The description of auto object adder variable

Objective

命令行工具可以通过–infopath –infovar 来确定宾语自动化的格式。这里提供一份简要说明。格式包括自动化处理的变量和设置。

config.json

形如

```
{
    "language" : "ch" // 仅允许"ch"和"eg"，命令返回语言
}
```

The variable

格式变量是一个字典，键为 info 名字，值为一个字典，表明该 info 所要格式化输出的方式。之后会详细介绍该字典的信息。

该字典表明了 info 格式化输出的方式，不同的键表示不同的属性和不同类型的值。以下是所有有效的键值对。标题为键。

例如：

```
auto_func_arg = {
    "city_info":{}
}
```

info_args

类型为 OrderedDict() 变量，是一个有顺序的字典。包含了 info 宾语中所有可能出现的属性（键）和值的类型（值）。

例如：

```
from collections import OrderedDict
city_info_args_dict = OrderedDict()
city_info_args_dict[INFOKEY.prefix] = str
city_info_args_dict[INFOKEY.idprefix] = str
city_info_args_dict[INFOKEY.detectReset] = str
city_info_args_dict[INFOKEY.addWarmup] = str
city_info_args_dict[INFOKEY.addReset] = str
city_info_args_dict[INFOKEY.unit] = str
```

之后将其放入变量中。(之后同理，不再提示)

例如:

```
auto_func_arg = {
    "city_info":{"info_args": city_info_args_dict}
}
```

Type of property

```
str #字符串，将按照原字符串读入
bool #bool值，将原字符串转为bool值
(list, str) #字符串列表，将原字符串按照","分割为字符串列表
(list, int) #整数列表，将原字符串按照" "分割后翻译为数字
(list, list, str) #字符串二维列表，将原字符串先按照";"再按照","分割形成字符串二维列表
(list, list, int) #整数二维列表，将原字符串先按照","再按照" "分割形成整数二维列表
```

default_args

类型是 dict，无顺序的字典。包含了 info 宾语中所有属性（键）和默认值。如果 info 宾语未提供该键值对，会自动添加。

例如:

```
city_info_default_args_dict = {
    "unitAddname": "",
    "unitAddoffset": "0 0",
    "unitAddoffsetsize": "0 0",

    "unitDetectname": "检测 {idprefix0}",
    "unitDetectoffset": "-10 0",
    "unitDetectoffsetsize": "20 0"
}
```

initial_brace

类型是 dict，无顺序的字典。包含了 info 宾语中属性（键）和表达式（值）。该默认操作会在 info 宾语将属性、外部引用属性载入后进行，会直接计算该表达式。

例如:

```
city_info_default_args_dict = {
    "lensetidTeam": "len(setidTeam)"#初始计算setidTeam的长度
```

```
}
```

default_brace

类型是 set，集合。包含了 info 宾语中的属性（键）。如果默认参数在这个集合中，那么默认值代入时将会执行表达式(初始表达式计算完成)，否则将会执行地图属性导入。

例如：

```
city_info_default_brace_set = {
    "teamDetectname",
    "teamDetectoffset",
    "teamDetectoffsetsize",
    "teamTextname",
    "teamTextoffset",
    "teamTextoffsetsize"
}
```

var_dependent

类型为 dict(str, str)。info 宾语可选参数独立性检查。键依赖于值，值 (bool) 存在且不为 False。值可以是被”，“隔开的不同参数。独立性类型检查。

optional

类型为 set。info 宾语可选参数列表。存在性类型检查。

no_check

类型为 bool。存在且为 True。独立性类型检查和必要性类型检测仍然存在，但放松参数导入要求。

prefix

类型是 str，字符串。包含了 info 宾语中的属性（键）。标志宾语中对应的值决定识别标志宾语的前缀 (str)。

seg

类型是 str，字符串。标志宾语参数分割字符串。

isprefixseg

类型是 str，字符串。包含了 info 宾语中的属性（键）。标志宾语中对应的值决定前缀之后的第一个固定参数是否使用 seg 分割 (bool)。

args

类型为 list(tuple(str, type))。标志宾语需要导入的参数及其类型。

例如：

```
city_info_args = [
    ("cityname", str)
]
```

opargs_seg

类型是 str，字符串。标志宾语可选参数分割字符串。

opargs_prefix_len

类型是 int。标志宾语可选参数前缀长度。

opargs

类型为 dict(str, tuple(str, type))。标志宾语可选参数前缀、标志宾语可选参数 | 默认值、类型。

例如：

```
city_info_opargs = {
    "t": ("team|-1", str)
}
```

当标记宾语中实际可选参数为”None”时，标记宾语效果为无。

cite_name

如果标志宾语最终的 operation 后字典中参数包含 cite_name，则该参数表明该宾语的引用标志。cite 引用要求：引用标志. 参数名称。使用的字典是完成了最终的 operation 后的字典。在任意位置可以使用。必须是在文件后面的标志宾语引用前面的标志宾语。

info_prefix

类型为 dict(str, str)。键为 info 名字，值为该 info 引用的 prefix。保证将对应 info 宾语所有键值对引用过来，进行默认参数输入、初始默认表达式计算，但不进行初始操作。默认参数、默认表达式、初始表达式优先当前信息宾语。

例如：

```
city_info_info_prefix = {
    "inadd_info": "inadd_prefix",
    "text_info": "text_prefix",
    "teamDetect_info": "teamDetect_prefix",
    "teamText_info": "teamText_prefix"
}
```

isinfo_sub

类型为 bool。是否仅供作为其他 info 的引用。

ids

类型为 list(list(str, int))。info 宾语中添加的 id 前缀，及对应 id 数目。如果在后续 operation_pre 中添加 ids，请进行初始化，至少设为 []

例如：

```
city_info_ids = [
    ["idprefix", 1]
]
```

isnot_cite_check

如果存在且为 True。那么引用过滤将会取消。引用过滤即仅允许以下部分可被引用。args opargs brace ids 产生的 id

is_cite_white_list

如果存在，那么将会特别地允许内部的参数可以引用。

operation_pre / operation

类型为 list(dict)。表明一系列操作。operation_pre 是 info 宾语执行的 operation 序列。operation 是 标记宾语执行的 operation 序列。以下将说明所有操作及适应范围。所有 operation 字典必须有一个键为 operation_type，该键值对决定了 operation 的类型，接下来将说明所有 operation_type 的作用及其他参数。

tag

标记一处位置可供跳转

参数：tag:str

goto

跳转至 goto_tag 的位置。会进行字符串翻译，再跳转。

参数：goto_tag:str

typeif

如果 ifvar 表达式计算结果类型为 str，或值为 False。跳转至 ifend_tag。会进行字符串翻译，再跳转。

参数：ifvar ifend_tag:str

typedelete

将其他键值对的值全部进行表达式计算，导入。键值对中的键将进行字符串翻译。

参数：其他参数

typeset_expression

将其他键值对的值全部进行表达式计算，导入。键值对中的键将进行字符串翻译。

参数：depth：值的最大翻译深度，不填默认 1024 层。brace_exp_depth: 值到下一次字符串翻译的翻译深度，不填默认 1024 层。其他参数

changetype

将 keyname_list 中的所有键的值（在当前字典中）转换为 totype 类型。键值对中的键将进行字符串翻译。

参数：totype keyname_list:list

typeset_exist

确定值是否在当前字典中作为键存在，将 bool 结果赋给键并导入。键值对中的键将进行字符串翻译。

参数：其他参数

error

认为当前错误，将错误信息返回。返回会进行字符串翻译。参数：error_info:str

pdb_pause

pdb 调试暂停 参数：ID:str 可选，ID 要求 name:str 可选，name 前缀要求 print:str 可选，输出内容，否则输出 object_dict

typeset_id(only operation_pre)

其他参数作为键值对导入 ids。键做字符串翻译，值做表达式。不过真实前缀是读取 real_idexp 中的内容（做字符串翻译）。

参数：其他参数（仅一个）real_idexp

typeadd_args(only operation_pre)

其他参数作为键值对导入 args。键做字符串翻译，值做表达式翻译，值再对应翻译为类型。表达式翻译不会进一步翻译 {} 了。

参数：其他参数

typeadd_opargs(only operation_pre)

其他参数作为键值对导入 opargs。键做字符串翻译，值做表达式翻译，再将值第二项翻译为类型。表达式翻译不会进一步翻译 {} 了。

参数：其他参数

typedelete_optional(only operation_pre)

进行表达式翻译后将为 list，并将所有内容从 optional 中除去

参数：namedelete_optional

typeadd_optional(only operation_pre)

进行表达式翻译后将为 list，并将所有内容加入 optional

参数：nameadd_optional

object(only operation)

标记宾语要添加一个宾语。接下来会提供详细格式要求。

参数：

exist:list 所有参数必须存在且为 true，才会添加宾语

death:list 所有参数必须不存在或者为 false，才会添加宾语

offset: 表达式计算，前两项为宾语 xy 偏移

offsetsize：表达式计算，前两项为宾语大小 xy 偏移

name: 字符串翻译后为宾语名称。如果为 tuple，第三项为 brace 时，第二项表达式为 True 允许产生该条目；第三项为 exist 时，exist 内所有条目（用”，“分割，分割后进行字符串翻译）均必须存在且不为’ None’，允许产生该条目。结果为空时不产生。

type: 字符串翻译后为宾语类型。如果为 tuple，第三项为 brace 时，第二项表达式为 True 允许产生该条目；第三项为 exist 时，exist 内所有条目（用”，“分割，分割后进行字符串翻译）均必须存在且不为’ None’，允许产生该条目。结果为空时不产生。

objectGroup_name: 字符串翻译后为宾语类型。如果为 tuple，第三项为 brace 时，第二项表达式为 True 允许产生该条目；第三项为 exist 时，exist 内所有条目（用”，“分割，分割后进行字符串翻译）均必须存在且不为’ None’，允许产生该条目。结果为空时不产生。无该条目时，将会将宾语加入 Triggers，或者加入该层。

optional: dict 宾语可选项键值对，值进行字符串翻译。值如果为 tuple，第三项为 brace 时，第二项表达式为 True 允许产生该条目；第三项为 exist 时，exist 内所有条目（用”，“分割，分割后进行字符串翻译）均必须存在且不为’ None’，允许产生该条目。结果为空时不产生。

Notations

以下不是参数，是一些注意事项。

brace translation

表达式翻译。会试图按照引用和字典进行 depth 次翻译，如果有大括号，会试图进行表达式翻译 (与前者交替进行，共 brace_exp_brace 次)。然后计算结果。计算发生错误，则按原结果进行。

str translation

字符串翻译。会将 {} 内的内容进行表达式翻译。{} 后出现 &，那么后面必须出现两个一位数字。分别是 depth 和 brace_exp_brace。