Kstream接口说明

# 注意点

*1,下文所列内容注意区分大小写且属性名不可更改;*

*2,同一topic的operation是有顺序的,具体如何操作下文阐述;*

*3,页面属性中文名称可参考注释(第一个逗号之前);*

*4,‘运行’和‘启动中’时不可修改，‘未部署’状态不可启动，‘启动中’不可再‘启动’，页面请做限制；*

# Response

成功的格式：{"success":true,【请求返回数据】}

失败的格式：{"success":false,"error":"error msg"}

故下文仅列出成功格式。

# 操作顺序

*无需关注不同input的顺序交叉,只需关注整个服务的操作顺序；*

*操作顺序接口基于已经保存好具体的操作数据。*

Method:post

Url:http://ip:port/**kstream/orderApp**

Data:{“service\_id”:””,“order”:”id1,id2,id3...”}

Response:{"success":true}

# 保存信息

*对应功能：创建，修改*

#################################################

Method:post

Url:http://ip:port/**kstream/storeApp**

Data:{“service\_id”:””,“type”:”下文定义”,”value”:”下文定义”}

Response:下文定义

#################################################

Type:[ main,input, operation, output ]

## 1,type是main时：

创建服务，修改服务。创建服务时service\_id为空。

Response:{"success":true,“id”:””}

value:{

”service\_name”:””, # 服务名称

“service\_desc”:””, # 服务描述

"bootstrap\_servers": "", # kafka地址，url,必须配置

###########高级###########

"ks\_zookeeper\_url": "", # zk地址,可配置

"auto\_offset\_reset": "latest", #位置，页面显示起始,最新[earliest,latest]必须配置

"cache\_max\_bytes\_buffering": 10485760, #缓存，long,必须配置

"num\_stream\_threads": 1, #线程，int,必须配置

"commit\_interval\_ms": 30000, #提交周期，long,必须配置

"buffered\_records\_per\_partition": 1000 #缓存记录数，long,必须配置

###########下方属性是filter,mapper操作的字典###########

“dic\_fields”:”” # 列名，dic映射的字段名,f1,f2,f3...

“dic\_type”:”” #类型, kafka(内容在kafka里),array(内容简单的数组列表)，选择一个类型对应下方显示，kafka还是array数组

“dic\_kafka\_topics”:”” #主题， 存储对应file内容的topic,t1,t2,t3...

“dic\_kafka\_address”:”” #地址， 存储dic topic的kafka地址，不配置使用上方的bootstrap\_servers

“dic\_array\_values”:””

#分组值， 存储type是array的多组值,如：f11,f12,f13...;f21,f22,f23... 分号标识分组

}

## 2,type是input时：

Response:{"success":true,“id”:””}

value:{

“input\_id”:”” # 创建的时候为空

"ks\_topics": "", # 主题,必须配置

"ks\_type": "stream", # 流类型，stream,table 必须配置

###########高级###########

"ks\_time\_type": "long", #时间类型， 时间字段值类型long,string

"ks\_time\_name": "" #时间列名,默认kafka record的timestamp

"ks\_time\_lang": "en", # 时间本地化,默认en[en,zh,fr....]

"ks\_time\_format": "", # 时间格式，时间值类型string时需要配置字符串格式

"ks\_time\_offsetId": "+08:00", # 时区，默认东八区(+08:00)

}

## 3,type是operation时：

下文的operation\_operator是页面中的select模块:

convertKV,convertTime,window,join,leftJoin,outerJoin,filter,mapper

同一input的operation之间是有顺序关系的，在每个具体操作保存后会返回一个operation\_id(可作为页面列表的id)在所有操作顺序都调整完成后下一步的时候，请求[操作顺序接口](#_操作顺序)保存操作顺序。

#################################################

Response:{"success":true,”id”:””}

### 2.1, convertKV

value:{

“operation\_id”:”” # 创建的时候为空

"operation\_operator": "convertKV", # 键值重组

"operation\_ks\_name": "" # 操作源，执行convertKV操作的topic 必须配置

"operation\_name": "", # 操作名

"kv\_value\_fields": "", #新值列， 新value的组成字段,key与value至少配置一项

"kv\_key\_fields": "", #新键列， 新key的组成字段,key与value至少配置一项

"kv\_key\_fields\_type": "json", #新键格式，[ json,value ]

"kv\_fields\_noExist\_append": false, # 补空值，如果field不存在追加"null",[true,false]

}

### 2.2,convertTime

value:{

“operation\_id”:”” # 创建的时候为空

"time\_error\_out": "", #异常输出，未配置则会中断运行

"time\_in\_formats": "", # 输入格式，long值无需配置format.long,string同时存在format对应的long要添加半角逗号.如[long,string]==>[,uuuu-MM-dd HH:mm:ss.SSS]

"time\_in\_offsetId": "+08:00", # 输入时区

"time\_out\_lang": "en", # 输出本地化

"time\_in\_lang": "en", # 输入本地化

"operation\_name": "", # 操作名

"time\_out\_offsetId": "+08:00", # 输出时区

"time\_out\_formats": "", # 输出格式，与输入字段对应.如:uuuu-MM-dd，如果只配置单一值,则统一输出此格式;如果不配置则输出unix时间戳。

"time\_in\_value\_types": "", #输入值类型，long(unix时间戳),string.与输入字段对应. 必须配置，如果string则配置format

"time\_out\_names": "", # 输出列，如:o1,o2,o3...与输入字段对应.不配置即默认输入字段

"time\_in\_names": "", # 输入列，如:f1,f2,f3... 必须配置

"operation\_operator": "convertTime", # 时间转换

"operation\_ks\_name": "" # 操作源，执行convertTime操作的topic 必须配置

}

### 2.3,window

value:{

“operation\_id”:”” # 创建的时候为空

"window\_sizeMs": 600\_000, # 窗口大小，必须配置

"operation\_name": "", # 操作名

"window\_endTime": "", # 结束时间列

"window\_startTime": "", # 开始时间

"window\_count": "", # 统计列，必须配置

" window\_store\_name": "", #窗口名，kafka内部主题名称

"operation\_operator": "window", # 窗口聚合

"window\_advanceMs": "", # 窗口间隔，默认window size

"operation\_ks\_name": "" #操作名， 执行window操作的topic 必须配置

###########高级###########

"window\_uncover\_fields": "", # 不覆盖列，(int自动求和,string尾部追加).如:f1,f2,f3...

"window\_uncover\_field\_prefix": "\_\_", # 不覆盖列前缀

"window\_uncover\_field\_interval": ",", # 不覆盖列间隔符.默认','

"window\_uncover\_field\_repeat": false, # 不覆盖列重复,对于int数据无效[false,true]

}

### 2.4,join[leftJoin,outerJoin]

支持如下：

#### KTable [join,leftJoin,outerJoin] KTable

#### KStream [join,leftJoin,outerJoin] KStream

#### KStream [join,leftJoin] KTable

value:{

“operation\_id”:”” # 创建的时候为空

# 如果join源两者有一者为table,则before,after,retention无需配置

"join\_afterMs": 0, # time after

"join\_beforeMs": 0, # time before

"join\_retentionMs": 1, # time retention， > before+after

"join\_output\_strategy": "cover", # 输出方式,cover:覆盖;uncover:后缀添加数字,如f=>f1,f1=>f11...

"operation\_name": "", # 操作名

"operation\_table\_store": "", # 临时结果， table-table 需要配置,

"join\_ks\_name": "", # 目标源，执行join操作的目标topic名称，必须配置

"operation\_operator": "", # join[leftJoin,outerJoin]

"operation\_ks\_name": "" # 操作源，执行join操作的topic 必须配置

###########高级###########

"join\_output\_fields\_value\_add": "", #不覆盖列，输出策略外的允许同名字段值追加,优先级高于输出策略,半角逗号分隔

"join\_output\_fields\_value\_add\_interval": ",", # 不覆盖列间隔符，默认','

"join\_target\_through": "", # 目标周转， target的partition与source不一致,运行前手动创建一个与source相同partition的topic,默认分区方式

"join\_source\_through": "", # 源周转，source的partition与target不一致,运行前手动创建一个与target相同partition的topic,默认分区方式

}

### 2.5,filter

value:{

“operation\_id”:”” # 创建的时候为空

"operation\_name": "", # 操作名

"operation\_operator": "filter", # 过滤

"operation\_ks\_name": "", # 操作源，执行filter操作的topic 必须配置

"filter\_conds": "", # 过滤条件，(in,notIn).如:in,notIn...和过滤字段对应 必须配置

"filter\_fields": "", # 过滤列，如f1,f2 必须配置

}

### 2.6,mapper

value:{

“operation\_id”:”” # 创建的时候为空

"operation\_name": "", # 操作名

"operation\_operator": "mapper", # 映射

"operation\_ks\_name": "", #操作源， 执行mapper操作的topic 必须配置

"mapper\_conds": "", # 映射条件，(in,notIn)如:in,notIn,in;notIn,in...分号标识分组,对应下面的appends

"mapper\_fields": "", # 映射列，如:k1,k2,k3;j1,j2....分号标识分组,对应下面的appends

“mapper\_appends”:”” # 添加键值对，输出value中添加满足条件的KV对.如:k1:v1;k2:v2...

}

## 4,type是output时：

Response:{"success":true}

value:{

"output\_ks\_name": "", # 输出源，输出完成计算的topic，必须配置

"output\_fields": "", # 输出列，如:f1,f2,f3不配置默认所有

"output\_targets": "", # 输出目标，结果数据输出[sysout,zbus,kafka]，支持多个目标

"output\_target\_zbus\_mq": "", #zubs队列， 输出的zbus队列名称

"output\_target\_zbus\_address": "", # zbus地址

"output\_target\_kafka\_topic": "", # kafka主题

"output\_target\_kafka\_address": "", # kafka地址，不配置使用计算的kafka地

址

“output\_fields\_noExist\_append”:false #补空值，如果提供的field不存在是否追加"null",默认false [true,false]

###########高级###########

"format\_enable": false #自定义输出

"format\_pattern": "", # 自定义格式，变量前后加上$且变量名不能含有双引号如:{"columns":["数量"],"index":["$window\_start$"]}

"expandWin\_enable": false, #聚合， 值true,false

"expandWin\_executorPeriod": , #聚合周期， 检查expandWin操作是否越界的调度时间，单位毫秒

"expandWin\_expireTime": "", # 聚合大小，expandWin时间大小，单位秒

"expandWin\_background\_threads": 1, #聚合线程， expandWin操作的后台线程数

"expandWin\_countFiled": "", # 聚合列，expandWin操作统计的字段名称

“expandWin\_store\_name”:”” # 聚合名称，expandWin操作的内存唯一标识

}

# 删除信息

Method:post

Url:http://ip:port/**kstream/deleteApp**

Data:{“service\_id”:””,“type”:”下文定义”,”type\_id”:”下文定义”}

Response:{"success":true}

#################################################

Type:[ main,input, operation, output ]

## 1,type是main时

删除整个服务，type\_id为空或者service\_id

## 2,type是input时

删除具体的输入，type\_id为input\_id

## 3,type是operation时

删除具体的操作，type\_id为operation\_id

## 4,type是output时

删除输出，type\_id为空或者service\_id

# 部署服务

Method:post

Url:http://ip:port/**kstream/deployApp**

Data:{“service\_id”:””}

Response:{"success":true}

# 启动服务

Method:post

Url:http://ip:port/**kstream/startApp**

Data:{“service\_id”:””}

Response:{"success":true}

# 停止服务

Method:post

Url:http://ip:port/**kstream/stopApp**

Data:{“service\_id”:””}

Response:{"success":true}

# 获取配置信息

Method:post

Url:http://ip:port/**kstream/getApp**

Data:{“service\_id”:””,“type”:”下文定义”}

Response:{"success":true,”results”:”下文定义”}

#################################################

Type:[main,input,operation,output]

## 1,type是main时：

获取的是service的信息。results:{obj}，obj即为[保存信息的type是main](#_1,type是main时：)保存的信息(value)。

## 2,type是input时：

获取的是输入源信息。results: [{obj},{obj},{obj}...]，现阶段而言最多就两个obj(join操作),obj即为[保存信息的type是input](#_1,type是input时：)保存的信息(value)。

## 3,type是operation时:

获取的是操作信息。results: [{obj},{obj},{obj}...],obj即为[保存信息的type是operation](#_2,type是operation时：)保存的信息(value)。results中的数据会依照保存的顺序。

## 4,type是output时：

获取的是输出信息。results: {obj},obj即为[保存信息的type是output](#_3,type是output时：)保存的信息(value)。

# 获取运行信息

Method:post

Url:http://ip:port/**kstream/getAppSys**

Data:{“service\_id”:””}

Response:{"success":true,”results”:”下文定义”}

#################################################

results: {obj}

obj: {

"service\_id": "" # 服务id

"service\_name": "", # 服务名

“service\_desc”:””, # 服务描述

"service\_cpu": "", # cpu使用率

"service\_time": "", #运行时间

"service\_status": "", # run,odd,stop,start,init(运行,启动异常,停止,启动中,未部署)

"service\_mem": "", #内存使用率

}

# 获取所有运行信息

Method:get

Url:http://ip:port/**kstream/getAllAppSys**

Response:{"success":true,”results”:”下文定义”}

#################################################

results: [{obj},{obj},{obj}...]，obj即为[获取运行信息](#_获取服务运行信息)的单个信息.

# 附录

## Mysql表

CREATE TABLE `ksapp` (

`service\_id` varchar(32) NOT NULL COMMENT '服务id',

`service\_name` varchar(100) NOT NULL COMMENT '服务名称',

`service\_desc` varchar(100) DEFAULT NULL COMMENT '服务描述',

`service\_status` varchar(6) NOT NULL DEFAULT 'init' COMMENT '服务状态run, odd, stop, start,init(运行,启动异常,停止,启动中,未部署)',

`main\_json` varchar(1000) NOT NULL COMMENT 'kstream的main配置',

`operation\_order` varchar(150) DEFAULT NULL COMMENT '操作的顺序',

PRIMARY KEY (`service\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `ksinput` (

`service\_id` varchar(32) NOT NULL COMMENT '服务id',

`input\_id` varchar(15) NOT NULL COMMENT 'input id',

`input\_json` varchar(500) NOT NULL COMMENT 'kstream的source配置',

KEY `input\_index` (`service\_id`,`input\_id`),

CONSTRAINT `fk\_ksinput` FOREIGN KEY (`service\_id`) REFERENCES `ksapp` (`service\_id`) ON DELETE CASCADE ON UPDATE NO ACTION

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `ksoperations` (

`service\_id` varchar(32) NOT NULL COMMENT '服务id',

`operation\_id` varchar(15) NOT NULL COMMENT '操作id',

`operation\_json` varchar(1500) NOT NULL COMMENT '具体操作内容json',

KEY `operation\_index` (`service\_id`,`operation\_id`),

CONSTRAINT `fk\_ksoperations` FOREIGN KEY (`service\_id`) REFERENCES `ksapp` (`service\_id`) ON DELETE CASCADE ON UPDATE NO ACTION

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `ksoutput` (

`service\_id` varchar(32) NOT NULL COMMENT '服务id',

`output\_json` varchar(1800) NOT NULL COMMENT 'kstream的output配置',

PRIMARY KEY (`service\_id`),

CONSTRAINT `fk\_ksoutput` FOREIGN KEY (`service\_id`) REFERENCES `ksapp` (`service\_id`) ON DELETE CASCADE ON UPDATE NO ACTION

) ENGINE=InnoDB DEFAULT CHARSET=utf8;