ETL操作简明

# 简述

- ETL是一个分布式任务调度平台。

- 分布式部署服务器间通过nfs做成集群磁盘共享，集群可以有多个。

- 分布式应用分主、从节点，由各服务器竞争获取，主节点掉线，从节点接替，主节点负责管理定时器，历史、异常任务入库，异常任务检测，主从节点都司获取集群或本机任务之责，从节点负责将主节点检测出的异常任务从本机上移除

- 调度的自然是任务，任务又称场景，场景由预设好的步骤组装而成，由场景配置的定时调度，步骤顺序执行，依赖zookeeper传递消息。举例：发现-下载-转换-入库，发现是第一步，扫描源端数据，产生文件队列，按照20（目前）个文件一个下载拆分，生成N个下载任务，分别下载，下载完产生转换任务，又分别转换，转换产生入库任务，分别入库。后续还可以配置一些不需要接收文件队列的步骤，比如kettle步骤、SQL执行步骤等。

- 任务的各步骤有执行状态：等待执行、正在运行、运行历史、运行异常、运行超时、进程僵死、执行超时、已忽略异常，运行异常的可重新运行

# 工程路径

当前目录./源码工程/etl3.0

# WEB端界面

http://ip:port/etl3.0

admin/1

http://132.228.68.41:8080/etl\_new/main.jsp

## 首页

首页可以发布些更新公告，显示当前主节点，以及清除缓存动作

## 资源配置

用于管理FTP/SFTP/DB资源，在任务管理中配置任务时使用。

配置资源时，分公共信息、FTP信息、数据库信息，配置FTP信息时数据库信息可不填，配置数据库信息时，FTP信息可不填

## 集群配置

集群即服务器集群，服务器集群会通过gfs设置公共访问目录，集群内部任务流转文件共享。

根目录即共享目录或共享目录下建立的数据目录，用于存放任务产生的文件

## 节点配置

节点即单台服务器。节点编码请以NODE\_IP格式填写，方便区分；FTP服务名称（或SFTP）是本机开放给其他服务器访问的方式，因为日志信息存放于各节点，界面日志查看需从各节点获取日志信息展示；日志目录即日志存放的位置；最大任务数即此节点上同时运行的最大任务数，通过观察服务器的资源限制变更此数字；是否获取集群任务，是因为有的服务器时外网服务器，而外网服务器资源又相对较少，这类服务器仅仅做下载动作而不做清洗、入库等其他动作，所以这类服务器只获取指定到本机的任务，不获取集群任务，当然这是一个举例。

## 正在运行

显示当前正在执行的任务，可通过步骤序列查看日志，可停止任务

## 等待运行

单次扫描有可能产生大量的下载任务，基于诸如此类情况，请在查询条件中限制集群或节点信息，限制节点只去该节点下的任务。等待任务可通过界面删除。

## 运行异常

可查询运行异常的任务，运行异常会带出异常日志，也可查看日志查看详情，异常任务可重新运行，忽略之后将不再可运行

## 运行历史

正常结束的任务，可查看日志，日志有保存周期，根据服务器空间会有所差异

## 任务管理

任务的创建、配置、启停、调测都在此菜单下。

先解释几个字段含义：

是否在用：只表示该任务当前是否在用，并不表示是否在运行

是否启动：当前是否在定时调度

最近执行时间：每次定时调度会产生一个任务，一个任务下有多个步骤，多个步骤前后会产生多个步骤任务，最近执行时间即最近任务的调度时间

运行状态：最近一次任务的运行状态

运行结果：最近一次任务的运行结果

周期：quartz表达式（秒 分 时 天 月 周）

任务描述：很重要，任务描述最好备注好该任务的基本信息以及一些不为人知的信息，比如，该任务来源、经过哪些阶段、入到哪、共享到哪、以及其他辅助性的脚本写在哪等，以方便维护跟踪

### 步骤编辑

#### 发现

发现（扫描）步骤，从源端ftp/sftp按照配置规则扫描出要处理的文件推送后续处理

字段简释：

源类型：追加与不追加，追加则表示需要断点续传，一般情况下请选择不追加，断点续传请配置转换步骤（因为下载直接入库，入库步骤只会入全文件，无法设置断点，断点续传经转换会根据断点切分文件）

压缩格式：如有压缩，请填写压缩格式，在下载中会自动解压出来以供后续解析，如不需要解析该文件，比如发现-下载-上传则无需配置压缩格式（即无压缩）。

源服务器：资源配置中配置的资源

根目录：以/结尾

子目录通配符：为防止目录动态设置，以/开头/结尾，如/\*/tmp/

开始时间：初次扫描时生效，从此时间点开始扫描，部分FTP/SFTP由于时差问题相差8小时，具体问题具体看，再次扫描时会根据上一次扫描到的最新时间点开始扫描

内存值：java运行给定的单个任务执行的内存值，如遇内存溢出异常可尝试调整内存大小。

运行方式：目前只支持进程

是否等待前置任务执行完成：是表示所有上一步骤任务执行完成才执行当前步骤任务，一般填否

#### SQL执行

此处sql不支持设置变量，但支持分号分割的多条sql语句。

#### 下载

下载目录已由程序设定好，配置此步骤名称以及上级关系即可。

#### 转换

父类：对应java中的注解类名称，@Service("CommonConvertExecuter")中的值，该类实现IConvertExecuter接口，重写方法doConvert，参数：行记录、文件名（包含全路径），基于行记录做解析即可，如果要基于整个文件做清洗，请添加接入新的步骤。

#### Oracle加载

预处理脚本、后处理脚本为文件入库前后的SQL执行，字段分隔符默认为#\_#，经过转换之后都会替换成#\_#，入库编码为AL32UTF8、ZHS16GBK这类编码，因可能涉及其他编码，所以暂时以文本方式提供配置

#### kettle步骤

kettle配置的xml文件或job文件，可通过此步骤执行

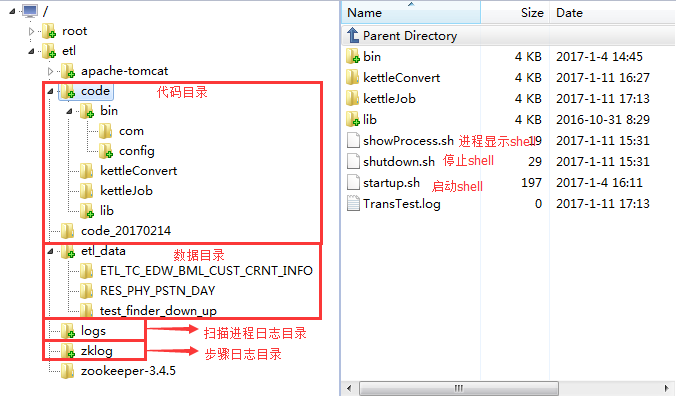
#### 文件上传

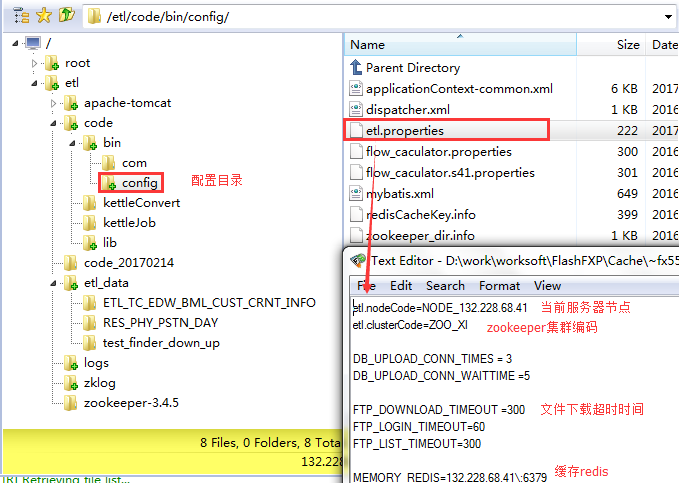
用于将前置步骤产生的文件队列长传至指定FTP/SFTP的指定目录下，用于文件共享等

# 分布式部署应用

## 目录结构

假设etl根目录为/etl





# 功能步骤接入

在**package** com.zbiti.etl.extend.executer;下，创建一个工厂类一个执行类，

如：工厂类：FileUpStepExecuterFactory,执行类：FileUpStepExecuter

工厂类：

@Service("fileUpStepExecuterFactory")

**public** **class** FileUpStepExecuterFactory **implements** ICommandExecuterFactory<Boolean>{

@Autowired(required=**true**)

IStepService stepService;

@Autowired(required=**true**)

IFileTransferService fileTransferService;

@Override

**public** ICommandExecuter<Boolean> createExecuter() {

FileUpStepExecuter fileUpStepExecuter=**new** FileUpStepExecuter();

fileUpStepExecuter.setStepService(stepService);

fileUpStepExecuter.setFileTransferService(fileTransferService);

**return** fileUpStepExecuter;

}

}

执行类：

**public** **class** FileUpStepExecuter **implements** ICommandExecuter<Boolean>{

**protected** **static** **final** Log *logger*=LogFactory.*getLog*(SqlStepExecuter.**class**);

@Override

**public** Boolean execute(ApplicationContext ctx,Node node,Step step,Command command,IFileDescQueue fileDescQueue) **throws** Exception {

*logger*.info("文件上传步骤-"+step.getStepName()+"["+step.getStepId()+"]开始执行!");

@SuppressWarnings("unchecked")

List<MorphDynaBean> fileQueue=(List<MorphDynaBean>) command.getParam().get("FILE\_QUEUE");

**if**(fileQueue==**null**||fileQueue.isEmpty()){

*logger*.info("队列为空，退出");

**return** **false**;

}

FileUpStep fileUpStep=stepService.getFileUpStepByStepId(step.getStepId());

IFileTransferClient fileTransferClient = **null**;

**try**{

fileTransferClient=fileTransferService.getClient(fileUpStep.getResourceName());

fileTransferClient.login();

**for**(MorphDynaBean bean:fileQueue){

FileDesc fileDesc=JSONUtil.*parse*(JSONUtil.*toJsonString*(bean), FileDesc.**class**);

String fromPath=fileDesc.getFileName();

String toPath=fileUpStep.getUpPath()+"/"+StringUtil.*getFileNameByDirectory*(fileDesc.getFileName());

fileTransferClient.upload(fromPath, toPath+".tmp");

fileTransferClient.rename(toPath+".tmp", toPath);

fileDescQueue.push(fileDesc);

}

}**catch** (Exception e) {

*logger*.error("文件上传出错：",e);

**throw** e;

}**finally**{

**if**(fileTransferClient!=**null**)

fileTransferClient.disconnectFtpClient();

}

**return** **true**;

}

IStepService stepService;

IFileTransferService fileTransferService;

**public** IStepService getStepService() {

**return** stepService;

}

**public** **void** setStepService(IStepService stepService) {

**this**.stepService = stepService;

}

**public** IFileTransferService getFileTransferService() {

**return** fileTransferService;

}

**public** **void** setFileTransferService(IFileTransferService fileTransferService) {

**this**.fileTransferService = fileTransferService;

}

}

然后发布到各服务器，在表select t.\*,t.rowid from ETL\_STEP\_TYPE t;中添加一条记录，其中FACTORY\_CLASS\_NAME为@Service("fileUpStepExecuterFactory")

中的值，然后开发对应的配置界面即可，配置界面可复制其他功能步骤的增删改查修改

# 转换（清洗）类编写

在**package** com.zbiti.etl.extend.executer.convert;包下创建一个业务目录，在创建目录下新建一个类实现IConvertExecuter接口，如：

@Service("CommonConvertExecuter")

**public** **class** CommonConvertExecuter **implements** IConvertExecuter{

@Override

**public** String doConvert(String data, String filePathName) **throws** Exception {

**return** data+"\n";

}

}

# 发布

发布只需将编译之后的类替换分布式部署应用code/bin目录下的类即可，也可全量打包编译后的包上传至各服务器code/bin目录下解压即可，非核心代码变更无需重启ETL