Python 脚本开发注意事项及问题解决方案

Python 脚本开发注意事项及问题解决方案

xxx.py，尽量不使用公共的名称，如 lib.py ,可能会与 python 中本身自带的名称冲突，导致加载问题。

from xxx import * 报错 ModuleNotFoundError: No module named 'xxx' ，解决方案： 则需要在对应的 from xxx import * 前加上如下代码

import sys

import os

current_dir = os.path.dirname(os.path.abspath(**file**))

sys.path.append(current_dir)

from xxx import *

如果报错 NameError: name 'xxx' is not defined，xxx应该是作为函数的输入的引用类型，

如下报错：

image.png

对应脚本代码

def button_ext(req: CallEntityExtButtonReq): base.logInfo(req.tc, base.jsonToString(req)) return base.jsonToString({"error": False}) 将 def button_ext(req: CallEntityExtButtonReq) 改为 def button_ext(req) 即可 ,去掉输入的引用类型.

def button_ext(req):

```
base.logInfo(req.tc, base.jsonToString(req))
return base.jsonToString({"error": False})
```

开发过程中，可以给先加上，便于提示，开发完成后再删除。

关于输入是 CallEntityExtButtonReq 的 函数方法默认需要返回参数，特别注意： 如果返回 modifiedEv ， 则直接写base.jsonToString("message":"","modifiedEv":req.ev)即可，如果加上了 error 则 modifiedEv不生效

'''

error: 是否报错

message : 报错信息，不报错传递 ""

modifiedEv : 只会存在与保存前的按钮里面，不传即可，且不能与 error 一起传

'''

return base.jsonToString({"error":False,"message":"","modifiedEv":req.ev})

不推荐使用 python 转字符串方法 str(xxx), 如果 xxx 为 json数据 ， 如

req = {"A": 1，"B":2 }

不推荐用法

str(req)

推荐用法

base.jsonToString(req)

在 java.py 中，如 HttpRequest 请求的时候，reqBody 需要传递的是 json 格式的字符串，如果使用 str的话则 req 实际的内容为 "{'A':1,'B':2}"，其中的 key 都会变为 单引号,由于 java.py中实际执行的是后端封装的 java函数，这样就会导致请求的内容不是一个正常的 json 格式。

正确使用：

如果调用的 java.py 中的函数中的输入参数是字符串，但是字符串中有 " 的时候，则需要用到 java.py 中提供的 base.jsonToString()方法。

不能使用 python 休眠方法 time.sleep(x) ，如： 猎鹰任务扩展块代码中，使用了 while 循环，每次 sleep x 秒，代码如下

def callForGet(tc, inputParams, taskInputParams):

tc = base.traceContext()

num = 0

while True:

```
    container = entity.findOne(tc, "FbContainer", cq.idEq("LX0001"), None)
```

if contaienr.get("locked"):

break

else:

错误用法

time.sleep(5)

正确用法

thread.sleep(5000)

base.logInfo(tc, "ok")

使用 python 自带的休眠方法，会导致猎鹰任务如果取消后，while 不会停止。

正确使用：

使用 java.py提供的 thread.sleep(xxx)

python 脚本中有时候可以使用 obj.filed 获取属性，有时候必须使用 obj["field"]，根本原因在于获取 obj 的方法返回值是不是一个具体的对象，比如 mars.mustGetRobotInfoAll(robotName, SCENE_NAME)，获取机器人信息返回值明确定义了是 MrRobotInfoAll。

guoshaoshuai

def mustGetRobotInfoAll(self, robotName: str, sceneName: str | None) -> MrRobotInfoAll:

pass

class MrRobotInfoAll:

```
def __init__(self, id: str, systemConfig: object, runtimeRecord: object |
None,
             selfReport: MrRobotSelfReport | None,
```

online: bool):

self.id = id

```
    self.systemConfig = systemConfig
    self.runtimeRecord = runtimeRecord
```

self.selfReport = selfReport

self.online = online

这种就需要通过点 '.' 来获取，例如 obj 是 MrRobotInfoAll，获取电量：obj.selfReport.main.battery，因为 selfReport 也是具体的对象可以 .main，同理 main 也是具体的对象，可以 .battery，一旦哪一层属性不是具体类，就需要当作字典类型的对象使用中括号 [] 获取属性了。 如果你不想在 python 脚本中时刻判断使用哪种方式，可以统一使用 base.jsonToString() 和 base.parseJsonString() 将所有对象转成 string 再转成 JSON 格式，这样就可以统一使用 obj['field'] 的方式获取属性了。

obj = mars.mustGetRobotInfoAll(robotName, sceneName) #具体机器人和场景 obj_str = base.jsonToString(obj) obj_json = base.parseJsonString(obj_str) battery = obj_json['selfReport']['main']['battery']