尚硅谷电商用户画像二

版本：V 1.1

张晨

# 第五章 开发任务一：

# 统计型和规则型标签的SQL处理

## 5.1 总体处理流程

目前虽然填写了一个标签任务，但是并不是光靠一个SQL就能生成标签的，还要结合任务定义中的其他规则来计算标签数据。

那么这个 “定义+规则+SQL=标签” 这个步骤需要一个spark程序Jar包来完成，并且存储在HDFS。

系统在每日的凌晨0点或者手动会把对应的标签任务生成为待执行的任务进程。

达到任务进程的要求时（1 任务执行时点、2任务执行层级），会推送JSON格式的作业提交信息传给远端的任务提交器。

任务提交器会根据JSON组合生成spark-submit及相关参数，提交YARN。

YARN会根据提交时的JAR包路径生成Spark作业。

Spark作业执行阶段会读取MySQL库中标签定义、规则和SQL，去查询Hive中的数据通过计算生成标签最后写入画像库的某张标签表。

在Spark作业提交后，远程提交器会一直监控该作业的运行状态并通过回调，传递给用户画像管理平台，用于监控。



## 5.2 任务目标

完成Spark作业执行的JAR包。

读取标签定义及规则，组合成SQL，在数仓中执行，并写入画像标签库。

## 5.3 设计分析

* 要完成的执行步骤
* 查询任务信息
* 查询标签信息
* 查询子标签对应规则
* 生成标签表（如果不存在）
* 组合写入标签SQL
* 执行SQL

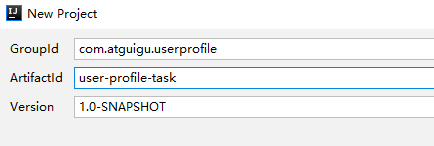
## 5.4 代码实现

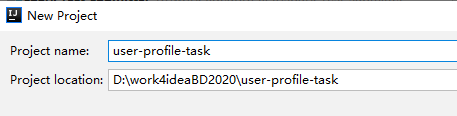
数据库准备，在hive中建立数据库

|  |
| --- |
| create database user\_profile2077 |

### 5.4.1 搭建工程

在idea中新建项目 命名为user-profile-task





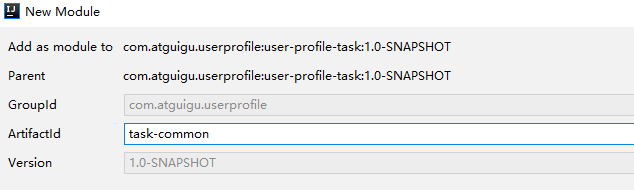
### 5.4.2 pom.xml

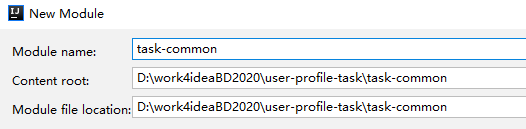
增加如下依赖

|  |
| --- |
| <**properties**>  <**spark.version**>3.0.0</**spark.version**>  <**scala.version**>2.12.11</**scala.version**> </**properties**>  <**dependencies**>   <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-core\_2.12</**artifactId**>  <**version**>${spark.version}</**version**>  *<!-- provider如果存在，那么运行时该Jar包不存在，也不会打包到最终的发布版本中，只是编译器有效 -->* <**scope**>provided</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-sql\_2.12</**artifactId**>  <**version**>${spark.version}</**version**>  <**scope**>provided</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.lz4</**groupId**>  <**artifactId**>lz4-java</**artifactId**>  <**version**>1.7.1</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.anarres.lzo</**groupId**>  <**artifactId**>lzo-hadoop</**artifactId**>  <**version**>1.0.5</**version**>  <**exclusions**>  <**exclusion**>  <**groupId**>org.apache.hadoop</**groupId**>  <**artifactId**>hadoop-core</**artifactId**>  </**exclusion**>   </**exclusions**>  </**dependency**>  <**dependency**>  <**groupId**>org.scala-lang</**groupId**>  <**artifactId**>scala-library</**artifactId**>  <**version**>${scala.version}</**version**>  <**scope**>provided</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-hive\_2.12</**artifactId**>  <**version**>${spark.version}</**version**>  <**scope**>provided</**scope**>   </**dependency**>  <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-databind</**artifactId**>  <**version**>2.10.0</**version**>  </**dependency**>   <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-core</**artifactId**>  <**version**>2.10.0</**version**>  </**dependency**>   <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.47</**version**>  <**scope**>provided</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  <**version**>1.2.68</**version**>  </**dependency**>  <**dependency**>  <**groupId**>ru.yandex.clickhouse</**groupId**>  <**artifactId**>clickhouse-jdbc</**artifactId**>  <**version**>0.1.55</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.projectlombok</**groupId**>  <**artifactId**>lombok</**artifactId**>  <**version**>1.18.18</**version**> </**dependency**>  </**dependencies**> <**dependencyManagement**>  <**dependencies**>  <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-mllib\_2.11</**artifactId**>  <**version**>${spark.version}</**version**>  <**scope**>provided</**scope**>  </**dependency**>  </**dependencies**> </**dependencyManagement**> |
|  |

### 5.4.3 增加通用模块

在这个工程下未来要实现多个task子模块，这些task子模块需要一些通用的工具类和实体类。





加入Properties和MySQL工具类

MyPropertiesUtil

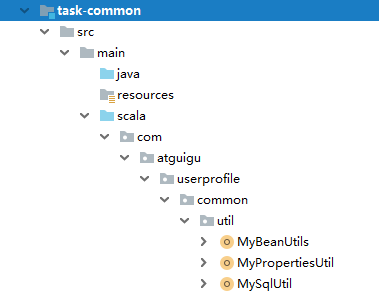
|  |
| --- |
| **import** java.io.InputStreamReader **import** java.util.Properties  **object** MyPropertiesUtil {    **def** main(args: Array[String]): Unit = {  **val** properties: Properties = MyPropertiesUtil.*load*(**"config.properties"**)  *println*(properties.getProperty(**"kafka.broker.list"**))  }   **def** load(propertieName:String): Properties ={  **val** prop=**new** Properties();  prop.load(**new** InputStreamReader(Thread.*currentThread*().getContextClassLoader.  getResourceAsStream(propertieName) , **"UTF-8"**))  prop  }  } |

MySqlUtil

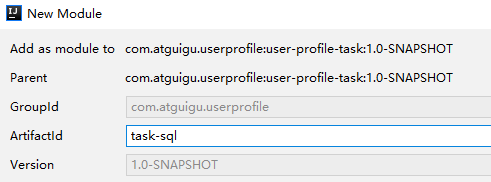
|  |
| --- |
| **import** java.lang.reflect.Field **import** java.sql.{Connection, DriverManager, PreparedStatement, ResultSet, ResultSetMetaData, Statement} **import** java.util.Properties  **import** com.alibaba.fastjson.JSONObject **import** com.google.common.base.CaseFormat  **import** scala.collection.mutable.ListBuffer  **object** MySqlUtil {   **def** main(args: Array[String]): Unit = {  **val** list: java.util.List[JSONObject] = *queryList*(**"select \* from base\_province"**)  *println*(list)  }   **private val** *properties*: Properties = MyPropertiesUtil.*load*(**"config.properties"**)    **val** *MYSQL\_URL* = *properties*.getProperty(**"mysql.url"**)  **val** *MYSQL\_USERNAME* = *properties*.getProperty(**"mysql.username"**)  **val** *MYSQL\_PASSWORD* = *properties*.getProperty(**"mysql.password"**)    **def** queryList(sql: String): java.util.List[JSONObject] = {  Class.*forName*(**"com.mysql.jdbc.Driver"**)  *//创建结果列表* **val** resultList: java.util.List[JSONObject] = **new** java.util.ArrayList[JSONObject]()  *//创建连接* **val** conn: Connection = DriverManager.*getConnection*(*MYSQL\_URL*, *MYSQL\_USERNAME*, *MYSQL\_PASSWORD*)  *//创建会话* **val** stat: Statement = conn.createStatement  *println*(sql)  *//提交sql 返回结果* **val** rs: ResultSet = stat.executeQuery(sql)  *//为了获得列名 要取得元数据* **val** md: ResultSetMetaData = rs.getMetaData  **while** (rs.next) {  **val** rowData = **new** JSONObject();  **for** (i <- 1 to md.getColumnCount) {  *// 根据下标得到对应元数据中的字段名，以及结果中值* rowData.put(md.getColumnName(i), rs.getObject(i))  }  resultList.add(rowData)  }   stat.close()  conn.close()  resultList  }    **def** queryList[T<:AnyRef](sql: String, clazz: Class[T], underScoreToCamel: Boolean):List[T]={   Class.*forName*(**"com.mysql.jdbc.Driver"**);  **val** resultList: ListBuffer[T] = **new** ListBuffer[T]();  **val** connection: Connection = DriverManager.*getConnection*(*MYSQL\_URL*, *MYSQL\_USERNAME*, *MYSQL\_PASSWORD*)  **val** stat: Statement = connection.createStatement();  **val** rs: ResultSet = stat.executeQuery(sql);  **val** md: ResultSetMetaData = rs.getMetaData();  **while** (rs.next()) {  **val** obj: T = clazz.newInstance()  **for** (i <- 1 to md.getColumnCount) {  **var** propertyName: String = md.getColumnLabel(i)   **if** (underScoreToCamel) {  propertyName = CaseFormat.*LOWER\_UNDERSCORE*.to(CaseFormat.*LOWER\_CAMEL*, md.getColumnLabel(i))  }  MyBeanUtils.*setV*(obj,propertyName,rs.getObject(i))   }  resultList.append(obj);  }  stat.close()  connection.close()  resultList.toList   }   **def** queryOne[T<:AnyRef](sql: String, clazz: Class[T] ,  underScoreToCamel: Boolean): Option[T ] ={  *println*(sql)  Class.*forName*(**"com.mysql.jdbc.Driver"**);  **val** resultList: ListBuffer[T] = **new** ListBuffer[T]();  **val** connection: Connection =  DriverManager.*getConnection*(*MYSQL\_URL*, *MYSQL\_USERNAME*, *MYSQL\_PASSWORD*)  **val** stat: Statement = connection.createStatement();  **val** rs: ResultSet = stat.executeQuery(sql);  **val** md: ResultSetMetaData = rs.getMetaData();  **while** (rs.next()) {  **val** obj: T = clazz.newInstance()    **for** (i <- 1 to md.getColumnCount) {  **var** propertyName: String = md.getColumnName(i)  **if** (underScoreToCamel) {  propertyName =  CaseFormat.*LOWER\_UNDERSCORE*.to(CaseFormat.*LOWER\_CAMEL*, md.getColumnName(i))  }  MyBeanUtils.*setV*(obj,propertyName,rs.getObject(i))  }  resultList.append(obj);  }  stat.close()  connection.close()  **if**(resultList!=**null**){  Some(resultList(0))  }**else**{  None  }   }   **def** insertOne[T](sql: String, obj:T ): Unit ={  Class.*forName*(**"com.mysql.jdbc.Driver"**);  **val** resultList: ListBuffer[T] = **new** ListBuffer[T]();  **val** connection: Connection =  DriverManager.*getConnection*(*MYSQL\_URL*, *MYSQL\_USERNAME*, *MYSQL\_PASSWORD*)  **val** pstat: PreparedStatement = connection.prepareStatement(sql)  **val** fields: Array[Field] = obj.getClass.getDeclaredFields   **for**(i<- 1 to fields.size){  **val** field:Field = fields(i-1);  field.setAccessible(**true**)  **val** value: AnyRef = field.get(obj)  pstat.setObject(i,value)  }  **val** resInsert = pstat.execute()  *println*(**"插入数据，返回值=>"** + resInsert)   }  } |

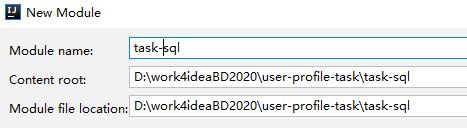
MyBeanUtils

|  |
| --- |
| **import** java.lang.reflect.{Field, Modifier} **import** java.text.SimpleDateFormat **import** java.util.Locale **import** scala.util.control.Breaks.\_  **object** MyBeanUtils {     */\*\*  \* 对象相同属性copy  \*  \** **@param obj** *\** **@param toResult** *\** **@return** *\** **@throws** *Exception  \* 转换报错  \*/* **def** copyProperties(obj: Object, toResult: Object): Unit = {  **if** (obj == **null**) {  **return** }  **try** {  **val** fields = toResult.getClass.getDeclaredFields  **for** (field <- fields) {  breakable {  field.setAccessible(**true**) *//修改访问权限* **if** (Modifier.*isFinal*(field.getModifiers()))  break  **if** (*isWrapType*(field)) {  **val** getMethodName = field.getName()  **val** setMethodName = field.getName()+**"\_$eq"  val** getMethod = {  **try** {  obj.getClass().getMethod(getMethodName)  } **catch** {  **case** ex: Exception =>  *// println(ex)* break  }  }   *//从源对象获取get方法* **val** setMethod =  toResult.getClass.getMethod(setMethodName, field.getType)  *//从目标对象获取set方法* **val** value = {  **val** objValue = getMethod.invoke(obj) *// get 获取的是源对象的值* **if** (objValue != **null** && objValue.isInstanceOf[java.util.Date]) {  *// GMT时间转时间戳* **val** format =  **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**, Locale.*ENGLISH*)  **val** formatValue = format.format(objValue)  java.lang.Long.*valueOf*(format.parse(formatValue).getTime)  } **else** objValue  }  setMethod.invoke(toResult, value)  }  }  }  } **catch** {  **case** ex: Exception => **throw** ex  }  }   */\*\*  \* 是否是基本类型、包装类型、String类型  \*/* **def** isWrapType(field: Field): Boolean = {  **val** typeList = *List*[String](**"java.lang.Integer"** , **"java.lang.Double"**, **"java.lang.Float"**, **"java.lang.Long"**, **"java.util.Optional"** , **"java.lang.Short"**, **"java.lang.Byte"**, **"java.lang.Boolean"**, **"java.lang.Char"** , **"java.lang.String"**, **"int"**, **"double"**, **"long"** , **"short"**, **"byte"**, **"boolean"**, **"char"**, **"float"**)  **if** (typeList.contains(field.getType().getName())) **true else false** }    **def** getV(ref:AnyRef, name: String): Any = ref.getClass.getMethods.find(\_.getName == name).get.invoke(ref)   **def** setV(ref:AnyRef,name: String, value: Any): Unit = {  ref.getClass.getMethods.find(\_.getName == name + **"\_$eq"**).get.invoke(ref, value.asInstanceOf[AnyRef])  }  } |



### 5.4.4 增加task-sql模块





#### pom.xml

增加以下依赖及工具

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**artifactId**>task-common</**artifactId**>  <**groupId**>com.atguigu.userprofile</**groupId**>  <**version**>1.0-SNAPSHOT</**version**>  </**dependency**> </**dependencies**>  <**build**>  <**plugins**>  *<!-- 该插件用于将Scala代码编译成class文件 -->* <**plugin**>  <**groupId**>net.alchim31.maven</**groupId**>  <**artifactId**>scala-maven-plugin</**artifactId**>  <**version**>3.4.6</**version**>  <**executions**>  <**execution**>  *<!-- 声明绑定到maven的compile阶段 -->* <**goals**>  <**goal**>compile</**goal**>  <**goal**>testCompile</**goal**>  </**goals**>  </**execution**>  </**executions**>  </**plugin**>   <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-assembly-plugin</**artifactId**>  <**version**>3.0.0</**version**>  <**configuration**>  <**descriptorRefs**>  <**descriptorRef**>jar-with-dependencies</**descriptorRef**>  </**descriptorRefs**>  </**configuration**>  <**executions**>  <**execution**>  <**id**>make-assembly</**id**>  <**phase**>package</**phase**>  <**goals**>  <**goal**>single</**goal**>  </**goals**>  </**execution**>  </**executions**>  </**plugin**>  </**plugins**> </**build**> |

#### 增加config.properties

|  |
| --- |
| **hdfs-store.path**=**hdfs://bigdata01:8020/user\_profile data-warehouse.dbname**=**gmall2021 user-profile.dbname**=**user\_profile2077  *#mysql配置* mysql.url**=**jdbc:mysql://bigdata01:3306/user\_profile\_manager?characterEncoding=utf-8&useSSL=false mysql.username**=**root mysql.password**=**Flzx3qc!** |

#### 增加core-site.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?> <?***xml-stylesheet type="text/xsl" href="configuration.xsl"***?>* <**configuration**>  <**property**>  <**name**>io.compression.codecs</**name**>  <**value**>  org.apache.hadoop.io.compress.GzipCodec,  org.apache.hadoop.io.compress.DefaultCodec,  org.apache.hadoop.io.compress.BZip2Codec,  org.apache.hadoop.io.compress.SnappyCodec,  com.hadoop.compression.lzo.LzoCodec,  com.hadoop.compression.lzo.LzopCodec  </**value**>  </**property**>  <**property**>  <**name**>io.compression.codec.lzo.class</**name**>  <**value**>com.hadoop.compression.lzo.LzoCodec</**value**>  </**property**> </**configuration**> |

#### 增加hdfs-site.xml

|  |
| --- |
| <**configuration**>   *<!-- nn web端访问地址-->* <**property**>  <**name**>dfs.namenode.http-address</**name**>  <**value**>bigdata01:9870</**value**>  </**property**>  *<!-- 2nn web端访问地址-->* <**property**>  <**name**>dfs.namenode.secondary.http-address</**name**>  <**value**>bigdata02:9868</**value**>  </**property**>  <**property**>  <**name**>dfs.client.use.datanode.hostname</**name**>  <**value**>true</**value**>   </**property**>  </**configuration**> |

#### 增加hive-site.xml

|  |
| --- |
| *<?***xml version="1.0"***?> <?***xml-stylesheet type="text/xsl" href="configuration.xsl"***?>* <**configuration**>  <**property**>  <**name**>javax.jdo.option.ConnectionURL</**name**>  <**value**>jdbc:mysql://bigdata01:3306/metastore?createDatabaseIfNotExist=true**&amp;**characterEncoding=utf-8**&amp;**useSSL=false</**value**>  <**description**>JDBC connect string for a JDBC metastore</**description**>  </**property**>   <**property**>  <**name**>javax.jdo.option.ConnectionDriverName</**name**>  <**value**>com.mysql.jdbc.Driver</**value**>  <**description**>Driver class name for a JDBC metastore</**description**>  </**property**>   <**property**>  <**name**>javax.jdo.option.ConnectionUserName</**name**>  <**value**>root</**value**>  <**description**>username to use against metastore database</**description**>  </**property**>   <**property**>  <**name**>javax.jdo.option.ConnectionPassword</**name**>  <**value**>Flzx3qc!</**value**>  <**description**>password to use against metastore database</**description**>  </**property**>  <**property**>  <**name**>hive.cli.print.header</**name**>  <**value**>true</**value**>  </**property**>   <**property**>  <**name**>hive.cli.print.current.db</**name**>  <**value**>true</**value**>  </**property**>  <**property**>  <**name**>hive.metastore.schema.verification</**name**>  <**value**>false</**value**>  </**property**>  </**configuration**> |

#### 增加log4j.properties

|  |
| --- |
| log4j.rootLogger=error, stdout,R log4j.appender.stdout=org.apache.log4j.ConsoleAppender log4j.appender.stdout.layout=org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %5p --- [%50t] %-80c(line:%5L) : %m%n  log4j.appender.R=org.apache.log4j.RollingFileAppender log4j.appender.R.File=../log/agent.log log4j.appender.R.MaxFileSize=1024KB log4j.appender.R.MaxBackupIndex=1  log4j.appender.R.layout=org.apache.log4j.PatternLayout log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %5p --- [%50t] %-80c(line:%6L) : %m%n |

#### 实体类

在task-common中增加实体类

TagInfo

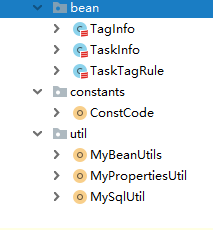
|  |
| --- |
| **package** com.atguigu.userprofile.bean  **import** java.util.Date **import** java.lang.Long **case class** TagInfo(**var** id:Long,  **var** tagCode:String,  **var** tagName:String,  **var** tagLevel:Long,  **var** parentTagId:Long,  **var** tagType:String,  **var** tagValueType:String,  **var** tagValueLimit:Long,  **var** tagValueStep:Long,  **var** tagTaskId:Long,  **var** tagComment:String,  **var** createTime:Date,  **var** updateTime:Date  ) {  **def this**() ={  **this**(**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null**,**null** )  } } |

|  |
| --- |
| **package** com.atguigu.userprofile.bean  **import** java.util.Date **import** java.lang.Long **case class** TaskInfo (**var** id: Long = **null** ,  **var** taskName:String=**null**,  **var** taskStatus:String=**null**,  **var** taskComment:String=**null**,  **var** taskTime:String=**null**,  **var** taskType:String=**null**,  **var** execType:String=**null**,  **var** mainClass:String=**null**,  **var** fileId:Long= **null**,  **var** taskArgs:String=**null**,  **var** taskSql:String=**null**,  **var** taskExecLevel:Long =**null**,  **var** createTime:Date=**null**) {   ***//补充无参构造函数* def this**() ={  **this**(0L,**null**,**null**,**null**,**null**,**null**,**null**,**null**,-1L,**null**,**null**,**null**,**null**)  } } |

|  |
| --- |
| **package** com.atguigu.userprofile.bean  **import** java.lang.Long **case class** TaskTagRule(**var** id:Long,  **var** tagId:Long,  **var** taskId:Long,  **var** queryValue:String,  **var** subTagId:Long,  **var** subTagValue:String  ) {  **def this**() ={  **this**(**null**,**null**,**null**,**null**,**null**,**null** )  }  } |

#### 增加常量类

|  |
| --- |
| **object** ConstCode {   **val** TAG\_VALUE\_TYPE\_LONG=**"1"  val** TAG\_VALUE\_TYPE\_DECIMAL=**"2"  val** TAG\_VALUE\_TYPE\_STRING=**"3"  val** TAG\_VALUE\_TYPE\_DATE=**"4"    val** TASK\_PROCESS\_SUCCESS=**"1"  val** TASK\_PROCESS\_ERROR=**"2"   val** TASK\_STAGE\_START=**"1"  val** TASK\_STAGE\_RUNNING=**"2"** } |



#### 增加 查询 TaskInfo、TagInfo、TaskTagRule的DAO

|  |
| --- |
| **import** com.atguigu.userprofile.common.bean.TagInfo **import** com.atguigu.userprofile.common.util.MySqlUtil  **object** TagInfoDAO {   **def** getTagInfoByTaskId(taskId:String): TagInfo ={  **val** tagInfoSql: String =  **s"""select id,tag\_code,tag\_name,  | parent\_tag\_id,tag\_type,tag\_value\_type,  | tag\_value\_limit,tag\_task\_id,tag\_comment,  | create\_time  | from tag\_info  | where tag\_task\_id=$**taskId**"""**.stripMargin  **val** tagInfoOpt: Option[TagInfo] =  MySqlUtil.*queryOne*(tagInfoSql, *classOf*[TagInfo], **true**)  **var** tagInfo: TagInfo = **null**;  **if** (tagInfoOpt != None) {  tagInfo = tagInfoOpt.get  } **else** {  **throw new** RuntimeException(**"no tag for task\_id : "**+taskId)  }  tagInfo  }  } |

|  |
| --- |
| **object** TaskInfoDAO {   **def** getTaskInfo( taskId:String): TaskInfo ={  **val** taskInfoSql: String =  **s"""select id,task\_name,  | task\_status,task\_comment,task\_time ,  | task\_type,exec\_type,main\_class,file\_id,  | task\_args,task\_sql,task\_exec\_level,create\_time  | from task\_info where id=$**taskId**"""**.stripMargin  **val** taskInfoOpt: Option[TaskInfo] =  MySqlUtil.*queryOne*(taskInfoSql, *classOf*[TaskInfo], **true**)  **var** taskInfo: TaskInfo = **null**;  **if** (taskInfoOpt != None) {  taskInfo = taskInfoOpt.get  } **else** {  **throw new** RuntimeException(**"no task for task\_id : "**+taskId)  }  taskInfo  }  } |

|  |
| --- |
| **import** com.atguigu.userprofile.common.bean.TaskTagRule **import** com.atguigu.userprofile.common.util.MySqlUtil  **object** TaskTagRuleDAO {   **def** getTaskTagRuleListByTaskId(taskId:String): List[TaskTagRule] ={   **val** taskRuleSql: String =  **s"""select tr.id,tr.tag\_id,tr.task\_id,tr.query\_value,  | sub\_tag\_id,ti.tag\_name as sub\_tag\_value  | from task\_tag\_rule tr,tag\_info ti  | where tr.sub\_tag\_id=ti.id and tr.task\_id=$**taskId**"""**.stripMargin  **val** taskTagRuleList: List[TaskTagRule] =  MySqlUtil.*queryList*(taskRuleSql, *classOf*[TaskTagRule], **true**)  taskTagRuleList  } } |

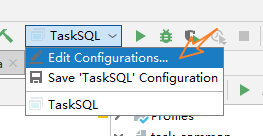
#### 任务主程序

|  |
| --- |
| **object** TaskSQLApp{   **def** main(args: Array[String]): Unit = {   **val** properties: Properties = MyPropertiesUtil.*load*(**"config.properties"**)   **val** userProfileDbName = properties.getProperty(**"user-profile.dbname"**)  **val** wareHouseDbName = properties.getProperty(**"data-warehouse.dbname"**)  **val** hdfsStorePath = properties.getProperty(**"hdfs-store.path"**)  ***//1 初始化环境* val** sparkConf: SparkConf = **new** SparkConf().setAppName(**"tag\_sql\_app"**).setMaster(**"local[\*]"**)  **val** sparkSession: SparkSession = SparkSession.*builder*().config(sparkConf).enableHiveSupport().getOrCreate()   ***// 第一个参数 会传入任务编号* val** taskId: String = args(0)  ***// 第二个参数 会传入任务的业务日期* val** taskDate: String = args(1)    ***// 查询任务信息* val** taskInfo: TaskInfo = TaskInfoDAO.*getTaskInfo*(taskId)   ***// 查询标签信息* val** tagInfo: TagInfo = TagInfoDAO.*getTagInfoByTaskId*(taskId)   ***// 查询标签规则信息* val** taskTagRuleList: List[TaskTagRule] = TaskTagRuleDAO.*getTaskTagRuleListByTaskId*(taskId)   ***// 根据 标签名称建表 ，如果没有就建表 根据字段类型  //用tagcode作为表名  //根据标签的值类型来 定义字段类型* val** tableName: String = tagInfo.tagCode  **val** tagValueType: String = tagInfo.tagValueType **match** {  **case** ConstCode.*TAG\_VALUE\_TYPE\_STRING* => **"STRING"  case** ConstCode.*TAG\_VALUE\_TYPE\_LONG* => **"BIGINT"  case** ConstCode.*TAG\_VALUE\_TYPE\_DECIMAL* => **"DECIMAL(16,2)"  case** ConstCode.*TAG\_VALUE\_TYPE\_DATE* => **"STRING"** }     **val** createTableSql = **s"""create table if not exists $**userProfileDbName**.$**tableName **| ( uid string, tag\_value $**tagValueType **)  | comment '$**{tagInfo.tagName}**' PARTITIONED BY (`dt` STRING)  | ROW FORMAT DELIMITED FIELDS TERMINATED BY '\\t'  | LOCATION '$**hdfsStorePath**/$**userProfileDbName**/$**{tableName.toLowerCase}**/'"""**.stripMargin   ***// sparkSession.sql("select \* from gmall2021.dwd\_order\_info ").show(20)*** sparkSession.sql(**" use "** + userProfileDbName)  sparkSession.sql(createTableSql)   ***//提取任务sql  //处理时间* var** taskSql: String = taskInfo.taskSql.replace(**"${dt}"**, taskDate)  *println*(taskSql)   ***//组合SQL* val** insertSqlBuilder: StringBuilder =  **new** StringBuilder(**"insert overwrite table "** + userProfileDbName + **"."** + tableName.toLowerCase  + **" partition (dt='"** + taskDate + **"' )"**)  **val** selectSqlBuilder: StringBuilder = **new** StringBuilder(**" select uid, "**)  **if**(taskTagRuleList.size>0){  selectSqlBuilder.append(**"case query\_value "**)  **for** (taskTagRule <- taskTagRuleList) {  selectSqlBuilder.append(**" when '"** + taskTagRule.queryValue + **"' "** +  **"then "** + *checkQuota*(tagInfo.tagValueType, taskTagRule.subTagValue) + **""**)  }  selectSqlBuilder.append(**" end as tagValue from ("**)  .append(taskSql).append(**") as user\_value\_view"**)  }**else**{  selectSqlBuilder.append(**" query\_value as tagValue from ("**)  .append(taskSql).append(**") as user\_value\_view"**)  }  **val** insertTagSql: String = insertSqlBuilder.append(selectSqlBuilder).toString()  ***//执行SQL*** sparkSession.sql(**" use "** + wareHouseDbName)  sparkSession.sql(insertTagSql)   }   ***//根据类型加单引* def** checkQuota(tagValueType: String, subTagValue: String): String = {  **if** (tagValueType == ConstCode.*TAG\_VALUE\_TYPE\_STRING* || tagValueType == ConstCode.*TAG\_VALUE\_TYPE\_DATE*) {  **"'"** + subTagValue + **"'"** } **else** {  subTagValue  }  } |

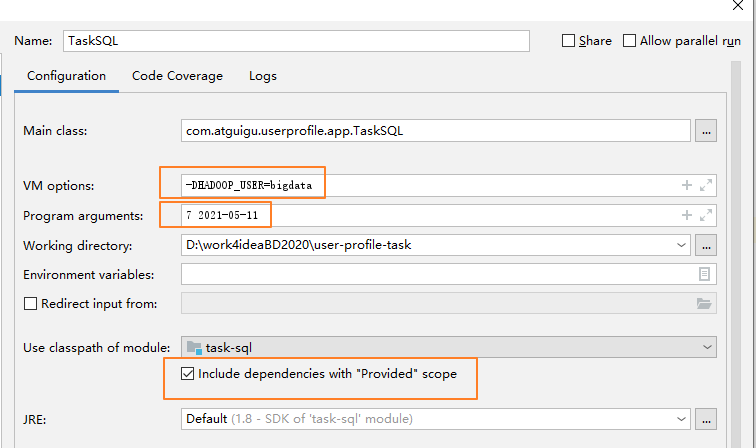
## 5.5 调试运行

在执行task-sql模块前，先把task-common这个模块在maven中进行install。

首次运行时，要对执行参数进行一定的配置。（如果下图的选项没出来可以先执行一下TaskSQLApp，然后立刻停掉。）

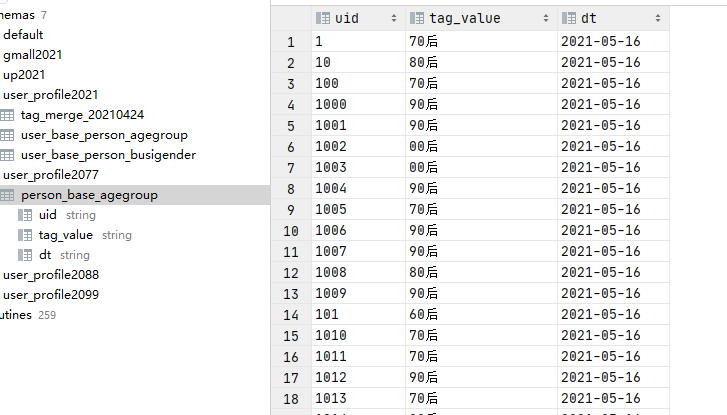


如下图：



* VM options: 后面要注明HDFS的用户，不然会报访问权限不足
* Program arguments: 后面要跟两个参数第一个是数据库中task\_info 的id ,第二个是执行业务日期。
* 勾选 “Provided” scope 表示在Maven依赖中标记了Provided的依赖虽然不会打入程序包，但是调试运行的时候还是要加载的。

运行成功后 ，去查看user\_profile2077 下是否有 person\_base\_agegroup 这张表，并且数据是否正确。



## 5.6 任务发布

### 5.6.1 准备工作

1. 首先要把程序中的.setMaster(“local[\*]”) 注释掉

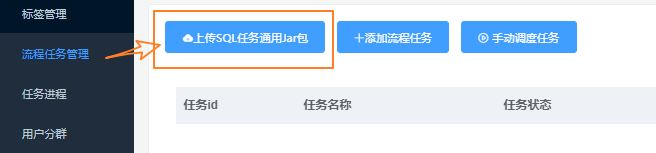


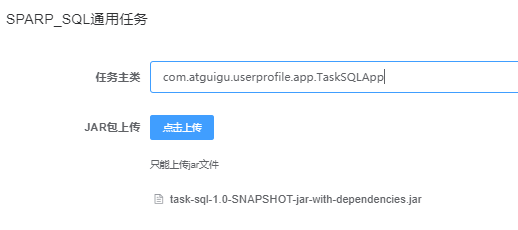
2） 要把user-profile-task模块和task-common模块执行maven的install

3）把task-sql 这个模块用maven进行package。

### 5.6.2 程序上传

在流程任务管理栏位中点击左上角的【上传SQL任务通用jar包】





任务主类填写通用Jar包的执行main方法的类路径

同时上传对应的jar包。

**通用标签通用SQL专用表**

|  |  |
| --- | --- |
| 字段 | 说明 |
| id | 主键 |
| task\_file\_id | 关联的文件id |
| main\_class | 任务主类 |
| update\_time | 最后更新时间 |

**目前该表实质上只存一行数据**

**文件表**

|  |  |
| --- | --- |
| 字段 | 说明 |
| id | 主键 |
| file\_name | 文件名称 |
| file\_ex\_name | 文件扩展名 |
| file\_path | 文件路径 |
| file\_system | 文件系统 |
| file\_status | 是否生效 |
| create\_time | 创建时间 |

## 5.7 启动调度

### 5.7.1 调度前确保：

* + hdfs集群启动
  + yarn 集群启动
  + 远程提交器启动

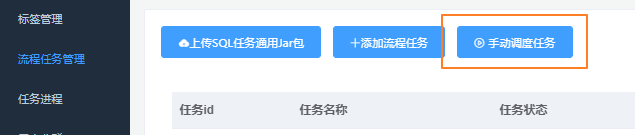
#### 远程提交器的启动

在spark-rest-submitter-xxx.jar的同级目录执行

|  |
| --- |
| nohup java -jar spark-rest-submitter-0.0.2-SNAPSHOT.jar >submitter.log 2>&1 & |

### 提交任务

在【流程任务管理】中点击【指定日期执行】



填写目前数仓最新的业务日期：



点击确定。

去【任务进度】中查看已经提交的任务，



注意：任务进程并不一定直接就提交到YARN。

还要看另外两个条件：

1 同一批次是否还有任务层级更低的任务

2 是否达到执行时间点。

达到条件后任务状态会从“TODO”变为”START” ，最终变为FINISHED或者FAILED

可以通过回调产生的yarn应用id去Yarn控制台查看对应的作业具体错误日志。



## 5.8 结果验证

