风控引擎架构的设计思路

全局思维的能力

抽象思维的能力

需求是否合理，是否能解决问题

能划分多少个子系统

每个子系统能划分多少个模块

这个系统需要可靠性吗，需要扩展能力吗？成本需要控制吗？

表如何设计？API如何设计？模块之间如何通信？

本节课大纲

风控引擎架构设计核心点

架构会围绕核心点进行设计

课程风控引擎设计的核心点

1. 高效率的规则（策略）迭代 --> 风险规则可以动态，自由组合的调整

风险规则设计思路

风险规则可以由多个基础规则（因子）组成

风险规则就是与（AND）或（OR）非（NOT）组合的逻辑运算

不同业务场景的风险规则也不同


m : n
m : n
业务场景
风险规则
基础规则（因子）

优惠券场景 – 风险规则 1：检测时间差 > 3h 且 用户活跃度 > 5

注册场景 – 风险规则1：手机号段非170 或 检测时间差 > 1h

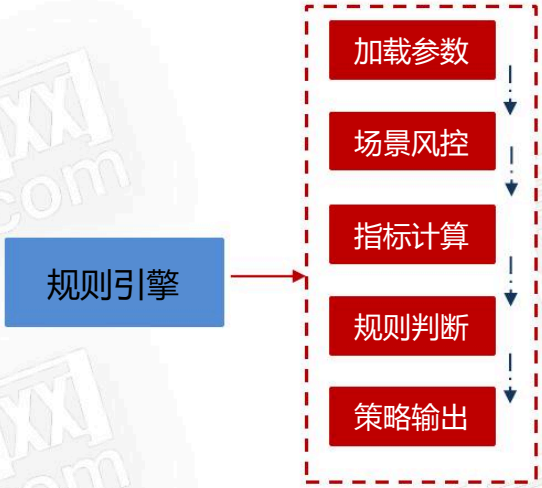
基础规则（因子）

输入的
上下文参数


场景id +
上下文参数
统一接入SDK
场景1
场景2
场景n
规则1
规则2
规则3
规则n
根据命中规则
输出决策

优惠券场景 -- 风控规则 1： 检测时间差 > 3h 且 用户活跃度 > 5 * 活跃系数

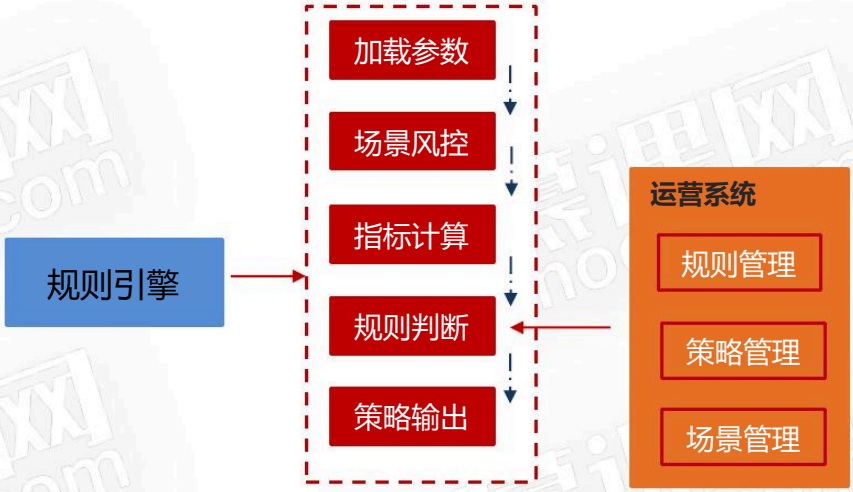
注册场景 – 风控规则1：手机号段非170 或 检测时间差 > 1h


规则引擎
加载参数
场景风控
指标计算
规则判断
策略输出

课程风控引擎设计的核心点

1. 高效率的规则（策略）迭代 --> 风险规则可以动态，自由组合的调整

2. 充分的运营支撑 --> 监控大屏 + 完善的运营后台


加载参数
场景风控
指标计算
规则判断
策略输出
规则引擎
运营系统
规则管理
策略管理
场景管理

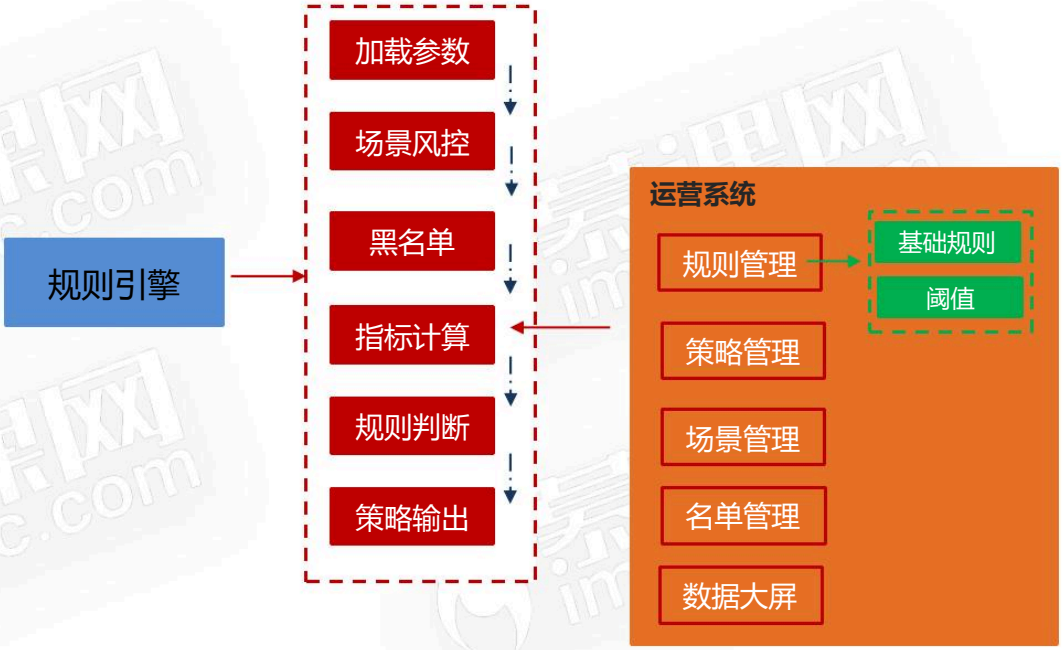

加载参数
场景风控
指标计算
规则判断
策略输出
规则引擎
运营系统
规则管理
基础规则
阈值
策略管理
场景管理


规则引擎
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理

课程风控引擎设计的核心点

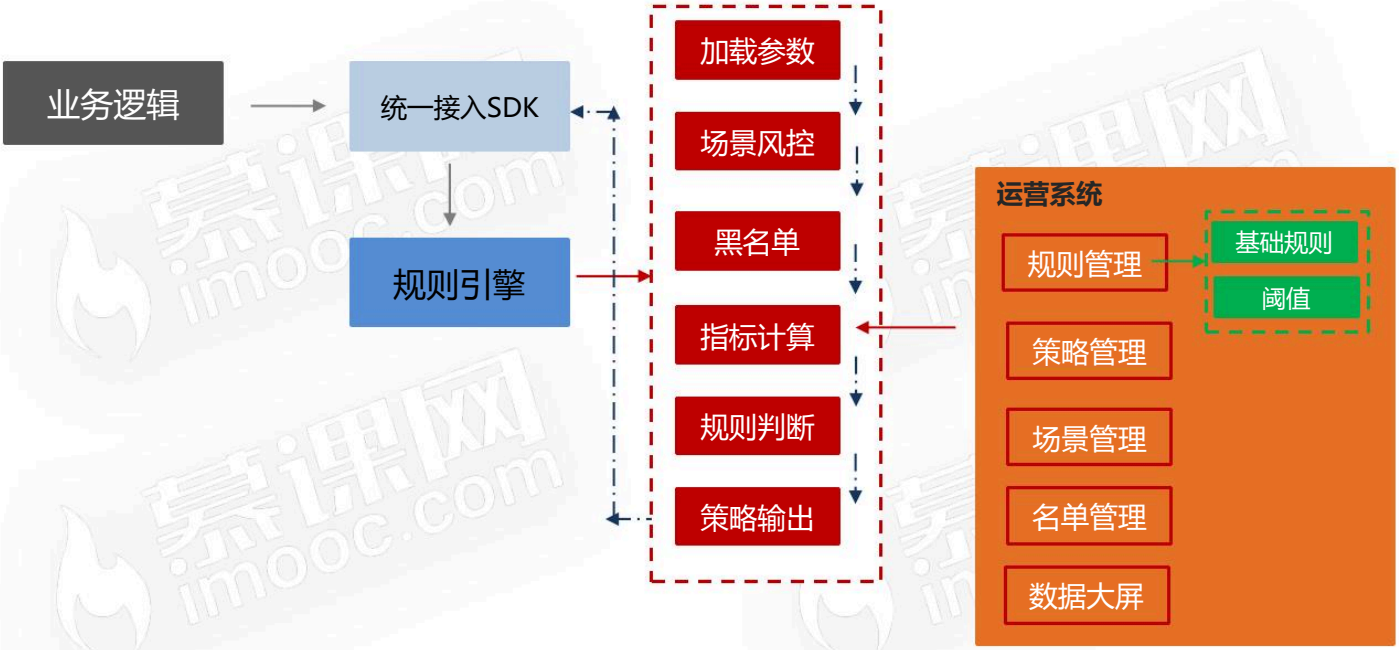
1. 高效率的规则（策略）迭代 --> 风险规则可以动态，自由组合的调整

2. 充分的运营支撑 --> 监控大屏 + 完善的运营后台


规则引擎
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

课程风控引擎设计的核心点

1. 高效率的规则（策略）迭代 --> 风险规则可以动态，自由组合的调整
2. 充分的运营支撑 --> 风控大屏 + 完善的运营后台
3. 无缝的对接不同业务 --> 统一的接入SDK


业务逻辑
统一接入SDK
规则引擎
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

为什么需要事件接入中心？

将所有的事件数据进行统一的管理

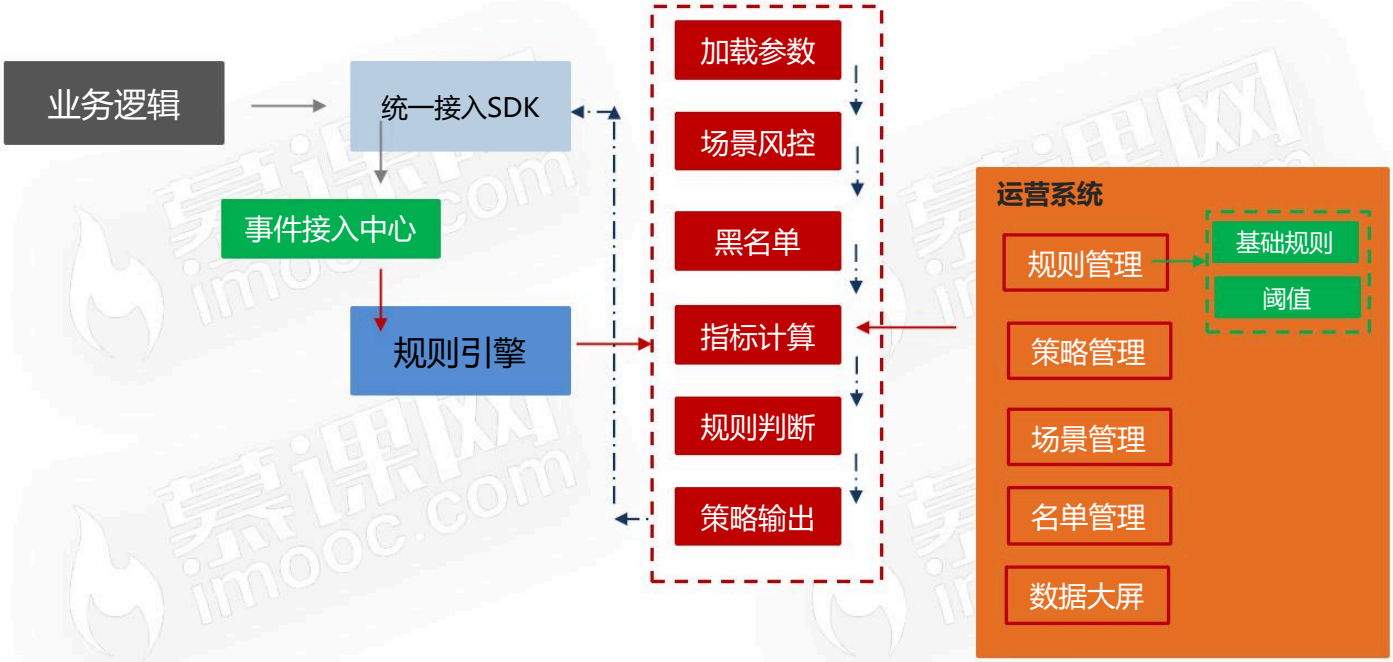
从任意的数据源以流式传输大量的事件数据

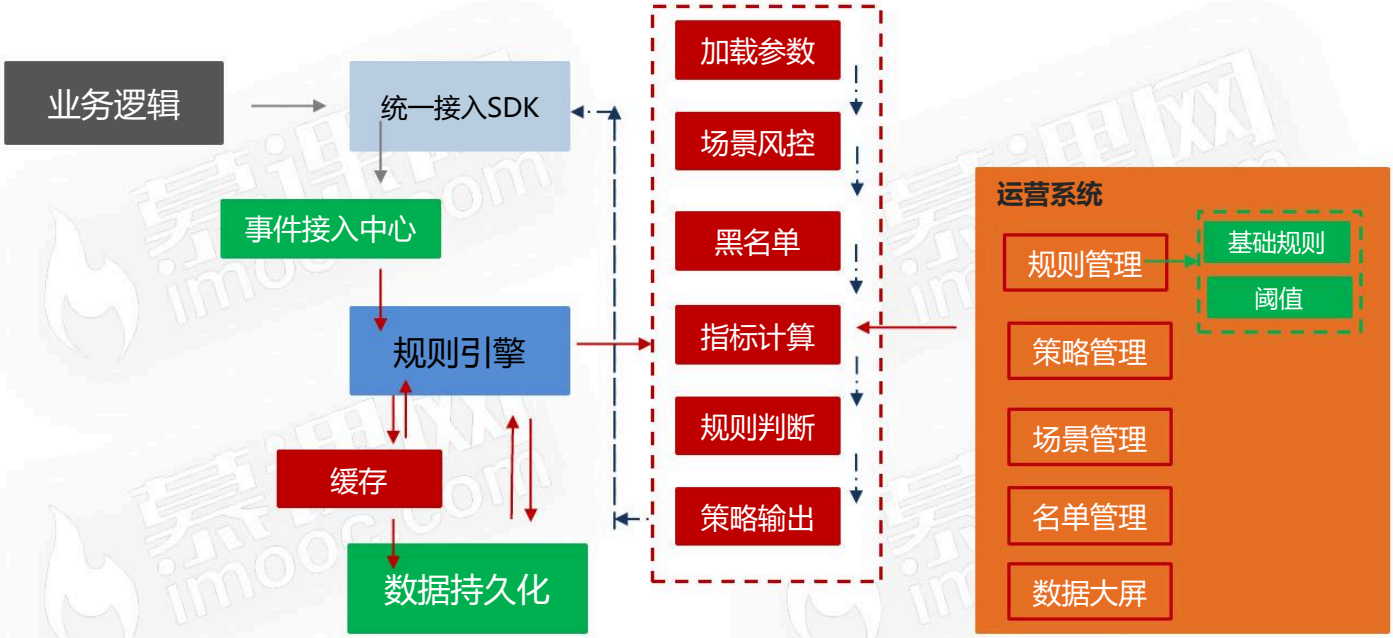
事件接入中心：

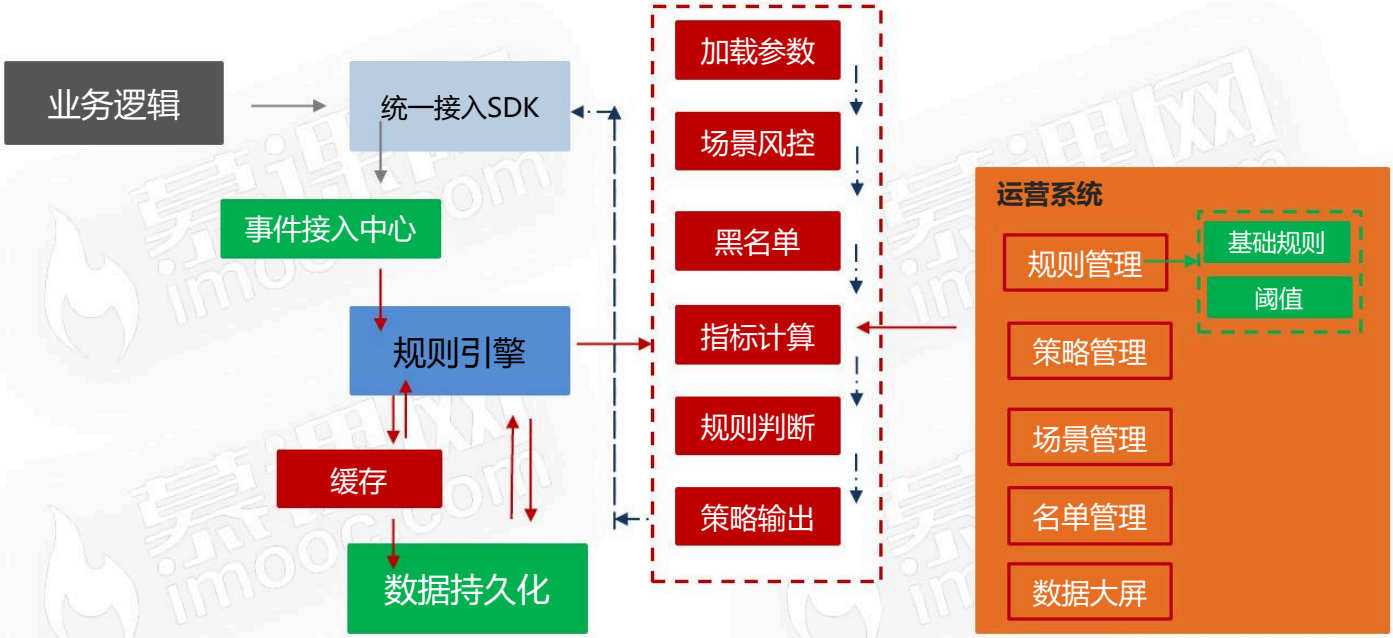
不同的业务场景，包含不同的**事件类型（evenType）**

事件接入中心是整个风控引擎的**数据流入口**

事件接入中心


业务逻辑
统一接入SDK
事件接入中心
规则引擎
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

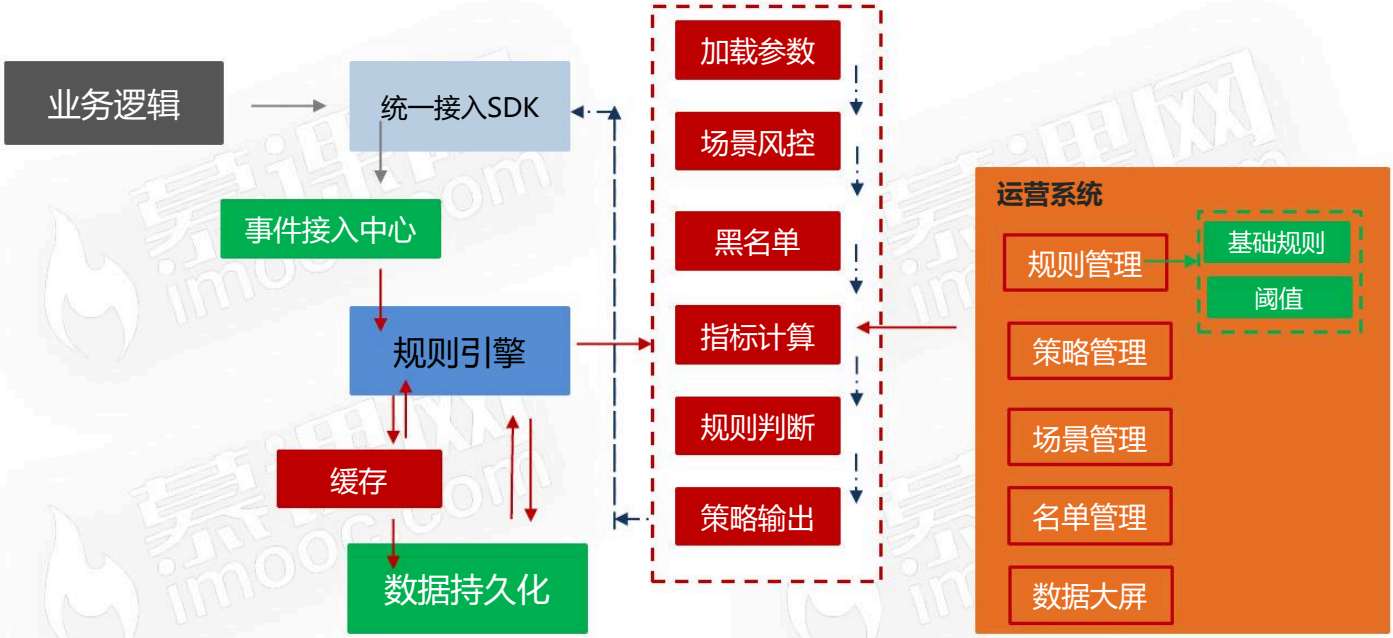

业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏


业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

课程风控引擎设计的核心点

1. 高效率的规则（策略）迭代 --> 风险规则可以动态，自由组合的调整
2. 充分的运营支撑 --> 风控大屏 + 完善的运营后台
3. 无缝的对接不同业务 --> 统一的接入SDK
4. 风控服务稳定可靠 --> 服务高可用 + 熔断降级

画出风控引擎的系统架构图


业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

本节课大纲

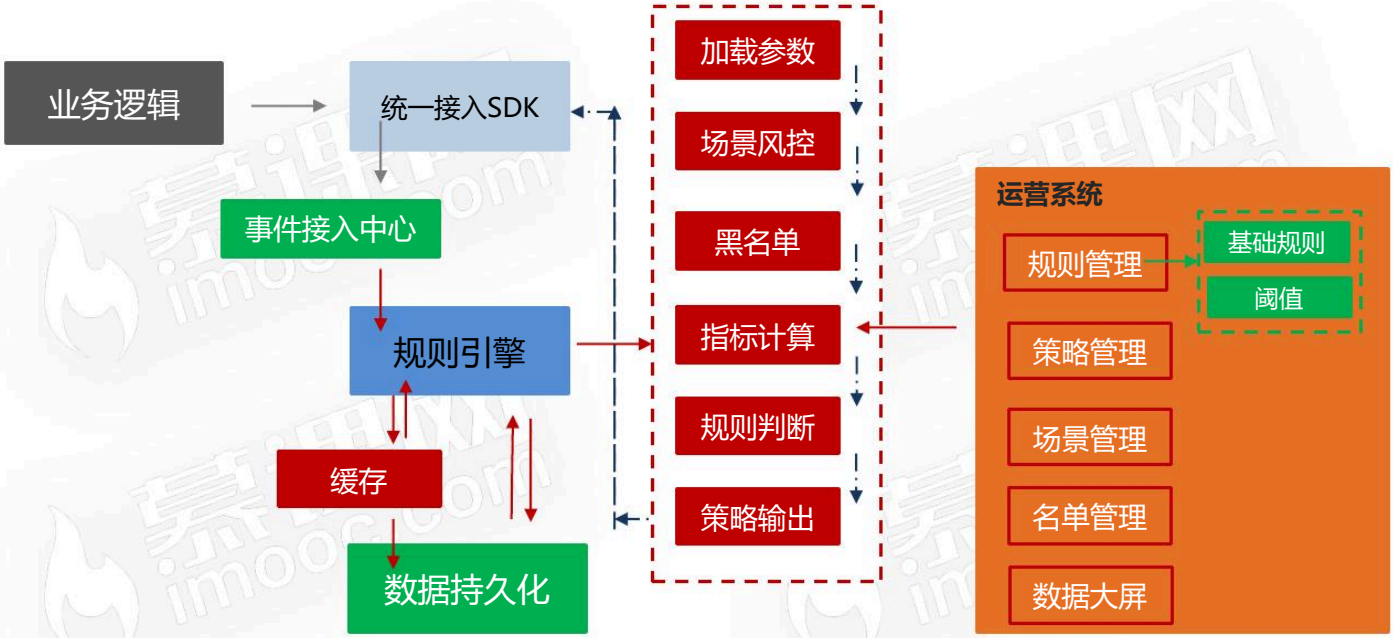
画出应用架构图

画出技术架构图

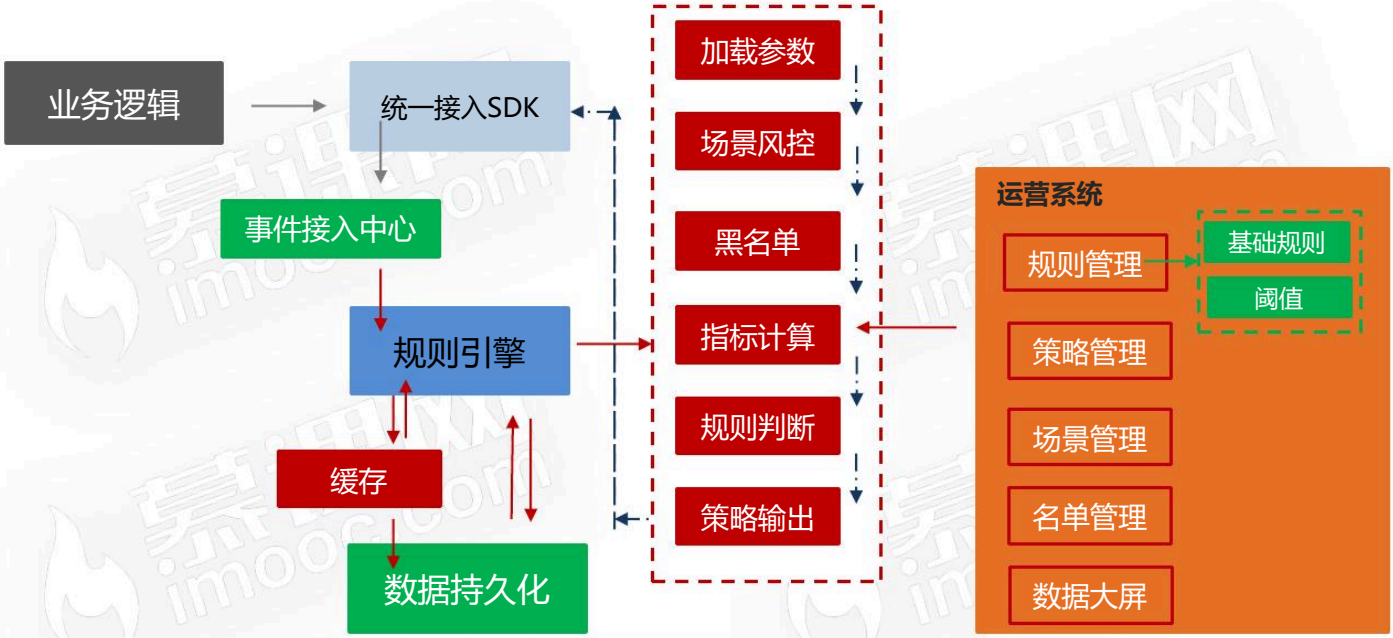
说一大段话，不如画一张图让人更加容易理解

- 业务架构图
- 应用架构图
- 数据架构图
- 技术架构图

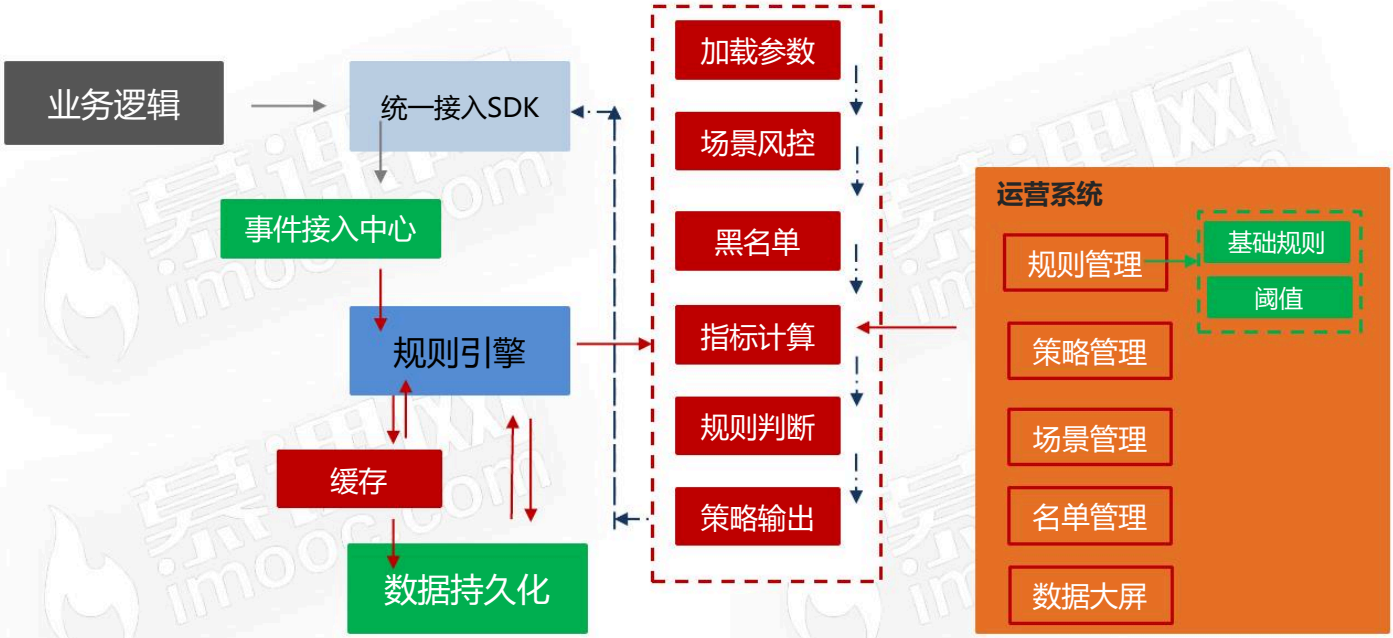
业务架构图：不需要考虑用什么技术，什么组件，着重对业务边界的划分


业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

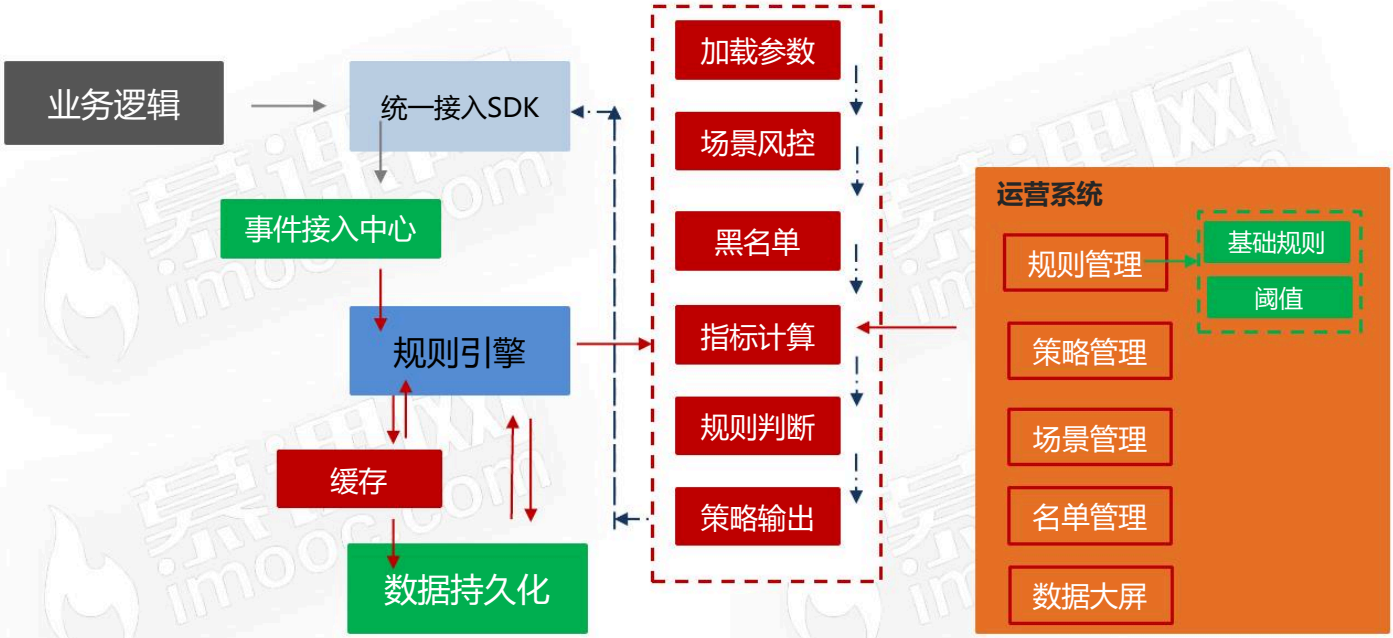
应用架构图：需要**划分出系统的层级，各个层级的应用服务**


业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

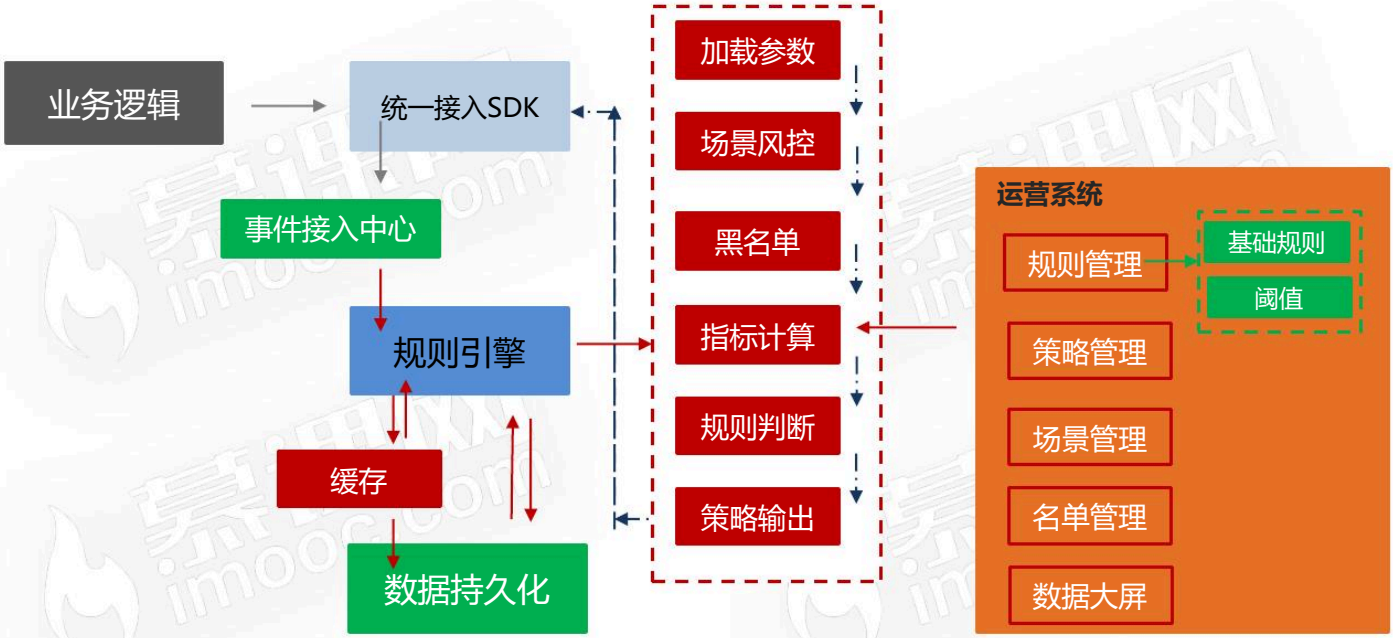

接入层
统一接入 SDK
事件接入中心


业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏

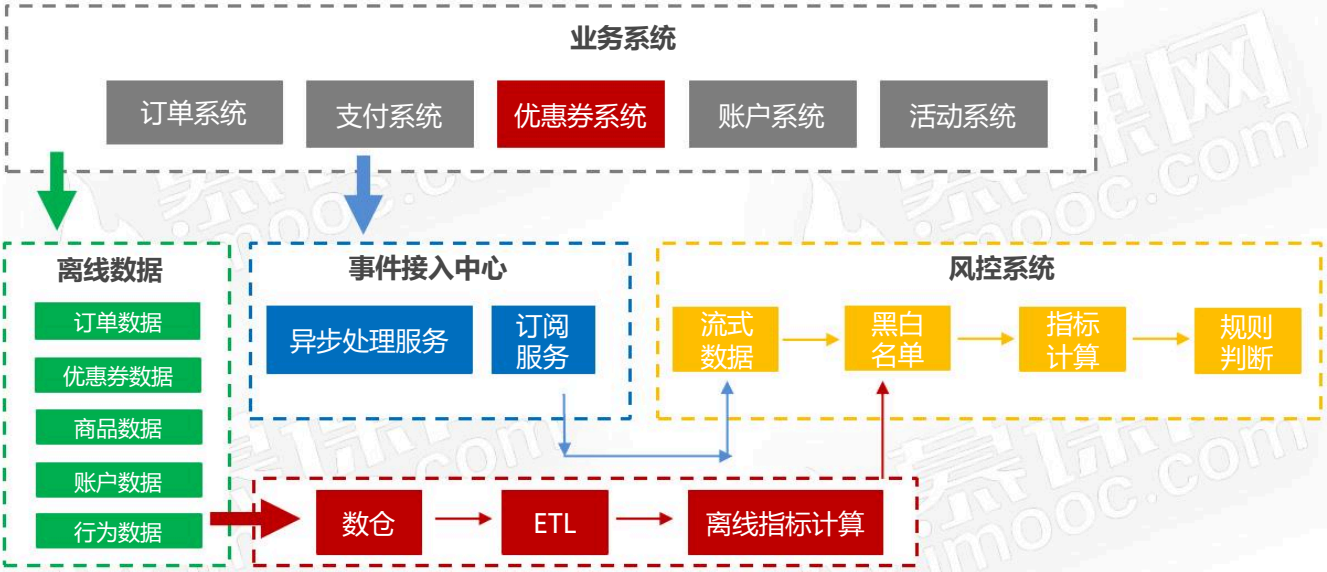

接入层
统一接入 SDK
事件接入中心
服务层
数据大屏
事件查询
数据录入
配置中心

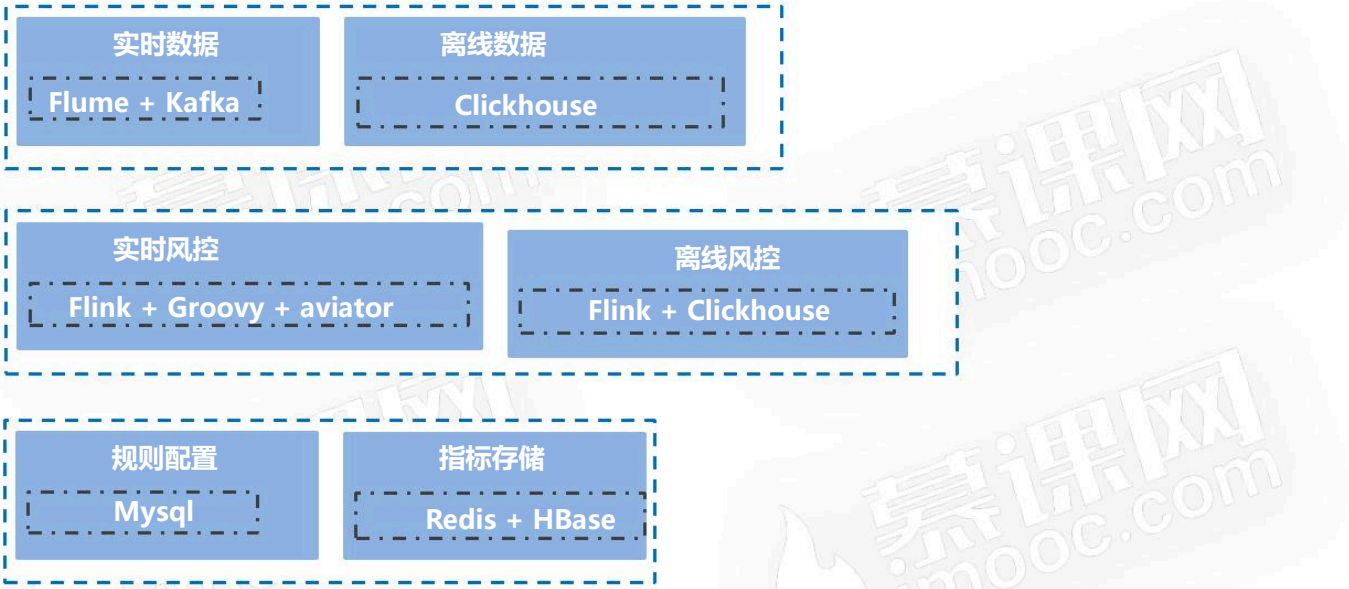

业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏


接入层
统一接入 SDK
事件接入中心
服务层
数据大屏
事件查询
数据录入
配置中心
引擎层
规则判断
查询引擎
计算层
（ 指标 ）计算引擎
（ Flink SQL ）算子引擎

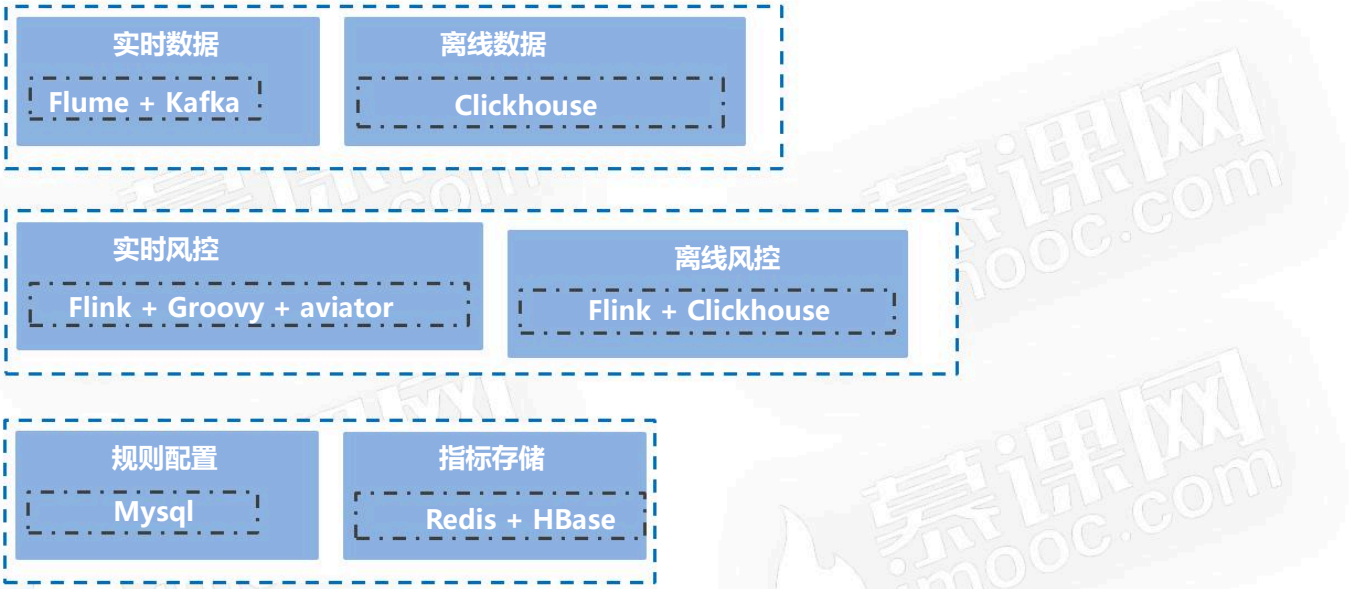

业务逻辑
统一接入SDK
事件接入中心
规则引擎
缓存
数据持久化
加载参数
场景风控
黑名单
指标计算
规则判断
策略输出
运营系统
规则管理
基础规则
阈值
策略管理
场景管理
名单管理
数据大屏


接入层
统一接入 SDK
事件接入中心
服务层
数据大屏
事件查询
数据录入
配置中心
引擎层
规则判断
查询引擎
计算层
（ 指标 ）计算引擎
（ Flink SQL ）算子引擎
存储层
指标存储
行为数据存储


业务系统
订单系统
支付系统
优惠券系统
账户系统
活动系统
离线数据
订单数据
优惠券数据
商品数据
账户数据
行为数据
事件接入中心
异步处理服务
订阅服务
风控系统
流式数据
黑白名单
指标计算
规则判断
数仓
ETL
离线指标计算


实时数据
Flume + Kafka
离线数据
Clickhouse
实时风控
Flink + Groovy + aviator
离线风控
Flink + Clickhouse
规则配置
Mysql
指标存储
Redis + HBase


接入层
统一接入 SDK
事件接入中心
服务层
数据大屏
事件查询
数据录入
配置中心
引擎层
规则判断
查询引擎
表达式引擎
计算层
（ 指标 ）计算引擎
（ Flink SQL ）算子引擎
存储层
指标存储
行为数据存储


实时数据
Flume + Kafka
离线数据
Clickhouse
实时风控
Flink + Groovy + aviator
离线风控
Flink + Clickhouse
规则配置
Mysql
指标存储
Redis + HBase

风控规则引擎选用 Groovy 的原因

本节课大纲

为什么需要规则引擎　　抛弃 if-else

为什么选用 groovy　　对比主流的规则引擎框架

为什么需要 规则引擎

如果：

规则1：　最近 1 个月订单金额 <= 20

AND

规则2：　最近 1 个月订单取消次数 > 30

OR

规则3：　登录 IP 不一致次数 >= 10

那么：

选项　将此用户列入可疑名单

```
If ( 规则1 && 规则2 || 规则3)
        // 列入可疑名单
else
        // TO DO
```

```
If ( 规则1 && 规则2 || 规则3)
    // 列入可疑名单
If ( 规则4 || 规则5 )
    // 列入可疑名单
else
    // TO DO
```

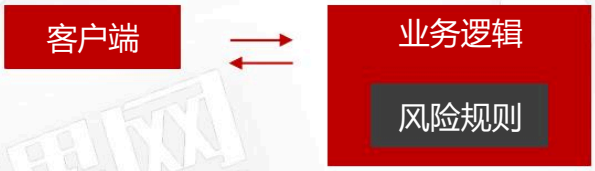
```
If ( (规则1 || 规则4 ) && ( 规则2 || ( 规则3 && 规则5) ) )
      // 列入可疑名单
else
     // TO DO
```

抛弃 if-else，拥抱规则引擎

风控规则本质上是通过 AND，OR 等逻辑运算组成的规则集

风控规则往往是一个复杂且不断变化的规则组合

引入规则引擎之前



引入规则引擎之后

风险规则编写痛点

风险规则变动频率高，组合复杂，数量多

If-else 越写越长

因为是硬编码，即使很小的改动，也要经常重启系统

规则引擎如何解决痛点

不再是 If-else 流程分支，而是交给规则引擎执行

规则文件可以预加载

规则变更，直接修改规则文件，动态加载，无需重启系统

规则的变更，可以交给运营直接修改

规则引擎更多的业务场景

- 手机运营商资费套餐
- 商城积分计算规则
- 用户画像智能生成标签

为什么选用 Groovy

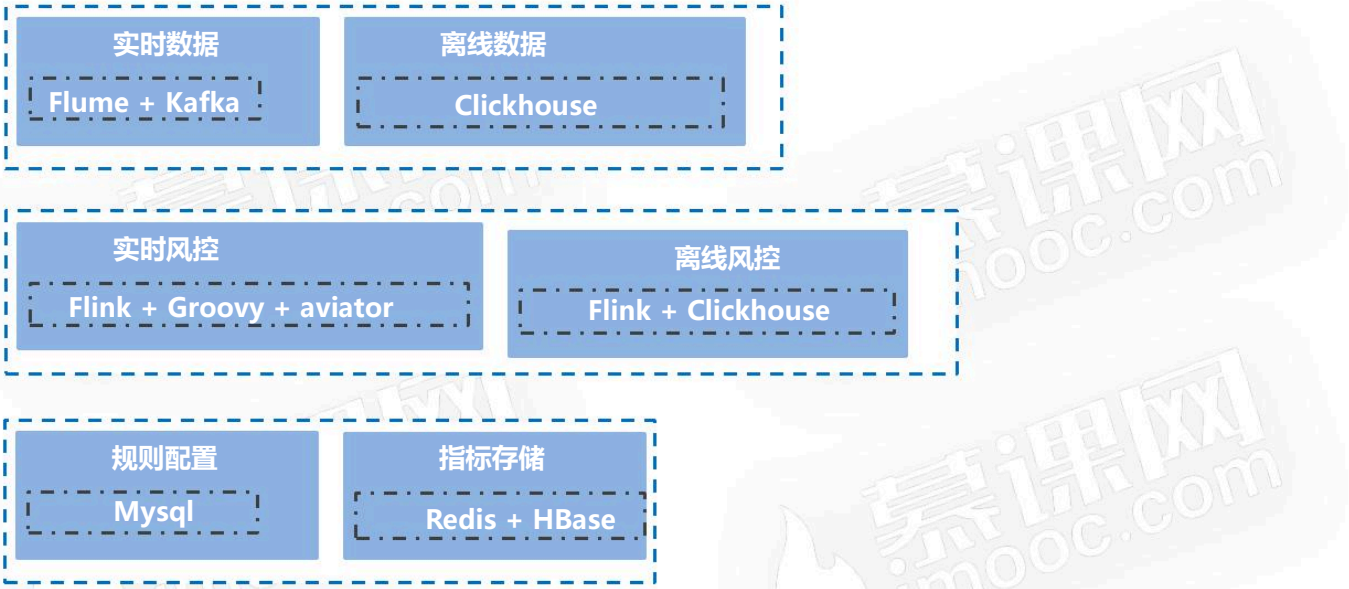
主流开源规则引擎

- Groovy
- Drools

Java 开发为什么选用 Groovy

- Groovy 和 Java 无缝兼容
- Groovy 是动态的
- Groovy 是一把瑞士军刀

风控引擎整体技术栈以及版本


实时数据
Flume + Kafka
离线数据
Clickhouse
实时风控
Flink + Groovy + aviator
离线风控
Flink + Clickhouse
规则配置
Mysql
指标存储
Redis + HBase