TStarSDK

## 一、概述

为方便用户基于TStar平台进行业务开发，平台提供了SDK，本文是TStarSDK使用手册，旨在帮助用户了解和学习如何使用TStarSDK进行快速的业务开发。TStarSDK包括三部分：1、网关（Gateway），2、处理流（Processes），3、离线计算（OP）。为更加直观的理解SDK的使用，SDK中已提供DEMO，且DEMO均以天泽信息模拟协议[PLANA（协议详见附件）](#_附录)为例。本SDK所提供的示例都是有效可运行的。

## 二、目录结构

|- bin -----------------------------------------------------离线计算命令工具

|-run.sh --------------------------------------------------离线计算程序执行脚本

|- conf -----------------------------------------------------配置文件目录

|- common----------------------------------------------公共配置文件

|- op-------------------------------------------------------离线计算配置文件

|- gateways----------------------------------------------网关配置文件

|- process------------------------------------------------处理流配置文件

|- dist -------------------------------------------------------用户jar文件目录

|- external-lib --------------------------------------------程序依赖的第三方jar包

|- op------------------------------------------------------离线计算依赖第三方jar包

|- gateways----------------------------------------------网关依赖三方jar包

|- process------------------------------------------------处理流依赖三方jar包

|- lib -------------------------------------------------------- 平台提供的依赖包

|- common----------------------------------------------公共配置jar包

|- op------------------------------------------------------ 离线计算依赖jar包

|- gateways----------------------------------------------网关依赖jar包

|- process------------------------------------------------处理流依赖jar包

|- opSrc ------------------------------------------------------离线计算源代码

|- gwSrc ------------------------------------------------------网关源代码

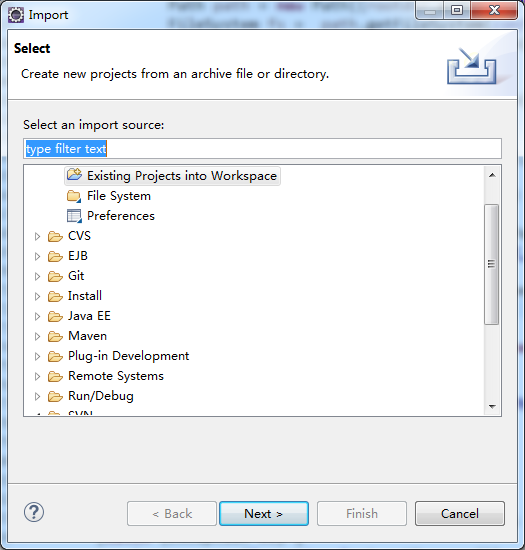
|- procSrc ----------------------------------------------------处理流源代码

|- build.properties------------------------------------------编译配置参数

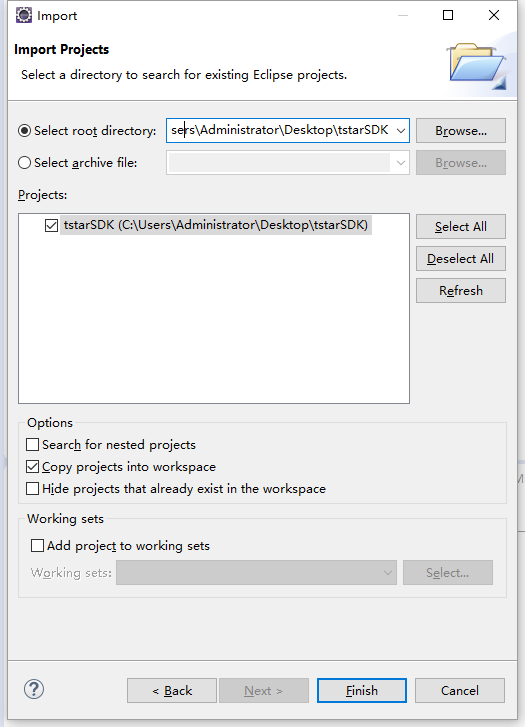
|- build.xml --------------------------------------------------Apache Ant工具build文件

## 二、SDK导入

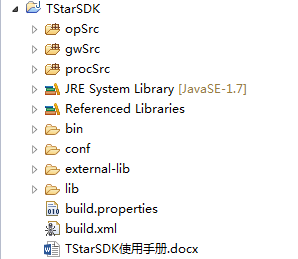
导入eclipse，步骤File -> Import -> Existing Projects into Workspace



单击Next –> Browse，选择TStarSDK工程，单击Finish。



导入后Eclipse目录如下所示：



注：SDK初始导入到eclipse中的project名称是TStarSDK,如果用户需要修改project name的话 ，需要把build.xml中的project name同时修改。

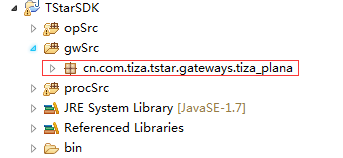
## 三、网关（Gateway）

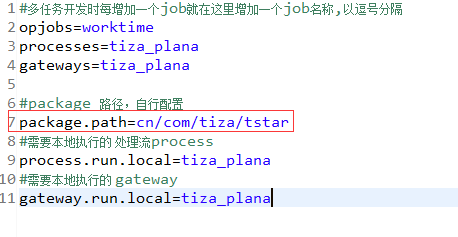
网关是TStar平台和终端交互的唯一接口，任何上行数据（终端->网关）、下行数据（网关->终端）都要经过网关的初步加工，因此网关程序的开发是TStar平台的首要开发任务。经过多年的物联网项目经验总结，TStar Gateway已经对网络层高并发的IO处理进行了充分的封装，使得即使不懂NIO的开发人员，只需要充分理解通讯协议就能很快的进行TStar网关模块的开发，大大减少了数据接入开发的工作量。以下步骤将详细介绍如何使用SDK进行网关开发，网关协议使用PLANA协议，详见[附录](#_附录)。

### 3.1、开发

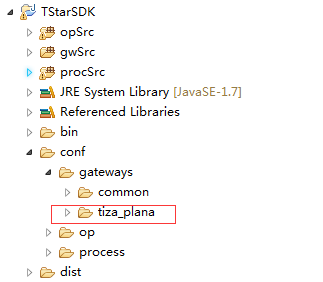
#### 3.1.1、约定

开发网关时需要把代码放在gwSrc目录下，支持多个不同协议类型网关同时开发，不同协议网关代码需放在不同包路径下，包名称约定如下:xxx.xxx.xxx.gateways.{协议类型名称}。以天泽信息PLANA协议网关代码为例，其包名称为cn.com.tiza.tstar.gateways.tiza\_plana,其中tiza\_plana就是PLANA的协议类型名称，而gateways之前的包名称则需配置到build.properties文件中的package.path属性，如下图：

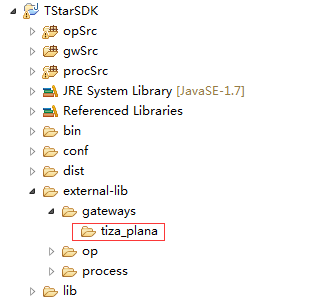




然后在conf/gateways目录下增加配置文件目录，目录名称必须为{协议类型名称}。如PLANA网关的配置文件目录为：conf/gateways/tiza\_plana，如下图：



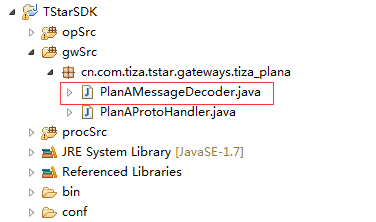
如果开发过程中需要使用第三方的jar包，则需要在external-lib/gateways目录下创建相应的第三方jar包目录，目录名称必须为{协议类型名称}。如PLANA网关使用到的第三方jar包目录为：external-lib/gateways/tiza\_plana，如下图：



#### 3.1.2、实现

##### 3.1.2.1、实现自定义协议解码器

TStar平台已经提供一些标准协议的解码器，比如MQTT、JSON、XML等，若使用这些协议用户则不需要自行实现协议解码器，通过在AdminUI上配置相应解码器即可。如果用户使用的协议为自定义协议，则需编写协议解码器（Decoder）。用户的协议解码器只需要一个类，该类需要继承cn.com.tiza.tstar.gateway.codec.CustomDecoder类并重写decode()方法，decode()方法实现将数据流解析成数据包。以PLANA协议为例，在cn.com.tiza.tstar.gateways.tiza\_plana 包下新建类PlanAMessageDecoder ，如下图：

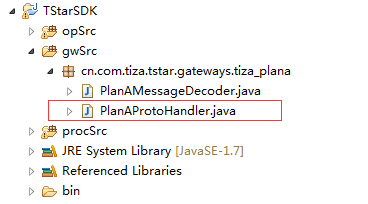


PlanAMessageDecoder代码如下：

|  |
| --- |
| **public** **class** PlanAMessageDecoder **extends** CustomDecoder {  **private** **final** **static** **byte** *PROTOCOL\_HEAD* = 0x7E;  **private** **final** **static** **byte** *PROTOCOL\_TAIL* = 0x7E;  **private** **final** **static** **byte** *MIN\_LENGTH* = 17;    /\* 协议相关，该方法必须实现将上行数据流组装成一个完整的消息数据包，生成的数据必须为ByteBuf对象。\*/  @Override  **protected** **void** decode(ChannelHandlerContext ctx, ByteBuf in,  List<Object> out) **throws** Exception {    **if** (in.readableBytes() < *MIN\_LENGTH*) {  **return**;  }    in.markReaderIndex();  **byte** head = in.readByte();    **if** (head != *PROTOCOL\_HEAD*) {  **return**;  }    **int** start = in.readerIndex();  in.readShort();  **int** length = in.readUnsignedShort();    in.readBytes(**new** **byte**[10]);    **if** (in.readableBytes() < (length+2)) {  in.resetReaderIndex();  **return**;  }    in.readBytes(**new** **byte**[length+1]);  **byte** tail = in.readByte();    **if** (tail != *PROTOCOL\_TAIL*) {  **return**;  }  **int** end = in.readerIndex();  **byte**[] msg = **new** **byte**[end - start + 1];  in.resetReaderIndex().readBytes(msg);    ByteBuf bb = Unpooled.*wrappedBuffer*(msg);  out.add(bb);  }  } |
|  |

##### 3.1.2.2、实现自定义数据解析器

进行协议解码后的数据包为一段二进制字节码，用户需根据协议解析出终端ID，命令ID，命令序号，消息体等并封装成TStarData，该部分工作由数据解析器负责（ProtoHandler）。数据解析器只需要一个类，该类需要继承cn.com.tiza.tstar.gateway.handler.BaseUserDefinedHandler类并必须重写handleRecvMessage(ChannelHandlerContext ctx, ByteBuf buf)方法，另有几个可选方法可以被覆盖，可根据实际需求进行实现。以PLANA协议为例，在cn.com.tiza.tstar.gateways.tiza\_plana 包下新建类PlanAProtoHandler，如下图：



PlanAProtoHandler代码如下：

|  |
| --- |
| **public** **class** PlanAProtoHandler **extends** BaseUserDefinedHandler {  **private** **static** **final** Logger *LOG* = LoggerFactory  .*getLogger*(PlanAProtoHandler.**class**);  // 将收到的终端数据解析并封装为TStarData  @Override  **public** TStarData handleRecvMessage(ChannelHandlerContext ctx, ByteBuf buf) {  **byte**[] msg = **new** **byte**[buf.readableBytes()];  buf.getBytes(0, msg);  TStarData tstarData = **null**;  tstarData = **new** TStarData();  tstarData.setMsgBody(msg);  buf.readByte();  **int** cmd = buf.readUnsignedShort();  tstarData.setCmdID(cmd);  buf.readShort();  **byte**[] terminalBCD = **new** **byte**[6];  buf.readBytes(terminalBCD);  **long** terminalLong = 0;  **for** (**int** i = 0; i < 6; i++) {  **int** temp = terminalBCD[i] & 0xff;  terminalLong = terminalLong \* 10 + (temp >> 4);  terminalLong = terminalLong \* 10 + (temp & 0x0F);  }  String terminalID = String.*valueOf*(terminalLong);  tstarData.setTerminalID(terminalID);  **int** cmdSerial = buf.readInt();  tstarData.setCmdSerialNo(cmdSerial);  **long** currTime = System.*currentTimeMillis*();  tstarData.setTime(currTime);  *LOG*.debug("TStarData:{}", tstarData);  **return** tstarData;  }  // 判断是否为应答指令，是则封装为AckData，否则返回null  @Override  **public** AckData ackHandle(ChannelHandlerContext ctx, TStarData msg) {  **byte**[] msgBody = msg.getMsgBody();  AckData ackData = **null**;  **if** (msg.getCmdID() == 0x0201) {  **int** respSerial = ((msgBody[17] & 0xFF) << 8) + msgBody[18] & 0xFF;  **if** (*LOG*.isDebugEnabled()) {  *LOG*.debug("Terminal:{} ack:0x0201, ackCmdSerial:{}", msg.getTerminalID(),  respSerial);  }  **int** respCmd = 0x8201;  ackData = **new** AckData(msg, respCmd, respSerial);  }  **return** ackData;  }  // 网关读写都空闲时触发此事件  @Override  **public** **void** allIdleHandle(ChannelHandlerContext ctx) {  System.*out*.println("allIdle");  }  // 网关给终端下发指令时触发此事件  @Override  **public** **void** commandReceived(ChannelHandlerContext ctx, CommandData cmd) {  System.*out*.println(cmd);  }  // 当终端第一次登陆网关时触发此事件，用户判断终端是否合法login，是返回true，否则false  @Override  **public** **boolean** login(ChannelHandlerContext ctx, TStarData data) {  System.*out*.println(data.getTerminalID() + " login");  **return** **true**;  }  // 网关判断终端超时logout触发此事件  @Override  **public** **void** logout(String terminalId) {  System.*out*.println(terminalId + " logout");  }  // 网关读空闲时触发此事件  @Override  **public** **void** readIdleHandle(ChannelHandlerContext ctx) {  System.*out*.println("readIdle");  }  // 网关写空闲时触发此事件  @Override  **public** **void** writeIdleHandle(ChannelHandlerContext ctx) {  System.*out*.println("writeIdle");  }  } |

#### 3.1.3、配置

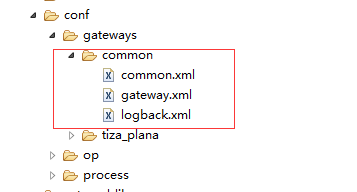
配置文件主要包括公共配置和协议私有配置，公共配置文件在conf/gateways/common目录下，该目录包以下含几个配置文件：

1）gateway.xml，网关公有配置文件

2）common.xml，tstar平台公有配置文件

3）logback.xml，日志配置文件

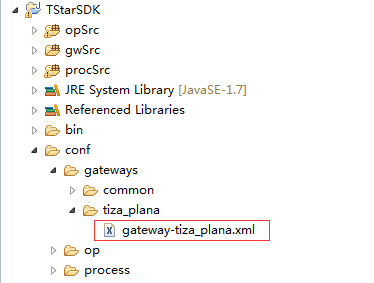
如图：



协议私有配置文件在conf/gateways/{协议类型名称}目录下：

1）配置文件名称格式：gateway-{协议类型名称}.xml

以PLANA协议为例，应在conf/gateways/tiza\_plana目录下创建gateway-tiza\_plana.xml文件。如图：



这些配置文件仅在本地测试运行时需要，提交到TStar平台时由AdminUI自动生成。

gateway-tiza\_plana.xml配置文件内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd"*>  <!-- CustomProtoHandler配置，用户只需修改class属性为用户实现的CustomProtoHandler即可 -->  <bean id=*"CustomProtoHandler"* class=*"cn.com.tiza.tstar.gateway.tiza\_plana.PlanAProtoHandler"* scope=*"prototype"* />  <!—CustomDecoder配置，用户需修改class属性为用户实现的CustomDecoder即可 -->  <bean id=*"CustomDecoder"* class=*"cn.com.tiza.tstar.gateway.tiza\_plana.PlanAMessageDecoder"* scope=*"prototype"* />  <!--网关配置，启动监听端口、协议类型、是否需要转发等配置 -->  <bean id=*"ProtocolManager"* class=*"cn.com.tiza.tstar.gateway.core.ProtocolManager"*>  <!-- 网关接收数据的端口 -->  <property name=*"port"*>  <value>8090</value>  </property>    <!-- 网关通信协议类型 -->  <property name=*"commType"*>  <!-- TCP,UDP,HTTP,WEBSOCKET -->  <value>TCP</value>  </property>    <!-- 是否为MQTT协议 -->  <property name=*"isMqtt"*>  <value>false</value>  </property>    <!-- 数据是否需要转发，如果不需要转发数据，下面几项配置可以忽略 -->  <property name=*"forward"*>  <value>false</value>  </property>    <!-- 转发目标地址 -->  <property name=*"forwardIp"*>  <value>192.168.1.1</value>  </property>    <!-- 转发目标端口-->  <property name=*"forwardPort"*>  <value>6066</value>  </property>    <!-- 转发重连时间间隔,单位秒 -->  <property name=*"forwardReconnInterval"*>  <value>30</value>  </property>    </bean>  </beans> |
|  |

公共配置文件内容如下：

gateway.xml:

|  |
| --- |
| <configuration>  <property>  <name>tstar.gateway.version</name>  <value>1</value>  <desc>0:简化版 1:标准版</desc>  </property>  <!-- Override common.xml -->  <property>  <name>tstar.common.database.pool.initial.connections</name>  <value>10</value>  </property>  <property>  <name>tstar.gateway.ip</name>  <value>192.168.10.216</value>  <description>网关ip地址</description>  </property>  <property>  <name>tstar.gateway.forward.queue.size</name>  <value>10000</value>  <description>网关转发数据缓冲队列大小，默认大小10000</description>  </property>  <property>  <name>tstar.gateway.reader.idle.seconds</name>  <value>0</value>  <description>网关读空闲时间，默认0s</description>  </property>    <property>  <name>tstar.gateway.writer.idle.seconds</name>  <value>0</value>  <description>网关写空闲时间，默认0s</description>  </property>    <property>  <name>tstar.gateway.all.idle.seconds</name>  <value>0</value>  <description>网关读写都空闲时间，默认0s</description>  </property>  <property>  <name>tstar.gateway.terminal.idle.seconds</name>  <value>60</value>  <description>终端允许空闲的最大时间，单位秒，默认时间间隔60s</description>  </property>    <property>  <name>tstar.gateway.statistics.interval</name>  <value>60</value>  <description>网关统计时间间隔，单位秒，默认时间间隔60s</description>  </property>  <property>  <name>tstar.gateway.terminal.statistics.interval</name>  <value>300</value>  <description>终端统计时间间隔，单位秒，默认时间间隔300s</description>  </property>  <property>  <name>tstar.gateway.intraday.statistic.interval</name>  <value>600</value>  <description>网关当天统计时间间隔，单位秒，默认时间间隔600s</description>  </property>  <property>  <name>tstar.gateway.kafka.downstream.iswrite</name>  <value>true</value>  <description>网关下行数据是否写入kafka，默认true</description>  </property>    <property>  <name>tstar.gateway.alarm.period</name>  <value>60</value>  <description>网关报警检查周期</description>  </property>    <property>  <name>tstar.gateway.data.access.boss.thread.num</name>  <value>2</value>  <description>网关数据接收线程，默认为2</description>  </property>  <property>  <name>tstar.gateway.data.access.worker.thread.num</name>  <value>2</value>  <description>网关数据处理线程，默认为2</description>  </property>    <!-- Kafka -->  <property>  <name>tstar.gateway.kafka.metadata.broker.list</name>  <value>tzcdh188:9092,tzcdh189:9092:tzcdh190:9092</value>  <description>Kafka broker list</description>  </property>    <!-- redis -->  <property>  <name>tstar.gateway.redis.reconnect.cnt</name>  <value>3</value>  <description>redis操作重试次数</description>  </property>    <!-- SSL加密配置 -->  <property>  <name>tstar.gateway.issecure</name>  <value>false</value>  </property>  <property>  <name>tstar.gateway.securechat.protocal</name>  <value>SSL</value>  </property>  <property>  <name>tstar.gateway.securechat.istowway</name>  <value>true</value>  </property>  <property>  <name>tstar.gateway.securechat.keystore.path</name>  <value></value>  </property>  <property>  <name>tstar.gateway.securechat.keystore.password</name>  <value></value>  </property>  <property>  <name>tstar.gateway.securechat.key.password</name>  <value></value>  </property>    <!-- 单向验证时不用配置 -->  <property>  <name>tstar.gateway.securechat.keystoretrust.path</name>  <value></value>  </property>  <property>  <name>tstar.gateway.securechat.keystoretrust.password</name>  <value></value>  </property>  </configuration> |
|  |

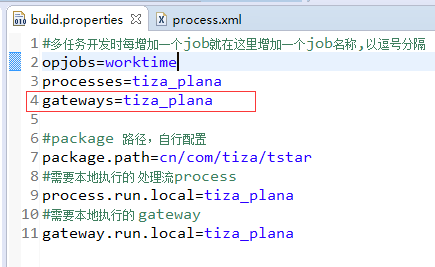
common.xml:

|  |
| --- |
| <configuration>  <property>  <name>tstar.common.database.driver</name>  <value>com.mysql.jdbc.Driver</value>  </property>  <property>  <name>tstar.common.database.url</name>  <value>jdbc:mysql://192.168.103.95:3306/tstar103</value>  </property>  <property>  <name>tstar.common.database.username</name>  <value>temp</value>  </property>  <property>  <name>tstar.common.database.password</name>  <value>temp</value>  </property>  <property>  <name>tstar.common.database.pool.max.connections</name>  <value>50</value>  </property>  <property>  <name>tstar.common.database.pool.initial.connections</name>  <value>1</value>  </property>  <property>  <name>tstar.common.database.pool.incremental</name>  <value>5</value>  </property>  <property>  <name>tstar.common.alarmer.root.dir</name>  <value>/tstar/alarm</value>  </property>  <!-- Redis basic configurations -->  <property>  <name>tstar.common.redis.host</name>  <value>192.168.103.202</value>  <descrption>Redis server host.</descrption>  </property>  <property>  <name>tstar.common.redis.port</name>  <value>6379</value>  <descrption>Redis server port.</descrption>  </property>  <property>  <name>tstar.common.redis.password</name>  <value></value>  <descrption>Redis server password.</descrption>  </property>  <property>  <name>tstar.common.redis.db</name>  <value>0</value>  <descrption>Redis server db. (0,1,2...)</descrption>  </property>  <!-- Redis client pool configurations -->  <property>  <name>tstar.common.redis.pool.maxTotal</name>  <value>100</value>  <descrption>Redis pool max connections.</descrption>  </property>  <property>  <name>tstar.common.redis.pool.maxIdle</name>  <value>100</value>  <descrption>Redis pool max idle connections.</descrption>  </property>  <property>  <name>tstar.common.redis.pool.maxWaitMillis</name>  <value>5000</value>  <descrption>Max wait time in milliseconds when borrowing a jedis instance from jedisPool.</descrption>  </property>  </configuration> |

logback.xml使用默认即可。

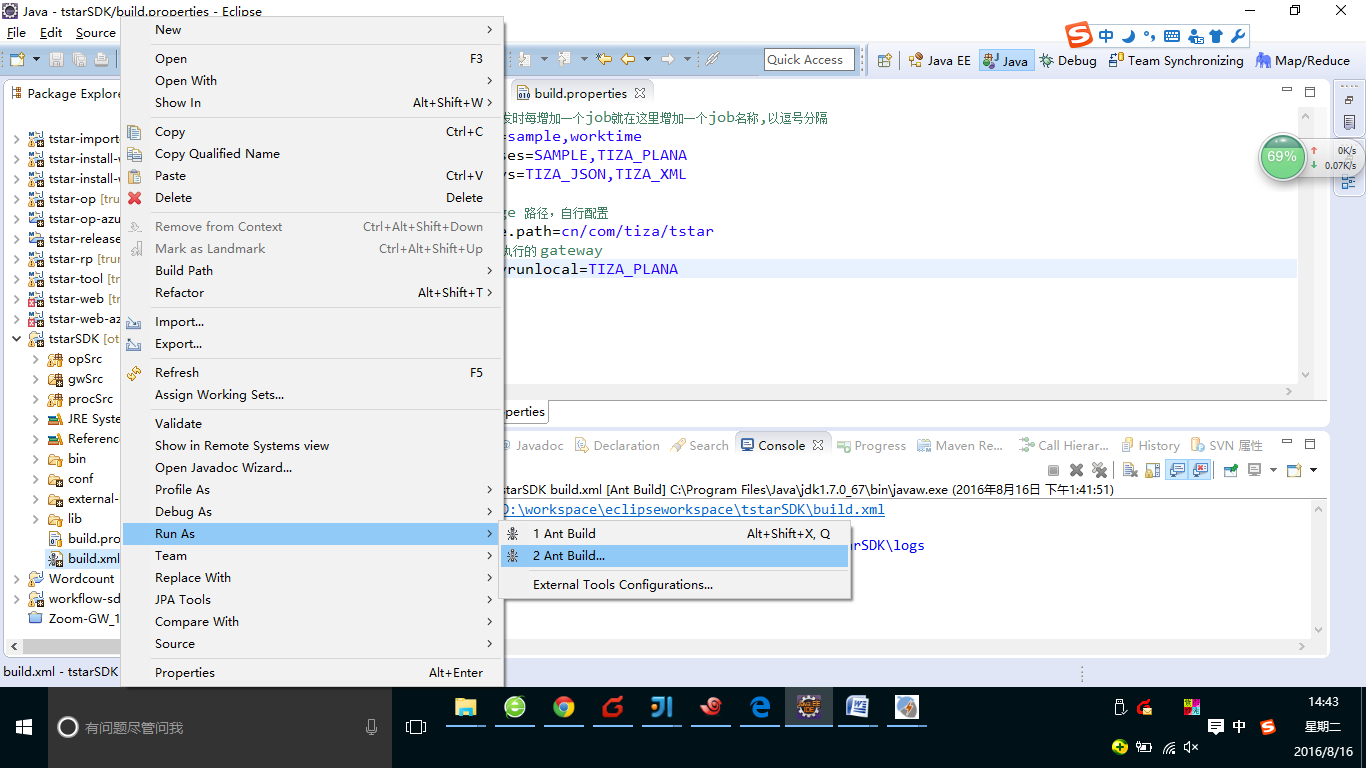
### 3.2、编译

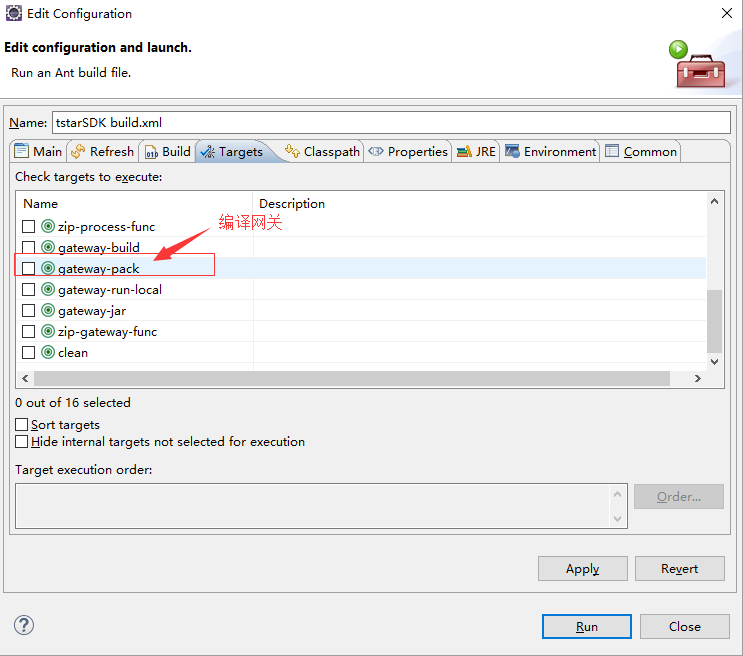
网关代码开发完成以后，使用Ant工具进行编译打包，用户需要将编译的网关对应的协议类型名称配置到build.properties文件中的gateways这个变量中，多个协议类型的网关同时编译需以逗号隔开。运行Ant build target会将对应协议class文件、配置文件目录，第三方jar包目录自动打包生成zip包。以PLANA协议网关编译为例，build.properties文件内容如下：



**build步骤如下**：

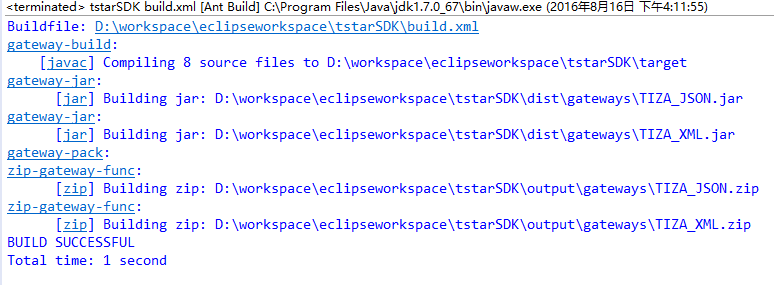
单击build.xml -> Run As -> Ant Build…。



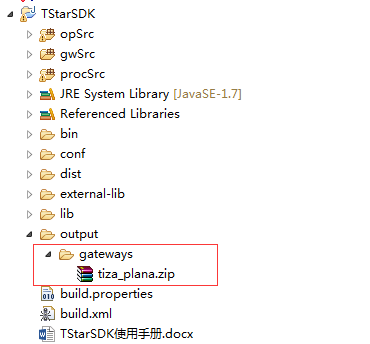


选择要编译的target，点击Run。

出现BUILD SUCCESSFUL 即表示编译成功。



编译打包的结果会存在根目录下面的output目录下：



### 3.3、网关本地测试

##### 3.3.1测试说明

由于网关测试依赖TStar环境，所以测试前提是有可用的TStar平台环境。TStar平台的部署安装可参考TStar安装部署文档.docx。

##### 3.3.2环境准备

网关依赖的TStar平台组件有kafka、mysql、Redis，所以需要将提供相关服务的机器名和IP地址，添加到系统hosts文件中。

##### 3.3.3注册终端类型

终端注册可通过AdminUI操作完成，详见TStar-AdminUI用户手册.doc

##### 3.3.4网关配置

1、修改gateway-{协议类型名称}.xml、gateway.xml、common.xml配置详见[3.1.3 配置文件](#_配置文件)

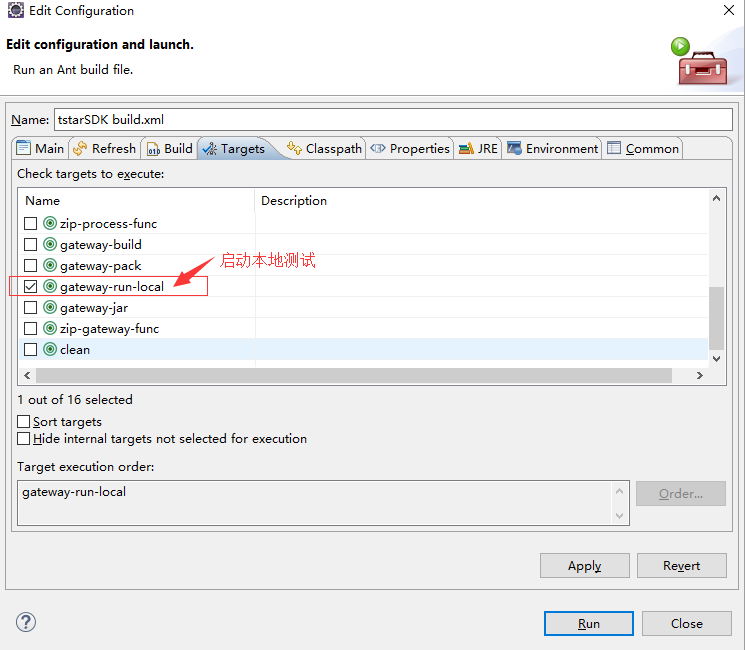
2、修改build.properties 文件中的gateway.run.local配置。



##### 3.3.5启动网关测试

网关启动：

右键单击build.xml -> Run As -> Ant Build，勾选run target



点击run即启动网关进行测试。

启动成功：

11.png

用户可根据协议自行编写模拟终端发送数据进行测试。模拟终端连接网关以后通过AdminUI管理系统可以查看网关状态，并查看通讯数据记录。详见TStar-AdminUI用户手册.doc

##### 3.3.6停止网关

执行Ant脚本run target，网关进程会在后台运行，可通过Windows任务管理器停止进程。



选择java.exe，点击结束进程即可停止网关。

### 3.4、运行

网关运行部署详见 admin UI 使用手册。

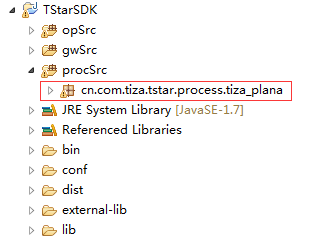
## 四、处理流（Processes）

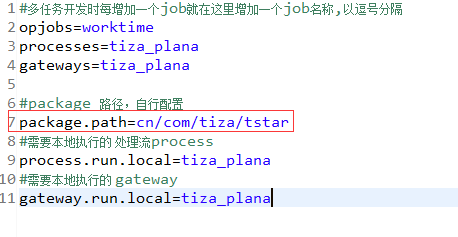
TStar实时计算使用的Storm流式计算框架，处理流的概念是将多个实时计算中的数据处理模块串联起来进行数据处理。为了使用户能够基于TStar平台快速进行处理流开发，TStar平台屏蔽了Storm中的相关概念，使不懂storm的开发人员也可以进行处理模块的开发，在每个模块中用户只需要关心本模块数据处理逻辑即可。SDK提供了多种数据落地方式，包括rdbms、kafka、redis，用户只需要调用相应接口就可以使用。

### 4.1、开发

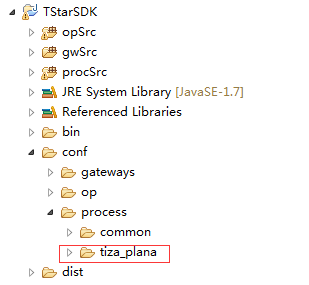
#### 4.1.1、约定

开发处理流时需要把代码放在procSrc目录下，支持多个不同协议类型处理流同时开发，不同协议处理流代码需放在不同包路径下，包名称约定如下:xxx.xxx.xxx.process.{协议类型名称}。以天泽信息PLANA协议处理流代码为例，其包名称为cn.com.tiza.tstar.process.tiza\_plana,其中tiza\_plana就是PLANA的协议类型名称，而process之前的包名称则需配置到build.properties文件中的package.path属性，如下图：

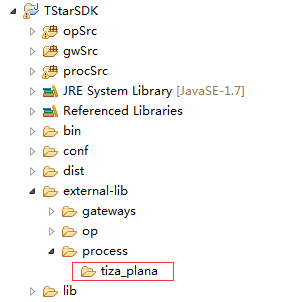




然后在conf/process目录下增加配置文件目录，目录名称必须为{协议类型名称}。如PLANA处理流的配置文件目录为：conf/process/tiza\_plana，如下图：



如果开发过程中需要使用第三方的jar包，则需要在external-lib/process目录下创建相应的第三方jar包目录，目录名称必须为{协议类型名称}。如PLANA网关使用到的第三方jar包目录为：external-lib/process/tiza\_plana，如下图：



#### 4.1.2、实现

下面以PLANA协议的解析为例来介绍下基于SDK开发process的步骤。

首先在cn.com.tiza.tstar.process.tiza\_plana包下新建一个解析类PlanaParser并继承TStar的cn.com.tiza.tstar.common.process.BaseHandle类,并重写handle方法和init方法，handle方法实现业务代码，init方法实现一些初始化的配置，用户的配置项可以用this.processorConf获取。

如果处理结果需要发送给下一个Processor处理，则需要在handle方法中将数据封装成RPTuple并返回，若数据到达最后一个Processor，则可返回null值。RPTuple包含一个Map<String, String>类型的context属性，用户可把解析之后的数据以及上下文的数据放入context，如报警数据，发送给报警模块处理。

PlanaParser代码如下：

|  |
| --- |
| package cn**.**com**.**tiza**.**tstar**.**process**.**tiza\_plana**;**  public class PlanaParser **extends** BaseHandle **{**  private static final Logger LOG **=** LoggerFactory**.**getLogger**(**PlanaParser**.**class**);**  private long count**;**  private String topic**;**  private DecimalFormat df **=** **new** DecimalFormat**(**"#.0"**);**  @Override  public RPTuple handle**(**RPTuple tuple**)** **{**  count**++;**  **if** **(**count **%** 200000 **==** 0**)** **{**  LOG**.**info**(**"Terminal:{}, Cmd:{}, No:{}, Msg:{}"**,**  tuple**.**getTerminalID**(),** tuple**.**getCmdID**(),**  tuple**.**getCmdSerialNo**(),** byte2HexStr**(**tuple**.**getMsgBody**()));**  **}**  String terminalID **=** tuple**.**getTerminalID**();**  int cmdID **=** tuple**.**getCmdID**();**  long serial **=** tuple**.**getCmdSerialNo**();**  byte**[]** body **=** tuple**.**getMsgBody**();**  ByteBuffer buf **=** ByteBuffer**.**wrap**(**body**);**  **if** **(**cmdID **==** 0x0100**)** **{**  // 注册  buf**.**get**();** // 0x7E  buf**.**getShort**();**  buf**.**getShort**();**  buf**.**get**(new** byte**[**6**]);**  buf**.**getInt**();**  int log **=** buf**.**get**();**  **if** **(**count **%** 10000 **==** 0**)** **{**  LOG**.**info**(**"{} login"**,** terminalID**);**  **}**  StringBuilder parsedBody **=** **new** StringBuilder**();**  parsedBody**.**append**(**terminalID**).**append**(**"\t"**).**append**(**cmdID**)**  **.**append**(**"\t"**).**append**(**serial**).**append**(**"\t"**).**append**(**log**);**  **switch** **(this.**dataOutType**)** **{**  **case** 0**:**  **break;**  **case** 1**:**  **try** **{**  tuple**.**setMsgBody**(**parsedBody**.**toString**().**getBytes**(**"UTF-8"**));**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  LOG**.**error**(**""**,** e**);**  **}**  **super.**storeInKafka**(**tuple**,** **this.**tbname**);**  **break;**  **case** 2**:**  **try** **{**  tuple**.**setMsgBody**(**parsedBody**.**toString**().**getBytes**(**"UTF-8"**));**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  LOG**.**error**(**""**,** e**);**  **}**  saveInDB**(**tuple**);**  **break;**  **default:**  **break;**  **}**  **}** **else** **if** **(**cmdID **==** 0x0200**)** **{**  // 位置  buf**.**get**();** // 0x7E  buf**.**getShort**();**  int length **=** buf**.**getShort**();**  buf**.**get**(new** byte**[**6**]);**  buf**.**getInt**();**  double lon **=** **(**buf**.**getInt**()** **&** 0xFFFFFFFF**)** **/** 100.0**;**  double lat **=** **(**buf**.**getInt**()** **&** 0xFFFFFFFF**)** **/** 100.0**;**  int height **=** buf**.**getShort**();**  int speed **=** **(**buf**.**getShort**()** **&** 0xFFFF**)** **/** 10**;**  int direction **=** buf**.**getShort**();**  // TODO parse time  byte**[]** dataTimes **=** **new** byte**[**6**];**  buf**.**get**(**dataTimes**);**  String dataTime **=** **this.**getTimeStr**(**dataTimes**);**  int state **=** buf**.**getInt**();**  int alarm **=** buf**.**getInt**();**  double milleage **=** **(**buf**.**getInt**()** **&** 0xFFFFFFFF**)** **/** 1000.0**;**  milleage **=** Double**.**parseDouble**(**df**.**format**(**milleage**));**  int worktime **=** buf**.**getInt**()** **&** 0xFFFFFFFF**;**  int oil **=** buf**.**getShort**();**  int v **=** buf**.**getShort**();**  int t **=** buf**.**getShort**()** **+** 60**;**  int rpm **=** buf**.**getShort**();**  byte**[]** rev **=** **new** byte**[**length **-** 44**];**  buf**.**get**(**rev**);**  String info **=** ""**;**  **try** **{**  info **=** **new** String**(**rev**,** "UTF-8"**);**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  LOG**.**error**(**""**,** e**);**  **}**  **if** **(**count **%** 200000 **==** 0**)** **{**  LOG**.**info**(**"{}, {}, {}, {}, {}km/h, {}km, {}s"**,** terminalID**,**  dataTime**,** lon**,** lat**,** speed**,** milleage**,** worktime**);**  **}**  StringBuilder parsedBody **=** **new** StringBuilder**();**  parsedBody**.**append**(**terminalID**).**append**(**"\t"**).**append**(**cmdID**)**  **.**append**(**"\t"**).**append**(**serial**).**append**(**"\t"**).**append**(**lon**)**  **.**append**(**"\t"**).**append**(**lat**).**append**(**"\t"**).**append**(**height**)**  **.**append**(**"\t"**).**append**(**speed**).**append**(**"\t"**).**append**(**direction**)**  **.**append**(**"\t"**).**append**(**dataTime**).**append**(**"\t"**).**append**(**state**)**  **.**append**(**"\t"**).**append**(**alarm**).**append**(**"\t"**).**append**(**milleage**)**  **.**append**(**"\t"**).**append**(**worktime**).**append**(**"\t"**).**append**(**oil**)**  **.**append**(**"\t"**).**append**(**v**).**append**(**"\t"**).**append**(**t**).**append**(**"\t"**)**  **.**append**(**rpm**).**append**(**"\t"**).**append**(**info**);**  **switch** **(this.**dataOutType**)** **{**  **case** 0**:**  **break;**  **case** 1**:**  **try** **{**  tuple**.**setMsgBody**(**parsedBody**.**toString**().**getBytes**(**"UTF-8"**));**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  LOG**.**error**(**""**,** e**);**  **}**  **super.**storeInKafka**(**tuple**,** **this.**tbname**);**  **break;**  **case** 2**:**  **try** **{**  tuple**.**setMsgBody**(**parsedBody**.**toString**().**getBytes**(**"UTF-8"**));**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  LOG**.**error**(**""**,** e**);**  **}**  saveInDB**(**tuple**);**  **break;**  **default:**  **break;**  **}**  **}**  **return** tuple**;**  **}**  @Override  public void init**()** **throws** Exception **{**  count **=** 0**;**  **}**    private void saveInDB**(**TStarData tstarData**)** **{**  String sql **=** "insert into "  **+** **this.**tbname  **+** " (terminalID,cmdID,cmdSerialNo,time,msgBody) values (?,?,?,?,?)"**;**  List**<**Object**>** params **=** **new** ArrayList**<>();**  params**.**add**(**tstarData**.**getTerminalID**());**  params**.**add**(**tstarData**.**getCmdID**());**  params**.**add**(**tstarData**.**getCmdSerialNo**());**  params**.**add**(**tstarData**.**getTime**());**  params**.**add**(new** String**(**tstarData**.**getMsgBody**()));**  **try** **{**  **this.**storeInDB**(**sql**,** params**);**  **}**  **catch** **(**Exception e**)** **{**  LOG**.**error**(**"store TstarData into Rmdb error {} - {} - {}"**,**  tstarData**.**toString**(),**  e**.**getMessage**(),**  e**);**  **}**  **}**  private String byte2HexStr**(**byte**[]** data**)** **{**  StringBuffer buf **=** **new** StringBuffer**();**  buf**.**append**(**"[pos=0"**);**  buf**.**append**(**" lim="**);**  buf**.**append**(**data**.**length**);**  buf**.**append**(**" cap="**);**  buf**.**append**(**data**.**length**);**  buf**.**append**(**": "**);**  **for** **(**int i **=** 0**;** i **<** data**.**length**;** i**++)** **{**  buf**.**append**(**String**.**format**(**"%02X"**,** data**[**i**]));**  buf**.**append**(**" "**);**  **}**  buf**.**append**(**']'**);**  **return** buf**.**toString**();**  **}**  public String getTimeStr**(**byte**[]** time**)** **{**  **return** String**.**format**(**"%02d-%02d-%02d %02d:%02d:%02d"**,** **(**time**[**0**]),**  **(**time**[**1**]),** **(**time**[**2**]),** **(**time**[**3**]),** **(**time**[**4**]),** **(**time**[**5**]));**  **}**  **}** |

平台已经封装好几种数据持久化接口，在模块中可以根据dataOutType字段来判断数据输出的类型.dataOutType 有4种类型：0：无输出、1：kafka、2：rdbms、3：redis

|  |
| --- |
| **switch** **(this.**dataOutType**)** **{**  **case** 0**:**  **break;**  **case** 1**:**  **this.**storeInKafka**(**tuple**,** **this.**tbname**);**  **break;**  **case** 2**:**  saveInDB**(**tuple**);**  **break;**  **default:**  logger**.**error**(**" dataOutType error : {}"**,** **this.**dataOutType**);**  **break;**    **}** |

Tstar 平台提供了三种数据持久化接口，

1. 写入kafka

用法：只需要调用storeInKafka方法即可，this.storeInKafka(tuple, this.tbname);

1. 写入rdbms

用法：需要在模块配置中添加数据库的配置，配置项必须跟如下表格中保持一致。

|  |
| --- |
| <property>  <name>**business.database.driver**</name>  <value>**com.mysql.jdbc.Driver**</value>  </property>  <property>  <name>**business.database.url**</name>  <value>**jdbc:mysql://192.168.103.201:3306/metatest?rewriteBatchedStatements=true**</value>  </property>  <property>  <name>**business.database.username**</name>  <value>**temp**</value>  </property>  <property>  <name>**business.database.password**</name>  <value>**temp**</value>  </property>  <property>  <name>**business.database.pool.max.connections**</name>  <value>**50**</value>  </property>  <property>  <name>**business.database.pool.initial.connections**</name>  <value>**1**</value>  </property>  <property>  <name>**business.database.pool.incremental**</name>  <value>**5**</value>  </property> |

然后直接调用方法this.storeInDB(sql, params); 本方法接受两个参数，一个是sql，一个是list类型的参数列表。

|  |
| --- |
| String sql **=** "insert into "  **+** **this.**tbname  **+** " (terminalID,cmdID,cmdSerialNo,time,msgBody) values (?,?,?,?,?)"**;**  List**<**Object**>** params **=** **new** ArrayList**<>();**  params**.**add**(**tstarData**.**getTerminalID**());**  params**.**add**(**tstarData**.**getCmdID**());**  params**.**add**(**tstarData**.**getCmdSerialNo**());**  params**.**add**(**tstarData**.**getTime**());**  params**.**add**(new** String**(**tstarData**.**getMsgBody**()));**  **try** **{**  **this.**storeInDB**(**sql**,** params**);**  **}**  **catch** **(**Exception e**)** **{**  logger**.**error**(**"store TstarData into Rmdb error {} - {} - {}"**,**  tstarData**.**toString**(),**  e**.**getMessage**(),**  e**);**  **}** |

1. 写入redis

用法：需要在模块配置文件中添加redis的配置，配置项必须跟如下表格中保持一致。

|  |
| --- |
| <property>  <name>**redis.ip**</name>  <value>**192.168.103.202**</value>  </property>  <property>  <name>**redis.port**</name>  <value>**6379**</value>  </property>  <property>  <name>**redis.database**</name>  <value>**11**</value>  </property>  <property>  <name>**redis.read.timeout**</name>  <value>**5000**</value>  </property> |

Tstar平台提供获取jedis的方法，Jedis jedis = this.getJedis();用户拿到jedis对象后可以对redis操作。

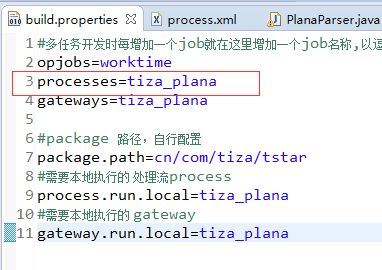
整个解析模块的完整代码见sdk中给出的示例：

### 4.2、配置

开发一个处理流（process）必须的配置有process.xml、{协议类型名称}.xml以及每个模块自己的配置文件。配置文件的配置只需按照SDK示例中给出配置文件中文注释修改即可。

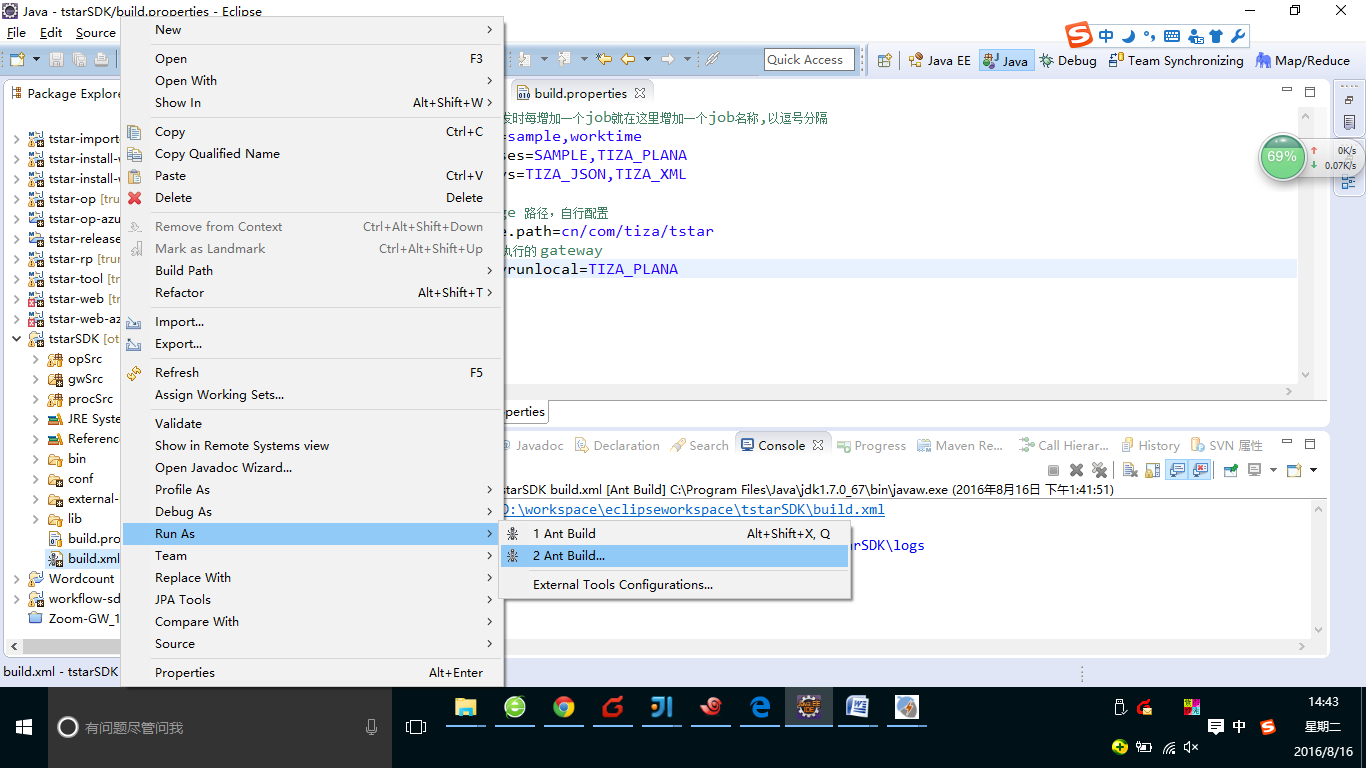
### 4.3、编译

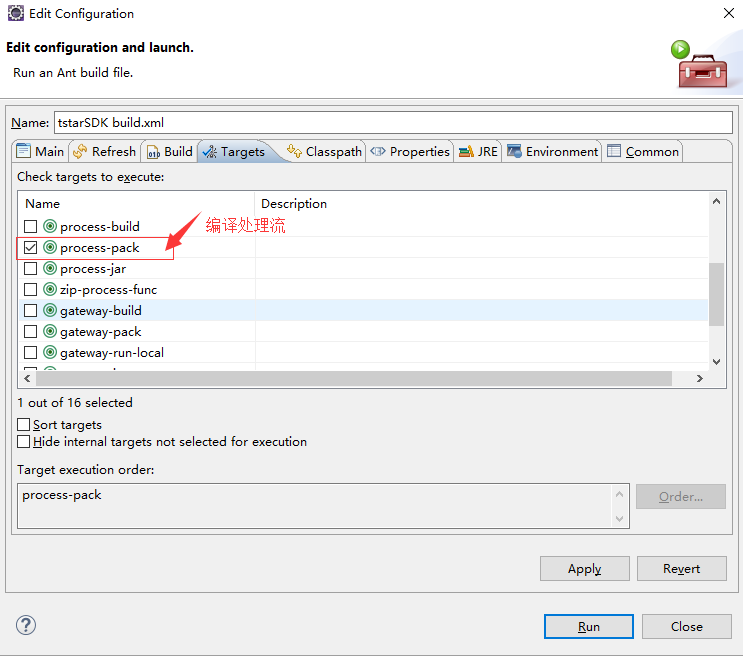
使用 Ant进行编译，用户将所要编译的处理流对应的协议类型名称配置到build.properties文件中processes这个变量中，多个处理流以逗号隔开即可同时编译多个处理流。运行Ant build target会将对应协议class文件、配置文件目录，第三方jar包目录自动打包生成zip包。以PLANA协议处理流编译为例，build.properties文件内容如下：



**build步骤如下**：

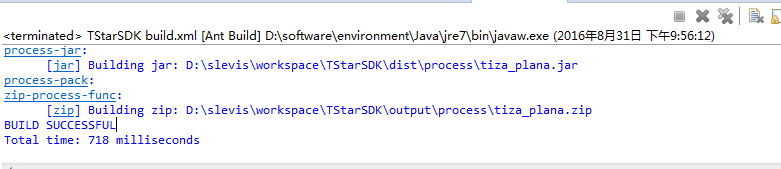
单击build.xml -> Run As -> Ant Build…。



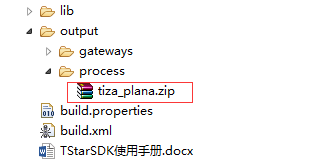


选择要编译的target，点击Run。

出现BUILD SUCCESSFUL 即表示编译成功。



编译打包的结果会存在根目录下面的output目录下。



### 4.4、本地测试

Process处理流是运行在TStar平台集群中的，需要打包上传到集群上，开发中测试不方便，为方便用户测试程序，TStar提供了本地执行驱动 ，来模拟处理流执行。

#### 4.4.1 测试前提

本地测试的前提是，TStar平台已安装好、用户已经在创建好终端类型、网关启动起、kafka中有数据。

#### 4.4.2 测试步骤

1. 修改build.propertites文件中process.run.local



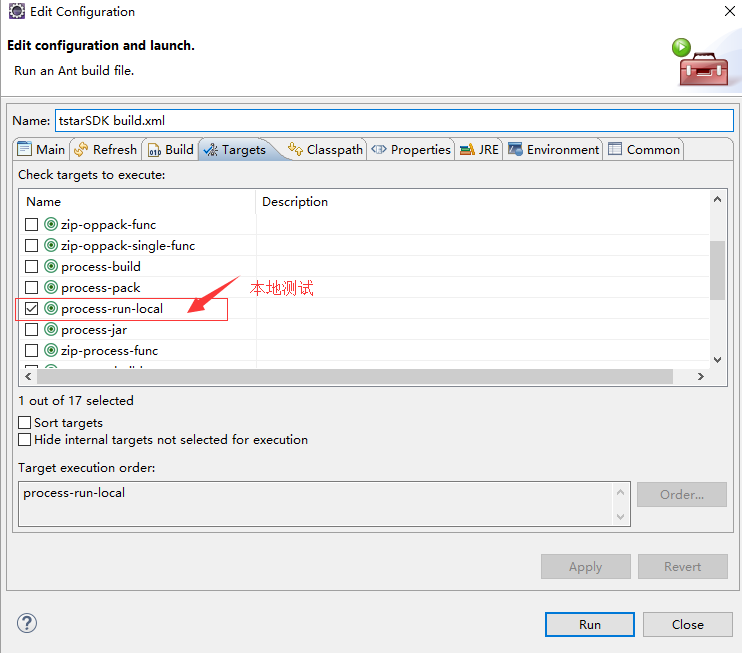
1. 修改conf/process/common 目录下的process-common.xml



注：由于kafka里的数据是实时产生的，测试阶段只需要测试部分数据即可，需要分析的条数配置test.record.count。

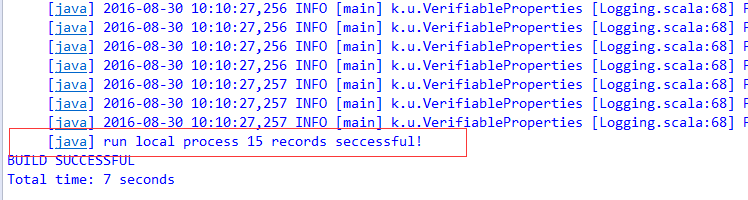
3、启动测试：

右键单击build.xml -> Run As -> Ant Build，勾选run target



1. 测试结果

当出现输出如下图所示即为测试成功。



### 4.5、运行

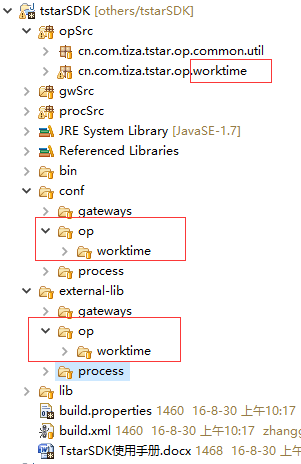
运行process详见TStar adminUI 使用手册实时计算部分。

## 五、离线计算（OP）

本模块主要是为了帮助客户迅速开发出离线计算任务，用户只需要关心计算逻辑即可不需要知道详细的处理逻辑，sdk提供了直接将计算结果输出到关系型数据库中。SDK打出的包可以支持在TStar和脱离TStar运行两种运行方式。

### 5.1、 开发

开发OP时需要把代码放在opSrc目录下，支持多个job同时开发，包名约定如下xxx.xxx.xxx.op.{jobname}，以PLANA协议的工作时间分析为例（jobname为worktime），如下图所示：



注：如果用户想使用自己的包路径命名，也是可以的，编译的时候只需要修改下build.properties文件中package.path变量。

在开发多任务的时候，在jobname包下开发各自的job，然后在conf下增加各自的配置文件。如果开发过程中需要依赖第三方的jar包，则需要在external-lib下相应的目录下增加。

下面以对plana协议解析后工况数据进行工作时间的分析为例来介绍SDK op的开发。

1、首先新建一个主类Main并继承tstar-op.jar中的cn.com.tiza.tstar.op.client.BaseJob基类，重写getJob方法，为方便用户在脱离TStar离线计算平台下也可以调试。在主类中增加一个main函数即可。

|  |
| --- |
| package cn**.**com**.**tiza**.**tstar**.**op**.**worktime**;**    public class Main **extends** BaseJob **{**  private static Logger logger **=** LoggerFactory**.**getLogger**(**Main**.**class**);**  /\*\*  \* 必须实现的函数，以供ngp离线计算平台调度使用  \*/  @Override  public Job getJob**()** **throws** Exception **{**    //conf.setBoolean("wordcount.case.sensitive", true);  job**.**setJobName**(**"Plana Worktime"**);**  job**.**setJarByClass**(**Main**.**class**);**    job**.**setMapperClass**(**WorkMapper**.**class**);**  job**.**setMapOutputKeyClass**(**WorkKey**.**class**);** //指定map输出key的类型  job**.**setMapOutputValueClass**(**Text**.**class**);** //指定map输出value的类型    job**.**setGroupingComparatorClass**(**GroupingComparator**.**class**);** //指定分组类型  job**.**setReducerClass**(**WorkReducer**.**class**);** //指定reduce类  job**.**setNumReduceTasks**(**3**);**  job**.**setInputFormatClass**(**TextInputFormat**.**class**);**  job**.**setOutputFormatClass**(**DBOutputFormat**.**class**);**    logger**.**info**(this.**data\_time**+**"----------"**);**  //数据库中列名数组  String**[]** fields **=** **{**"terminal\_id"**,** "first\_date"**,** "last\_date"**,** "worktime"**,** "datetime"**};**  //表名 worktime  DBOutputFormat**.**setOutput**(**job**,** "worktime"**,** fields**);**  **return** job**;**  **}**    /\*\*  \* 主函数  \* @param args  \* @throws Exception  \*/  public static void main**(**String**[]** args**)** **throws** Exception **{**  Job job **=** **new** Main**().**getJob**();**  FileInputFormat**.**addInputPath**(**job**,** **new** Path**(**args**[**0**]));**  System**.**exit**(**job**.**waitForCompletion**(true)** **?** 0 **:** 1**);**  **}**    **}** |

1. 计算出的结果写入到mysql中，需要创建一个跟mysql中表字段一一对应的实体类并需要实现DBWritable,只需要实现write方法即可。

|  |
| --- |
| package cn**.**com**.**tiza**.**tstar**.**op**.**worktime**;**  /\*\*  \* @author meta  \* 类数属性必须跟数据库中的字段对应  \*/  public class MyRecord **implements** DBWritable **{**    private String terminaID**;**  private Timestamp firstDate**;**  private Timestamp lastDate**;**  private int worktime**;**  // 20150826  private int date**;**  //……set get 方法省略  public MyRecord**(**String terminaID**,** Timestamp firstDate**,**  Timestamp lastDate**,** int worktime**,** int date**)** **{**  **super();**  **this.**terminaID **=** terminaID**;**  **this.**firstDate **=** firstDate**;**  **this.**lastDate **=** lastDate**;**  **this.**worktime **=** worktime**;**  **this.**date **=** date**;**  **}**    public MyRecord**()** **{**  **super();**  **}**  @Override  public void write**(**PreparedStatement statement**)** **throws** SQLException **{**  statement**.**setString**(**1**,this.**terminaID**);**  statement**.**setTimestamp**(**2**,** **this.**firstDate**);**  statement**.**setTimestamp**(**3**,** **this.**lastDate**);**  statement**.**setInt**(**4**,** **this.**worktime**);**  statement**.**setInt**(**5**,** **this.**date**);**  **}**  @Override  public void readFields**(**ResultSet resultSet**)** **throws** SQLException **{**    **}**    public static String changeCharset**(**String str**,**String newCharset**)**  **{**  **if(**str **!=** **null)**  **{**  **try** **{**  byte**[]** bs **=** str**.**getBytes**(**"ISO8859\_1"**);**  **return** **new** String**(**bs**,**newCharset**);**  **}** **catch** **(**UnsupportedEncodingException e**)** **{**  e**.**printStackTrace**();**  **}**  **}**  **return** **null;**  **}**  @Override  public String toString**()** **{**  **return** "MyRecord [terminaID=" **+** terminaID **+** ", firstDate=" **+** firstDate  **+** ", lastDate=" **+** lastDate **+** ", worktime=" **+** worktime  **+** ", date=" **+** date **+** "]"**;**  **}**  **}** |

1. 按正常Mapreduce编码流程编写Mapper类和Reduce类

Map 类

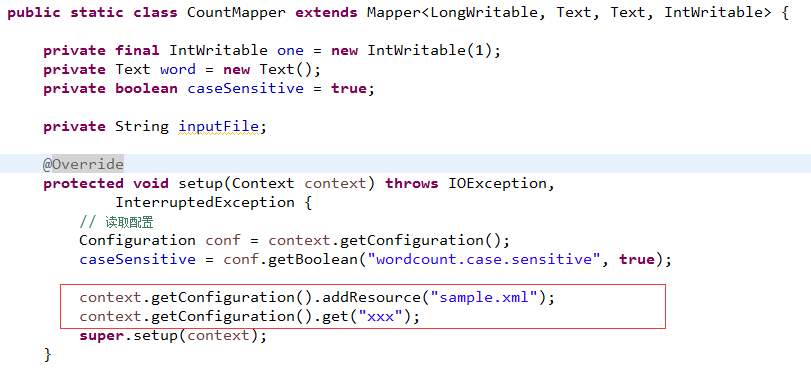
|  |
| --- |
| package cn**.**com**.**tiza**.**tstar**.**op**.**worktime**;**  public class WorkMapper **extends** Mapper**<**LongWritable**,** Text**,** WorkKey**,** Text**>** **{**  private static Logger logger **=** LoggerFactory**.**getLogger**(**WorkMapper**.**class**);**    static DateFormat df **=** **new** SimpleDateFormat**(**"yyyy-MM-dd HH:mm:ss"**);**    private final static String separator **=** "\t"**;**    @Override  protected void map**(**LongWritable key**,** Text value**,**  Mapper**<**LongWritable**,** Text**,** WorkKey**,** Text**>.**Context context**)**  **throws** IOException**,** InterruptedException **{**    **if** **(!**value**.**toString**().**equals**(**""**))** **{**  // 分割单行数据  //logger.info(value.toString());  String**[]** arr **=** value**.**toString**().**split**(**separator**);**    **if** **(**0x0200 **==** Integer**.**parseInt**(**arr**[**1**]))** **{**  String terminalID **=** arr**[**0**];**  long timepoint **=** 0**;**  **try** **{**  timepoint **=** df**.**parse**(**"20" **+** arr**[**8**]).**getTime**();**  **}** **catch** **(**ParseException e**)** **{**  logger**.**error**(**"parse time error:{}"**,** e**);**  **}**    WorkKey wk **=** **new** WorkKey**(**terminalID**,** timepoint**);**  Text workValue **=** **new** Text**();**  String totaltime **=** arr**[**12**];**    // logger.info("{}, {}, {}", terminalID, arr[8], totaltime);  workValue**.**set**(**timepoint **+** separator **+** totaltime**);**  context**.**write**(**wk**,** workValue**);**    **}**  **}**  **}**    **}** |

Reduce 类

|  |
| --- |
| package cn**.**com**.**tiza**.**tstar**.**op**.**worktime**;**  /\*\*  \* @author meta  \*  \*/  public class WorkReducer **extends** Reducer**<**WorkKey**,** Text**,** MyRecord**,** Text**>** **{**  private static Logger logger **=** LoggerFactory**.**getLogger**(**WorkReducer**.**class**);**  static DateFormat df **=** **new** SimpleDateFormat**(**"yyyy-MM-dd HH:mm:ss"**);**  SimpleDateFormat sdf **=** **new** SimpleDateFormat**(**"yyyyMMdd"**);**  private final static String separator **=** "\t"**;**  private final static String WROKING\_TIME\_LIMIT **=** "working.time.limit"**;**  private final static String WORKING\_TIME\_INTERVAL **=** "working.time.interval"**;**  private int interval **=** 5**;**  private Configuration conf **=** **null;**  private Connection conn**;**  private int daylimit **=** 0**;**    @Override  protected void setup**(**Reducer**<**WorkKey**,** Text**,** MyRecord**,** Text**>.**Context context**)**  **throws** IOException**,** InterruptedException **{**  conf **=** context**.**getConfiguration**();**  conf**.**addResource**(**"worktime.xml"**);**  conf**.**addResource**(**"op-core.xml"**);**  daylimit **=** Integer**.**parseInt**(**conf**.**get**(**WROKING\_TIME\_LIMIT**));**  interval **=** Integer**.**parseInt**(**conf**.**get**(**WORKING\_TIME\_INTERVAL**));**    **try** **{**  conn **=** DBUtil**.**getConnection**(**conf**);**  **}**  **catch** **(**SQLException e**)** **{**  logger**.**error**(**"get Connection fail! {}{}" **+** e**.**getMessage**(),** e**);**  **}**    **}**    @Override  protected void cleanup**(**  Reducer**<**WorkKey**,** Text**,** MyRecord**,** Text**>.**Context context**)**  **throws** IOException**,** InterruptedException **{**  DBUtil**.**close**(**conn**);**  **}**  @Override  protected void reduce**(**WorkKey wk**,** Iterable**<**Text**>** values**,**  Reducer**<**WorkKey**,** Text**,** MyRecord**,** Text**>.**Context context**)**  **throws** IOException**,** InterruptedException **{**  Iterator**<**Text**>** iterator **=** values**.**iterator**();**  List**<**String**>** dataList **=** **new** ArrayList**<**String**>();**    **while** **(**iterator**.**hasNext**())** **{**  String st **=** iterator**.**next**().**toString**();**  dataList**.**add**(**st**);**  **}**  int sum **=** 0**;**  **for** **(**int i **=** 0**;** i **<** dataList**.**size**()** **-** 1**;** i**++)** **{**  String**[]** v1 **=** dataList**.**get**(**i**).**split**(**separator**);**  String**[]** v2 **=** dataList**.**get**(**i **+** 1**).**split**(**separator**);**  //累计工作时间差  double tempTime **=** Double**.**parseDouble**(**v2**[**1**])** **-** Double**.**parseDouble**(**v1**[**1**]);**  // logger.info("temptime:{}", tempTime);  **if** **(**tempTime **>** 0 **&&** tempTime **<=** interval**)** **{**  sum **+=** tempTime**;**  **}**    **}**    // 第一个时间点  String firstInfo **=** dataList**.**get**(**0**).**split**(**separator**)[**0**];**  // 最后一个时间点  String lastInfo **=** dataList**.**get**(**dataList**.**size**()** **-** 1**).**split**(**separator**)[**0**];**    Timestamp firstData **=** **new** Timestamp**(**Long**.**parseLong**(**firstInfo**));**  Timestamp lastData **=** **new** Timestamp**(**Long**.**parseLong**(**lastInfo**));**    Date date **=** **new** Date**(**Long**.**parseLong**(**lastInfo**));**  int datetime **=** Integer**.**parseInt**(**sdf**.**format**(**date**));**    MyRecord record **=** **new** MyRecord**(**wk**.**getTerminalID**(),** firstData**,** lastData**,** sum**,** datetime**);**  logger**.**info**(**record**.**toString**());**  context**.**write**(**record**,** **null);**    **}**    /\*\*  \* 删除当天的之前的计算结果  \*  \* @param cd\_vp\_uniqueno 终端唯一编号  \* @param timestam 时间戳  \* @return  \*/  @SuppressWarnings**(**"unchecked"**)**  public boolean deleteWorkTime**(**String cd\_vp\_uniqueno**,** long timestam**)** **{**  boolean res **=** **false;**  String sql **=** "delete from ALY\_WORKTIME\_"  **+** dateFormat**(**"yyyyMM"**,** **new** Date**(**timestam**))**  **+** " w WHERE w.cd\_vp\_uniqueno = ? AND w.wt\_day = ?"**;**  // logger.info("excute sql: "+sql);  List params **=** **new** ArrayList**();**  params**.**add**(**cd\_vp\_uniqueno**);**  params**.**add**(**dateFormat**(**"yyyyMMdd"**,** **new** Date**(**timestam**)));**  **try** **{**  res **=** DBUtil**.**updateByPreparedStatement**(**conn**,**sql**,** params**);**  **}**  **catch** **(**SQLException e**)** **{**  logger**.**error**(**"sql error {}{}"**,** e**.**getMessage**(),** e**);**  **}**  **return** res**;**  **}**    /\*\*  \* 获取前一次工作累计时间 如果一次查询上一次的工作时间即昨天的不存在，就往往前推一天，最多往前推10天  \*/  @SuppressWarnings**({** "unchecked"**,** "rawtypes" **})**  public Map**<**String**,**Object**>** getfrontWorktime**(**String cd\_vp\_uniqueno**,** long timestam**)** **{**  long daytime **=** 24 **\*** 60 **\*** 60 **\*** 1000**;**  long timelong **=** timestam **-** daytime**;**  // 查询前一天最后一个时间点的累积工作时间  String sql **=** "SELECT w.wt\_totalworktime,w.wt\_lasttime FROM ALY\_WORKTIME\_"  **+** dateFormat**(**"yyyyMM"**,** **new** Date**(**timelong**))**  **+** " w WHERE w.cd\_vp\_uniqueno = ? AND w.wt\_day = ?"**;**  List params **=** **new** ArrayList**();**  params**.**add**(**cd\_vp\_uniqueno**);**  params**.**add**(**dateFormat**(**"yyyyMMdd"**,** **new** Date**(**timelong**)));**  Map resMap **=** **null;**  **try** **{**  resMap **=** DBUtil**.**findSimpleResult**(**conn**,**sql**,** params**);**  **}**  **catch** **(**SQLException e**)** **{**  logger**.**error**(**"sql error: {}{}"**,** e**.**getMessage**(),** e**);**  **}**  int count **=** 0**;**  **while** **(**resMap**.**size**()** **<** 1**)** **{**  timelong **-=** daytime**;**  params**.**clear**();**  params**.**add**(**cd\_vp\_uniqueno**);**  params**.**add**(**dateFormat**(**"yyyyMMdd"**,** **new** Date**(**timelong**)));**  **try** **{**  resMap **=** DBUtil**.**findSimpleResult**(**conn**,**sql**,** params**);**  **}**  **catch** **(**SQLException e**)** **{**  logger**.**error**(**"sql error: {}{}"**,** e**.**getMessage**(),** e**);**  **}**  count**++;**  **if** **(**count **>** daylimit**)**  **break;**  **}**    /\*if (resMap.size() > 1) {  worktime = Double.parseDouble((String) resMap.get("wt\_totalworktime"));  }\*/    **return** resMap**;**  **}**    /\*\*  \* 将double类型保留2位有效小数  \*  \* @param value  \* @return  \*/  public double cutdouble**(**double value**)** **{**    **return** **((**double**)** Math**.**round**(**value **\*** 100**)** **/** 100**);**    **}**    /\*\*  \* 格式化时间  \*/  public static String dateFormat**(**String format**,** Date date**)** **{**  DateFormat df **=** **new** SimpleDateFormat**(**format**);**  **return** df**.**format**(**date**).**toString**();**  **}**    public static void main**(**String**[]** args**)** **{**  System**.**out**.**println**(new** WorkReducer**().**getfrontWorktime**(**"33423432233"**,**  132897978978988l**));**  **}**  **}** |

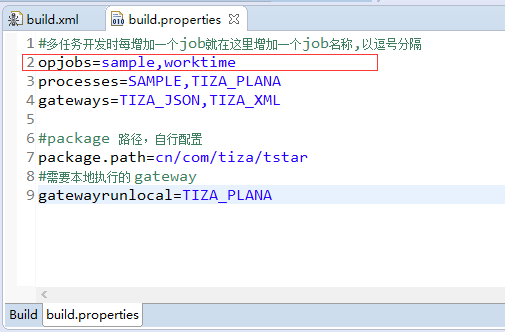
到目前为止整个MapReduce代码已经写完，可以跑在TStar离线计算平台之上。

如果想在MapReduce中使用用户自定义配置文件，如worktime.xml,如下图红框中。



### 5.2、 编译

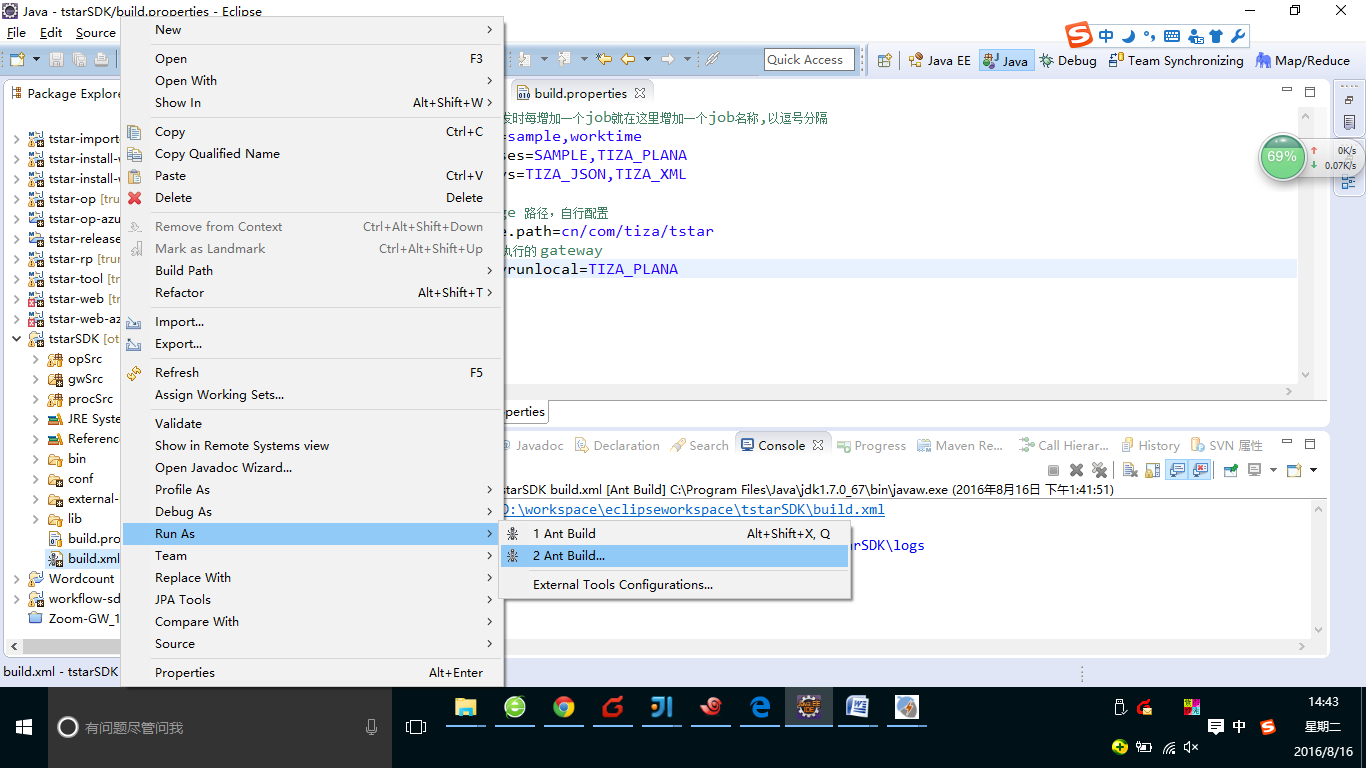
项目Build使用Apache Ant工具，用将所要编译的job的jobname配置到build。Properties文件中opjobs这个变量中，多个jobname以逗号隔开即可同时编译多个job，运行Ant build target会自动打入项目包

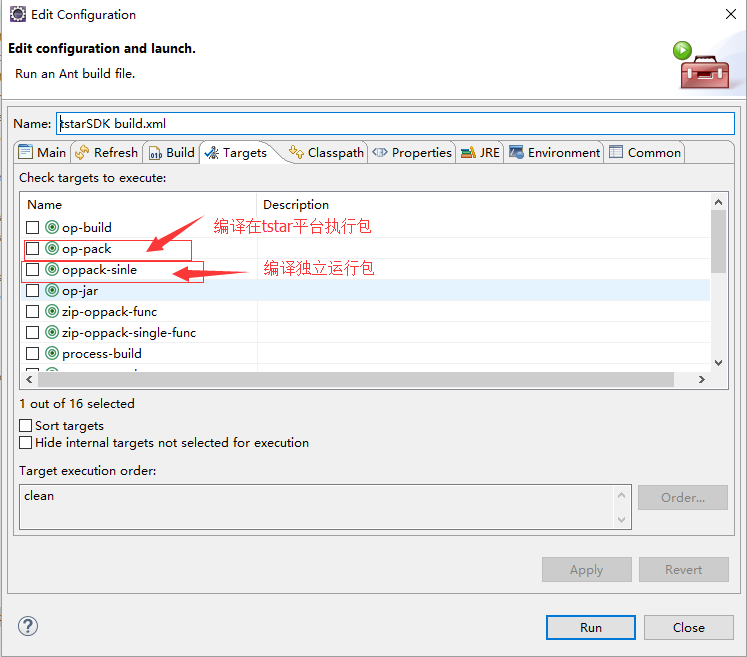


**build步骤如下**：

build分为两种:一、编译带有可执行脚本的安装包，可在没有部署tstar-op的环境上测试运行。二、编译仅可以部署在tstar-op上运行的安装包，

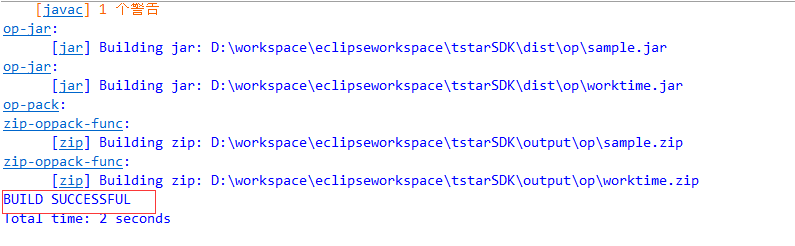
单击build.xml -> Run As -> Ant Build…。



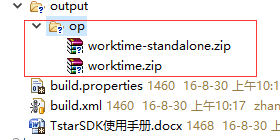


选择要编译的target，点击Run。

出现BUILD SUCCESSFUL 即表示编译成功。

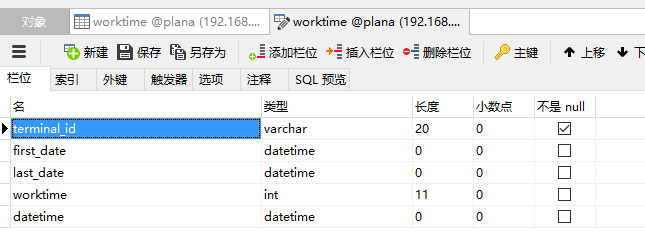


编译打包的结果会存在根目录下面的output目录下。



### 5.3、运行

运行提供两种方式，一种是依赖TStar离线计算平台，一种是独立运行。在运行之前需要手动创建分析结果需要入库的表，这个是表worktime。



#### 5.3.1、独立运行

##### 5.3.1.1 上传程序包

将编译出来的worktime-standalone.zip上传到安装了hadoop的集群上的任意一个节点的任意位置，解压缩，进入到worktime目录。

解压缩：unzip worktime-standalone.zip –d worktime

##### 5.3.1.2、配置运行参数

进入conf目录配置op-core.xml根据中文注释配置

配置数据输入：

在op-core.xml中配置数据根目录dataDir，如果数据在hdfs上的存储目录为/tstar/tiza\_plana\_trackdata/20160830/datawriter-1472547845420则只需配置到/tstar/tiza\_plana\_trackdata即可，如下图：

**Job配置**



op.perid是时间周期标记。如果配置成d，则表示按天执行，op.plan\_time表示每天什么时间开始执行。如果配置成w，则表示按周执行，op.plan\_time表示每周一什么时间开始执行，如果配置成m，则表示按月执行，op.plan\_time表示每月一号什么时间开始执行。

对数据库的配置：

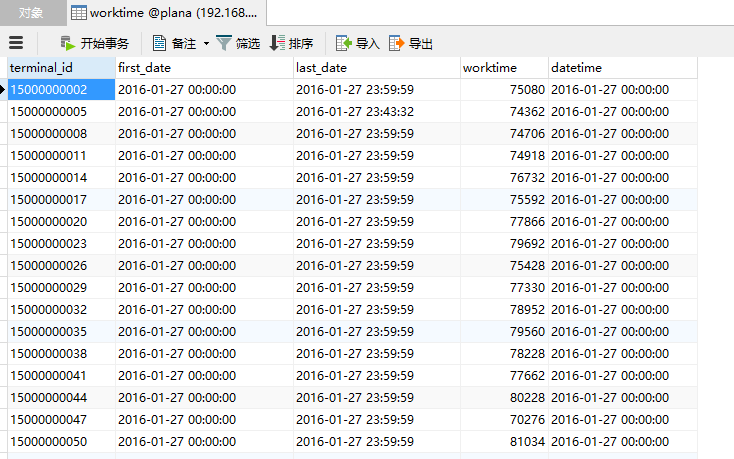


##### 5.3.1.3、运行Job

进入到bin目录，给目录中run.sh增加可执行权限，chmod u+x run.sh，修改脚本中的HADOOP\_HOME,直接执行/run.sh 数据目录。

/run.sh /tstar/tiza\_plana\_trackdata/20160830

查看mysql中执行结果。



直接运行job只提供用于job的测试之用，建议测试通过部署在tstar-op上管理job。

#### 5.3.2、基于TStar离线平台运行Job

基于TStar平台运行job，提供了两种方式，本文只介绍在shell环境下的操作，界面的操作详见adminUI使用手册。

##### 5.3.2.1、部署

将编译出来的{jobname}.zip上传到TStar平台的集群运维机上的/tstar-operation/op目录下，解压、将{jobname}.zip 删除，

解压缩：Unzip jobname.zip –d jobname

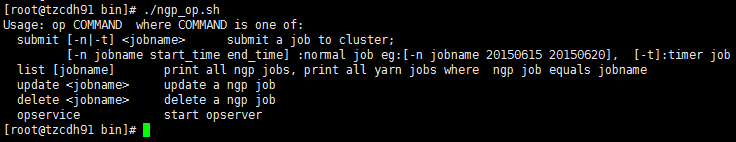
删除：rm –rf jobname.zip

##### 5.3.2.2、操作job

进入到/tstar-operation/bin目录下

**1）查看帮助**

/tstar-op.sh --help



**2）提交任务**

1、普通任务：

Nohup /tstar-op.sh submit -n jobname 20150615 20150620 &

查看日志

tail –f /tstar/tstar-service/logs/tstar-op.log

2、周期任务：

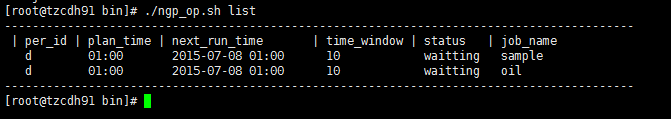
/tstar-op.sh submit -t jobname



**3）查看任务**

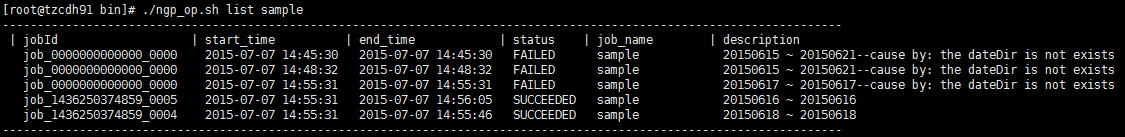
1、查看所有任务

/tstar-op.sh list



2、查看job的在集群运行情况

/tstar-op.sh list jobname



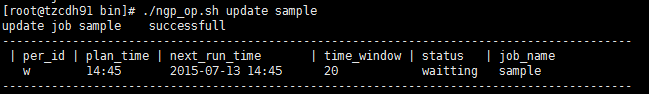
**4）删除周期任务**

/tstar-op.sh delete jobname



**5）更新周期任务**

/tstar-op.sh update jobname



## 附录

**模拟终端通信协议**

**1 数据类型**

|  |  |
| --- | --- |
| 数据类型 | 描述及要求 |
| BYTE | 无符号单字节整型(字节，8位) |
| WORD | 无符号双字节整型(字，16位) |
| DWORD | 无符号四字节整型(双字，32位) |
| BCD[n] | 8421码，n字节 |
| STRING | UTF8 编码，若无数据，置空 |

**2 消息组成**

* 1. **消息结构**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标识位 | 消息头 | 消息体 | 校验码 | 标识位 |

* 1. **标识位**

采用0x7e

* 1. **消息头**

|  |  |  |  |
| --- | --- | --- | --- |
| 起始字节 | 字段 | 数据类型 | 描述及要求 |
| 0 | 消息ID | WORD |  |
| 2 | 消息体长度 | WORD |  |
| 4 | 终端ID | BCD[6] | 终端ID长度为12位，不足前补0 |
| 10 | 消息流水号 | DWORD | 按发送顺序从0开始循环累加 |

* 1. **校验码**

校验码指从消息头开始，同后一个字节异或，直到校验码前一个字节，占用一个字节。