  - `(?N)` — 在位置 `N` 匹配条件参数。 条件在编号 `[1, 32]` 范围。 例如, `(?1)` 匹配传递给 `cond1` 参数。
  - `.*` — 匹配任何事件的数字。 不需要条件参数来匹配这个模式。
  - `(?t operator value)` — 分开两个事件的时间。 例如： `(?1)(?t>1800)(?2)` 匹配彼此发生超过1800秒的事件。 这些事件之间可以存在任意数量的任何事件。 您可以使用 `>=`, `>`, `<`, `<=`, `==` 运算符。

  - **例**

    考虑在数据 `t` 表:

    ```
    ┌─time─┬─number─┐
    │    1 │      1 │
    │    2 │      3 │
    │    3 │      2 │
    └──────┴────────┘
    ```

  - 基于sequenceMatch实现顺序规则匹配

在第一个查询中：

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2) FROM t
```

这里的模式 `'(?1)(?2)'` 表示我们正在寻找一个事件链，其中条件 `cond1`（即 `number = 1`）的事件发生后，紧接着条件 `cond2`（即 `number = 2`）的事件也发生。在这个模式中，我们只关心 `number` 等于 1 和 2 的事件。

给定的数据是这样的：

```
┌─time─┬─number─┐
│    1 │      1 │
│    2 │      3 │
│    3 │      2 │
└──────┴────────┘
```

根据这个模式，我们只考虑 `number` 等于 1 和 2 的事件。因此，`number` 等于 3 的事件被忽略，**因为它没有在条件中被描述。**

所以，当我们查看数据时，我们实际上是在看以下简化的序列：

```
┌─time─┬─number─┐
│    1 │      1 │
│    3 │      2 │
└──────┴────────┘
```

在这个简化的序列中，我们可以看到 `number` 等于 1 的事件在时间 1 发生，然后 `number` 等于 2 的事件在时间 3 发生。它们之间没有其他 `number` 等于 1 或 2 的事件，因此模式匹配，**函数返回 1。**

函数 `sequenceMatch` **只考虑那些在条件参数中指定的事件。**如果一个事件不符合任何条件，它就不会被考虑在内，即使它在时间上位于两个符合条件的事件之间。这就是为什么即使数字 3 的事件在时间 2 发生，它也不会影响模式匹配的结果。

第二个查询：

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2, number = 3) FROM
t
```

在这个查询中，我们**指定了三个条件**：`number = 1`，`number = 2`，和 `number = 3`。这意味着函数将考虑所有这三个数字的事件。

给定的数据是这样的：

```
┌─time─┬─number─┐
│    1 │      1 │
│    2 │      3 │
│    3 │      2 │
└──────┴────────┘
```

按照条件，我们考虑的事件序列是：

```
┌─time─┬─number─┐
│    1 │      1 │  // cond1
│    2 │      3 │  // cond3
│    3 │      2 │  // cond2
└──────┴────────┘
```

在这个序列中：

1. `number = 1` 的事件在时间 1 发生。
2. `number = 3` 的事件在时间 2 发生。
3. `number = 2` 的事件在时间 3 发生。

在这个序列中，`number = 1` 的事件在时间 1 发生，然后 `number = 3` 的事件在时间 2 发生，最后 `number = 2` 的事件在时间 3 发生。由于 `number = 3` 的事件打断了 `number = 1` 和 `number = 2` 之间的连续性，所以模式 `(?1)(?2)` 不匹配，**函数返回 0。**

- **第二个查询**：`number = 3` 的事件打断了 `number = 1` 和 `number = 2` 之间的**连续性**（'(?1)(?2)'），所以模式不匹配。

第三个查询：

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2, number = 4) FROM
t
```

在这个查询中，我们指定了三个条件：`number = 1`，`number = 2`，和 `number = 4`。然而，`number = 4` 的事件在给定的**数据中并没有发生。**

给定的数据是这样的：

```
┌─time─┬─number─┐
│    1 │      1 │
│    2 │      3 │
│    3 │      2 │
└──────┴────────┘
```

按照条件，我们考虑的事件序列实际上是：

```
┌─time─┬─number─┐
│    1 │      1 │  // cond1 匹配
│    2 │      3 │  // 没有匹配的条件
│    3 │      2 │  // cond2 匹配
└──────┴────────┘
```

在这个序列中：

1. `number = 1` 的事件在时间 1 发生。
2. `number = 3` 的事件在时间 2 发生，但它不满足任何条件（`number = 4`），所以被忽略。
3. `number = 2` 的事件在时间 3 发生。

在这种情况下，`number = 3` 的事件确实打断了 `number = 1` 和 `number = 2` 之间的连续性，因为 `sequenceMatch` 函数在寻找模式 `(?1)(?2)` 时，会忽略所有不匹配任何条件的事件。但是，由于 `number = 3` 的事件不满足任何条件（因为没有 `number = 3` 的条件），它实际上被函数忽略了。

因此，函数实际上只看到了以下序列：

```
┌─time─┬─number─┐
│    1 │      1 │  // cond1 匹配
│    3 │      2 │  // cond2 匹配
└──────┴────────┘
```

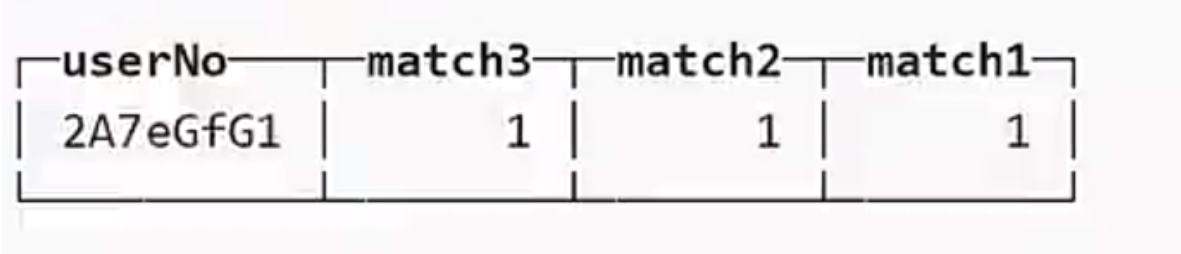
在这个简化的序列中，`number = 1` 的事件在时间 1 发生，然后 `number = 2` 的事件在时间 3 发生，**它们之间没有其他符合条件的事件**，因此模式 `(?1)(?2)` 匹配，**函数返回 1。**

- **第三个查询**：由于 `number = 3` 的事件**不满足任何指定的条件**，它被函数忽略了。因此，它并没有打断 `number = 1` 和 `number = 2` 之间的连续性（'(?1)(?2)'）。

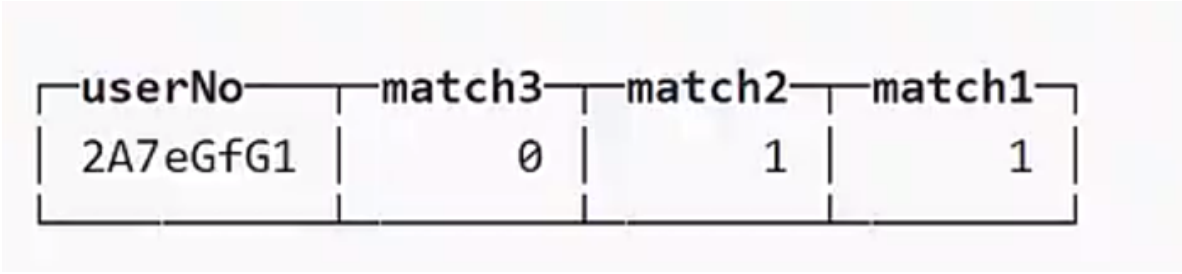
顺序组合匹配:SQL实现，假设给3组规则:

根据特定的事件ID和属性值进行筛选,根据用户编号（`userNo`）进行分组，返回每个用户编号对应的**匹配序列计数**

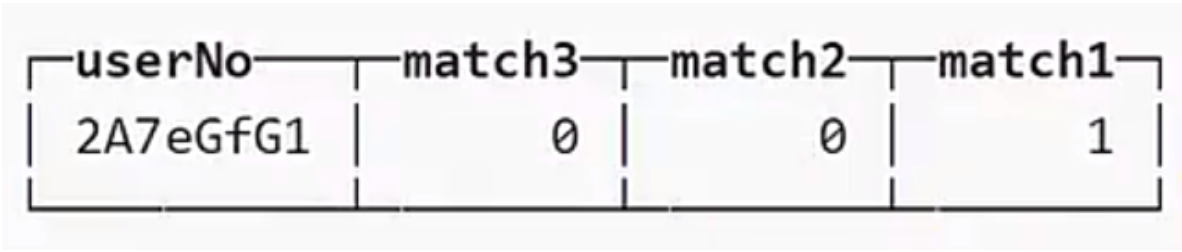
```
select
    userNo,
    sequenceMatch('.*(?1).*(?2).*(?3)')(
        toDateTime(`detailTime`),
        eventId='M' and properties['tagCondition1']='tagValue1',
        eventId='A' and properties['tagCondition1']='tagValue1',
        eventId='C' and properties['tagCondition1']='tagValue1'
    ) as match3, -- 检测一个完整的序列，其中必须包含事件 M、A 和 C，且这些事件的属性 tagCondition1 的值必须为 tagValue1,这个序列可以有任意数量的其他事件在其中
    sequenceMatch('.*(?1).*(?2).*')(
        toDateTime(`detailTime`),
        eventId='M' and properties['tagCondition1']='tagValue1',
        eventId='A' and properties['tagCondition1']='tagValue1',
        eventId='C' and properties['tagCondition1']='tagValue1'
    ) as match2,-- 检测一个序列，其中包含事件 M、A 和 C 中的 M、A，且这两个事件的属性 tagCondition1 的值必须为 tagValue1,这个序列同样可以有任意数量的其他事件在其中
    sequenceMatch('.*(?1).*')(
        toDateTime(`detailTime`),
        eventId='M' and properties['tagCondition1']='tagValue1',
        eventId='A' and properties['tagCondition1']='tagValue1',
        eventId='C' and properties['tagCondition1']='tagValue1'
    ) as match1 -- 检测一个序列，其中包含事件 M、A 和 C 中的 M，且这个事件的属性 tagCondition1 的值必须为 tagValue1,这个序列也可以有任意数量的其他事件在其中
from
    clientLogDetail
where -- 筛选事件ID为M、A或C，且属性tagCondition1的值为tagValue1
    userNo='2A7eGfG1'
    and detailTime >= 0
    and detailTime <= 1728563492
    and (
        (eventId='M' and properties['tagCondition1']='tagValue1')
        or (eventId='A' and properties['tagCondition1']='tagValue1')
        or (eventId='C' and properties['tagCondition1']='tagValue1')
    )
group by userNo;
```



userNo	match3	match2	match1
2A7eGfG1	1	1	1

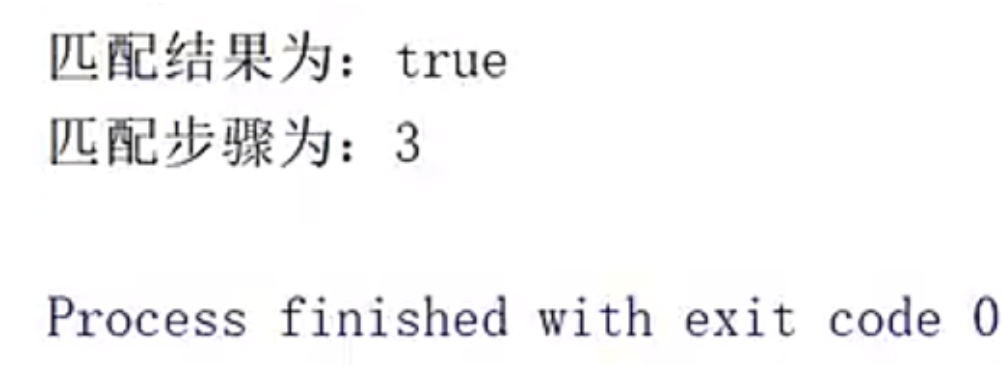


userNo	match3	match2	match1
2A7eGfG1	0	1	1



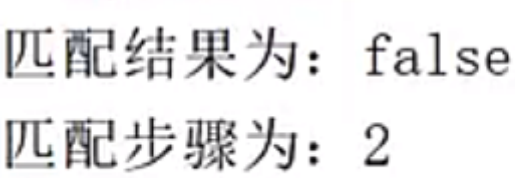
userNo	match3	match2	match1
2A7eGfG1	0	0	1

规则都匹配：



```
匹配结果为：true
匹配步骤为：3

Process finished with exit code 0
```

匹配两个规则：



```
匹配结果为：false
匹配步骤为：2
```

第三版ClickHouseServer查询服务实现

通过Redis缓存解决ClickHouse查询服务的压力

双流join替代方案

```
<!--添加编译proto文件的编译程序和对应的编译插件-->
            <plugin>
                <groupId>org.xolstice.maven.plugins</groupId>
                <artifactId>protobuf-maven-plugin</artifactId>
```

```
                <version>0.5.0</version>
                <configuration>

 <protoSourceRoot>${project.basedir}/src/main/proto</protoSourceRoot>

 <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                    <clearOutputDirectory>false</clearOutputDirectory>

 <protocArtifact>com.google.protobuf:protoc:3.1.0:exe:${os.detected.classifier}
</protocArtifact>
                    <pluginId>grpc-java</pluginId>
                    <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.14.0:exe:${os.detected.classifier}</pluginArtifact>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>compile-custom</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>


 <dependency>
            <groupId>com.googlecode.protobuf-java-format</groupId>
            <artifactId>protobuf-java-format</artifactId>
            <version>1.2</version>
        </dependency>
```

客户端行为日志

```
syntax = "proto3";//指定proto3的语法，默认是proto2
package yushu;//名称空间定义，导入的需要加上名称空间
option java_package = "com.yushu.pojo";
option java_outer_classname = "EventClientLogProto";//生成的Java类名
option java_multiple_files = true;//编译后可以生成多个Meassge类

message EventClientLog {
    //基础字段1-15预留出来
    string request_id = 1;
    //时间戳
    uint64 timestamp = 2;

    //客户端的标识信息
    string imei = 11;
    string ip = 12;
    string network = 13;
    string net_type = 14;
    string os_name = 16;
    string os_version = 17;

    //user info
```

```
    string user_id = 21;

    //活动编号
    string event_id = 50;
    //活动举办者
    string event_user_id = 51;
    //活动开始时间
    uint64 event_start_time = 52;
    //活动举办的城市
    string event_city = 53;
    //活动举办的国家
    string event_country = 54;
    //活动举办的省份
    string event_province = 55;

    //活动的参与情况：报名，不报名，报名后参加，报名后不参加，不报名后参加活动
    string event_type = 60;
}
```

推荐活动信息数据

```
syntax = "proto3";//指定proto3的语法，默认是proto2
package yushu;//名称空间定义，导入的需要加上名称空间
option java_package = "com.yushu.pojo";
option java_outer_classname = "EventServerLogProto";//生成的Java类名
option java_multiple_files = true;//编译后可以生成多个Meassge类

message EventServerLog {
    //基础字段1-15预留出来
    string request_id = 1;
    //时间戳
    uint64 timestamp = 2;

    //客户端的标识信息
    string imei = 21;
    string ip = 22;
    string network = 23;

    //根据当前用户的标签数据匹配对应的活动信息
    string user_id = 31;
    string locale = 32;
    uint64 birthyear = 33;
    string gender = 34;
    uint64 joinedAt = 35;
    string location = 36;
    string country = 37;
    string province = 38;
    string city = 39;
    string timezone = 40;

    //活动信息数据
    //活动编号
    string event_id = 50;
    //活动举办者
    string event_user_id = 51;
    //活动举办时间
```

```
    uint64 event_start_time = 52;
    //活动举办的城市
    string event_city = 53;
    //活动举办的国家
    string event_country = 54;
    //活动举办的省份
    string event_province = 55;

    //活动的参与情况：报名，不报名，报名后参加，报名后不参加，不报名后参加活动
    string event_type = 60;
}
```

客户端行为日志join推荐信息数据结果

```
syntax = "proto3";//指定proto3的语法，默认是proto2
package yushu;//名称空间定义，导入的需要加上名称空间
option java_package = "com.yushu.pojo";
option java_outer_classname = "EventJoinLogProto";//生成的Java类名
option java_multiple_files = true;//编译后可以生成多个Meassge类

//程序调试的时候，需要的一些监控信息
message ProcessInfo {
    //用户是否关联活动
    bool join_server_log = 1;
    //重试次数
    int32 retry_count = 2;
    //处理时间
    int64 process_timestamp = 3;
}

//拼接完成的数据
message EventJoinLog {
    //基础字段1-15预留出来
    string request_id = 1;
    //时间戳
    uint64 timestamp = 2;
    ProcessInfo process_info = 3;

    //客户端的标识信息
    string imei = 11;
    string ip = 12;
    string network = 13;
    string net_type = 14;
    string os_name = 15;
    string os_version = 16;

    //user info
    string user_id = 31;
    string locale = 32;
    uint64 birthyear = 33;
    string gender = 34;
    uint64 joinedAt = 35;
    string location = 36;
    string country = 37;
    string province = 38;
```

```
    string city = 39;
    string timezone = 40;

    //活动编号
    string event_id = 50;
    //活动举办者
    string event_user_id = 51;
    //活动开始时间
    uint64 event_start_time = 52;
    //活动举办的城市
    string event_city = 53;
    //活动举办的国家
    string event_country = 54;
    //活动举办的省份
    string event_province = 55;

    //活动的参与情况：报名，不报名，报名后参加，报名后不参加，不报名后参加活动
    string event_type = 60;
    int64 join = 61;
    int64 not_join = 62;
    int64 invited_join = 63;
    int64 not_invited_join = 64;
    int64 interest = 65;
    int64 not_interest = 66;


}
```

EventClientLogSchema

```
package com.yushu.schema;

import org.apache.flink.api.common.serialization.DeserializationSchema;
import org.apache.flink.api.common.serialization.SerializationSchema;
import org.apache.flink.api.common.typeinfo.TypeHint;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import com.yushu.pojo.*;

import java.io.IOException;

/**
 *  implements DeserializationSchema<EventClientLog>,
SerializationSchema<EventClientLog>
 */
public class EventClientLogSchema implements
DeserializationSchema<EventClientLog>, SerializationSchema<EventClientLog> {

   // 实现反序列化方法
    @Override
    public EventClientLog deserialize(byte[] message) throws IOException {
        return EventClientLog.parseFrom(message);
    }

    // 是否是流的结尾
    @Override
    public boolean isEndOfStream(EventClientLog nextElement) {
```

```
        return false;
    }

    // 实现序列化方法
    @Override
    public byte[] serialize(EventClientLog eventClientLog) {
        return eventClientLog.toByteArray();
    }

    // 返回序列化后的类型
    @Override
    public TypeInformation<EventClientLog> getProducedType() {
        // 具体TypeInformation是做什么的在哔站上有讲解，这里就不做过多解释
        return TypeInformation.of(new TypeHint<EventClientLog>() {});
    }
}
```

EventServerLogSchema

```
package com.yushu.schema;

import org.apache.flink.api.common.serialization.DeserializationSchema;
import org.apache.flink.api.common.serialization.SerializationSchema;
import org.apache.flink.api.common.typeinfo.TypeHint;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import com.yushu.pojo.*;

import java.io.IOException;

/**
 * 自定义序列化
 * */
public class EventServerLogSchema implements
DeserializationSchema<EventServerLog>, SerializationSchema<EventServerLog> {

    /**
     * 反序列化的方法
     * */
    @Override
    public EventServerLog deserialize(byte[] message) throws IOException {
        //返回对象
        return EventServerLog.parseFrom(message);
    }

    //是否是流的结尾
    @Override
    public boolean isEndOfStream(EventServerLog nextElement) {
        return false;
    }

    //序列化的方法
    @Override
    public byte[] serialize(EventServerLog eventServerLog) {
        return eventServerLog.toByteArray();
    }
```

```
    //指定反序列化以后对象的类型
    @Override
    public TypeInformation<EventServerLog> getProducedType() {
        return TypeInformation.of(new TypeHint<EventServerLog>() {});
    }
}
```

EventJoinLogSchema

```
package com.yushu.schema;


import com.yushu.pojo.EventJoinLog;
import org.apache.flink.api.common.serialization.DeserializationSchema;
import org.apache.flink.api.common.serialization.SerializationSchema;
import org.apache.flink.api.common.typeinfo.TypeHint;
import org.apache.flink.api.common.typeinfo.TypeInformation;

import java.io.IOException;

public class EventJoinLogSchema implements DeserializationSchema<EventJoinLog>,
SerializationSchema<EventJoinLog> {
    @Override
    public EventJoinLog deserialize(byte[] message) throws IOException {
        return EventJoinLog.parseFrom(message);
    }

    @Override
    public boolean isEndOfStream(EventJoinLog nextElement) {
        return false;
    }

    @Override
    public byte[] serialize(EventJoinLog element) {
        return element.toByteArray();
    }

    @Override
    public TypeInformation<EventJoinLog> getProducedType() {
        return TypeInformation.of(new TypeHint<EventJoinLog>() {});
    }
}
```

5.第二套动态规则体系开发

5.1 场景管理模块开发

场景名： 场景类型： 搜索 新增

序号	场景名	场景类型	描述	运行jar包名	关联维度	操作
1	交易注册	注册	用于对注册场景进行风控	1.jar	ID、用户名、身份证、手机号	开启 删除
2	XXX	登录	XXX	3.jar	ID、用户名、身份证、手机号	

5.2 场景增加

* 场景名称： 场景名称1

* 场景类型： 注册

* 描述： 用户注册场景的风控，降低恶意注册风险

* 任务运行jar包： 请上传jar包

场景说明：一个场景相当于一个Flink任务，每次增加一个场景，就会启动一个Flink任务。

黑白名单、规则等都与场景关联。

场景关联维度：

字段名	ID	useName	idCard	tel
中文含义	ID	用户名	身份证	手机号

重置 提交

5.3 黑白名单管理

场景名： 维度名： 维度值： 搜索 新增

序号	场景名	维度名	维度值	描述	来源	操作
1	交易注册	身份证	32015518751205xxxx	此身份证存在异常	风控任务	删除
2	交易注册	身份证	32015518751205xxxx		手动	

- 实时动态规则架构定义

5.4 ys_rule_backend后台工程开发

5.5 ys_rule_front前端工程开发

6. 推荐引擎的实现

6.1 Flume数据集成到Kafka

```
数据集整理
-rw-r--r-- 1 root root  121060703 12月 13 17:33 event_attendees.csv
-rw-r--r-- 1 root root 1172819579 12月 13 17:41 events.csv       3137973
-rw-r--r-- 1 root root    3261791 12月 13 17:33 test.csv
-rw-r--r-- 1 root root     946413 12月 13 17:33 train.csv
-rw-r--r-- 1 root root  326802655 12月 13 17:33 user_friends.csv
-rw-r--r-- 1 root root    2796198 12月 13 17:33 users.csv        38210
```

- 数据集成到Kafka

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 users topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
users

#创建 users topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
users --partitions 1 --replication-factor 1

#修改 topic保存时间
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
users --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic users --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic users -time -1 --offsets 1

//创建数据存放目录和Flume的checkpoint目录
mkdir -p /var/data_ingestion_flume/checkpoint/users
mkdir -p /var/data_ingestion_flume/data/users

//修改目录权限，后面Flume读取需要权限
chmod 777 -R /var/data_ingestion_flume/

//数据准备工作：指定spoolDir采集数据的目录
mkdir -p  /events/input/instr/users/
chmod 777 -R /events

install -m 777 users.csv /events/input/instr/users/users_2024-12-13.csv


users.sources = usersSource
users.channels = usersChannel
users.sinks = usersSink

# Use a channel which buffers events in a directory
users.channels.usersChannel.type = file
users.channels.usersChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/users
users.channels.usersChannel.dataDirs = /var/data_ingestion_flume/data/users
```

```
# Setting the source to spool directory where the file exists
users.sources.usersSource.type = spooldir
users.sources.usersSource.deserializer = LINE
users.sources.usersSource.deserializer.maxLineLength = 6400
users.sources.usersSource.spoolDir = /events/input/instr/users
users.sources.usersSource.includePattern = users_[0-9]{4}-[0-9]{2}-[0-9]{2}.csv
users.sources.usersSource.interceptors = head_filter
users.sources.usersSource.interceptors.head_filter.type = regex_filter
users.sources.usersSource.interceptors.head_filter.regex =
^user_id,locale,birthyear,gender,joinedAt,location,timezone$
users.sources.usersSource.interceptors.head_filter.excludeEvents = true
users.sources.usersSource.channels = usersChannel

# Define / Configure sink
users.sinks.usersSink.type = org.apache.flume.sink.kafka.KafkaSink
users.sinks.usersSink.batchSize = 640
users.sinks.usersSink.brokerList = sandbox-hdp.hortonworks.com:6667
users.sinks.usersSink.topic = users
users.sinks.usersSink.channel = usersChannel


集成user_friends_raw Topic
################################################################################
###########
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 users topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
user_friends_raw

#创建 users topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
user_friends_raw --partitions 1 --replication-factor 1

#修改 topic保存时间
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
user_friends_raw --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic user_friends_raw --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic user_friends_raw -time -1 --offsets 1

//生成数据
install -m 777 user_friends.csv
/events/input/instr/user_friends/userFriends_2024-12-13.csv

user_friends.sources = userFriendsSource
user_friends.channels = userFriendsChannel
user_friends.sinks = userFriendsSink

# Use a channel which buffers events in a directory
user_friends.channels.userFriendsChannel.type = file
user_friends.channels.userFriendsChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/user_friends
```

```
user_friends.channels.userFriendsChannel.dataDirs =
/var/data_ingestion_flume/data/user_friends

# Setting the source to spool directory where the file exists
user_friends.sources.userFriendsSource.type = spooldir
user_friends.sources.userFriendsSource.deserializer = LINE
user_friends.sources.userFriendsSource.deserializer.maxLineLength = 128000
user_friends.sources.userFriendsSource.spoolDir =
/events/input/instr/user_friends
user_friends.sources.userFriendsSource.includePattern = userFriends_[0-9]{4}-[0-
9]{2}-[0-9]{2}.csv
users.sources.usersSource.interceptors = head_filter
users.sources.usersSource.interceptors.head_filter.type = regex_filter
users.sources.usersSource.interceptors.head_filter.regex = ^user,friends$
users.sources.usersSource.interceptors.head_filter.excludeEvents = true
user_friends.sources.userFriendsSource.channels = userFriendsChannel

# Define / Configure sink
user_friends.sinks.userFriendsSink.type = org.apache.flume.sink.kafka.KafkaSink
user_friends.sinks.userFriendsSink.batchSize = 640
user_friends.sinks.userFriendsSink.brokerList = sandbox-hdp.hortonworks.com:6667
user_friends.sinks.userFriendsSink.topic = user_friends_raw
user_friends.sinks.userFriendsSink.channel = userFriendsChannel


集成events Topic
################################################################################
###########
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
events

#创建 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
events --partitions 1 --replication-factor 1

#修改 topic保存时间
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
events --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic events --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic events -time -1 --offsets 1

//生成数据
install -m 777 events.csv /events/input/instr/events/events-12-13.csv


# Initialize agent's source, channel and sink
events.sources = eventsSource
events.channels = eventsChannel
events.sinks = eventsSink1 eventsSink2 eventsSink3
```

```
events.sinkgroups = grpEvents
events.sinkgroups.grpEvents.sinks = eventsSink1 eventsSink2 eventsSink3
events.sinkgroups.grpEvents.processor.type = load_balance
events.sinkgroups.grpEvents.processor.backoff = true
events.sinkgroups.grpEvents.processor.selector = round_robin
/var/data_ingestion_flume/checkpoint/events
# Use a channel which buffers events in a directory
events.channels.eventsChannel.type = file
events.channels.eventsChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/events
events.channels.eventsChannel.dataDirs = /var/data_ingestion_flume/data/events
events.channels.eventsChannel.transactionCapacity = 5000

# Setting the source to spool directory where the file exists
events.sources.eventsSource.type = spooldir
events.sources.eventsSource.deserializer = LINE
events.sources.eventsSource.deserializer.maxLineLength = 32000
events.sources.eventsSource.spoolDir = /events/input/instr/events
events.sources.eventsSource.includePattern = ^events_[0-9]{4}-[0-9]{2}-[0-9]
{2}.csv$
events.sources.eventsSource.channels = eventsChannel

# Define / Configure sinks
events.sinks.eventsSink1.type = org.apache.flume.sink.kafka.KafkaSink
events.sinks.eventsSink1.batchSize = 1280
events.sinks.eventsSink1.brokerList = sandbox-hdp.hortonworks.com:6667
events.sinks.eventsSink1.topic = events
events.sinks.eventsSink1.channel = eventsChannel
events.sinks.eventsSink2.type = org.apache.flume.sink.kafka.KafkaSink
events.sinks.eventsSink2.batchSize = 1280
events.sinks.eventsSink2.brokerList = sandbox-hdp.hortonworks.com:6667
events.sinks.eventsSink2.topic = events
events.sinks.eventsSink2.channel = eventsChannel
events.sinks.eventsSink3.type = org.apache.flume.sink.kafka.KafkaSink
events.sinks.eventsSink3.batchSize = 1280
events.sinks.eventsSink3.brokerList = sandbox-hdp.hortonworks.com:6667
events.sinks.eventsSink3.topic = events
events.sinks.eventsSink3.channel = eventsChannel
```

- event_Attendees数据集成
  - 记录数：37145

```
################################################################################
###########
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
event_attendees_raw

#创建 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
event_attendees_raw --partitions 1 --replication-factor 1

#修改 topic保存时间
```

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
events --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic event_attendees_raw --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic event_attendees_raw -time -1 --offsets 1

//生成数据
install -m 777 event_attendees.csv
/events/input/instr/event_attendees/eventAttendees_2022-03-02.csv

# Initialize agent's source, channel and sink
event_attendees.sources = eventAttendeesSource
event_attendees.channels = eventAttendeesChannel
event_attendees.sinks = eventAttendeesSink

# Use a channel which buffers events in a directory
event_attendees.channels.eventAttendeesChannel.type = file
event_attendees.channels.eventAttendeesChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/event_attendees
event_attendees.channels.eventAttendeesChannel.dataDirs =
/var/data_ingestion_flume/data/event_attendees

# Setting the source to spool directory where the file exists
event_attendees.sources.eventAttendeesSource.type = spooldir
event_attendees.sources.eventAttendeesSource.deserializer = LINE
event_attendees.sources.eventAttendeesSource.deserializer.maxLineLength = 128000
event_attendees.sources.eventAttendeesSource.spoolDir =
/events/input/instr/event_attendees
event_attendees.sources.eventAttendeesSource.includePattern = eventAttendees_[0-
9]{4}-[0-9]{2}-[0-9]{2].csv
event_attendees.sources.eventAttendeesSource.channels = eventAttendeesChannel

# Define / Configure sink
event_attendees.sinks.eventAttendeesSink.type =
org.apache.flume.sink.kafka.KafkaSink
event_attendees.sinks.eventAttendeesSink.batchSize = 640
event_attendees.sinks.eventAttendeesSink.brokerList = sandbox-
hdp.hortonworks.com:6667
event_attendees.sinks.eventAttendeesSink.topic = event_attendees_raw
event_attendees.sinks.eventAttendeesSink.channel = eventAttendeesChannel
```

- test测试数据集导入
    - 记录数：10238

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
test

#创建 events topic
```

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
test --partitions 1 --replication-factor 1

#修改 topic保存时间
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
test --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic test --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic test -time -1 --offsets 1

#生成数据
install -m 777 test.csv /events/input/instr/test/test_2022-03-02.csv


# Initialize agent's source, channel and sink
test.sources = testSource
test.channels = testChannel
test.sinks = testSink

# Use a channel which buffers events in a directory
test.channels.testChannel.type = file
test.channels.testChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/test
test.channels.testChannel.dataDirs = /var/data_ingestion_flume/data/test

# Setting the source to spool directory where the file exists
test.sources.testSource.type = spooldir
test.sources.testSource.deserializer = LINE
test.sources.testSource.deserializer.maxLineLength = 6400
test.sources.testSource.spoolDir = /events/input/instr/test
test.sources.testSource.includePattern = test_[0-9]{4}-[0-9]{2}-[0-9]{2}.csv
test.sources.testSource.channels = testChannel

# Define / Configure sink
test.sinks.testSink.type = org.apache.flume.sink.kafka.KafkaSink
test.sinks.testSink.batchSize = 640
test.sinks.testSink.brokerList = sandbox-hdp.hortonworks.com:6667
test.sinks.testSink.topic = test
test.sinks.testSink.channel = testChannel
```

- train训练数据集导入
    - 记录数：15399

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

#删除 events topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
train

#创建 events topic
```

```
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
train --partitions 1 --replication-factor 1

#修改 topic保存时间
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --alter --topic
train --config retention.ms=864000000

#监控数据流，开启消息监控
kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic train --from-beginning

#验证kafka结果集
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic train -time -1 --offsets 1

#生成数据
install -m 777 train.csv /events/input/instr/test/train_2022-03-02.csv

train.sources = trainSource
train.channels = trainChannel driverChannel
train.sinks = trainSink driverSink

# Use a channel which buffers events in a directory
train.channels.trainChannel.type = file
train.channels.trainChannel.checkpointDir =
/var/data_ingestion_flume/checkpoint/train
train.channels.trainChannel.dataDirs = /var/data_ingestion_flume/data/train

# Setting the channel to memory
train.channels.driverChannel.type = memory
train.channels.driverChannel.capacity = 64000
train.channels.driverChannel.transactioncapacity = 16000

# Setting the source to spool directory where the file exists
train.sources.trainSource.type = spooldir
train.sources.trainSource.deserializer = LINE
train.sources.trainSource.deserializer.maxLineLength = 3200
train.sources.trainSource.spoolDir = /events/input/instr/train
train.sources.trainSource.includePattern = train_[0-9]{4]-[0-9]{2]-[0-9]{2].csv
train.sources.trainSource.channels = trainChannel driverChannel

# Define / Configure sink
train.sinks.trainSink.type = org.apache.flume.sink.kafka.KafkaSink
train.sinks.trainSink.batchSize = 640
train.sinks.trainSink.brokerList = sandbox-hdp.hortonworks.com:6667
train.sinks.trainSink.topic = train
train.sinks.trainSink.channel = trainChannel

# Setting the sink to HDFS
train.sinks.driverSink.type = hdfs
train.sinks.driverSink.hdfs.fileType = DataStream
train.sinks.driverSink.hdfs.filePrefix = train
train.sinks.driverSink.hdfs.fileSuffix = .csv
train.sinks.driverSink.hdfs.path = /user/events/driver/%Y-%m-%d
train.sinks.driverSink.hdfs.useLocalTimeStamp = true
train.sinks.driverSink.hdfs.batchSize = 6400
# Number of events written to file before it rolled (0 = never roll based on
number of events)
```

```
train.sinks.driverSink.hdfs.rollCount = 3200
# File size to trigger roll, in bytes (0: never roll based on file size)
train.sinks.driverSink.hdfs.rollSize = 640000
# Number of seconds to wait before rolling current file (0 = never roll based on
time interval)
train.sinks.driverSink.hdfs.rollInterval = 300
train.sinks.driverSink.channel = driverChannel
```

6.2 数据变换

```
//需求一：将user_friends_raw这个topic中的数据，转为user_id,friend_id的kv数据结构，将转换
后的结果存储到user_friends topic中

//需求二：将event_attendees_raw这个topic中的数据，转换为kv数据结构，将转换后的结果存储到
event_attendees topic中

第一步：
测试准备工作需要的配置文件：
settings.properties
zookeeperUrl=sandbox-hdp.hortonworks.com:2181
brokerUrl=sandbox-hdp.hortonworks.com:6667
stateDir=/tmp
coreSite=/usr/hdp/current/hadoop-client/conf/core-site.xml
hdfsSite=/usr/hdp/current/hadoop-client/conf/hdfs-site.xml
hbaseSite=/usr/hdp/current/hbase-client/conf/hbase-site.xml


第二步：
创建 user_friends Topic

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
user_friends

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
user_friends --partitions 3 --replication-factor 1

kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic user_friends -time -1 --offsets 1

kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic user_friends --from-beginning

java -jar event.jar com.yushu.ingestion.kafka.streaming.UserFriendsStreamer
./settings.properties

//重置偏移量
kafka-streams-application-reset.sh --zookeeper sandbox-hdp.hortonworks.com:2181
--bootstrap-servers sandbox-hdp.hortonworks.com:6667 --application-id user-
friends-streamming --input-topics user_friends_raw

数据对比：
user_friends_raw:0:38203
user_friends:2:10128847
user_friends:1:10128847
user_friends:0:10128848
```

```
第三步：
创建 user_friends Topic

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --delete --topic
event_attendees

kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
event_attendees --partitions 3 --replication-factor 1

kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic event_attendees -time -1 --offsets 1

kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --
topic event_attendees --from-beginning

java -jar xxxx.jar com.yushu.ingestion.kafka.streaming.EventAttendeesStreamer
./settings.properties

//重置偏移量
kafka-streams-application-reset.sh --zookeeper sandbox-hdp.hortonworks.com:2181
--bootstrap-servers sandbox-hdp.hortonworks.com:6667 --application-id event-
attendees-streamming --input-topics event_attendees_raw

event_attendees_raw:0:24145
event_attendees:2:3752880
event_attendees:1:3752881
event_attendees:0:3752881
```

6.3 Kafka数据集成到HBase

```
1. 创建名称空间
create_namespace 'ys_events_db'
create 'ys_events_db:users','profile','region','registration'


2. 集成users数据到HBase的ys_events_db:users表
检查users topic是否有数据
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list sandbox-
hdp.hortonworks.com:6667 --topic users -time -1 --offsets 1

java -jar xxxx.jar com.yushu.ingestion.kafka.UserConsumer ./settings.properties

//重置偏移量
kafka-streams-application-reset.sh --zookeeper sandbox-hdp.hortonworks.com:2181
--bootstrap-servers sandbox-hdp.hortonworks.com:6667 --application-id grpUsers -
-input-topics users

//验证结果集
scan 'ys_events_db:users',{LIMIT =>3}
```