ALIBABA CLOUD

阿里云

E-MapReduce

开发指南

文档版本：20210730

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.准备

1.1. 开发准备

本文介绍E-MapReduce开发的准备工作。

准备工作如下：

- 请确认您已经开通了阿里云服务，并创建了AccessKey ID和AccessKey Secret。请确认您已开通OSS。
- 您已经对Spark、Hadoop、Hive和Pig具备一定的认识。

  文中不对Spark、Hadoop、Hive和Pig开发实践进行额外的介绍。
- 您已经对阿里云E-MapReduce开发组件有一定了解。

1.2. OSS参考使用说明

本文介绍在E-MapReduce作业配置中使用的OSS URI。

OSS URI

使用E-MapReduce时，通常会使用两种OSS URI：

- native URI：*oss://[accessKeyId:accessKeySecret@]bucket[.endpoint]/object/path*。

  您在作业中指定输入输出数据源时使用此URI，等同于*hdfs://*。您操作OSS数据时，可以将AccessKey Id、AccessKey Secret以及Endpoint配置到Configuration中，也可以在URI中直接指定AccessKey Id、AccessKey Secret以及Endpoint。
- ref URI：*ossref://bucket/object/path*

  仅在E-MapReduce作业配置时有效，用来指定作业运行需要的资源。

例如，以下作业配置示例：

> **注意**
>
> 当前所有操作仅支持标准存储类型的OSS。
>
> E-MapReduce使用Multipart方式向OSS上传大文件。当作业异常中断后，OSS中会残留作业的部分结果数据，需要您手动删掉。此方式和使用HDFS的方式是一致的，区别在于，E-MapReduce会用到Multipart方式上传大文件，会将文件碎片上传到OSS的碎片管理中，所以您不仅要删除OSS文件管理中的作业残留文件，还需将OSS碎片管理中的文件碎片清理一次，否则会产生数据存储费用。您也可以配置碎片的生命周期，配置完成后过期的文件碎片会被自动清理掉，详情请参见OSS的文件生命周期管理说明。

1.3. 示例项目使用说明

本文介绍的项目都是完整的可编译可运行的项目，包括MapReduce、Pig、Hive和Spark。

示例项目

示例名称如下所示，详情代码示例请参见集群运行。

- MapReduce

  WordCount：单词统计

- Hive

  sample.hive：表的简单查询

- Pig

  sample.pig：Pig处理OSS数据实例

- Spark
  - SparkPi：计算Pi
  - SparkWordCount：单词统计
  - LinearRegression：线性回归
  - OSSSample：OSS使用示例
  - MaxComputeSample：MaxCompute使用示例
  - MNSSample：MNS使用示例
  - LoghubSample：Loghub使用示例

依赖资源

- 测试数据（data目录下）：
  - The_Sorrows_of_Young_Werther.txt：可作为WordCount（MapReduce或Spark）的输入数据。
  - patterns.txt：WordCount（MapReduce）作业的过滤字符。
  - u.data：sample.hive脚本的测试表数据。
  - abalone：线性回归算法测试数据。
- 依赖JAR包（lib目录下）：

  tutorial.jar：sample.pig作业需要的依赖JAR包。

准备工作

本文提供了一些测试数据，您将其上传到OSS中即可使用。您还可以自行准备以下可选测试数据，例如，MaxCompute、MNS、ONS和LogService等。

- 创建LogService，请参见快速入门。
- 创建MaxCompute项目和表，请参见创建MaxCompute项目和创建表。
- 创建ONS，请参见主账号快速入门。
- 创建MNS，请参见快速入门概述。

基本概念

- OSSURI：*oss://accessKeyId:accessKeySecret@bucket.endpoint/a/b/c.txt*，用户在作业中指定输入输出数据源时使用，类似*hdfs://*。
- 阿里云AccessKeyId和AccessKeySecret是您访问阿里云API的密钥，您可以在安全信息管理页面获取。

集群运行

开源项目运行示例如下：

- Spark

- SparkWordCount：

```
spark-submit --class SparkWordCount examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <outputPath> <numPartition>
```

参数说明如下：

参数	描述
inputPath	输入数据路径。
outputPath	输出路径。
numPartition	输入数据RDD分片数目。

- SparkPi:

```
spark-submit --class SparkPi examples-1.0-SNAPSHOT-shaded.jar
```

- OSSSample：

```
spark-submit --class OSSSample examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartition>
```

参数说明如下：

参数	描述
inputPath	输入数据路径。
numPartition	输入数据RDD分片数目。

- ONSSample：

```
spark-submit --class ONSSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId>
    <accessKeySecret> <consumerId> <topic> <subExpression> <parallelism>
```

参数说明如下：

参数	描述
accessKeyId	阿里云AccessKeyId。
accessKeySecret	阿里云AccessKeySecret。
consumerId	消费者ID。
topic	每个消息队列都有一个 topic。
subExpression	消息子主题。
parallelism	指定多少个接收器来消费队列消息。

- MaxComputeSample：

```
spark-submit --class MaxComputeSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId>
    <accessKeySecret> <envType> <project> <table> <numPartitions>
```

参数说明如下：

参数	描述
accessKeyId	阿里云AccessKeyId。
accessKeySecret	阿里云AccessKeySecret。
envType	0表示公网环境，1表示内网环境。 如果是本地调试选择 0，如果是在 E-MapReduce上执行请选择 1。
project	请参见MaxCompute术语表。
table	请参见MaxCompute术语表。
numPartition	输入数据RDD分片数目。

- MNSSample：

```
spark-submit --class MNSSample examples-1.0-SNAPSHOT-shaded.jar <queueName>
    <accessKeyId> <accessKeySecret> <endpoint>
```

参数说明如下：

参数	描述
queueName	Queue的名称。
accessKeyId	阿里云AccessKeyId。
accessKeySecret	阿里云AccessKeySecret。
endpoint	队列数据访问地址。

- LoghubSample:

```
spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar <sls project> <sls
    logstore> <loghub group name> <sls endpoint> <access key id> <access key secret> <batch
    interval seconds>
```

参数说明如下:

参数	描述
sls project: LogService	项目名。
sls logstore	日志库名。
loghub group name	作业中消费日志数据的组名，可以任意取。sls project和 sls store相同时，相同组名的作业会协同消费sls store中的数据；不同组名的作业会相互隔离地消费sls store中的数据。
sls endpoint	请参见日志服务入口。
accessKeyId	阿里云AccessKeyId。
accessKeySecret	阿里云AccessKeySecret。

- LinearRegression:

```
spark-submit --class LinearRegression examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartitions>
```

参数说明如下:

参数	描述
inputPath	输入数据路径。
numPartition	输入数据RDD分片数目。

- Mapreduce

WordCount:

```
hadoop jar examples-1.0-SNAPSHOT-shaded.jar WordCount
    -Dwordcount.case.sensitive=true <inputPath> <outputPath> -skip <patternPath>
```

参数说明如下。

参数	描述
inputPath	输入数据路径。
outputPath	输出路径。
patternPath	过滤字符文件，可以使用*data*或*patterns.txt*。

- Hive

  ```
  hive -f sample.hive -hiveconf inputPath=<inputPath>
  ```

  参数说明如下。

参数	描述
inputPath	输入数据路径。

- Pig

  ```
  pig -x mapreduce -f sample.pig -param tutorial=<tutorialJarPath> -param
      input=<inputPath> -param result=<resultPath>
  ```

  参数说明如下：

参数	描述
tutorialJarPath	依赖Jar包，可使用lib或tutorial.jar。
inputPath	输入数据路径。
resultPath	输出路径。

> 注意
>
> - 在E-MapReduce上使用时，请将测试数据和依赖jar包上传到OSS中，路径规则遵循OSSURI定义。
> - 如果集群中使用，可以放在机器本地。

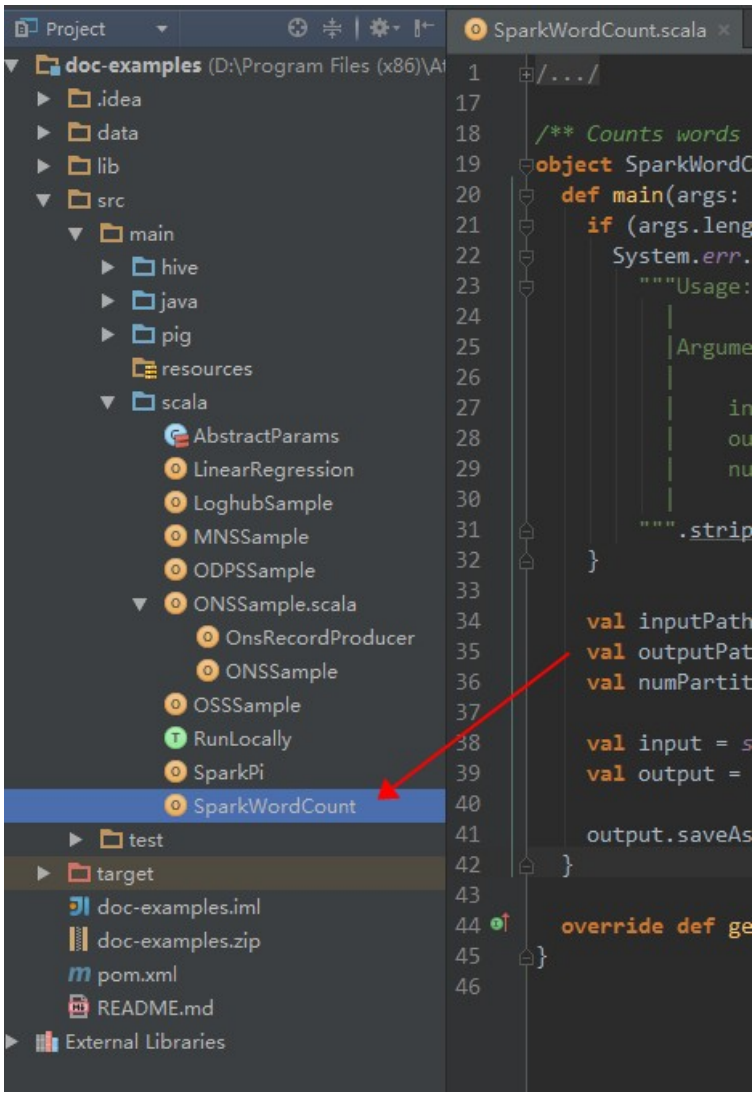
本地运行

这里主要介绍如何在本地运行Spark程序访问阿里云数据源，例如OSS等。如果希望本地调试运行，最好借助一些开发工具，例如IntelliJ IDEA或者 Eclipse，尤其是对于 Windows环境，否则需要在Windows机器上配置Hadoop和Spark运行环境。

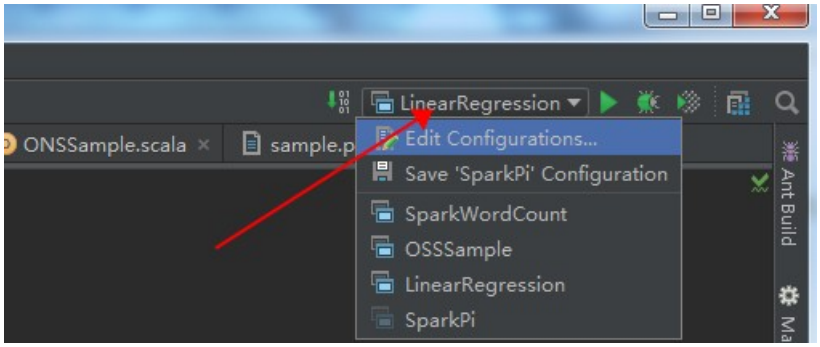
- IntelliJ IDEA
  - 准备工作

    安装IntelliJ IDEA、Maven、 IntelliJ IDEA Maven插件、Scala和IntelliJ IDEA Scala插件。
  - 开发流程

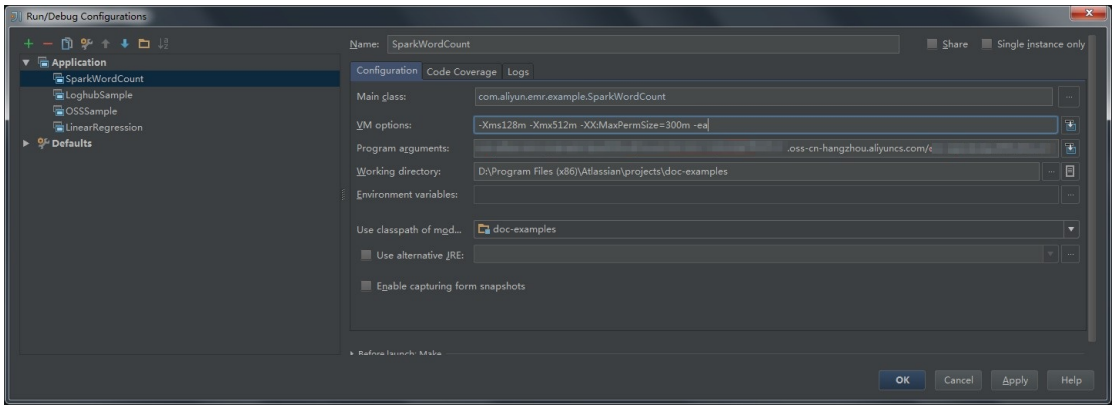
a. 双击进入SparkWordCount.scala。



b. 进入作业配置界面。

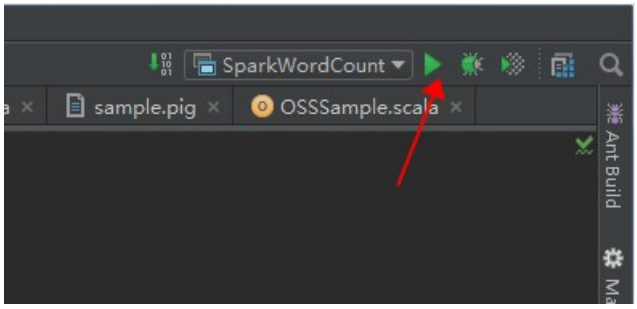


c. 选择 SparkWordCount，在作业参数框中按照所需传入作业参数。

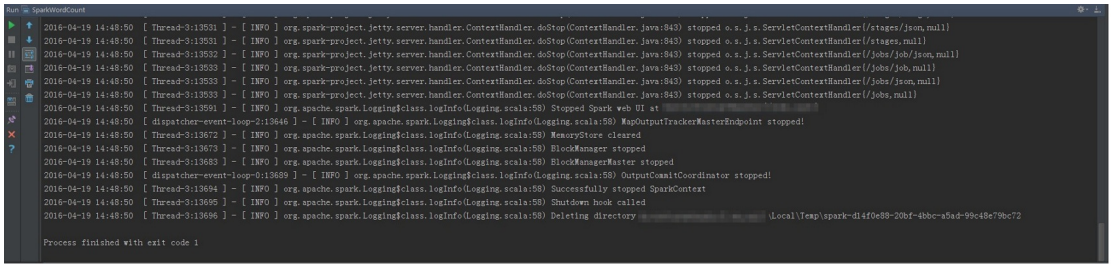


d. 单击 **OK**。

e. 单击**运行**，执行作业。



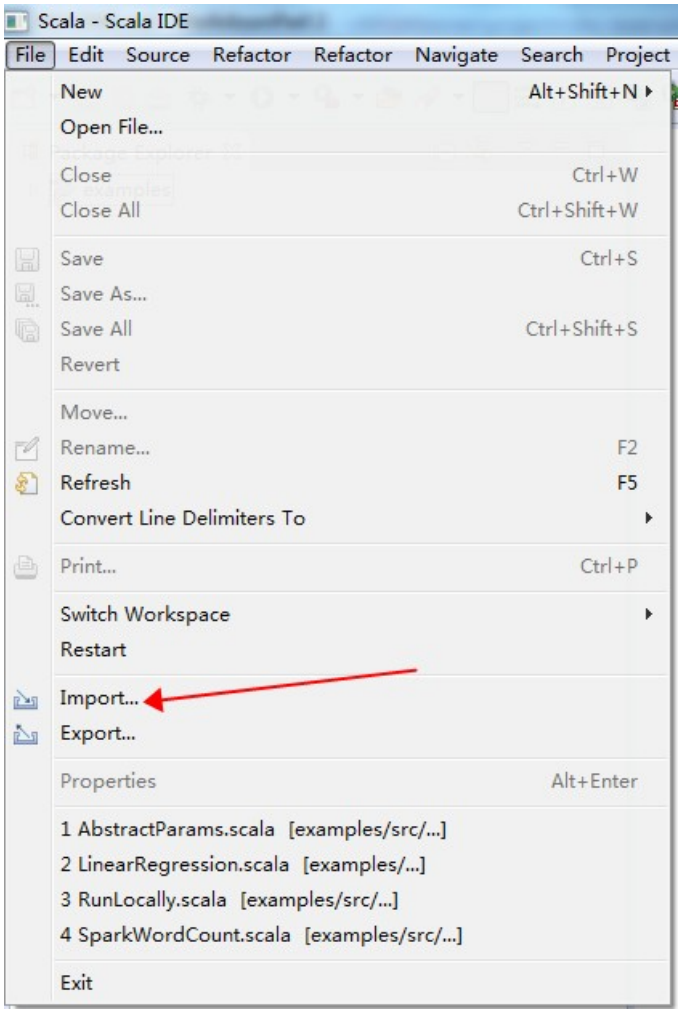
f. 查看作业执行日志。

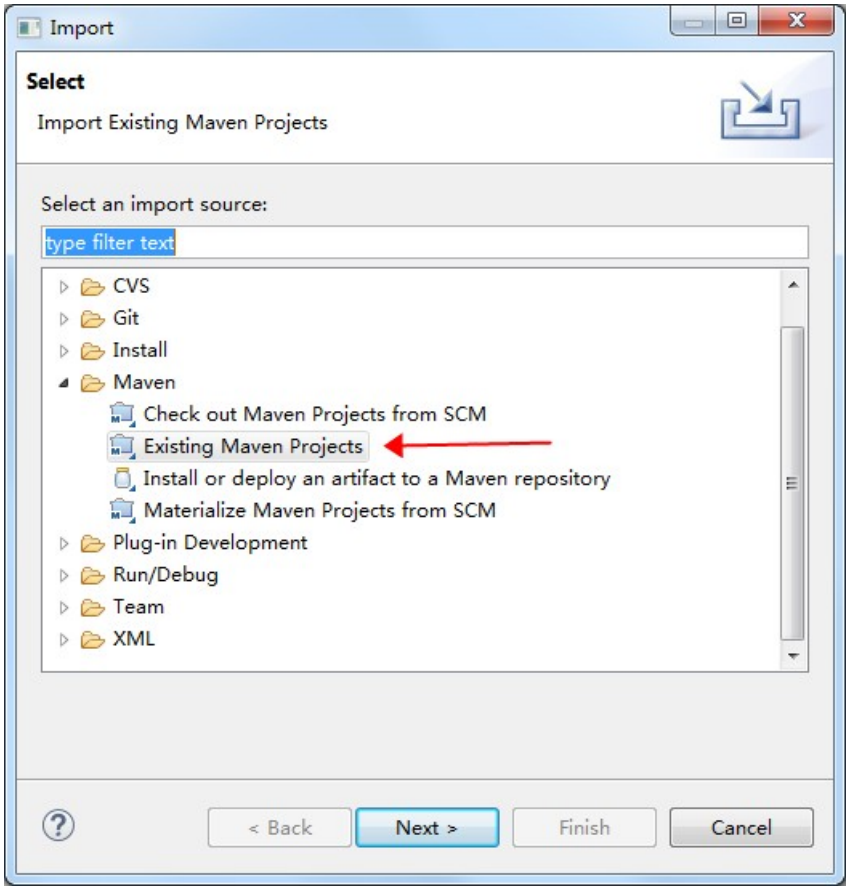


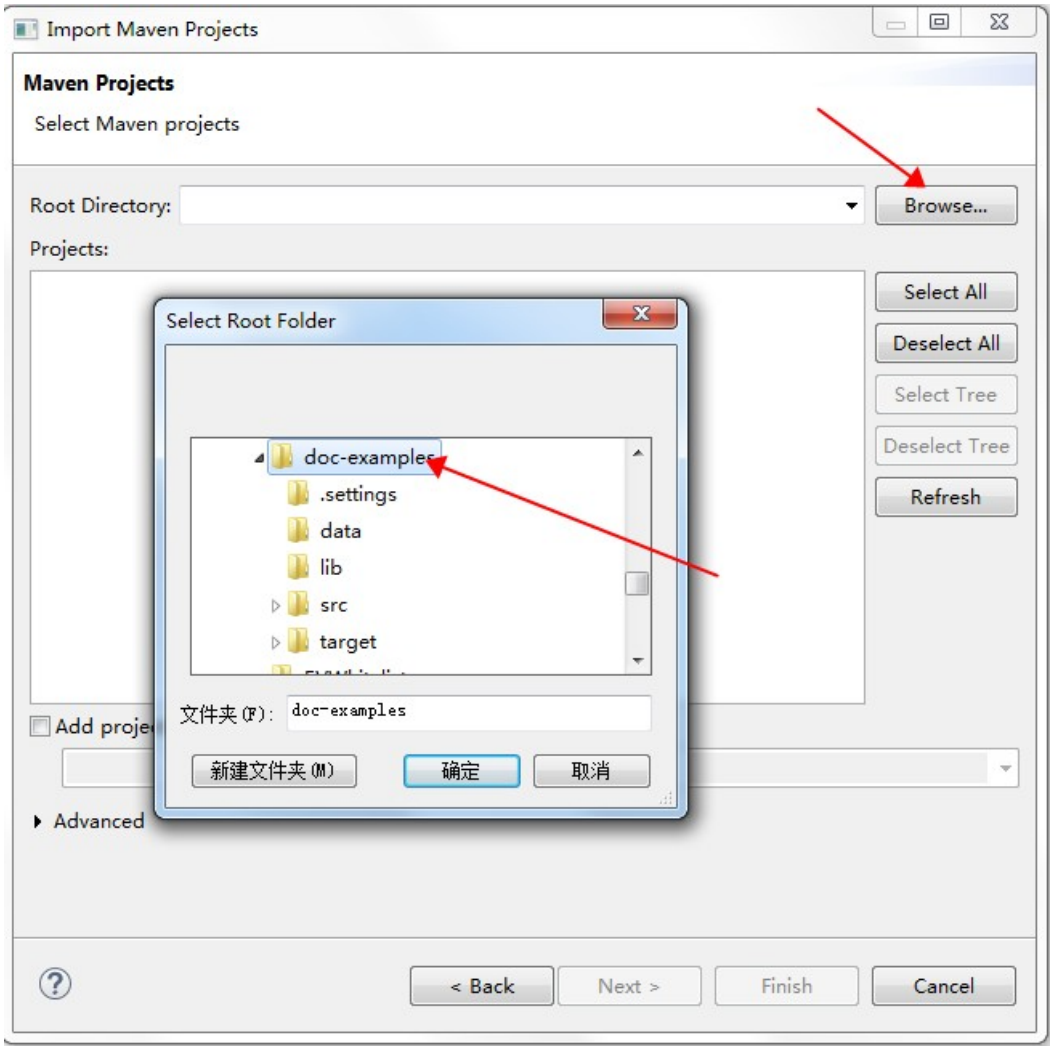
- Scala IDE for Eclipse
  - 准备工作

    安装Scala IDE for Eclipse、Maven、Eclipse Maven插件。
  - 开发流程

    a. 请根据以下图示导入项目。


Scala - Scala IDE
File Edit Source Refactor Refactor Navigate Search Project
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 AbstractParams.scala [examples/src/...]
2 LinearRegression.scala [examples/...]
3 RunLocally.scala [examples/src/...]
4 SparkWordCount.scala [examples/src/...]
Exit

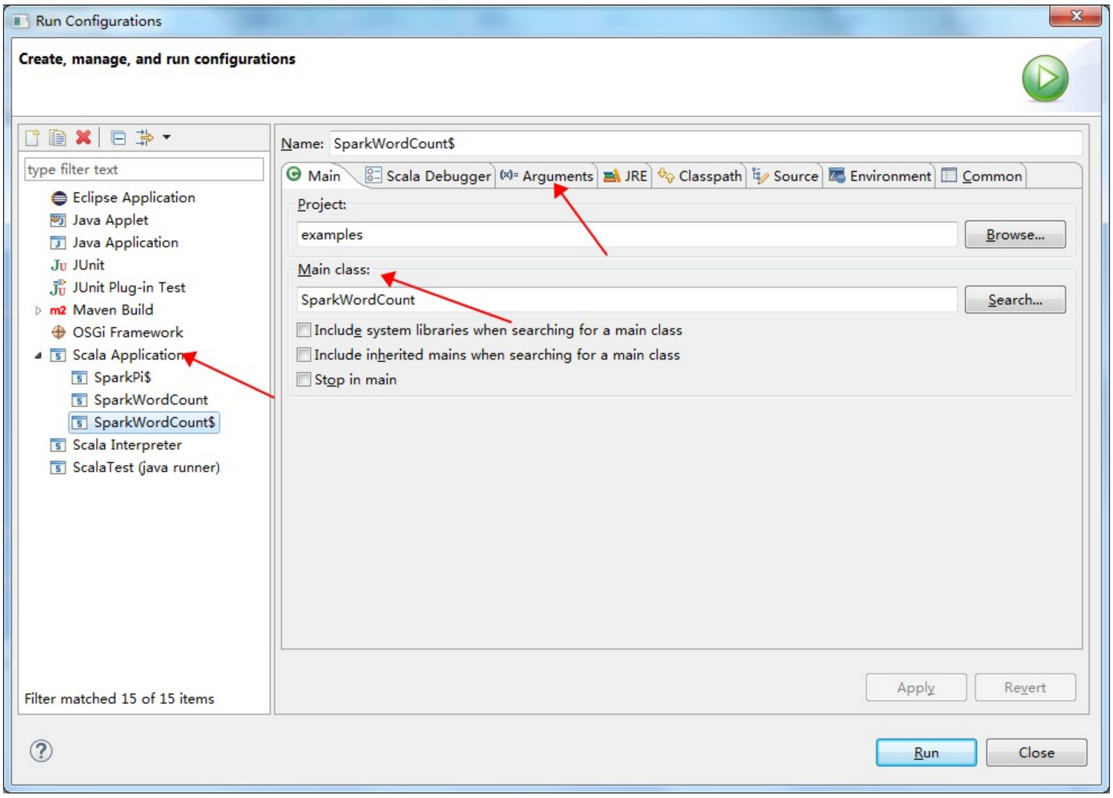

Import
Select
Import Existing Maven Projects
Select an import source:
type filter text
CVS
Git
Install
Maven
Check out Maven Projects from SCM
Existing Maven Projects
Install or deploy an artifact to a Maven repository
Materialize Maven Projects from SCM
Plug-in Development
Run/Debug
Team
XML
< Back
Next >
Finish
Cancel



b. Run As Maven build，快捷键是Alt + Shilft + X, M，也可以在项目名上右键选择**Run As > Maven build**。

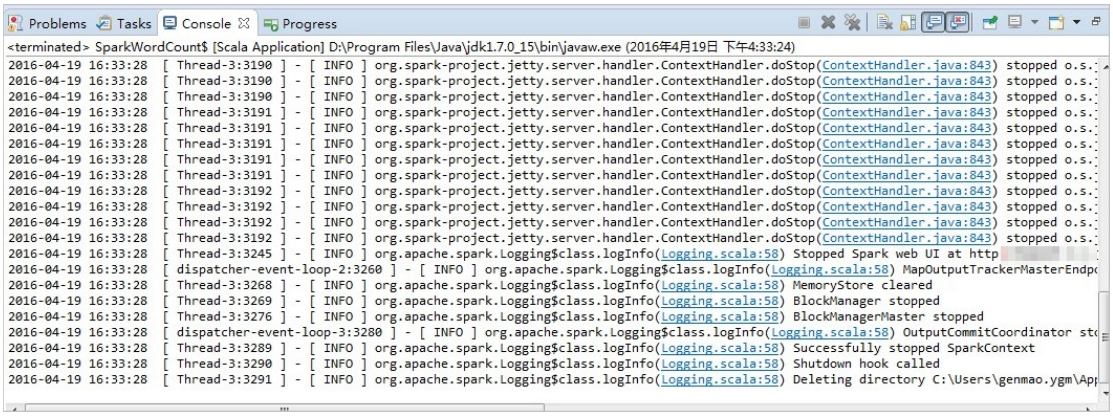
c. 等待编译完后，在需要运行的作业上右键，选择**Run Configuration**，进入配置页。

d. 在配置页中，选择 Scala Application，并配置作业的Main Class和参数等等。如下图所示。



e. 单击Run。

f. 查看控制台输出日志，如下图所示。



1.4. Python使用说明

E-MapReduce使用的Python 3版本为Python 3.6.4。

支持Python 3.6.4

- EMR-2.10.0及后续版本和EMR-3.10.0及后续版本，支持Python 3.6.4。

  Python文件安装目录为*/usr/bin/python3.6*。

- EMR-2.10.0和EMR-3.10.0之前版本默认不支持Python 3版本，您需要自行下载安装，步骤如下：

  i. 下载Python 3软件包：Python-3.6.4.tgz。

ii. 使用文件传输工具（SSH Secure File Transfer Client），上传JAR包至Master节点的*/usr/local*目录。

iii. 解压下载文件并安装。

a. 登录Master节点，详情请参见登录集群。

b. 创建Python 3的安装目录。

```
mkdir -p /usr/local/python3
```

c. 解压缩下载文件。

```
tar zxvf Python-3.6.4.tgz
```

d. 进入解压后的目录，指定安装路径。

```
cd Python-3.6.4
./configure --prefix=/usr/local/python3
```

e. 执行如下命令，进行编译和安装。

```
make && make install
```

f. 建立Python 3的软链。

```
ln -s /usr/local/python3/bin/python3 /usr/bin/python3
```

iv. 查看Python 3是否配置正确。

```
python3 -V
```

返回如下信息说明Python 3安装成功。

```
Python 3.6.4
```

v. 查看PiP 3是否配置正确。

```
pip3 -V
```

返回如下信息说明PiP 3安装成功。

```
pip 9.0.1 from /usr/local/Python-3.6.4/lib/python3.6/site-packages (python 3.6)
```

2.Spark

2.1. 准备工作

本文介绍Spark开发所需要的准备工作。

安装E-MapReduce SDK

您可以通过以下两种方法安装E-MapReduce SDK：

- 直接在Eclipse中使用JAR包，步骤如下：
  i. 从Maven库源下载E-MapReduce开发的依赖包。
  ii. 将所需的JAR包拷贝到您的工程文件夹中。（根据您运行作业集群的Spark版本选择SDK版本，Spark 1.x版本建议使用2.10，Spark 2.x建议使用2.11）
  iii. 在Eclipse中工程的名字上单击右键，然后依次选择**Properties > Java Build Path > Add JARs**。
  iv. 选择您下载的JAR包。

  完成后，您可以在工程中读写OSS、LogService、MNS、ONS、OTS和MaxCompute等数据。
- 创建Maven工程，添加如下依赖。

```
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-tablestore</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-mns_2.11</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-logservice_2.11</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-maxcompute_2.11</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-ons_2.11</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-datahub_2.11</artifactId>
  <version>2.0.0</version>
</dependency>
```

Spark代码本地调试

本地调试运行Spark代码读写OSS数据时，需要配置SparkConf，设置 `spark.hadoop.mapreduce.job.run-local` 为 `true` ，其余参数保持默认即可。示例代码如下。

- EMR-3.24.0及后续版本

```
val conf = new SparkConf().setAppName(getAppName).setMaster("local[4]")
  conf.set("spark.hadoop.fs.oss.impl", "com.aliyun.emr.fs.oss.JindoOssFileSystem")
  conf.set("spark.hadoop.mapreduce.job.run-local", "true")
  val sc = new SparkContext(conf)
  val data = sc.textFile("oss://...")
  println(s"count: ${data.count()}")
```

- EMR-3.24.0之前版本

```
val conf = new SparkConf().setAppName(getAppName).setMaster("local[4]")
  conf.set("spark.hadoop.fs.oss.impl", "com.aliyun.fs.oss.nat.NativeOssFileSystem")
  conf.set("spark.hadoop.mapreduce.job.run-local", "true")
  val sc = new SparkContext(conf)
  val data = sc.textFile("oss://...")
  println(s"count: ${data.count()}")
```

常见问题

- 三方依赖说明

  您的作业需要依赖三方包，即可在E-MapReduce上操作阿里云的数据源，例如OSS和MaxCompute等。

  增删需要依赖的三方包时，请参见pom。

- OSS输出目录设置

  在本地的Hadoop配置文件中增加配置参数fs.oss.buffer.dirs，参数值是本地的目录路径。如果您没有设置这个参数值，则在本地运行Spark程序，写入OSS数据时提示空指针异常。

- 垃圾清理

  Spark作业失败后，产生的数据不会主动清理。因此当您的Spark作业运行失败后，请检查OSS输出目录是否有文件存在，并且检查OSS碎片管理中是否存在提交的碎片，如果存在请及时清理。

- PySpark使用说明

  当您使用Python来编写Spark作业时，请参见aliyun-emapreduce-sdk配置您的环境。

2.2. 参数说明

本文介绍Spark代码中参数及Smart Shuffle优化配置参数。

Spark代码中可以使用如下参数配置。

属性名	默认值	说明
spark.hadoop.fs.jfs.cache.oss-accessKeyId	无	访问OSS所需的AccessKey ID（可选）。
spark.hadoop.fs.jfs.cache.oss-accessKeySecret	无	访问OSS所需的AccessKey Secret（可选）。

属性名	默认值	说明
spark.hadoop.fs.jfs.cache.oss-securityToken	无	访问OSS所需的STS token（可选）。
spark.hadoop.fs.jfs.cache.oss-endpoint	无	访问OSS的Endpoint（可选）。
spark.hadoop.fs.oss.copy.simple.max.byte	134217728	使用普通接口进行OSS内部拷贝的文件大小上限。
spark.hadoop.fs.oss.multipart.split.max.byte	67108864	使用普通接口进行OSS内部拷贝的文件分片大小上限。
spark.hadoop.fs.oss.impl	• EMR-3.24.0及后续版本：com.aliyun.emr.fs.oss.JindoOssFileSystem • EMR-3.24.0之前版本：com.aliyun.fs.oss.nat.NativeOssFileSystem	OSS文件系统实现类。
spark.hadoop.job.runlocal	false	当数据源是OSS时，是否需要本地调试运行Spark代码，需要时设置此项为true，否则为false。
spark.logservice.fetch.interval.millis	200	Receiver向LogHub取数据的时间间隔。
spark.logservice.fetch.inOrder	true	是否有序消费分裂后的Shard数据。
spark.logservice.heartbeat.interval.millis	30000	消费进程的心跳保持间隔。
spark.mns.batchMsg.size	16	批量拉取MNS消息条数，最大值为16。
spark.mns.pollingWait.seconds	30	MNS队列为空时的拉取等待间隔。
spark.hadoop.io.compression.codec.snappy.native	false	标识Snappy文件是否为标准Snappy文件。Hadoop默认识别的是Hadoop修改过的Snappy格式文件。

Smart Shuffle优化配置

Smart Shuffle是EMR-3.16.0针对Spark SQL提供的一种Shuffle实现，适合处理Shuffle数据量较大的查询。通过提前将Shuffle数据经过网络传输到远程节点，提高Spark SQL的执行效率。

启用Smart Shuffle后，Shuffle Output数据不会存储在本地磁盘，而是提前通过网络传输到远端节点，所有属于相同Partition的数据会被传输到同一个节点，并存储在同一个文件中。使用Smart Shuffle有以下限制：

- Smart Shuffle不保证Task执行的原子性，当Task失败时，需要重启整个Spark job。
- 因为没有提供External Smart Shuffle实现，所以不支持Dynamic Resource Allocation。
- 不支持Speculative Task特性。

- 与Adaptive Execution不兼容。

您可以通过Spark配置文件或者spark-submit参数启用Smart Shuffle，相关配置如下。

参数名	默认值	说明
spark.shuffle.manager	sort	通过spark.shuffle.manager=org.apache.spark.shuffle.sort.SmartShuffleManager或spark.shuffle.manager=smart配置Spark Session的Shuffle，启用Smart shuffle功能。
spark.shuffle.smart.spill.memorySizeForceSpillThreshold	128m	设置Shuffle数据占用内存的大小，超过阈值后，Shuffle数据会根据Sartition通过网络传输到对应远程节点。
spark.shuffle.smart.transfer.blockSize	1m	设置Shuffle在网络传输数据块的大小。

2.3. Spark对接OSS

本文介绍Spark如何读取OSS中的数据。

背景信息

当前E-MapReduce：

- 支持MetaService服务。
- 支持通过免AccessKey方式访问OSS数据源。
- 支持通过显式写AccessKey和Endpoint方式访问OSS数据源。

> 说明 OSS Endpoint需使用内网域名。域名详情信息，请参见OSS Endpoint。

Spark接入OSS示例

本示例为您展示，Spark如何以免AccessKey方式读取OSS中数据，并将处理完的数据写回至OSS。

```
val conf = new SparkConf().setAppName("Test OSS")
val sc = new SparkContext(conf)
val pathIn = "oss://bucket/path/to/read"
val inputData = sc.textFile(pathIn)
val cnt = inputData.count
println(s"count: $cnt")
val outputPath = "oss://bucket/path/to/write"
val outpuData = inputData.map(e => s"$e has been processed.")
outpuData.saveAsTextFile(outputPath)
```

完整示例代码，请参见Spark对接OSS。

PySpark接入OSS示例

本示例为您展示，PySpark如何以免AccessKey方式读取OSS中数据，并将处理完的数据写回至OSS。

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark SQL OSS example").getOrCreate()
pathIn = "oss://bucket/path/to/read"
df = spark.read.text(pathIn)
cnt = df.count()
print(cnt)
outputPath = "oss://bucket/path/to/write"
df.write.format("parquet").mode('overwrite').save(outputPath)
```

2.4. Spark对接MaxCompute

本文介绍如何在Spark中进行MaxCompute数据的读写操作。

操作步骤

1. 初始化一个OdpsOps对象。

   在Spark中，MaxCompute的数据操作通过OdpsOps类完成。

```
import com.aliyun.odps.TableSchema
  import com.aliyun.odps.data.Record
  import org.apache.spark.aliyun.odps.OdpsOps
  import org.apache.spark.{SparkContext, SparkConf}
  object Sample {
   def main(args: Array[String]): Unit = {
    // == Step-1 ==
    val accessKeyId = "<accessKeyId>"
    val accessKeySecret = "<accessKeySecret>"
    // 以内网地址为例。
    val urls = Seq("http://odps-ext.aliyun-inc.com/api", "http://dt-ext.odps.aliyun-inc.com")
    val conf = new SparkConf().setAppName("Test Odps")
    val sc = new SparkContext(conf)
    val odpsOps = OdpsOps(sc, accessKeyId, accessKeySecret, urls(0), urls(1))
    // 下面是一些调用代码。
    // == Step-2 ==
    ...
    // == Step-3 ==
    ...
   }
   // == Step-2 ==
   // 方法定义1
   // == Step-3 ==
   // 方法定义2
  }
```

2. 加载MaxCompute中的表数据至Spark。

   通过OdpsOps对象的readTable方法，您可以将MaxCompute中的表数据加载到Spark中。

```
// == Step-2 ==
    val project = <odps-project>
    val table = <odps-table>
    val numPartitions = 2
    val inputData = odpsOps.readTable(project, table, read, numPartitions)
    inputData.top(10).foreach(println)
    // == Step-3 ==
    ...
```

在上面的代码中，您还需要定义一个read函数，用来解析和预处理MaxCompute表数据，代码如下所示。

```
def read(record: Record, schema: TableSchema): String = {
     record.getString(0)
    }
```

3. 保存Spark中的结果数据至MaxCompute表中。

   通过OdpsOps对象的saveToTable方法，您可以保存Spark中的结果数据至MaxCompute表中。

   ```
   val resultData = inputData.map(e => s"$e has been processed.")
       odpsOps.saveToTable(project, table, dataRDD, write)
   ```

   在上面的代码中，您还需要定义一个write函数，用来进行数据预处理，代码如下所示。

   ```
   def write(s: String, emptyReord: Record, schema: TableSchema): Unit = {
        val r = emptyReord
        r.set(0, s)
       }
   ```

4. 分区表参数写法说明

   SDK支持对MaxCompute分区表的读写，分区名的写法标准是分区列名=分区名，多个分区时以逗号（,）分隔。

   - 读分区pt为1的表数据：pt='1'。
   - 读分区pt为1和分区ps为2的表数据：pt='1', ps='2'。

附录

完整示例代码，请参见Spark对接MaxCompute。

2.5. Spark对接RocketMQ

本文介绍如何通过Spark Streaming消费消息队列RocketMQ（简称MQ）中的数据并计算每个Batch中的单词。

通过Spark访问MQ

代码示例如下。

```
val Array(cId, topic, subExpression, parallelism, interval) = args
  val accessKeyId = "<accessKeyId>"
  val accessKeySecret = "<accessKeySecret>"
  val numStreams = parallelism.toInt
  val batchInterval = Milliseconds(interval.toInt)
  val conf = new SparkConf().setAppName("Test ONS Streaming")
  val ssc = new StreamingContext(conf, batchInterval)
  def func: Message => Array[Byte] = msg => msg.getBody
  val onsStreams = (0 until numStreams).map { i =>
   println(s"starting stream $i")
   OnsUtils.createStream(
    ssc,
    cId,
    topic,
    subExpression,
    accessKeyId,
    accessKeySecret,
    StorageLevel.MEMORY_AND_DISK_2,
    func)
  }
  val unionStreams = ssc.union(onsStreams)
  unionStreams.foreachRDD(rdd => {
   rdd.map(bytes => new String(bytes)).flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _).collect().foreach(e => println(s"word: ${e._1}, cnt: ${e._2}"))
  })
  ssc.start()
  ssc.awaitTermination()
```

附录

完整示例代码，请参见通过Spark访问RocketMQ。

2.6. Spark对接Table Store

本文介绍如何在Spark中消费Table Store的数据。

Spark接入Table Store

准备一张数据表pet，其中name为主键。

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	-
Claws	Gwen	cat	m	1994-03-17	-
Buffy	Harold	dog	f	1989-05-13	-
Fang	Benny	dog	m	1990-08-27	-
Bowser	Diane	dog	m	1979-08-31	1995-07-29

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	-
Whistler	Gwen	bird	-	1997-12-09	-
Slim	Benny	snake	m	1996-04-29	-
Puffball	Diane	hamster	f	1999-03-30	-

以下示例演示了如何在Spark中消费Table Store的数据。

```
private static RangeRowQueryCriteria fetchCriteria() {
  RangeRowQueryCriteria res = new RangeRowQueryCriteria("pet");
  res.setMaxVersions(1);
  List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
  List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
  lower.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MIN));
  upper.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MAX));
  res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
  res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
  return res;
}
public static void main(String[] args) {
  SparkConf sparkConf = new SparkConf().setAppName("RowCounter");
  JavaSparkContext sc = new JavaSparkContext(sparkConf);
  Configuration hadoopConf = new Configuration();
  JavaSparkContext sc = null;
  try {
    sc = new JavaSparkContext(sparkConf);
    Configuration hadoopConf = new Configuration();
    TableStore.setCredential(
        hadoopConf,
        new Credential(accessKeyId, accessKeySecret, securityToken));
    Endpoint ep = new Endpoint(endpoint, instance);
    TableStore.setEndpoint(hadoopConf, ep);
    TableStoreInputFormat.addCriteria(hadoopConf, fetchCriteria());
    JavaPairRDD<PrimaryKeyWritable, RowWritable> rdd = sc.newAPIHadoopRDD(
        hadoopConf, TableStoreInputFormat.class,
        PrimaryKeyWritable.class, RowWritable.class);
    System.out.println(
      new Formatter().format("TOTAL: %d", rdd.count()).toString());
  } finally {
    if (sc != null) {
      sc.close();
    }
  }
}
```

附录

完整示例代码，请参见Spark接入Table Store。

2.7. Spark对接LogService

2.7.1. 开发入门

本文介绍Spark Streaming如何消费Log Service中的日志数据和统计日志条数。

Spark接入Log Service

- 方法一：Receiver Based DStream

```
val logServiceProject = args(0)   // LogService中的project名。
  val logStoreName = args(1)    // LogService中的logstore名。
  val loghubConsumerGroupName = args(2)  // loghubGroupName相同的作业将共同消费logstore的数据。
  val loghubEndpoint = args(3)  // 阿里云日志服务数据类API Endpoint。
  val accessKeyId = "<accessKeyId>"    // 访问日志服务的AccessKey Id。
  val accessKeySecret = "<accessKeySecret>" // 访问日志服务的AccessKey Secret。
  val numReceivers = args(4).toInt  // 启动多少个Receiver来读取logstore中的数据。
  val batchInterval = Milliseconds(args(5).toInt * 1000) // Spark Streaming中每次处理批次时间间隔。
  val conf = new SparkConf().setAppName("Test Loghub Streaming")
  val ssc = new StreamingContext(conf, batchInterval)
  val loghubStream = LoghubUtils.createStream(
   ssc,
   logServiceProject,
   logStoreName,
   loghubConsumerGroupName,
   loghubEndpoint,
   numReceivers,
   accessKeyId,
   accessKeySecret,
   StorageLevel.MEMORY_AND_DISK)
  loghubStream.foreachRDD(rdd => println(rdd.count()))
  ssc.start()
  ssc.awaitTermination()
```

- 方法二： Direct API Based DStream

```
val logServiceProject = args(0)
  val logStoreName = args(1)
  val loghubConsumerGroupName = args(2)
  val loghubEndpoint = args(3)
  val accessKeyId = args(4)
  val accessKeySecret = args(5)
  val batchInterval = Milliseconds(args(6).toInt * 1000)
  val zkConnect = args(7)
  val checkpointPath = args(8)
  def functionToCreateContext(): StreamingContext = {
   val conf = new SparkConf().setAppName("Test Direct Loghub Streaming")
   val ssc = new StreamingContext(conf, batchInterval)
   val zkParas = Map("zookeeper.connect" -> zkConnect, "enable.auto.commit" -> "false")
   val loghubStream = LoghubUtils.createDirectStream(
    ssc,
    logServiceProject,
    logStoreName,
    loghubConsumerGroupName,
    accessKeyId,
    accessKeySecret,
    loghubEndpoint,
    zkParas,
    LogHubCursorPosition.END_CURSOR)
   ssc.checkpoint(checkpointPath)
   val stream = loghubStream.checkpoint(batchInterval)
   stream.foreachRDD(rdd => {
    println(rdd.count())
    loghubStream.asInstanceOf[CanCommitOffsets].commitAsync()
   })
   ssc
  }
  val ssc = StreamingContext.getOrCreate(checkpointPath, functionToCreateContext _)
  ssc.start()
  ssc.awaitTermination()
```

从E-MapReduce SDK 1.4.0版本开始，提供基于Direct API的实现方式。此种方式可以避免将Loghub数据重复存储到Write Ahead Log中，即无需开启Spark Streaming的WAL特性即可实现数据的at least once。目前Direct API实现方式处于experimental状态，需要注意以下几点：

- 在DStream的action中，必须做一次commit操作。
- 一个Spark Streaming中，不支持对LogStore数据源做多个action操作。
- Direct API方式需要Zookeeper服务的支持。

支持MetaService

上面的例子中是显式地将AccessKey传入到接口中，但是从E-MapReduce SDK 1.3.2版本开始，Spark Streaming可以基于MetaService实现免AccessKey处理LogService数据，具体可以参见E-MapReduce SDK中的LoghubUtils类说明。

```
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, storageLevel)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, numReceivers, storageLevel)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, storageLevel, cursorPosition, mLoghubCursorStartTime, forceSpecial)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, numReceivers, storageLevel, cursorPosition, mLoghubCursorStartTime, forceSpecial)
```

说明

- E-MapReduce SDK支持Log Service的三种消费模式，即BEGIN_CURSOR、END_CURSOR和SPECIAL_TIMER_CURSOR，默认是 END_CURSOR。
  - BEGIN_CURSOR：从日志头开始消费，如果有checkpoint记录，则从checkpoint处开始消费。
  - END_CURSOR：从日志尾开始消费，如果有checkpoint记录，则从checkpoint处开始消费。
  - SPECIAL_TIMER_CURSOR：从指定时间点开始消费，如果有checkpoint记录，则从checkpoint处开始消费，单位为秒。

  以上三种消费模式都受到checkpoint记录的影响，如果存在checkpoint记录，则从checkpoint处开始消费，不管指定的是什么消费模式。E-MapReduce SDK基于“SPECIAL_TIMER_CURSOR”模式支持用户强制在指定时间点开始消费，在LoghubUtils#createStream接口中，以下参数需要组合使用：
  - cursorPosition：LogHubCursorPosition.SPECIAL_TIMER_CURSOR
  - forceSpecial：true
- E-MapReduce的服务器（除了Master节点）无法连接公网。配置LogService endpoint时，请注意使用Log Service提供的内网 endpoint，否则无法请求到Log Service。

附录

完整示例代码，请参见Spark接入LogService。

2.7.2. 实时Spark Streaming消费示例

本文简单介绍如何使用Spark DataFrame API开发一个流式作业消费LogService数据。

示例代码

```
## StructuredLoghubWordCount.Scala
object StructuredLoghubSample {
 def main(args: Array[String]) {
  if (args.length < 7) {
   System.err.println("Usage: StructuredLoghubSample <logService-project> " +
    "<logService-store> <access-key-id> <access-key-secret> <endpoint> " +
    "<starting-offsets> <max-offsets-per-trigger> [<checkpoint-location>]")
   System.exit(1)
  }
  val Array(project, logStore, accessKeyId, accessKeySecret, endpoint, startingOffsets, maxOffsetsPerTrigge
r, outputPath, _*) = args
  val checkpointLocation =
   if (args.length > 8) args(8) else "/tmp/temporary-" + UUID.randomUUID.toString
  val spark = SparkSession
   .builder
   .appName("StructuredLoghubSample")
   .master("local[5]")
   .getOrCreate()
  import spark.implicits._
  // Create DataSet representing the stream of input lines from loghub
  val lines = spark
   .readStream
   .format("loghub")
   .option("sls.project", project)
   .option("sls.store", logStore)
   .option("access.key.id", accessKeyId)
   .option("access.key.secret", accessKeySecret)
   .option("endpoint", endpoint)
   .option("startingoffsets", startingOffsets)
   .option("zookeeper.connect.address", "localhost:2181")
   .option("maxOffsetsPerTrigger", maxOffsetsPerTrigger)
   .load()
   .selectExpr("CAST(content AS STRING)")
   .as[String]
  val query = lines.writeStream
   .format("parquet")
   .option("checkpointLocation", checkpointLocation)
   .option("path", outputPath)
   .outputMode("append")
   .trigger(Trigger.ProcessingTime(30000))
   .start()
  query.awaitTermination()
 }
}
```

> **说明** Maven pom文件可以参见aliyun-emapreduce-demo。

编译运行

```
## 编译命令
mvn clean package -DskipTests
## 编译完后，作业JAR包位于target/shaded/下。
## 提交执行
spark-submit --master yarn-cluster --executor-cores 2 --executor-memory 1g --driver-memory 1g
--num-executors 2--class x.x.x.StructuredLoghubSample xxx.jar <logService-project>
<logService-store> <access-key-id> <access-key-secret> <endpoint> <starting-offsets>
<max-offsets-per-trigger> <zookeeper-connect-address> <output-path> <checkpoint-location>
```

> **注意**
>
> - x.x.x.StructuredLoghubSample和xxx.jar需要替换成真实的类路径和包路径。
> - 作业资源需要根据实际数据规模和实际集群规模调整，如果集群太小，直接运行以上命令可能无法执行。

注意事项

由于Spark某些版本的不兼容性问题，emr_logservice sdk也存在默认EMR主版本运行时不兼容，具体的版本兼容性列表如下。

emr-logservice sdk版本	Spark版本	EMR主版本
1.6.0	2.3.1	EMR-3.18.x以下
1.7.0	2.4.3	EMR-3.19.x以上

2.7.3. 离线Spark消费示例

本文简单介绍如何使用Spark RDD API开发一个离线作业消费LogService数据。

示例代码

```
## TestBatchLoghub.Scala
object TestBatchLoghub {
 def main(args: Array[String]): Unit = {
  if (args.length < 6) {
   System.err.println(
    """Usage: TestBatchLoghub <sls project> <sls logstore> <sls endpoint>
     | <access key id> <access key secret> <output path> <start time> <end time=now>
    """.stripMargin)
   System.exit(1)
  }
  val loghubProject = args(0)
  val logStore = args(1)
  val endpoint = args(2)
  val accessKeyId = args(3)
  val accessKeySecret = args(4)
  val outputPath = args(5)
  val startTime = args(6).toLong
  val sc = new SparkContext(new SparkConf().setAppName("test batch loghub"))
  var rdd:JavaRDD[String] = null
  if (args.length > 7) {
   rdd = LoghubUtils.createRDD(sc, loghubProject, logStore, accessKeyId, accessKeySecret, endpoint, startT
ime, args(7).toLong)
  } else {
   rdd = LoghubUtils.createRDD(sc, loghubProject, logStore, accessKeyId, accessKeySecret, endpoint, startT
ime)
  }
  rdd.saveAsTextFile(outputPath)
 }
}
```

> **说明** Maven pom文件可以参见aliyun-emapreduce-demo。

编译运行

```
## 编译命令
mvn clean package -DskipTests
## 编译完后，作业JAR包位于target/shaded/下。
## 提交执行
spark-submit --master yarn-cluster --executor-cores 2 --executor-memory 1g --driver-memory 1g
--num-executors 2--class x.x.x.TestBatchLoghub xxx.jar <sls project> <sls logstore>
<sls endpoint> <access key id> <access key secret> <output path> <start time> [<end time=now>]
```

> **注意**
>
> - x.x.x.TestBatchLoghub和xxx.jar需要替换成真实的类路径和包路径。
> - 作业资源需要根据实际数据规模和实际集群规模调整，如果集群太小，直接运行以上命令可能无法执行。

2.8. Spark对接MNS

本文介绍如何通过Spark Streaming消费消息服务MNS（Message Notification Service）中的数据，并统计每个Batch内的单词个数。

Spark接入MNS

示例代码如下。

```
val conf = new SparkConf().setAppName("Test MNS Streaming")
  val batchInterval = Seconds(10)
  val ssc = new StreamingContext(conf, batchInterval)
  val queuename = "queuename"
  val accessKeyId = "<accessKeyId>"
  val accessKeySecret = "<accessKeySecret>"
  val endpoint = "http://xxx.yyy.zzzz/abc"
  val mnsStream = MnsUtils.createPullingStreamAsRawBytes(ssc, queuename, accessKeyId, accessKeySecret, endpoint,
    StorageLevel.MEMORY_ONLY)
  mnsStream.foreachRDD( rdd => {
   rdd.map(bytes => new String(bytes)).flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _).collect().foreach(e => println(s"word: ${e._1}, cnt: ${e._2}"))
  })
  ssc.start()
  ssc.awaitTermination()
```

支持MetaService

上面的示例是显式地将AccessKey传入到MNS中。从E-MapReduce SDK 1.3.2版本开始，Spark Streaming可以基于MetaService实现免AccessKey处理MNS数据。具体可以参见E-MapReduce SDK中的MnsUtils类说明。

```
MnsUtils.createPullingStreamAsBytes(ssc, queueName, endpoint, storageLevel)
MnsUtils.createPullingStreamAsRawBytes(ssc, queueName, endpoint, storageLevel)
```

附录

完整示例代码，请参见Spark接入MNS。

2.9. Spark对接HBase

本文介绍Spark如何写入数据至Hbase。

> **注意**
>
> 计算集群需要和Hbase集群处于一个安全组内，否则网络无法打通。在E-Mapreduce创建集群时，请注意选择Hbase集群所处的安全组。

Spark接入Hbase

代码如下。

```
object ConnectionUtil extends Serializable {
    private val conf = HBaseConfiguration.create()
    conf.set(HConstants.ZOOKEEPER_QUORUM,"ecs1,ecs1,ecs3")
    conf.set(HConstants.ZOOKEEPER_ZNODE_PARENT, "/hbase")
    private val connection = ConnectionFactory.createConnection(conf)
    def getDefaultConn: Connection = connection
  }
  //创建数据流unionStreams
  unionStreams.foreachRDD(rdd => {
   rdd.map(bytes => new String(bytes))
    .flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
    .mapPartitions {words => {
     val conn = ConnectionUtil.getDefaultConn
     val tableName = TableName.valueOf(tname)
     val t = conn.getTable(tableName)
     try {
      words.sliding(100, 100).foreach(slice => {
       val puts = slice.map(word => {
        println(s"word: $word")
        val put = new Put(Bytes.toBytes(word._1 + System.currentTimeMillis()))
        put.addColumn(COLUMN_FAMILY_BYTES, COLUMN_QUALIFIER_BYTES,
         System.currentTimeMillis(), Bytes.toBytes(word._2))
        put
       }).toList
       t.put(puts)
      })
     } finally {
      t.close()
     }
     Iterator.empty
    }}.count()
  })
  ssc.start()
  ssc.awaitTermination()
```

附录

完整示例代码，请参见Spark接入Hbase。

2.10. Spark对接Kafka

本文介绍如何在E-MapReduce的Hadoop集群运行Spark Streaming作业，处理Kafka集群的数据。

背景信息

因为E-MapReduce上的Hadoop集群和Kafka集群都是基于纯开源软件，所以在编程使用上参见相应官方文档即可。

- Spark官方文档：streaming-kafka-integration和structured-streaming-kafka-integration。
- E-MapReduce-demo：github地址。

访问Kerberos Kafka集群

E-MapReduce支持创建基于Kerberos认证的Kafka集群。当Hadoop集群作业需要访问Kerberos Kafka集群时，有以下两种使用方式：

- 非Kerberos Hadoop集群：提供用于Kafka集群的Kerberos认证的*kafka_client_jaas.conf*和*krb5.conf*文件。
- Kerberos Hadoop集群：基于Kerberos集群跨域互信，提供用于Hadoop集群的Kerberos认证的*kafka_client_jaas.conf*和*krb5.conf*文件。

  跨域互信详细信息，请参见跨域互信。

以上两种方式都需要运行作业时提供*kafka_client_jaas.conf*文件，用于Kerberos认证。

*kafka_client_jaas.conf*文件格式如下。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  serviceName="kafka"
  keyTab="/path/to/kafka.keytab"
  principal="kafka/emr-header-1.cluster-12345@EMR.12345.COM";
};
```

- *keytab*文件的获取，请参见兼容MIT Kerberos认证。
- *krb5.conf*文件，请从Kafka集群的*/etc/*目录获取。

Spark Streaming访问Kerberos Kafka集群

添加Kafka集群各个节点的长域名和IP信息至Hadoop集群各个节点的*/etc/hosts*中。长域名和IP信息，您可以在*/etc/hosts*中获取，长域名形式为 `emr-xxx-x.cluster-xxx` 。

当运行Spark Streaming作业访问Kerberos kafka时，可以在spark-submit命令行参数中提供所需的*kafka_client_jaas.conf*和*kafka.keytab*文件。

```
spark-submit --conf "spark.driver.extraJavaOptions=-Djava.security.auth.login.config={{PWD}}/kafka_client_jaas.conf -Djava.security.krb5.conf={{PWD}}/krb5.conf" --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config={{PWD}}//kafka_client_jaas.conf -Djava.security.krb5.conf={{PWD}}/krb5.conf" --files /local/path/to/kafka_client_jaas.conf,/local/path/to/kafka.keytab,/local/path/to/krb5.conf --class  xx.xx.xx.KafkaSample --num-executors 2 --executor-cores 2 --executor-memory 1g --master yarn-cluster xxx.jar arg1 arg2 arg3
```

*kafka_client_jaas.conf*文件中，keytab文件路径需要写相对路径，请严格按照如下keyTab配置项写法。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  serviceName="kafka"
  keyTab="kafka.keytab"
  principal="kafka/emr-header-1.cluster-12345@EMR.12345.COM";
};
```

附录

示例代码，请参见Spark对接Kafka。

2.11. Spark对接MySQL

本文介绍如何在E-MapReduce的Hadoop集群运行Spark作业、统计单词个数并将结果写入MySQL。Spark Streaming作业写入MySQL的方法与此类似。

Spark接入MySQL

示例代码如下。

```
val input = getSparkContext.textFile(inputPath, numPartitions)
  input.flatMap(_.split(" ")).map(x => (x, 1)).reduceByKey(_ + _)
   .mapPartitions(e => {
    var conn: Connection = null
    var ps: PreparedStatement = null
    val sql = s"insert into $tbName(word, count) values (?, ?)"
    try {
     conn = DriverManager.getConnection(s"jdbc:mysql://$dbUrl:$dbPort/$dbName", dbUser, dbPwd)
     ps = conn.prepareStatement(sql)
     e.foreach(pair => {
      ps.setString(1, pair._1)
      ps.setLong(2, pair._2)
      ps.executeUpdate()
     })
     ps.close()
     conn.close()
    } catch {
     case e: Exception => e.printStackTrace()
    } finally {
     if (ps != null) {
      ps.close()
     }
     if (conn != null) {
      conn.close()
     }
    }
   Iterator.empty
  }).count()
```

附录

完整示例代码，请参见Spark写入MySQL。

2.12. Spark对接DataHub

本文介绍如何在E-MapReduce的Hadoop集群，运行Spark Streaming作业消费DataHub数据、统计数据个数并打印出来。

Spark接入DataHub

- 准备工作

  使用DataHub的订阅功能订阅Topic，详细信息请参见DataHub订阅功能介绍。

- 消费DataHub数据

  运行Spark Streaming作业消费DataHub数据有两种使用方式：

  - 指定特定的ShardId，消费该ShardId的数据。

```
datahubStream = DatahubUtils.createStream(
        ssc,
        project, // DataHub Project
        topic, // DataHub topic
        subId, // DataHub的订阅ID
        accessKeyId,
        accessKeySecret,
        endpoint, // DataHub endpoint
        shardId, // DataHub Topic中的一个ShardId。
        read, // 处理DataHub数据的RecordEntry。
        StorageLevel.MEMORY_AND_DISK)
datahubStream.foreachRDD(rdd => println(rdd.count()))
// 取出RecordEntry中第一个Field的数据。
def read(record: RecordEntry): String = {
 record.getString(0)
}
```

  - 消费所有Shard的数据。

```
datahubStream = DatahubUtils.createStream(
        ssc,
        project, // DataHub Project
        topic, // DataHub topic
        subId, // DataHub的订阅ID
        accessKeyId,
        accessKeySecret,
        endpoint, // DataHub endpoint
        read, // 处理DataHub数据的RecordEntry
        StorageLevel.MEMORY_AND_DISK)
datahubStream.foreachRDD(rdd => println(rdd.count()))
// 取出RecordEntry中第一个Field的数据。
def read(record: RecordEntry): String = {
 record.getString(0)
}
```

附录

完整示例代码，请参见Spark对接DataHub。

2.13. Spark-Submit参数设置说明

本文介绍如何在E-MapReduce集群中设置Spark-Submit的参数。

集群配置

- 软件配置

E-MapReduce产品版本1.1.0

- Hadoop 2.6.0
- Spark 1.6.0

- 硬件配置
  - Master节点
    - 8核16 GiB 500 GB高效云盘
    - 1台
  - Worker节点 x 10 台
    - 8核16 GiB 500 GB高效云盘
    - 10台
  - 总资源：8核16 GiB（Worker）x 10 + 8核16 GiB（Master）

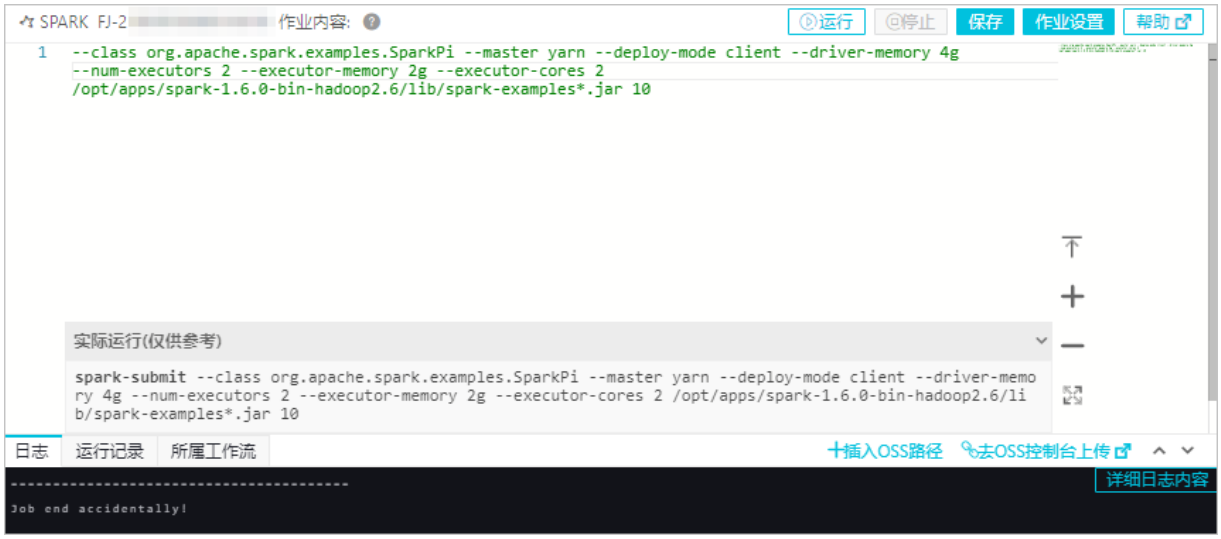
    > 注意 由于作业提交的时候资源只计算CPU和内存，所以这里磁盘的大小并未计算到总资源中。

  - Yarn可分配总资源：12核12.8 GiB（Worker）x 10

    > 注意 默认情况下，Yarn可分配核 = 机器核 x 1.5，Yarn可分配内存 = 机器内存 x 0.8。

提交作业

创建集群后，您可以提交作业。



上图所示的作业，因为直接使用了Spark官方的example包，所以不需要自己上传JAR包。

代码示例如下：

```
--class org.apache.spark.examples.SparkPi --master yarn --deploy-mode client --driver-memory 4g --num-executors 2 --executor-memory 2g --executor-cores 2 /opt/apps/spark-1.6.0-bin-hadoop2.6/lib/spark-examples*.jar 10
```

参数	说明
class	作业的主类。
master	E-MapReduce使用Yarn模式。
	等同于--master yarn —deploy-mode client， 此时不需要指定deploy-mode。
	等同于--master yarn —deploy-mode cluster， 此时不需要指定deploy-mode。
deploy-mode	client模式表示作业的AM会放在Master节点上运行。如果设置此参数，需要指定Master为yarn。
	cluster模式表示AM会随机的在Worker节点中的任意一台上启动运行。如果设置此参数，需要指定Master为yarn。
driver-memory	Driver使用的内存，不可超过单机的总内存。
num-executors	创建Executor的个数。
executor-memory	各个Executor使用的最大内存，不可以超过单机的最大可使用内存。
executor-cores	各个Executor使用的并发线程数目，即每个Executor最大可并发执行的Task数目。

资源计算

在不同模式、不同的设置下运行时，作业使用的资源情况如下表所示：

- yarn-client模式的资源计算

节点	资源类型	资源量（结果使用上面的例子计算得到）
master	core	1
	mem	driver-memroy = 4
worker	core	num-executors * executor-cores = 4
	mem	num-executors * executor-memory = 4

  - 作业主程序（Driver程序）会在Master节点上执行。按照作业配置将分配4 GB（由—driver-memroy指定）的内存给它（当然实际上可能没有用到）。
  - 会在Worker节点上起2个（由—num-executors指定）Executor，每一个Executor最大能分配2 GB（由—executor-memory指定）的内存，并最大支持2个（由—executor-cores指定）Task的并发执行。

- yarn-cluster模式的资源计算

节点	资源类型	资源量（结果使用上面的例子计算得到）
master	-	一个很小的Client程序，负责同步job信息，占用很小。
worker	core	num-executors * executor-cores+spark.driver.cores = 5
	mem	num-executors * executor-memory + driver-memroy = 8

> **说明** *spark.driver.cores*默认是1。

资源使用的优化

- yarn-client模式

  如果您的大作业，使用yarn-client模式，想要多用一些这个集群的资源，请参见如下配置。

  ```
  --master yarn-client --driver-memory 5g –-num-executors 20 --executor-memory 4g --executor-cores 4
  ```

  > **注意**
  > - Spark在分配内存时，会在用户设定的内存值上溢出375 MB或7%（取大值）。
  > - Yarn分配Container内存时，遵循向上取整的原则，这里也就是需要满足1 GB的整数倍。

  按照上述的资源计算公式：

  - Master的资源量为：
    - core：1
    - mem：6 GiB（5 GiB + 375 MB向上取整为6 GiB）
  - Worker的资源量为：
    - core：20*4 = 80
    - mem：20*5 GiB （4 GiB + 375 MB向上取整为5 GiB）= 100 GiB

  可以看到总的资源没有超过集群的总资源，那么遵循这个原则，您还可以有很多种配置，例如：

  ```
  --master yarn-client --driver-memory 5g --num-executors 40 --executor-memory 1g --executor-cores 2
  ```

  ```
  --master yarn-client --driver-memory 5g --num-executors 15 --executor-memory 4g --executor-cores 4
  ```

  ```
  --master yarn-client --driver-memory 5g --num-executors 10 --executor-memory 9g --executor-cores 6
  ```

  原则上，按照上述的公式计算出来的需要资源不超过集群的最大资源量就可以，但在实际场景中，因为系统、HDFS以及E-MapReduce的服务会需要使用Core和Mem资源，如果把Core和Mem都占用完了，反而会导致性能的下降，甚至无法运行。

  executor-cores数通常也都会被设置成和集群的可使用核一致，因为如果设置的太多，CPU会频繁切换，性能并不会提高。

- yarn-cluster模式

  当使用yarn-cluster模式后，Driver程序会被放到Worker节点上。会占用到Worker资源池里的资源，这时若想要多用一些这个集群的资源，请参见如下配置。

  ```
  --master yarn-cluster --driver-memory 5g --num-executors 15 --executor-memory 4g --executor-cores 4
  ```

配置建议

- 如果将内存设置的很大，要注意GC所产生的消耗。通常推荐每一个Executor的内存<=64 GB。
- 如果是进行HDFS读写的作业，建议每个Executor中使用<=5个并发来读写。
- 如果是进行OSS读写的作业，建议是将Executor分布在不同的ECS上，这样可以将每一个ECS的带宽都用上。例如，有10台ECS，那么就可以配置num-executors=10，并设置合理的内存和并发。
- 如果作业中使用了非线程安全的代码，则在设置executor-cores的时候需要注意多并发是否会造成作业的不正常。如果会造成作业不正常，推荐设置executor-cores=1。

3.Spark Streaming SQL

3.1. 简介

本文介绍Spark SQL流式处理中关键字常用类型和使用关键字字符的方法。

背景信息

从EMR-3.21.0版本开始提供Spark Streaming SQL的功能，支持使用SQL来开发流式分析作业。

Spark Streaming SQL基于Spark Structured Streaming开发完成，所有语法功能和使用限制遵循Spark Structured Streaming。

关键字常用类型

常用类型	关键字
DDL	CREATE TABLE、CREATE TABLE AS SELECT、CREATE SCAN、CREATE STREAM
DML	INSERT INTO、MERGE INTO
SELECT子句	SELECT FROM、WHERE、GROUP BY 、JOIN、UNION ALL

使用关键字字符

如果您需要使用关键字字符作为字段名称，请在关键字两端添加撇号（`），例如 `value` 。

3.2. 流式查询

3.2.1. 作业模板（EMR-3.23.0及之后版本）

本文为您介绍如何在EMR-3.23.0及之后版本中，使用Spark SQL开发一个流式处理作业。

作业模板

```
-- dbName：数据库名。
CREATE DATABASE IF NOT EXISTS ${dbName};
USE ${dbName};
-- 创建Log Service数据表。
-- slsTableName：Log Service表的名称。
-- logProjectName：LogService的项目名。
-- logStoreName：LogService的logstore名。
-- accessKeyId：阿里云账号的AccessKey ID。
-- accessKeySecret：阿里云账号的AccessKey Secret。
-- endpoint：LogService的logstore所在的Endpoint。
-- 当显式指定LogStore的各个字段时，必须定义为STRING类型。
-- 保留6个系统字段： (`__logProject__` STRING, `__logStore__` STRING, `__shard__` INT, `__time__` TIM
ESTAMP, `__topic__` STRING, `__source__` STRING)
CREATE TABLE IF NOT EXISTS ${slsTableName} (col1 dataType[, col2 dataType])。
USING loghub
OPTIONS (
sls.project = '${logProjectName}',
sls.store = '${logStoreName}',
access.key.id = '${accessKeyId}',
access.key.secret = '${accessKeySecret}',
endpoint = '${endpoint}');
-- 创建HDFS数据表，需要完成表的列字段定义。
-- hdfsTableName：HDFS表的名称。
-- location: 存储数据路径，支持HDFS和OSS路径。
-- 数据格式支持：delta, csv, json, orc, parquet等，默认为delta。
CREATE TABLE IF NOT EXISTS ${hdfsTableName} (col1 dataType[, col2 dataType])
USING delta
LOCATION '${location}';
-- 配置读表的方式，支持STREAM和BATCH，默认为BATCH。
CREATE SCAN tmp_read_sls_table
ON ${slsTableName}
USING STREAM;
-- 创建一个流式查询作业。
CREATE STREAM ${queryName}
OPTIONS(
outputMode='Append',
triggerType='ProcessingTime',
triggerInterval='30000',
checkpointLocation='${checkpointLocation}')
INSERT INTO ${hdfsTableName}
SELECT col1, col2
FROM tmp_read_sls_table
WHERE ${condition};
```

> **说明** Endpoint详细信息请参见访问域名和数据中心。

3.2.2. 作业模板

本文为您介绍如何使用Spark SQL开发一个流式处理作业。

> **说明** ：EMR-3.23.0（含）后续版本已不建议使用这个模板，但仍然会支持。

查询语句块

类似 `streaming.query.name` 等作业参数无法通过SQL表达，因此需要在SQL查询语句前使用SET进行必要的参数配置。

合法的查询语句块如下。

```
SET streaming.query.name=${queryName};
queryStatement
```

作业模板

```
-- 创建数据库。
-- dbName：数据库名。
CREATE DATABASE IF NOT EXISTS ${dbName};
USE ${dbName};
-- 创建Log Service数据表。
-- slsTableName：Log Service表的名称。
-- logProjectName：LogService的项目名。
-- logStoreName：LogService的LogStore名。
-- accessKeyId：阿里云账号的AccessKey ID。
-- accessKeySecret：阿里云账号的AccessKey Secret。
-- endpoint：LogService的LogStore所在Endpoint。
-- 当显式指定LogStore的各个字段时，必须定义为STRING类型。
-- 保留6个系统字段：（`__logProject__` STRING, `__logStore__` STRING, `__shard__` INT, `__time__` TIM
ESTAMP, `__topic__` STRING, `__source__` STRING)
CREATE TABLE IF NOT EXISTS ${slsTableName} (col1 dataType[, col2 dataType])
USING loghub
OPTIONS (
sls.project = '${logProjectName}',
sls.store = '${logStoreName}',
access.key.id = '${accessKeyId}',
access.key.secret = '${accessKeySecret}',
endpoint = '${endpoint}');
-- 创建HDFS数据表，需要完成表的列字段定义。
-- hdfsTableName：HDFS表的名称。
-- location: 存储数据路径，支持HDFS和OSS路径。
-- 数据格式支持：delta, csv, json, orc, parquet等，默认为delta。
CREATE TABLE IF NOT EXISTS ${hdfsTableName} (col1 dataType[, col2 dataType])
USING delta
LOCATION '${location}';
-- 需要为每个流式查询定义一些运行参数。
-- streaming.query.name: 流式查询作业名称。
-- spark.sql.streaming.checkpointLocation.${queryName}：本次流式查询作业的Checkpoint路径。
SET streaming.query.name=${queryName};
SET spark.sql.streaming.query.options.${queryName}.checkpointLocation=${checkpointLocation};
-- 以下为可选参数。
-- outputMode：查询结果输出方式，默认为append。
-- trigger：查询执行模式，可选ProcessingTime，默认为ProcessingTime。
-- trigger.intervalMs：批次间隔，单位毫秒，默认为0。
-- SET spark.sql.streaming.query.outputMode.${queryName}=${outputMode};
SET spark.sql.streaming.query.trigger.${queryName}=ProcessingTime;
SET spark.sql.streaming.query.trigger.intervalMs.${queryName}=30;
INSERT INTO ${hdfsTableName}
SELECT col1, col2
FROM ${slsTableName}
WHERE ${condition};
```

> **说明** Endpoint请参见服务入口。

参数

以下列出几个关键性参数。

参数	说明	默认值
streaming.query.name	查询名。	必须显式配置。
spark.sql.streaming.query.options.${queryName}.checkpointLocation	流式作业的Checkpoint路径。	必须显式配置。
spark.sql.streaming.query.outputMode.${queryName}	查询的Output模式。	append
spark.sql.streaming.query.trigger.${queryName}	查询执行模式，当前只支持ProcessingTime模式。	ProcessingTime
spark.sql.streaming.query.trigger.intervalMs.${queryName}	查询批次间隔时间，单位毫秒。	0

3.2.3. 配置说明

本文介绍流式查询配置的相关概念及配置参数。

查询配置

> **说明** 流式查询配置在EMR-3.23.0及之后版本不建议使用，最新的查询配置请参见SCAN语句或STREAM语句。

在使用Spark SQL进行流式查询前，您需要了解以下两个概念：

- 数据源配置：即Table的定义。

  Table的定义只包含数据源的配置，例如，Kafka数据源的连接地址和Topic名等。因为您可以对一个Table同时做多个业务无关的查询，所以Table定义中不应该包含具体的查询实例的运行配置。

- 查询实例配置：具体每个Stream Query运行时的参数配置。

  每一个查询实例均需要单独配置。通过queryName，可以减少对查询SQL进行不必要的修改。查询实例参数设置使用的是 `SET` 语法，详情请参见配置说明。

查询实例包括：

- `INSERT INTO ...`
- `CREATE TABLE ... AS SELECT ...`

每个查询实例的queryName均为SQL上下文中最近的一个，查询示例说明如下：

- 情况一

```
SET streaming.query.name=one_test_job
-- query 1
INSERT INTO tb_test_1 SELECT ...
-- query 2
INSERT INTO tb_test_2 SELECT ...
-- 以上query1和query2的queryName都是"one_test_job"，当然这是一种非法情况，每个查询实例必须有唯一queryName。
```

- 情况二

```
SET streaming.query.name=one_test_job_1
SET streaming.query.name=one_test_job_2
-- query 1
CREATE TABLE tb_test_1 AS SELECT ...
-- query1的queryName是"one_test_job_2"。
```

配置说明

配置类别	对应于DataFrame API	SQL配置格式	说明	是否必选
queryName	writeStream.queryName(...)	SET streaming.query.name=$queryName	每个Stream Query的名称，各个Query的配置项会根据名称来区分。	是
option	writeStream.option(...)	SET spark.sql.streaming.query.options.$queryName.$optionName=$optionValue	checkpointLocation: checkpoint目录。	是
			自定义。	否
outputMode	writeStream.outputMode(...)	SET spark.sql.streaming.query.outputMode.$queryName=$outputMode	output模式，默认为append模式。	否
trigger	writeStream.trigger(...)	SET spark.sql.streaming.query.trigger.$queryName=$triggerType	trigger模式，默认为ProcessingTime。	否
		SET spark.sql.streaming.query.trigger.intervalMs.$queryName=$intervalMs	trigger间隔，单位毫秒，默认为0。	否

3.3. DDL概述

3.3.1. 建表语句

本文为您介绍Spark SQL建表语法。

语法

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING providerName
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

使用CTAS语句建表语法如下，将创建表和查询结果写入表的语句合并到一起。

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING providerName
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*)
AS
queryStatement;
```

说明

从语法中可以看出，表的字段信息是可选的。

```
CREATE TABLE kafka_table
USING kafka
OPTIONS (
kafka.bootstrap.servers = "${BOOTSTRAP_SERVERS}",
subscribe = "${TOPIC_NAME}");
```

3.3.2. SCAN语句

EMR-3.23.0及后续版本支持SCAN语法。

背景信息

SCAN语法支持DataSource V2设计，实现在Spark SQL中的统一批流查询。例如，您可以定义一个Kafka数据源表，然后定义两个SCAN，分别对应批式读和流式读。

SCAN语法定义Table时只需要定义Table数据源的基本信息，无需定义如何读这个Table的参数。SCAN语法约束如下：

- SCAN语法定义的视图，仅能用作数据源表，不可以作为数据输出表。
- SCAN语法定义可以直接处理原始表，但仅能进行批式读。

语法

```
CREATE SCAN tbName_alias
ON tbName
USING queryType
OPTIONS (propertyName=propertyValue[,propertyName=propertyValue]*)
```

queryType支持两种类型：

- BATCH：将源表tbName按照批读的方式，定义一个临时视图tbName_alias。
- STREAM：将源表tbName按照流读的方式，定义一个临时视图tbName_alias。

OPTIONS中可以定义数据源读取的运行期参数，针对不同的数据源，运行期参数不同。运行期参数主要包含如何定义流式读的行为参数。

示例

1. 创建一个LogService数据源表。

```
spark-sql> CREATE TABLE loghub_table_intput_test(content string)
    > USING loghub
    > OPTIONS
    > (...)
```

2. 离线处理SLS数据，统计截止当前数据条数。

```
spark-sql> CREATE SCAN loghub_table_intput_test_batch
    > ON loghub_table_intput_test
    > USING BATCH;
spark-sql> SELECT COUNT(*) FROM loghub_table_intput_test_batch;
```

3. 流式处理SLS数据。

```
spark-sql> CREATE TABLE loghub_table_output_test(content string)
    > USING loghub
    > OPTIONS
    > (...);
spark-sql> CREATE SCAN loghub_table_intput_test_stream
    > ON loghub_table_intput_test
    > USING STREAM;
```

测试非法操作：例如对流表进行Select。

```
spark-sql> SELECT COUNT(*) FROM loghub_table_test_stream;
```

报错如下所示。

```
Error in query: Queries with streaming sources must be executed with writeStream.start();
```

正确对流表进行Select语句如下。

```
spark-sql> INSERT INTO loghub_table_output_test SELECT content FROM loghub_table_intput_test_st
ream;
```

3.3.3. STREAM语句

EMR-3.23.0版本开始支持STREAM语法。

背景信息

E-MapReduce支持SET和STREAM两种方法配置WriteStream参数，推荐使用STREAM方法配置WriteStream的必要参数，包括checkpointLocation、outputMode、triggerType和triggerIntervalMs。

语法

```
CREATE STREAM queryName
OPTIONS (propertyName=propertyValue[,propertyName=propertyValue]*)
INSERT INTO tbName
queryStatement;
```

以下列出WriteStream必要的参数。

参数名	说明	默认值
checkpointLocation	流式查询作业的checkpoint路径。	无
outputMode	流式查询的输出模式。	Append
triggerType	流式查询的执行模式。	ProcessingTime
triggerIntervalMs	流式查询的执行间隔，单位毫秒。	0

示例

```
CREATE STREAM job1
OPTIONS(
checkpointLocation='/tmp/spark',
outputMode='Append',
triggerType='ProcessingTime'
triggerIntervalMs='3000')
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
```

3.4. DML概述

3.4.1. MERGE INTO

本文为您介绍如何在Spark SQL流式处理中使用MERGE INTO语句。

语法

```
mergeInto
  : MERGE INTO target=tableIdentifier tableAlias
    USING (source=tableIdentifier (timeTravel)? | '(' subquery = query ')') tableAlias
    mergeCondition?
    matchedClauses*
    notMatchedClause?
mergeCondition
  : ON condition=booleanExpression
matchedClauses
  : deleteClause
  | updateClause
notMatchedClause
  : insertClause
deleteClause
  : WHEN MATCHED (AND deleteCond=booleanExpression)? THEN deleteAction
  | WHEN deleteCond=booleanExpression THEN deleteAction
updateClause
  : WHEN MATCHED (AND updateCond=booleanExpression)? THEN updateAction
  | WHEN updateCond=booleanExpression THEN updateAction
insertClause
  : WHEN NOT MATCHED (AND insertCond=booleanExpression)? THEN insertAction
  | WHEN insertCond=booleanExpression THEN insertAction
```

示例

- 源表和目标表定义。

```
-- source table
source_table
  : id int,      --主键
  | name string,
  | opType string   --数据操作类型
-- target table
target_table
  : id int,      --主键
  | name string
```

- Merge Into Delta。

```
MERGE INTO target_table t
USING source_table s
ON s.id = t.id
WHEN MATCHED AND s.opType = 'delete' THEN DELETE
WHEN MATCHED AND s.opType = 'update' THEN UPDATE SET id = s.id, name = s.name
WHEN NOT MATCHED AND s.opType = 'insert' THEN INSERT (key, value) VALUES (key, value)
```

- Merge Into Kudu。

```
MERGE INTO target_table t
USING source_table s
WHEN s.opType = 'delete' THEN DELETE
WHEN s.opType = 'update' THEN UPDATE SET *
WHEN s.opType = 'insert' THEN INSERT *
```

说明

MERGE INTO语句需要遵循以下规则：

- 最多支持3个子句。
- 当支持WHEN NOT MATCHED子句时，最多支持2个WHEN子句，并且WHEN NOT MATCHED子句必须放在最后。

MERGE INTO的子句需要遵循以下规则：

- WHEN MATCHED子句：
  - 最多有一个UPDATE或者DELETE操作。
  - 每个子句可以有一个可选的condition。如果有两个WHEN MATCHED子句时，则第一个WHEN MATCHED必须写上condition。
  - WHEN MATCHED子句有顺序关系，且每条数据只能执行一个WHEN MATCHED子句。
  - 支持 `UPDATE SET *` 写法，表示 `UPDATE SET column1 = source.column1 [, column2 = source.column2 ...]` 。如果源表和目标表的字段个数不一致，则抛出解析失败异常。
- WHEN NOT MATCHED子句：
  - 仅能使用INSERT操作，且可以有一个可选的condition。
  - 支持 `INSERT *` 写法，表示 `INSERT (column1 [, column2 ...]) VALUES (source.value1 [, source.value2 ...])` 。如果源表和目标表的字段个数不一致，则抛出解析失败异常。

对于支持UPSERT操作的目标存储系统，可以基于MERGE的变种来实现。支持情况如下：

- 最多支持3个WHEN子句。
- `mergeCondition`：无需配置Merge条件。
- 可以省略`MATCHED`和`NOT MATCHED`关键字，直接根据condition操作。

3.4.2. INSERT INTO

本文为您介绍如何在Spark SQL流式处理中使用INSERT INTO语句。

语法

```
INSERT INTO tbName[(columnName[,columnName]*)]
queryStatement;
```

示例

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
```

说明

- 不支持单独的SELECT查询，必须有CTAS或者在`INSERT INTO ...`内才能操作。
- 单个作业支持在一个SQL文件里面包含多个DML操作，允许包含多个数据源和多个数据目标端。

3.5. 查询概述

3.5.1. SELECT语句

SELECT语句用于从表中选取数据。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

测试数据

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

示例一

- 测试语句

```
SELECT * FROM 表名;
```

- 测试结果

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

示例二

- 测试语句

```
SELECT a, c AS d FROM 表名;
```

- 测试结果

a（VARCHAR）	d（DATE）
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

示例三

- 测试语句

```
SELECT DISTINCT a FROM 表名;
```

- 测试结果

a（VARCHAR）
a1
b1
c1

子查询

普通的SELECT是从几张表中读数据，如 `SELECT column_1, column_2 … FROM table_name` 。

> **说明** 当查询的对象是另一个SELECT操作时，必须为子查询加别名。

- 示例语句

```
INSERT INTO result_table
SELECT * FROM
        (SELECT  t.a,
             sum(t.b) AS sum_b
         FROM   t1 t
         GROUP BY t.a
        ) t1
WHERE t1.sum_b > 100;
```

- 示例结果

a（VARCHAR）	b（INT）
a1	211
b1	120
a1	257

3.5.2. WHERE语句

WHERE语句可用于对SELECT语句中的数据进行筛选。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

示例

- 测试数据

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	shanghai

- 测试语句

```
SELECT * FROM XXXX WHERE City='Beijing';
```

- 测试结果

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

3.5.3. GROUP BY语句

GROUP BY语句用于根据一个或多个列对结果集进行分组。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ GROUP BY { groupItem [, groupItem ]* } ];
```

示例

- 测试数据

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM xxx
GROUP BY Customer;
```

- 测试结果

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

3.5.4. JOIN语句

E-MapReduce的JOIN和传统批处理JOIN的语义一致，都用于将两张表关联起来。

语法

```
tableReference [, tableReference ]* | tableexpression
[ joinType ] JOIN tableexpression [ joinCondition ];
```

参数描述如下：

- tableReference：表名称。
- tableexpression：表达式。
- joinCondition：JOIN条件。

约束

当执行流数据的JOIN操作时，部分JOIN类型是不支持的，具体请参见Spark官方文档说明。

左表	右表	JOIN类型	是否支持
流式表	静态表	内连接	支持
		左连接	支持
		右连接	不支持
		全连接	不支持
静态表	流式表	内连接	支持
		左连接	不支持
		右连接	支持
		全连接	不支持
流式表	流式表	内连接	支持
		左连接	支持
		右连接	支持
		全连接	不支持

3.5.5. UNION ALL语句

UNION ALL语句将两个流式数据合并。两个流式数据的字段完全一致，包括字段类型和字段顺序。

语法

```
select_statement
UNION ALL
select_statement;
```

示例

- 测试数据

  表1：test_source_union1

a（varchar）	b（bigint）	c（bigint）
test1	1	10

  表2：test_source_union2

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20

  表3：test_source_union3

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20
test1	1	10

- 测试语句

```
SELECT
  a,
  sum(b),
  sum(c)
FROM
  (SELECT * from test_source_union1
  UNION ALL
  SELECT * from test_source_union2
  UNION ALL
  SELECT * from test_source_union3
  )t
 GROUP BY a;
```

- 测试结果

d（varchar）	e（bigint）	f（bigint）
test1	1	10
test2	2	20
test1	2	20
test1	3	30

d（varchar）	e（bigint）	f（bigint）
test2	4	40
test1	4	40

3.5.6. WATERMARK语句

WATERMARK语句在流式查询中用来处理数据乱序问题。本文介绍WATERMARK语法及相关的示例。

语法

```
SELECT watermark(projectItem, durationSpec) as watermarkItem, projectItem [, projectItem ]*
FROM tableExpression
```

WATERMARK主要是为了解决数据流场景中常见的数据延迟问题。Spark在Aggregate和Join计算过程中，计算引擎会维护中间State数据。引入WATERMARK特性可以在数据出现延迟时，正确地丢弃延迟超时的数据，并且自动地从内存中丢弃过期的中间State数据，这个对流式查询的稳定运行至关重要。

更多WATERMARK的详细信息请参见如下信息：

- Aggregate+Watermark
- Join+Watermark
- 多Watermark策略

示例

stream_source表需要有一个TIMESTAMP类型的表示Event time的字段，例如数据延迟最大为10秒。Aggregate+Watermark和Join+Watermark示例如下：

- Aggregate+Watermark

```
SELECT watermark(ts, interval 10 seconds), count(*) as col1, col2
FROM stream_source
GROUP BY ts, col2
```

- Join+Watermark

```
SELECT
  watermark(stream_source_1.ts, interval 20 second) as ts1,
  watermark(stream_source_2.ts, interval 10 second) as ts2,
  stream_source_1.col1 as col1,
  stream_source_2.col2 as col2
FROM
stream_source_1
INNER JOIN
stream_source_2
ON stream_source_1.col1=stream_source_2.col1
AND ts1 >= ts2
AND ts1 <= ts2 + interval 10 seconds
```

3.6. 窗口函数

3.6.1. 概述

本文介绍Spark SQL流式处理支持的窗口函数及其时间属性。

窗口函数

窗口函数是对一个特定窗口的聚合。例如，您可以通过定义窗口来收集过去1分钟内某网站的用户点击量，并对这个窗口内的数据进行计算。Spark SQL流式处理支持两类窗口：

- 滚动窗口（TUMBLING）
- 滑动窗口（HOPPING）

时间属性

Spark SQL支持Event Time时间属性，对数据进行窗口内聚合。

Event Time：事件时间，通常是您提供在Schema中数据最原始的创建时间。

说明

查询已存在的时间窗口时，窗口函数自动生成 `window` 列，包含窗口的起止时间信息，即 `window.start` 和 `window.end` 。

3.6.2. 滚动窗口

本文为您介绍如何使用Spark SQL流式处理中的滚动窗口函数。

什么是滚动窗口

滚动窗口（TUMBLING）将每个元素分配到一个指定大小的窗口中。通常滚动窗口有一个固定的大小，并且不会出现重叠。例如，如果指定了一个5分钟大小的滚动窗口，无限流的数据会根据时间划分成[0:00 - 0:05)、 [0:05, 0:10)和[0:10, 0:15)等窗口。

滚动窗口函数语法

```
GROUP BY TUMBLING (colName, windowDuration)
```

示例

```
SELECT avg(inv_quantity_on_hand) qoh
FROM kafka_inventory
GROUP BY TUMBLING (inv_data_time, interval 1 minute)
```

3.6.3. 滑动窗口

本文介绍如何使用Spark SQL流式处理中的滑动窗口函数。

什么是滑动窗口

滑动窗口（HOPPING），也被称作Sliding Window。不同于滚动窗口，滑动窗口的窗口可以重叠。

滑动窗口有windowDuration和slideDuration两个参数。windowDuration为窗口的大小，slideDuration为每次滑动的步长，两者关系如下：

- slideDuration < windowDuration：则窗口会重叠，每个元素会被分配到多个窗口。
- slideDuration = windowDuration：则等同于滚动窗口（TUMBLING）。

滑动窗口函数语法

```
GROUP BY HOPPING ( colName, windowDuration, slideDuration )
```

示例

```
SELECT avg(inv_quantity_on_hand) qoh
FROM kafka_inventory
GROUP BY HOPPING (inv_data_time, interval 1 minute, interval 30 second)
```

3.7. 内建函数

3.7.1. DTS_BINLOG_PARSER

DTS_BINLOG_PARSER用于解析数据传输服务DTS（Data Transmission Service）传输的Binlog数据。

背景信息

> **注意** 当前只支持解析DTS同步的RDS Binlog数据。

DTS_BINLOG_PARSER解析结果是STRUCT类型，包含字段如下。

```
struct
 : recordID long,          -- binlog的RecordId。
 | source string,          -- 数据源信息，包括数据库类型和版本等。
 | dbTable string,          -- 数据表名。
 | recordType string,       -- 操作类型，包括INSERT、DELETE、UPDATE和INIT。
 | recordTimestamp timestamp, -- 记录的时间戳。
 | extraTags string,        -- 记录的属性信息，例如primary key和unique key等。
 | fields string,           -- 数据表的Schema信息。
 | beforeImages string,     -- 本记录生成前的记录数据。
 | afterImages string       -- 本记录生成后的记录数据。
```

语法

```
SELECT DTS_BINLOG_PARSER(column)
FROM table
```

示例

```
SELECT
recordID,
source,
dbTable,
recordType,
recordTimestamp,
extraTags,
fields,
beforeImages,
afterImages
FROM(
 SELECT DTS_BINLOG_PARSER(value) as (recordID, source, dbTable, recordType, recordTimestamp, extraTa
gs, fields, beforeImages, afterImages)
 FROM dts_binlog_table
)
```

3.8. 数据源

3.8.1. 数据源支持概述

从EMR-3.21.0版本开始支持使用Spark SQL开发流式分析作业。本文介绍Spark SQL支持的数据源类型，以及支持数据源的方式。

支持的数据源

数据源	批量读	批量写	流式读	流式写
Kafka	有	无	有	有
Loghub	有	有	有	有
Tablestore	有	有	有	有
DataHub	无	无	有	有
HBase	有	有	无	有
JDBC	有	有	无	有
Druid	无	无	无	有
Redis	无	无	无	有
Kudu	有	有	无	有
DTS	有	无	有	无

支持数据源的方式

Spark SQL支持数据源的方式包括以下两种：

- 命令行方式

i. 下载预编译好的数据源JAR包。

您只需要使用该JAR包，就可以完成Loghub、TableStore、HBase、JDBC和Redis数据源的实现以及相关的依赖包。Kafka和Druid数据源暂未在此JAR包中发布，后续会陆续加入，具体请参见Release Notes。

ii. 使用streaming-sql命令行进行交互式开发。

```
[hadoop@emr-header-1 ~]# streaming-sql --master yarn-client --jars emr-datasources_shaded_2.11-${version}.jar --driver-class-path emr-datasources_shaded_2.11-${version}.jar
```

- 工作流方式

详情请参见Streaming SQL作业配置。

3.8.2. Kafka数据源

本文介绍如何使用Kafka数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING kafka
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

配置参数说明

参数名	说明	是否必选
subscribe	关联的Kafka Topic名称。	是
kafka.bootstrap.servers	Kafka集群连接地址。	是

相关文档

Kafka数据源详细文档，请参见Structured Streaming + Kafka Integration Guide。

3.8.3. Loghub数据源

本文介绍如何使用Loghub数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName(columnName dataType [,columnName dataType]*)
USING loghub
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table Schema

创建Loghub表时，必须显式地定义表的字段信息。自定义Schema字段名称要和SLS日志中key的名称相同。

```
spark-sql> CREATE TABLE loghub_table_test(content string)
      > USING loghub
      > OPTIONS(
      > endpoint="sls.aliyuncs.com",
      > access.key.id="yHiu*******BG2s",
      > access.key.secret="ABctuw0M***************iKKljZy",
      > sls.project="test",
      > sls.store="myInstance");
spark-sql> DESC loghub_table_test;
content  string  NULL
Time taken: 0.436 seconds, Fetched 1 row(s)
```

不带Schema时如下。

```
spark-sql> CREATE TABLE loghub_table_test
      > USING loghub
      > OPTIONS
      > (...)
spark-sql> DESC loghub_table_test;
__logProject__ string  NULL
__logStore__ string NULL
__shard__ string NULL
__time__ string NULL
__topic__ string NULL
__source__ string NULL
__value__ string NULL
__sequence_number__ string NULL
Time taken: 0.436 seconds, Fetched 1 row(s)
```

配置参数说明

参数名	说明	是否必选
endpoint	LogService API Endpoint。	是
access.key.id	您阿里云账号的AccessKey ID。	是
access.key.secret	您阿里云账号的AccessKey Secret。	是
sls.store	LogService项目名。	是
sls.project	LogStore名。	是

对于EMR SDK 2.0.0以及以上版本，Loghub Schema为可选项，参数说明如下。

参数	类型	说明
__logProject__	String	LogStore名。
__logStore__	String	LogService项目名。

参数	类型	说明
__shard__	String	Logstore的Shard。
__time__	String	该记录日志时间。
__topic__	String	日志主题。
__source__	String	日志服务的来源IP。
__value__	String	日志服务的内容。JSON格式。
__sequence_number__	String	记录序列号（通过 appendSequenceNumber='true' 开启该字段，默认为NULL）。

3.8.4. HBase数据源

本文介绍如何使用HBase数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName
USING hbase
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table Schema

创建HBase表时，无需显式地定义表的字段信息，示例如下所示。

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS hbase_table_test;
spark-sql> CREATE TABLE hbase_table_test
     > USING hbase
     > OPTIONS(
     > catalog='{"table":{"namespace":"default","name":"test"},"rowkey":"key", "columns":{"key":{"cf":"rowkey", "col":"key", "type":"string"},"data":{"cf":"info", "col":"data", "type":"string"}}}',
     > hbaseConfiguration='{"hbase.zookeeper.quorum":"a.b.c.d:2181"}');
spark-sql> DESC hbase_table_test;
key string  NULL
data  string  NULL
Time taken: 0.436 seconds, Fetched 2 row(s)
```

配置参数说明

参数名	说明	是否必选
catalog	HBase表字段说明，JSON格式。	是

参数名	说明	是否必选
hbaseConfiguration	HBase配置，JSON格式。例如配置HBase连接地址，{"hbase.zookeeper.quorum":"a.b.c.d:2181"}。	是

本示例定义了一个HBase表table1的Schema，Catalog配置示例如下所示。

```
{
 "table":{"namespace":"default", "name":"table1"},
 "rowkey":"key",
 "columns":{
  "col0":{"cf":"rowkey", "col":"key", "type":"string"},
  "col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
  "col2":{"cf":"cf2", "col":"col2", "type":"double"},
  "col3":{"cf":"cf3", "col":"col3", "type":"float"},
  "col4":{"cf":"cf4", "col":"col4", "type":"int"},
  "col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
  "col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
  "col7":{"cf":"cf7", "col":"col7", "type":"string"},
  "col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
 }
}
```

3.8.5. JDBC数据源

本文介绍如何使用JDBC数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName
USING jdbc2
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table Schema

创建JDBC表时，无需显式地定义表的字段信息，示例如下所示。

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS rds_table_test;
spark-sql> CREATE TABLE rds_table_test
      > USING jdbc2
      > OPTIONS (
      > url="jdbc:mysql://rm-bp11*********i7w9.mysql.rds.aliyuncs.com:3306/default?useSSL=true",
      > driver="com.mysql.jdbc.Driver",
      > dbtable="test",
      > user="root",
      > password="thisisapassword",
      > batchsize="100",
      > isolationLevel="NONE");
spark-sql> DESC rds_table_test;
id  int NULL
name  string  NULL
Time taken: 0.413 seconds, Fetched 2 row(s)
```

配置参数说明

参数名	说明	是否必选
url	数据库地址。	是
driver	数据库连接的JDBC驱动。例如 com.mysql.jdbc.Driver”eper. quorum":"a.b.c.d:2181"} 。	是
dbtable	数据表名称。	是
user	连接的用户名。	是
password	连接的密码。	是
batchsize	每个批次更新的数据条数。 仅向数据库写入数据时生效。	否
isolationLevel	事务隔离级别，默认值为READ_UNCOMMITTED。	否

事务隔离级别详情如下。

事务隔离级别	脏读	不可重复读	幻读
READ_UNCOMMITTED	是	是	是
READ_COMMITTED	否	是	是
REPEATABLE_READ	否	否	是
SERIALIZABLE	否	否	否

事务隔离级别	脏读	不可重复读	幻读
NONE	无	无	无

写入数据

当您需要向数据库中写入数据时，可以通过以下命令设置一个关联的SQL Statement来实现。

```
spark-sql> SET streaming.query.${queryName}.sql=insert into `test` (`id`,`name`) values(?, ?);
spark-sql> SET ...
spark-sql> INSERT INTO rds_table_test SELECT ...
```

3.8.6. TableStore数据源

本文介绍如何使用TableStore数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName
USING tablestore
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table Schema

创建TableStore表时，无需显式定义表的字段信息，示例如下。

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS ots_table_test;
spark-sql> CREATE TABLE ots_table_test
      > USING tablestore
      > OPTIONS(
      > endpoint="http://xxx.cn-hangzhou.vpc.ots.aliyuncs.com",
      > access.key.id="yHiu*******BG2s",
      > access.key.secret="ABctuw0M***************iKKljZy",
      > table.name="test",
      > instance.name="myInstance",
      > batch.update.size="100",
      > catalog='{"columns":{"pk":{"col":"pk","type":"string"},"data":{"col":"data","type":"string"}}}');
spark-sql> DESC ots_table_test;
pk string NULL
data string NULL
Time taken: 0.501 seconds, Fetched 2 row(s)
```

配置参数说明

参数名	说明
access.key.id	阿里云AccessKey ID。

参数名	说明
access.key.secret	阿里云AccessKey Secret。
endpoint	TableStore API Endpoint。
table.name	TableStore的表名。
instance.name	TableStore的实例名。
batch.update.size	表示每次向TableStore批量更新的数据条数，默认为0表示逐条更新。 仅向数据库写入数据时生效。
catalog	TableStore表字段说明，JSON格式。

本示例定义了一个TableStore表table1的Schema，Catalog配置示例如下所示。

```
{"columns":{
 "col0":{"cf":"cf0", "col":"col0", "type":"string"},
 "col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
 "col2":{"cf":"cf2", "col":"col2", "type":"double"},
 "col3":{"cf":"cf3", "col":"col3", "type":"float"},
 "col4":{"cf":"cf4", "col":"col4", "type":"int"},
 "col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
 "col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
 "col7":{"cf":"cf7", "col":"col7", "type":"string"},
 "col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
}
```

3.8.7. DataHub数据源

本文介绍如何使用DataHub数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName
USING datahub
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table Schema

创建DataHub表时，无需显式定义表的字段信息，示例如下所示。

```
spark-sql> CREATE TABLE datahub_table_test
      > USING datahub
      > OPTIONS
      > ( access.key.id = '<your access key id>',
      >  access.key.secret = '<you access key secret>',
      >  endpoint = '<your end point>',
      >  project = '<your project name>',
      >  topic = '<your topic>'
      > )
spark-sql> DESC datahub_table_test;
id  string  NULL
name string NULL
Time taken: 0.401 seconds, Fetched 2 row(s)
```

配置参数说明

参数名	说明	是否必选
access.key.id	阿里云AccessKey ID。	是
access.key.secret	阿里云AccessKey Secret。	是
endpoint	DataHub API Endpoint。	是
project	DataHub项目名。	是
topic	DataHub的topic。	是
decimal.precision	当topic字段中包含decimal字段时，需要指定。	否
decimal.scale	当topic字段中包含decimal字段时，需要指定。	否

3.8.8. Druid数据源

本文介绍如何使用Druid数据源进行数据分析或者交互式开发。

建表语法

```
create table tbName
using druid
options(propertyKey=propertyValue[, propertyKey=propertyValue]*);
```

Table Schema

创建Druid数据表时，无需显式地定义表的字段信息，示例如下。

```
create table druid_test_table
using druid
options(
curator.connect="${ZooKeeper-host}:${ZooKeeper-port}}",
index.service="druid/overlord",
data.source="test_source",
discovery.path="/druid/discovery",
firehose="druid:firehose:%s",
rollup.aggregators="{\"metricsSpec\":[{\"type\":\"count\",\"name\":\"count\"},
                    {\"type\":\"doubleSum\",\"fieldName\":\"value\",\"name\":\"sum\"},
                    {\"type\":\"doubleMin\",\"fieldName\":\"value\",\"name\":\"min\"},
                    {\"type\":\"doubleMax\",\"fieldName\":\"value\",\"name\":\"max\"}]}",
rollup.dimensions="timestamp,metric,userId",
rollup.query.granularities="minute",
tuning.segment.granularity="FIVE_MINUTE",
tuning.window.period="PT5M",
timestampSpec.column="timestamp",
timestampSpec.format="posix");
```

配置参数说明

参数名	说明	是否必选
curator.connect	ZooKeeper的Host，例如emr-header-1:2181。	是
curator.max.retries	ZooKeeper连接失败后的重试次数，默认值为5。	否
curator.retry.base.sleep	ZooKeeper连接失败后重试的间隔初始值，默认值为100，单位毫秒。	否
curator.retry.max.sleep	ZooKeeper连接失败后重试的间隔最大值，默认值为3000，单位毫秒。	否
index.service	例如druid或overlord。	是
data.source	写入Druid的data source。	是
discovery.path	默认值为druid或discovery。	否
firehose	例如 `druid:firehose:%s` 。	是

参数名	说明	是否必选
rollup.aggregators	tranquility聚合器，JSON格式，示例如下。 {\"metricsSpec\":[{\"type\":\"count\",\"name\":\"count\"}, {\"type\":\"doubleSum\",\"fieldName\":\"value\",\"name\":\"sum\"}, {\"type\":\"doubleMin\",\"fieldName\":\"value\",\"name\":\"min\"}, {\"type\":\"doubleMax\",\"fieldName\":\"value\",\"name\":\"max\"}]} 其中，metricsSpec为固定值。	是
rollup.dimensions	写入Druid的维度。	是
rollup.query.granularities	聚合粒度，例如minute。	是
tuning.window.period	时间窗大小，默认值为PT10M。	否
tuning.segment.granularity	segment粒度，默认值为DAY。	否
tuning.partitions	分区数量，默认值为1。	否
tuning.replications	副本数，默认值为1。	否
timestampSpec.column	写入Druid时的timestamp列名，默认值为timestamp。	否
timestampSpec.format	写入Druid时的timestamp列名的格式，默认值为iso。	否

3.8.9. Redis数据源

本文介绍如何使用Redis数据源进行数据分析或者交互式开发。

建表语法

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING redis
OPTIONS(propertyKey=propertyValue[, propertyKey=propertyValue]*);
```

Table Schema

创建Redis表时，必须显式地定义表的字段信息，示例如下所示。

```
spark-sql> CREATE TABLE redis_test_table(`key0` STRING, `value0` STRING, `key1` STRING, `value1` STR
ING)
      > USING redis
      > OPTIONS(
      > table="test",
      > redis.save.mode="append",
      > model="hash",
      > filter.keys.by.type="false",
      > key.column="uuid",
      > max.pipeline.size="100",
      > host="localhost",
      > port="6379",
      > dbNum="0");
```

配置参数说明

参数名	说明	是否必选
table	写入Redis数据的kev前缀。key格式为 ${table}:${kev.column} ，其中 ${key.column} 为配置项。	是
redis.save.mode	数据已经存在时的处理方式，包含append、overwrite、errorifexists或ignore，依次表示append到当前数据中、覆盖、抛出异常或丢弃数据，默认值为append。	否
model	数据存储格式，包含hash和binaray，默认值为hash。	否
filter.keys.by.type	是否过滤不符合数据存储格式的数据，默认值为false。	否
key.column	用来指定key的column。不指定时默认值为uuid。	否
ttl	不设置数值时表示默认永久保存；设置数值即为过期时间，单位是秒。	否
max.pipeline.size	通过pipeline方式写入数据的数据量最大值，默认值为100。	否
host	Redis实例的本地服务器地址，默认值为localhost。	否
port	Redis服务的端口，默认值为6379。	否
dbNum	数据存入Redis的数据库序号，默认值为0。	否

4.Hadoop

4.1. 参数说明

本文介绍Hadoop代码中的参数。

Hadoop代码中可使用如下参数配置。

属性名	默认值	说明
fs.jfs.cache.oss-accessKeyId	无	访问OSS所需的AccessKey ID（可选）。
fs.jfs.cache.oss-accessKeySecret	无	访问OSS所需的AccessKey Secret（可选）。
fs.jfs.cache.oss-securityToken	无	访问OSS所需的STS token（可选）。
fs.jfs.cache.oss-endpoint	无	访问OSS的Endpoint（可选）。
fs.oss.copy.simple.max.byte	134217728	使用普通接口进行OSS内部拷贝文件的大小上限。
fs.oss.multipart.split.max.byte	67108864	使用普通接口进行OSS内部拷贝文件的分片大小上限。
fs.oss.impl	• EMR-3.24.0及后续版本：com.aliyun.emr.fs.oss.JindoOssFileSystem • EMR-3.24.0之前版本：com.aliyun.fs.oss.nat.NativeOssFileSystem	OSS文件系统实现类。
io.compression.codec.snappy.native	false	标识Snappy文件是否为标准Snappy文件。Hadoop默认识别的是Hadoop修改过的Snappy格式文件。

4.2. MapReduce开发手册

本文以EMR-3.27.0集群为例，通过以下示例为您介绍如何在E-MapReduce集群中开发MR作业。

在MapReduce中使用OSS

在MapReduce中读写OSS，需要配置如下参数。

```
conf.set("fs.oss.accessKeyId", "${accessKeyId}");
conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
conf.set("fs.oss.endpoint","${endpoint}");
```

参数说明：

- ${accessKeyId}：阿里云账号的AccessKey ID。
- ${accessKeySecret}：阿里云账号的AccessKey Secret。
- ${endpoint}：OSS对外服务的访问域名。

  由您集群所在的地域决定，对应的OSS也需要是在集群对应的地域，详情请参见访问域名和数据中心。

WordCount示例

本示例为您介绍MapReduce如何从Master节点的OSS中读取文本，然后统计其中单词的数量并将数据写回到OSS中。

1. 通过SSH方式登录集群，详情请参见登录集群。
2. 执行以下命令，新建目录*wordcount_classes*。

   ```
   mkdir wordcount_classes
   ```

3. 执行以下命令，新建文件*EmrWordCount.java*。
   i. 执行以下命令，打开文件*EmrWordCount.java*。

   ```
   vim EmrWordCount.java
   ```

   ii. 按下 i 键进入编辑模式。
   iii. 在*EmrWordCount.java*文件中添加以下信息。

   ```
   package org.apache.hadoop.examples;
   import java.io.IOException;
   import java.util.StringTokenizer;
   import org.apache.hadoop.conf.Configuration;
   import org.apache.hadoop.fs.Path;
   import org.apache.hadoop.io.IntWritable;
   import org.apache.hadoop.io.Text;
   import org.apache.hadoop.mapreduce.Job;
   import org.apache.hadoop.mapreduce.Mapper;
   import org.apache.hadoop.mapreduce.Reducer;
   import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
   import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
   import org.apache.hadoop.util.GenericOptionsParser;
   public class EmrWordCount {
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();
     public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
      }
     }
    }
    public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
   ```

```
  private IntWritable result = new IntWritable();
  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
 }
 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }
  conf.set("fs.oss.accessKeyId", "${accessKeyId}");
  conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
  conf.set("fs.oss.endpoint","${endpoint}");
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(EmrWordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  for (int i = 0; i < otherArgs.length - 1; ++i) {
    FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
  }
  FileOutputFormat.setOutputPath(job,
    new Path(otherArgs[otherArgs.length - 1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

iv. 按下 `Esc` 键退出编辑模式，输入 `:wq` 保存并关闭文件。

4. 编译并打包文件。

i. 执行以下命令，编译程序。

```
javac -classpath <HADOOP_HOME>/share/hadoop/common/hadoop-common-X.X.X.jar:<HADOOP_HOME>/share/hadoop/mapreduce/hadoop-mapreduce-client-core-X.X.X.jar:<HADOOP_HOME>/share/hadoop/common/lib/commons-cli-1.2.jar -d wordcount_classes EmrWordCount.java
```

- `HADOOP_HOME`：Hadoop的安装目录，通常Hadoop的安装目录为*/usr/lib/hadoop-current*。

  您也可以通过`env |grep hadoop`命令获取安装目录。
- `X.X.X`：JAR包的具体版本号，需要根据实际集群中Hadoop的版本来修改。

  hadoop-common-X.X.X.jar，您可以在*<HADOOP_HOME>/share/hadoop/common/*目录下查看。*hadoop-mapreduce-client-core-X.X.X.jar*，您可以在*<HADOOP_HOME>/share/hadoop/mapreduce/*目录下查看。

ii. 执行以下命令，打包JAR文件。

```
jar cvf wordcount.jar -C wordcount_classes .
```

> **说明** 本示例中，打包后的*wordcount.jar*文件默认保存在*/root*目录下。

5. 创建作业。

i. 将步骤4的*wordcount.jar*上传到OSS，详情请参见上传文件。

例如，本示例中JAR文件在OSS上的路径为*oss://<yourBucketName>/jars/wordcount.jar*。

ii. 在E-MapReduce中新建MR作业，详情请参见Hadoop MapReduce作业配置。

作业内容如下所示：

```
ossref://<yourBucketName>/jars/wordcount.jar org.apache.hadoop.examples.EmrWordCount oss://<yourBucketName>/data/WordCount/Input oss://<yourBucketName>/data/WordCount/Output
```

代码中的`<yourBucketName>`需要替换为您实际的OSS Bucket，*oss://<yourBucketName>/data/WordCount/Input*和*oss://<yourBucketName>/data/WordCount/Output*分别为输入输出路径。

iii. 在作业编辑中，单击**运行**。

MR作业就会在指定的集群中运行起来。

Wordcount2示例

当您的工程规模比较大时，您可以使用类似Maven的项目管理工具来进行管理。本示例为您介绍如何通过Maven工程来管理MR作业。

1. 在本地安装Maven和Java环境。

   本示例中Maven是3.0版本，Java是1.8版本。

2. 执行如下命令，生成工程框架。

   例如，您的工程开发根目录是*D:/workspace*。

   ```
   mvn archetype:generate -DgroupId=com.aliyun.emr.hadoop.examples -DartifactId=wordcountv2 -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```

   通过以上命令会自动生成一个空的Sample工程位于*D:/workspace/wordcountv2*（和您指定的artifactId一致），里面包含一个简单的*pom.xml*和App类（类的包路径和您指定的groupId一致）。

3. 加入Hadoop依赖。

   使用IDE打开Sample工程，编辑*pom.xml*文件，当Hadoop是2.8.5版本时，需要添加如下内容。

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-common</artifactId>
    <version>2.8.5</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.8.5</version>
</dependency>
```

4. 编写代码。

   i. 在*com.aliyun.emr.hadoop.examples*中和App类平行的位置添加新类*EMapReduceOSSUtil*。

```
package com.aliyun.emr.hadoop.examples;
import org.apache.hadoop.conf.Configuration;
public class EMapReduceOSSUtil {
  private static String SCHEMA = "oss://";
  private static String EPSEP = ".";
  private static String HTTP_HEADER = "http://";
  /**
   * complete OSS uri
   * convert uri like: oss://bucket/path to oss://bucket.endpoint/path
   * ossref do not need this
   *
   * @param oriUri original OSS uri
   */
  public static String buildOSSCompleteUri(String oriUri, String endpoint) {
    if (endpoint == null) {
      System.err.println("miss endpoint");
      return oriUri;
    }
    int index = oriUri.indexOf(SCHEMA);
    if (index == -1 || index != 0) {
      return oriUri;
    }
    int bucketIndex = index + SCHEMA.length();
    int pathIndex = oriUri.indexOf("/", bucketIndex);
    String bucket = null;
    if (pathIndex == -1) {
      bucket = oriUri.substring(bucketIndex);
    } else {
      bucket = oriUri.substring(bucketIndex, pathIndex);
    }
    StringBuilder retUri = new StringBuilder();
    retUri.append(SCHEMA)
        .append(bucket)
        .append(EPSEP)
        .append(stripHttp(endpoint));
    if (pathIndex > 0) {
      retUri.append(oriUri.substring(pathIndex));
    }
    return retUri.toString();
  }
  public static String buildOSSCompleteUri(String oriUri, Configuration conf) {
    return buildOSSCompleteUri(oriUri, conf.get("fs.oss.endpoint"));
  }
  private static String stripHttp(String endpoint) {
    if (endpoint.startsWith(HTTP_HEADER)) {
      return endpoint.substring(HTTP_HEADER.length());
    }
    return endpoint;
  }
}
```

ii. 在*com.aliyun.emr.hadoop.examples*中和App类平行的位置添加新类*WordCount2.java*。

```
package com.aliyun.emr.hadoop.examples;
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.StringUtils;
public class WordCount2 {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    static enum CountersEnum { INPUT_WORDS }
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private boolean caseSensitive;
    private Set<String> patternsToSkip = new HashSet<String>();
    private Configuration conf;
    private BufferedReader fis;
    @Override
    public void setup(Context context) throws IOException,
        InterruptedException {
      conf = context.getConfiguration();
      caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
      if (conf.getBoolean("wordcount.skip.patterns", true)) {
        URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
        for (URI patternsURI : patternsURIs) {
          Path patternsPath = new Path(patternsURI.getPath());
          String patternsFileName = patternsPath.getName().toString();
          parseSkipFile(patternsFileName);
        }
      }
    }
    private void parseSkipFile(String fileName) {
      try {
        fis = new BufferedReader(new FileReader(fileName));
        String pattern = null;
        while ((pattern = fis.readLine()) != null) {
          patternsToSkip.add(pattern);
        }
      } catch (IOException ioe) {
        System.err.println("Caught exception while parsing the cached file '"
```

```
            + StringUtils.stringifyException(ioe));
      }
    }
    @Override
    public void map(Object key, Text value, Context context
    ) throws IOException, InterruptedException {
      String line = (caseSensitive) ?
          value.toString() : value.toString().toLowerCase();
      for (String pattern : patternsToSkip) {
        line = line.replaceAll(pattern, "");
      }
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
        Counter counter = context.getCounter(CountersEnum.class.getName(),
            CountersEnum.INPUT_WORDS.toString());
        counter.increment(1);
      }
    }
  }
  public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
    ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("fs.oss.accessKeyId", "${accessKeyId}");
    conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
    conf.set("fs.oss.endpoint","${endpoint}");
    GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
    String[] remainingArgs = optionParser.getRemainingArgs();
    if (!(remainingArgs.length != 2 || remainingArgs.length != 4)) {
      System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
      System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount2.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    List<String> otherArgs = new ArrayList<String>();
    for (int i=0; i < remainingArgs.length; ++i) {
```

```
      for (int i=0; i < remainingArgs.length; ++i) {
        if ("-skip".equals(remainingArgs[i])) {
          job.addCacheFile(new Path(EMapReduceOSSUtil.buildOSSCompleteUri(remainingArgs[++i
], conf)).toUri());
          job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
        } else {
          otherArgs.add(remainingArgs[i]);
        }
      }
      FileInputFormat.addInputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(oth
erArgs.get(0), conf)));
      FileOutputFormat.setOutputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(o
therArgs.get(1), conf)));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
  }
```

5. 在工程的目录下，执行如下命令，编译并打包文件。

   ```
   mvn clean package -DskipTests
   ```

   您可以在工程目录的*target*目录下看到名称为*wordcountv2-1.0-SNAPSHOT.jar*的JAR包。

6. 创建作业。

   i. 将步骤5的*wordcountv2-1.0-SNAPSHOT.jar*上传到OSS，详情请参见上传文件。

   例如，本示例中JAR文件在OSS上的路径为*oss://<yourBucketName>/jars/wordcountv2-1.0-SNAPSHOT.jar*。

   ii. 下载并上传以下文件至您OSS的对应目录。

   - The_Sorrows_of_Young_Werther.txt
   - patterns.txt

   > **说明** *The_Sorrows_of_Young_Werther.txt*为待统计单词的文本文件，*patterns.txt*文件用来处理需要忽略（不计频次）的单词。

   iii. 在E-MapReduce中新建MR作业，详情请参见Hadoop MapReduce作业配置。

   作业内容如下所示：

   ```
   ossref://<yourBucketName>/jars/wordcountv2-1.0-SNAPSHOT.jar com.aliyun.emr.hadoop.exampl
   es.WordCount2 -D wordcount.case.sensitive=true oss://<yourBucketName>/jars/The_Sorrows_of
   _Young_Werther.txt oss://<yourBucketName>/jars/output -skip oss://<yourBucketName>/jars/patt
   erns.txt
   ```

   代码中的 `<yourBucketName>` 需要替换为您实际的OSS Bucket，输出路径为*oss://<yourBucketName>/jars/output*。

   iv. 在作业编辑中，单击**运行**。

   MR作业就会在指定的集群中运行起来。

4.3. Hive开发手册

本文介绍如何在E-MapReduce集群中开发Hive作业流程。

在Hive中使用OSS

在Hive中读写OSS时，先创建一个external的表。

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://emr/users';
```

当上面的方式无法支持，或者您希望使用非本账号的AccessKey来访问其他位置的OSS数据的时候，请使用如下方式。

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/users';
```

参数说明：

- `${accessKeyId}`：您账号的AccessKey ID。
- `${accessKeySecret}`：该AccessKey ID对应的密钥。
- `${endpoint}`：访问OSS使用的网络，由您集群所在的Region决定，对应的OSS也需要是在集群对应的Region。

  详情请参见OSS Endpoint。

使用示例

Hive作业流程示例如下：

- 示例1
  i. 编写如下脚本，保存为*hiveSample1.sql*文件，并上传至OSS。

  上传详情请参见上传文件。

  ```
  USE DEFAULT;
  set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
  set hive.stats.autogather=false;
  DROP TABLE emrusers;
  CREATE EXTERNAL TABLE emrusers (
   userid INT,
   movieid INT,
   rating INT,
   unixtime STRING )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  STORED AS TEXTFILE
  LOCATION 'oss://${bucket}/yourpath';
  SELECT COUNT(*) FROM emrusers;
  SELECT * from emrusers limit 100;
  SELECT movieid,count(userid) as usercount from emrusers group by movieid order by usercount de
  sc limit 50;
  ```

  ii. 测试用数据资源

  您可以下载如下Hive作业需要的资源，然后将其上传至您OSS对应的目录。

资源下载：公共测试数据。

iii. 创建作业。

在E-MapReduce中新建一个Hive作业，详情请参见Hive作业配置。

作业内容如下。

```
-f ossref://${bucket}/yourpath/hiveSample1.sql
```

其中 ${bucket} 是您的OSS Bucket， yourpath 是Bucket上的路径，需要您填写实际保存Hive脚本的位置。

iv. 运行作业

单击**运行**以运行作业。您可以关联一个已有的集群，也可以自动按需创建一个，然后关联上创建的作业。

- 示例2

以HiBench中的scan为例。

i. 编写如下脚本，上传至OSS。

```
USE DEFAULT;
 set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
 set mapreduce.job.maps=12;
 set mapreduce.job.reduces=6;
 set hive.stats.autogather=false;
 DROP TABLE uservisits;
 CREATE EXTERNAL TABLE uservisits (sourceIP STRING,destURL STRING,visitDate STRING,adRevenue
DOUBLE,userAgent STRING,countryCode STRING,languageCode STRING,searchWord STRING,duratio
n INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS SEQUENCEFILE LOCATION 'oss:
//${bucket}/sample-data/hive/Scan/Input/uservisits';
```

例如存储路径为*oss://emr/jars/scan.hive*。

ii. 准备测试数据

您可以通过下面的地址下载作业需要的资源，然后将其上传至您OSS对应的目录。

资源下载：uservisits。

iii. 在E-MapReduce中创建Hive作业，详情请参见Hive作业配置。

iv. 运行作业

单击**运行**以运行作业。您可以关联一个已有的集群，也可以自动按需创建一个，然后关联上创建的作业。

4.4. Pig开发手册

本文介绍如何在E-MapReduce集群中开发Pig作业流程。

在Pig中使用OSS

在Pig中使用OSS路径时，请使用类似如下的形式。

```
oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/${path}
```

参数说明：

- ${accessKeyId}：您账号的AccessKey ID。
- ${accessKeySecret}：该AccessKey ID对应的密钥。
- ${bucket}：该AccessKey ID对应的Bucket。
- ${endpoint}：访问OSS使用的网络，由您集群所在的Region决定，对应的OSS也需要是在集群对应的Region。

  详情请参见OSS Endpoint。
- ${path}：Bucket中的路径。

实施步骤

以Pig中带*script1-hadoop.pig*为例进行说明，将Pig中的tutorial.jar和excite.log.bz2上传至OSS。例如上传路径为*oss://emr/jars/tutorial.jar*和*oss://emr/data/excite.log.bz2*。

1. 编写脚本

   根据准备工作中所上传的OSS路径，修改脚本中的JAR包文件路径和输入输出路径。

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
* or more contributor license agreements.  See the NOTICE file
* distributed with this work for additional information
* regarding copyright ownership.  The ASF licenses this file
* to you under the Apache License, Version 2.0 (the
* "License"); you may not use this file except in compliance
* with the License.  You may obtain a copy of the License at
*
*    http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
-- Query Phrase Popularity (Hadoop cluster)
-- This script processes a search query log file from the Excite search engine and finds search phrases th
at occur with particular high frequency during certain times of the day.
-- Register the tutorial JAR file so that the included UDFs can be called in the script.
REGISTER oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/tutorial.jar;
-- Use the  PigStorage function to load the excite log file into the ▒raw▒ bag as an array of records.
-- Input: (user,time,query)
raw = LOAD 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/excite.log.bz2' USIN
G PigStorage('\t') AS (user, time, query);
-- Call the NonURLDetector UDF to remove records if the query field is empty or a URL.
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
-- Call the ToLower UDF to change the query field to lowercase.
clean2 = FOREACH clean1 GENERATE user, time,    org.apache.pig.tutorial.ToLower(query) as query;
-- Because the log file only contains queries for a single day, we are only interested in the hour.
-- The excite query log timestamp format is YYMMDDHHMMSS.
-- Call the ExtractHour UDF to extract the hour (HH) from the time field.
houred = FOREACH clean2 GENERATE user, org.apache.pig.tutorial.ExtractHour(time) as hour, query;
-- Call the NGramGenerator UDF to compose the n-grams of the query.
```

```
ngramed1 = FOREACH houred GENERATE user, hour, flatten(org.apache.pig.tutorial.NGramGenerator(q
uery)) as ngram;
-- Use the  DISTINCT command to get the unique n-grams for all records.
ngramed2 = DISTINCT ngramed1;
-- Use the  GROUP command to group records by n-gram and hour.
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
-- Use the  COUNT function to get the count (occurrences) of each n-gram.
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT($1) as count;
-- Use the  GROUP command to group records by n-gram only.
-- Each group now corresponds to a distinct n-gram and has the count for each hour.
uniq_frequency1 = GROUP hour_frequency2 BY group::ngram;
-- For each group, identify the hour in which this n-gram is used with a particularly high frequency.
-- Call the ScoreGenerator UDF to calculate a "popularity" score for the n-gram.
uniq_frequency2 = FOREACH uniq_frequency1 GENERATE flatten($0), flatten(org.apache.pig.tutorial.Sc
oreGenerator($1));
-- Use the  FOREACH-GENERATE command to assign names to the fields.
uniq_frequency3 = FOREACH uniq_frequency2 GENERATE $1 as hour, $0 as ngram, $2 as score, $3 as cou
nt, $4 as mean;
-- Use the  FILTER command to move all records with a score less than or equal to 2.0.
filtered_uniq_frequency = FILTER uniq_frequency3 BY score > 2.0;
-- Use the  ORDER command to sort the remaining records by hour and score.
ordered_uniq_frequency = ORDER filtered_uniq_frequency BY hour, score;
-- Use the  PigStorage function to store the results.
-- Output: (hour, n-gram, score, count, average_counts_among_all_hours)
STORE ordered_uniq_frequency INTO 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint
}/data/script1-hadoop-results' USING PigStorage();
```

上传*script1-hadoop.pig*脚本至OSS，例如*oss://emr/jars/script1-hadoop.pig*。

2. 创建作业

   在数据开发中创建Pig作业，详情请参见Pig作业配置。

   作业内容如下。

   ```
   -f ossref://emr/jars/script1-hadoop.pig
   ```

3. 运行作业

   单击**运行**以运行作业。您可以关联一个已有的集群，也可以自动按需创建一个，然后关联上创建的作业。

4.5. Hadoop Streaming

本文为您介绍如何使用Python提交Hadoop Streaming作业。

前提条件

已在E-MapReduce控制台上创建Hadoop集群。

创建集群详情，请参见创建集群。

操作步骤

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 新建文件*mapper.py*。

i. 执行以下命令，打开文件*mapper.py*。

```
vim /home/hadoop/mapper.py
```

ii. 按下 `i` 键进入编辑模式。

iii. 在*mapper.py*文件中添加以下信息。

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
  line = line.strip()
  words = line.split()
  for word in words:
    print '%s\t%s' % (word, 1)
```

iv. 按下 `Esc` 键退出编辑模式，输入 `:wq` 保存并关闭文件。

3. 新建文件*reducer.py*。

i. 执行以下命令，打开文件*reducer.py*。

```
vim /home/hadoop/reducer.py
```

ii. 按下 `i` 键进入编辑模式。

iii. 在*reducer.py*文件中添加以下信息。

```
#!/usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
  line = line.strip()
  word, count = line.split('\t', 1)
  try:
    count = int(count)
  except ValueError:
    continue
  if current_word == word:
    current_count += count
  else:
    if current_word:
      print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word
if current_word == word:
  print '%s\t%s' % (current_word, current_count)
```

iv. 按下 `Esc` 键退出编辑模式，输入 `:wq` 保存并关闭文件。

4. 执行以下命令，上传*hosts*文件到HDFS。

```
hdfs dfs -put /etc/hosts /tmp/
```

5. 执行以下命令，提交Hadoop Streaming作业。

```
hadoop jar /usr/lib/hadoop-current/share/hadoop/tools/lib/hadoop-streaming-X.X.X.jar -file /home/hadoop/mapper.py -mapper mapper.py -file /home/hadoop/reducer.py -reducer reducer.py -input /tmp/hosts -output /tmp/output
```

参数	描述
input	输入路径，本示例为*/tmp/hosts*。
output	输出路径，本示例为*/tmp/output*。

> **说明** *hadoop-streaming-X.X.X.jar*中的 X.X.X 表示JAR包的具体版本号，需要根据实际集群中Hadoop的版本来修改。您可以在*/usr/lib/hadoop-current/share/hadoop/tools/lib/*目录下查看JAR包具体版本号。

4.6. Hive+TableStore

本章节介绍如何在Hive中处理TableStore中的数据。

Hive接入TableStore

- 准备一张数据表

  创建一张表pet，其中name为主键。

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	无
Claws	Gwen	cat	m	1994-03-17	无
Buffy	Harold	dog	f	1989-05-13	无
Fang	Benny	dog	m	1990-08-27	无
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	无
Whistler	Gwen	bird	-	1997-12-09	无
Slim	Benny	snake	m	1996-04-29	无
Puffball	Diane	hamster	f	1999-03-30	无

- 以下示例展示了如何在Hive中处理TableStore中的数据。

  i. 命令行

  ```
  $ HADOOP_HOME=YourHadoopDir HADOOP_CLASSPATH=emr-tablestore-<version>.jar:tablestore-4.1.0-jar-with-dependencies.jar:joda-time-2.9.4.jar bin/hive
  ```

  ii. 创建一张外表

```
CREATE EXTERNAL TABLE pet
 (name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
 STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
 WITH SERDEPROPERTIES(
  "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
 TBLPROPERTIES (
  "tablestore.endpoint"="YourEndpoint",
  "tablestore.access_key_id"="YourAccessKeyId",
  "tablestore.access_key_secret"="YourAccessKeySecret",
  "tablestore.table.name"="pet");
```

iii. 向外表中插入数据

```
INSERT INTO pet VALUES("Fluffy", "Harold", "cat", "f", "1993-02-04", null);
INSERT INTO pet VALUES("Claws", "Gwen", "cat", "m", "1994-03-17", null);
INSERT INTO pet VALUES("Buffy", "Harold", "dog", "f", "1989-05-13", null);
INSERT INTO pet VALUES("Fang", "Benny", "dog", "m", "1990-08-27", null);
INSERT INTO pet VALUES("Bowser", "Diane", "dog", "m", "1979-08-31", "1995-07-29");
INSERT INTO pet VALUES("Chirpy", "Gwen", "bird", "f", "1998-09-11", null);
INSERT INTO pet VALUES("Whistler", "Gwen", "bird", null, "1997-12-09", null);
INSERT INTO pet VALUES("Slim", "Benny", "snake", "m", "1996-04-29", null);
INSERT INTO pet VALUES("Puffball", "Diane", "hamster", "f", "1999-03-30", null);
```

iv. 查询数据

```
> SELECT * FROM pet;
Bowser Diane dog m 1979-08-31 1995-07-29
Buffy Harold dog f 1989-05-13 NULL
Chirpy Gwen bird f 1998-09-11 NULL
Claws Gwen cat m 1994-03-17 NULL
Fang Benny dog m 1990-08-27 NULL
Fluffy Harold cat f 1993-02-04 NULL
Puffball Diane hamster f 1999-03-30 NULL
Slim Benny snake m 1996-04-29 NULL
Whistler Gwen bird NULL 1997-12-09 NULL
> SELECT * FROM pet WHERE birth > "1995-01-01";
Chirpy Gwen bird f 1998-09-11 NULL
Puffball Diane hamster f 1999-03-30 NULL
Slim Benny snake m 1996-04-29 NULL
Whistler Gwen bird NULL 1997-12-09 NULL
```

数据类型转换

类型	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	支持，损失精度	支持，损失精度	支持，损失精度	支持	支持，损失精度	支持，损失精度	不支持	不支持	不支持
DOUBLE	支持，损失精度	支持，损失精度	支持，损失精度	支持，损失精度	支持，损失精度	支持	不支持	不支持	不支持

类型	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
BOOLEAN	不支持	不支持	不支持	不支持	不支持	不支持	支持	不支持	不支持
STRING	不支持	不支持	不支持	不支持	不支持	不支持	不支持	支持	不支持
BINARY	不支持	不支持	不支持	不支持	不支持	不支持	不支持	不支持	支持

附录

完整示例代码请参见：Hive+TableStore。

5.HBase

5.1. 访问HBase

本文介绍如何配置HBase集群以及HBase存储服务使用流程。

前提条件

已创建集群，并添加HBase服务，详情请参见创建集群。

HBase配置

您可以在创建HBase集群的**软件配置**页面，利用**高级设置**的**软件自定义配置**功能，结合使用场景，修改HBase的默认参数，示例如下。

```
{
 "configurations": [
  {
   "classification": "hbase-site",
   "properties": {
    "hbase.hregion.memstore.flush.size": "268435456",
    "hbase.regionserver.global.memstore.size": "0.5",
    "hbase.regionserver.global.memstore.lowerLimit": "0.6"
   }
  }
 ]
}
```

HBase集群部分默认配置如下所示。

key	value
zookeeper.session.timeout	180000
hbase.regionserver.global.memstore.size	0.35
hbase.regionserver.global.memstore.lowerLimit	0.3
hbase.hregion.memstore.flush.size	128MB

访问HBase

1.
2. 执行如下命令，进入HBase Shell。

   ```
   hbase shell
   ```

   返回如下信息。

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hbase/1.4.9-1.0.0/package/hbase-1.4.9-1.0.0/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.5.3/package/hadoop-2.8.5-1.5.3/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.9, r8214a16c5d80f077abf1aa01bb312851511a2b15, Thu Jan 31 20:35:22 CST 2019
```

> **说明** 如果您是通过ECS控制台创建的实例，请参见使用HBase Shell访问。

示例

- Spark访问HBase

  详情请参见spark-hbase-connector。

- Hadoop访问HBase

  详情请参见HBase MapReduce Examples。

- Hive访问HBase

  i. 登录Hive集群主节点，修改*hosts*文件，增加如下内容。

  ```
  $zk_ip emr-cluster //$zk_ip为Hbase集群的Zookeeper节点IP。
  ```

  ii. 执行Hive操作，详情请参见Hive HBase Integration。

5.2. 备份HBase集群

本文介绍如何备份E-MapReduce的HBase集群。

前提条件

已创建两个Hadoop集群，并添加HBase和Zookeeper服务，详情请参见创建集群。

操作步骤

1.
2. 创建Table并添加数据。

   i. 打开HBase Shell。

   ```
   hbase shell
   ```

   ii. 创建表。

   ```
   create 'test','cf'
   ```

iii. 添加数据。

```
put 'test','a','cf:c1',1
put 'test','a','cf:c2',2
put 'test','b','cf:c1',3
put 'test','b','cf:c2',4
put 'test','c','cf:c1',5
put 'test','c','cf:c2',6
```

iv. 退出HBase Shell。

```
exit
```

3. 创建并查看快照。

i. 创建快照。

```
hbase snapshot create -n test_snapshot -t test
```

ii. 打开HBase Shell。

```
hbase shell
```

iii. 查看快照。

```
list_snapshots
```

返回如下信息。

```
SNAPSHOT                        TABLE + CREATION TIME
 test_snapshot                   test (Tue Aug 18 14:35:28 +0800 2020)
1 row(s) in 0.2450 seconds
=> ["test_snapshot"]
```

iv. 退出HBase Shell。

```
exit
```

4. 导出数据至OSS。

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-to oss://$accessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/test
```

> 说明 OSS使用内网Endpoint。

5. 通过SSH方式登录另一个集群。

详情请参见登录集群。

6. 导出OSS快照。

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-from oss://$accessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/test -copy-to /hbase/
```

7. 从快照恢复数据并新建表。

i. 打开HBase Shell。

```
hbase shell
```

ii. 从快照恢复数据。

```
restore_snapshot 'test_snapshot'
```

iii. 查看表。

```
scan 'test'
```

返回如下信息。

```
ROW             COLUMN+CELL
 a              column=cf:c1, timestamp=1472992081375, value=1
 a              column=cf:c2, timestamp=1472992090434, value=2
 b              column=cf:c1, timestamp=1472992104339, value=3
 b              column=cf:c2, timestamp=1472992099611, value=4
 c              column=cf:c1, timestamp=1472992112657, value=5
 c              column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```

8. 从快照创建新表并查看数据。

i. 从指定的快照生成新表。

```
clone_snapshot 'test_snapshot','test_2'
```

ii. 查看新表数据。

```
scan 'test_2'
```

返回信息如下。

```
ROW             COLUMN+CELL
 a              column=cf:c1, timestamp=1472992081375, value=1
 a              column=cf:c2, timestamp=1472992090434, value=2
 b              column=cf:c1, timestamp=1472992104339, value=3
 b              column=cf:c2, timestamp=1472992099611, value=4
 c              column=cf:c1, timestamp=1472992112657, value=5
 c              column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```