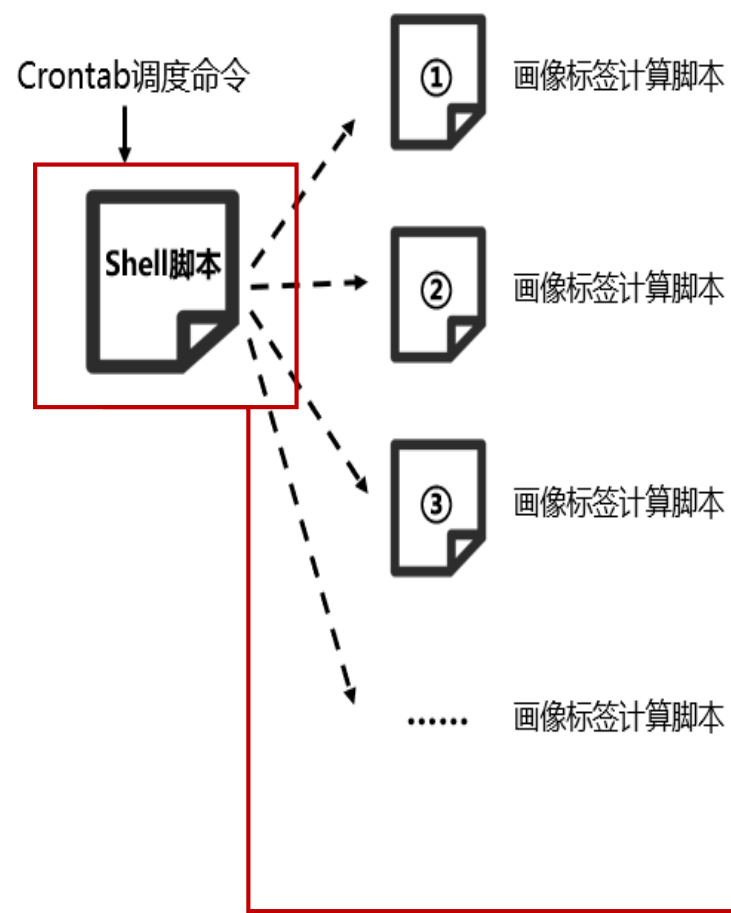


作业流程调度

讲师：watermelon

Crontab命令调度

Crontab命令调度—调度流程



画像开发的初期阶段，为了数据尽快上线迭代，对于标签的调度作业可通过shell脚本和crontab命令完成简单的ETL任务

这里在shell脚本中写一个循环，依次扫描画像标签计算脚本对应文件夹下的文件，即可分别调起各个标签的脚本

对应脚本：main_workflow.py

Crontab命令调度—脚本示例

```
#!/bin/sh

export SPARK_HOME=/usr/local/spark-2.1.1-bin-hadoop2.6
export JAVA_HOME=/usr/local/jdk1.8.0_162/
export PATH=$JAVA_HOME/bin:$PATH

/usr/bin/python /home/userprofile/work/main_workflow.py

/home/userprofile/work/userprofile_to_eda_hbase.sh
```

Shell脚本配置环境变量，然后调起python任务

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import datetime
import os

if len(sys.argv) < 2:
    today = datetime.datetime.today()
    oneday = datetime.timedelta(days=1)
    yesterday = today - oneday
    datestr = yesterday.strftime("%Y%m%d")
else:
    datestr= sys.argv[1]
```

传入日期参数，便于回溯数据

```
os.system("export PYTHONIOENCODING=utf8")
os.system("export SPARK_HOME=/usr/local/spark-2.1.1-bin-hadoop2.6")
os.system("export JAVA_HOME=/usr/local/jdk1.8.0_162/")
os.system("export PATH=$JAVA_HOME/bin:$PATH")
```

配置环境变量

```
os.system("/usr/local/spark-2.1.1-bin-hadoop2.6/bin/spark-submit --master yarn --deploy-mode client --driver-memory 1g
--executor-memory 8g --executor-cores 2 --num-executors 200 /home/userprofile/work/userprofile_cookieid_gender.py " + datestr)
os.system("/usr/local/spark-2.1.1-bin-hadoop2.6/bin/spark-submit --master yarn --deploy-mode client --driver-memory 1g
--executor-memory 4g --executor-cores 2 --num-executors 50 /home/userprofile/work/userprofile_cookieid_country.py " + datestr)
os.system("/usr/local/spark-2.1.1-bin-hadoop2.6/bin/spark-submit --master yarn --deploy-mode client --driver-memory 4g
--executor-memory 8g --executor-cores 2 --num-executors 50 /home/userprofile/work/userprofile_cookieid_last_behavior_days.py "
+ datestr)
```

调起spark任务，执行计算标签作业

```
.....

os.system("python /home/userprofile/userprofile_email_send.py " + datestr)
```

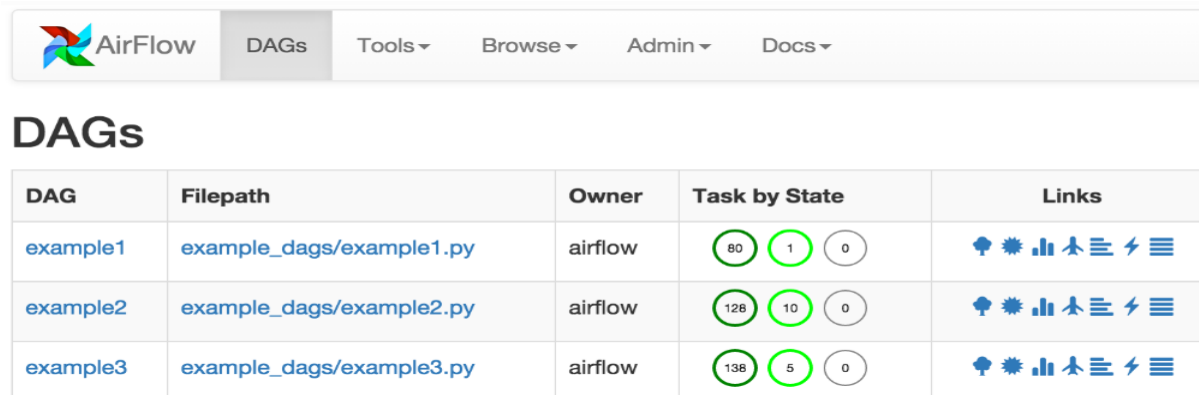
标签监控预警，检查标签数量异常

设置定时任务进行调度，插入执行日志到指定文件中

```
00 08 * * * /usr/bin/sh /home/userprofile/work/main_workflow.sh >> /home/userprofile/work/crontablog/cron_`date +%Y%m%d`.log 2>&1 &
```

Airflow基础概念

Airflow调度—基础概念

The image shows a screenshot of the Apache Airflow web interface. At the top, there is a navigation bar with the Airflow logo and several menu items: 'DAGs' (which is highlighted), 'Tools', 'Browse', 'Admin', and 'Docs'. Below the navigation bar, the title 'DAGs' is displayed. Underneath, there is a table listing three example DAGs. Each row in the table contains the DAG name, its filepath, the owner, a 'Task by State' section with three circular progress indicators (green, yellow, and red), and a 'Links' section with various icons for actions like refresh, view, and delete.

DAG	Filepath	Owner	Task by State	Links
example1	example_dags/example1.py	airflow	<div><div>80</div><div>1</div><div>0</div></div>	🔄 📊 👤 ⚡ ☰
example2	example_dags/example2.py	airflow	<div><div>128</div><div>10</div><div>0</div></div>	🔄 📊 👤 ⚡ ☰
example3	example_dags/example3.py	airflow	<div><div>138</div><div>5</div><div>0</div></div>	🔄 📊 👤 ⚡ ☰

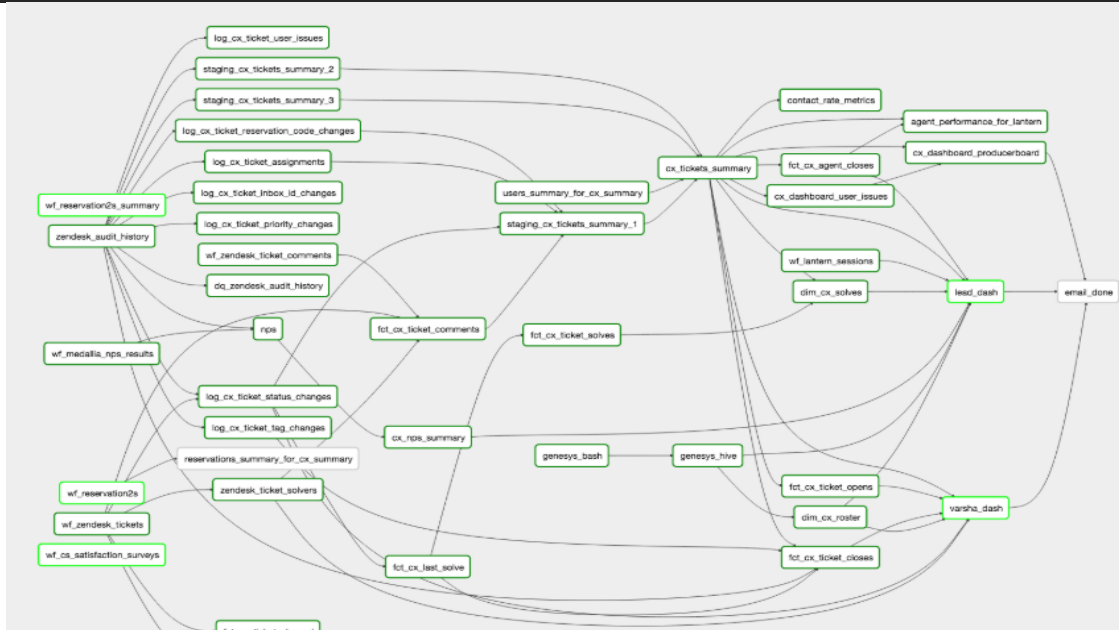
Airflow是Airbnb内部发起的一个工作流管理平台。使用Python编写实现的任务管理、调度、监控工作流平台。

Airflow的调度依赖于crontab命令，与crontab相比Airflow可以方便查看任务的执行状况（执行是否成功、执行时间、执行依赖等），可追踪任务历史执行情况，任务执行失败时可以收到邮件通知，查看错误日志。对于管理调度任务有很大的帮助。

Crontab命令管理调度的方式总结来看存在以下几方面的弊端：

- 1、在多任务调度执行的情况下，难以理清任务之间的依赖关系；
- 2、不便于查看当前执行到哪一个任务；
- 3、不便于查看调度流下每个任务执行的起止消耗时间，而这对于优化task作业是非常重要的；
- 4、不便于记录历史调度任务的执行情况，而这对于优化作业和排查错误是非常重要的；
- 5、执行任务失败时不便于查看执行日志，不方便定位报错的任务和接收错误告警邮件；

Airflow调度—基础概念

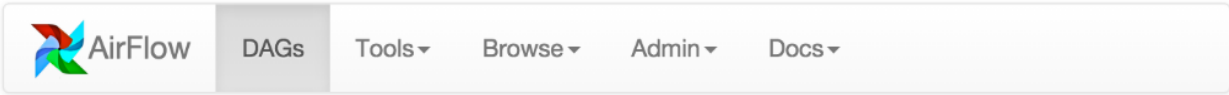


在介绍Airflow这个调度工具前先介绍几个相关的基础概念

- **DAG**：即有向无环图（Directed Acyclic Graph），DAG用于描述数据流的计算过程；
- **Operators**：描述了DAG中一个具体task要执行的任务，如BashOperator为执行一条bash命令，EmailOperator用于发送邮件，HTTPOperator用于发送HTTP请求，PythonOperator用于调用任意的Python函数；
- **Task**：是Operator的一个实例，也就是DAG中的一个节点；
- **Task Instance**：记录task的一次运行。Task Instance有自己的状态，包括“running”、“success”、“failed”、“skipped”、“up for retry”等；
- **Trigger Rules**：指task的触发条件；

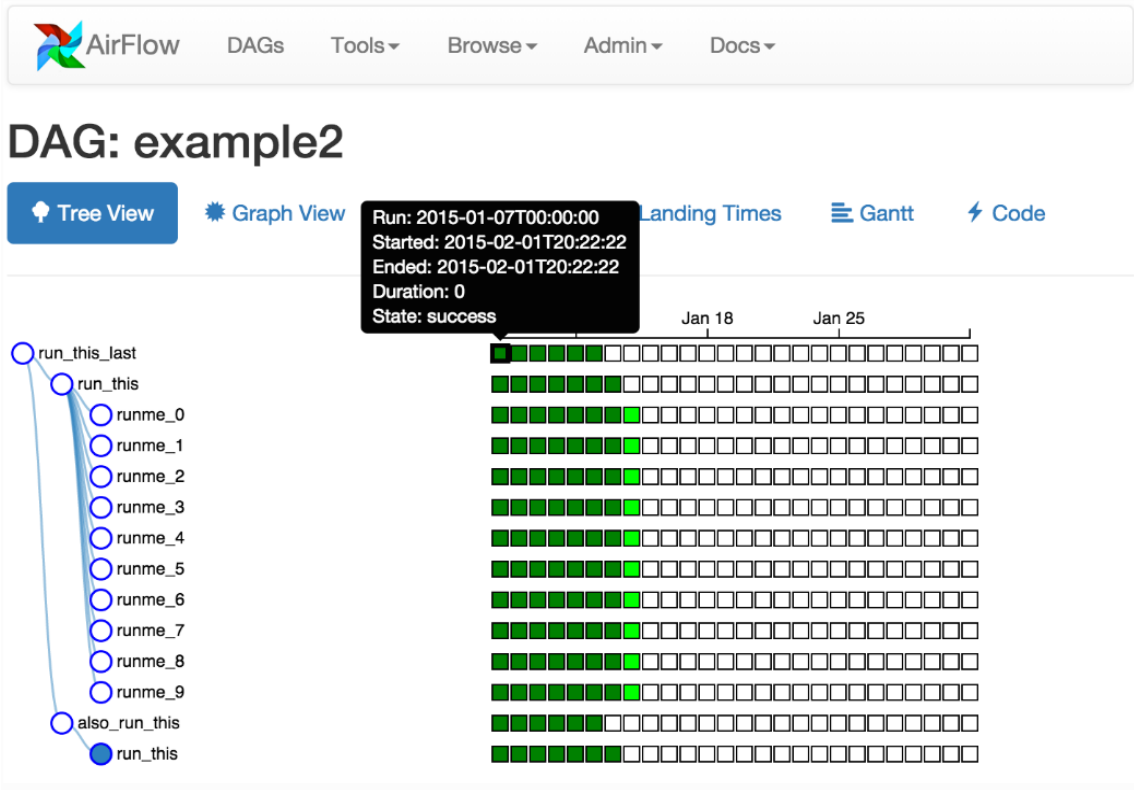
Airflow调度—基础概念

Airflow的DAG文件可视为linux下调度任务的shell脚本

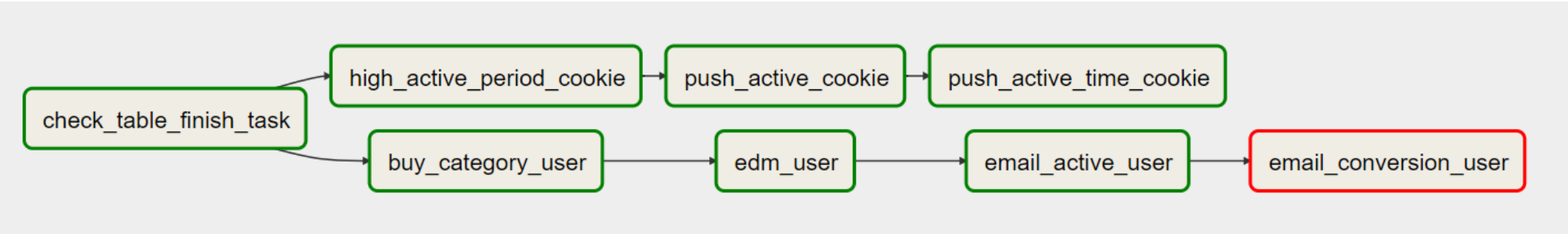


DAGs

DAG	Filepath	Owner	Task by State	Links
example1	example_dags/example1.py	airflow	<div><div>80</div><div>1</div><div>0</div></div>	📌 🔍 📊 👤 ⚙️ ⚡ ☰
example2	example_dags/example2.py	airflow	<div><div>128</div><div>10</div><div>0</div></div>	📌 🔍 📊 👤 ⚙️ ⚡ ☰
example3	example_dags/example3.py	airflow	<div><div>138</div><div>5</div><div>0</div></div>	📌 🔍 📊 👤 ⚙️ ⚡ ☰




上图的DAG相当于执行的调度任务。点击某个DAG任务后，可以进入查看该DAG任务的执行脚本及作业情况。如右图



一个调度用户画像脚本的流程图

<http://airflow.incubator.apache.org/tutorial.html>

 Airflow

Search docs

Project

License

Quick Start

Installation

☰ Tutorial

Example Pipeline definition

It's a DAG definition file

Importing Modules

Default Arguments

Instantiate a DAG

Tasks

Templating with Jinja

Setting up Dependencies

Recap

☰ Testing

Docs » Tutorial

[View page source](#)

Tutorial

This tutorial walks you through some of the fundamental Airflow concepts, objects, and their usage while writing your first pipeline.

Example Pipeline definition

Here is an example of a basic pipeline definition. Do not worry if this looks complicated, a line by line explanation follows below.

```
"""
Code that goes along with the Airflow tutorial located at:
https://github.com/apache/incubator-airflow/blob/master/airflow/example_dags/tutorial.py
"""

from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
```

Airflow的安装、配置、使用文档，在airflow的官网中有详细的介绍和demo

Airflow安装

Airflow安装

Airflow的安装十分简单，使用：pip install airflow即可

修改airflow对应的环境变量：export AIRFLOW_HOME=/home/airflow

修改默认数据库：vim /home/airflow/airflow.cfg

修改sql的配置：sql_alchemy_conn=sqlite:///home/userprofile/airflow/airflow.db

初始化数据库：airflow initdb

启动web服务器：airflow webserver -p 8080

```
[root@node-2 ~]#  
[root@node-2 ~]# pip install airflow  
Collecting airflow  
  Downloading https://files.pythonhosted.org/packages/e7/ac/5f1ec362fc0695167d29b3c7b6f28d79898f1221e5a32ab1c6e651a55564/airflow-1.8.0.tar.gz (8.4MB)  
    100% |#####| 8.4MB 291kB/s  
Collecting alembic<0.9,>=0.8.3 (from airflow)  
  Downloading https://files.pythonhosted.org/packages/f0/7d/7fcd63887d9726e0145e98802baf374ec8cf889325e469194cd7926c98e/alembic-0.8.10.tar.gz (976kB)  
    100% |#####| 983kB 879kB/s  
Collecting croniter<0.4,>=0.3.8 (from airflow)  
  Downloading https://files.pythonhosted.org/packages/1f/96/700cea52151d14af35cfe33966da872e04b0440b86bcbee25c0abda85745/croniter-0.3.25-py2.py3-none-any.whl  
Collecting dill<0.3,>=0.2.2 (from airflow)  
  Downloading https://files.pythonhosted.org/packages/6f/78/8b96476f4ae426db71c6e86a8e6a81407f015b34547e442291cd397b18f3/dill-0.2.8.2.tar.gz (150kB)  
    100% |#####| 153kB 1.0MB/s  
Collecting flask<0.12,>=0.11 (from airflow)  
  Downloading https://files.pythonhosted.org/packages/63/2b/01f5ed23a78391f6e3e73075973da0ecb467c831376a0b09c0ec5afd7977/Flask-0.11.1-py2.py3-none-any.whl (80kB)  
    100% |#####| 81kB 493kB/s  
Collecting flask-admin==1.4.1 (from airflow)  
  Downloading https://files.pythonhosted.org/packages/39/65/eb6238c40cd9ec20279969ef9ee7ac412fc7c9a6681965bcd58a63eeac2b/Flask-Admin-1.4.1.tar.gz (922kB)  
    93% |#####| 860kB 608kB/s eta 0:00:01
```

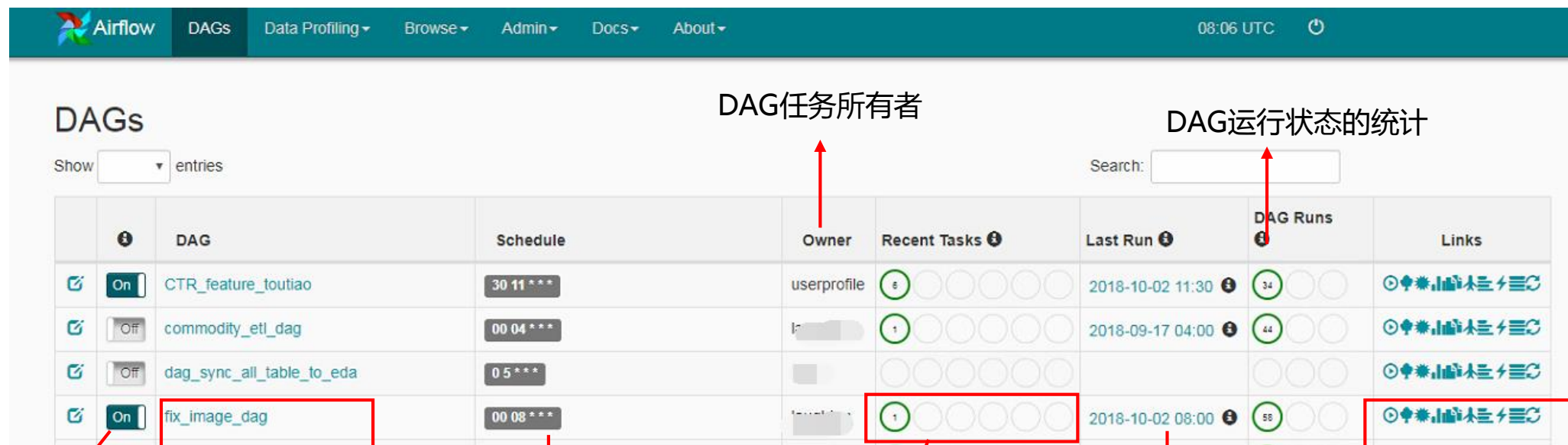
Airflow主要功能模块

Airflow调度—主要功能模块

本节内容主要通过Airflow管理界面的各个模块带大家了解一下其功能

下面这个界面是Airflow调度任务管理的主界面

可以查看当前的DAG任务列表，有多少任务运行成功，失败以及正在当前运行中



The screenshot shows the Airflow web interface with the 'DAGs' tab selected. The table lists several DAGs with columns for status, name, schedule, owner, recent tasks, last run, and DAG runs. Red boxes and arrows highlight specific features: the 'On' toggle for 'fix_image_dag', the DAG name 'fix_image_dag', the schedule '00 08 ***', the '1' in the 'Recent Tasks' column, the '2018-10-02 08:00' in the 'Last Run' column, and the 'DAG Runs' column with its sub-label 'DAG运行状态的统计'.

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	CTR_feature_toutiao	30 11 ***	userprofile	5	2018-10-02 11:30	34	🔍 📊 🔄 📄 ⚙️
<input type="checkbox"/>	commodity_etl_dag	00 04 ***	l...	1	2018-09-17 04:00	44	🔍 📊 🔄 📄 ⚙️
<input type="checkbox"/>	dag_sync_all_table_to_eda	0 5 ***					🔍 📊 🔄 📄 ⚙️
<input checked="" type="checkbox"/>	fix_image_dag	00 08 ***		1	2018-10-02 08:00	58	🔍 📊 🔄 📄 ⚙️

是否开启DAG
调度任务

DAG调度任务的名称，
点击该链接进入该
DAG的调度流程中

DAG调度时间

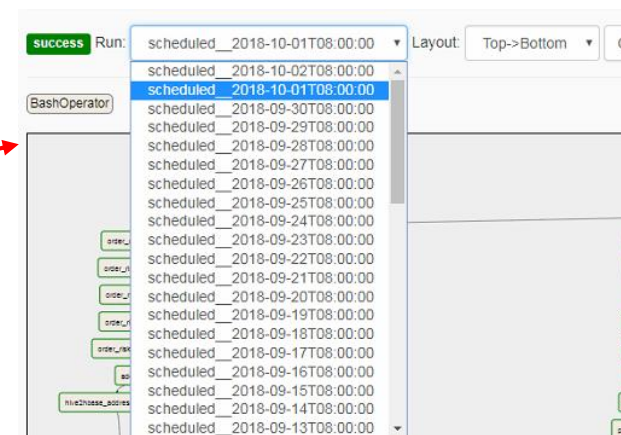
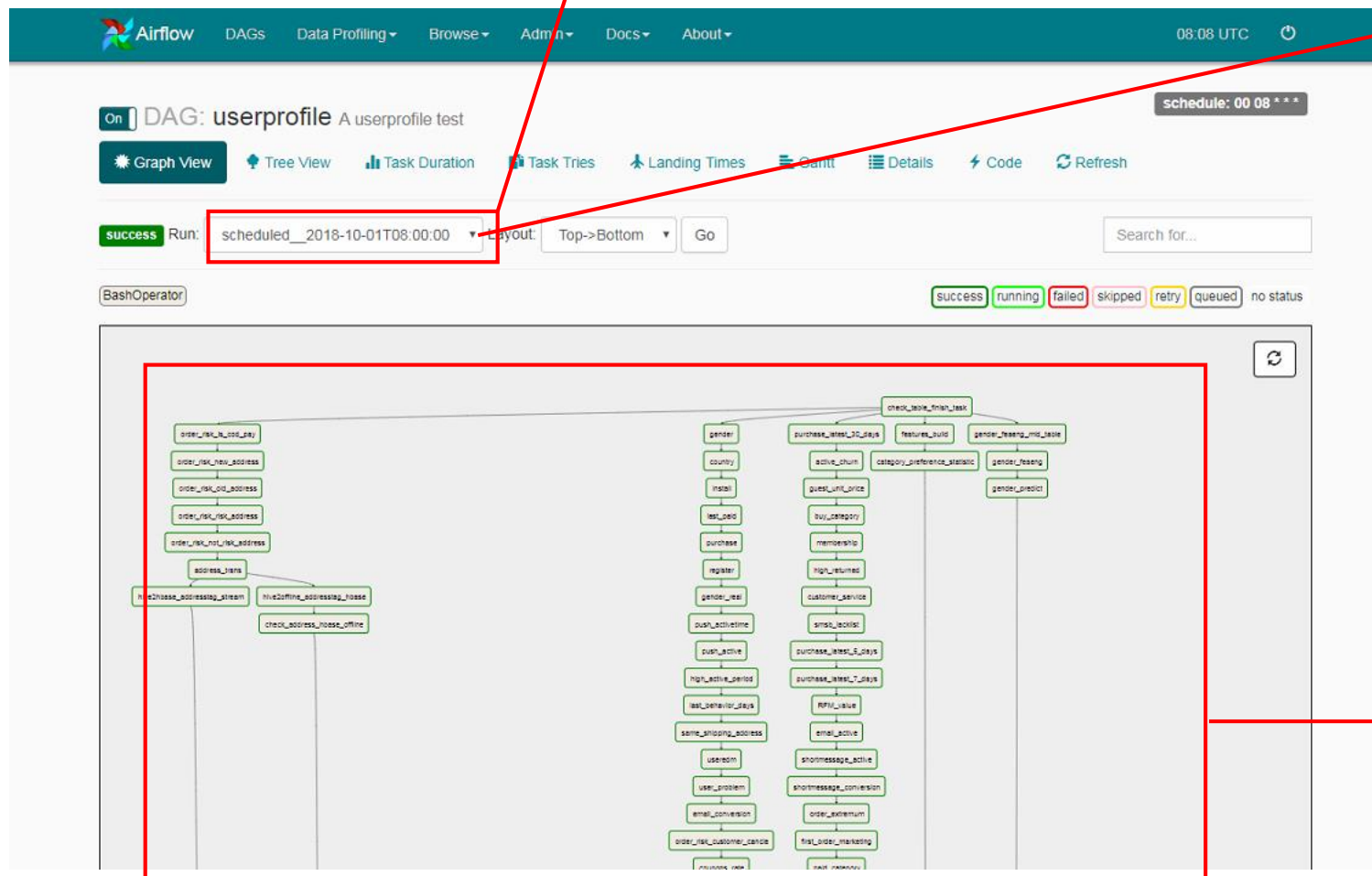
显示最近有多少任务运
行成功/失败/等待

最近一次运行时间

查看每个任务的运行状态/
甘特图/DAG脚本等

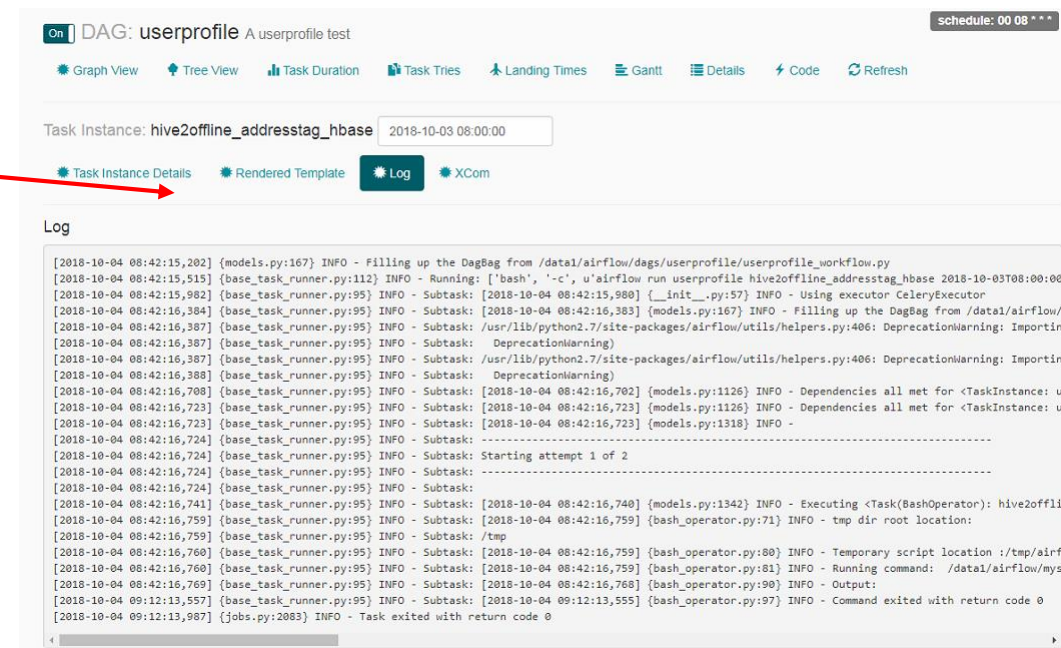
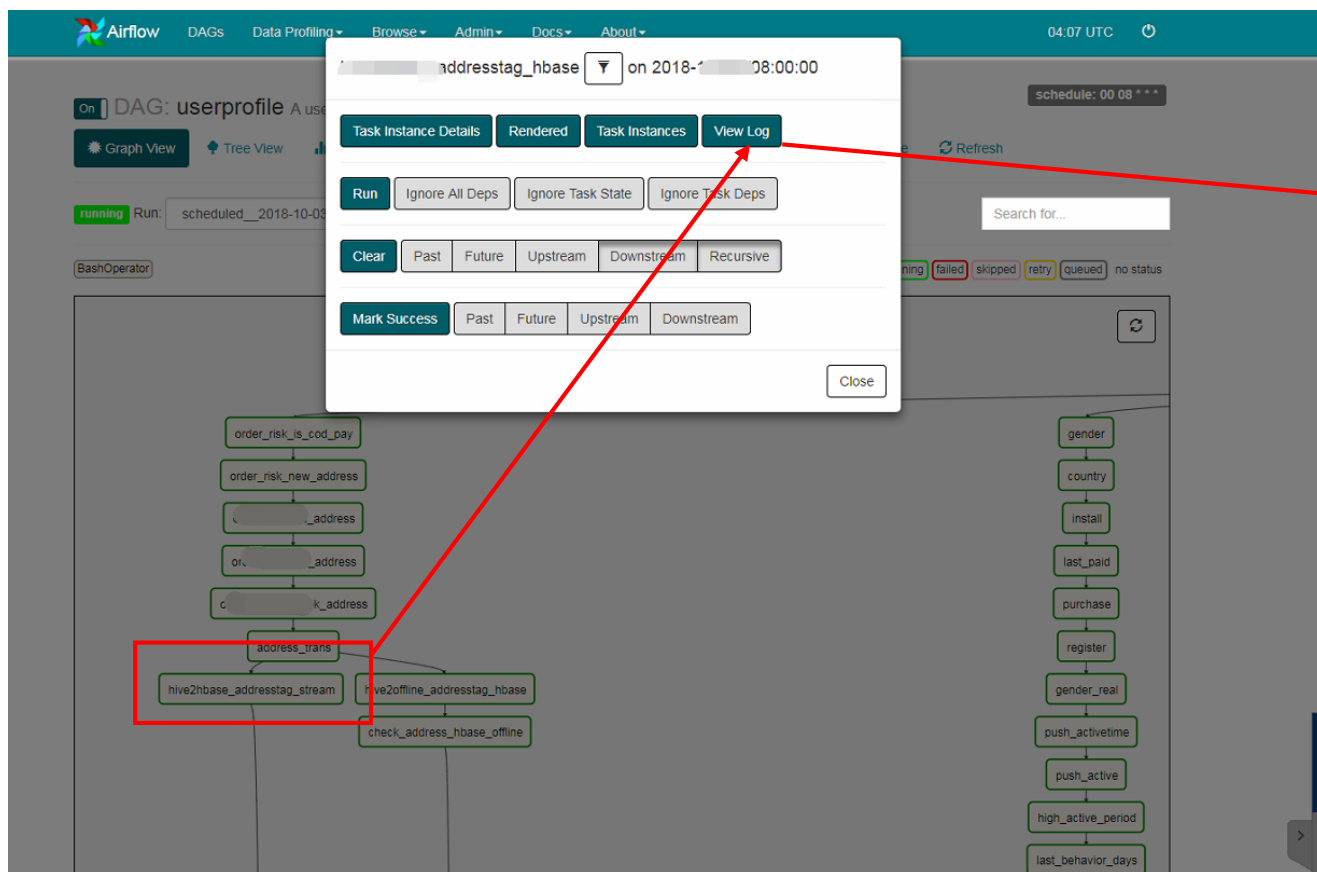
Airflow调度—主要功能模块

显示该DAG的调度日期，可以追溯查看之前日期的调度情况



显示当前DAG调度中，各个模块之间调度的依赖和先后顺序

Airflow调度—主要功能模块



查看每个task的运行日志，对应运行报错的task，可根据日志文件进行排错

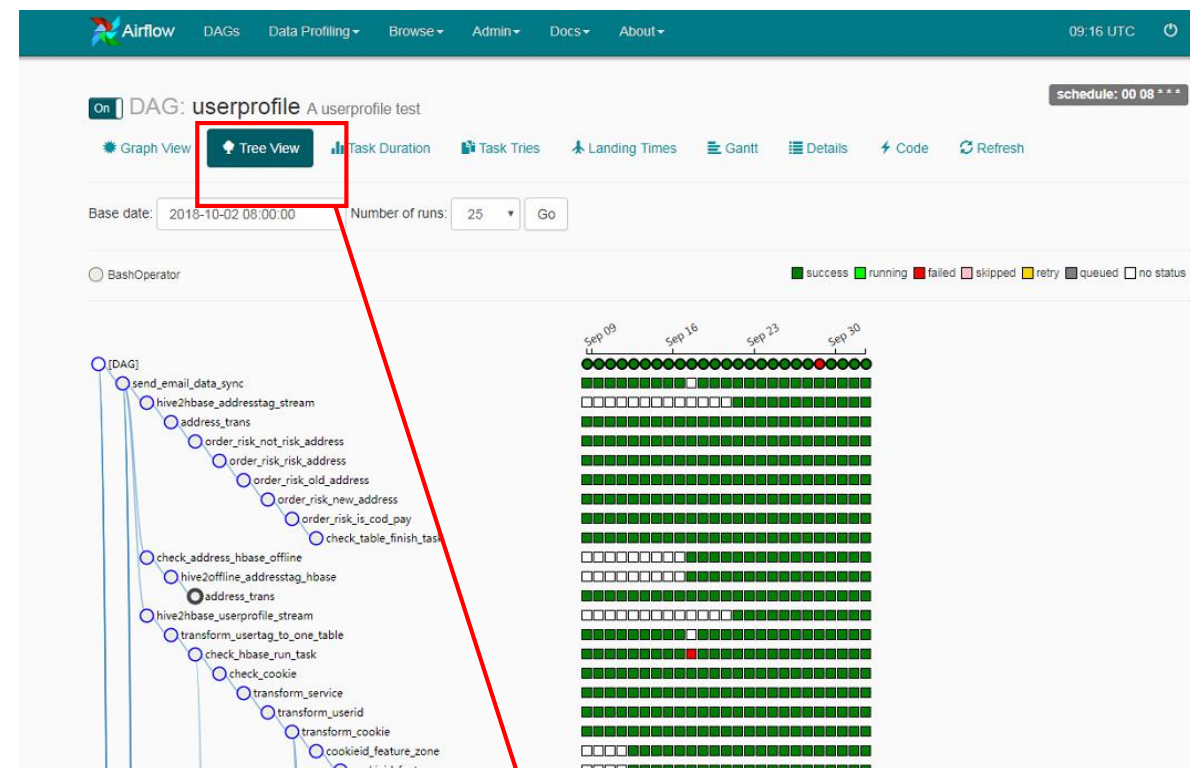
点击对应任务的task，选择“View log”可以查看该task对应的运行日志

Run：运行当前task任务；

Clear：清除当前task及之后task的任务状态；

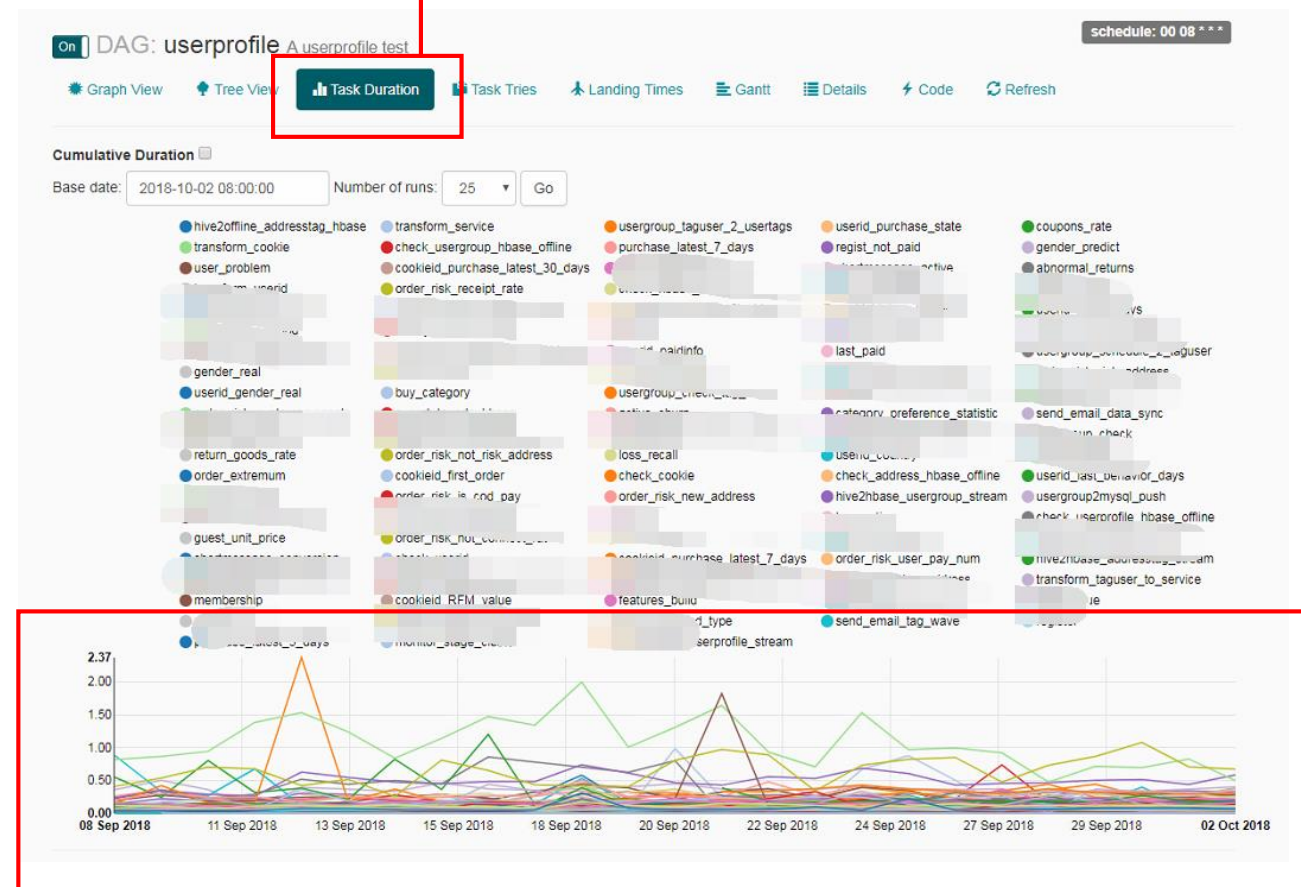
Mark Success：标记当前task的状态为成功，对于后续任务依赖前一个的状态为成功的来说，标记成功不影响后续任务运行；

Airflow调度—主要功能模块



这个是保留历史状态的DAG树，以树状图的形式展示各个task任务的调度情况（成功/失败/正在运行）

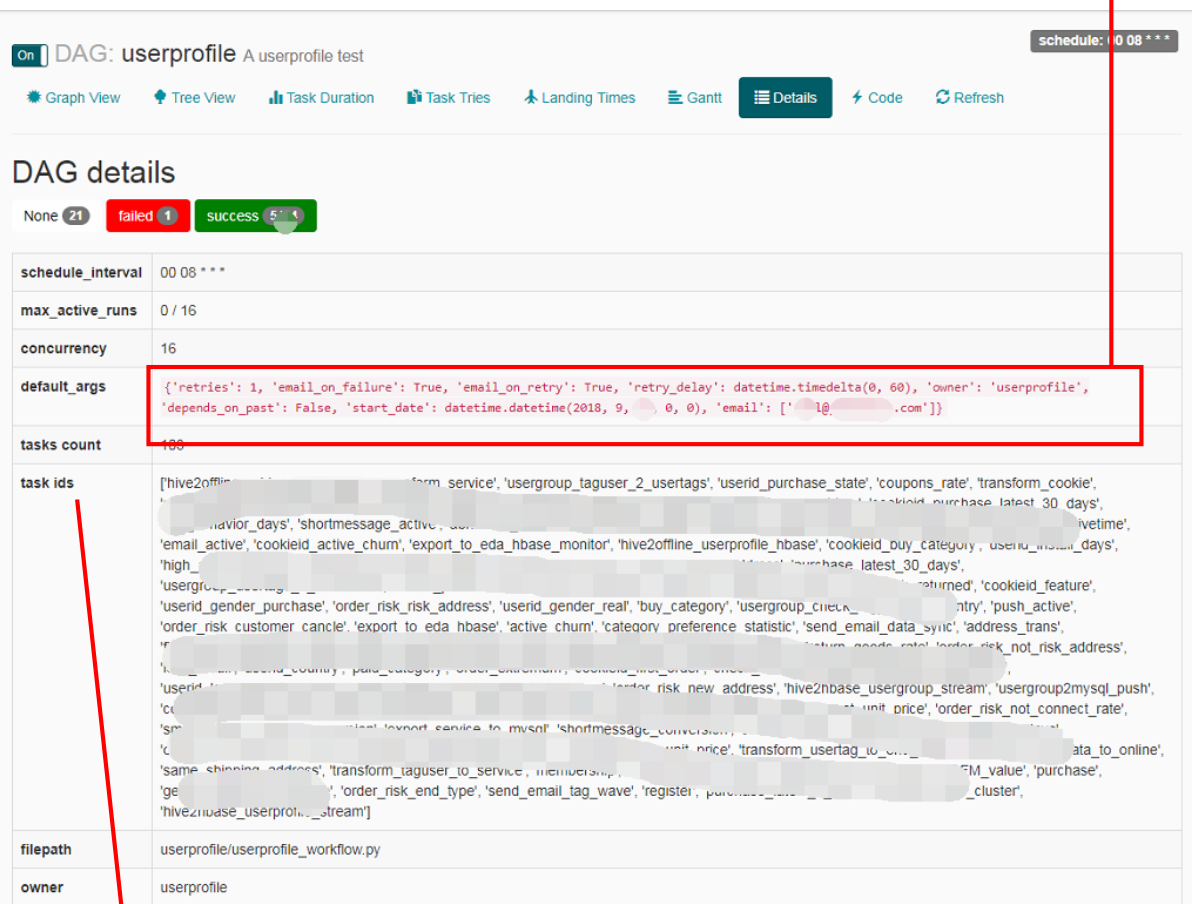
显示该DAG调度的持续时间



这里显示过去n日，不同任务（task）的持续时间。可以帮助找到异常值，快速理解每个任务的运行时间，以便进行排错和调优

配置DAG运行的默认参数

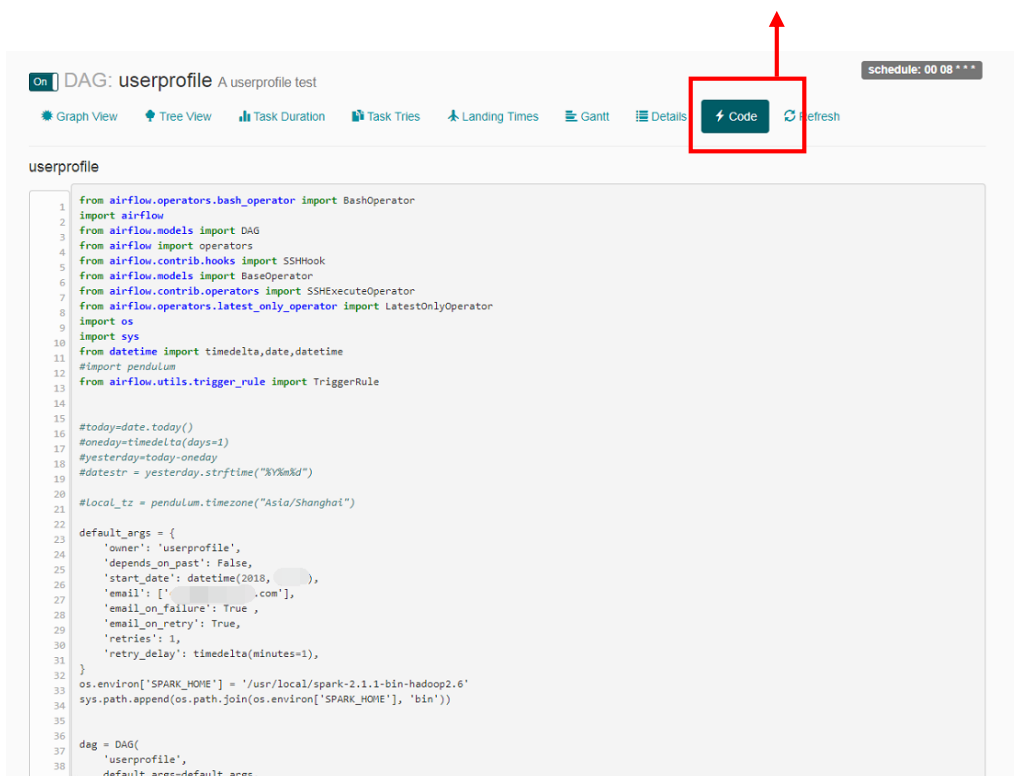
配置DAG运行的默认参数



待调度的任务task

Airflow调度—主要功能模块

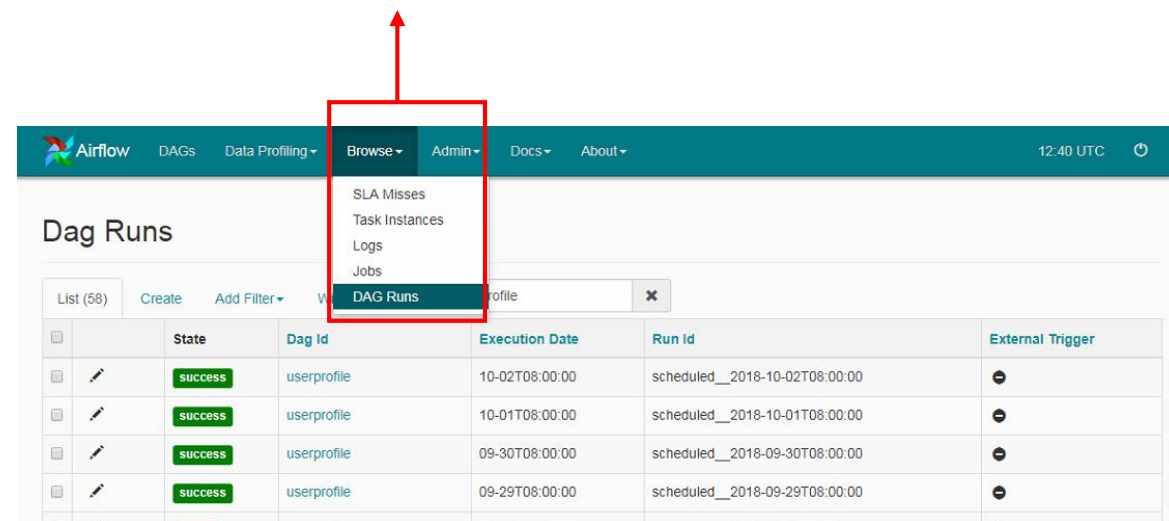
查看该DAG对应的调度脚本



The screenshot shows the Airflow web interface for a DAG named 'userprofile'. The 'Code' button is highlighted with a red box and an arrow pointing to it. The DAG is in a 'scheduled' state with a schedule of '00 08 * * *'. The code editor displays the following Python code:

```
1 from airflow.operators.bash_operator import BashOperator
2 import airflow
3 from airflow.models import DAG
4 from airflow import operators
5 from airflow.contrib.hooks import SSHHook
6 from airflow.models import BaseOperator
7 from airflow.contrib.operators import SSHExecuteOperator
8 from airflow.operators.latest_only_operator import LatestOnlyOperator
9 import os
10 import sys
11 from datetime import timedelta, date, datetime
12 #import pendulum
13 from airflow.utils.trigger_rule import TriggerRule
14
15 #today=date.today()
16 #oneday=timedelta(days=1)
17 #yesterday=today-oneday
18 #datestr = yesterday.strftime("%Y%m%d")
19
20 #local_tz = pendulum.timezone("Asia/Shanghai")
21
22 default_args = {
23     'owner': 'userprofile',
24     'depends_on_past': False,
25     'start_date': datetime(2018, , ),
26     'email': ['@.com'],
27     'email_on_failure': True,
28     'email_on_retry': True,
29     'retries': 1,
30     'retry_delay': timedelta(minutes=1),
31 }
32
33 os.environ['SPARK_HOME'] = '/usr/local/spark-2.1.1-bin-hadoop2.6'
34 sys.path.append(os.path.join(os.environ['SPARK_HOME'], 'bin'))
35
36 dag = DAG(
37     'userprofile',
38     default_args=default_args,
```

如果DAG调度出现问题，可以从该模块筛选对应的DAG进行查看和整理



关于DAG的调度有几个地方需要注意一下：

- 配置参数里的Start_date是DAG首次运行的时间，如果配置时间在前面，会把历史任务同时调起来；
- 依赖参数里面常用的上游依赖包括“ALL_SUCCESS” “ALL_DONE”，前者只有在上游执行成功时才会调起下游任务，后者只要上游任务执行完毕（不论是否执行成功）都可以调起下游任务；

Airflow workflow scheduling

DAG脚本示例 (1)

```
from airflow.operators.bash_operator import BashOperator
import airflow
from airflow.models import DAG
from airflow import operators
from airflow.contrib.hooks import SSHHook
from airflow.models import BaseOperator
from airflow.contrib.operators import SSHExecuteOperator
from airflow.operators.latest_only_operator import LatestOnlyOperator
import os
import sys
from datetime import timedelta, date, datetime
import pendulum
from airflow.utils.trigger_rule import TriggerRule

default_args = {
    'owner': 'userprofile',
    'depends_on_past': False,
    'start_date': datetime(2018, 07, 01),
    'email': ['administer@testemail.com'],
    'email_on_failure': True,
    'email_on_retry': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}

os.environ['SPARK_HOME'] = '/usr/local/spark-2.1.1-bin-hadoop2.6'
sys.path.append(os.path.join(os.environ['SPARK_HOME'], 'bin'))

dag = DAG(
    'userprofile_dag',
    default_args=default_args,
    description='A userprofile test',
    schedule_interval='00 07 * * *')
```

引入需要的包

DAG默认参数配置

实例化DAG

- depends_on_past：是否依赖上游任务，即上一个调度任务执行失败时，该任务是否执行。可选项包括True和False，False表示当前执行脚本不依赖上游执行任务是否成功；
- start_date：表示首次任务的执行日期；
- email：设定当任务出现失败时，用于接受失败报警邮件的邮箱地址；
- email_on_failure：当任务执行失败时，是否发送邮件。可选项包括True和False，True表示失败时将发送邮件；
- retries：表示执行失败时是否重新调起任务执行，1表示会重新调起；
- retry_delay：表示重新调起执行任务的时间间隔；
- 设定该DAG脚本的id为用户profile_dag；
- 设定每天的定时任务执行时间为早上7点

DAG脚本示例（2）

```
high_active_period = BashOperator(↵
    task_id='high_active_period', ↵
    bash_command='spark-submit --master yarn --deploy-mode client --driver-memory 4g --
executor-memory 8g --executor-cores 2 --num-executors 100 cookieid_high_active_period.py
{{ ds_nodash }} ', ↵
    dag=dag, ↵
    trigger_rule=TriggerRule.ALL_DONE) ↵
```

```
↵
push_active = BashOperator(↵
    task_id='push_active', ↵
    bash_command='spark-submit --master yarn --deploy-mode client --driver-memory 4g --
executor-memory 8g --executor-cores 2 --num-executors 100 cookieid_push_active.py
{{ ds_nodash }} ', ↵
    dag=dag, ↵
    trigger_rule=TriggerRule.ALL_DONE) ↵
```

```
... # 配置相应用户画像标签脚本的 task, 这里省略 ↵
buy_category >> userid_edm >> email_active ↵
email_conversion >> paid_category >> sms_blacklist ↵
```

- 左面这段脚本中引入了需要执行的task_id，并对dag进行了实例化。
- 其中以high_active_period这个task_id来说，里面的bash_command参数对于具体执行这个task任务的脚本，cookieid_high_active_period.py文件为执行加工用户高活跃标签对应的脚本。
- Trigger_rule参数为该task任务执行的触发条件，官方文档里面该触发条件有5种状态，一般常用的包括“ALL_DONE”和“ALL_SUCCESS”两种。其中“ALL_DONE”为当上一个task执行完成，该task即可执行，而“ALL_SUCCESS”为只当上一个task执行成功时，该task才能调起执行，执行失败时，本task不执行任务。
- “Bug_category>>userid_edm”命令为task脚本的调度顺序，在该命令中先执行“buy_category”任务后执行“userid_edm”任务。

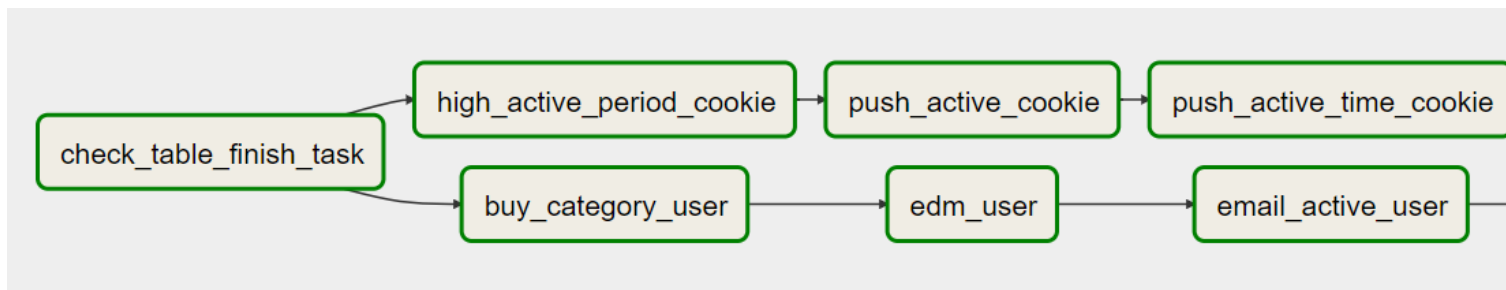
DAG脚本示例（3）

- 一旦Operator被实例化，它被称为“任务”。实例化为在调用抽象Operator时定义一些特定值，参数化任务使之成为DAG中的一个节点

```
op1 >> op2      ==      op1.set_downstream(op2) ↵  
op2 << op1      ==      op2.set_upstream(op1)  ↵
```

这里来看一个DAG调度示例

DAG调度流程图



task执行依赖

```
check_table_finish_task >> high_active_period_cookie >> push_active_cookie >> push_active_time_cookie  
check_table_finish_task >> buy_category_user >> edm_user >> email_active_user
```

首先执行的脚本，如果执行失败，需要过段时间重试

该脚本执行依赖上游任务成功

后面脚本的执行依赖上游执行完成

Airflow常用命令行

Airflow通过可视化界面的方式实现了调度管理的界面操作，但在测试脚本或界面操作失败的时候，可通过命令行的方式调起任务。下面介绍几个常用的命令：

- **airflow list_tasks userprofile**

该命令用于查看当前DAG任务下的所有task列表，其中userprofile是DAG名称；

- **airflow test userprofile age_task 20180701**

该命令用于测试DAG下面某个task是否能正常执行，其中userprofile是DAG名称，age_task是其中一个task名称；

- **airflow backfill -s 2018-07-01 -e 2018-07-02 userprofile**

该命令用于调起整个DAG脚本执行任务，其中userprofile是DAG名称，2018-07-01是脚本执行的开始日期；

Airflow工程案例

工程调度概览

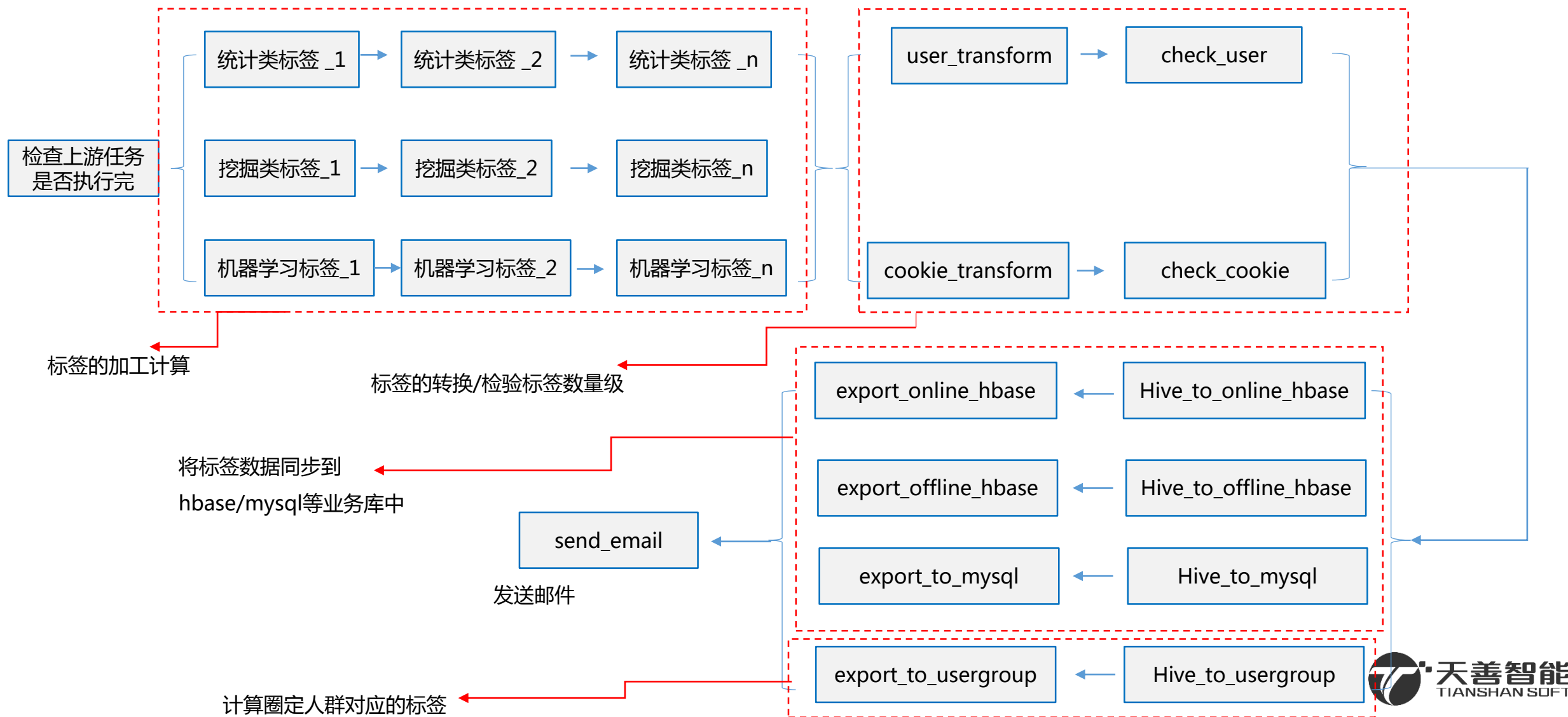


在用户画像工程化调度中主要涉及到的环节/模块：

- **标签调度**：主要的调度任务，负责每天调度计算用户身上的标签，插入对应的标签表中；
- **标签校验**：分为多个模块。①校验每天插入hive表中的标签数据是否出现异常；②校验同步到hbase、关系数据库中的标签数据是否和hive中标签数量级一致；③校验“圈人功能”中计算出来人群对应的标签是否出现异常波动；
- **数据同步**：将用户标签同步到hbase、MySQL等关系数据库的业务系统中。这算是数据服务层的任务；
- **人群计算**：根据产品端业务人员圈定的用户标签组合，计算对应的人群。计算任务使用MapReduce或spark作业将数据插入到hive中，然后同步到对应的业务系统中
- **通知邮件**：数据插入到hive、hbase或关系数据库后校验标签的数量级或波动情况。如超出正常范围则触发报警邮件

工程化调度模块

工程化实践中，每天除了对用户标签执行ETL作业，还需要将与用户标签相关的数据同步到其他数据库或业务系统中，本页内容主要讲整体调度流程，该调度的控制在DAG脚本中即可进行配置



同步到数据服务层（1）

开发后的标签支持到数据服务层（如push系统、广告系统、外呼系统等），可以选择离线方式或实时方式。

离线方式的推送数据质量较为稳定，对于数据团队的运维压力较小，能满足大多数情况的需求；

实时方式对效率要求较高，数据团队既要保证数据准确性又要保障推送出去数据的质量，运维压力较大。本节主要讲下离线支持到服务层的方式

对应筛选的用户维度是在userid或cookieid，筛选不同的维度，推送不同的数据到对应的业务系统中

人群同时满足下列规则

近30日购买次数

区间 大于 小于

活跃度

高活跃 中活跃 低活跃 已流失

人群减法

计算人群数量 18500 人

人群名称

人群描述

保存

对于统计类型标签，用户可以自定义筛选该标签的权重值

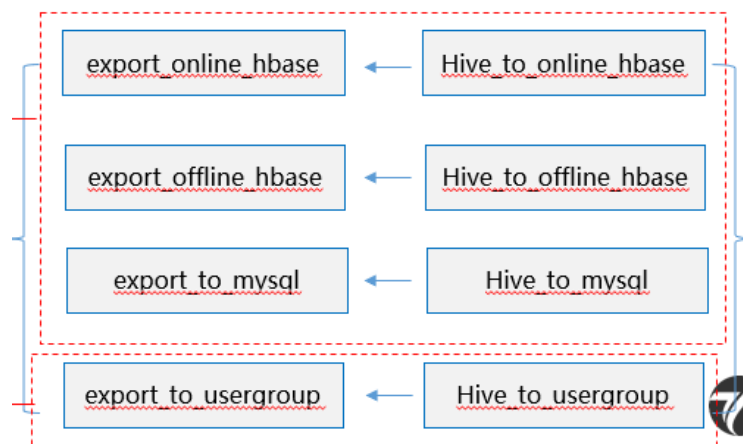
对于分类类型标签，用户不能筛选该标签对应的权重值(该标签只是用户的一个标识，不含统计数值)

根据用户组合标签定义的人群，计算出该人群覆盖的用户数

对自己组合标签筛选出来的用户群命名，然后保存

下页进行详细介绍“人群减法”的功能

天善智能 TIANSHAN SOFT



运营人员在产品端根据标签圈定人群，选择对应的推送系统。
这步操作对应的标签及人群记录存在MySQL中

离线调度流每日ETL调度时定时解析MySQL中记录的人群标签规则，跑作业，计算每个业务规则下人群的用户id或cookieid，推送到对应的业务系统中

同步到数据服务层（2）

不同的业务系统会用到不同的数据库去存储/读取数据，这里例举3种不同的数据库用于存储业务系统的数据，包括MySQL业务库/Hbase业务库/FTP文件传输

MySQL业务库

通过sqoop命令把hive中的数据同步到对应的MySQL库表中

```
os.system("sqoop export --connect jdbc:mysql://xxx.xx.23.142:3307/userprofile --username userprofile --password userprofile --table tag_tmp_userid --export-dir hdfs://master:9000/user/hive/warehouse/dw.db/dw_profile_user_tag_service/data_date=20180901/business=push --input-fields-terminated-by '\\001'")
```

同步后的存储结果

cookie_user_id	tagsmap	reserve	reserve1
10005353	{'A121U013_0_006': '', 'B121U031_0_002': '10', 'A120U014_0_001': '3'}		
10005433	{'B121U031_0_001': '1', 'A121U013_0_006': '', 'A120U014_0_001': '2'}		
10005439	{'A121U013_0_005': '', 'B121U031_0_002': '6', 'A120U014_0_001': '2'}		
10005447	{'A121U013_0_006': '', 'B121U031_0_002': '4', 'A120U015_0_001': '0.4', 'A120U014_0_001': ''}		
10005493	{'B121U031_0_002': '2', 'A120U014_0_001': '2', 'A121U013_0_006': ''}		
10005499	{'B121U031_0_002': '3', 'A120U014_0_001': '3', 'A121U013_0_006': ''}		
10005671	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005707	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005911	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005941	{'A121U013_0_005': '', 'B121U031_0_002': '2', 'A120U014_0_001': '2'}		
10005955	{'A121U013_0_004': '', 'B121U031_0_002': '6', 'A120U014_0_001': '2'}		
10005983	{'A121U013_0_004': '', 'B121U031_0_002': '10', 'A120U014_0_001': '3'}		
10006011	{'B121U031_0_001': '1', 'A121U013_0_005': '', 'A120U015_0_001': '0.5555555555555556'}		

这里可以写一个Python脚本，把对应的hive数据同步到MySQL库表下面

Hbase业务库

```
hive --auxpath $HIVE_HOME/lib/zookeeper-3.4.6.jar,$HIVE_HOME/lib/hive-hbase-handler-2.3.3.jar,$HIVE_HOME/lib/hbase-server-1.1.1.jar --hiveconf hbase.zookeeper.quorum=master,node-1,node-2
```

启动hive

```
CREATE TABLE dw.userprofile_hive_2_hbase
(
  key string,
  value string
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES ("hbase.table.name" = "userprofile_hbase");
```

创建一张映射到hbase的hive表

```
INSERT OVERWRITE TABLE dw.userprofile_hive_2_hbase
SELECT userid,tagid FROM dw.profile_tag_userid;
```

向该映射表插入测试数据

同步到数据服务层 (3)

```
hbase(main):006:0> list
TABLE
userprofile_hbase
1 row(s) in 0.0370 seconds

=> ["userprofile_hbase"]
hbase(main):007:0> scan 'userprofile_hbase'
ROW
16254215 COLUMN+CELL
25083679 column=cf1:val, timestamp=1537758596270, value=A220U029_001
30765587 column=cf1:val, timestamp=1537758593401, value=A220U083_001
32171777 column=cf1:val, timestamp=1537758593401, value=A220U083_001
39730187 column=cf1:val, timestamp=1537758596270, value=A220U029_001
40382657 column=cf1:val, timestamp=1537758593401, value=A220U083_001
4212236 column=cf1:val, timestamp=1537758593401, value=A220U029_001
7306783 column=cf1:val, timestamp=1537758596270, value=A220U083_001
8 row(s) in 0.2590 seconds
```

查询对应hbase表中的数据

FTP传输文件

在呼叫中心进行主动外呼用户的场景中，业务人员根据业务规则圈定好用户，将该批用户相关的信息存储到csv文件中，通过FTP传输到呼叫中心

```
#!/bin/bash

hadoop fs -get /user/userprofile/watermelon/ /home/userprofile/watermelon/csv
cd /home/userprofile/watermelon/csv/hunter

# 处理spark跑完的csv文件
data_date=`date -d "1 days ago" +%Y%m%d` # 今天的前一天
for file in $(ls *)
do
    if [ "${file##*.}" = "csv" ];then
        mv $file 'userprofile_ftp_${data_date}'.csv
        echo $file
    fi
done

# 传输到FTP系统
function logPrint()
{
    echo `date "+%Y/%m/%d %T" - $1`;
}

start_date=`date "+%Y%m%d" -d'-1 day'` # 今天的前一天
data_date=`date -d "+0 day ${start_date}" +%Y%m%d` # 今天的前一天
last_day=`date -d "-1 day ${data_date}" +%Y%m%d` # 今天的前两天
cur_day=`date -d "+1 day ${data_date}" +%Y%m%d` # 今天
logPrint data_date=$data_date
logPrint last_day=$last_day
logPrint cur_day=$cur_day

logPrint
ftp -niv <<- EOF
open 169.xx.xx.xxx
user username password
cd upload
put *${start_date}.csv
bye
EOF
```

把跑完spark作业存储在
hdfs上的csv文件拉到本地

进入本地文件所在目录

将刚才拉取到本地的csv文件进行重命名

打印日志

FTP传输csv文件

标签数据监控预警

上游数据监控

用户画像标签的计算依赖上游数据仓库的数据，每日需要待上游数据仓库ETL调度完成，才开始标签的作业。

在调度前，首先需要检查用到上游数据仓库的表中数据是否已作业完成/是否数量准确

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import MySQLdb #pymysql
import sys
import datetime

data_date_1 = (datetime.datetime.today() - datetime.timedelta(days=1)).strftime("%Y%m%d")
data_date_2 = (datetime.datetime.today() - datetime.timedelta(days=1)).strftime("%Y-%m-%d")

def main():
    #host = "10.10.10.10"
    host = "10.10.10.10"
    db = MySQLdb.connect(host=host, port=3306, user="z", passwd="",
                        db="zydb", charset="utf8")
    #db = MySQLdb.connect(host=host, port=3307, user="u", passwd="", db="u",
    #                    charset="utf8")
    cursor = db.cursor()

    sql_1 = ''' SELECT remark \
                FROM t_step_log \
                WHERE data_date = '{}' \
                AND step = 0 \
                '''.format(data_date_1)
    sql_2 = ''' SELECT task_stat \
                FROM t_sys_log \
                WHERE task_name like '%job_dw_user%' \
                AND task_stat = 2 \
                AND date_format(task_sign, '%Y-%m-%d') = '{}' \
                '''.format(data_date_2)

    cursor.execute(sql_1)
    rows = cursor.fetchall() # 查询出来存在的表
    print(len(rows))
    cursor.execute(sql_2)
    number = cursor.fetchall()
    print(len(number))

    if len(rows) == 1 & len(number) == 1:
        print('All table done')
    else:
        raise RuntimeError("No task is done")

if __name__ == '__main__':
    main()
```

检查上游数据仓库对应的库表数据是否作业完成

如果上游数据作业正常，该脚本正常执行返回0，可以调起后续的作业。如果上游数据异常，该脚本抛出异常，后面画像标签的作业不被调起

对应脚本：check_table_finish.py

Hive标签量级监控

校验每日画像标签跑完ETL作业后，插入对应hive表中各个标签的数量是否正常，并发送邮件，报告标签数量异常对应的标签

```
def main():
    if len(sys.argv) < 2:
        today = datetime.datetime.today()
        oneday = datetime.timedelta(days=1)
        yesterday = today - oneday
        datestr = yesterday.strftime("%Y%m%d")
    else:
        datestr = sys.argv[1]
    db = MySQLdb.connect(host="192.168.1.100", port=3307, user="userprofile", passwd="123456", db="userprofile", charset="utf8")
    cursor = db.cursor()
    resdict = {}
    print "data===== "+datestr
    spark = SparkSession.builder.appName("checkdata_userid").enableHiveSupport().getOrCreate()
    dosql = "select * from (select userid, tagmap from tag_data_day where data_date='"+datestr+"') ta left outer join (select userid as user_id from dw_dw_cookie_user_login where data_date='"+datestr+"' and site_id in ('100', '101', '102', '103', '104', '105', '106', '107', '108', '109', '110', '111', '112', '113', '114', '115', '116', '117', '118', '119', '120')) tb on ta.userid = tb.user_id"
    temprdd = spark.sql(dosql).rdd.cache()
    tagusertotal = temprdd.map(lambda r: (r.userid, r.user_id)).reduceByKey(lambda x, y: x).count()
    daucountsql = "select userid as user_id from tag_data_day where data_date='"+datestr+"' and site_id in ('100', '101', '102', '103', '104', '105', '106', '107', '108', '109', '110', '111', '112', '113', '114', '115', '116', '117', '118', '119', '120') group by userid"
    daucount = spark.sql(daucountsql).count()
    res = temprdd.flatMap(lambda x: funflatmap(x.userid, str(x.tagmap.keys()), x.user_id)).reduceByKey(lambda a, b: a + b).collect()
    for line in res:
        if line[0][0:3] == "dau":
            tagkey = line[0][4:]
            if resdict.has_key(tagkey):
                resdict[tagkey] = (resdict[tagkey][0], resdict[tagkey][1], float(line[1])/float(daucount))
            else:
                resdict[tagkey] = (0, 0, float(line[1])/float(daucount))
        else:
            if resdict.has_key(line[0]):
                resdict[line[0]] = (line[1], float(line[1])/float(tagusertotal), resdict[line[0]][2])
            else:
                resdict[line[0]] = (line[1], float(line[1])/float(tagusertotal), 0)
    for key in resdict.keys():
        desql = "DELETE FROM tag_data_day where tagid='"+key+"' and date='"+datestr+"'"
        sql = "INSERT INTO tag_data_day(tagid,date,tag_count,tag_total_per,tag_dau_per) VALUES ('"+key+"', '"+datestr+"', "+str(resdict[key][0])+", "+str(resdict[key][1])+", "+str(resdict[key][2])+")"
        try:
            cursor.execute(desql)
            # print desql
            # print sql
            cursor.execute(sql)
            db.commit()
        except:
            db.rollback()
    db.close()
```

从hive对应表中抽取标签计算数量

计算每个标签的dau波动和标签数量波动

将计算结果插入到MySQL中

对应脚本：checkdata_tagnum.py



Hbase量级监控

```
# 查询 Hive 数据量
r = os.popen("hive -S -e\"select count(1) from dw.c_ where data_date='"+datestr+'\"")
hive_count = r.read()
r.close()
hive_count = str(int(hive_count))
if not hive_count:
    hive_count = -1
print "hive_result: " + str(hive_count)
print "Hive select finished!"
print "#####"
```

检查某标签插入到hive中的数量

```
# hbase 查询 导入Hbase 数据量
r = os.popen("source /etc/profile; hbase org.apache.hadoop.hbase.mapreduce.RowCounter 'addresstag_'+datestr+' 2>&1 |grep ROWS")
hbase_count = r.read().strip()[5:]
r.close()
if not hbase_count:
    hbase_count = 0
print "hbase result: " + str(hbase_count)
print "HBase select finished!"
print "#####"
```

检查该标签在hbase中的数量

写同步数据结果到 DB

```
try:
    cursor.execute("DELETE FROM tag_email_monitor WHERE date='"+datestr+"' and service_type='addresstag_offline'")
    db.commit()
    cursor.execute("INSERT INTO tag_email_monitor(date, service_type, hive_count, hbase_count) VALUES('"+datestr+"', 'addresstag_offline', '"+str(hive_count)+"', '"+str(hbase_count)+"'")
    db.commit()
    print "Write addresstag hbase check to DB Finished!"
except Exception as e:
    db.rollback()
    raise e
    exit(1)
finally:
    db.close()
```

将hive和hbase的校验结果写到MySQL中，便于追随历史数据

对应脚本：check_hive_hbase.py

监控记录表

id	tagid	date	tag_count	tag_total_per	tag_dau_per
116	A121H002_0_004	2018-04-23	2	0.0616066	0.0720222
117	A121H002_0_009	2018-04-23	2	0.0791658	0.0386327
118	A121H002_0_008	2018-04-23	1	0.0578758	0.0718029
119	A121H031_0_001	2018-04-23	3	0.100767	0.343185
120	A220H029_0_001	2018-04-23	2	0.947207	1.00702
121	A121H002_0_001	2018-04-23	2	0.0788661	0.0724553
122	A121H002_0_010	2018-04-23	2	0.0232904	0.0428363
123	A121H002_0_011	2018-04-23	2	0.0935266	0.111448
124	A121H002_0_005	2018-04-23	2	0.0270433	0.0368206

记录插入到hive中标签的数量、波动率

process_date	stage	state	is_online
2018-09-19	Cluster	1	2
2018-09-17	Cluster	1	2
2018-09-16	Cluster	1	2
2018-09-15	Cluster	1	2
2018-09-14	Cluster	0	2
2018-09-13	Cluster	1	2
2018-09-12	Cluster	0	2

校验完标签的准确性后，记录的标志位。如果标志位为正常，则进行后续调度，否则抛出异常，暂停作业

date	service_type	hive_count	hbase_count
2018-09-24	addresstag_offline	1	5
2018-09-23	userprofile_stream	1	1
2018-09-23	userprofile_offline	1	1
2018-09-23	usergroup_stream	3	3
2018-09-23	usergroup_offline	3	3
2018-09-23	addresstag_stream	3	3
2018-09-23	addresstag_offline	50	33
2018-09-22	userprofile_stream	15	15
2018-09-22	userprofile_offline	15	15

记录hive中标签的数量及同步到hbase中标签的数量

ETL异常问题排查及解决方案

ETL异常问题排查及解决方案

在画像标签每天ETL调度的过程中，难免会遇到调度失败的情况。作业失败时，短时间（小时级别）来看对于圈人服务、BI透视分析来说暂停服务的影响还不算很大，但是对于线上实时推荐的业务来说就会带来用户体验、推荐准确性等关系到营收的影响。因此在调度失败时，快速定位作业失败的原因很关键。

① 资源池内存不足导致作业失败

这个是最常见的失败原因。当集群资源竞争严重时，画像标签的ETL调度很有可能受到影响，关于该种原因的排查，只需查看调度失败任务对应的执行日志文件即可。日志文件中搜索“error”关键词可快速定位到报错原因的位置。

通常因内存不足而引起的作业失败，日志中会报出“java.lang.OutOfMemoryError”等错误类型。

这种情况，需要待资源充足时，重新调起任务

ETL异常问题排查及解决方案

②上游数据ETL延迟导致标签加工失败

该种错误的原因可通过标签数据监控预警邮件发现。标签数据监控预警会报出当日哪些标签的数据量下降幅度超过合理范围内。针对数据量下降异常的标签，查看该标签加工脚本依赖的上游表包括哪些，进一步查看上游表的ETL完成时间是否在标签脚本ETL时间之前。

例如：计算用户历史购买总金额的标签，是从上游的订单信息表中加工得来。平时上游订单信息表的ETL完成时间在早上8:00，计算用户历史购买总金额的标签的ETL时间在8:30，某日订单信息表在8:30还未完成ETL作业，按照设定此时该标签已经开始了ETL任务，当然加工出来的标签数据会失败。

如何定位是否由上游数据ETL延迟而引起的失败，只需要判断画像标签的调度时间是否在上游数据当日ETL完成时间之后进行。例如，对于判断上游订单信息表dw.order_fact数据的当日ETL完成时间，可通过命令“`hadoop fs -ls hdfs://data/user/hive/warehouse/dw/order_fact/data=20180701`”查看，其中“`hdfs://data/user/hive/warehouse/dw/order_fact`”是订单信息表对应的hdfs文件位置，可通过HiveQL语句：“`show create table dw.order_fact`”查看。

③上游数据ETL异常或失败导致标签加工失败

这种错误比第2条错误原因更难发现。当上游数据已经加工完毕，写表落仓后，即时在hdfs上查找上游文件的写入时间也不会发现问题。此时可通过横向对比，该上游表近期每日的数据量来发现是否存储问题。

例如：对于按日期分区的订单信息表dw.order_fact，可通过命令“`select data_date,count(*) from dw.order_fact where data_date>='20180701' and data_date<='20180707' group by data_date`”来看近几日的标签量是否存在很大的波动。当发现昨日数据量下降较多时，即很有可能是上游数据加工异常导致的标签问题。

ETL异常问题排查及解决方案

④标签脚本逻辑导致数据加工失败

这种情况也是可能引起错误之一。标签上线前期ETL作业正常，但随着时间的推移，积攒的问题终会爆发出来。这里举一个开发过程中遇到的坑来做详解。

举个例子：我们知道一个用户（userid）可能在多个设备上登录，同样一个设备（cookieid）上可能登录过多个用户，即userid和cookieid为多对多关系。在某次开发需要从cookieid关联到userid，取userid的状态标签时，忽略了这两个维度之间的多对多关系，未加条件限制。初始化标签数据时，脚本执行后跑出“看似合乎逻辑”的数据。但ETL作业调度2天后，这种直接多对多关联的错误逻辑，引起了数据膨胀，作业执行失败。因此，在排查问题时同样要检查开发标签的逻辑是否存在bug问题。

⑤线上业务变动，导致原有标签加工逻辑失效

这种问题虽然不常见，但发生时也会引起标签数据的波动。

例如：通过正则表达式解析网页链接来获取用户访问页面对应的商品品类，在这种场景中，当线上业务变动导致原有的链接改变，而正则表达式是写死的，导致不能解析变动后的链接。

对于这种情况应尽量避免，需要运营方在上线新的链接前通知到标签开发的人员。