填空

HDFS Shell

hdfs dfs -操作命令 参数

```
hdfs dfs -mkdir  /abc                                    #创建名为/abc的文件夹
hdfs dfs -ls  /                                          #列出根目录中的内容
hdfs dfs -ls -R  /                                       #递归列出多层文件夹的内容
hdfs dfs -put  /etc/hosts /abc/hosts                     #把Linux系统中/etc/hosts文件上传到HDFS中
hdfs dfs -appendToFile  /etc/hosts /abc/hosts            #向文件中追加内容
hdfs dfs -checksum  /abc/hosts                           #查看文件的MD5值
hdfs dfs -du -h  /              #查看文件/文件夹的大小  -h以人类友好的方式显示大小（过大时带单位）
hdfs dfs -get  /abc/hosts ./hosts #把HDFS中的文件下载到本地Linux中./hosts是下载后保存到本地的位置
hdfs dfs -cat  /abc/hosts                                #查看HDFS中文本文件的内容
hdfs dfs -tail  /abc/hosts                               #列出文件结尾处1KB的文件内容
hdfs dfs -mv  /abc/hosts /abc/xyz                        #修改文件名字或移动位置
hdfs dfs -cp  /abc/xyz /abc/hosts                        #复制文件
hdfs dfs -find  / -name xyz                              #查找名字为xyz的文件的位置
hdfs dfs -rmdir  /abc                          #删除名为/abc的文件夹  注意：如果其中还有文件则不能删除
hdfs dfs -rm  /abc/hosts                                 #删除文件
hdfs dfs -rm -r  /abc                                    #递归删除文件/文件夹，文件夹中有文件也能删除
hdfs dfs -df                                             #查看HDFS文件系统的磁盘使用情况
```

Hbase

Shell

```
list                                    #列出HBase中所有的表信息
create 'tempTable','f1','f2','f3'       #创建一个表，该表名称为tempTable，包含3个列族f1，f2和f3
put 'tempTable','r1','f1:c1','hello,dblab'   #向表tempTable中的第r1行、第“f1:c1”列，添加数据“hello,dblab”
get 'tempTable','r1',{COLUMN=>'f1:c1'}       #从tempTable中，获取第r1行、第“f1:c1”列的值
disable 'tempTable'
drop 'temptable'
```

Java API

```
createTable(String "tablename", String[] colFamily)
insertData("tablename", "rowKey", "colFamily", "col", "val")
getData("tablename", "rowKey", "colFamily", "col")
```

MR实现HBase

extends TableMapper，TableReducer

Hive

启动

```
#启动元数据服务
hive --service metastore &
./bin/hive
```

基本操作

```
1. create：创建数据库、表、视图
create database hive;  # 创建数据库hive，若hive已经存在，则会抛出异常
create database if not exists hive;

use hive;
# 在hive数据库中，创建表usr，含三个属性id，name，age
create table if not exists usr(id bigint,name string,age int);
# 在hive数据库中，创建表usr，含三个属性id，name，age，存储路径为“.../hive/usr”
create table if not exists hive.usr(id bigint,name string,age int)
    location ‘/usr/local/hive/warehouse/hive/usr’;
# 在hive数据库中，创建外部表usr，含三个属性id，name，age，可以读取路径“/usr/local/data”下以“，”分隔的数据
create external table if not exists hive.usr(id bigint,name string,age int)
    row format delimited fields terminated by ','
    location ‘/usr/local/data’;
# 在hive数据库中，创建分区表usr，含三个属性id，name，age，还存在分区字段sex
create table hive.usr(id bigint,name string,age int) partition by(sex boolean);
# 在hive数据库中，创建分区表usr1，它通过复制表usr得到
create table if not exists usr1 like usr;

# 创建视图little_usr，只包含usr表中id，age属性
create view little_usr as select id,age from usr;

2. drop：删除数据库、表、视图
drop database hive;
drop database if exists hive;
drop database if exists hive cascade; # 加上cascade关键字，可以删除当前数据库和该数据库中的表

#删除表usr，如果是内部表，元数据和实际数据都会被删除；如果是外部表，只删除元数据，不删除实际数据
drop table if exists usr;

drop view if exists little_usr;

# 为hive数据库设置dbproperties键值对属性值来描述数据库属性信息
alter database hive set dbproperties(‘edited-by’=’lily’);

3. alter：修改数据库、表、视图
alter table usr rename to user;    # 重命名
alter table usr add if not exists partition(age=10);    # 为表usr增加新分区
alter table usr drop if exists partition(age=10);    # 删除表usr中分区
```

```
# 把表usr中列名name修改为username，并把该列置于age列后
alter table usr change name username string after age;
# 在对表usr分区字段之前，增加一个新列sex
alter table usr add columns(sex boolean);
# 删除表usr中所有字段并重新指定新字段newid，newname，newage
alter table usr replace columns(newid bigint,newname string,newage int);
# 为usr表设置tblproperties键值对属性值来描述表的属性信息
alter table usr set tabproperties('notes'='the columns in usr may be null except id');

# 修改little_usr视图元数据中的tblproperties属性信息
alter view little_usr set tabproperties('create_at'='refer to timestamp');

4. show：查看数据库、表、视图
show databases;    # 查看Hive中包含的所有数据库
show databases like 'h.*';    # 查看Hive中以h开头的所有数据库

use hive;
show tables;   # 查看数据库hive中所有表和视图
show tables in hive like 'u.*';   # 查看数据库hive中以u开头的所有表和视图

5. describe：描述数据库、表、视图
describe database hive; # 查看数据库hive的基本信息，包括数据库中文件位置信息等
describe database extended hive; # 查看数据库hive的详细信息，包括数据库的基本信息及属性信息等

describe hive.usr/ hive.little_usr;   #查看表usr和视图little_usr的基本信息，包括列信息等
describe extended hive.usr/ hive.little_usr;  # 查看表usr和视图little_usr的详细信息，包括列信息、位置信息、属性信息等
describe extended hive.usr.id;   #查看表usr中列id的信息

6. load：向表中装载数据
# 把目录'/usr/local/data'下的数据文件中的数据装载进usr表并覆盖原有数据
load data local inpath '/usr/local/data' overwrite into table usr;
# 把目录'/usr/local/data'下的数据文件中的数据装载进usr表不覆盖原有数据
load data local inpath '/usr/local/data' into table usr;
# 把分布式文件系统目录'hdfs://master_srever/usr/local/data'下的数据文件数据装载进usr表并覆盖原有数据
load data inpath 'hdfs://master_srever/usr/local/data' overwrite into table usr;

7. insert：向表中插入数据或从表中导出数据
# 向表usr1中插入来自usr表的数据并覆盖原有数据
insert overwrite table usr1
    select * from usr where age=10;
# 向表usr1中插入来自usr表的数据并追加在原有数据后
insert into table usr1
    select * from usr
    where age=10;
```

Flume

配置文件 flume-conf.properties

```
agent.sources = src
agent.channels = memoryChannel
agent.sinks = loggerSink

agent.sources.src.type = netcat
agent.sources.src.bind=localhost  # 表示从本地获取数据
agent.sources.src.port=8090  # 绑定端口号
agent.sources.src.channels = memoryChannel

agent.sinks.loggerSink.type = logger
agent.sinks.loggerSink.channel = memoryChannel

agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 100
```

启动命令

```
bin/flume-ng  agent -n agent -c conf -f conf/flume-conf.properties -
Dflume.root.logger=INFO,console

-c：flume启动时读取的配置文件flume-env.sh路径
-f：配置文件flume-conf.properties的存储目录
-n：给flume-conf.properties中的agent起名为agent
```

Flume与Kafka集成

sinks.k1.type=KafkaSink

sinks.k1.kafka.producer.acks=1

Flume与Hbase集成

需要重写getRowKey函数

sinks.k1.type=hbase2

sinks.k1.table=testtable

sinks.k1.columFamily=info

sinks.k1.serializer=序列化方式

.*?尽可能少地匹配换行符之外的字符 .*尽可能多的匹配换行符之外的字符

Scala

scala解释器

```
scala  # 启动Scala解释器
:quit  #退出解释器
:load  /usr/local/scala/mycode/Test.scala   # 导入脚本
scalac  HelloWorld.scala  # 使用scalac命令进行编译（编译的结果为Java字节码）
scala -classpath . HelloWorld  # 使用scala或者java命令运行字节码文件
java  -classpath  .:/usr/local/scala/lib/scala-library.jar  HelloWorld
```

Spark

运行指令

```
本地模式
local[K]     #指定使用几个线程来运行计算
local[*]     #按照Cpu最多Cores来设置线程数
#local
./bin/spark-submit   --class org.apache.spark.examples.SparkPi  --master local[8]
examples/jars/spark-examples*.jar  100

#yarn-client
./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn --
deploy-mode client --driver-memory 1G --num-executors 3 --executor-memory 1G --
executor-cores 1 ./examples/jars/spark-examples*.jar 100

#yarn-cluster
./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn --
deploy-mode cluster --driver-memory 1G --num-executors 3 --executor-memory 1G --
executor-cores 1 ./examples/jars/spark-examples*.jar 10

#standalone-client
./bin/spark-submit --master spark://namenode:7077 --class
org.apache.spark.examples.SparkPi  ./examples/jars/spark-examples*.jar  10

#standalone-cluster
./bin/spark-submit --class org.apache.spark.examples.SparkPi --master
spark://namenode:7077 --deploy-mode cluster --supervise --executor-memory 2G --
total-executor-cores 1  /home/hadoop/spark-3.1.2-bin-
hadoop3.2/examples/jars/spark-examples*.jar 10
```

RDD

读取Hbase数据

sc.newAPIHadoopRDD根据conf中配置好的scan来从Hbase的数据列族中读取包含(ImmutableBytesWritable, Result)的RDD

```
object SparkOperateHBase {
    def main(args: Array[String]) {
        val conf = HBaseConfiguration.create()
        val sc = new SparkContext(new SparkConf())
        //设置查询的表名
        conf.set(TableInputFormat.INPUT_TABLE, "student")
        ##################这里正下方
        val stuRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
      classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
      classOf[org.apache.hadoop.hbase.client.Result])
        val count = stuRDD.count()
        println("Students RDD Count:" + count)
        stuRDD.cache()
        //遍历输出
        stuRDD.foreach({ case (_,result) =>
            val key = Bytes.toString(result.getRowKey)
            val name =
Bytes.toString(result.getValue("info".getBytes,"name".getBytes))
```

```
            val gender =
Bytes.toString(result.getValue("info".getBytes,"gender".getBytes))
            val age =
Bytes.toString(result.getValue("info".getBytes,"age".getBytes))
            println("Row key:"+key+" Name:"+name+" Gender:"+gender+" Age:"+age)
        })
    }
}
```

向HBase写入数据

```
object SparkWriteHBase {
  def main(args: Array[String]): Unit = {
    val sparkConf = new
SparkConf().setAppName("SparkWriteHBase").setMaster("local")
    val sc = new SparkContext(sparkConf)
    val tablename = "student"
    sc.hadoopConfiguration.set(TableOutputFormat.OUTPUT_TABLE, tablename)
    val job = new Job(sc.hadoopConfiguration)
    job.setOutputKeyClass(classOf[ImmutableBytesWritable])
    job.setOutputValueClass(classOf[Result])
########################################正下方
 job.setOutputFormatClass(classOf[TableOutputFormat[ImmutableBytesWritable]])
    val indataRDD = sc.makeRDD(Array("3,Rongcheng,M,26","4,Guanhua,M,27")) //构建
两行记录
    val rdd = indataRDD.map(_.split(',')).map{arr=>{
      val put = new Put(Bytes.toBytes(arr(0))) //行健的值
      put.add(Bytes.toBytes("info"),Bytes.toBytes("name"),Bytes.toBytes(arr(1)))
//info:name列的值

put.add(Bytes.toBytes("info"),Bytes.toBytes("gender"),Bytes.toBytes(arr(2)))
//info:gender列的值

put.add(Bytes.toBytes("info"),Bytes.toBytes("age"),Bytes.toBytes(arr(3).toInt))
//info:age列的值
      (new ImmutableBytesWritable, put)
    }}
    ############saveAsNewAPIHadoopDataset将RDD输出到Hadoop支持的存储系统中
    rdd.saveAsNewAPIHadoopDataset(job.getConfiguration())
  }
}
```

JSON读写

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import scala.util.parsing.json.JSON
object JSONRead {
    def main(args: Array[String]) {
        val conf = new SparkConf().setAppName("JSONRead")
        val sc = new SparkContext(conf)
        val jsonStrs = sc.textFile("file:///usr/people.json")
        ################################这里
        val result = jsonStrs.map(s => JSON.parseFull(s))
```

```
        result.foreach( {r => r match {
            case Some(map: Map[String, Any]) => println(map)
            case None => println("Parsing failed")
            case other => println("Unknown data structure: " + other)
            }
            }
    }
}
```

SparkSQL

RDD 入口 SparkContext

sparksql 入口 SparkSession

sparkstreaming 入口 StreamingContext

rdd，sparksql，sparkstreaming都是用什么函数读入文件的

RDD、DataFrame、DataSet间的相互转换

```
#需要通过import语句（即import spark.implicits._）导入相应的包，启用隐式转换。
import ss.implicits._

#DataFrame、Dataset转RDD：
val rdd1 = testDF.rdd
val rdd2 = testDS.rdd

#RDD转DataFrame：
val testDF = rdd.map(line => (line._1, line._2)).toDF("name", "age")

#RDD转Dataset：
case class Person(name:String, age: Int) extends Serializable
var personDS = rdd.map(line => Person(line._1, line._2)).toDS

#Dataset转DataFrame：
val testDF = testDS.toDF

#DataFrame转Dataset：
case class Person(name:String, age: Int) extends Serializable
var personDS = testDF.as[Person]
```

DataFrame常用操作

```
#创建一个SparkSession对象
val spark=SparkSession.builder().getOrCreate()

#从不同类型的文件中加载数据创建DataFrame
spark.read.json("people.json")
spark.read.csv("people.csv")

#DataFrame保存
testdf.write.json("people.json“)
testdf.write.csv("people.csv")
```

RDD转DF并使用SQL操作

```
#提前定义caseclass
利用反射机制推断RDD模式
case class Person(name: String, age: Long)
val peopleDF =
spark.sparkContext.textFile("file:///usr/people.txt").map(_.split(",")).map(attr
ibutes => Person(attributes(0), attributes(1).trim.toInt)).toDF()
peopleDF.createOrReplaceTempView("people") //必须注册为临时表才能供下面的查询使用
val personsDF2 = spark.sql("select name,age from people where age > 20")
personsRDD.map(t => "Name: "+t(0)+ ","+"Age: "+t(1)).show()

#不提前定义case class
采用编程方式定义RDD模式  StructField。StructType
val fields =
Array(StructField("name",StringType,true),StructField("age",IntegerType,true))
val schema = StructType(fields)

val rowRDD =
spark.sparkContext.textFile("file:///usr/people.txt").map(_.split(",")).map(attr
ibutes => Row(attributes(0), attributes(1).trim.toInt))
val peopleDF = spark.createDataFrame(rowRDD, schema)
peopleDF.createOrReplaceTempView("people")
val results = spark.sql("SELECT name,age FROM people")
```

DataSet常用操作

```
case class Girl(id:Int, name:String, sex:Int, height:Double)
object DataSetDemo1 extends App {
    val spark: SparkSession = SparkSession.builder()
    .appName("SparkSQL")
    .master("local")
    .getOrCreate()
    import spark.implicits._
    Private val girls = List(
        Girl(1,"lw",1,180.0),
        Girl(2,"wsw",2,179.0),
        Girl(3,"chh",1,183.0),
        Girl(4,"mnn",0,168.0))
    val ds= spark.createDataset(girls)
    ds.show()
    spark.stop()
}
#利用RDD创建DataSet对象
object rdd2DataSet extends App {
    val spark: SparkSession = SparkSession.builder()
    .appName("SparkSQL")
    .master("local")
    .getOrCreate()
    import spark.implicits._
    private val value: RDD[Girl]= spark.sparkContext.parallelize(List(
        Girl(1,"lw",1,180.0),
        Girl(2,"wsw",2,179.0),
        Girl(3,"mhh",1,183.0),
        Girl(4,"mnn",0,168.0))
    private val dataset: Dataset[Girl]= value.toDS()
```

```
    dataset.show()
}
```

DataSet其他操作

```
#Dataset转RDD
val rdd: RDD[Girl]= dataset.rdd
rdd.foreach(println)

#Dataset转DataFrame
val dataFrame: DataFrame = dataset.toDF()
dataFrame.show()
```

SparkStreaming

入口

```
val conf = new SparkConf().setAppName("TestDStream").setMaster("local[2]")
val ssc = new StreamingContext(conf, Seconds(1))
```

基本输入源

```
#文件流
val ssc = new StreamingContext(sc, Seconds(20))
val lines = ssc.textFileStream("file:///usr/logfile")
ssc.start()

#套接字流
val sparkConf = new
SparkConf().setAppName("NetworkWordCount").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(1))
val lines = ssc.socketTextStream(args(0), args(1).toInt,
StorageLevel.MEMORY_AND_DISK_SER)

#RDD队列流
val sparkConf = new SparkConf().setAppName("TestRDDQueue").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(2))
val rddQueue =new scala.collection.mutable.SynchronizedQueue[RDD[Int]]()
val queueStream = ssc.queueStream(rddQueue)
```

集合Kafka

```
#基于receiver接收Kafka数据
val kafkaStream = KafkaUtils.createStream(streamingContext,
     [ZK quorum], [consumer group id], [per-topic number of Kafka partitions to
consume])

#直接读取方法
KafkaUtils.createDirectStream
```