# 4.4 热门与新文章召回

* 通过对日志数据的处理，来实时增加文章的点击次数等信息
  + 维护一个文章和历史点击次数
* 新文章由头条后台审核通过的文章传入kafka
  + 写入kafka，topic

**Redis后台启动方法 redis-server /etc/redis.conf --daemonize yes**

**进入redis客户端方法是redis-cli，进入频道是select 10**

用flushall可以删除所有的数据

* redis:10
  + # ZRANGE page\_rank 0 -1  
    client.zadd("ch:{}:new".format(channel\_id), {article\_id: time.time()})  
      
    # 热门文章存储  
    # ZINCRBY key increment member 自增，增量放置  
    # ZSCORE  
    # 为有序集 key 的成员 member 的 score 值加上增量 increment 。  
    client.zincrby("ch:{}:hot".format(row['channelId']), 1, row['param']['articleId'])  
      
    # ZREVRANGE key start stop [WITHSCORES] 这里是取出所有数据  
    client.zrevrange(ch:{}:new, 0, -1)

演示代码是online\_update.py

热门文章输入数据：

{"actionTime":"2019-04-10 21:04:39","readTime":"","channelId":18,"param":{"action": "click", "userId": "2", "articleId": "141440", "algorithmCombine": "C2"}}

新文章通过ipython给数据,一定要ipython，不要用python



和hot一样都要等一会

如何查看

ZRANGE "ch:18:new" 0 -1 WITHSCORES

### 4.4.2 添加热门以及新文章kafka配置信息

* streaming 对接kafka配置
* kafka主题开启的新配置

### 4.4.3 编写热门文章收集程序

* 热门文章
  + 在线实时进行redis读取存储
  + 热门收集程序
  + 写入新日志数据进行测试

### 4.4.4 编写新文章收集程序

### 4.4.5 添加supervisor在线实时运行进程管理

# 5.1 实时推荐业务介绍

1、后端发送推荐请求，实时推荐系统拿到请求参数

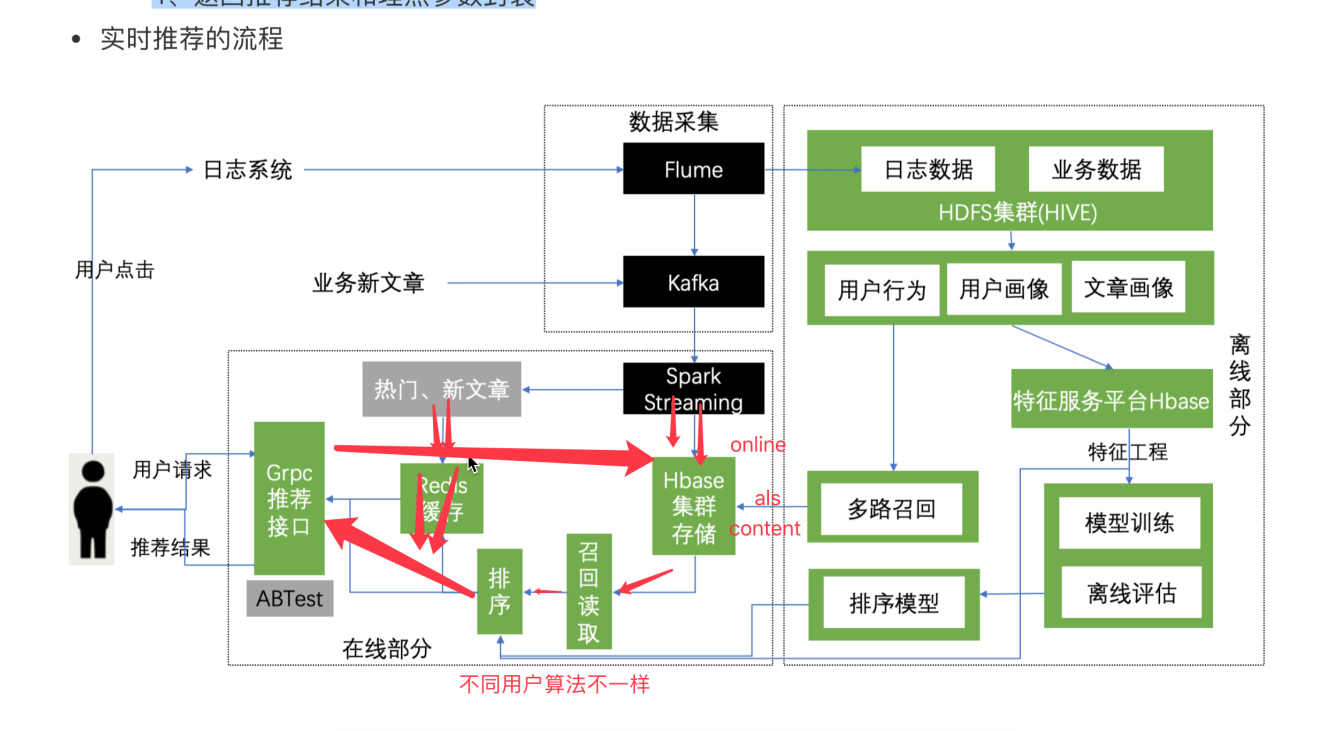
* grpc对接

2、根据用户进行ABTest分流

* ABTest实验中心，用于进行分流任务，方便测试调整不同的模型上线
* 30 LR , 70 wide&deep---用户分为两类

3、推荐中心服务

* 根据用户在ABTest分配的算法进行召回服务和排序服务读取返回结果
* 4、返回推荐结果和埋点参数封装



右边部分已做完，召回结果存入了hbase中

在线的spark streaming的online召回也做了

召回读取，通过LR,或者wide-deep排序后的结果推荐，加上redis里边的新文章，热门文章，通过grpc推荐接口，推荐给用户

# 5.2 grpc接口对接

### 5.2.1 头条推荐接口对接

* feed流推荐：用户ID，频道ID，推荐文章数量，请求推荐时间戳
* 相似文章获取：文章ID，推荐文章数量

返回参数：

* feed流推荐：曝光参数，每篇文章的所有行为参数，上一条时间戳

# 埋点参数参考：

# {

# "param": '{"action": "exposure", "userId": 1, "articleId": [1,2,3,4], "algorithmCombine": "c1"}',给用户传了1,2,3,4总计4篇文章

# "recommends": [构造好格式，用户做了哪些操作，后台直接存储到日志中

# {"article\_id": 1, "param": {"click": "{"action": "click", "userId": "1", "articleId": 1, "algorithmCombine": 'c1'}", "collect": "", "share": "","read":""}},

# {"article\_id": 2, "param": {"click": "", "collect": "", "share": "", "read":""}},

# {"article\_id": 3, "param": {"click": "", "collect": "", "share": "", "read":""}},

# {"article\_id": 4, "param": {"click": "", "collect": "", "share": "", "read":""}}

# ]

# "timestamp": 1546391572

# }



### 5.2.2 简介

1. 使用Protocol Buffers（proto3）的IDL接口定义语言定义接口服务，编写在文本文件（以.proto为后缀名）中。
2. 使用protobuf编译器生成服务器和客户端使用的stub代码

pip install grpcio-tools

安装不上需要

pip3 install *--upgrade pip*

python3 -m pip install *--upgrade setuptools*

pip3 install grpcio-tools

导入使用时，是import grpc

**Grpc的helloword的入门：**

**Server解析：**

**1、启动grpc.server**

**2、注册服务，对某个对象进行注册，**

**helloworld\_pb2\_grpc.add\_GreeterServicer\_to\_server(Greeter(), server)**

**3、开始监听，等待客户端情况**

**Client端解析：**

**1、通过stub = helloworld\_pb2\_grpc.GreeterStub(channel)得到stub对象**

**2、通关stub调用服务器写好的方法**

**response = stub.SayHello(helloworld\_pb2.HelloRequest(name='you'))**

### 5.2.3 代码结构

* 消息类型
  + 参数类型：字段类型
* 字段编号
  + 消息定义中的每个字段都有唯一的编号。
* **repeated string snippets：表示列表类型**
* 可以在单个.proto文件中定义多个消息类型。
* 编译生成代码

### 5.2.4 王道头条推荐接口protoco协议定义

* # 埋点参数参考：  
  # {  
  # "param": '{"action": "exposure", "userId": 1, "articleId": [1,2,3,4], "algorithmCombine": "c1"}',  
  # "recommends": [  
  # {"article\_id": 1, "params": {"click": "{"action": "click", "userId": "1", "articleId": 1, "algorithmCombine": 'c1'}", "collect": "", "share": "","read":""}},  
  # {"article\_id": 2, "params": {"click": "", "collect": "", "share": "", "read":""}},  
  # {"article\_id": 3, "params": {"click": "", "collect": "", "share": "", "read":""}},  
  # {"article\_id": 4, "params": {"click": "", "collect": "", "share": "", "read":""}}  
  # ]  
  # "timestamp": 1546391572  
  # }

python -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. user\_reco.proto

这个命令是在abtest目录下使用

* + 生成所需要的服务器编写部分代码和客户端部分代码
* user\_reco\_pb2\_grpc.py user\_reco\_pb2.py
* 上面两个文件，服务器端和客户端均需要使用

上面两个文件，服务器端和客户端均需要使用

代码执行流程 ：

用户请求给予文章，这时后端开发收到用户的请求，后端开发相当于grpctest，

Grpctest请求参数是用户id，频道，文章数目，服务器routing就会接收到请求，就会调用server文件夹下的reco\_cent进行数据处理，讲cb\_recall召回集拿出来，通过LR排序模型进行排序，排序以后就是要推荐给用户的结果，这个结果有100条，会放到hbase的wait\_recommend,为了提高效率，会将待推荐结果存入redis中，为了LR排序模型得到的排序结果存在相同的文章，因此每次存入wait\_recommend时，也存入history\_recommend。

### 5.2.4 王道头条grpc服务端编写

* 服务器端：返回的参数封装---协议封装
  + \_param1 = []  
     for \_ in \_track['recommends']:  
     # param的封装  
     \_params = user\_reco\_pb2.param2(click=\_['param']['click'],  
     collect=\_['param']['collect'],  
     share=\_['param']['share'],  
     read=\_['param']['read'])  
     \_p2 = user\_reco\_pb2.param1(article\_id=\_['article\_id'], params=\_params)  
     \_param1.append(\_p2)  
     # param  
     return user\_reco\_pb2.Track(exposure=\_track['param'], recommends=\_param1, time\_stamp=\_track['timestamp'])

按照proto协议

# 5.3 ABTest实验中心

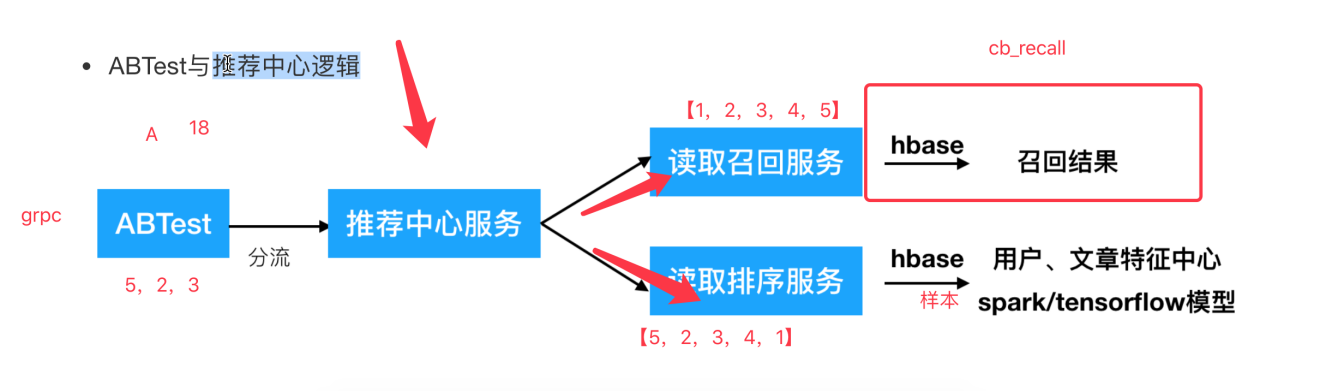
### 5.3.1 ABTest

* 一个是ABTest实时分流服务，根据用户设备信息、用户信息进行ab分流。
* 实时效果分析统计，将分流后程序点击、浏览等通过hive、hadoop程序统计后，在统计平台上进行展示。

### 5.3.2 流量切分

A/B测试的流量切分是在Rank Server端完成的。我们根据用户ID将流量切分为多个桶（Bucket），每个桶对应一种排序策略

* 实验参数:目前有哪些算法需要上线，或者已经运行的
* 哈希分桶，md5
* ABTest分流逻辑实现代码如下
  + 根据用户ID的md5加密，来进行分流----分桶操作



# 5.4 召回集读取服务---对应课件的5.5

### 5.4.1 召回集读取服务

* **这里先添加一个召回集的结果读取服务recall\_service.py**
* **utils.py中装有自己封装的hbase数据库读取存储工具**

### 5.4.1 Hbase读取存储等工具类封装

**在写happybase代码的时候会有过多的重复代码，将这些封装成简便的工具，减少代码冗余**

* get*table*row(self, table*name, key*format, column*format=None, include*timestamp=False):
  + 获取具体表中的键、列族中的行数据
* get*table*cells(self, table*name, key*format, column*format=None, timestamp=None, include*timestamp=False):
  + 获取Hbase中多个版本数据
* get*table*put(self, table*name, key*format, column\_format, data, timestamp=None):
  + 存储数据到Hbase当中
* get*table*delete(self, table*name, key*format, column\_format):
  + 删除Hbase中的数据

### 5.4.2 多路召回结果读取

目的：读取离线和在线存储的召回结果

* hbase的存储：cb\_recall, als, content, online
* 召回读取步骤：
* 1、初始化redis,hbase相关工具
* 2、在线画像召回，离线画像召回，离线协同召回数据的读取
* 3、redis新文章和热门文章结果读取
  + 热门文章读取:热门文章记录了很多，可以选取前K个

操作redis的方法

ZRANGE 'ch:18:new' 0 -1

* 4、相似文章读取接口

召回结果验证通过recall\_server进行实战

# 5.5 推荐中心----对应课件的5.4

### 5.5.3 推荐中心刷新逻辑



* 下拉刷新：请求新数据，召回读取、排序，放入缓存

请求刷新时间戳大于历史时间戳，下来刷新，得到500条数据，给用户发10条即可，490条先后台缓存一下，直到取完

* 上滑请求历史：按照请求的时间戳给出**具体时间的历史看过的文章**

针对历史推过的文章，用时间戳进行记录，用于用户查看历史

* 表设计
  + wait\_recommend: 经过各种多路召回，排序之后的待推荐结果
  + history\_recommend: 每次真正推荐出去给用户的历史推荐结果列表
    - **1、按照频道存储用户的历史推荐结果**
    - **2、需要保留多个版本，才需要建立版本信息**
    - **作用：读取历史记录用的**
    - 10 + 500 = 510 ，10， 500(能加入历史记录中吗？)
      * 缓存中还有数据，并没有加入历史表中

alter 'history\_recommend',NAME => 'channel', VERSIONS=>999999, TTL=>7776000

我们最多保存一个用户3个月内，999999个历史版本

* 这里与上次召回cb *recall以及history* recall有不同用处：
  + 过滤热门和新文章等推荐过的**历史记录**，history\_recommend存入的是真正推荐过的历史记录
  + history\_recall只过滤召回的结果

### 5.5.4 feed流 推荐中心逻辑

* 目的：根据ABTest分流之后的用户，进行制定算法的召回和排序读取
* 1、根据时间戳进行推荐逻辑判断
  + 下拉刷新：
  + 2、读取召回结果(无实时排序)
  + 上滑请求历史
  + 3、返回历史结果
* 请求当前时间戳，返回新推荐结果，+历史记录当中最近的一条时间戳给后台
* 如果请求历史记录1558143073173，返回这个数据
  + [19200, 17665, 16151, 16411, 19233, 13090, 15140, 16421, 19494, 14381]
  + time\_stamp = 1558143073173的前面一条历史时间戳，没有就是0

这里我们运行来验证理解