KFC推荐系统

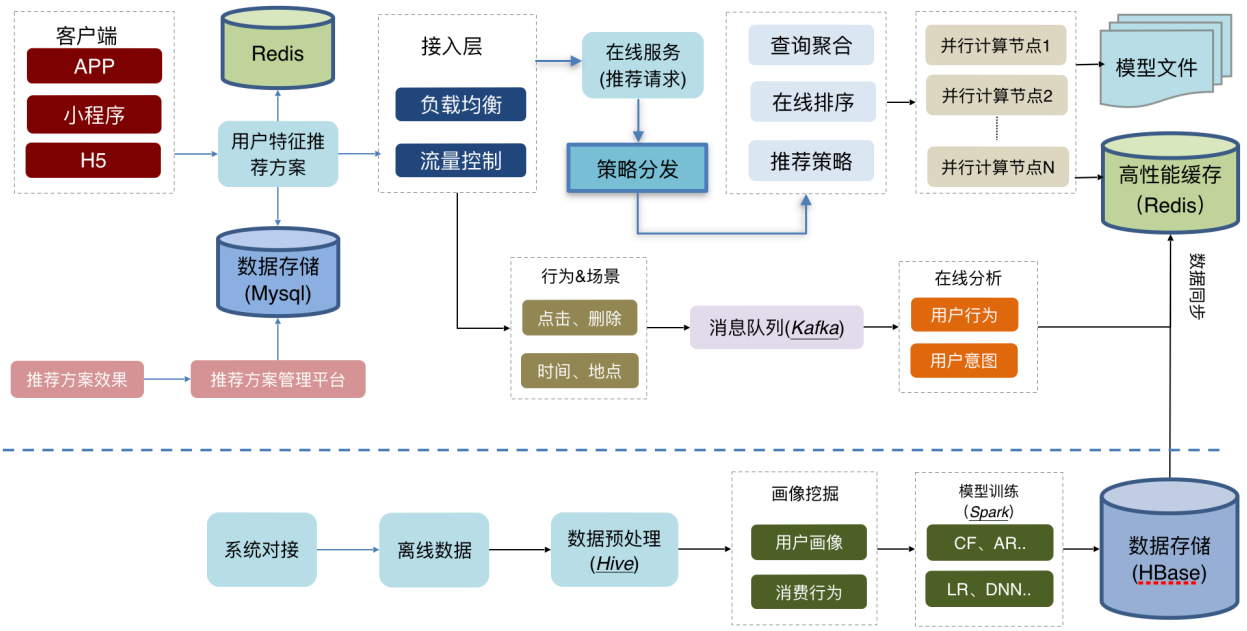
在线模块方案设计

编写人：钱辉

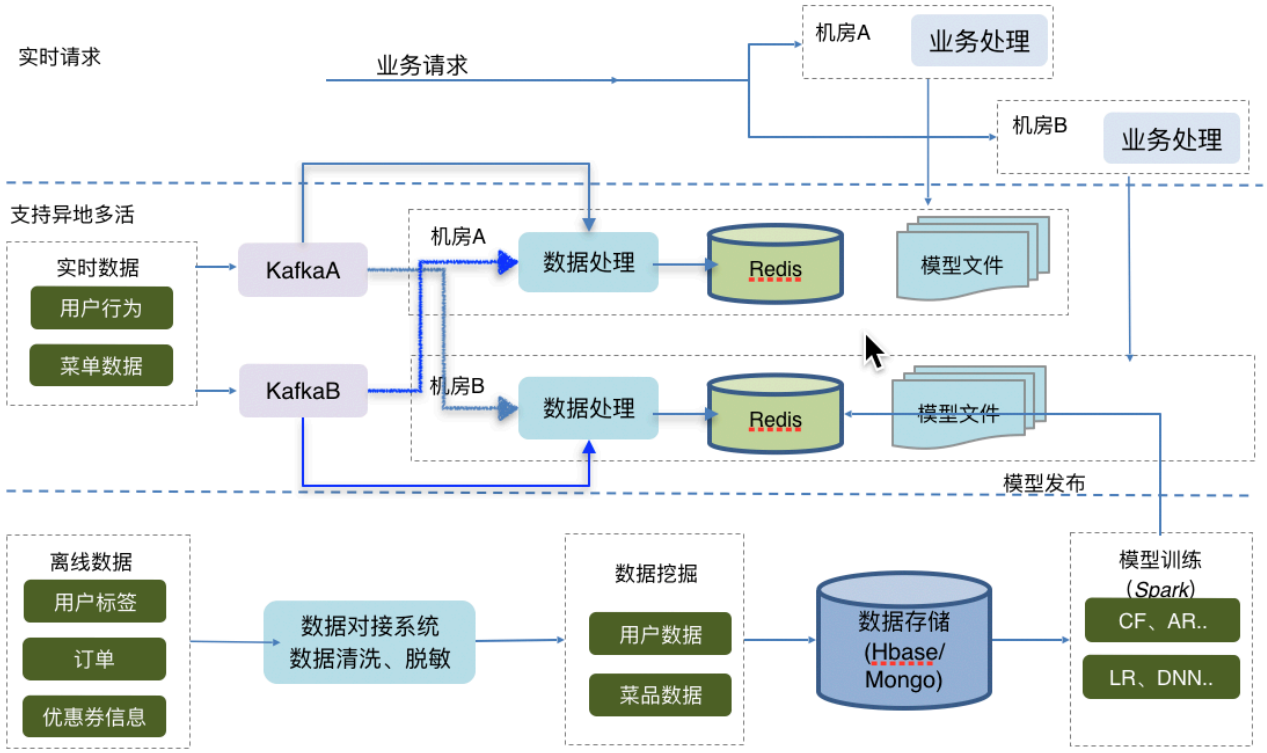
日　期：2018年11月29日

目录

一、架构

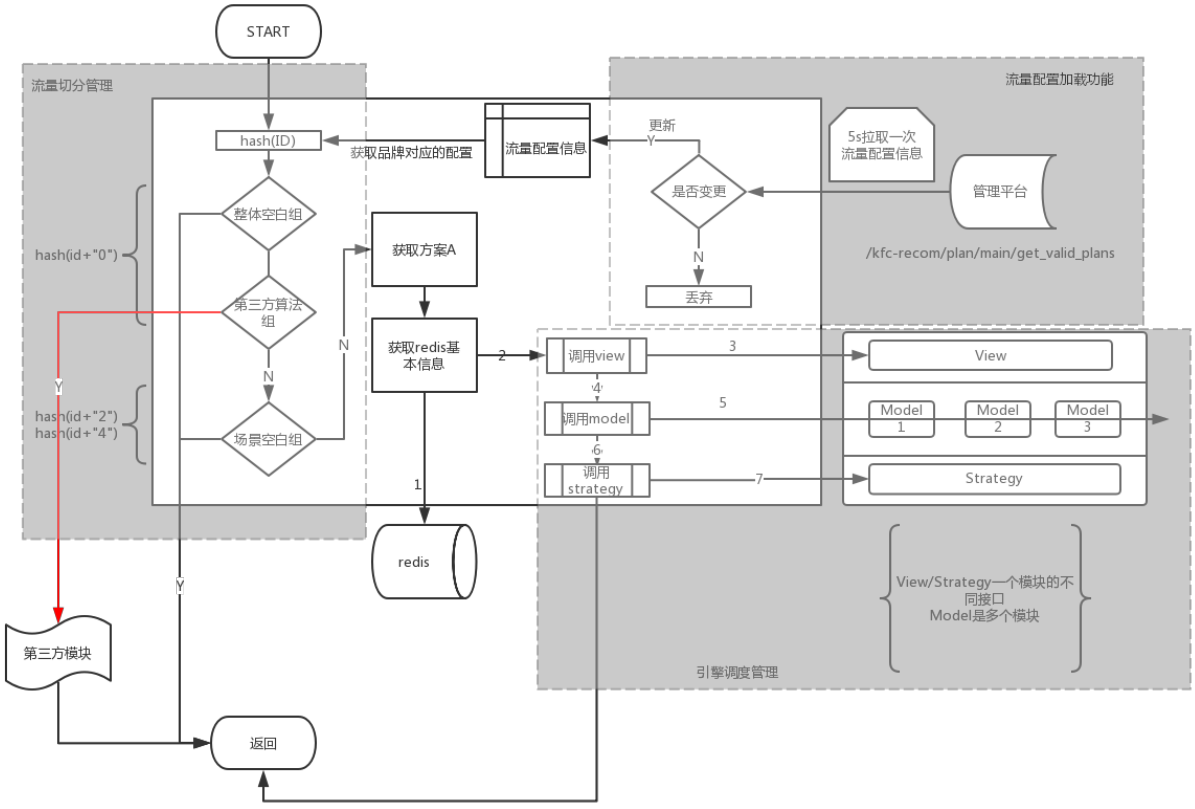


1-1 整体架构



1-2 异地多活架构

二、业务流程图



三、数据库存储格式

Redis

用户信息

- 主键：kfc:person:
- 子健：推荐接口参数userCode
- 结构：k/v
- 操作：策略SET，架构GET

城市信息

- 主键：kfc:city:
- 子健：推荐接口参数cityCode
- 结构：k/v
- 操作：策略SET，架构GET

商户信息

- 主键：kfc::
- 子健：推荐接口参数storeCode，店铺代码
- 结构：k/v
- 操作：策略SET，架构GET

新品信息

- 主键：kfc:new_product:
- 子健：推荐接口参数storeCode，店铺代码
- 结构：k/v
- 操作：策略SET，架构GET

补充信息

- 主键：kfc:info
- 子健：无
- 结构：k/v
- 操作：策略SET，架构GET

tradeup信息

- 主键：kfc:tradeup:info
- 子健：无
- 结构：k/v
- 操作：策略SET，架构GET

linkid和subclass隐射关系

- 主键：kfc:shop:menu:
- 子健：推荐接口参数storeCode，店铺代码
- 结构：hash
- 操作：策略hmset，架构hmget

```
link01:"[\"subclass1\",\"subclass2\"]",
link02:[\"subclass2\",\"subclass3\"]"
```

subclass和单品信息

- 主键：kfc:shop:menu:tag:
- 子健：推荐接口参数storeCode，店铺代码
- 结构：hash
- 操作：策略hmset，架构hmget

```
subclass1:"{}"
subclass2:"{}"
```

subclass和套餐子项

- 主键：kfc:shop:menu:unit:tag:
- 子健：推荐接口参数storeCode，店铺代码
- 结构：hash

- 操作：策略hmset，架构hmget

```
subclass1:"{}"
subclass2:"{}"
```

四 、异地多活

数据

1.实时推荐请求走接口，接口中的数据要带用户唯一标识(用户id或者cookie)，trade up页推荐要带已选择菜品数据。
2.策略计算过程中如果使用一些用户行为数据，这个要要写入到kafka中，数据消费模块同时消费两个机房的kafka来写入到本机房，来保证异地多活，因为有时效性，入redis后设置ttl，自动过期。如果没有实时数据写入的需求，不需要kafka和数据消费模块，双主互备就可以保证双机房。
3.策略通过训练得到的数据为离线数据，自己写入到两边机房的redis中。维度可能为日、小时级别。

方案说明

上图中描述的异地多活方案是采用了同时消费不同机房kafka的方案，消费本地kafka的同时也要消费异地的kafka，两个机房都保存全量数据。满足频繁切换网络的的需求，无须切库操作。

当对数据没那么严格的情况下可以考虑该方案。

其他方案

数据库主从互备的方案：机房A的数据库主库是机房B主库的备份机器。机房B的数据库主库是机房A主库的备份机器，当一个机房挂点，需要手动切换另外一个机房的从库为该机房的主库，这样数据就写到另外一个机房了。
优点：数据不冗余，最终结果一致性。
缺点：无法满足正确情况下用户切换网路造成的访问不同机房数据情形。

对数据要求必须一致性的情况下，可以采用该方案

五、分流管理

方案说明

线上需要按百分比切分用户，每个用户集使用不同的方案，对比不同的策略方案的效果，挑选最优解，所以需要有分流的功能。

1.方案配置功能：策略更新方案配置信息提供给平台，平台提供写入接口，策略编写脚本通过平台提供的接口写入到平台的db。
2.流量配置：当流量配置有更新时，写入平台自己的库。
3.平台开接口，供架构调用，接口返回的信息，是当前场景的配置方案。返回的结果一定是可用方案，具体判断逻辑，以及方案升不生效都在平台接口里面判断。
3.架构层开线程，每隔10s或（1s）从平台拉取一次配置，更新内存中的方案配置。
3.当接入层调用推荐接口时，根据内存中的方案配置，首先根据usercode过滤整体空白组，然后进入真正的方案流量判断，根据transactionId判断用户落在哪个区间，从而判断属于哪个方案，然后调用方案配置中的view，model，strategy，返回推荐结果，方案id写入日志，供DA采集数据分析。
4.策略view、strategy采用python插件化开发，model采用服务化。
5.架构采用go语言。
6.策略层使用python语言，需要保证日志库统一，具有logid传递功能。
7.通信协议采用HTTP。
8.日志和结果中都要加入是否是整体空白组字段，0表示否，1表示是，方便DA判断。

新增逻辑

需要整体空白组和场景空白组两个概念。

0.UserCode为用户手机号MD5值，transactionId为随机数，用户一进入菜单页时重新创建。
1.整体空白组：所有场景都不会有推荐结果。进入任何一个场景的时候，采用的分流策略是一样的。使用的ID为transactionId，维持一定百分比用户没有推荐结果。
2.场景空白组：单一场景的空白组，进入该场景时，使用的ID为transactionId，保持同一次操作过程中，推荐结果是一样的，多次重新登录时推荐结果不一样。
3.每个场景之间采用的判断分流的依据位数不一样。就比如Menu页取的是transactionId的最后一位作为判断标准，那么TradeUp就不能也用最后一位作为标准，这样是为了防止在Menu场景空白组中用户，也一定在TradeUp的场景空白组中。想要的效果是在Menu空白组的用户，到TradeUp页的时候可能会被选到其他推荐方案。
4.每个用户进入场景后，用transactionId判断的话，如果下一次同一个用户的transactionId不同的话，推荐结果肯定也不一样。说明流量分流在场景层是用transactionId来区分，不用UserCode，DA

需要注意统计在非整体空白组的统计中，使用的ID为transactionId。而在整体空白组中使用的是UserCode。
5.整体空白组独立存在，具体方案的分流百分比等于100%。

六、推荐接口

见接口文档

参考文档

1.https://blog.csdn.net/javahongxi/article/details/79500861
2.http://afghl.github.io/2018/02/11/distributed-system-multi-datacenter-1.html
3.https://blog.csdn.net/lsjseu/article/details/9922267
4.https://www.cnblogs.com/cuishuai/p/8194345.html