技 术 文 件

版　　本：V1.0g

拟 制 Bill

实时构建系统

修改记录

文件编号	版本号	拟制人/修改人	拟制/修改日期	更改理由	主要更改内容 (写要点即可)
	V1.0a	Bill	2019/03/22	文档初稿	
注1：每次更改归档文件(指归档到事业部或公司档案室的文件)时，需填写此表。 注2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。					

概述

buleline是一个非常易用，高性能，能够应对海量数据的实时数据处理产品，构建于Apache Spark之上。

1.1 为了解决什么问题?

分布式数据处理入门易，精通难

太多Spark重复代码，重复逻辑可以抽象和简化

开发、测试、上线周期长

减少学习成本，加快分布式数据处理能力在生产环境落地

1.2 使用场景

- 海量数据ETL
- 海量数据聚合
- 多源数据处理

1.3 特性

- 简单易用，灵活配置，无需开发
- 实时流式处理
- 离线多源数据分析
- 高性能
- 海量数据处理能力
- 模块化和插件化，易于扩展
- 支持利用SQL做数据处理和聚合
- 支持Spark 2.x

1 input

1.2 StreamingKafkaStream

说明:基于spark streaming 接受kafka数据

支持的Kafka版本 >= 0.10.0

Name	Type	Require	Default
topics	String	yes	
consumer.group.id	String	yes	
consumer.bootstrap.servers	String	yes	
consumer.*	String	no	

topics[string]
Kafka topic名称。如果有多个topic，用","分割，例如: “tpc1,tpc2”。

consumer.group.id [string]
Kafka consumer group id，用于区分不同的消费组。

consumer.bootstrap.servers [string]
Kafka集群地址，多个用”,"隔开

consumer.* [string]
指定参数的方式是在原参数名称上加上前缀"consumer."，如指定rebalance.max.retries的方式是: consumer.rebalance.max.retries = 100

Examples

```
kafkaStream {
    topics = "waterdrop"
    consumer.bootstrap.servers = "localhost:9092"
    consumer.group.id = "waterdrop_group"
    consumer.rebalance.max.retries = 100
}
```

1.3 StructuredKafkaStream

说明:基于Structuredstreamkafka数据

支持的Kafka版本 >= 0.10.0

Name	Type	Require	Default
topics	String	yes	
consumer.bootstrap.servers	String	yes	

topics[string]
Kafka topic名称。如果有多个topic，用","分割，例如: “tpc1,tpc2”。

consumer.bootstrap.servers [string]
Kafka集群地址，多个用”,"隔开

Examples

```
kafkaStream {
    topics = "waterdrop"
    consumer.bootstrap.servers = "localhost:9092"
  }
```

1.3 Hive

说明：从hive中获取数据

name	type	required	default value
pre_sql	string	yes	-

table_name	string	yes	-

pre_sql [string]

进行预处理的sql, 如果不需要预处理,可以使用select * from hive_db.hive_table

table_name [string]

经过pre_sql获取到的数据，注册成临时表的表名

Example

```
hive {
  pre_sql = "select * from mydb.mytb"
  table_name = "myTable"
}
```

Notes

必须保证hive的metastore是在服务状态。启动命令 hive --service metastore 服务的默认端口的9083 cluster、client、local模式下必须把hive-site.xml置于提交任务节点的$HADOOP_CONF目录下,IDE本地调试将其放在resources目录

1.4 Mysql

说明:获取mysql数据

name	type	required	default value
password	string	yes	-
table	string	yes	-
table_name	string	yes	-
url	string	yes	-
user	string	yes	-

password [string]

密码

table [string]

表名

table_name [string]

注册为Spark临时表的表名

url [string]

JDBC连接的URL。参考一个案例：jdbc:mysql://localhost:3306/info

user [string]=

用户名

2.filter

2.1：Json

说明：对原始数据集指定字段进行Json解析

name	type	required	default value
source_field	string	no	raw_message
target_field	string	no	__root__
schema_dir	string	no	-
schema_file	string	no	-

source_field [string]
源字段，若不配置默认为raw_message

target_field [string]
目标字段，若不配置默认为__root__，Json解析后的结果将统一放置Dataframe最顶层

schema_dir [string]
样式目录，若不配置默认为$*/plugins/json/files/schemas/（建议使用）**

```
json {
    source_field = "message"
    schema_file = "demo.json"
}
```

Schema

在 Driver Node 的 /opt/waterdrop/plugins/json/files/schemas/demo.json 中放置内容如下：

```
{
 "name": "demo",
 "age": 24,
 "city": "LA"
}
```

schema_file [string]
样式文件名，若不配置默认为空，即不指定结构，由系统根据数据源输入自行推导。

2.2 Date

说明：对指定字段进行时间格式转换

name	type	required	default value
default_value	string	no	${now}
locale	string	no	Locale.US
source_field	string	no	__root__
source_time_format	string	no	UNIX_MS
target_field	string	no	datetime

target_time_format	string	no	yyyy/MM/dd HH:mm:ss
time_zone	string	no	-

default_value [string]
如果日期转换失败将会使用当前时间生成指定格式的时间

locale [string]
编码类型

source_field [string]
源字段，若不配置将使用当前时间

source_time_format [string]
源字段时间格式，当前支持UNIX(10位的秒时间戳)、UNIX_MS(13位的毫秒时间戳)以及SimpleDateFormat时间格式。

Symbol	Description
y	Year
M	Month
d	Day of month
H	Hour in day (0-23)
m	Minute in hour
s	Second in minute

target_field [string]
目标字段，若不配置默认为datetime

target_time_format [string]
目标字段时间格式

time_zone [string]
时区

Examples
```
date {
    source_field = "timestamp"
    target_field = "date"
    source_time_format = "UNIX"
    target_time_format = "yyyy/MM/dd"
}
```

2.3 Split

说明:根据delimiter分割字符串。

name	type	required	default value
delimiter	string	no	" "(空格)
fields	array	yes	-
source_field	string	no	root
target_field	string	no	raw_message

delimiter [string]

分隔符，根据分隔符对输入字符串进行分隔操作，默认分隔符为一个空格(" “)。

fields [list]
分割后的字段名称列表，按照顺序指定被分割后的各个字符串的字段名称。 若fields长度大于分隔结果长度，则多余字段赋值为空字符。

source_field [string]
被分割前的字符串来源字段，若不配置默认为raw_message

target_field [string]
target_field 可以指定被分割后的多个字段被添加到Event的位置，若不配置默认为_root_，即将所有分割后的字段，添加到Event最顶级。 如果指定了特定的字段，则被分割后的字段将被添加到这个字段的下面一级。

Examples

```
split {
    source_field = "message"
    delimiter = "&"
    fields = ["field1", "field2"]
}
```

2.4 Sql

使用SQL处理数据，支持Spark丰富的UDF函数

请参考：http://spark.apache.org/docs/latest/api/sql/

name	type	required	default value
sql	string	yes	-

table_name	string	yes	-

sql [string]
SQL语句

table_name [string]
表名，可为任意字符串, 这也是sql参数中使用的表名

Examples

```
sql {
    sql = "select username, address from user_info",
    table_name = "user_info"
}
```

2.4 watermark

事件时间水印设置值。抛弃超过设置值对对数据只允许基于spark structurestream 引擎中

name	type	required	default value
event_time	string	yes	-
delay_threshold	string	yes	-

event_time [string]
选择作为水印的字段 必须是timemap时间格式。

delay_threshold [string]

水印值　例子：30 seconds

Examples

```
watermark {
    event_time = "logTime"
    delay_threshold = "30 seconds"
    }
```

3.output

3.1KafkaSink

输出数据到Kafka 基于structurestream

Name	Type	Require	Default
topic	String	yes	
checkpointLocation	String	yes	
producer.bootstrap.servers	String	yes	
outputMode	String	no	Append
triggerMode	string	no	
Interval	String	No	

topic[string]

Kafka topic名称

checkpointLocation[string]

设置checkpoint 进行故障恢复

producer.bootstrap.servers[string]

Kafka集群地址，多个用","隔开

outputMode [string]

追加(append)(默认) 这是默认的模式.在上次触发后,只有新行添加到结果表(result-table)才会触发。这是只支持那些查询,行添加到结果表永远不会改变,因为这种模式保证每一行将只输出一次。例如,只有选择查询,地图,flatMap,过滤,加入等将支持附加模式

完全(complete) 每次触发时都会将整个结果表(result-table)输出.例如聚合结果等

更新(update) 只有结果表在上次被触发后被更新才会触发

不同的查询支持不同的输出模式,具体见下:

查询类型		支持模式	描述
带聚合的查询	聚合带事件时间的水印	Append, Update, Complete	Append 使用水印来删除旧的聚集状态 窗口将在水印最终阈值的时候聚合.所以结果将在水印过期,最终结果确定的情况下添加进结果表 Update 使用水印来删除旧的聚集状态 Complete 不会使用水印来删除旧的聚集状态
	不带水印的聚合	Complete, Update	由于没有使用水印,旧的聚集状态不会被删除.不支持append,因为对整数据的聚合结果会不断更新,不符合append的语义
mapGroupsWithState		Update	

flatMapGroupsWithState	追加操作模式	Append	flatMapGroupsWithState 之后允许 Aggregations （聚合）
	更新操作模式	Update	flatMapGroupsWithState 之后不允许 Aggregations （聚合）。
带join的查询		Append	join不支持Update,Complete模式
其它查询		Append, Update	不支持Complete.因为非聚合数据的结果表,全部保存进结果表时不允许的

triggerModep[string]

触发器

如果不设置任何触发器,将默认使用此触发器.触发规则是:上一个微批处理结束,启动下一个微批处理
建议不使用

Examples

```
kafka {
    producer.bootstrap.servers="172.18.50.66:9091"
    topic="test"
    checkpointLocation="/staructstreaming"
    outputMode = "Append"#Append,Update,Complete
    # triggerMode = "ProcessingTime"
    # interval = "5 seconds"
  }
```

3.2 StreamingKafkaSink

输出数据到Kafka 基于sparkstreaming

name	type	required	default value
producer.bootstrap.servers	string	yes	-
topic	string	yes	-
producer.*	string	no	-

producer.bootstrap.servers [string]
Kafka Brokers List

topic [string]
Kafka Topic

producer [string]
除了以上必备的kafka producer客户端必须指定的参数外，用户还可以指定多个producer客户端非必须参数

Examples
```
kafka {
    topic = "waterdrop"
    producer.bootstrap.servers = "localhost:9092"
}
```

3.3 StreamingMysqlSink

streaming输出数据到MySQL

name	type	required	default value
password	string	yes	-
save_mode	string	no	append
table	string	yes	-
url	string	yes	-
user	string	yes	-

password [string]
密码
save_mode [string]
存储模式，当前支持overwrite，append，ignore以及error。每个模式具体含义见
http://spark.apache.org/docs/2.2.0/sql-programming-guide.html#save-modes

table [string]
表名

url [string]
JDBC连接的URL。参考一个案例：jdbc:mysql://localhose:3306/info

user [string]
用户名

Example
```
mysql {
    url = "jdbc:mysql://localhost:3306/info"
    table = "access"
    user = "username"
```

```
    password = "password"
    save_mode = "append"
}
```

3.4 Elasticsearch

输出数据到Elasticsearch，支持的Elasticsearch版本为 >= 2.x。

name	type	required	default value
hosts	array	yes	-
index_type	string	no	log
index_time_format	string	no	yyyy.MM.dd
index	string	no	waterdrop
es	string	no	

hosts [array]
Elasticsearch集群地址，格式为host:port，允许指定多个host。如["host1:9200", “host2:9200"]。

index_type [string]
Elasticsearch index type

index_time_format [string]
当index参数中的格式为时xxxx-${now}，index_time_format可以指定index名称的时间格式，默认值为yyyy.MM.dd。常用的时间格式列举如下：

Symbol	Description
y	Year
M	Month
d	Day of month
H	Hour in day (0-23)
m	Minute in hour
s	Second in minute

index [string]
Elasticsearch index名称，如果需要根据时间生成index，可以指定时间变量，如：${now}。now 代表当前数据处理的时间。
es.* [string]
用户还可以指定多个非必须参数，详细的参数列表见
如es.batch.size.entries指定的方式是:es.batch.size.entries = 100000。如果不指定这些非必须参数，它们将使用官方文档给出的默认值。

Examples
```
elasticsearch {
    hosts = ["localhost:9200"]
    index = "waterdrop"
}
```
将结果写入Elasticsearch集群的名称为waterdrop的index中
```
elasticsearch {
    hosts = ["localhost:9200"]
    index = “name-${now}"
    es.batch.size.entries = 100000
    index_time_format = "yyyy.MM.dd"
}
```

3.5 Hdfs

输出数据到HDFS文件

name	type	required	default value
options	object	no	-
partition_by	array	no	-
path	string	yes	-
path_time_format	string	no	yyyyMMddHHmmss
save_mode	string	no	error
serializer	string	no	json

options [object]
自定义参数

partition_by [array]
根据所选字段对数据进行分区

path [string]
Hadoop集群文件路径，以hdfs://开头

path_time_format [string]

当path参数中的格式为时xxxx-${now}，path_time_format可以指定HDFS路径的时间格式，默认值为 yyyy.MM.dd。常用的时间格式列举如下：

Symbol	Description
y	Year
M	Month
d	Day of month
H	Hour in day (0-23)
m	Minute in hour
s	Second in minute

save_mode [string]
存储模式，当前支持overwrite，append，ignore以及error。

serializer [string]
序列化方法，当前支持csv、json、parquet、orc和text

Example

```
hdfs {
    path = "hdfs:///var/logs-${now}"
    serializer = "json"
    path_time_format = "yyyy.MM.dd"
}
```

3.6 MysqlSink

---

structuredstreaming 下输出到mysql

name	type	required	default value
password	string	yes	-
save_mode	string	no	append
table	string	yes	-
url	string	yes	-
user	string	yes	-
checkpointLocation	string	yes	
outputMode	string	no	append
prepareStatement	String	Yes	

password [string]
密码
save_mode [string]
存储模式，当前支持overwrite，append，ignore以及error。每个模式具体含义见
http://spark.apache.org/docs/2.2.0/sql-programming-guide.html#save-modes

table [string]
表名

url [string]
JDBC连接的URL。参考一个案例：jdbc:mysql://localhose:3306/info

user [string]

用户名

checkpointLocation[string]
设置checkpoint

outputMode[string]
输出模式 详情请看上文

prepareStatement[string]

Example
MysqlSink{
mysqlurl="jdbc:mysql://rm-uf6lh1cc24p213t4l.mysql.rds.aliyuncs.com:3306/tuya_puti?useSSL=false"
outputMode="Append"
checkpointLocation="/Metric_updata"
user="smart"
password="smart.daily_rw_1206"
prepareStatement="INSERT INTO puti_meter_data(`namespace`, `biz_type`, `currency`, `user_id`, `data1`, `data2`, `gmt_begin`, `gmt_end`, `creator`, `modifier`, `gmt_create`, `gmt_modified`, `env`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, 'bigData', 'bigData', ?, ?, ?)"
}

4.自定义input filter output以及udf

1/首先引入 blueline-apis_2.11-0.1.jar到工程

4.1 自定义input

4.1.1 新建类 继承 blueline-apis中 BaseStaticInput

例子:

```
class mystatisticstable extends BaseStaticInput{
  var config: Config = ConfigFactory.empty()

  override def getDataset(spark: SparkSession):
Dataset[Row] = {}

  override def setConfig(config: Config): Unit =
this.config=config

  override def getConfig(): Config = this.config

  override def checkConfig(): (Boolean, String) = {}
}
```

Input插件在调用时会先执行checkConfig方法核对调用插件时传入的参数是否正确，然后调用prepare方法配置参数的缺省值以及初始化类的成员变量，最后调用getDataset方法将外部数据源转换为Dataset[Row]

4.2 自定义filter

Filter是transform操作，负责对Dataset[Row]的数据结构进行操作

新建一个类，并继承blueline-apis提供的父类BaseFilter

```
class myfilter extends BaseFilter{
  var conf: Config = ConfigFactory.empty()
  override def process(spark: SparkSession, df: Dataset[Row]):
Dataset[Row] = {
```

```
  }

  override def setConfig(config: Config): Unit = this.conf=config

  override def getConfig(): Config = this.conf

  override def checkConfig(): (Boolean, String) = {

  }
}
```

Filter插件在调用时会先执行checkConfig方法核对调用插件时传入的参数是否正确，然后调用prepare方法配置参数的缺省值以及初始化类的成员变量，最后调用process方法对 Dataset[Row] 格式数据进行处理。

4.3 自定义output

Output是action操作，负责将Dataset[Row]输出到外部数据源或者打印到终端

```
class myoutput extends BaseOutput{
  var config: Config = ConfigFactory.empty()
  override def process(df: Dataset[Row]): Unit = {
  }

  override def setConfig(config: Config): Unit = {this.config=config}

  override def getConfig(): Config = {this.config}

  override def checkConfig(): (Boolean, String) = {}
}
```

Output插件调用结构与Filter插件相似。在调用时会先执行checkConfig方法核对调用插件时传入的参数是否正确，然后调用prepare方法配置参数的缺省值以及初始化类的成员变量，最后调用process方法将 Dataset[Row] 格式数据输出到外部数据源。

4.4 自定义udf

新建一个类，并继承blueline-apis提供的父类BaseFilter

```
class myfilter extends BaseFilter{
  var conf: Config = ConfigFactory.empty()
  override def process(spark: SparkSession, df: Dataset[Row]):
Dataset[Row] = {
df
  }

  override def setConfig(config: Config): Unit = this.conf=config

  override def getConfig(): Config = this.conf

  override def checkConfig(): (Boolean, String) = {

  }
 override def prepare(spark: SparkSession): Unit = {
spark.udf.register("isAdult", isAdult _)
}
def isAdult(age: Int) = {
  if (age < 18) {
    false
  } else {
    true
  }

}
}
```

在调用时会先执行checkConfig方法核对调用插件时传入的参数是否正确，然后调用prepare方法配置参数的缺省值以及初始化类的成员变量和udf的注册，最后调用process方法将 Dataset[Row] 格式数据输出到外部数据源。之后便可在sql 内使用udf方法。

4.5 打包

需要将第三方Jar包放到， plugins/your_plugin_name/lib/your_jar_name (必须新建lib文件夹)
例子：plugins/app/lib/app.jar

5 部署与运行

Waterdrop 依赖Java运行环境和Spark

5.1 部署

解压即可使用

5.2 运行

```
./bin/start-blueline.sh --master local[4] --deploy-mode client
--engine structuredstreaming --config ./config/Metric3.conf
```

—master 运行模式 目前支持三种 local[4], yarn,mesos

—deploy-mode:client,cluster

—engine:sparkstreaming,structuredstreaming

5.3 sparkstreaming模式 分为batch和streaming 根据input不同 自动进行识别

batch模式：

input	filter	output
Hive	不包括WaterMark	不包括kafkasink和Mysqlsink
Jdbc		
Mysql		

---

streaming模式

input	filter	output
StreamingKafkaStream	不包括WaterMark	不包括kafkasink和Mysqlsink

5.4 structuredstreaming 模式

input	filter	output
StructuredKafkaStream	都适用	KafkaSink
		MysqlSink

6 任务并行计算 StreamingPipeline
可以设置多个filter 和 output

例子:

```
spark {
  spark.streaming.batchDuration = 5
  spark.app.name = “app_statistics"
 spark.executor.instances = 2
  spark.executor.cores = 1
  spark.executor.memory = "1g"
}
input {
    kafkaStream {
    topics = "app_real_time_log"
    consumer.bootstrap.servers = "172.18.50.66:9091"
    consumer.group.id = "app_statistics_2010305"
    consumer.rebalance.max.retries = 100
    }
  }
filter{
json {
      source_field = "raw_message"
      #target_field = "info"
     }
}
StreamingPipeline <p1> {
filter {
    sql {
      sql = "select * from app_info group by info",
      table_name = "app_info"
  }
}
output {
    stdout {}
}
}
StreamingPipeline <p2> {
filter {
    sql {
      sql = "select * from app_info group by info",
      table_name = "app_info"
  }
}
output {
    stdout {}
}
}
```