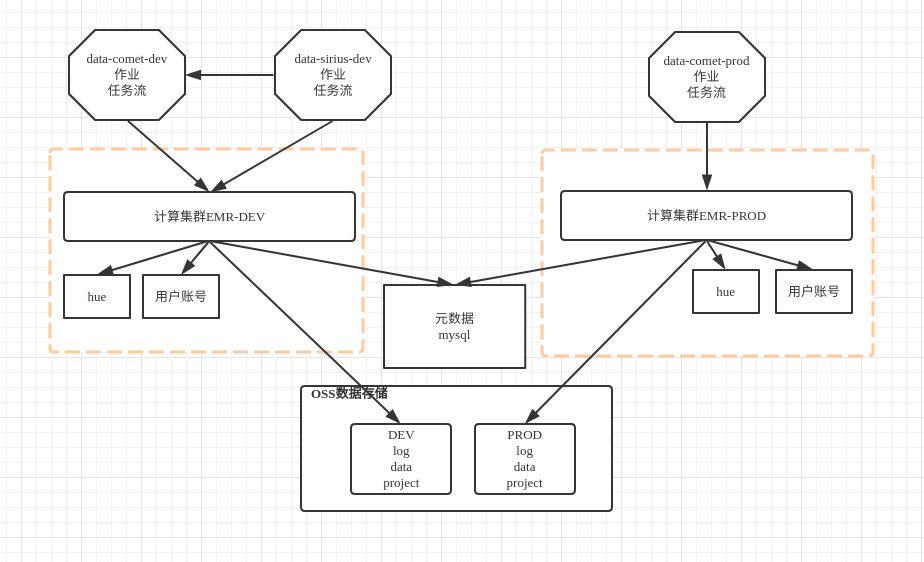
大数据开发指南

## 大数据架构



#### 项目管理

任何项目有dev和prod两个版本，各自独立管理。

项目可以管理作业和任务流。

任务流功能中，不同项目的任务流可以互相感知，能够设置为运行前置条件。

项目命名(projectName)规则：${project}\_${env}

例：

data\_comet\_dev

data\_comet\_prod

项目和大数据计算集群低耦合。

##### 作业

作业存放路径：${projectName}/JOB/${jobDir}

作业命名与项目中的py/sql文件名一致

##### 工作流

工作流存放路径： ${projectName}/FLOW/${jobDir}

工作流命名：flow{seqNO}\_xxx\_yyy

告警配置

**节点失败**和**任务流执行失败**需要配置告警组

问题

1. 工作流怎么管理？

工作流管理作业的聚合和执行。当前直接在aliyun后台配置，没有在项目中留下任何信息。

需要将这些数据固化下来，便于其他人理解，便于工作流迁移

#### 元数据

元数据存放mysql中，开发环境和生产环境分开**。独立管理。**

问题

1. 共用元数据库、共用大数据集群？

1. 元数据库、计算集群，各自独立

优点是互不影响，各自演变。

缺点是迁移集群略显麻烦

2. 元数据库共用、计算集群独立

优点是迁移集群方便

缺点是计算资源无法最大化利用；数据库容易误操作

3. 元数据库共用、计算集群共用

优点是迁移集群方便，计算资源可以最大化利用

缺点是数据库容易误操作；开发任务可能影响到生产任务

最终方案：**元数据库、计算集群，各自独立**

#### OSS数据存储层

负责存放项目代码和数据包，存放于**fanggeek-bigdatas** oss库中。

地址：<https://oss.console.aliyun.com/bucket/oss-cn-shenzhen/fanggeek-bigdatas/object>

* 项目存放位置：**/${env}/project/${projectName}**

例：

/prod/project/data-comet

/dev/project/data-comet

* 原始数据存放位置：**/${env}/mongo/${table}**

例：

/prod/mongo/trident-trident-prod-shops

/prod/mongo/fanggeek-fanggeek-unit

* 数据仓库数据存放位置：**/${env}/dw/${layer}/${table}**

例：

/prod/dw/ods/unit

/prod/dw/ods\_subsection/unit\_children

* 中间结果存放：**/${env}/out-result/${jobClassName}**

例：

/prod/out-result/UserStartupLocationDailyJob

/prod/out-result/ShopIndex

* 用户行为日志存放路径
  + 开发环境：**/user\_ab\_log\_dev**
  + 生产环境：**/user-ab-log**
* 大数据集群任务日志：**/e-mapreduce-test-log**

问题

1. 数据是放入OSS上，还是hdfs上？
   1. 放入oss。集群迁移便利；费用少，安全性高；可能性能不如hdfs
   2. 放入hdfs。性能比oss好；费用高；集群迁移不便

离线计算的存储与计算分离 <https://help.aliyun.com/document_detail/63831.html>

最终方式：**放入OSS，性能略差与HDFS，但是存储量大特便宜**。不与大数据计算集群绑定。

## 全局

### 外部业务数据库规范

1. 字段只增不删不改

表上线后，字段不要重命名、类型不要变。表字段废弃时不要删除

1. 表可以增加和修改，不要删除

表结构变更较大时，建立新表。 表废弃后，不要删除

### SQL 规范

1. 表名、字段名小写，使用字母、数字和下划线\_
2. 逗号’,‘前置，数据查询和表定义
3. 字段引用必须带表名或别名，哪怕只有一张表
4. 字段和表都要有注释，尽可能的详细
5. with as 表名称命名： “ t\_ ”为前缀
6. 创建表时指定数据来源表（bloodsFrom），加入TBLPROPERTIES，为数据血缘提供数据

```

TBLPROPERTIES("bloodsFrom"="ts.agent,ts.order")

```

1. 参数尽可能的从外部传入，有全局时间。尽量不要使用current\_date这类函数
2. 缩进、换行 规范

脚本顶格开始

()内的内容缩进四个字符

分割字段的 , 放在段落前并添加一个空格后再编写

避免使用关键字作为字段名

每一个SELECT查询整体需要另起一行

同一查询中 AS 尽可能对齐（如单个字段处理逻辑复杂导致远长于其他字段可以特殊处理 其他字段保证对齐）

每个查询中的操作符（select/from/where/join/union 等）都需要另起一行编写，并与该查询中其他操作符对齐

查询中 ON AND要另起一行且相比于操作符缩进四个字符

### 作业文件规范

1. 作业文件必须包含注释头，如开发人（@auther），开发时间（@createTime），维护人（@maintainer），维护时间（@maintainTime），维护内容（@maintainContent）

-- @auther tianjp

-- @createTime 2021-04-30 11:39:29

--

-- @maintainer tianjp

-- @maintainTime 2021-04-30 11:39:20

-- @maintainContent 根据规范重新建立agent表

1. 表名和文件名称一致

这个要细化

### 字段命名统一(词根)

1. 地区，国省市区片五级

adcode\_country、adcode\_province、adcode\_city、adcode\_area、adcode\_subarea

name\_country、name\_province、name\_city、name\_area、name\_subarea

1. 时间，日周月季年

date\_day、date\_week、date\_month、date\_season、date\_year

1. 以时间为分区的字段命名为dates
2. 指标名后缀
3. 维度名后缀
4. 常用词 全称 缩写

## 彗星项目（数据仓库）

彗星项目主要职责是数据仓库建设。包括数据抽取、清洗、整理等工作。为后续的业务统计、BI系统提供服务。

项目构建和发布必须使用自动化工具。

### 分层LAYER

总共分为**ods、ods\_subsection(ods\_sub)、ts、dim、dwd、dws、dm、app**等8层。

接下来说明各层的责任和规范。

#### ODS 原始数据层

主要工作

1. 建立外表，与数据源绑定。与业务数据（bson）绑定、日志数据（UserActionLog）绑定
2. 绑定分区数据

分区表建外部表

#### ODS\_SUBSECTION 结构转换

表结构拆分。将嵌套结构拆解为单层结构，将数组拆解为子表。简单数组类型也拆解为子表。

##### 结构转换

**List拆分**

* 遇到List拆分出子表（inline），使用母表的`\_id`去关联
* 保留List中的顺序信息

**Struct拍平**

* 遇到Struct结构把多维结构转化为一维结构（inline 或者 explode）
* 拍平后的字段名需要保持原来的所有意思(如包含原结构的外层的含义)

**废弃字段处理**

* 保留

**表结构变动**

* 只允许这三种变动
  + 新增字段
  + 废弃字段
  + 新增分区

#### TS 数据清洗

仅做单表数据清洗，不做联合多表的数据清洗。

##### 空值处理

总体来说需要将null转换其它值，null用来做过滤、做比较非常不便。

* 重要字段的空值填充默认值。
  + int、float、double等数值类型默认为0
  + string默认为''
  + 枚举类型必须有有效的默认值

##### 单位统一

统一库中所有相同属性的单位

* 租价、售价等价格类字段的单位为元

**脏数据处理（暂不处理，到后期需要的时候再处理）**

* 关键字段丢失的数据视为脏数据（如订单表的金额，但是不同场景情况种类太多，不好统一处理，难度和工作量都较大，缺失情况占比极少）
* 数据与实际情况偏差过大的视为脏数据（根据业务，需要判定标准，但是不同场景情况种类太多，不好统一处理，难度和工作量都较大）
* 数据格式与其他不符的视为脏数据（如数据字段里存在字符串等情况，情况较少）
* 脏数据一并丢弃处理

**重复数据处理**

* 解决部分业务表因并发引起的数据重复问题
* 解决唯一事物存在多个问题（如存在多个相同的小区等）

#### DIM基础维度层

主要有两类数据：基础维度表、数据字典和业务维度表。

基础维度表有时间表、区域表。

数据字典根据各个业务表的情况抽取而来，表示字段所有枚举常量。

业务维度表指的是业务强相关的维度表。维度表属性要尽可能的丰富、准确、开箱即用。

##### 基础维度表

基础维度表存放在job\_dim目录下

1. 区域表 constant\_city.dim.sql
2. 时间表 constant\_date.dim.sql

##### 数据字典

数据字典建设标准，满足2或3时需要建立字典

1. 枚举至少有3个
2. 枚举值在多个地方共用
3. 枚举值未来会变化。如果大概率不变，不用专门建立数据字典，写在表转换sql中即可

数据字典需要加入属性，方便追踪

1. TBLPROPERTIES("refTables"="dwd.dim\_product.type")

refTables表示有那些表字段在使用该字典，有多个表使用逗号分割，必须精确到表字段。

数据字典存放在job\_dim/dict下，命名规则

1. 文件命名：dict\_{表名}.dict.dim.sql。如：dict\_order\_status.dict.dim.sql
2. 表命名：dim.dict\_{表名}。 如：dim.dict\_order\_status
3. 数据存放路径（LOCATION）： oss://fanggeek-bigdatas/${env}/dw/dim/dict/dict\_{表名}。如：oss://fanggeek-bigdatas/${env}/dw/dim/dict/dict\_order\_status

##### 业务维度表

业务维度表存放在job\_dim/biz下。

业务维度表建设标准：

1. 表中的枚举字段，一定要加中文信息。如vip\_level字段需要增加对应的vip\_level\_desc，增加可读性

业务维度表命名规范

1. 文件命名：{表名}.biz.dim.sql。如 agent.biz.dim.sql
2. 表命名：dim.{表名}。如 dim.agent
3. 文件存放路径： oss://fanggeek-bigdatas/${env}/dw/dim/{表名}。 如 oss://fanggeek-bigdatas/${env}/dw/dim/agent

#### DWD明细层

存放最基础的事实表。

事实表粒度最小，尽可能详细。

任务存放于job\_dwd目录之下。事实表根据数据域存放，数据域目录存放在job\_dwd之下。

事实表

1. 事件要单一，粒度要最小，内容要详实
2. 事实表要有视图，方面查看完整的模型

不要视图

事实表命名规范 dwd\_数据域缩写\_表名.sql

1. 文件命名：{数据域缩写}\_fact\_{表名}.{数据域全名}.dwd.sql。如3 ord\_fact\_agent\_order.order\_domain.dwd.sql
2. 表命名：dwd.{数据域缩写}\_fact\_{表名}。如 dwd.ord\_fact\_agent\_order
3. 文件存放路径： oss://fanggeek-bigdatas/${env}/dw/dwd/{表名}。 如 oss://fanggeek-bigdatas/${env}/dw/dwd/order\_domain/ord\_fact\_agent\_order

#### DWS汇总层

汇总层主要存在两种表，模型汇总和分析宽表。

模型汇总是一种轻度汇总，目的共享统计结果，避免其它业务重复统计。

分析宽表，用于专项分析，分析范围不定，数据粒度可大可小。

OK

这里要写详细一点，具体设计，不要用dims ,天周月分开

模型汇总表。主要将常用的维度统计结果保存下来。

表设计规范

1. 需要有dims字段，表示选用了那些维度做聚合
2. 时间字段，这两个字段用于分区
   1. date\_type: 时间类型。天周月季年-DWMSY
   2. dates: 时间，精确到天
3. calc\_time表示数据的计算时间
4. 常用维度纳入汇总，不常用的可以交给后续的kylin做聚合
5. 维度要有维度值和维度描述，方便使用。轻度汇总层属于基建，要做扎实

任务文件设计规范：

1. 文件命名： {主题域前缀}\_stat\_{表名}.{主题域}.dws.sql， 如 ord\_stat\_agent\_order.order\_domain.dws.sql
2. 表命名： dws.{主题域前缀}\_stat\_{表名}。如 dws.ord\_stat\_agent\_order
3. 文件存放路径： oss://fanggeek-bigdatas/${env}/dw/dws/{主题域}/主题域前缀}\_stat\_{表名}。 如 oss://fanggeek-bigdatas/${env}/dw/dws/order\_domain/ord\_stat\_agent\_order
4. 存储格式ORC

建表示例：

CREATE TABLE IF NOT EXISTS `dws`.`ord\_stat\_agent\_order` (

calc\_time timestamp comment '计算时间'

, dims string comment '维度组合'

, payment\_amount\_total bigint comment '总金额'

, adcode\_subarea string

, adcode\_area string

, adcode\_city string

, adcode\_province string

, adcode\_country string

, order\_status string

, product\_type string

)

COMMENT '经纪人个人订单汇总'

PARTITIONED BY (

`date\_type` string comment '时间类型。天周月季年-DWMSY'

, `dates` date

)

STORED AS ORC

LOCATION 'oss://fanggeek-bigdatas/${env}/dw/dws/order\_domain/ord\_stat\_agent\_order'

;

维度组合语法

with select\_datas as (

select

ord.payment\_amount

, ord.agent\_id

, ord.order\_id

, ord.adcode\_subarea

, ord.adcode\_area

, ord.adcode\_city

, ord.adcode\_province

, ord.adcode\_country

, ord.order\_status

, ord.product\_type

, 'D' as date\_type

, ord.`date` as dates

from `dwd`.`ord\_fact\_agent\_order\_view` as ord

where ord.del\_flag\_inner = 0

and ord.`date` = date\_sub(current\_date(), 1)

)

insert overwrite table `dws`.`ord\_stat\_agent\_order` PARTITION(date\_type, dates)

select

current\_timestamp() as calc\_time

, lower(concat\_ws(','

, if(grouping(date\_type) = 0, 'date\_type', null)

, if(grouping(dates) = 0, 'dates', null)

, if(grouping(adcode\_country) = 0, 'adcode\_country', null)

, if(grouping(adcode\_province) = 0, 'adcode\_province', null)

, if(grouping(adcode\_city) = 0, 'adcode\_city', null)

, if(grouping(adcode\_area) = 0, 'adcode\_area', null)

, if(grouping(adcode\_subarea) = 0, 'adcode\_subarea', null)

, if(grouping(order\_status) = 0, 'order\_status', null)

, if(grouping(product\_type) = 0, 'product\_type', null)

)) as dims

, sum(ord.payment\_amount) as payment\_amount\_total

, count(distinct ord.agent\_id) as agent\_total

, count(distinct ord.order\_id) as order\_total

, ord.adcode\_subarea

, ord.adcode\_area

, ord.adcode\_city

, ord.adcode\_province

, ord.adcode\_country

, ord.order\_status

, ord.product\_type

, ord.date\_type

, ord.dates

from select\_datas as ord

group by

adcode\_subarea

, adcode\_area

, adcode\_city

, adcode\_province

, adcode\_country

, order\_status

, product\_type

, date\_type

, dates

grouping sets (

(date\_type, dates, adcode\_city, order\_status, product\_type)

, (date\_type, dates, adcode\_city, order\_status)

, (date\_type, dates, adcode\_city, product\_type)

)

;

宽表建设规范

待续......

#### DM 数据集市

数据集市（Data marts）是为满足已定义的用户组或业务领域（Business Area）对于特定业务信息的需求而创建的。它们比数据仓库更小且更关注在数据中构建复杂的业务规则来支持功能强大的分析。

根据不同的需求，选取数据，组建各自的分析主题，获取数据组合后的新数据。各个主题之间相对独立，互不影响。

像房源鉴真（金星）、企业房源标准库、经纪人评分、小区推荐等重要业务就可以在集市层实现。

Dm全部改成hive任务，app层去做埋点定制化处理

#### APP应用层

根据应用需要将数据汇总，可供应用直接使用，不必做任何转化。

该数据存储于hive和mysql中。hive长期存储，mysql大概存一个月，具体时间跟业务商讨。

#### TMP临时层

存放临时表，会定期删除。临时表根据业务增加前缀，便于管理。

### 工作流

工作流负责将任务关联、定义执行顺序，并管理执行时间、前置依赖以及事件告警等事务。

**规范**

1. 命名：flow{number}\_{name}。变量number从1开始取值，每个目录的number独立。变量name，根据实际含义命名，做到意义明确，必须是英文、数字、下划线
2. 部署
   1. 有前置依赖工作流，需要单独配置；否则通过执行时间，先后执行

工作流执行顺序：

1. ods
2. ods\_sub
3. ts
4. dim
5. dwd-维度表
6. dwd-事实表
7. dws-轻度汇总表
8. dws-主题宽表
9. dm
10. app

### 开发工具

主要使用**HiveSQL**，Spark/SparkSQL辅助。

**python2.7**作为主要语言，负责spark任务开发。

### 开发规范

#### 项目目录

* 任务目录

1. job\_ods
   1. spark：存放spark任务。主要是通过sql操作hive数据。
   2. sql：存放sql语句
      1. create：存放表创建sql语句。命名：agent\_create.ods.sql
2. job\_ods\_subsection
   1. spark：同上
   2. sql：
      1. create：同上
      2. load：存放数据加载sql语句。主表和子表存放在一个sql文件中，但是要注意先后顺序。命名：agent\_load.ods\_subsection.sql
3. job\_ts
   1. spark：同上
   2. sql：同上
      1. create：同上
      2. load：存放数据清洗sql语句。主表和子表存放在一个sql文件中，但是要注意先后顺序。只需要清洗枚举字段，将null设为默认值。命名：agent\_laod.ts.sql
4. job\_dwd
   1. dim\_biz：业务维度表
   2. 数据域目录
5. job\_dws
   1. 数据域目录
   2. wide\_table： 业务宽表
6. job\_dm
   1. {x}：主题域（主题组）
      1. {y}：主题。
7. job\_app
8. job\_dim

* 其它目录

1. schema\_json\_folder/for\_maintenance/

存放所有的业务表原始结构数据

任务执行流程

ods， ods\_sub， ts， dim， dwd-维度表， dwd-事实表， dws-模型汇总表，dws-宽表，dws-宽表汇总表，dm，app

#### 数据存放路径

见oss数据存储层

#### 业务表schema规范

1. meta元数据

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **类型** | **取值范围** | **含义** |
| type | string | v1：属性类型需要反推  v2：属性类型直接获取 | 数据格式类型 |
| version | int |  | 版本。schema每更改一次，版本号加1 |
| dbhost | string | fanggeek  trident  mercury | 表所属的数据库 |
| importType | string | increment：增量  full：全量 | 数据导入方式 |
| partition | string[] |  | 表分区字段。  例：["createTime", "cityInfo.adcodeCity"], |
| locationMain | string |  | 主表数据存放路径，支持oss |
| locationSub | string |  | 表拆分后的数据存放路径，支持oss |

1. struct数据结构

包含表所有属性

例如

```json

{

"meta": {

"type": "v1|v2",

"version": 1,

"dbhost": "fanggeek|trident|mercury",

"importType": "increment|full|...",

"partition": ["createTime", "cityInfo.adcodeCity"],

"locationMain": "oss://fanggeek-bigdatas/{env}/mongo/fanggeek-fanggeek-orders",

"locationSub": "oss://fanggeek-bigdatas/{env}/{db: ods|ods\_stage}/base\_da\_orders"

},

"stuct": {

"\_id": "string",

"modifyTime": "datetime",

"createTime": "datetime",

"del\_flag\_inner": "int"

}

}

```

#### hive sql开发注意事项

1. sql需要有注释，将功能、参数等信息说明清楚，字段注释不强制要求。
2. sql需要使用参数，来适配开发和生产环境的不同
3. 表注释（COMMENT）、表属性（TBLPROPERTIES）

blood\_relation\_ship：血缘。用逗号分割多个来源

1. 如何在EMR中运行？

在emr中建立hive作业，然后键入命令：

```

-d env=dev -d env2=dev2 -f ossref://fanggeek-bigdatas/dev/project/data-comet/job\_test/job1\_user\_ab\_log\_simple.sql

```

例：

```

*/\*\**

*建立user-ab-log*

*参数*

*dataDir:*

*user-ab-log: 生产环境*

*user\_ab\_log\_dev: 开发环境*

*\*/*

**CREATE EXTERNAL TABLE** oss.base\_user\_ab\_log (

`\_\_time\_\_` BIGINT **comment '日志上报时间'**,

`\_\_topic\_\_` ***INT* comment '主题. 如ios, 安卓, 后台等'**,

`\_\_source\_\_` ***INT* comment '来源. 如小鹿选房, 看房日记等'**,

c BIGINT **comment '城市'**,

t ***INT*** ,

st ***INT***,

f ***INT* comment 'flag. 取值:0或1'**,

v BIGINT **comment 'value. 数值'** ,

**id** STRING **comment '主ID'**,

ip STRING **comment 'ipV4'**,

i string **comment 'info. dict类型, 包含大量业务数据'**

)

comment "表中文含义"

PARTITIONED **BY** (dates ***Date***)

stored **AS** parquet

location **'oss://fanggeek-bigdatas/${dataDir}/'**

TBLPROPERTIES(

"blood\_relation\_ship"="oss.table1, oss.table2",

)

```

### 用户行为记录表（埋点）user-ab-log

1. ods存放原始数据。**ods.base\_user\_ab\_log**
2. ods\_sub将原始数据拆分，分区存放，并将时间按照天的范围对齐。**ods\_subsection.base\_user\_ab\_log**

通过dates、source、topic来分区。

1. ts将 根据分区数据，将记录表简化。**ts.user\_ab\_log\_simple**

通过dates、source、topic来分区。

## 流星项目（实时计算）

### 实时任务开发注意事项

一个实时任务需要有以下考虑点：

0. 任务在故障发生之后保证那种类型的一致性

- at-most-once：计数结果可能丢失

- at-least-once：发生故障后可能多算，但是绝不会少算

- exactly-once：发生故障后得到的计数结果与正确值一致

1. 如何保证端到端的一致性。涉及状态、时间、checkpoint

2. 任务或异常、或重启、或更新，如何快速恢复，追赶进度，还要保证一致性。涉及状态、checkpoint

3. 任务异常要及时知晓。涉及监控和告警

#### 时间

1. 一旦使用event-time，任何地方都要拿到watermark来做时间比较，千万不要使用系统时间

2. 迟到的数据不要丢弃，要尽可能的处理。

// 使用事件时间

env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

#### 状态

1. 想要保证端到端的一致性，需要将状态放入flink中，谨慎使用外部存储（一致性、性能都难以控制）。输出一定要满足幂等性，两阶段提交2PC会有意向不到的错误，处理比较麻烦

2. 状态算子必须人工制定ID，全局唯一

3. 状态需要考虑失效后处理方式

4. 需要考虑状态数据高效序列化，尤其是泛型

#### Checkpoint检查点

1. 必须开启检查点，配置检查频率，检查点执行最大时长，同时生成检查点的数量，以及检查点与检查点的时间间隔

2. 需要注意状态特别大的情况下，检查点的执行时间可加长，生产环境需要注意

3. 任务停止后，checkpoint不要删除，多保存几份，最好是保留20份

//检查点，每10分钟保存一次checkpoint

env.enableCheckpointing(10 \* 60 \* 100)

//设置模式为exactly\_once（默认值）

env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY\_ONCE)

// 确保检查点之间有至少1000ms的间隔（ck最小间隔）

env.getCheckpointConfig.setMinPauseBetweenCheckpoints(1000)

// 检查点必须在9分钟内完成，或者被丢弃（ck超时时间）

env.getCheckpointConfig.setCheckpointTimeout(9 \* 60 \* 100)

//同一时间只允许进行一个检查点

env.getCheckpointConfig.setMaxConcurrentCheckpoints(1)

// flink程序被cancel后，会保留ck数据

env.getCheckpointConfig.enableExternalizedCheckpoints(ExternalizedCheckpointCleanup.RETAIN\_ON\_CANCELLATION)

// 设置状态后端

env.setStateBackend(new FsStateBackend("hdfs://VisitorHighIntentionPriceStreaming/"))

#### 异常和容错

1. 配置重启策略，固定重启三次、一次。默认重启次数是int的最大值，相当于无限重启。

env.setRestartStrategy(RestartStrategies.fixedDelayRestart(1, 10 \* 1000))

1. 增加告警埋点，任务退出时可以上报埋点

EnvironmentSetting.setJobListenter(env, config, XXXJob.getClass.getName)

#### 恢复

1. 需要考虑任务异常退出后快速恢，赶上最新的消费进度

#### 其它

1. 算子从外部读取数据时，要注意数据的大小和时长。时间太长会引起task-manager心跳失败。数据太大会引起full-gc，甚至是OOM

## 作业和流程部署自动化（数仓、天狼星）

### 背景

大数据开发离不开任务和流程。当前任务和流程需要手工配置，在开发和生产环境各配置一遍，比较麻烦，容易引入人工错误。

现在通过json来描述任务、流程，包括任务的特殊配置，流程的顺序依赖等，统统纳入管理，只需要将json配置完成，各个环境一键发布。以此来提高工作效率。

### 概念

* project：项目。负责
* cluster：大数据集群。负责数据运算
* job：作业。指的是一项具体的工作
* flow：工作流。根据作业分类，将其串联、并联完成一批工作

### 规范

1. 命名只可以使用这几类：英文、数字、中划线、下划线
2. 名称必须全局（**项目内**）唯一。包括并不局限于：项目名称（projectName）、大数据集群名称（clusterName）、任务名称（jobName）、流程名称（flowName）

### JSON配置

#### 配置详情

配置分为三个部分：props、job、flow

props负责全局变量、环境变量。其中全局变量可以引用环境变量。

job负责任务描述。包括目录，每个任务的详细配置等。

flow负责流程描述。包括目录，每个流程的详细配置等。flow可以以任务名称来引用job任务。流程依赖可以依赖其它项目流程。

##### props变量

common：全局变量。可以引用环境变量。

dev：开发环境变量。

prod：生产环境变量。

**重要：**{env}为全局变量，在初始化时传入，取值：dev、prod

|  |
| --- |
| **"props"**: {  *// 公共变量。公共变量可引用环境变量*  **"common"**: {  **"regionId"**: **"cn-shenzhen"**,  **"projectOss"**: **"ossref://fanggeek-bigdatas/{env}/project/data-comet"**  },  *// dev环境特有变量*  **"dev"**: {  **"projectName"**: **"auto-deploy-test"**,  **"clusterName"**: **"fanggeek-bigdata-dev"**  },  *// prod环境特有变量*  **"prod"**: {  **"projectName"**: **""**,  **"clusterName"**: **""**  }  } |

##### job作业

负责作业配置

dirs：目录管理。顺序非常重要，必须是从外层到内层。文件夹以"/"开头，表示根目录

jobs：作业管理。

第一层是目录。可生成参数：dir

第二层是作业名称，也是文件名，名称全局唯一，不得冲突。可生成参数：jobName

每一个作业都可以独立配置。参数如下：

desc：必选。作业描述

type：必选。作业类型。目前有：**HIVE、SHELL、SPARK**

command：作业的执行命令。可以使用变量来维护，可大大减少工作量

props：该作业的参数。common不支持参数引用

others：作业的额外配置。指的是阿里云特有的参数，使用时透传，不做任何额外处理

|  |
| --- |
| *// 任务配置*  **"job"**: {  *// 任务目录，顺序非常重要，按照次序编写*  **"dirs"**: [  **"/tianjp\_job\_test"**,  **"/tianjp\_job\_test/load"**  ],  **"jobs"**: {  *// 目录。此目录必须在目录列表中存在。该目录的环境变量是{dir}*  **"/tianjp\_job\_test/load"**: {  *// 任务名称。全局唯一，任务命名、流程命名。该任务环境变量{jobName}*  **"user\_ab\_log\_create.ods.sql"**: {  *// 任务描述。必选*  **"desc"**: **"xx"**,  *// 任务类型。必须。HIVE | SHELL | SPARK*  **"type"**: **"HIVE"**,  *// 任务命令。可使用变量, 包括任务变量、任务名、目录名、全局变量*  **"command"**: **"-d env={env} -d dataDir={dataDir} -f {projectOss}/{dir}/{jobName}"**,  *// 任务变量*  **"props"**: {  *// 任务变量*  **"common"**: {  **"table"**: **"agent"**  },  *// 任务在dev环境下的取值*  **"dev"**: {  **"dataDir"**: **"user\_ab\_log\_dev"**  },  *// 任务在prod环境下的取值*  **"prod"**: {  **"dataDir"**: **"user-ab-log"**  }  },  *// 任务参数。原始参数，直接上报，不做二次处理*  **"others"**: {  **"FailAct"**: **"CONTINUE"**,  **"MaxRetry"**: 3  }  },  **"agent\_create.ods.sql"**: {  **"desc"**: **"创建user-ab-log"**,  **"type"**: **"SHELL"**,  **"command"**: **"env={env}\ntable={table}\n\n{biz\_data\_extract\_command}"**,  **"props"**: {  **"common"**: {  **"table"**: **"agent"**  }  },  **"others"**: {  **"FailAct"**: **"CONTINUE"**,  **"MaxRetry"**: 3  }  }  }  }  }, |

##### flow工作流

流程管理，核心是作业串联、流程依赖、定时调度

dirs：规则与job下dirs一致

flows：流程管理

第一层是目录

第二层是流程名称。必须全局唯一

流程配置参数：

enable：必选。是否开启。1-开启；0-关闭。关闭状态时**定时（cron）**、**工作流依赖（dependences）**配置无效

cron：选填。定时。时间表达式：秒-分-时-日-月-星期

desc：必选。描述

dependences：流程依赖

project：依赖的项目名称。没有时表示本项目

flow：依赖的流程，使用流程名称表示

alertUG：通知组ID，用于告警。执行失败、节点失败时告警

edges：job管理。job按顺序连接。必须包含START、END节点，使用任务名代表任务（也这是为什么命名要全局唯一）。当遇到连线没有闭合、跳线时会报edge错误，需谨慎处理，必要时自己动手画图。此处有些考验想象力。

others：作业的额外配置。指的是阿里云特有的参数，使用时透传，不做任何额外处理

props：该工作流的参数。目前仅有dev、prod两类变量，无common类变量。目的是工作流的配置在不同环境中独立，互不影响。

|  |
| --- |
| *// 工作流配置*  **"flow"**: {  *// 目录，顺序非常重要，按照次序编写*  **"dirs"**: [  **"/dir1"**,  **"/dir1/subDir2"**  ],  **"flows"**: {  *// 目录。此目录必须在目录列表中存在*  **"/dir1/subDir2"**: {  *// 任务名称。全局唯一，任务命名、流程命名*  **"flow1\_ods\_create\_tables"**: {  *// 是否开启。1=开启；0=关闭；*  **"enable"**: 1,  *// 定时调度。格式：秒-分-时-年-月-星期*  **"cron"**: **""**,  *// 描述。必选*  **"desc"**: **"flow1\_ods\_create\_tables desc"**,  *// 依赖工作流。可依赖其它项目和本项目工作流。没有project参数，表示依赖本项目的工作流*  **"dependences"**: [  {  **"project"**: **"data-comet"**,  **"flows"**: [  **"flow1"**  ]  }  ],  *// 告警配置*  **"alertUG"**: **"AUG-1fd01234c5434e618d871a1db1f9c4ce"**,  *// 任务运转流程*  **"edges"**: [  [  **"START"**,  **"user\_ab\_log\_create.ods.sql"**,  **"END"**  ],  [  **"START"**,  **"user\_ab\_log\_create2.ods.sql"**,  **"user\_ab\_log\_create3.ods.sql"**,  **"END"**  ]  ],  **"others"**: {  },  **"props"**: {  "dev": {  "cron": "0 0 0 \* \* \*"  },  "prod": {  "cron": "0 5 0 \* \* \*"  }  }  }  }  }  } |

#### 部署策略

删除所有的流程、任务、目录，在从新部署目录、任务和流程。

此策略有缺点，流程和任务历史执行日志丢失。

### 示例

|  |
| --- |
| {  *// 全局变量*  **"props"**: {  *// 公共变量。公共变量可引用环境变量*  **"common"**: {  **"regionId"**: **"cn-shenzhen"**,  **"projectOss"**: **"ossref://fanggeek-bigdatas/{env}/project/data-comet"**  },  *// dev环境特有变量*  **"dev"**: {  **"projectName"**: **"auto-deploy-test"**,  **"clusterName"**: **"fanggeek-bigdata-dev"**  },  *// prod环境特有变量*  **"prod"**: {  **"projectName"**: **""**,  **"clusterName"**: **""**  }  },  *// 任务配置*  **"job"**: {  *// 任务目录，顺序非常重要，按照次序编写*  **"dirs"**: [  **"/tianjp\_job\_test"**,  **"/tianjp\_job\_test/load"**  ],  **"jobs"**: {  *// 目录。此目录必须在目录列表中存在。该目录的环境变量是{dir}*  **"/tianjp\_job\_test/load"**: {  *// 任务名称。全局唯一，任务命名、流程命名。该任务环境变量{jobName}*  **"user\_ab\_log\_create.ods.sql"**: {  *// 任务描述。必选*  **"desc"**: **"xx"**,  *// 任务类型。必须。HIVE | SHELL | SPARK*  **"type"**: **"HIVE"**,  *// 任务命令。可使用变量, 包括任务变量、任务名、目录名、全局变量*  **"command"**: **"-d env={env} -d dataDir={dataDir} -f {projectOss}/{dir}/{jobName}"**,  *// 任务变量*  **"props"**: {  *// 任务变量*  **"common"**: {  **"table"**: **"agent"**  },  *// 任务在dev环境下的取值*  **"dev"**: {  **"dataDir"**: **"user\_ab\_log\_dev"**  },  *// 任务在prod环境下的取值*  **"prod"**: {  **"dataDir"**: **"user-ab-log"**  }  },  *// 任务参数。原始参数，直接上报，不做二次处理*  **"others"**: {  **"FailAct"**: **"CONTINUE"**,  **"MaxRetry"**: 3  }  },  **"agent\_create.ods.sql"**: {  **"desc"**: **"创建user-ab-log"**,  **"type"**: **"SHELL"**,  **"command"**: **"env={env}\ntable={table}\n\n{biz\_data\_extract\_command}"**,  **"props"**: {  **"common"**: {  **"table"**: **"agent"**  }  },  **"others"**: {  **"FailAct"**: **"CONTINUE"**,  **"MaxRetry"**: 3  }  }  }  }  },  *// 工作流配置*  **"flow"**: {  *// 目录，顺序非常重要，按照次序编写*  **"dirs"**: [  **"/dir1"**,  **"/dir1/subDir2"**  ],  **"flows"**: {  *// 目录。此目录必须在目录列表中存在*  **"/dir1/subDir2"**: {  *// 任务名称。全局唯一，任务命名、流程命名*  **"flow1\_ods\_create\_tables"**: {  *// 是否开启。1=开启；0=关闭；*  **"enable"**: 1,  *// 定时调度。格式：秒-分-时-年-月-星期*  **"cron"**: **""**,  *// 描述。必选*  **"desc"**: **"flow1\_ods\_create\_tables desc"**,  *// 依赖工作流。可依赖其它项目和本项目工作流。没有project参数，表示依赖本项目的工作流*  **"dependences"**: [  {  **"project"**: **"data-comet"**,  **"flows"**: [  **"flow1"**  ]  }  ],  *// 告警配置*  **"alertUG"**: **"AUG-1fd01234c5434e618d871a1db1f9c4ce"**,  *// 任务运转流程*  **"edges"**: [  [  **"START"**,  **"user\_ab\_log\_create.ods.sql"**,  **"END"**  ],  [  **"START"**,  **"user\_ab\_log\_create2.ods.sql"**,  **"user\_ab\_log\_create3.ods.sql"**,  **"END"**  ]  ],  **"others"**: {  }  }  }  }  }  } |