目录

[1 ETL工具使用文档 2](#_Toc132733329)

[1.1 环境准备 2](#_Toc132733330)

[1.1.1 运行环境： 3](#_Toc132733331)

[1.1.2 代码部署 4](#_Toc132733332)

[1.1.3 Python 3.6.10安装部署 12](#_Toc132733333)

[1.1.4 REST服务启动以及维护 15](#_Toc132733334)

[1.2 数据库模型 17](#_Toc132733335)

[1.2.1 py\_jobtype：支持的数据库类型 17](#_Toc132733336)

[1.2.2 py\_connections：数据库连接 17](#_Toc132733337)

[1.2.3 py\_jobs：Job信息配置 17](#_Toc132733338)

[1.2.4 py\_jobtype：Job类型维表 18](#_Toc132733339)

[1.2.5 py\_transforms：转换信息配置 18](#_Toc132733340)

[1.2.6 py\_trans\_type：transform类型维表 18](#_Toc132733341)

[1.2.7 py\_procedure\_params Oracle存储过程出入参 19](#_Toc132733342)

[1.2.8 py\_header\_cfg HTTP Header配置 19](#_Toc132733343)

[1.2.9 py\_variable\_cfg Javascript全局参数定义 19](#_Toc132733344)

[1.2.10 py\_data\_parser XML/JSON解析配置 19](#_Toc132733345)

[1.3 配置案例 20](#_Toc132733346)

[1.3.1 数据库间表同步 20](#_Toc132733347)

[1.3.2 存储过程回写 22](#_Toc132733348)

[1.3.1 直接回写表 25](#_Toc132733349)

[1.3.2 WebService回写 25](#_Toc132733350)

[1.3.3 HTTP回写 30](#_Toc132733351)

[1.3.4 动态SQL执行 30](#_Toc132733352)

# ETL工具使用文档

**更新记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **修改人** | **更新内容** | **对应版本号** |
| 2021-07-13 | 苏鹭彬 | 初版：当前支持XML配置 | zpd-pyetl-1.0.0 |
| 2021-07-16 | 苏鹭彬 | Milestone：数据资源库方式配置，数据库密码加密 | zpd-pyetl-2.0.0 |
| 2021-07-21 | 苏鹭彬 | 支持：PostgreSQL数据库 | zpd-pyetl-2.0.1 |
| 2021-07-27 | 苏鹭彬 | 支持：PHP后台调用时返回状态码 | zpd-pyetl-2.0.1 |
| 2021-07-27 | 苏鹭彬 | BUG：修复数据源表为空的报错 | zpd-pyetl-2.0.3 |
| 2021-08-12 | 苏鹭彬 | 支持：增加JavaScript调用功能，HTTP回写功能 | zpd-pyetl-2.0.4 |
| 2021-08-31 | 苏鹭彬 | Milestone：运行日志支持在数据库中查询 | zpd-pyetl-3.0.0 |
| 2021-08-31 | 苏鹭彬 | 支持：日志自动清理，支持Oracle两种连接方式 | zpd-pyetl-3.0.1 |
| 2021-09-17 | 苏鹭彬 | 优化：优化配置表数据的获取方式，提速0.2毫秒 | zpd-pyetl-3.0.3 |
| 2021-09-18 | 苏鹭彬 | 支持：新增支持sqlserver回写时调用存储 | zpd-pyetl-3.0.4 |
| 2021-09-30 | 苏鹭彬 | 支持：日志级别为可配置，配置文件：db.xml | zpd-pyetl-3.0.5 |
| 2021-12-10 | 苏鹭彬 | 支持：守护进程功能，以服务的方式启动的话，能在原来基础上再节约2-3s的时间，可以直接通过url方式运行数据同步流程。 | zpd-pyetl-3.0.6 |
| 2022-1-11 | 苏鹭彬 | 可配置失败继续执行，job不存在时记录ERROR日志而不停止。 | zpd-pyetl-3.0.7 |
| 2022-01-20 | 苏鹭彬 | Milestone：增加普通HTTP、WebService的调用功能，可通过配置的方式解析JSON、XML、实现变量存入数据库能力。 | zpd-pyetl-4.0.0 |
| 2022-03-03 | 苏鹭彬 | 修补部分bug | zpd-pyetl-4.0.1 |
| 2022-03-10 | 苏鹭彬 | 添加调用shell脚本功能 | zpd-pyetl-4.0.2 |
| 2022-05-19 | 苏鹭彬 | 增加适配小平台功能：减小循环提交数据量，增加mysql多SQL执行功能，增加更新行数信息，单条SQL执行状态回写功能。 | zpd-pyetl-4.0.4 |
| 2022-07-13 | 苏鹭彬 | 服务化功能完善，多线程日志记录完善 | zpd-pyetl-4.0.6 |
| 2022-08-18 | 苏鹭彬 | 同步流程添加自动建表功能 | zpd-pyetl-4.0.7 |
|  |  |  |  |

## 环境准备

ETL

1、适配多数库源：适用于各家医院异构数据库之间的数据同步

2、支持多种数据接口：适用于不同数据服务之间的数据同步，支持数据库、Web服务等

3、支持不同调度方式：定时任务、实时常驻服务等

4、支持不同处理方式：表对表，流处理、存储过程、shell调用、HTTP（WebService）、JavaScript、结构化数据处理：XML、JSON解析

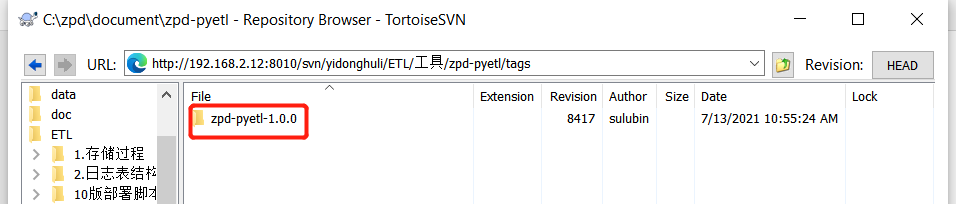
### 运行环境：

* Python 3.6.10
* Linux/Windows

#### 版本路径

打tag版本后的代码地址，部署时，请使用最新版本：

http://xxx:8010/svn/yidonghuli/ETL/工具/zpd-pyetl/tags

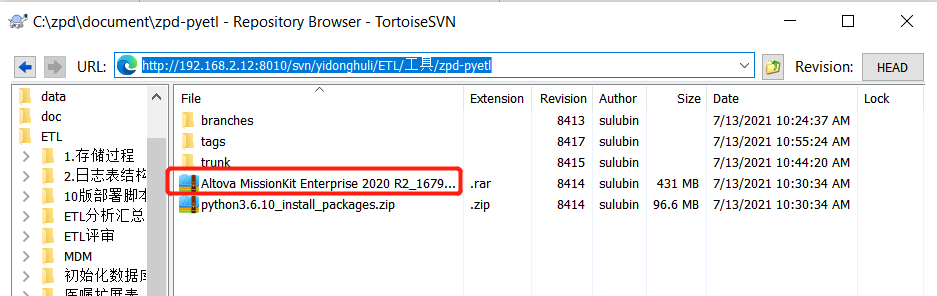


#### Python安装包和xml编辑软件

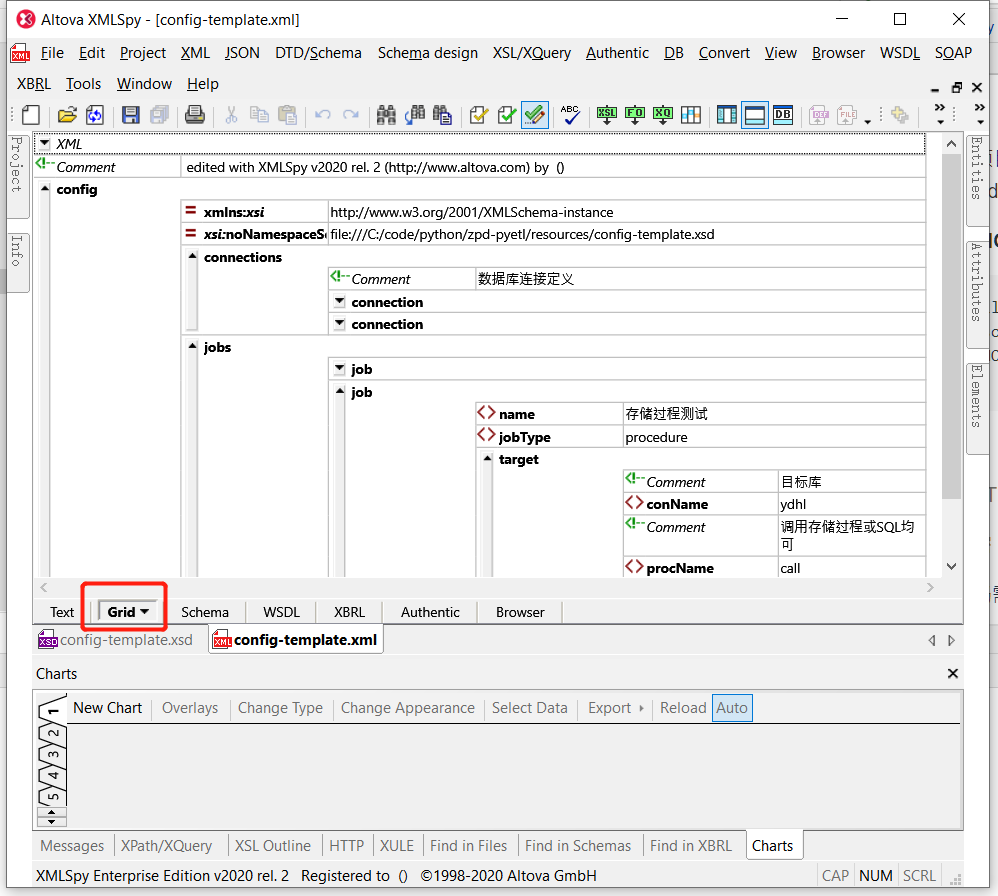
xml编辑软件安装 **——不再使用，直接阅读1.1.2**

（**不打算通过xml配置的，可以跳过这个步骤，直接通过数据库表的方式进行配置**）：

http://xxx/svn/xxx/ETL/工具/zpd-pyetl



拉取下来解压安装，并破解即可使用。通过该软件打开xml文件编辑，非常方便。



### 代码部署

#### 拷贝源码

将最新的zpd-pyetl项目整个文件夹拷贝到服务器目录下，如：/soft/etl/zpd-pyetl-1.0.0

#### 配置环境变量

配置ETL目录：ZPD\_PYETL\_HOME

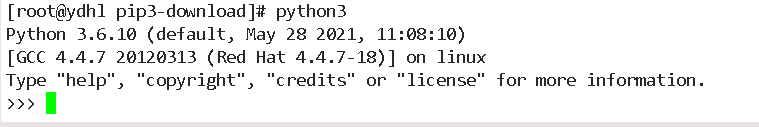
*vim ~/.bash\_profile  
ZPD\_PYETL\_HOME=/soft/etl/zpd-pyetl-1.0.0  
PATH=$ZPD\_PYETL\_HOME:$PATH:$HOME/bin  
export PATH*

配置完执行source

*source ~/.bash\_profile*

#### 环境检查

在终端执行python3，看是否弹出如下界面，是则表示运行环境已经部署，否则需要安装Python3运行环境。参考 [Python 3.6.10安装部署](#_Python_3.6.10安装部署)



#### ETL流程配置

我们配置ETL流程有两种方式，分别是文件和数据库表，具体的参考模板如下所示：

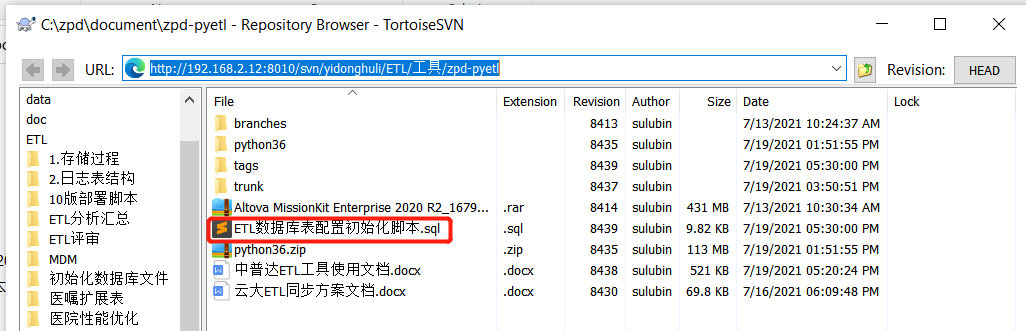
##### 配置文件的配置方式——不再使用

配置文件配置参考模板，根据文档中的释义进行配置：

${ZPD\_PYETL\_HOME}/resources/config-template.xml

##### 数据库表的配置方式

表结构与案例数据初始化脚本（建议直接在ydhl数据库执行建表脚本）：



需要配置resources/db.xml(资源库配置表所在的数据库)

*示例如下所示：*

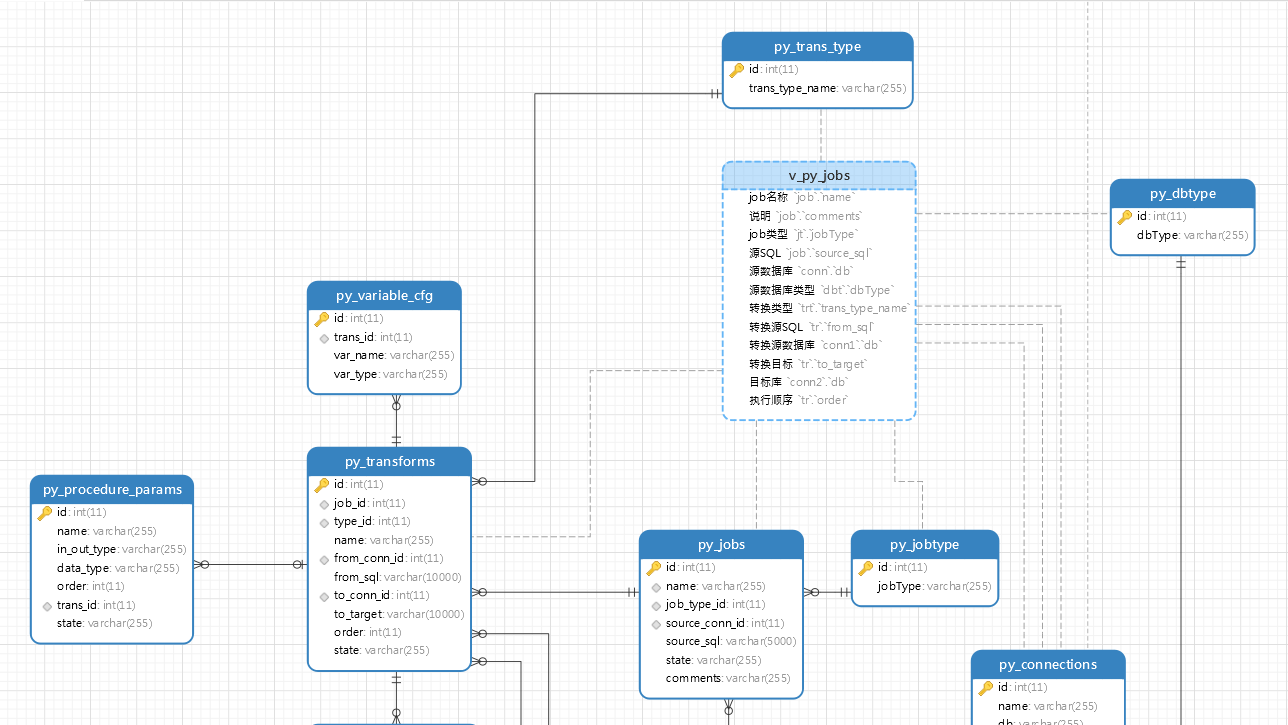
PS配置中的数据库密码需要进行加密，加密方式如下：

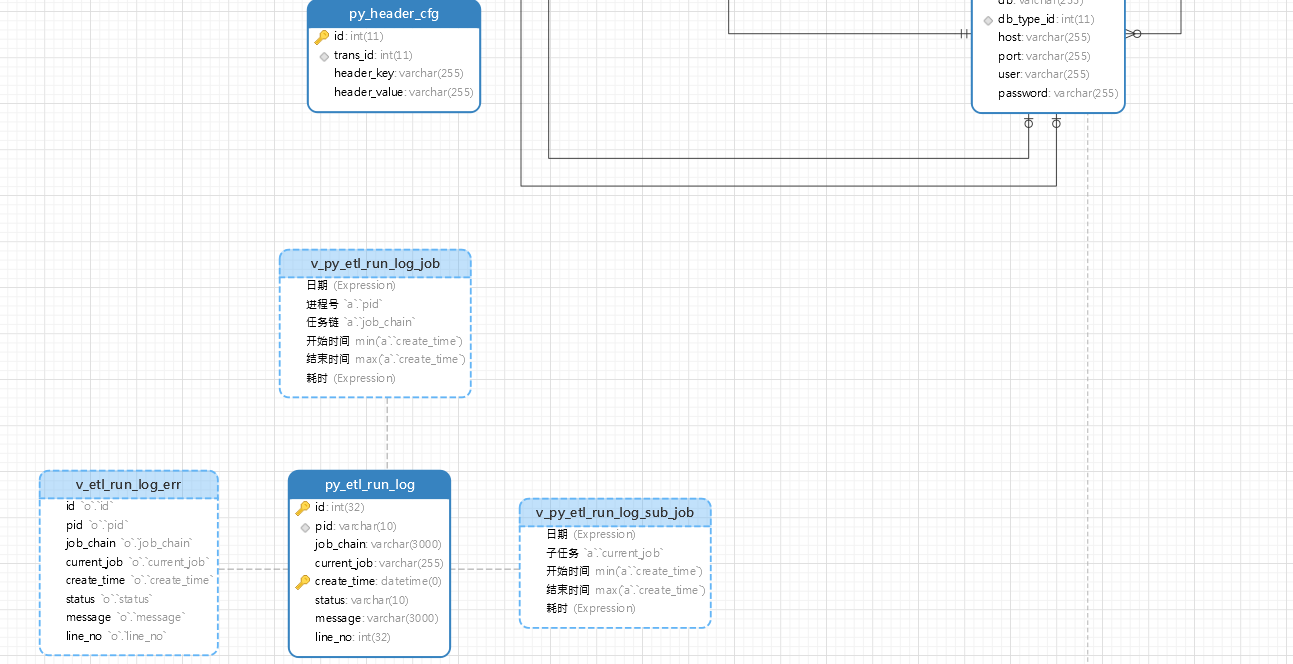
*python3 $ZPD\_PYETL\_HOME/encryption/base64\_encode.py*

*示例如下所示：*

*<?*xml version="1.0" encoding="UTF-8"*?>  
<!-- job数据库配置方式所在地址 -->*<config>  
 <connection>  
 *<!--链接名称，随意定义，最好是英文，或者与迁移的KETTLE一致-->* <name>ydhl</name>  
 *<!--数据库名称-->* <db>ydhl</db>  
 *<!--数据库类型，目前支持mysql-->* <dbType>mysql</dbType>  
 *<!--数据库地址-->* <host>192.168.41.136</host>  
 *<!--端口号-->* <port>3306</port>  
 *<!--用户名-->* <user>root</user>  
 *<!--密码-->* <password>TVRJek5EVTI=</password>  
 </connection>  
</config>

数据库逆向模型

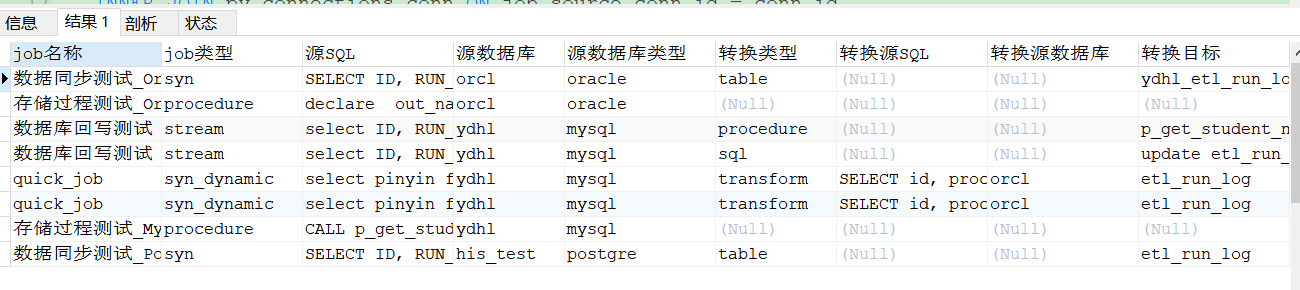




###### 参考JOB配置案例：

*-- 查询当前可执行的JOB*SELECT  
 job.`name` job名称,

job.comments 说明,  
 jt.jobType job类型,  
 job.source\_sql 源SQL,  
 conn.db 源数据库,  
 dbt.dbType 源数据库类型,  
 trt.trans\_type\_name 转换类型,  
 tr.from\_sql 转换源SQL,  
 conn1.db 转换源数据库,  
 tr.to\_target 转换目标,  
 conn2.db 目标库,  
 tr.`order` 执行顺序  
FROM  
 py\_jobs job  
 INNER JOIN py\_jobtype jt ON job.job\_type\_id = jt.id  
 INNER JOIN py\_connections conn ON job.source\_conn\_id = conn.id  
 INNER JOIN py\_dbtype dbt ON conn.db\_type\_id = dbt.id  
 LEFT JOIN py\_transforms tr ON job.id = tr.job\_id   
 AND tr.state = 'Y'  
 LEFT JOIN py\_connections conn1 ON tr.from\_conn\_id = conn1.id  
 LEFT JOIN py\_connections conn2 ON tr.to\_conn\_id = conn2.id  
 LEFT JOIN py\_trans\_type trt ON tr.type\_id = trt.id   
WHERE  
 job.state = 'Y'   
ORDER BY  
 job.id ASC,  
 tr.`order` ASC;



###### 存储过程参数配置案例

*-- 目前仅适用于回写流程中使用到（Oracle）存储过程时的参数配置*

*--查询当前存在配置存储过程参数的 transform*

SELECT

tr.`name` 转换名称,

tr.to\_target 目标过程,

conn2.db 目标数据库,

pm.`name` 参数名,

pm.in\_out\_type 出参入参,

pm.data\_type 数据类型

FROM py\_transforms tr

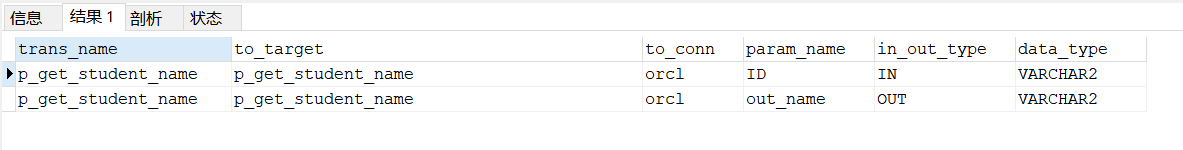
LEFT JOIN py\_connections conn1 ON tr.from\_conn\_id = conn1.id

LEFT JOIN py\_connections conn2 ON tr.to\_conn\_id = conn2.id

INNER JOIN py\_procedure\_params pm ON tr.id = pm.trans\_id and pm.state = 'Y'

where tr.state = 'Y'

order by pm.`order` asc;



###### JavaScript配置的变量

SELECT

tr.id,

pvc.var\_name,

pvc.var\_type

FROM

py\_transforms tr

INNER JOIN py\_trans\_type tp ON tr.type\_id = tp.id

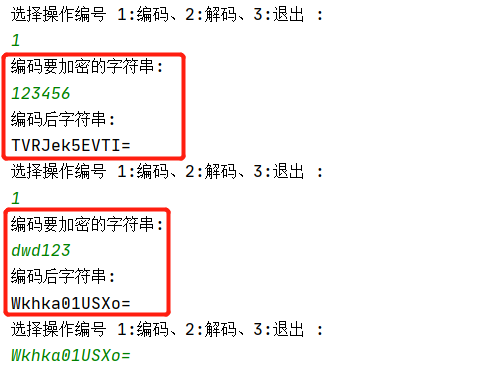
INNER JOIN py\_variable\_cfg pvc ON tr.id = pvc.trans\_id

WHERE

tp.trans\_type\_name = 'javascript'

PS配置中的数据库密码需要进行加密，加密方式如下：

*python3 $ZPD\_PYETL\_HOME/encryption/base64\_encode.py*

**加密后将密码串，如：TVRJek5EVTI= 配置到配置文件中，或者数据库表配置中**



#### 运行示例

**配置完resources目录下的配置文件即可运行如下语句执行ETL流程**

**shell脚本：**

vim /soft/etl/zpd\_pyetl\_files/quick.sh

*```shell script  
#!/bin/bash  
source ~/.bash\_profile  
cd $ZPD\_PYETL\_HOME****# 强烈推荐使用数据库资源库的配置方式，使用下面语句：job以逗号隔开***

***python3 main.py db jobname1,jobname2,jobname3***

*# xml文件配置方式使用下面语句：job以逗号隔开**——不再使用  
python3 main.py zhuyuan\_fuzhujiancha.xml jobname1,jobname2,jobname3*

*```*

***首次同步需要建表时可以添加如下参数：则同步过程中会自动在目标库中建表***

*python3 main.py db jobname1,jobname2,jobname3* ***–create=true***

##### crontab定时任务配置

\*/1 \* \* \* \* flock -xn /tmp/quick.lock -c 'timeout -s 9 180 /soft/etl/zpd\_pyetl\_files/quick.sh'

#### 数据库运行日志查看方式

数据库的配置方式，运行日志可以通过表查询到：

SELECT \*

FROM py\_etl\_run\_log a

ORDER BY a.id DESC;



#### 运行日志所在目录

cd ${ZPD\_PYETL\_HOME}/log/

如：

aaa.xml文件方式配置的，则日志文件名为aaa.log*——不再使用*

数据库方式配置的，直接在数据库表查看日志：py\_etl\_run\_log

#### 注意事项

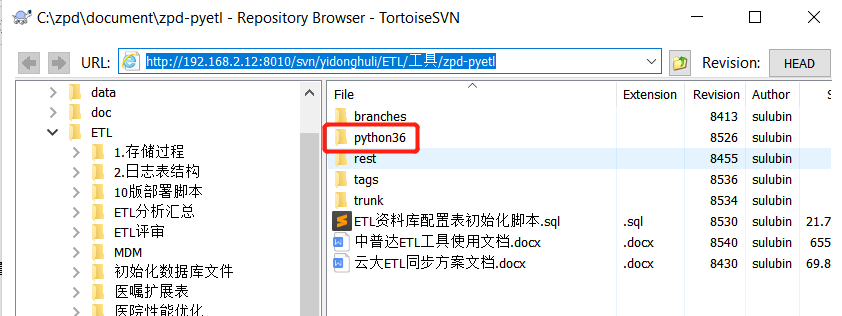
配置过程中from源字段必须严格和目标表的字段一致，from的字段必须在目标表中存在

### Python 3.6.10安装部署

#### python36安装包

安装包路径如下所示：

http://192.168.2.12:8010/svn/yidonghuli/ETL/工具/zpd-pyetl

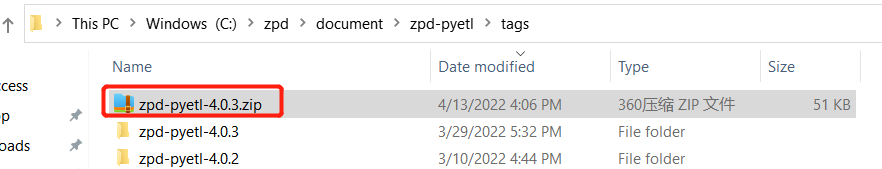


拉取下来，压缩后，上传到服务器。

#### etl代码：zpd-pyetl

同样，压缩最新版本的etl代码，上传到同一目录下：

版本地址：http://192.168.2.12:8010/svn/yidonghuli/ETL/工具/zpd-pyetl/tags

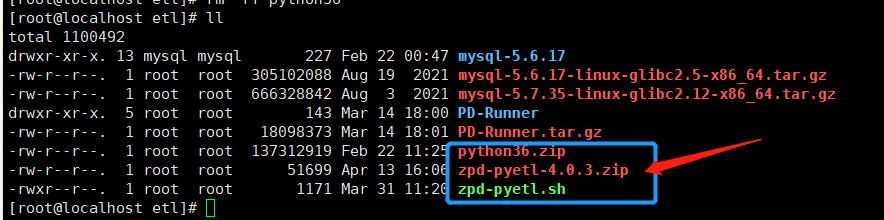


#### 一键安装脚本

拉取一键安装脚本，上传到同一目录下：

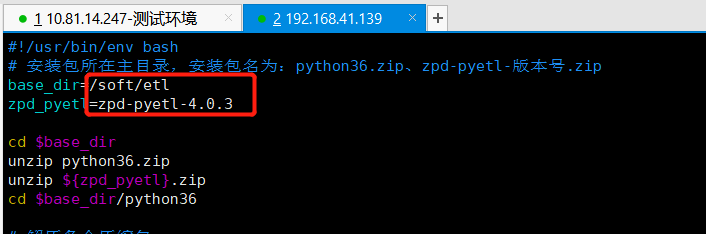
<http://192.168.2.12:8010/svn/yidonghuli/ETL/工具/zpd-pyetl/zpd-pyetl.sh>

最终需要如下三样：



#### 配置一键安装脚本

将这三个安装文件所在目录，以及etl代码版本配置上



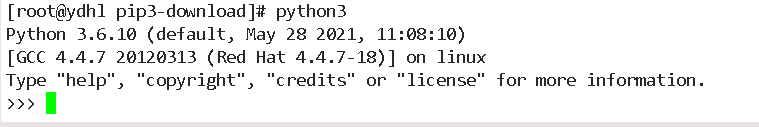
#### 执行安装

添加可执行权限，一键安装：

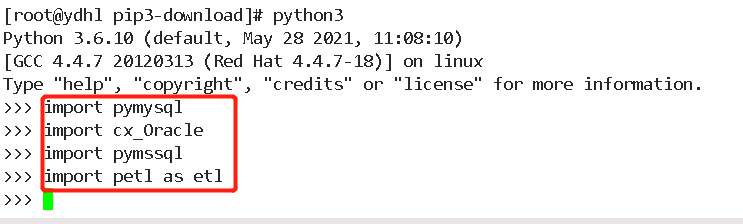
chmod u+x zpd-pyetl.sh

./zpd-pyetl.sh

安装完验证，出现如下信息，表示安装完成



输入如下命令，如果没有报错，表示引用包安装也没问题：



import pymysql

import cx\_Oracle

import pymssql

import petl as etl

…

至此部署结束。

### REST服务启动以及维护

#### 启动方式

执行脚本：sh $ZPD\_PYETL\_HOME/rest\_monitor.sh

#### 配置crontab守护进程

\*/1 \* \* \* \* /soft/etl/zpd-pyetl-4.1.0/rest\_monitor.sh >> /dev/etl\_rest\_daemon 2>&1

#### 编写etl监控函数

存储需在5s内正常返回结果,0表示任务运行正常，非0或者返回超时表示异常，会自动重启rest服务：

CREATE FUNCTION `f\_check\_rest\_status`

RETURNS varchar(10) CHARACTER SET utf8

BEGIN

DECLARE v\_time\_diff INT DEFAULT 0;

DECLARE v\_exit VARCHAR(10);

SELECT sum(cnt)

INTO v\_time\_diff

FROM inf\_fusedb.view\_trig\_detail\_info

WHERE msgdate = date\_format(curdate(), '%Y-%m-%d')

AND msg\_code = '0'

LIMIT 501;

IF v\_time\_diff >= 500 THEN

SET v\_exit = '1';

UPDATE `py\_etl\_monitor` a

SET a.mornitor\_value = v\_exit, a.mornitor\_msg = 'ETL服务存在异常'

WHERE a.mornitor\_type = 'rest\_status';

ELSE

SET v\_exit = '0';

UPDATE `py\_etl\_monitor` a

SET a.mornitor\_value = v\_exit, a.mornitor\_msg = 'ETL服务正常'

WHERE a.mornitor\_type = 'rest\_status';

END IF;

RETURN v\_exit;

END

#### 调用REST服务运行JOB

运行方式如下所示：

curl <http://127.0.0.1:8383/runjob/srv_md_department_info,srv_md_user_dept_rel_info>



正常返回任务结束，则表示JOB运行成功

## 数据库模型

### py\_jobtype：支持的数据库类型

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| jobType | varchar(255) | NO |  |  |  | 目前支持的job类型 |

### py\_connections：数据库连接

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  |  | 自增ID |
| name | varchar(255) | NO | UNI |  |  | 自定义数据库连接名称 |
| db | varchar(255) | NO |  |  |  | 数据库名称 |
| db\_type\_id | int(11) | NO | MUL |  |  | 数据库类型：来源：py\_dbtype |
| host | varchar(255) | NO |  |  |  | 数据库IP |
| port | varchar(255) | NO |  |  |  | 数据库端口 |
| user | varchar(255) | NO |  |  |  | 用户名 |
| password | varchar(255) | NO |  |  |  | 密码 |

### py\_jobs：Job信息配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| name | varchar(255) | NO | UNI |  |  | 自定义job名称 |
| job\_type\_id | int(11) | NO | MUL |  |  | job类型，关联：py\_jobtype |
| source\_conn\_id | int(11) | NO | MUL |  |  | 来源数据库 |
| source\_sql | varchar(5000) | NO |  |  |  | 取数SQL，如果是存储过程则写调用语句call pro( )，Oracle则填写调用语句块，包含BEGIN...END |
| state | varchar(255) | NO |  | Y |  | 状态，默认Y，N则为不在用 |
| comments | varchar(255) | YES |  |  |  | 备注，job说明 |
| is\_fail\_continue | varchar(2) | YES |  | N |  | 失败是否继续执行，默认为N，不执行 |

### py\_jobtype：Job类型维表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra | Comment |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| jobType | varchar(255) | NO |  |  |  | 目前支持的job类型 |

### py\_transforms：转换信息配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| job\_id | int(11) | NO | MUL |  |  | job ID |
| type\_id | int(11) | NO | MUL |  |  | transform类型，关联:py\_trans\_type |
| name | varchar(255) | YES |  |  |  | transform名称 |
| from\_conn\_id | int(11) | YES | MUL |  |  | （目前仅动态SQL取数时需要配置）来源数据库，关联py\_connections |
| from\_sql | varchar(10000) | YES |  |  |  | （目前仅动态SQL取数时需要配置）取数SQL |
| to\_conn\_id | int(11) | YES | MUL |  |  | 目标库 |
| to\_target | varchar(10000) | YES |  |  |  | 目标表，或者目标执行sql |
| order | int(11) | NO |  |  |  | 执行顺序 |
| state | varchar(255) | YES |  | Y |  | 状态，默认Y，N则为不在用 |

### py\_trans\_type：transform类型维表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| trans\_type\_name | varchar(255) | NO |  |  |  | transform支持的类型名称 |

### py\_procedure\_params Oracle存储过程出入参

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| name | varchar(255) | NO |  |  |  | 存储过程参数名称 |
| in\_out\_type | enum('IN','OUT') | YES |  |  |  | 4.0.5版本以前，字符串需大写 |
| data\_type | varchar(255) | NO |  |  |  | 参数类型，4.0.5版本以前，字符串需大写 |
| order | int(11) | NO |  |  |  | 参数顺序 |
| trans\_id | int(11) | YES | MUL |  |  | 关联的transform ID |
| state | varchar(255) | YES |  | Y |  | 状态，默认Y，N则为不在用 |

### py\_header\_cfg HTTP Header配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增主键 |
| trans\_id | int(11) | NO | MUL |  |  | 对应的转换 |
| header\_key | varchar(255) | NO |  |  |  | key |
| header\_value | varchar(255) | NO |  |  |  | value |

### py\_variable\_cfg Javascript全局参数定义

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增主键 |
| trans\_id | int(11) | NO | MUL |  |  | 对应的转换 |
| var\_name | varchar(255) | NO |  |  |  | 变量名称 |
| var\_type | varchar(255) | NO |  |  |  | 变量类型 |

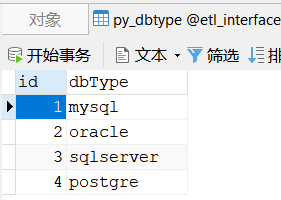
### py\_data\_parser XML/JSON解析配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** | **Comment** |
| id | int(11) | NO | PRI |  | auto\_increment | 自增ID |
| trans\_id | int(11) | YES | MUL |  |  | 关联的transform ID |
| match\_path | varchar(2000) | YES |  |  |  | 匹配路径 |
| match\_rule | varchar(255) | NO |  | common |  | 匹配规则：common |
| map\_field | varchar(255) | YES |  |  |  | 映射的字段名 |
| comment | varchar(255) | NO |  |  |  | 参数类型 |
| order | int(11) | NO |  | 0 |  | 参数顺序 |
| state | varchar(255) | YES |  | Y |  | 状态，默认Y，N则为不在用 |

## 配置案例

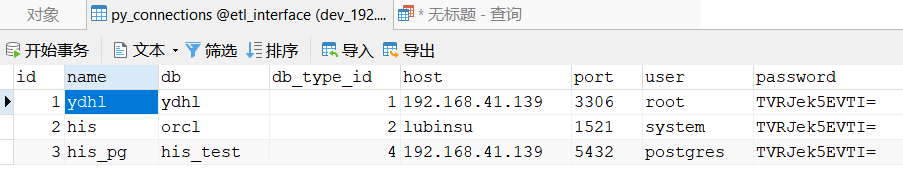
### 数据库间表同步

目前zpd-pyetl支持的数据库有：mysql、oracle、sqlserver、postgre，（其他的如果有需求可以再加）



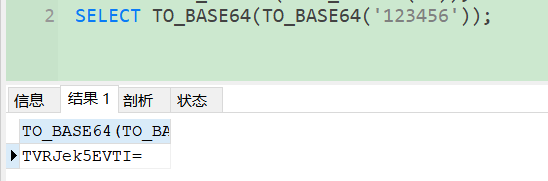
#### 新建数据库连接

如图所示配置py\_connections表：



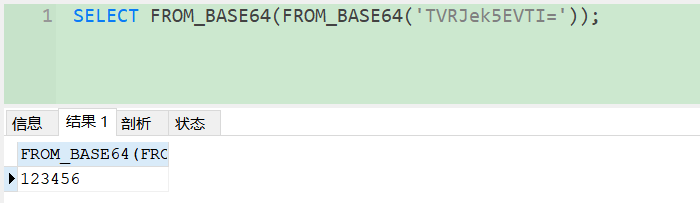
其中密码需要进行加密，加密方式如下：

SELECT TO\_BASE64(TO\_BASE64(**'123456'**));



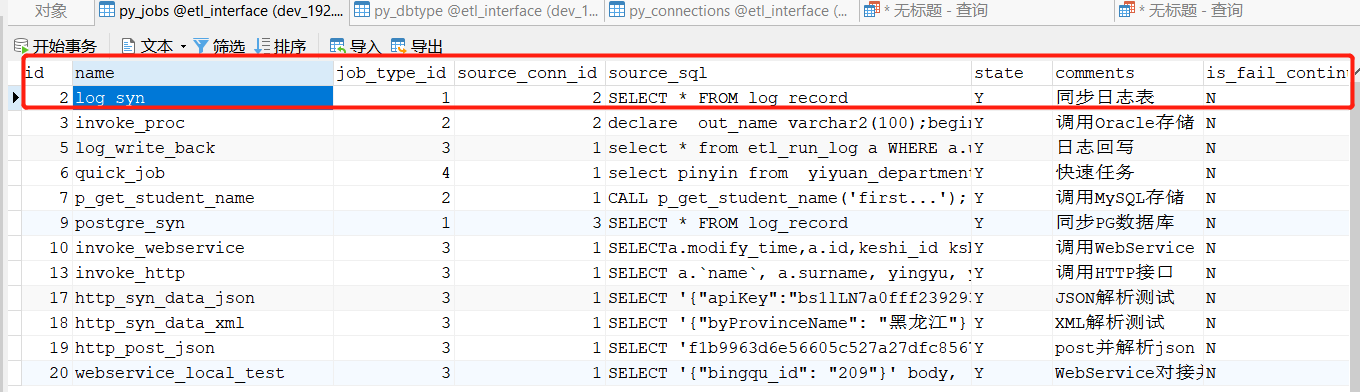
将编码后的字符串配置到password字段即可，如果需要解码则：

SELECT FROM\_BASE64(FROM\_BASE64('**TVRJek5EVTI=**'));



#### 配置源数据

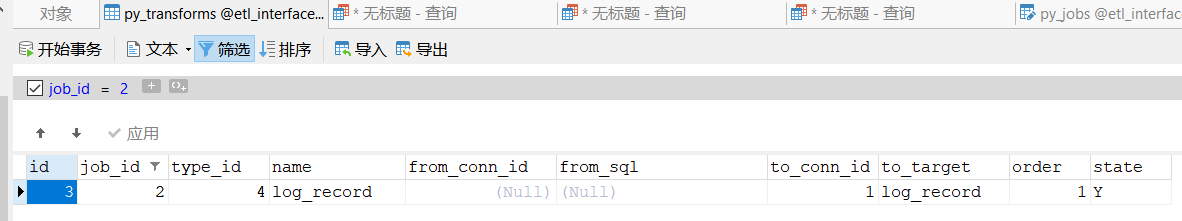
INSERT INTO `py\_jobs` (`id`, `name`, `job\_type\_id`, `source\_conn\_id`, `source\_sql` , `state`, `comments`, `is\_fail\_continue`) VALUES (2, 'log\_syn', 1, 2, 'SELECT \* FROM log\_record' , 'Y', NULL, 'N');



#### 配置目标表

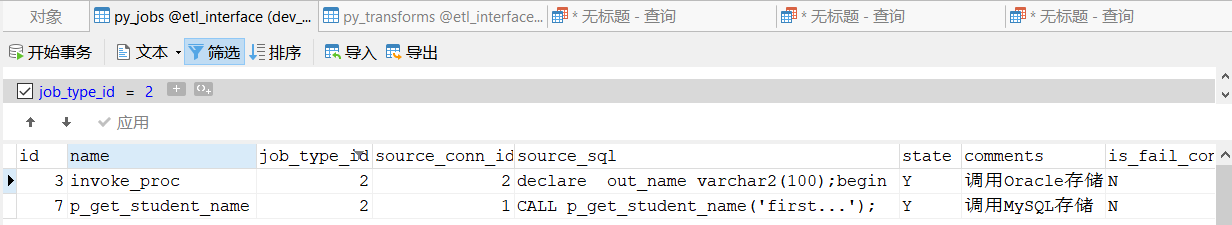
将目标表配置到py\_transforms中

job\_id = 2，type\_id = 4



#### 调用存储过程

仅仅需要配置py\_jobs 表一条记录即可

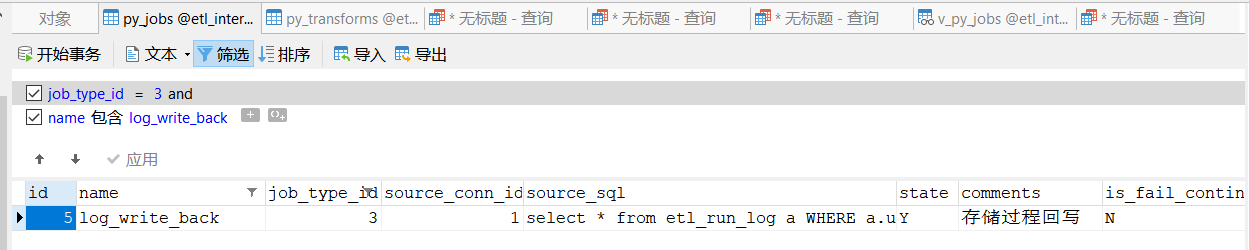


### 存储过程回写

根据py\_jobs中的查询结果，再调用transform中的存储，并将执行结果返回到原表

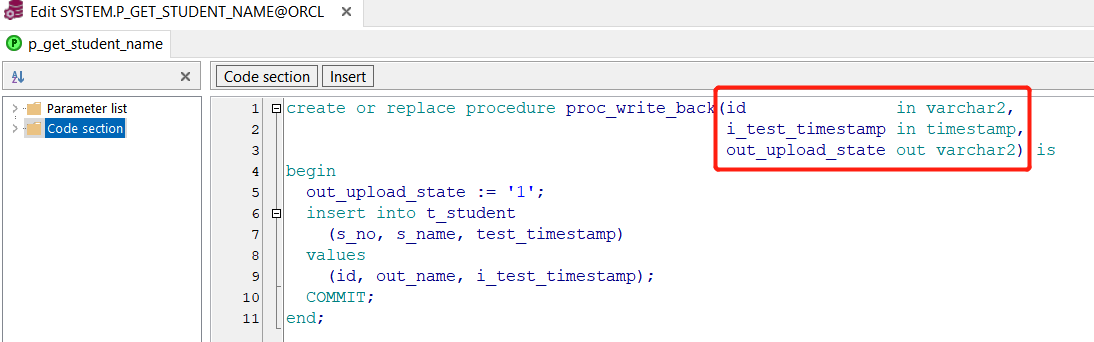
#### 配置抽取的SQL

字段名即为入参名字,job\_type\_id配置为：3：stream

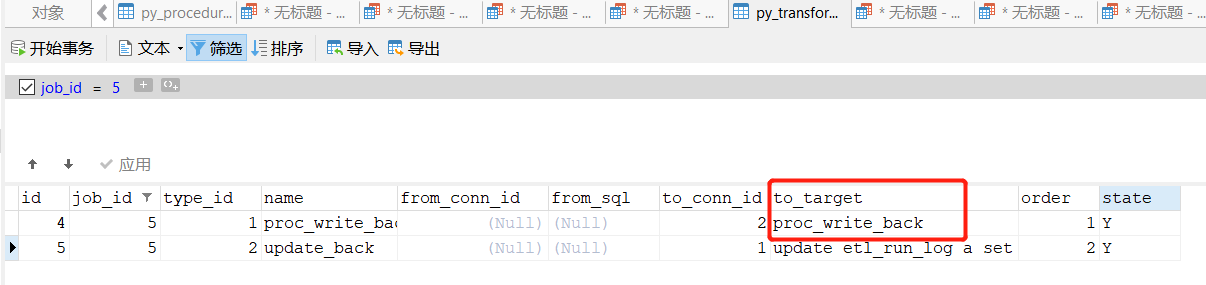


#### 配置目标存储过程（Oracle）

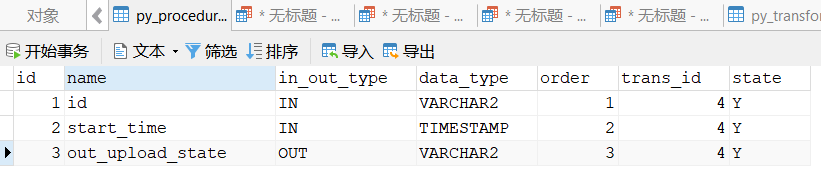
示例存储



在py\_transforms中配置存储名称，type\_id置为（1:procedure）：



调用Oracle存储过程需要配置py\_procedure\_params出入参表，其他类型的数据库不需要。



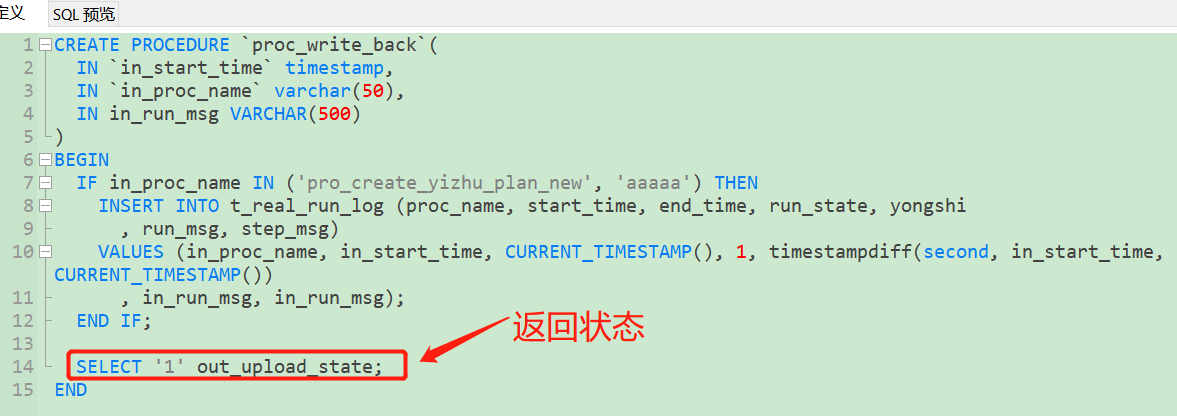
#### 配置目标存储过程（MySQL、SQL SERVER）

MySQL、SQL SERVER直接在字段to\_target配置即可，不需要配置py\_procedure\_params，如下所示，引用的参数名字来自源SQL的字段：

##### MySQL

call proc\_write\_back('{start\_time}', 'aaaaa', '{run\_msg}')

如果需要返回值，可以在存储中添加SELECT语句，如下所示：



##### SQL SERVER

将如下调用存储的语句，填入to\_target字段即可：

BEGIN

DECLARE @upload\_state INT;

-- 调用存储

EXEC dbo.v\_ydhl\_biaoben\_hedui\_historynew @zhuyuan\_id = '{zhuyuan\_id}',

@biaoben\_tiaoma= '{biaoben\_tiaoma}',

@operid = '{jiancha\_hedui\_hushi\_id}',

@opertime = '{jiancha\_hedui\_time}',

@RTNBOOLEAN = @upload\_state output ;

-- 返回值，赋值给upload\_state，更新回写状态时可以引用该字段

**SELECT @upload\_state AS out\_upload\_state**

END;

#### 更新回写状态

回写到mysql，type\_id配置为2：sql

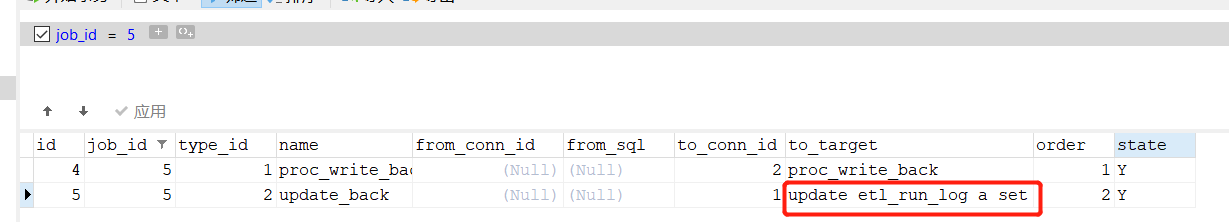
更新语句如下所示：

UPDATE etl\_run\_log a

SET update\_time = CURRENT\_TIME,

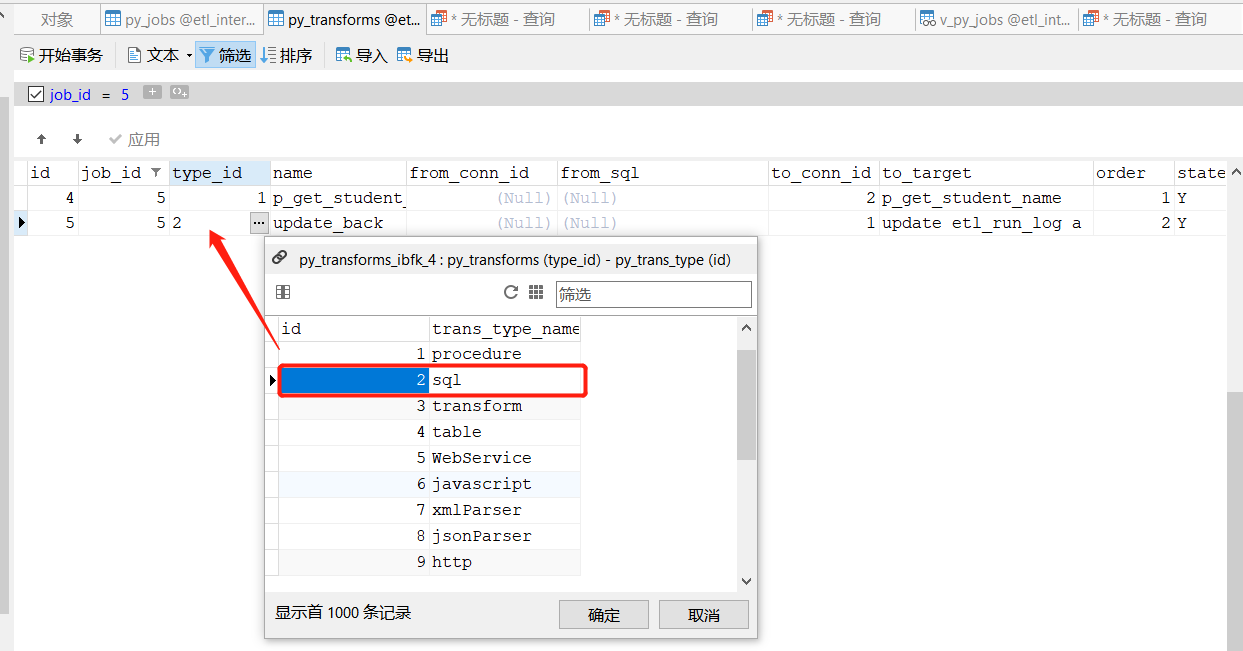
upload\_state = '{out\_upload\_state}'

WHERE a.id = '{id}'



### 直接回写表

py\_jobs配置方式同上，只要将transform中的配置类型（type\_id）置为2，并在to\_target字段中配置SQL即可，也可以写Oracle、SQL SERVER的语句块，或者执行多行MySQL语句。



### WebService回写

示例WebService：

<http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl>



打开连接，可知接口中

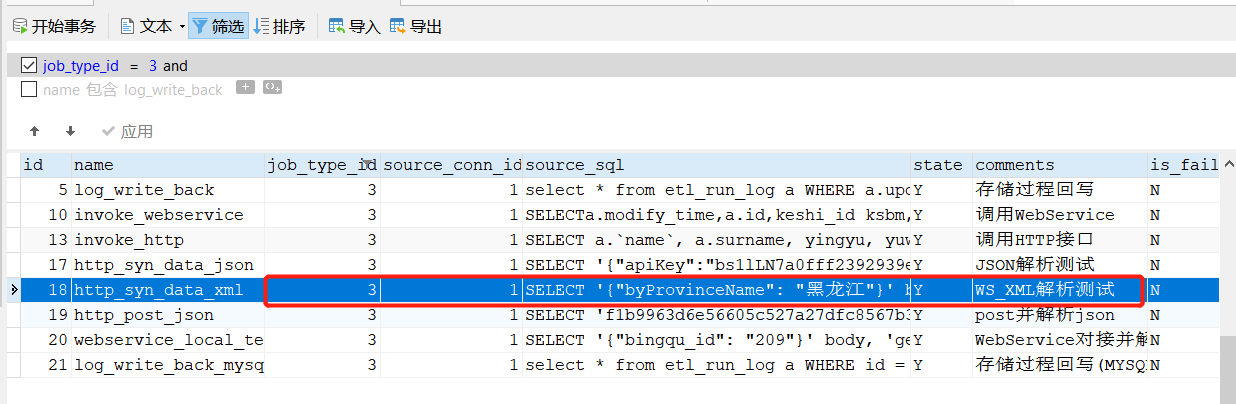
存在方法：getSupportCity

入参：byProvinceName

#### 入参配置

轮询记录

首先配置一条py\_jobs记录，job\_type\_id=3（stream）将轮询处理所有查询的记录：



这里source\_sql配置为：

SELECT '{"byProvinceName": "黑龙江"}' body, 'getSupportCity' method, 'UTF-8' encode

method:对应示例中的方法名， Body：以JSON格式配置入参，如果有其他参数也可以在SQL中配置上。

SQL结果只有一条记录，所以只会轮询一遍。

入参中json key 为byProvinceName（方法名）， json value为：黑龙江

如果入参为xml格式，则需要拼接xml入参，如新建医院的医嘱执行记录回写案例如下所示，方法为：invoke，入参为 code和indata：

SELECT CONCAT('{"code": "OrderExecInfoAdd", "indata": "<request><headers>',

                '<serial>',uuid(),'</serial>',

                '<orgId>2</orgId>',

                '<authCode>1</authCode>',

                '<oprId>00001</oprId>',

                '<hosId>96</hosId>',

        '</headers>',

        '<inPatId>',a.inPatId,'</inPatId>',

        '<orders>',

                '<order>',

                        '<orderId>',a.orderId,'</orderId>',

                        '<execTime>',a.execTime,'</execTime>',

                        '<execCount>',a.execCount,'</execCount>',

                        '<state>1</state>',

                        '<execTimeLow>',ifnull(a.execTimeLow,''),'</execTimeLow>',

                        '<execTimeHigh>',a.execTimeHigh,'</execTimeHigh>',

                        '<nurseBeginTime>',ifnull(a.execTimeLow,''),'</nurseBeginTime>',

                        '<nurseEndTime>',a.execTimeHigh,'</nurseEndTime>',

                        '<nurseBeginNo>',ifnull(a.nurseBeginNo,'00001'),'</nurseBeginNo>',

                        '<nurseBegin>',ifnull(a.nurseBegin,'管理员'),'</nurseBegin>',

                        '<nurseEndNo>',ifnull(a.nurseEndNo,'00001'),'</nurseEndNo>',

                        '<nurseEnd>',ifnull(a.nurseEnd,'管理员'),'</nurseEnd>',

                        '<nurseCheckNo>',ifnull(a.nurseCheckNo,'00001'),'</nurseCheckNo>',

                        '<nurseCheck>',ifnull(a.nurseCheck,'管理员'),'</nurseCheck>',

                '</order>',

        '</orders>',

'</request>"}')body,

'invoke' method, 'UTF-8' encode,a.zhixing\_id

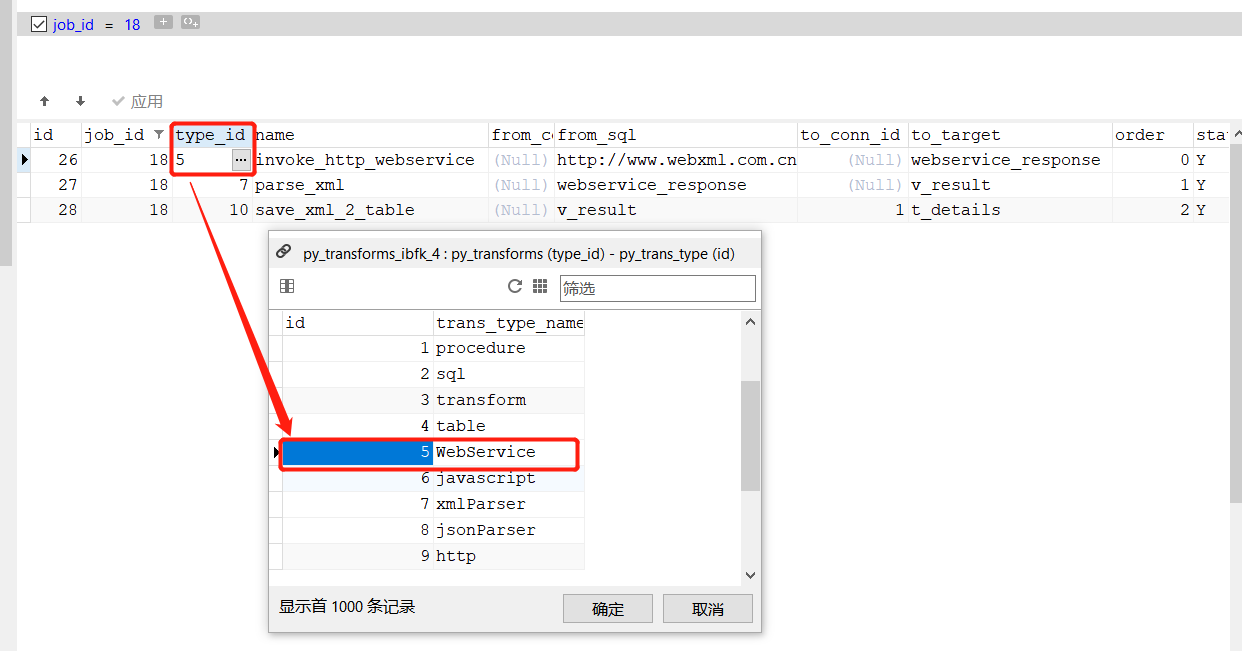
FROM

        view\_yizhu\_zhixing\_record a

-- where a.inPatId='22050910', elapsed time：0:00:00.001695

#### 调用url接口

配置py\_transforms，type\_id=5将url（<http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl>）填入from\_sql字段，并将返回结果赋值给to\_target中配置的变量（webservice\_response），以供我们讲返回结果写入表中。



#### 解析XML

配置py\_transforms，type\_id=7，将返回结果webservice\_response配置到from\_sql字段，并将解析后的结果存储到to\_target中配置的变量：v\_result中。

另外，需要在（py\_data\_parser）中配置解析的节点：

案例待完善。

#### 解析JSON

配置py\_transforms，type\_id=8，将返回结果webservice\_response配置到from\_sql字段，并将解析后的结果存储到to\_target中配置的变量：v\_result中。

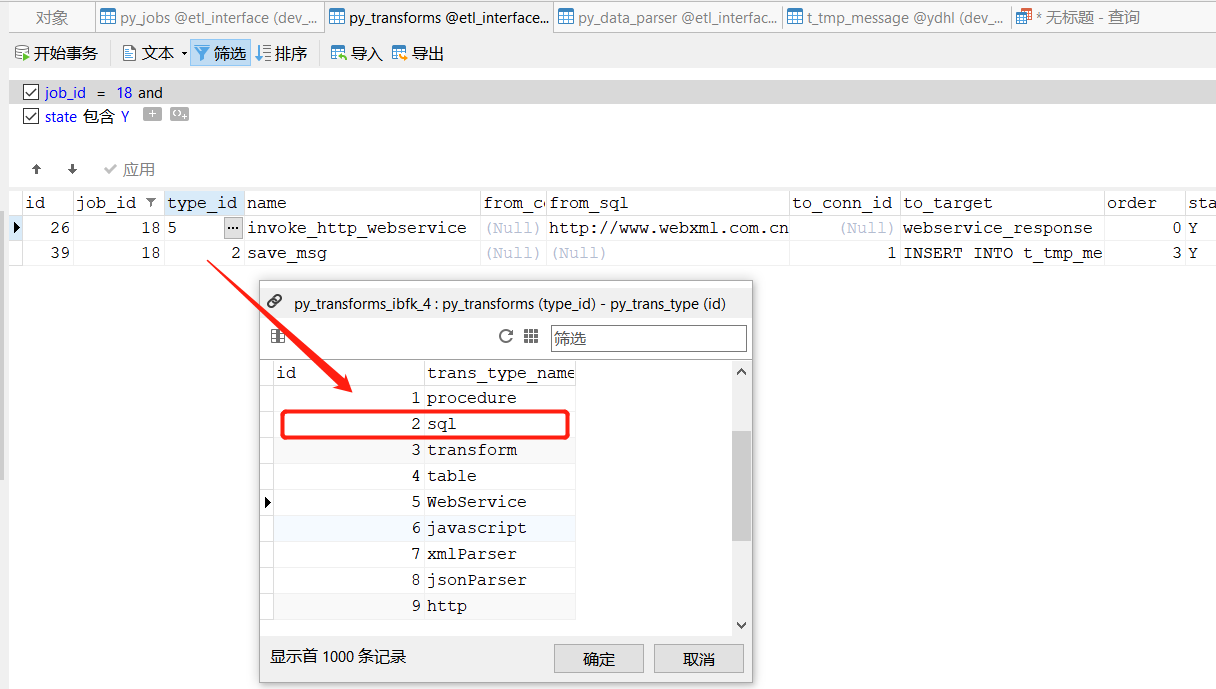
另外，需要在（py\_data\_parser）中配置解析的节点：

案例待完善。

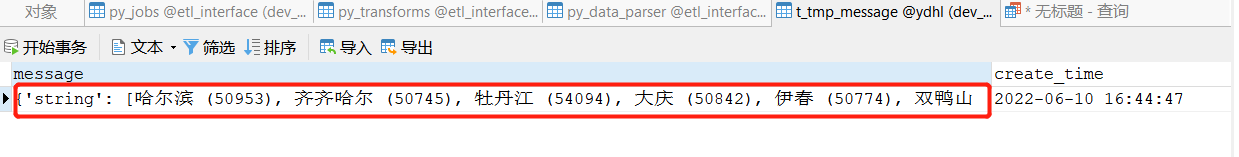
#### 整个返回消息存入结果表

可以直接将返回结果保存到数据库表中，只需要在py\_transforms中配置一条sql：

INSERT INTO t\_tmp\_message (message) VALUES ('{webservice\_response}')



返回结果如下所示



将解析后的结果存入

### HTTP回写

案例job\_id in(13, 19)

HTTP回写（type\_id=9）的配置方式与WebService类似：

入参（py\_jobs.source\_sql）可以不需要method字段：如下所示：

SELECT 'f1b9963d6e56605c527a27dfc8567b3c' AS p\_key, '116.481028,39.989643' AS origin, '116.434446,39.90816' AS destination, '{}' AS headers, '{}' AS body, 'utf-8' as encode -- post的入参需要配置body

调用的url（py\_transforms.from\_sql）可以通过大括号，将参数引用进来：如下所示：

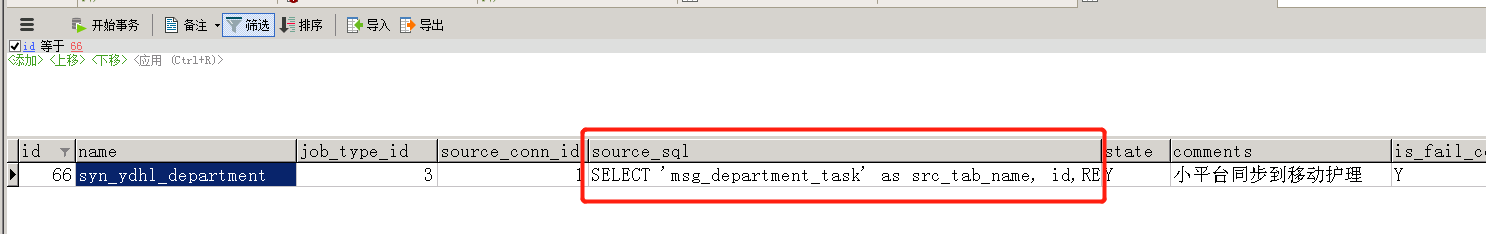
[https://restapi.amap.com/v3/direction/walking?key={p\_key}&origin={origin}&destination={destination}](https://restapi.amap.com/v3/direction/walking?key=%7bp_key%7d&origin=%7borigin%7d&destination=%7bdestination%7d)

### 动态SQL执行

以盱眙小平台配置为例：

直接将py\_jobs配置出来的字段，作为一整个SQL，配置到py\_transforms中：

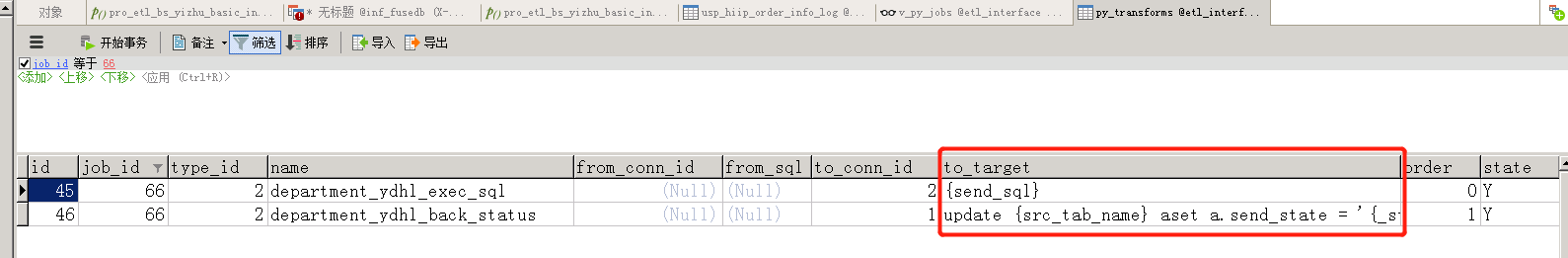
源数据配置：



source\_sql为：

SELECT 'msg\_department\_task' as src\_tab\_name, id,REPLACE (send\_sql, '''''', 'NULL') send\_sql FROM msg\_department\_task WHERE send\_syscode = '1' AND send\_state = '0' and ifnull(send\_sql, '') != ''

transform执行逻辑如下：



to\_target配置为：

{send\_sql}，直接执行send\_sql字段中的SQL

执行完后，将执行结果回写到表中：

update {src\_tab\_name} a

set a.send\_state = '{\_status}', a.send\_desc = '{\_errmsg}'

WHERE id = '{id}'