# 使用指南

## 工具配置与运行逻辑

### 数据解析流程

数据整合脚本流程主要分两大部分：一部分为数据源获取，生成数据矩阵与中间文件；另一部分为对生成的数据矩阵进行整合、筛选、统计等操作，生成目标数据矩阵与中间文件。

包括了数据获取、数据统计、数据筛选、数据合并、数据输出这几个操作模块，脚本全部使用矩阵计算的方式进行。基本流程呈树状结构，最终收敛到目标文件



### 脚本执行方式

引入库文件DataLib，修改main.py文件配置。在文件底部执行函数可配置中间文件以及Excel目标源的路径，不设默认为main.py同级路径 ，以及是否打印中间矩阵到控制台：

pDeal = DataDealModel(<节点文件输出目录>, needPrint = <是否打印数据True|False>)

pDeal.runPrepare(prepareData)

pDeal.runCompile(formatData)

修改prepareData与formatData两个数组配置文件，命令行执行python main.py即会输出目标文件。

Python库依赖：

较新版本的lxml，（linux下安装需要依赖gcc库，可命令行执行：apt-get install libxml2-dev libxslt1-dev python-dev），

MySQLdb 用以python读取数据库，可通过ez\_install源码安装，地址：

https://sourceforge.net/projects/mysql-python/files/mysql-python-test/

numpy、pandas 数据分析处理库， pip install 方法安装即可。

openpyxl, xlrd Excel读写操作库， pip install 方法安装即可，升级到较新版本

### 脚本配置逻辑

脚本配置的逻辑结构为：

1. 数据库连接配置

sql\_config = dict({ # dict为python字典类型，相当于json对象

'host': '127.0.0.1',

'user': 'root',

'pwd': '',

'db': 'tushare',

'port': 3306

})

1. 目标文件源配置，支持csv/mysql

prepareData = [

{ 'target': <目标节点名>, # 最终生成内存的DataFrame与以此命名的.xlsx

'sql\_db': <数据库连接配置>,

'from\_type': ‘mysql’,

'query': <数据库query语句> }, # 此为数据库源

{ 'target': <目标节点名>,

'sheet\_name': <Excel表格名>,

'from\_type': 'excel' }, …… # 此为文档源

]

1. 中间文件处理，包括数据合并、数据筛选、数据统计

formatData = [

{ 'target': <中间目标节点名>, # 生成DataFrame与以此命名的.xlsx

'method': { # 多个源数据合并方式

'type': <合并方式>,

'how': <left|right|outer|inner>,

'relation': [<列名>,<列名>,<列名1==列名2>]

},

'src': [

[ <第一个数据源>, { # 数据源默认取DataFrame，不存在取文件

# 对源节点进行数据统计与数据筛选

'statistic': [<数据列名|统计函数>,'volume|mean', 'E|sum' …… ],

'filter': [<筛选方法:{筛选参数}', 'sort:{[B,C],[1,0]}', ……]

}],

[ <第二个数据源>, {

'statistic': [<数据列名|统计函数>,<数据列名|统计函数>, …… ],

'filter': [ <筛选方法:{筛选参数}', <筛选方法:{筛选参数}', ……]

}], ……

}, {<多个数据处理方法顺序执行>}, { …… } ……

]

采用此方法的优势为数据筛选逻辑与数据统计逻辑高度封装、逻辑清晰、数据源多样化（跨库跨表跨源类型，如mysql、excel、csv，还可支持redis、sqlite等等）。

该方法每进行一般数据筛选与合并，均会生成中间文件（excel格式），方便调试，且出错或需要调整，均可基于原准确中间数据进行二次操作，类似于下载文件的断点续传，也便于调试。

d. 目标结果生成

示例如下： 目前支持mysql与excel两种。可设置from\_type说明，也可缺省，工具会自动识别sql\_db自动确定是存储到mysql还是excel

saveData = [

{ 'target': 'test\_4',

'src': 'test\_1',

'sql\_db': nim\_test2,

'unique\_key': 'A',

'if\_exists': ‘replace’

},

{ 'target': 'test\_4',

'src': 'test\_2',

'sql\_db': nim\_test2,

'unique\_key': 'A',

'if\_exists': 'append'

},

{ 'target': 'test\_4',

'src': 'test\_3',

'if\_exists': 'replace'

},

]

具体会在数据存储章节详细说明。

## 数据源获取：

数据源获取执行的是pDeal.runPrepare(prepareData)函数，prepareData为配置节点数组（配置为json格式）

目前支持mysql、excel(xlsx格式)、csv三种数据格式，在main.py文件中prepareList中添加对应的数据源json信息，执行脚本即可获取并生成数据矩阵与中间文件。

内存中的数据矩阵形如：

date open high close low volume

0 2015-09-25 15.15 15.62 15.59 15.15 136070896

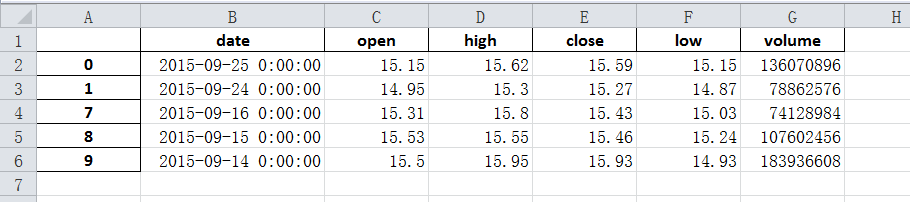
1 2015-09-24 14.95 15.30 15.27 14.87 78862576

7 2015-09-16 15.31 15.80 15.43 15.03 74128984

8 2015-09-15 15.53 15.55 15.46 15.24 107602456

9 2015-09-14 15.50 15.95 15.93 14.93 183936608

对应的文件节点则为：



### Mysql数据获取：

由于有多个数据源节点是对同一个数据库进行操作的情况，为方便起见，需要为每一个数据库对应配置变量。对数据库进行数据获取时，需要传入该配置变量与SQL的query语句，结果会返回一个数据矩阵，并存储到一个excel文件。

#### 数据库基本配置：

首先配置数据库信息，例如：

sqlCfg = dict({

'host': '127.0.0.1',

'user': 'root',

'pwd': '123456',

'db': 'dev\_netease\_im',

'port': 3306

})

#### 数据库节点矩阵生成

然后在prepareList数组中添加数据节点，例如：

{ 'target': 'docNode', # 此为数据节点目标名称，用于标记矩阵，生成excel

'from\_type': 'mysql', # 若配置了sql\_db，此项默认为mysql，可不填

'sql\_db': sqlCfg, # 数据库配置信息

'query': 'select \* from doc\_content where doc\_id < 100' # 需要查询的sql语句

},

脚本执行中，会从数据库中query出相应信息，生成数据矩阵，并将副本进行保存（副本可继续作为源节点多次操作），副本格式文件为excel，可通过源程序修改为csv等，命名为<target>.xlsx

上述配置可简写为：

{ 'target': 'docNode',

'sql\_db': sqlCfg,

'query': 'select \* from doc\_content where doc\_id < 100'

},

### Excel数据获取:

目前支持csv与excel两种文件格式的获取，对应低配置python环境可以使用csv，高配置python环境建议使用excel，因为csv的逗号分隔方式会出现符号混淆产生的矩阵错乱。

#### 文档数据节点矩阵生成：

相比读取mysql，读取excel源数据显得更为简单，配置如下：

csv文件读取：

{ 'target': 'doc\_con\_csv',

'from\_type': 'csv' }

xlsx文件读取

{ 'target': 'doc', # 读取文件源为<target>.xlsx

'sheet\_name': 'Sheet1', # 读取的excel文件的sheet，可不填，默认Sheet1

'from\_type': 'excel' } # 类型为excel、或csv，可不填，默认excel

如果是xlsx文件，且默认sheet，则可简写为：

{ 'target': 'doc', }

读取之后，脚本会在内存中建立命名空间为doc的数据矩阵

## 数据解析：

数据矩阵分析执行的是pDeal.runCompile(formatData)函数，runCompile为解析节点数组（配置为json格式），解析节点格式形为：

{ 'target': 'tarNode', # 解析完成后生成的目标节点，同时产生tarNode.xlsx副本

'method': { # src 节点列表数据矩阵合并方式

'type': 'merge', # 合并类型，目前支持merge与concat

'how': 'outer', # 合并模式，inner/outer/left/right

'relation': ['A','B==F'] # 合并关联项，将t1.A与t2.B 作为基列

},

'src': [

['test\_1',{'filter': ['in:{A:[1, 3, 5]}']} ], # 数据源，在内存中或xlsx中

['test\_2', {'statistic': ['A|mean','E|sum'], 'filter': ['A > 2', ' E>{A|mean} ']} ]

] # 对每一个数据矩阵，先执行数据统计，再执行数据过滤

}

statistic 为数据统计标识，filter为数据行过滤标识

### 数据矩阵合并

对于不同的数据源，无论是excel还是mysql不同数据库不同数据表，都会生成数据矩阵。数据处理经常需要对不同类型的矩阵数据进行合并（类似于对跨库跨表查询的数据整合）

当配置中，src列表项的数据源多于一个时，就需要进行数据矩阵合并操作。

#### concat操作（待完善）

concat操作目前仅适用于同类型矩阵数据的行拼接，参数配置例如：

'method': {

'type': 'concat',

'how': 'outer', # inner/outer，inner类型取交集，outer取并集（有重复）

},

源：

Src1:

A F C M G

7 9 1 4 1 3

6 8 2 8 2 1

5 7 3 5 3 3

Src2:

A F C M G

7 9 1 4 1 3

4 6 3 7 5 6

5 7 3 5 3 3

目标：

Target（inner）

A F C M G

1 9 1 4 1 3

2 7 3 5 3 3

Target（outer）

A F C M G

1 9 1 4 1 3

2 8 2 8 2 1

3 7 3 5 3 3

4 9 1 4 1 3

5 6 3 7 5 6

6 7 3 5 3 3

#### merge操作

merge操作较为复杂，可以对不同索引，不同行数的矩阵进行合并，并可指定合并的基准，前者类似于数据库跨表操作中的left join、right join（此外还可取交集、并集），后者类似于跨表比较判断如t1.A = t2.B。相比concat方法，merge会去重

配置如：

'method': {

'type': 'merge', # 此项若不填则为merge

'how': 'outer', # inner/outer/left/right，交集并集内联外联

'relation': ['A','B==F'] # 列名合并标准，若没再此列出的重名列，

# 会被重命名，’B==F’为t1矩阵的B列与t2矩阵

# 的F列进行关联，并将结果矩阵的列名命为B

},

假设源矩阵：

Src1:

A B C D G

1 3 2 9 7 8

3 5 2 8 5 5

Src2:

A E F D

3 4 1.000000 7 4

4 5 3.000000 5 7

6 7 1.000000 4 3

8 8 4.166667 2 3

9 6 2.000000 1 4

合并配置： {'how': 'outer', 'type': 'merge', 'relation': ['A', 'D']}

A B C D G E F

0 3.0 2.0 9.0 7.0 8.0 NaN NaN

1 5.0 2.0 8.0 5.0 5.0 NaN NaN

2 4.0 NaN NaN 4.0 NaN 1.000000 7.0

3 5.0 NaN NaN 7.0 NaN 3.000000 5.0

4 7.0 NaN NaN 3.0 NaN 1.000000 4.0

5 8.0 NaN NaN 3.0 NaN 4.166667 2.0

6 6.0 NaN NaN 4.0 NaN 2.000000 1.0

合并配置： {'how': 'outer', 'type': 'merge', 'relation': ['A']}

仅以A为基准进行合并，由于两个源都有D，则会重命名加以区别

A B C D\_x G E F D\_y

0 3.0 2.0 9.0 7.0 8.0 NaN NaN NaN

1 5.0 2.0 8.0 5.0 5.0 3.000000 5.0 7.0 # 重合项

2 4.0 NaN NaN NaN NaN 1.000000 7.0 4.0

3 7.0 NaN NaN NaN NaN 1.000000 4.0 3.0

4 8.0 NaN NaN NaN NaN 4.166667 2.0 3.0

5 6.0 NaN NaN NaN NaN 2.000000 1.0 4.0

合并配置： {'how': 'inner', 'type': 'merge', 'relation': ['A']}

A B C D\_x G E F D\_y

0 5 2 8 5 5 3.0 5 7

合并配置： {'how': 'inner', 'type': 'merge', 'relation': ['A', 'D']}

则需要满足两个矩阵的A列与D列同时相同才会输出，故此处为空矩阵

合并配置： {'how': 'outer', 'type': 'merge', 'relation': ['A', 'B==F']}

此处除了将A作为基准外，还将src2的F列与src1的B列作为基准合并，B列数据无缺失。

A B C D\_x G E D\_y

0 3.0 2.0 9.0 7.0 8.0 NaN NaN

1 5.0 2.0 8.0 5.0 5.0 NaN NaN

2 4.0 7.0 NaN NaN NaN 1.000000 4.0

3 5.0 5.0 NaN NaN NaN 3.000000 7.0

4 7.0 4.0 NaN NaN NaN 1.000000 3.0

5 8.0 2.0 NaN NaN NaN 4.166667 3.0

6 6.0 1.0 NaN NaN NaN 2.000000 4.0 # 原为src2的F列数据

合并配置： {'how': 'left', 'type': 'merge', 'relation': ['A']}

此处类似于left join，即以src1数据为基准，将src2数据在src1基础上做填充

A B C D\_x G E F D\_y

0 3 2 9 7 8 NaN NaN NaN

1 5 2 8 5 5 3.0 5.0 7.0

### 数据统计

工具的数据统计较为方便，即在statistic对象中输入需要统计的数据的数组，以 | 进行分割，[‘<列名>|<统计方法名>’, ‘<列名>|<统计方法名>’]，不输入则不做统计。

统计结果可作为中间比较变量传入数据筛选器，传入方式使用花括号 {<列名>|<统计方法名>}，例如筛选出所有colume列中大于均值的行，B>{A|mean}'

如源矩阵：

A B C D G

1 3 2 9 7 8

3 5 6 8 5 5

配置项为：

[‘target’, {

'statistic': [ ‘A|mean'],

'filter': [ ‘B>{A|mean}' ]

} ]

控制台打印结果为：

\*\* deal with src - 1 : from test\_2

== start statistic 1 ==

!! A|mean - result: 4

== start filter 1 ==

volume>{volume|mean} : succeed

!! Temp DataFrame Src 1

A B C D G

3 3 6 9 7 8

#### 数据统计函数表：

|  |  |  |  |
| --- | --- | --- | --- |
| 统计方法 | 配置格式 | 说明 | 样例 |
| 均值mean | <列名|mean> |  | ‘key|mean’ |
| 项数count | <列名|count > | 返回非Nan的项数 | ‘key|count’ |
| 平均绝对偏差mad | <列名| mad> | 类似于标准差 | ‘key|mad’ |
| 中位数median | <列名| median > | 等价于50%位的数 | ‘key| median’ |
| 最大值max | <列名| max> |  | ‘key|max’ |
| 最小值min | <列名| min> |  | ‘key|min’ |
| 方差var | <列名|var > |  | ‘key|var’ |
| 标准差std | <列名| std> | 方差平方根 | ‘key|std’ |
| 总数sum | <列名| sum> |  | ‘key|sum’ |
| 众数mode | <列名|mode > | 出现最多的变量值 | ‘key|mode’ |

### 数据筛选器

数据筛选器即为配置中filter中的方法，标准格式为：’<方法名>:{方法参数}’，若进行日期时间比较与简单数值比较(<|=|>)，则可省略输入<方法名>，直接输入比较方法。

配置例如：

'filter': ['shape:{ drop :[amount]}', 'timefunc :{date>{2015-9-11 19:01:01}}', 'name ==”昵称”',

'date<{2015-9-30}', 'volume>={volume|mean}', 'sort:{[B,C],[1,0]}' ] } ]

过滤器在数组中进行级联，后一次过滤是在前一次过滤结果中进行的。如果是字符串比较，则需要对字符串添加双引号。

#### 过滤器列表

##### Operator （普通行比较筛选）

Operator 方法为最常用数值比较方法，标准写法为：

'operator :{<列名> <比较操作符><比较值>}'

返回为满足比较条件的行，可使用数据统计的结果。

可去掉operator操作符，简写为'<列名> <比较操作符><比较值>'

例如：

'A > 2', ' E<{A|mean} ', ' C=”colume” ', 'operator:{A >= 2}'

##### Timefunc （时间行比较筛选）

时间函数比较器，用于比较时间，可以比较的时间类型格式包括yyyy-MM-dd hh:mm:ss、yyyy-MM-dd、long类型的时间戳三种，脚本会自动进行数据格式转换，标准写法为：

'timefunc :{<列名> <比较操作符><比较值>}'

返回为满足比较条件的行，可使用数据统计的结果。

可去掉timefunc操作符，简写为'<列名> <比较操作符><比较值>'

例如：

'timefunc :{date>{2015-9-11 19:01:01}}', 'date<{2015-9-30}'

##### In （指定值匹配的行）

In操作符，类似于mysql用法，用于选择在列表中已列举目标的行数，标准写法为：

'in:{<列名>:[<变量>, <变量>, <变量>…]}'

例如：

'in:{A:[1, 3, 5]}'

返回A列中，只在该数组中出现值的行，例如：

Src:

A B C D G

1. 3 2 9 7 8
2. 6 1 3 1 4
3. 5 2 8 5 5

返回结果：

A B C D G

1 3 2 9 7 8

3 5 2 8 5 5

虽然原来的矩阵A列中并没有数值1，但并不会报错

##### Iloc （指定行）

Iloc函数用于返回用户所指定的行，例如第3-10行，第3、5、8行的结果。标准写法：

iloc:{<序号起始>:<序号终止>}， 或iloc:{<序号1>, <序号2>,<序号3>}

例如：

iloc:{2:6}

##### Sort （按列排序）

Sort函数用于对目标列进行排序，类似于mysql的order by，excel中的列升序降序，标准写法：

'sort:{ [ <列名>,<列名>, … ], [<1|0>, <1|0>, … ] }'

传入两个数组，第一个数组为需要排序的列，排序越前，权重越大；第二个数组为对应的排序规则，1表示按值升序，0表示按值降序，可不传默认均为升序，若传该参数，则必须保证数组维数与列名数组相同。例如：

'sort:{[B, C], [1, 0]}', 'sort:{[A, C]}'

##### Shape（列筛选，left or drop）

对数据矩阵的列进行筛选，可以选择留下目标列或删除目标列，标准写法

'shape:{ <drop|left> :[ <列名>,<列名>, … ]}'

选项drop，表示需要对列表中的列名进行删除后返回数据

选项left，表示仅保留列表中的列。例如：

'shape:{ drop :[A,B]}', 'shape:{left:[D\_y,C,D\_x]}'

表示先删除A,B列，最后却只保留D\_y,D\_x，输入不存在的列，方法会被略过

##### Rename （列重命名）

对列名进行重命名，例如将A命名为B，标准写法为：

'rename:{ <列名>-> <列名>, <列名>-> <列名> … }'

例如：

'rename:{D->M,B->F}'

方法对数组中序列采用依次进行的方式，若类如

'rename:{D->M,M->F}'

则实际只是对D列名改为F而已。

##### OperatorCol （列操作运算）

对列进行操作运算，并将运算结果赋值到新的列，标准写法为：

'operatorCol: { <新列名> = <列运算操作> }'

例如：

'operatorCol:{ M = - C \* (D\_x + 2) - D\_x}'

'operatorCol:{ N = (date2 - date1)/3600.0/24.0 }'

注意，若为时间格式的列进行运算，会事先转化为以秒为单位的时间戳，最后返回的结果也是以秒为单位，用户可根据需求，将时间单位进行转化。

##### rawFunc （使用pandas原生函数进行操作）

对于基本函数不能解决的问题，额外添加了使用原生pandas库函数操作的方法，中间变量为df，并将最终运算结果返回给df。标准写法为：

‘rawFunc: {df相关运算}'

例如：

‘rawFunc:{ df.dropna() }’ //去除空白单元格

‘rawFunc:{ df[‘A’].duplicate() }’ //去除重复元素

## 四、数据存储

本工具可实现双向数据存储，即excel ⬄ excel， excel ⬄ mysql，mysql⬄ excel，mysql ⬄ mysql，由于所有中间数据结构均是pandas的DataFrame，这类的绑定是可行的。

### excel 存储

如果在pDeal = DataDealModel('nim/', needPrint = False, needInterFile = True) 配置中选择needInterFile（中间交叉文件）为True，则在compile过程中，会自动生成excel文件，文件名为节点名（target）+ .xlsx命名。

如果needInterFile = False，所有的中间结果均在内存中，需要显式确定存储格式，例如：

{ 'target': 'test\_4', # 存储的目标文件名

'src': 'test\_1', # 源节点名，可在内存中，也可读取excel

'from\_type': ‘excel’, # 可缺省，表示目标为excel

'if\_exists': 'append' # 可选replace/fail ，

},

If\_exists字段表示是否替换为新的文件，replace表示存在则替换，fail表示如发现相应文件则放弃存储

### mysql存储

mysql存储示例配置如下：

{ 'target': <table\_name>, # 目标存储的表名字，

'src': <节点名>, # 需要存储的节点，为pandas dataframe矩阵

'sql\_db': <数据库配置>, # 数据库配置对象，参考脚本配置逻辑

'unique\_key': <unique key>, # <是否存在unique key> 主要用于去重

'if\_exists': 'append' # 追加类型，有append，replace， fail， append\_ignore

‘need\_datetime’: True/False # 是否强制生成created\_at updated\_at字段

},

如果表不存在，工具会自动创建对应数据表，根据dataframe的列名即数据类型创建数据表，

如果数据库配置不存在unique\_key，全部采用pandas默认的to\_sql方法，不会做去重管理，如果存在duplicate key，可能出现存储失败。

如果指定了unique\_key，创建的数据表会以对应的unique\_key设置unique key，自动创建名为id的primary key。

如果指定了need\_datetime，工具会自动先查找待存储数据中是否存在created\_at与updated\_at字段，如果存在则不做处理，如果不存在，会强制建立这两个字段的表，类型为datetime（需要创建表的情形）

if\_exists 表示如果表存在的情况下的处理状况，如果为fail则不做任何操作，replace表示会把原有表删除重建，并添加数据，如果为append或append\_ignore，则会追加数据，区别为，append\_ignore会忽略duplicate，而append会执行update（不包括created\_at）