day9笔记

简历

1. 所有看到的项目都不能用
2. 第一个项目重点在流计算
3. 也可以通过spark完成流计算：通过redis存储历史数据、评估报告；借助于json解析工具/字节数组

   dstream.map(log=>{

   1.判断log是登录日志还是评估日志

   if(登录日志){

   读取历史数据（到redis中读取）、把日志转换成评估数据对象、创建空的评估报告

   走评估链，生成一个评估报告

   把生成的评估报告写入到redis，同时设置一个过期时间

   }esle{

   把日志转换成登录成功数据对象、读取历史数据（到redis中读取）

   走更新链，生成新的历史数据；写入到redis中（以应用名:用户名作为key）

   }

   })
4. 所有的项目都需要写出来：用文字，把项目背景、个人职责全部描述处理
5. 专业技能写上去的就一定能讲出来
6. 技术单独出现：RDB/NOSQL

要求：

1. 把简历修改完毕之后，直接打印;自己确认一遍，两个人交换确认一遍
2. 项目描述写完打印

面试官，您好。我叫XXX,什么时间毕业于什么学校什么专业，在大学期间学习了什么什么课程。在毕业的时候，通过什么方式入职到了什么公司。**在这公司中主要使用了什么技术，做了什么项目。**在什么时间因为什么原因离职，到什么公司。**在这个公司用了什么技术，做了什么项目。**因为什么原因，离职啦。看到了贵公司在招聘，感觉凭我的经验、知识储备以及专业技能，比较符合公司的要求，来面试了。感谢贵公司给我这一次面试的机会，我相信，接下来的面试，肯定不会让面试官失望。也相信，如果有幸入职到贵公司，一定能给贵公司带来价值。这就是我的自我介绍，谢谢。

写出来

状态可查询

1. 需要在服务器中设置
   - 在配置文件中开启状态可查询

- 把flink提供的状态可查询依赖复制到lib目录下（opt目录中）

2. 在代码中，把对应的状态设置为可查询

   在状态描述者的方法里面设置

3. 把项目打包，部署到服务器，就可以实现状态的可查询

描述状态可查询的运行架构

queryableStateClient

queryableStateClientProxy

queryableStateServer

状态可查询的运行流程

发布状态可查询接口

1. flink支持第三方程序直接到服务器中获取到可查询的状态
2. 就项目来说，不允许业务系统直接读取flink服务器

   发布一个接口，可以供业务系统远程调用

   1. 写一个springboot项目
   2. 接收业务系统发送过来的登录uuid（如果要做控制，还应该接收业务系统用户名、密码）
   3. 响应出去一个评估报告（json）

TTL

评估报告在状态中设置了状态可查询，为了让业务系统读取

业务系统读取了状态中存储的评估报告，评估报告就没有任何存储的意义啦

评估报告就应该设置一个时间，过期了自动清除掉===》TTL

给评估报告状态设置一个TTL

运行起来之后发现可查询的状态不支持TTL

```
public void enableTimeToLive(StateTtlConfig ttlConfig) {
    Preconditions.checkNotNull(ttlConfig);
    Preconditions.checkArgument(
        condition: ttlConfig.getUpdateType() != StateTtlConfig.UpdateType.Disabled &&
            queryableStateName == null,
        errorMessage: "Queryable state is currently not supported with TTL");
    this.ttlConfig = ttlConfig;
}
```

可查询的状态不支持ttl

```
2021-07-12 16:04:57,169 INFO  org.apache.flink.runtime.taskmanager.Task                    - Map -> Filter -> Sink: Unnamed (2/2)
(a2105a298954588a592df12f0ed81e94) switched from RUNNING to FAILED.
java.lang.IllegalArgumentException: Queryable state is currently not supported with TTL
    at org.apache.flink.util.Preconditions.checkArgument(Preconditions.java:139)
    at org.apache.flink.api.common.state.StateDescriptor.enableTimeToLive(StateDescriptor.java:269)
    at com.baizhi.ure.job.MyMapFunction.open(UserRiskEvaluateJob.scala:105)
    at org.apache.flink.api.common.functions.util.FunctionUtils.openFunction(FunctionUtils.java:36)
    at org.apache.flink.streaming.api.operators.AbstractUdfStreamOperator.open(AbstractUdfStreamOperator.java:102)
```

解决方案：

1. 不设置TTL
2. 在规定的时间，把数据清除掉

   可以使用processFunction完成业务的功能计算，不用mapFunction

   在processFunction里面可以通过onTimer显式清除状态中的数据

```
evaluateReportState.clear()//似乎把所有的数据都清楚掉啦
```

```
//获取到listState里面的第一个元素
    val iterable: lang.Iterable[String] = listState.get()

    val scalaIterable: Iterable[String] = iterable.asScala
    val list: scala.List[String] = scalaIterable.toList

    val javaList: util.List[String] = list.asJava
    val uuid: String = javaList.get(0)//取listState中的第一个元素


    //从评估报告状态中，把要过去的那一个移除掉
    evaluateReportState.remove(uuid)

    //同时把listState中的数据更新一下
    //把ScalaList转换成buffer，从里面把第一个移除掉
    val buffer: mutable.Buffer[String] = list.toBuffer
    buffer.remove(0)
    val list1: scala.List[String] = buffer.toList

    listState.update(list1.asJava);
```

Checkpoint

为了故障恢复使用的，在环境中设置对应的checkpoint

复习

1. checkpoints的机制：barrier，二段提交机制（预提交处理、jobmanager完成提交）
2. checkpoint对应的state backend
3. 对比savepoint

当有问题的时候，就可以通过checkpoint目录实现数据的恢复

准备spark环境

把spark环境安装起来；参考之前上课的笔记，以standalone为例完成环境安装。这个环境就是为了程序部署

> 准备工作：正常安装JDK、Hadoop(启动hdfs)
>
> 1）上传并解压spark安装包

```
[root@spark1 opt]# tar -zxf spark-2.4.3-bin-hadoop2.7.tgz
[root@spark1 opt]# mv spark-2.4.3-bin-hadoop2.7 spark
```

> 2) 修改配置文件

```
[root@spark1 opt]# cd spark/conf
[root@spark1 conf]# mv slaves.template slaves
[root@spark1 conf]# mv spark-env.sh.template spark-env.sh
[root@spark1 conf]# vi slaves
#配置Spark集群节点主机名
spark1
[root@spark1 conf]# cat spark-env.sh
#声明Spark集群中Master的主机名和端口号
SPARK_MASTER_HOST=spark1
SPARK_MASTER_PORT=7077
```

> 3) 在spark中配置JAVA_HOME

```
[root@spark1 conf]# cd ..
[root@spark1 spark]# cd sbin
[root@spark1 sbin]# vi spark-config.sh
#在最后增加 JAVA_HOME 配置
export JAVA_HOME=/opt/jdk1.8
```

> 4) 启动spark

```
[root@spark1 spark]# sbin/start-all.sh
[root@spark1 spark]# jps
2054 Jps
2008 Worker
1933 Master
```

创建spark项目

- 添加spark-sql依赖

```
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.4.3</version>
</dependency>
```

- 写代码

```
package com.baizhi.spark.statistics

import java.text.SimpleDateFormat
import java.util.Properties

import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

object CityLoginCountStatistics {



  def main(args: Array[String]): Unit = {
    //准备spark环境
    //读取数据--》RDD
    //RDD--》DataFrame
    //table操作

    //集成jdbc；把结果写入到RDBMS

    val spark: SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("SparkDemo").getOrCreate()

    //为了能够把RDD转换成DataFrame，需要引入一个隐式转换
    import spark.implicits._

    /*println("************")

    val frame: DataFrame =
spark.sqlContext.read.parquet("hdfs://projectCentOS.baizhiedu.com:9000/flink-result/2020-
06-23").limit(20)


    frame.show();
    println("********************")*/

//    val line: RDD[String] = spark.sparkContext.textFile("file:///d:/a.log")
//    val line: RDD[String] =
spark.sparkContext.textFile("hdfs://projectCentOS.baizhiedu.com:9000/log/")



    val line: RDD[String] =
```

```
spark.sparkContext.textFile("hdfs://projectCentOS.baizhiedu.com:9000/flink-result/2020-06-23")


    val sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    val lineT: RDD[(String, String, String, String, String, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean)] = line.map(_.split("\\s+"))
      .map(ts => (ts(0), ts(1), sdf.format(ts(3).toLong), ts(4), ts(5), ts(6).toBoolean, ts(7).toBoolean, ts(8).toBoolean, ts(9).toBoolean, ts(10).toBoolean, ts(11).toBoolean, ts(12).toBoolean))


    //把RDD转换成DataFrame
    val dataFrame: DataFrame = lineT.toDF("appName","userIdentify","evaluateTime","cityName","geoPoint","area","device","inputFeature","similarity","speed","timeslot","total");

    //注册一个表
    dataFrame.createOrReplaceTempView("t_report")

    var start="2020-06-20"
    var end="2020-06-23"

    var sql=
      s"""
       select appName,cityName,count(*) from (select * from t_report where evaluateTime between '${start}' and '${end}')
       group by appName,cityName
      """


    //为了能够使用常量函数，需要引入
    import org.apache.spark.sql.functions._
    /*spark.sql(sql)
    .withColumn("start",lit(start))
      .withColumn("end",lit(end))
      .show();*/

    //1.把数据写入到CSV文件中，通过sqoop工具把CSV文件导入到mysql
    //2.spark可以集成JDBC

    val frame: DataFrame = spark.sql(sql)
      .withColumn("start", lit(start))
      .withColumn("end", lit(end))

    val props: Properties = new Properties()
    props.put("user","root")
    props.put("password","root")
    var url:String="jdbc:mysql://localhost:3306/bigdata?useUnicode=true&characterEncoding=utf8"
    var tableName:String="t_report"


    frame.write.mode(SaveMode.Append).jdbc(url,tableName,props)
```

```
  }

}
```

- 评估因子触发率

  通过case when then完成对应的统计

  通过某一个时间段，评估因子的触发率。先对数据进行时间段分析，以appName进行分组，然后统计每一个评估因子true的个数占总个数的百分比

- 把项目部署到spark
  - 把mysql驱动传输到spark的jars目录下
  - 修改代码中的spark远程地址以及mysql数据库连接的远程地址

    ```
    al spark: SparkSession = SparkSession.builder()
          .master("spark://projectcentos.baizhiedu.com:7077")
          .appName("SparkDemo").getOrCreate()

     var url:String="jdbc:mysql://192.168.45.5:3306/bigdata?
    useUnicode=true&characterEncoding=utf8"
    ```

  - 把项目打成jar包，并传输到spark服务器某一个目录下
  - 把mysql数据库设置为允许远程访问
  - 通过执行spark文件运行项目

    ```
    [root@projectCentOS spark-2.4.3-bin-hadoop2.7]# bin/spark-submit --master
    spark://projectcentos.baizhiedu.com:7077 --class
    com.baizhi.spark.statistics.CityLoginCountStatistics /opt/code/OffLineAnalysis-1.0-
    SNAPSHOT.jar
    ```

  - 到mysql数据库中确认已经把数据导入到了mysql里面