第14章

数据质量

数据质量的高低代表了该数据满足数据消费者期望的程度，这种程度基于他们对数据的使用预期，只有达到数据的使用预期才能给予管理层正确的决策参考。数据质量管理作为数据仓库的一个重要模块，主要可以分为数据的健康标准量化、监控和保障。

随着大数据时代的带来，数据的应用也日趋繁茂，越来越多的应用和服务都基于数据而建立，数据的重要性不言而喻。而且，数据质量是数据分析和数据挖掘结论有效性和准确性的基础，也是这一切的数据驱动决策的前提！如何保障数据质量，确保数据可用性是每一位数据人都不可忽略的重要环节。

## 14.1 数据质量管理概述

### 14.1.1 数据质量管理定义

数据质量管理（Data Quality Management），是指对[数据](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE/5947370)从计划、获取、存储、共享、维护、应用、消亡生命周期的每个阶段里可能引发的各类数据质量问题，进行识别、度量、监控、预警等一系列管理活动，并通过改善和提高组织的管理水平使得[数据质量](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%B4%A8%E9%87%8F/5134536)获得进一步提高。

数据质量管理是循环管理过程，其终极目标是通过可靠的数据提升数据在使用中的价值，并最终为企业赢得经济效益。

### 14.1.2 数据质量评估

由于[数据清洗](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%B8%85%E6%B4%97/4402497" \t "_blank)（DataCleaning）工具通常简单地被称为[数据质量](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%B4%A8%E9%87%8F/5134536" \t "_blank)（Data Quality）工具，因此很多人认为数据质量管理，就是修改数据中的错误、是对错误数据和垃圾数据进行清理。这个理解是片面的，其实数据清洗只是数据质量管理中的一步。数据质量管理（DQM），不仅包含了对数据质量的改善，同时还包含了对组织的改善。针对数据的改善和管理，主要包括数据分析、数据评估、数据清洗、数据监控、错误预警等内容；针对组织的改善和管理，主要包括确立组织数据[质量改进](https://baike.baidu.com/item/%E8%B4%A8%E9%87%8F%E6%94%B9%E8%BF%9B)目标、评估[组织流程](https://baike.baidu.com/item/%E7%BB%84%E7%BB%87%E6%B5%81%E7%A8%8B)、制定组织流程改善计划、制定组织监督审核机制、实施改进、评估改善效果等多个环节。

任何改善都是建立在评估的基础上，知道问题在哪才能实施改进。通常[数据质量](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%B4%A8%E9%87%8F/5134536)评估和管理评估需通过以下几个维度衡量，详见表14-1所示。

表14-1 数据质量评估维度

|  |  |  |
| --- | --- | --- |
| 数据特性 | 描述 | 监控项 |
| 唯一性 | 指主键保持唯一 | 字段唯一性检查 |
| 一致性 | 逻辑的一致性，总分的一致性 | 指标值平衡关系检查 |
| 总分平衡检查 |
| 完整性 | 主要包括实体缺失、属性缺失、记录缺失和字段值缺失四个方面 | 字段枚举值检查 |
| 字段记录数检查 |
| 字段空值检查 |
| 精确度 | 数据生成的正确性，数据在整个链路流转的正确性 | 波动阀值检查 |
| 合法性 | 主要包括格式、类型、域值和业务规则的有效性 | 字段日期格式检查 |
| 字段长度检查 |
| 字段值域检查 |
| 时效性 | 主要包括数据处理的时效性 | 批处理是否按时完成，或者是批处理完成分区无数据 |

* 及时性：数据获取是否及时，主要指数据提取、传送、转换、加载、展现的及时性。在数据处理的各个环节，都会涉及到及时性。我们一般考虑两个方面第一就是接口数据是否能够及时的抽取过来。第二就是展现层能否及时的展现出来。
* 完整性：是指数据是否完整，描述的数据要素，要素属性及要素关系存在或不存在，主要包括实体缺失、属性缺失、记录缺失以及主外键参照完整性的内容。
* 一致性：第一就是原始数据即文件接口和入库的数据记录条数是一致的。第二就是同一指标在任何地方都应该保持一致。
* 有效性：描述数据取值是否在界定的值域范围内，主要包括数据格式、数据类型、值域和相关业务规则的有效性。
* 准确性：主要是指指标算法、数据处理过程的准确性。这个准确性主要是通过元数据管理中定义的指标的算法、数据处理顺序和人工检查相结合的方式来保证。

### 14.1.3 数据质量影响因素

影响[数据质量](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%B4%A8%E9%87%8F/5134536)的因素主要来源于四方面：信息因素、技术因素、流程因素和管理因素。

**1.信息因素**

产生这部分数据质量问题的原因主要有：[元数据](https://baike.baidu.com/item/%E5%85%83%E6%95%B0%E6%8D%AE/1946090)描述及理解错误、数据[度量](https://baike.baidu.com/item/%E5%BA%A6%E9%87%8F)的各种性质（如：数据源规格不统一）得不到保证和变化频度不恰当等。

**2.技术因素**

主要是指由于具体数据处理的各技术环节的异常造成的数据质量问题。数据质量问题的产生环节主要包括数据创建、[数据获取](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%8E%B7%E5%8F%96/10813444)、数据传输、[数据装载](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%A3%85%E8%BD%BD/3200976)、数据使用、数据维护等方面的内容。

**3.流程因素**

是指由于系统作业流程和人工操作流程设置不当造成的数据质量问题，主要来源于系统数据的创建流程、传递流程、装载流程、使用流程、维护流程和[稽核](https://baike.baidu.com/item/%E7%A8%BD%E6%A0%B8)流程等各环节。

**4.管理因素**

是指由于人员素质及[管理机制](https://baike.baidu.com/item/%E7%AE%A1%E7%90%86%E6%9C%BA%E5%88%B6)方面的原因造成的数据质量问题。如人员培训、[人员管理](https://baike.baidu.com/item/%E4%BA%BA%E5%91%98%E7%AE%A1%E7%90%86)、培训或者奖惩措施不当导致的管理缺失或者管理缺陷。

## 14.2 数据质量监控需求

我们的数仓项目主要是监视以下四类数据的指标：

* 原始数据层采集的数据，每日环比和每周同比增长不能超过一定范围
* 维度层id空值，重复值不能出现；
* 数据明细层不能出现空值id；
* 用户浏览页面的时间和订单金额的异常值（太大或者太小）不能过多。

数据质量监控需求规划如表14-1所示。

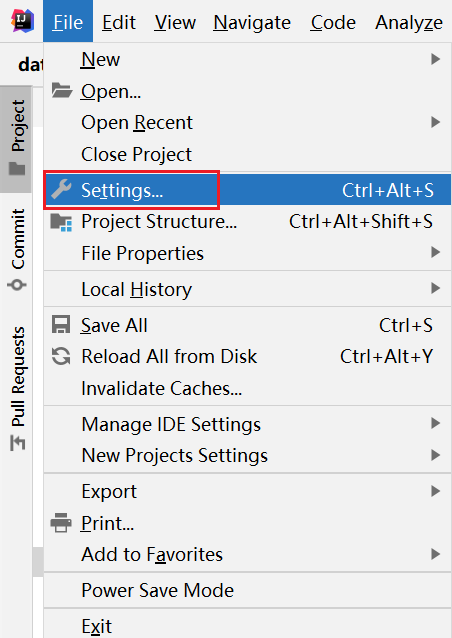
表14-1 数据质量监控需求规划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表 | 检查项目 | 依据 | 异常值上限 | 异常值下限 |
| ods\_order\_detail | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_order\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| 值域检查 | final\_amount | 0 | 100 |
| ods\_order\_refund\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_payment\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_refund\_payment | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_user\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| dwd\_order\_detail | 空值检查 | id | 0 | 10 |
| dwd\_order\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dwd\_order\_refund\_info | 空值检查 | id | 0 | 10 |
| dwd\_payment\_info | 空值检查 | id | 0 | 10 |
| dwd\_refund\_payment | 空值检查 | id | 0 | 10 |
| dim\_activity\_rule\_info | 空值检查 | active\_id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_coupon\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_sku\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_user\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |

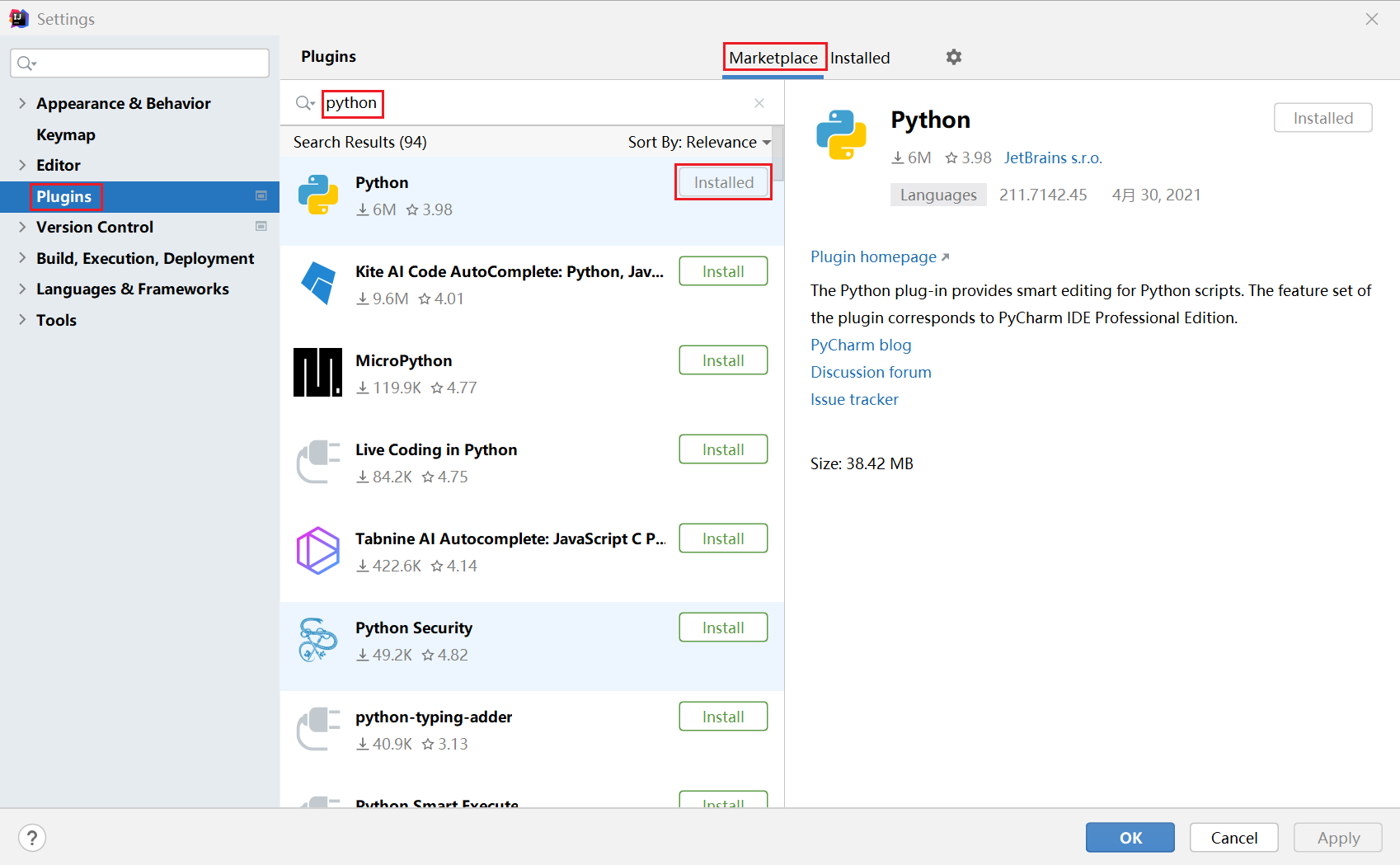
## 14.3 数据质量监控实现

### 14.3.1 IDEA中Python环境配置

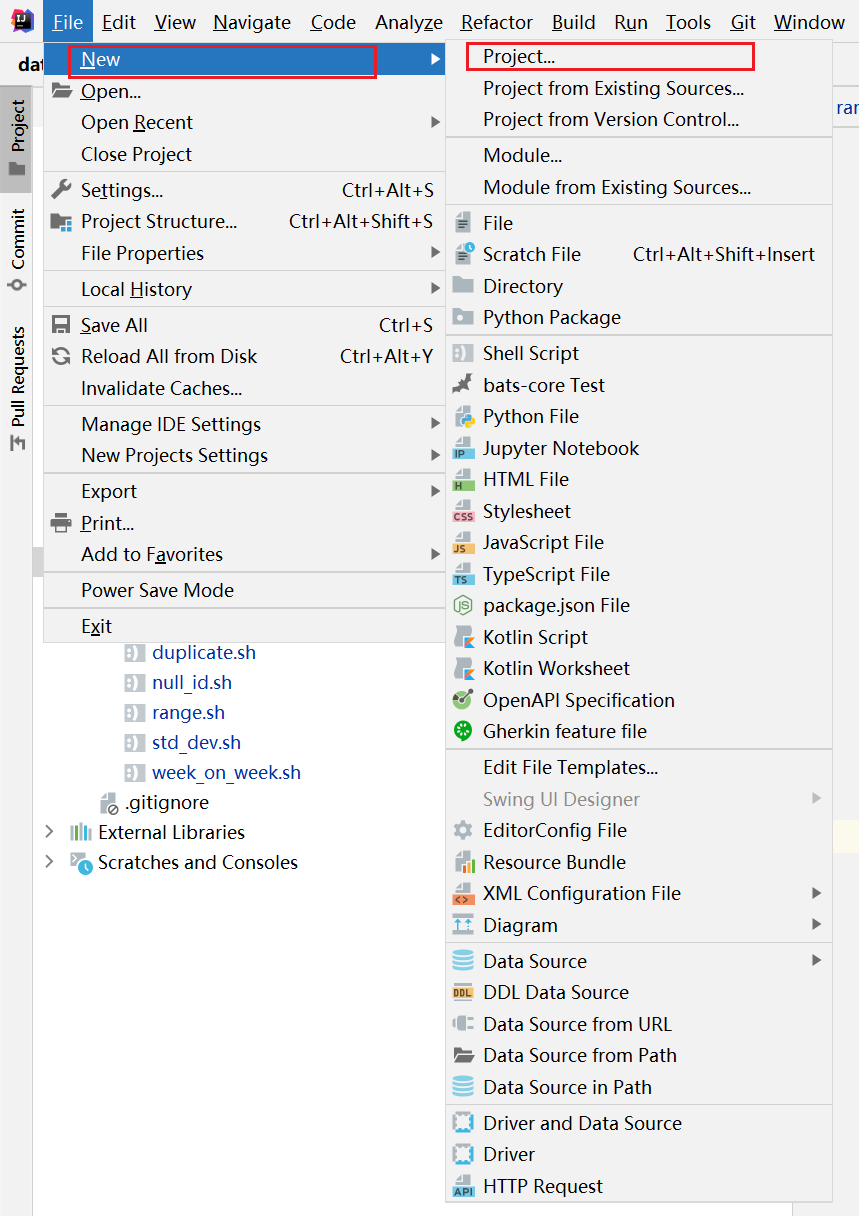
**1. 安装python插件**

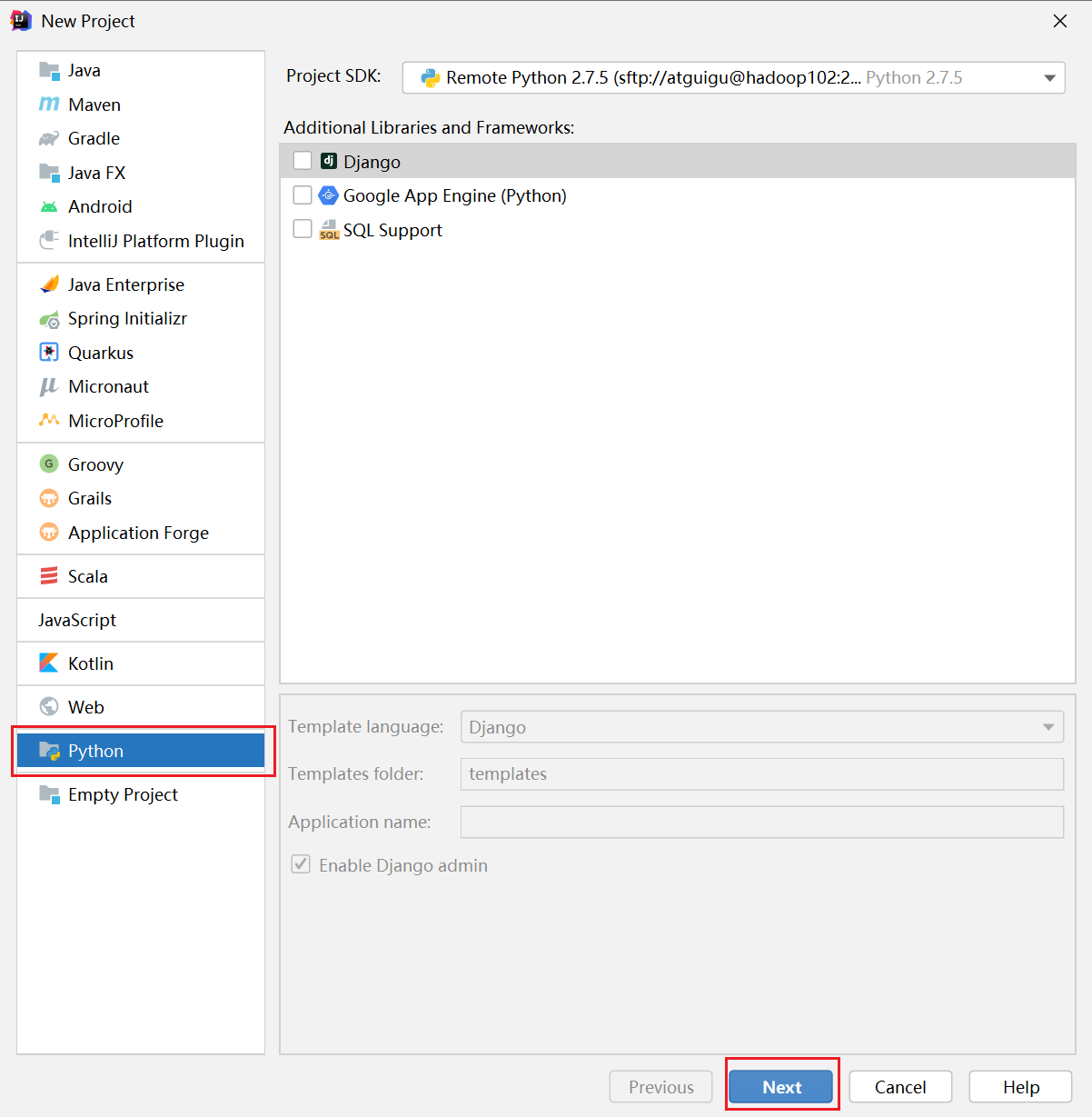


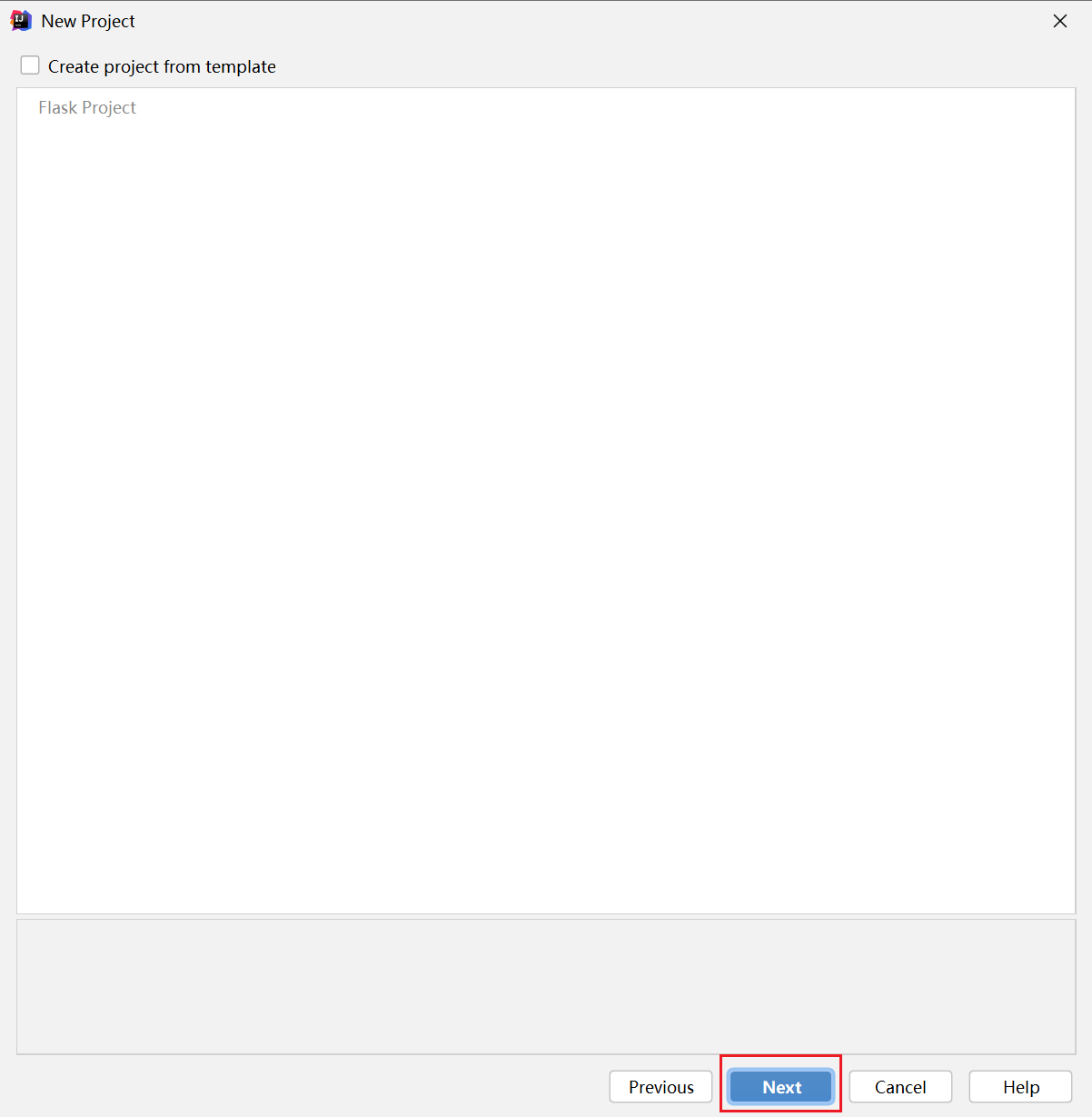
打开插件菜单，搜索python，安装并重启：

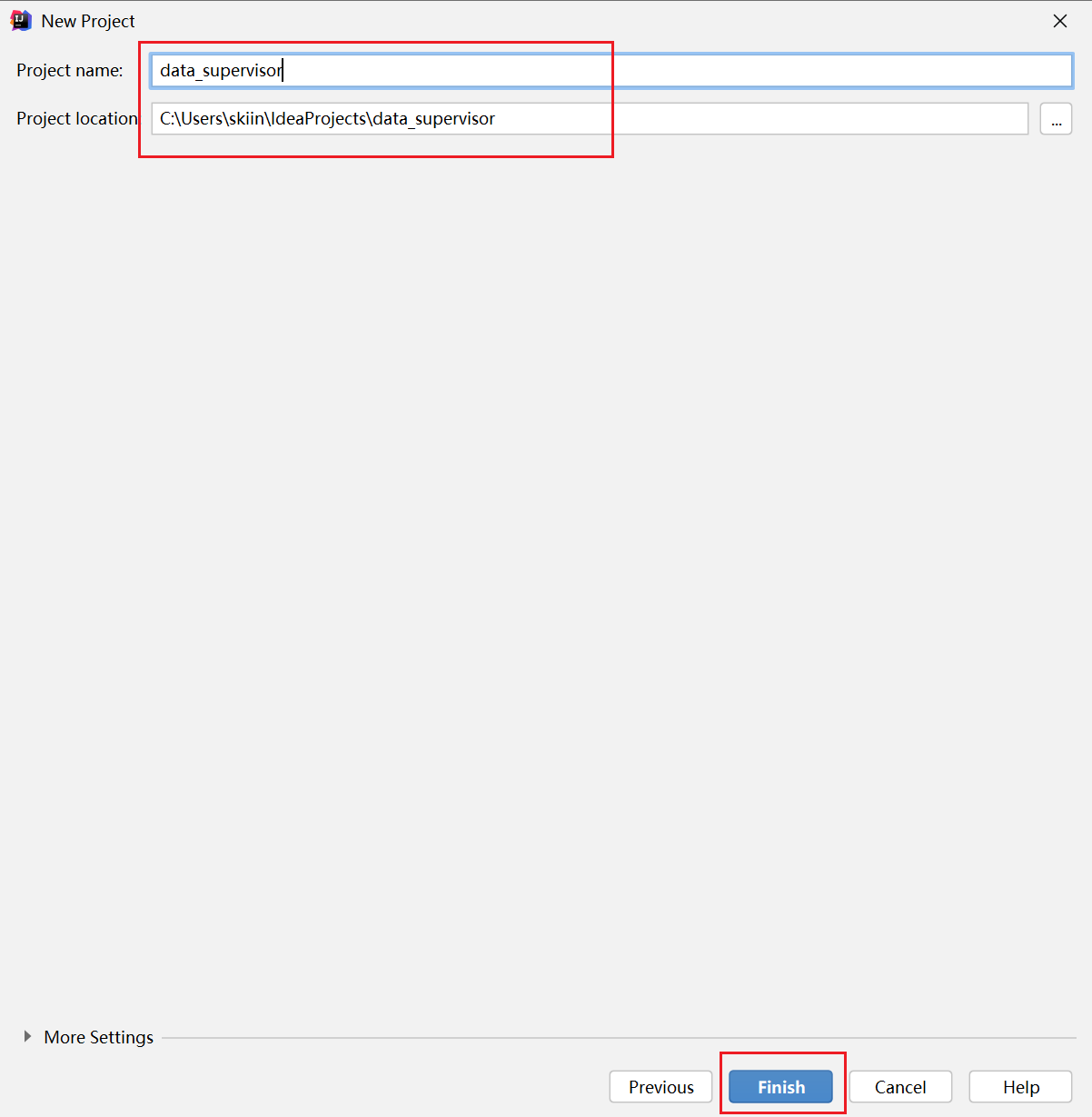


**2. 新建一个python项目**



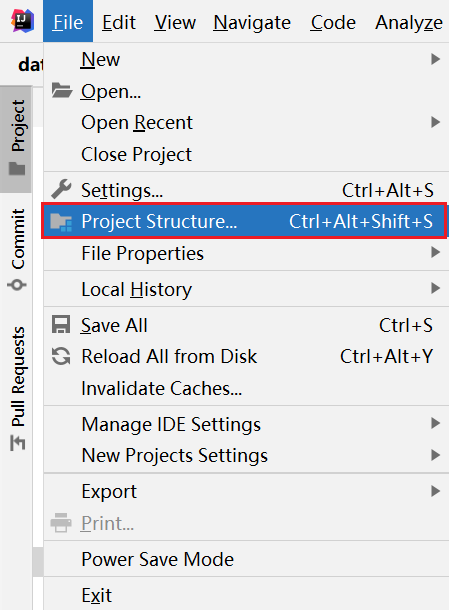


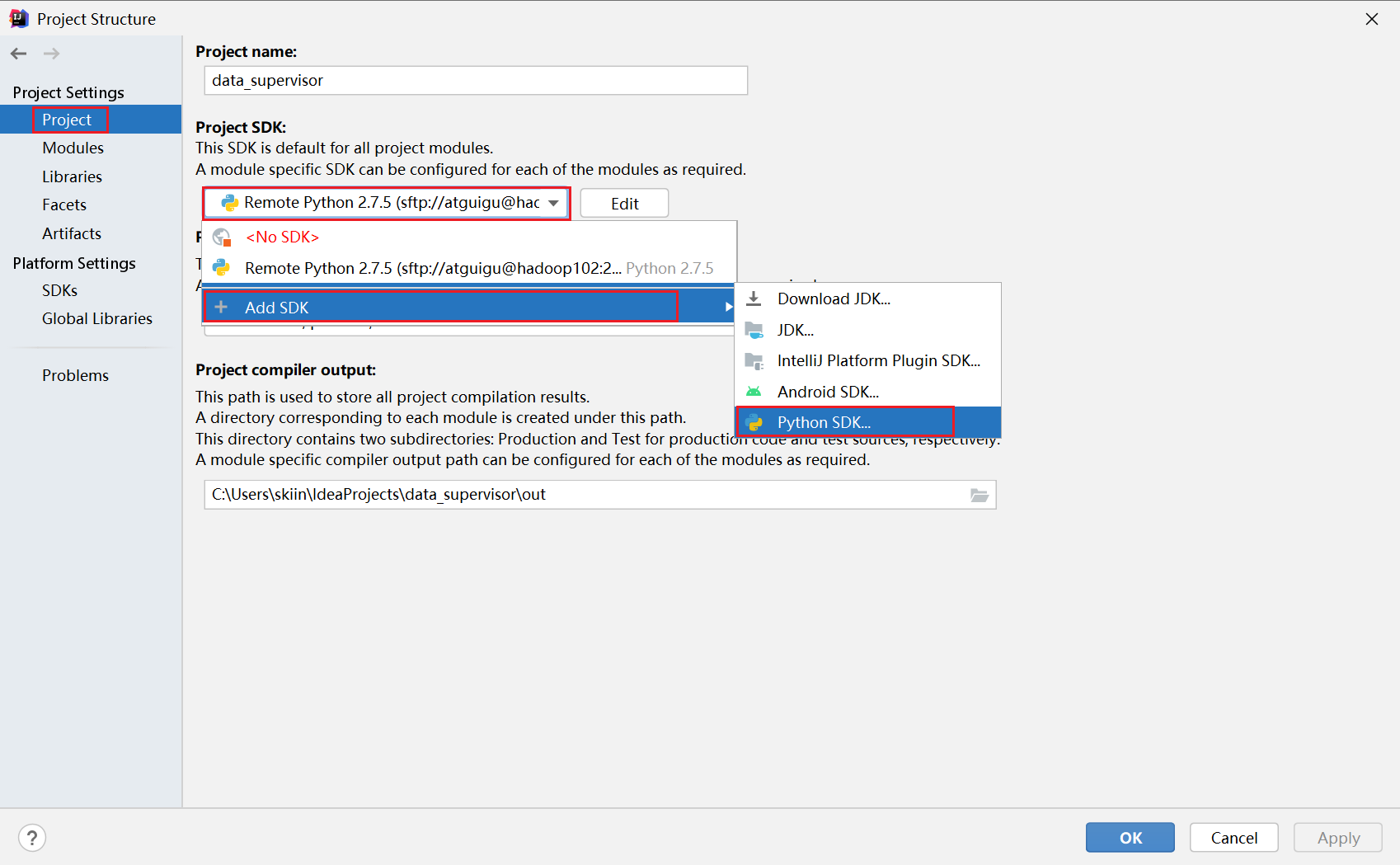


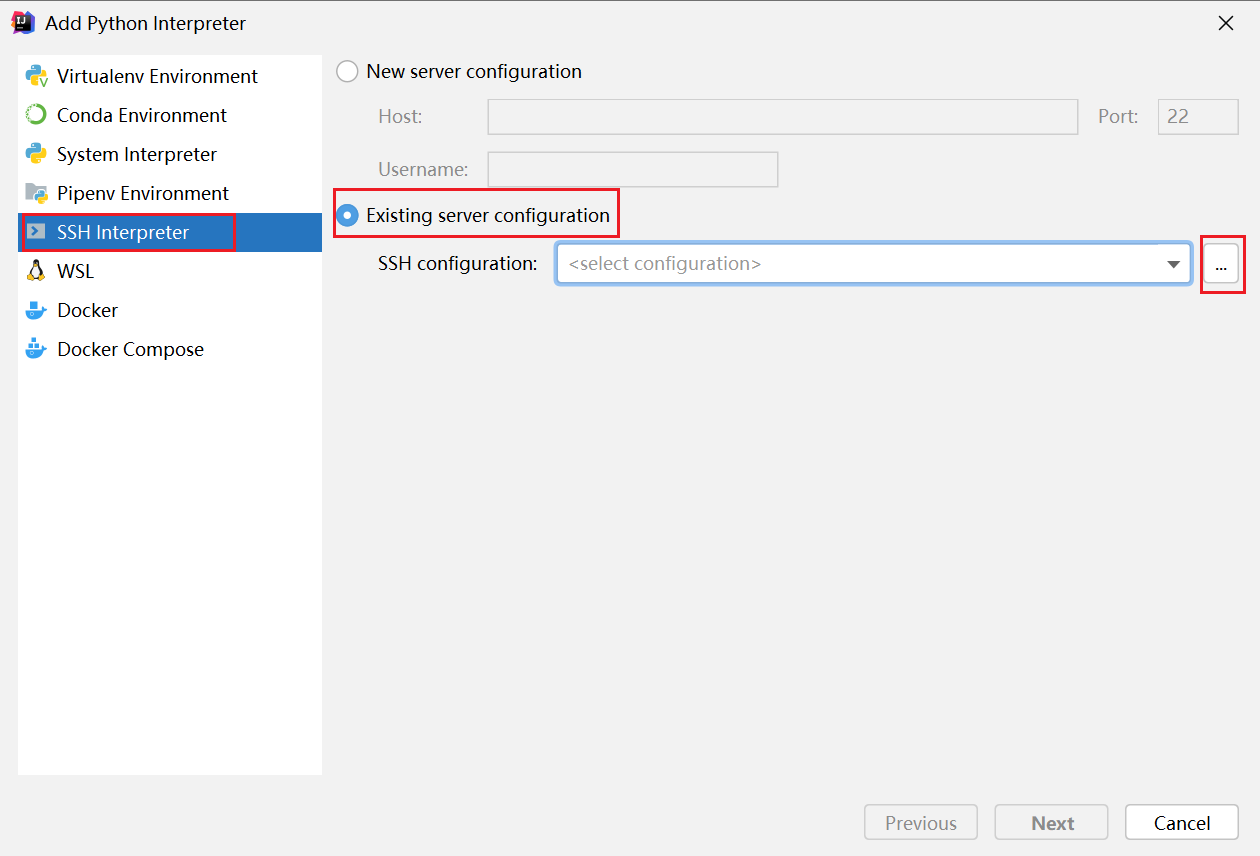


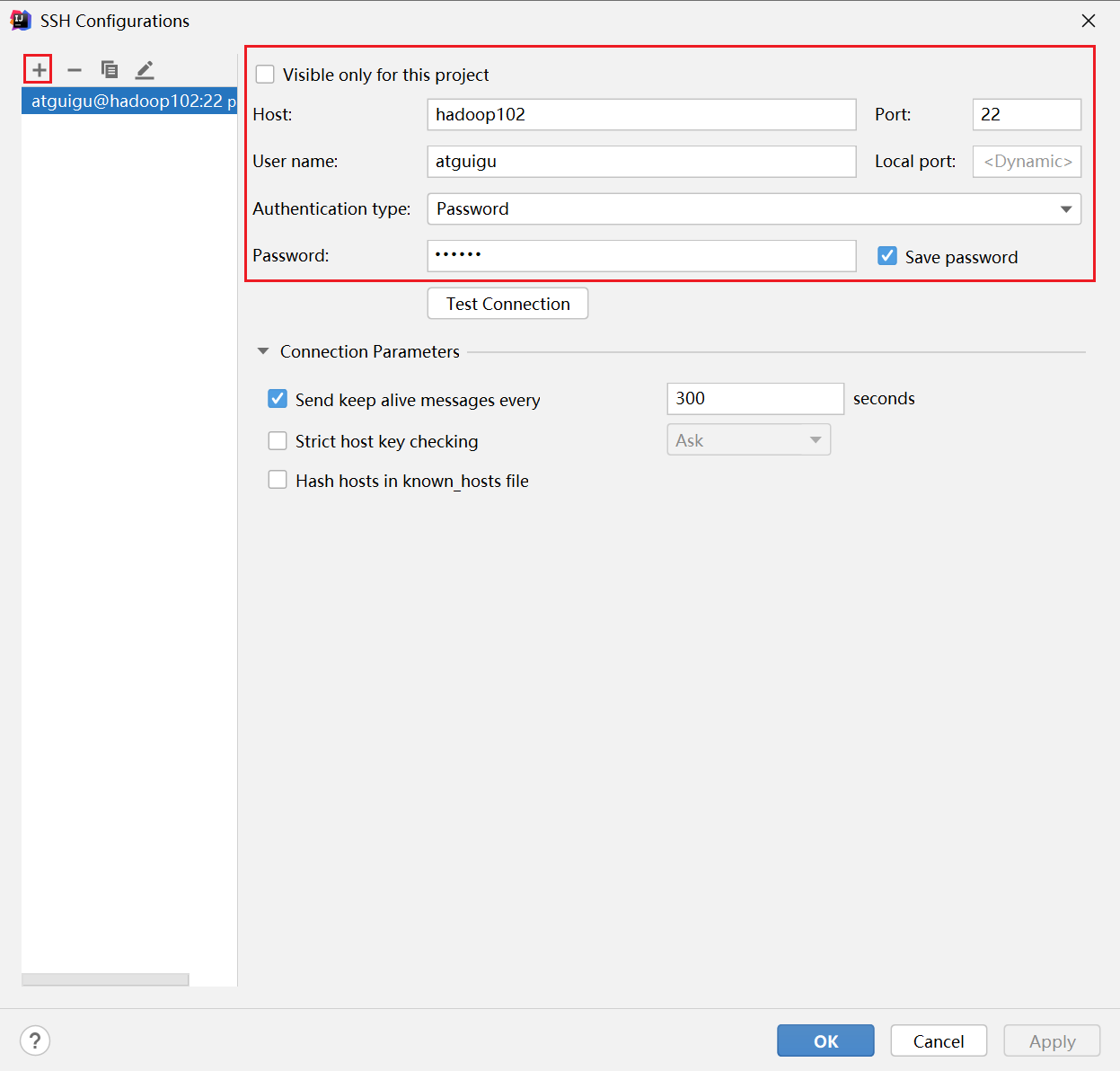
**3. 配置ssh的python解释器**

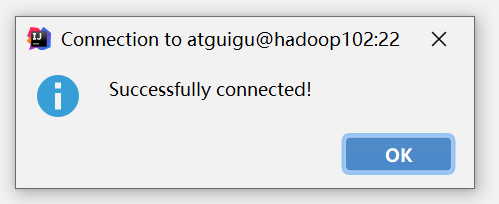
由于我们需要编写在集群运行的python脚本，所以将项目的python解释器配置为远程ssh解释器，更好的保证环境一致，不用在windows安装python解释器：



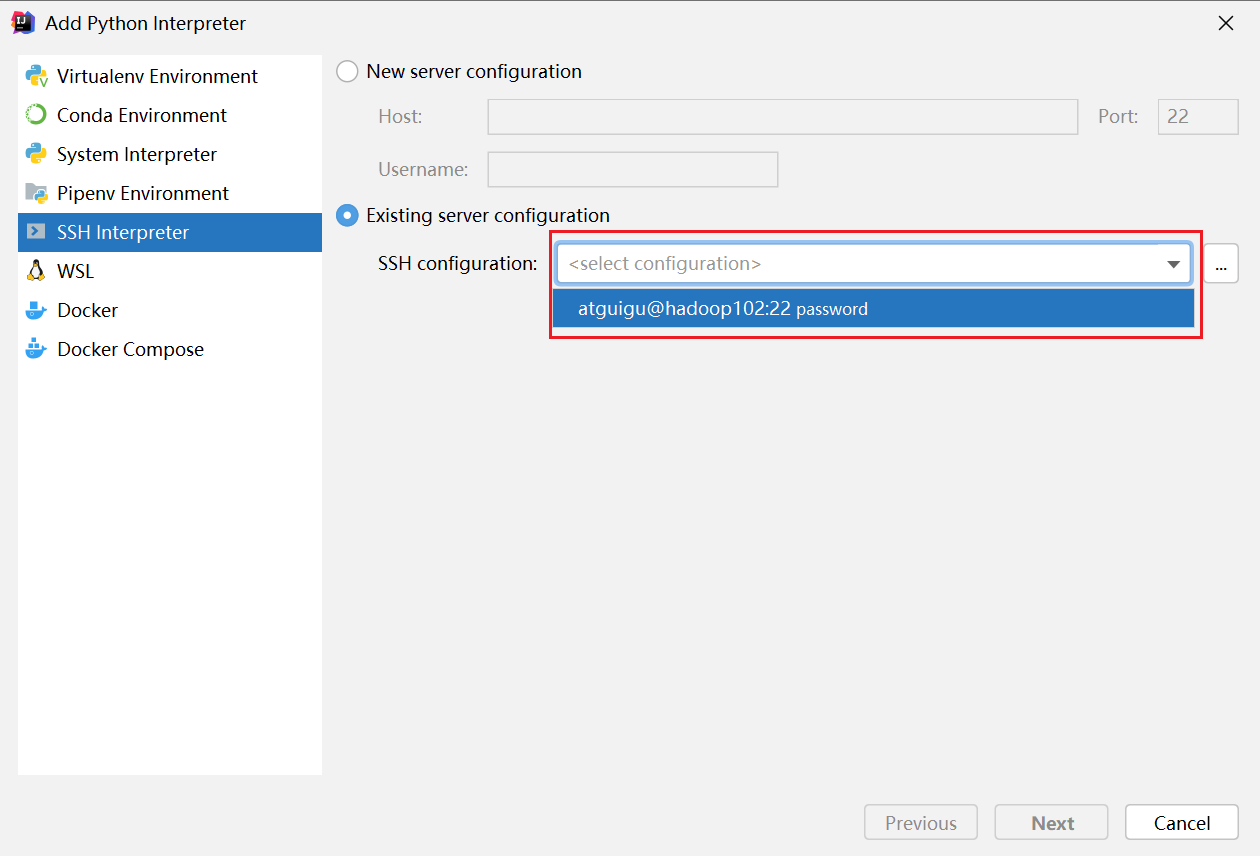


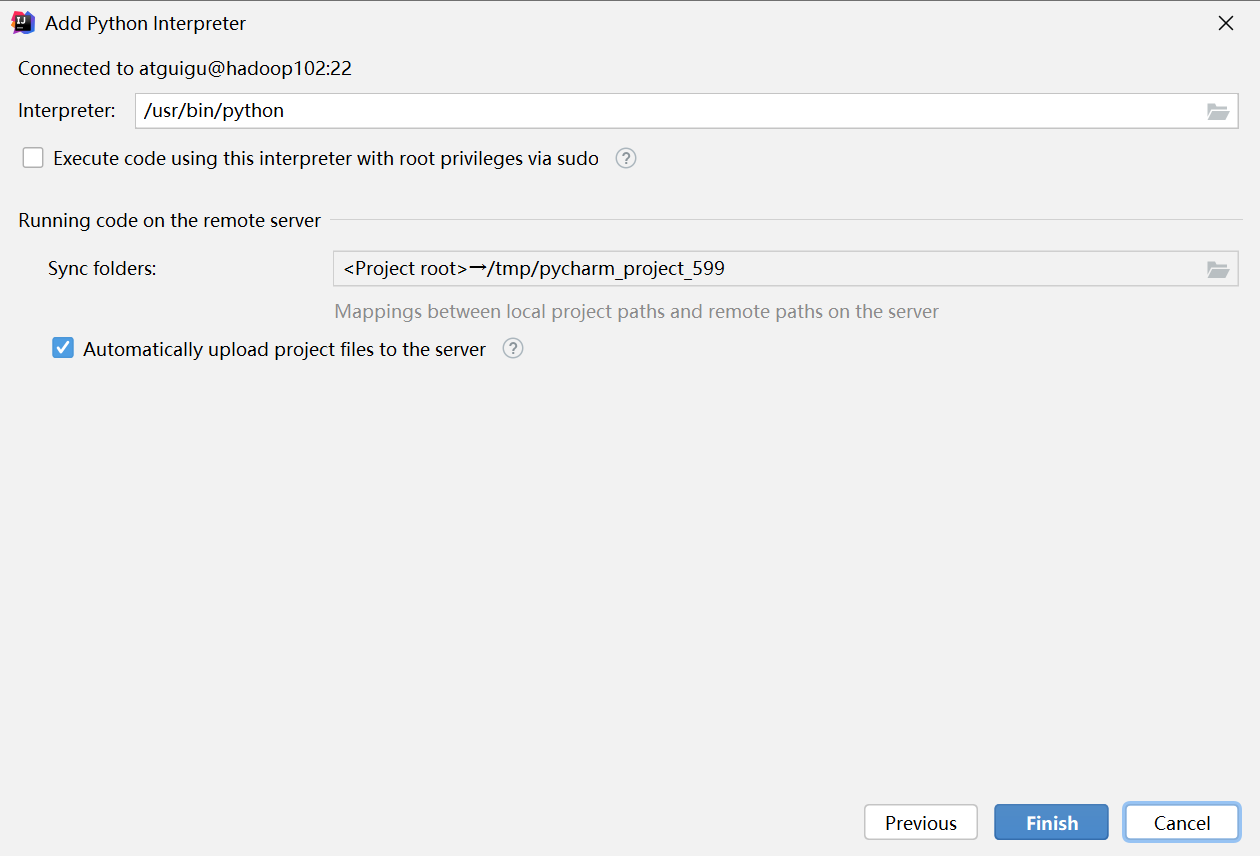


最后点击Test Connection，出现如下提示：

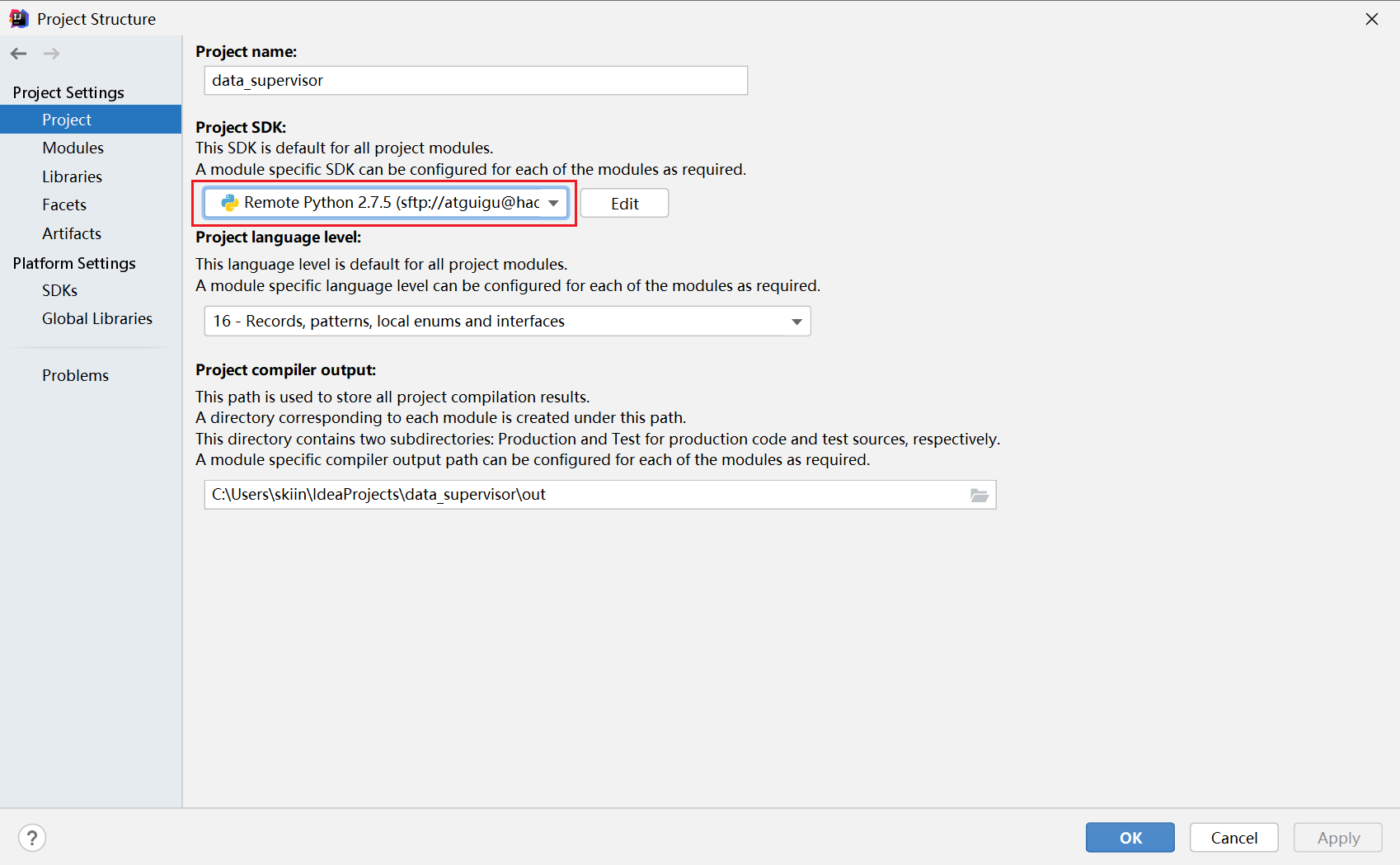


然后选择我们刚刚配置好的服务器：





然后我们就可以看到刚刚创建的远程解释器，点击OK即可



### 14.3.2 检测规则脚本编写

**1. 空id检查脚本**

新建文件 null\_id.sh, 内容如下：

#!/usr/bin/env bash

# -\*- coding: utf-8 -\*-

# 检查di为空，参数说明如下：

# -t 表名

# -d 日期

# -c 列名

# -s 指标下限

# -x 指标上限

# -l 告警级别

while getopts "t:d:c:s:x:l:" arg; do

case $arg in

t)

TABLE=$OPTARG

;;

d)

DT=$OPTARG

;;

c)

COL=$OPTARG

;;

s)

MIN=$OPTARG

;;

x)

MAX=$OPTARG

;;

l)

LEVEL=$OPTARG

;;

?)

echo "unkonw argument"

exit 1

;;

esac

done

[ $DT ] || DT=$(date -d '-1 day' +%F)

[ $LEVEL ] || LEVEL=0

HIVE\_DB=gmall

HIVE\_ENGINE=hive

mysql\_user="root"

mysql\_passwd="000000"

mysql\_host="hadoop102"

mysql\_DB="test"

mysql\_tbl="null\_id"

RESULT=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt='$DT' and $COL is null;")

mysql -h"$mysql\_host" -u"$mysql\_user" -p"$mysql\_passwd" \

-e"INSERT INTO $mysql\_DB.$mysql\_tbl VALUES('$DT', '$TABLE', '$COL', $RESULT, $MIN, $MAX, $LEVEL)

ON DUPLICATE KEY UPDATE \`value\`=$RESULT, value\_min=$MIN, value\_max=$MAX, notification\_level=$LEVEL;"

**2. 重复id检查脚本**

新建文件duplicate.sh，内容如下：

#!/usr/bin/env bash

# -\*- coding: utf-8 -\*-

# 检查di为空，参数说明如下：

# -t 表名

# -d 日期

# -c 列名

# -s 指标下限

# -x 指标上限

# -l 告警级别

while getopts "t:d:c:s:x:l:" arg; do

case $arg in

t)

TABLE=$OPTARG

;;

d)

DT=$OPTARG

;;

c)

COL=$OPTARG

;;

s)

MIN=$OPTARG

;;

x)

MAX=$OPTARG

;;

l)

LEVEL=$OPTARG

;;

?)

echo "unkonw argument"

exit 1

;;

esac

done

[ $DT ] || DT=$(date -d '-1 day' +%F)

[ $LEVEL ] || LEVEL=0

HIVE\_DB=gmall

HIVE\_ENGINE=hive

mysql\_user="root"

mysql\_passwd="000000"

mysql\_host="hadoop102"

mysql\_DB="test"

mysql\_tbl="null\_id"

RESULT=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt='$DT' and $COL is null;")

mysql -h"$mysql\_host" -u"$mysql\_user" -p"$mysql\_passwd" \

-e"INSERT INTO $mysql\_DB.$mysql\_tbl VALUES('$DT', '$TABLE', '$COL', $RESULT, $MIN, $MAX, $LEVEL)

ON DUPLICATE KEY UPDATE \`value\`=$RESULT, value\_min=$MIN, value\_max=$MAX, notification\_level=$LEVEL;"

**3. 值域检查脚本**

新建文件range.sh，内容如下：

#!/usr/bin/env bash

# -\*- coding: utf-8 -\*-

# 检查di为空，参数说明如下：

# -t 表名

# -d 日期

# -c 列名

# -s 指标下限

# -x 指标上限

# -l 告警级别

# -a 值域上限

# -b 值域下限

while getopts "t:d:l:c:s:x:a:b:" arg; do

case $arg in

t)

TABLE=$OPTARG

;;

d)

DT=$OPTARG

;;

c)

COL=$OPTARG

;;

s)

MIN=$OPTARG

;;

x)

MAX=$OPTARG

;;

l)

LEVEL=$OPTARG

;;

a)

RANGE\_MIN=$OPTARG

;;

b)

RANGE\_MAX=$OPTARG

;;

?)

echo "unkonw argument"

exit 1

;;

esac

done

[ $DT ] || DT=$(date -d '-1 day' +%F)

[ $LEVEL ] || LEVEL=0

HIVE\_DB=gmall

HIVE\_ENGINE=hive

mysql\_user="root"

mysql\_passwd="000000"

mysql\_host="hadoop102"

mysql\_DB="test"

mysql\_tbl="rng"

RESULT=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt='$DT' and $COL not between $RANGE\_MIN and $RANGE\_MAX;")

mysql -h"$mysql\_host" -u"$mysql\_user" -p"$mysql\_passwd" \

-e"INSERT INTO $mysql\_DB.$mysql\_tbl VALUES('$DT', '$TABLE', '$COL', $RESULT, $RANGE\_MIN, $RANGE\_MAX, $MIN, $MAX, $LEVEL)

ON DUPLICATE KEY UPDATE \`value\`=$RESULT, range\_min=$RANGE\_MIN, range\_max=$RANGE\_MAX, value\_min=$MIN, value\_max=$MAX, notification\_level=$LEVEL;"

**4. 数据量环比检查脚本**

新建文件day\_on\_day.sh，内容如下：

#!/usr/bin/env bash

# -\*- coding: utf-8 -\*-

# 检查di为空，参数说明如下：

# -t 表名

# -d 日期

# -c 列名

# -s 指标下限

# -x 指标上限

# -l 告警级别

while getopts "t:d:s:x:l:" arg; do

case $arg in

t)

TABLE=$OPTARG

;;

d)

DT=$OPTARG

;;

s)

MIN=$OPTARG

;;

x)

MAX=$OPTARG

;;

l)

LEVEL=$OPTARG

;;

?)

echo "unkonw argument"

exit 1

;;

esac

done

[ "$DT" ] || DT=$(date -d '-1 day' +%F)

[ "$LEVEL" ] || LEVEL=0

HIVE\_DB=gmall

HIVE\_ENGINE=hive

mysql\_user="root"

mysql\_passwd="000000"

mysql\_host="hadoop102"

mysql\_DB="test"

mysql\_tbl="day\_on\_day"

YESTERDAY=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt=date\_add('$DT',-1);")

TODAY=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt='$DT';")

if [ "$YESTERDAY" -ne 0 ]; then

RESULT=$(awk "BEGIN{print ($TODAY-$YESTERDAY)/$YESTERDAY\*100}")

else

RESULT=10000

fi

mysql -h"$mysql\_host" -u"$mysql\_user" -p"$mysql\_passwd" \

-e"INSERT INTO $mysql\_DB.$mysql\_tbl VALUES('$DT', '$TABLE', $RESULT, $MIN, $MAX, $LEVEL)

ON DUPLICATE KEY UPDATE \`value\`=$RESULT, value\_min=$MIN, value\_max=$MAX, notification\_level=$LEVEL;"

**5. 数据量同比检查脚本**

新建文件week\_on\_week.sh，内容如下：

#!/usr/bin/env bash

# -\*- coding: utf-8 -\*-

# 检查di为空，参数说明如下：

# -t 表名

# -d 日期

# -c 列名

# -s 指标下限

# -x 指标上限

# -l 告警级别

while getopts "t:d:s:x:l:" arg; do

case $arg in

t)

TABLE=$OPTARG

;;

d)

DT=$OPTARG

;;

s)

MIN=$OPTARG

;;

x)

MAX=$OPTARG

;;

l)

LEVEL=$OPTARG

;;

?)

echo "unkonw argument"

exit 1

;;

esac

done

[ $DT ] || DT=$(date -d '-1 day' +%F)

[ $LEVEL ] || LEVEL=0

HIVE\_DB=gmall

HIVE\_ENGINE=hive

mysql\_user="root"

mysql\_passwd="000000"

mysql\_host="hadoop102"

mysql\_DB="test"

mysql\_tbl="week\_on\_week"

LASTWEEK=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt between date\_add('$DT',-13) and date\_add('$DT',-7);")

THISWEEK=$($HIVE\_ENGINE -e "select count(1) from $HIVE\_DB.$TABLE where dt between date\_add('$DT',-6) and '$DT';")

if [ $LASTWEEK -ne 0 ]; then

RESULT=$(awk "BEGIN{print ($THISWEEK-$LASTWEEK)/$LASTWEEK\*100}")

else

RESULT=10000

fi

mysql -h"$mysql\_host" -u"$mysql\_user" -p"$mysql\_passwd" \

-e"INSERT INTO $mysql\_DB.$mysql\_tbl VALUES('$DT', '$TABLE', $RESULT, $MIN, $MAX, $LEVEL)

ON DUPLICATE KEY UPDATE \`value\`=$RESULT, value\_min=$MIN, value\_max=$MAX, notification\_level=$LEVEL;"

### 14.3.3 初始化MySQL环境

在MySQL中执行如下代码：

drop database if exists test;

create database test;

CREATE TABLE test.`null\_id`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`col` varchar(50) NOT NULL COMMENT '列名',

`value` int DEFAULT NULL COMMENT '空ID个数',

`value\_min` int DEFAULT NULL COMMENT '下限',

`value\_max` int DEFAULT NULL COMMENT '上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`, `col`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '空值指标表';

CREATE TABLE test.`duplicate`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`col` varchar(50) NOT NULL COMMENT '列名',

`value` int DEFAULT NULL COMMENT '重复值个数',

`value\_min` int DEFAULT NULL COMMENT '下限',

`value\_max` int DEFAULT NULL COMMENT '上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`, `col`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '重复值指标表';

CREATE TABLE test.`rng`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`col` varchar(50) NOT NULL COMMENT '列名',

`value` int DEFAULT NULL COMMENT '超出预定值域个数',

`range\_min` int DEFAULT NULL COMMENT '值域下限',

`range\_max` int DEFAULT NULL COMMENT '值域上限',

`value\_min` int DEFAULT NULL COMMENT '下限',

`value\_max` int DEFAULT NULL COMMENT '上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`, `col`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '值域指标表';

CREATE TABLE test.`day\_on\_day`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`value` double DEFAULT NULL COMMENT '同比增长百分比',

`value\_min` double DEFAULT NULL COMMENT '增长上限',

`value\_max` double DEFAULT NULL COMMENT '增长上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '环比增长指标表';

CREATE TABLE test.`week\_on\_week`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`value` double DEFAULT NULL COMMENT '环比增长百分比',

`value\_min` double DEFAULT NULL COMMENT '增长上限',

`value\_max` double DEFAULT NULL COMMENT '增长上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '同比增长指标表';

CREATE TABLE test.`std\_dev`

(

`dt` date NOT NULL COMMENT '日期',

`tbl` varchar(50) NOT NULL COMMENT '表名',

`col` varchar(50) NOT NULL COMMENT '列名',

`value` double DEFAULT NULL COMMENT '标准差',

`value\_min` double DEFAULT NULL COMMENT '标准差上限',

`value\_max` double DEFAULT NULL COMMENT '标准差上限',

`notification\_level` int DEFAULT NULL COMMENT '警告级别',

PRIMARY KEY (`dt`, `tbl`, `col`)

) ENGINE = InnoDB

DEFAULT CHARSET = utf8

comment '标准差增长指标表';

### 14.3.4 数据检查脚本编写

**1. ODS层**

ODS层需要检查的项目如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ods\_order\_detail | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_order\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| 值域检查 | final\_amount | 0 | 100 |
| ods\_order\_refund\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_payment\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_refund\_payment | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |
| ods\_user\_info | 同比增长 | 数据总量 | -10% | 10% |
| 环比增长 | 数据总量 | -10% | 50% |

编写如下脚本：

新建文件check\_ods.sh

#!/usr/bin/env bash

DT=$1

[ "$DT" ] || DT=$(date -d '-1 day' +%F)

# 检查表 ods\_order\_detail

bash day\_on\_day.sh -t ods\_order\_detail -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_order\_detail -d "$DT" -s -10 -x 50 -l 1

#检查表 ods\_order\_info

bash day\_on\_day.sh -t ods\_order\_info -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_order\_info -d "$DT" -s -10 -x 50 -l 1

bash range.sh -t ods\_order\_info -d "$DT" -c final\_amount -a 0 -b 100000 -s 0 -x 100 -l 1

#检查表 ods\_order\_refund\_info.sh

bash day\_on\_day.sh -t ods\_order\_refund\_info -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_order\_refund\_info -d "$DT" -s -10 -x 50 -l 1

#检查表 payment\_info

bash day\_on\_day.sh -t ods\_payment\_info -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_payment\_info -d "$DT" -s -10 -x 50 -l 1

#检查表 ods\_refund\_payment

bash day\_on\_day.sh -t ods\_refund\_payment -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_refund\_payment -d "$DT" -s -10 -x 50 -l 1

#检查表 ods\_user\_info

bash day\_on\_day.sh -t ods\_user\_info -d "$DT" -s -10 -x 10 -l 1

bash week\_on\_week.sh -t ods\_user\_info -d "$DT" -s -10 -x 50 -l 1

**2. DWD层**

DWD层需要检查的项目如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| dwd\_order\_detail | 空值检查 | id | 0 | 10 |
| dwd\_order\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dwd\_order\_refund\_info | 空值检查 | id | 0 | 10 |
| dwd\_payment\_info | 空值检查 | id | 0 | 10 |
| dwd\_refund\_payment | 空值检查 | id | 0 | 10 |

编写如下脚本：

新建文件check\_dwd.sh

#!/usr/bin/env bash

DT=$1

[ "$DT" ] || DT=$(date -d '-1 day' +%F)

# 检查表 dwd\_order\_detail

bash null\_id.sh -t dwd\_order\_detail -d "$DT" -c id -a 1000 -b 30000 -s 0 -x 100

# 检查表 dwd\_order\_info

bash null\_id.sh -t dwd\_order\_info -d "$DT" -c id -s 0 -x 10

bash duplicate.sh -t dwd\_order\_info -d "$DT" -c id -s 0 -x 5

# 检查表 dwd\_order\_refund\_info

bash null\_id.sh -t dwd\_order\_refund\_info -d "$DT" -c id -s 0 -x 10

# 检查表 dwd\_payment\_info

bash null\_id.sh -t dwd\_payment\_info -d "$DT" -c id -s 0 -x 10

# 检查表 dwd\_refund\_payment

bash null\_id.sh -t dwd\_refund\_payment -d "$DT" -c id -s 0 -x 10

**3. DIM层**

DIM层需要检查的项目如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| dim\_activity\_rule\_info | 空值检查 | active\_id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_coupon\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_sku\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |
| dim\_user\_info | 空值检查 | id | 0 | 10 |
| 重复值检查 | id | 0 | 5 |

编写如下脚本：

新建文件check\_dim.sh

#!/usr/bin/env bash

DT=$1

[ "$DT" ] || DT=$(date -d '-1 day' +%F)

#检查表 activity\_rule\_id

bash duplicate.sh -t dim\_activity\_rule\_info -d "$DT" -c activity\_rule\_id -s 0 -x 5

bash null\_id.sh -t dim\_activity\_rule\_info -d "$DT" -c activity\_rule\_id -s 0 -x 10

#检查表 dim\_coupon\_info

bash duplicate.sh -t dim\_coupon\_info -d "$DT" -c id -s 0 -x 5

bash null\_id.sh -t dim\_coupon\_info -d "$DT" -c id -s 0 -x 10

#检查表 dim\_sku\_info

bash duplicate.sh -t dim\_sku\_info -d "$DT" -c id -s 0 -x 5

bash null\_id.sh -t dim\_sku\_info -d "$DT" -c id -s 0 -x 10

#检查表 dim\_user\_info

bash duplicate.sh -t dim\_user\_info -d "$DT" -c id -s 0 -x 5

bash null\_id.sh -t dim\_user\_info -d "$DT" -c id -s 0 -x 10

### 14.3.5 编写告警脚本

如果数据监控发现有指标异常，将通过邮件或睿象云发出警告：

由于这个功能依赖于mysql-connector-python，需要独立安装组件：



将该rpm包上传虚拟机，并安装：

[atguigu@hadoop102 ~]$ sudo rpm -i mysql-connector-python-2.1.7-1.el7.x86\_64.rpm

并新建python脚本：

新建文件check\_notification.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import mysql.connector

import sys

import smtplib

from email.mime.text import MIMEText

from email.header import Header

import datetime

import urllib

import urllib2

import random

def get\_yesterday():

today = datetime.date.today()

one\_day = datetime.timedelta(days=1)

yesterday = today - one\_day

return str(yesterday)

def one\_alert(line):

"""

:type line: dict

"""

# 睿象云的rest api key

one\_alert\_key = "c2030c9a-7896-426f-bd64-xxxxxxxxxxxx"

one\_alert\_host = "http://api.aiops.com/alert/api/event"

data = {

"app": one\_alert\_key,

"eventType": "trigger",

"eventId": str(random.randint(10000, 99999)),

"alarmName": "".join(["表格", str(line["tbl"]), "数据异常."]),

"alarmContent": "".join(["指标", str(line["norm"]), "值为", str(line["value"]),

", 应为", str(line["value\_min"]), "-", str(line["value\_max"]),

", 参考信息：" + str(line["col"]) if line.get("col") else ""]),

"priority": line["notification\_level"] + 1

}

body = urllib.urlencode(data)

request = urllib2.Request(one\_alert\_host, body)

urlopen = urllib2.urlopen(request).read().decode('utf-8')

print urlopen

def mail\_alert(line):

"""

:type line: dict

"""

# smtp协议发送邮件的必要设置

mail\_host = "smtp.126.com"

mail\_user = "skiinder@126.com"

mail\_pass = "KADEMQZWCPFWZETF"

message = ["".join(["表格", str(line["tbl"]), "数据异常."]),

"".join(["指标", str(line["norm"]), "值为", str(line["value"]),

", 应为", str(line["value\_min"]), "-", str(line["value\_max"]),

", 参考信息：" + str(line["col"]) if line.get("col") else ""])]

sender = mail\_user

receivers = [mail\_user]

mail\_content = MIMEText("".join(["<html>", "<br>".join(message), "</html>"]), "html", "utf-8")

mail\_content["from"] = sender

mail\_content["to"] = receivers[0]

mail\_content["Subject"] = Header(message[0], "utf-8")

try:

smtp = smtplib.SMTP\_SSL()

smtp.connect(mail\_host, 465)

smtp.login(mail\_user, mail\_pass)

content\_as\_string = mail\_content.as\_string()

smtp.sendmail(sender, receivers, content\_as\_string)

except smtplib.SMTPException as e:

print e

def read\_table(table, dt):

# mysql必要参数设置

mysql\_user = "root"

mysql\_password = "000000"

mysql\_host = "hadoop102"

mysql\_schema = "test"

# 从mysql中取数据并转化为字典

connect = mysql.connector.connect(user=mysql\_user, password=mysql\_password, host=mysql\_host, database=mysql\_schema)

cursor = connect.cursor()

query = "desc " + table

cursor.execute(query)

head = map(lambda x: str(x[0]), cursor.fetchall())

query = ("select \* from " + table + " where dt='" + dt + "' and `value` not between value\_min and value\_max")

cursor.execute(query)

cursor\_fetchall = cursor.fetchall()

fetchall = map(lambda x: dict(x), map(lambda x: zip(head, x), cursor\_fetchall))

return fetchall

def main(argv):

# 如果没有传入日期参数，将日期定为昨天

if len(argv) >= 3:

dt = argv[2]

else:

dt = get\_yesterday()

notification\_level = 0

alert = None

if len(argv) >= 2:

alert = {

"mail": mail\_alert,

"one": one\_alert

}[argv[1]]

if not alert:

alert = one\_alert

# 查询所有错误内容，如果大于设定警告等级，就发送警告

for table in ["day\_on\_day", "duplicate", "null\_id", "rng", "std\_dev", "week\_on\_week"]:

for line in read\_table(table, dt):

if line["notification\_level"] >= notification\_level:

line["norm"] = table

alert(line)

if \_\_name\_\_ == "\_\_main\_\_":

# 两个命令行参数

# 第一个为警告类型：one或者mail

# 第二个为日期，留空取昨天

main(sys.argv)

### 14.3.6 Azkaban总调度

最后，通过azkaban框架调度质量检查，需要编写python调度脚本：

**1. 总检查脚本**

新建文件check\_all.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import os

import thread

import time

import urllib

import urllib2

import json

import sys

from check\_notification import get\_yesterday

az\_url = "http://hadoop102:8081/"

az\_username = "atguigu"

az\_password = "atguigu"

project = "gmall"

flow = "gmall"

def post(url, data):

"""

发送post请求到指定网址

:param url: 指定网址

:param data: 请求参数

:return: 请求结果

"""

body = urllib.urlencode(data)

request = urllib2.Request(url, body)

urlopen = urllib2.urlopen(request).read().decode('utf-8')

return json.loads(urlopen)

def get(url, data):

"""

发送get请求到指定网址

:param url: 指定网址

:param data: 请求参数

:return: 请求结果

"""

body = urllib.urlencode(data)

urlopen = urllib2.urlopen(url + body).read().decode('utf-8')

return json.loads(urlopen)

def login():

"""

进行azkaban身份认证，并返回session ID

:return: 返回session\_id

"""

data = {

"action": "login",

"username": az\_username,

"password": az\_password

}

auth = post(az\_url, data)

return str(auth.get(u"session.id"))

def get\_exec\_id(session\_id, project, flow):

"""

获取正在执行的Flow的ExecId

:param session\_id: 和azkaban通讯的session\_id

:param project: 项目名称

:param flow: 工作流名称

:return: 执行ID

"""

data = {

"session.id": session\_id,

"ajax": "getRunning",

"project": project,

"flow": flow

}

execs = get(az\_url + "executor?", data).get(u"execIds")

if execs:

return str(execs[0])

else:

return None

def wait\_node(session\_id, exec\_id, node\_id):

"""

等待指定Flow中的一个节点执行完毕

:param session\_id: 和azkaban通讯的session\_id

:param exec\_id: 执行ID

:param node\_id: 指定节点

:return: 该节点是否成功执行完毕

"""

data = {

"session.id": session\_id,

"ajax": "fetchexecflow",

"execid": exec\_id

}

status = None

while status not in ["SUCCEEDED", "FAILED", "CANCELLED", "SKIPPED"]:

flow\_exec = get(az\_url + "executor?", data)

for node in flow\_exec.get(u"nodes"):

if unicode(node\_id) == node.get(u"id"):

status = str(node.get(u"status"))

print " ".join([node\_id, status])

time.sleep(1)

return status == "SUCCEEDED"

def exec\_flow(session\_id, project\_id, flow\_id):

"""

执行指定Flow

:param session\_id: 和azkaban通讯的session\_id

:param project\_id: 指定的Project

:param flow\_id: 指定的Flow

:return: 执行ID

"""

data = {

"session.id": session\_id,

"ajax": "executeFlow",

"project": project\_id,

"flow": flow\_id,

}

execs = post(az\_url + "executor?", data)

return str(execs.get(u"execid"))

def check\_ods(dt, session\_id, exec\_id):

"""

检查ODS层数据质量

:param dt: 日期

:param session\_id: 和azkaban通讯的session\_id

:param exec\_id: 指定的执行ID

:return: None

"""

if wait\_node(session\_id, exec\_id, "hdfs\_to\_ods\_db") and wait\_node(session\_id, exec\_id, "hdfs\_to\_ods\_log"):

os.system("bash check\_ods.sh " + dt)

def check\_dim(dt, session\_id, exec\_id):

"""

检查DIM层数据质量

:param dt: 日期

:param session\_id: 和azkaban通讯的session\_id

:param exec\_id: 指定的执行ID

:return: None

"""

if wait\_node(session\_id, exec\_id, "ods\_to\_dim\_db"):

os.system("bash check\_dim.sh " + dt)

def check\_dwd(dt, session\_id, exec\_id):

"""

检查DWD层数据质量

:param dt: 日期

:param session\_id: 和azkaban通讯的session\_id

:param exec\_id: 指定的执行ID

:return: None

"""

if wait\_node(session\_id, exec\_id, "ods\_to\_dwd\_db") and wait\_node(session\_id, exec\_id, "ods\_to\_dwd\_log"):

os.system("bash check\_dwd.sh " + dt)

if \_\_name\_\_ == '\_\_main\_\_':

argv = sys.argv

# 获取session\_id

session\_id = login()

# 获取执行ID。只有在原Flow正在执行时才能获取

exec\_id = get\_exec\_id(session\_id, project, flow)

# 获取日期，如果不存在取昨天

if len(argv) >= 2:

dt = argv[1]

else:

dt = get\_yesterday()

# 检查各层数据质量

if exec\_id:

check\_ods(dt, session\_id, exec\_id)

check\_dim(dt, session\_id, exec\_id)

check\_dwd(dt, session\_id, exec\_id)

**2. ODS层调度脚本**

新建文件check\_ods.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import sys

import check\_all

if \_\_name\_\_ == '\_\_main\_\_':

argv = sys.argv

# 获取session\_id

session\_id = check\_all.login()

# 获取执行ID。只有在原Flow正在执行时才能获取

exec\_id = check\_all.get\_exec\_id(session\_id, check\_all.project, check\_all.flow)

# 获取日期，如果不存在取昨天

if len(argv) >= 2:

dt = argv[1]

else:

dt = check\_all.get\_yesterday()

# 检查各层数据质量

if exec\_id:

check\_all.check\_ods(dt, session\_id, exec\_id)

**3. DWD层调度脚本**

新建文件check\_dwd.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import sys

import check\_all

if \_\_name\_\_ == '\_\_main\_\_':

argv = sys.argv

# 获取session\_id

session\_id = check\_all.login()

# 获取执行ID。只有在原Flow正在执行时才能获取

exec\_id = check\_all.get\_exec\_id(session\_id, check\_all.project, check\_all.flow)

# 获取日期，如果不存在取昨天

if len(argv) >= 2:

dt = argv[1]

else:

dt = check\_all.get\_yesterday()

# 检查各层数据质量

if exec\_id:

check\_all.check\_dwd(dt, session\_id, exec\_id)

**4. DIM层调度脚本**

新建文件check\_dim.py

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import sys

import check\_all

if \_\_name\_\_ == '\_\_main\_\_':

argv = sys.argv

# 获取session\_id

session\_id = check\_all.login()

# 获取执行ID。只有在原Flow正在执行时才能获取

exec\_id = check\_all.get\_exec\_id(session\_id, check\_all.project, check\_all.flow)

# 获取日期，如果不存在取昨天

if len(argv) >= 2:

dt = argv[1]

else:

dt = check\_all.get\_yesterday()

# 检查各层数据质量

if exec\_id:

check\_all.check\_dim(dt, session\_id, exec\_id)

**5. Azkaban工作流文件**

新建文件azkaban.project

azkaban-flow-version: 2.0

新建文件data\_supervisor.flow

nodes:

- name: check\_ods

type: command

config:

command: python check\_ods.py ${dt}

- name: check\_dwd

type: command

config:

command: python check\_dwd.py ${dt}

- name: check\_dim

type: command

config:

command: python check\_dim.py ${dt}

- name: check\_notification

type: command

dependsOn:

- check\_ods

- check\_dwd

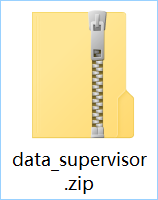
- check\_dim

config:

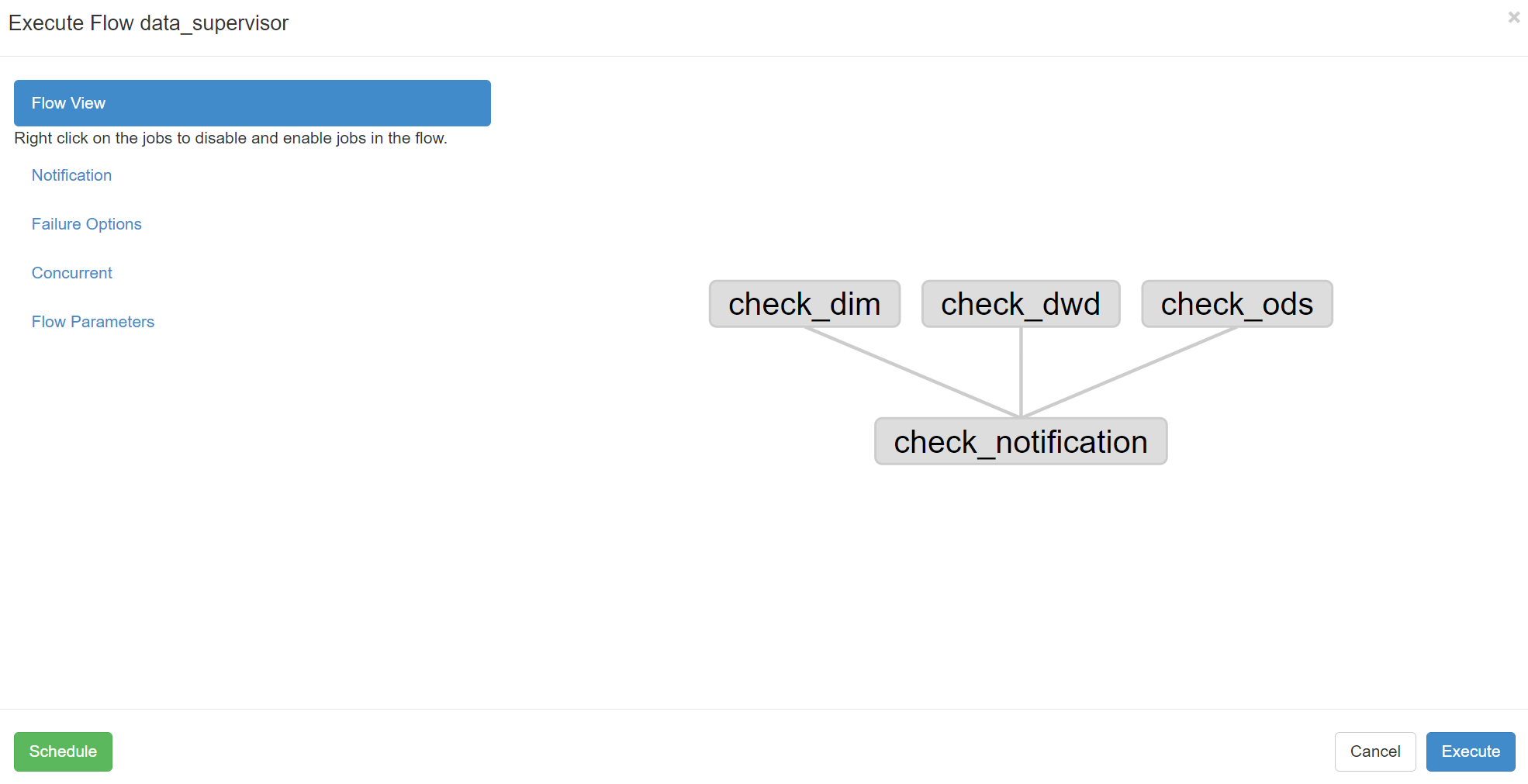
command: python check\_notification.py ${alert} ${dt}

将所有文件打包成data\_supervisor.zip文件:

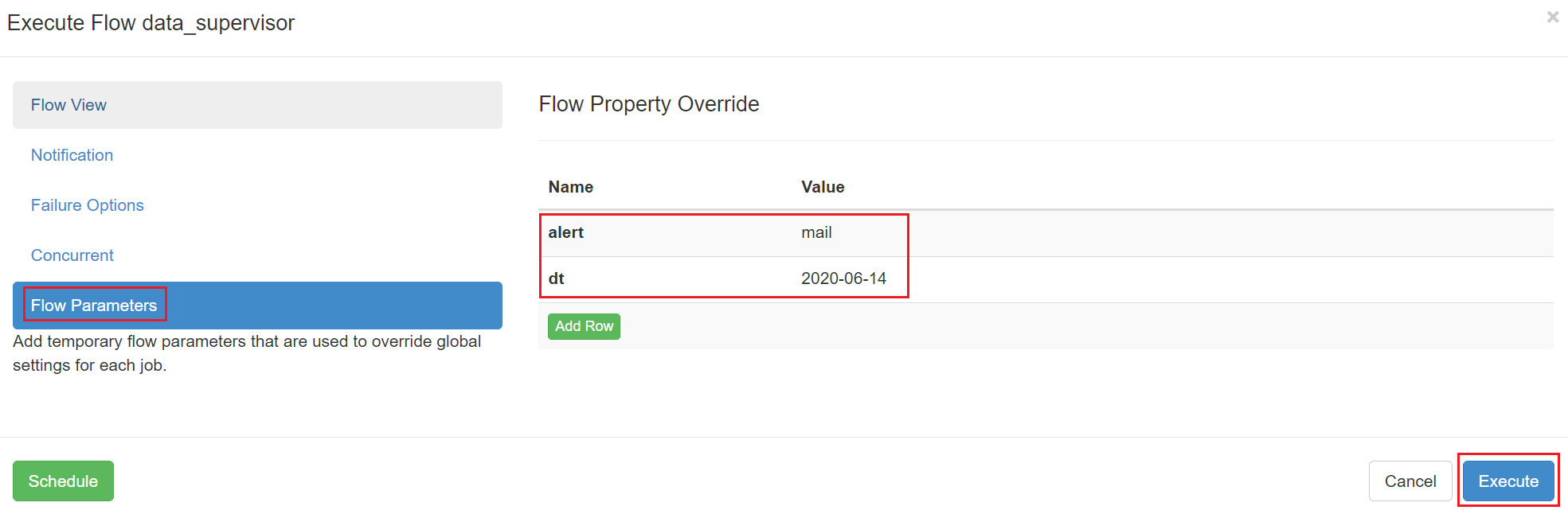




在azkaban框架中新建项目并上传该文件，看到如下工作流：



先启动数仓调度流程，在执行过程中，启动质量监控工作流：



等待任务执行完毕，观察邮箱是否有告警邮件

## 14.4 本章总结