Python基于组件开发

概念：
基于组件的开发(Component-Based Development简称CBD)是一种软件开发范型，它是现今软件复用理论实用化的研究热点，在组件对象模型的支持下，通过复用已有的构件，软件开发者可以"即插即用"地快速构造应用软件.

优点：

1)灵活性高：各个功能模块之间的耦合很低，每一个组件都是独立的，它附着在整个插件框架上执行，真正的实现有则加载无则忽略.

2)复用性强：由于组件之间的通信或者交互都是通过插件框架提供的接口来执行,无论需要哪个模块的功能，都只需要将该插件直接拿走复用即可.

框架描述：
之前有这么一个有趣的笑话，一个人一大早起来想吃火锅，但是他又不想出门，于是他想了个主意，他给A打电话说：今天请大家吃火锅，别的东西都有了，就差一份羊肉了，来的时候带着；完了给B打电话说：今天请大家吃火锅，别的东西都有的，火锅料忘了买了，来的时候捎上……，他用这样的方法将所有的菜凑够，足不出户，就能吃火锅，而且想吃啥就吃啥．这个例子中，这个在家里想吃火锅并且挨个给大家打电话的人便是插件式框架中的总框架，本身不提供任何的功能，角色就是总指挥．而小A 小B这些朋友则是各个组件，自己只负责自己的部分，但是每一个组件都无法单独执行，只能在总框架中执行．组件为整个开发提供基本的功能，组件之间的通信也都是通过总框架来实现的，这就是整个插件式框架．

目录结构：

```
(proj_b) [root@backup-platform PluginManager]# tree
.
├── main.py
└── plugins
    ├── AbcBase.py
    ├── demo1.py
    ├── demo2.py
    ├── demo3.py
    ├── demo3.pyc
    ├── __init__.py
    ├── __init__.pyc
    └── __pycache__
        ├── AbcBase.cpython-36.pyc
        ├── demo1.cpython-36.pyc
        ├── demo2.cpython-36.pyc
        ├── demo3.cpython-36.pyc
        └── __init__.cpython-36.pyc
```

示例代码PluginManager.py:

In [ ]:
```python
import os, traceback, sys, threading
import queue as Queue

class PluginOperator:
    lock = threading.Lock()

    def __new__(cls, *args, **kwargs):
        PluginOperator.lock.acquire()
        if not hasattr(PluginOperator, "_instance"):
            PluginOperator._instance = object.__new__(cls)
        PluginOperator.lock.release()
        return PluginOperator._instance

    def __init__(self):
        self.plugins = {}
        self.q = Queue.Queue()

    def load_plugins(self):
        for filename in os.listdir("plugins"):
            if not filename.endswith(".py") or filename.startswith("_") or filename.startswith("Abc"):
                continue
            plugin_name = filename.split('.')[0]
            plugin = __import__("plugins." + plugin_name, fromlist=[plugin_name])
            clazz = plugin.GetPluginClass()
            self.plugins[plugin_name] = clazz

    def __callback(self, *, result):
        if isinstance(result, dict):
            self.q.put(result)
        else:
            self.q.put({})


    def start_plugins(self):
        result = ''
        self.load_plugins()
        for (plugin_name, plugin_object) in self.lst_plugins().items():
            try:
                plugin_object.start()
                x = plugin_object.run()
                self.__callback(result=x)
            except Exception as e:
                traceback.print_exc()
                t = 'plugins {plugin_name} start failed! err:{err_msg}'.format(plugin_name=plugin_name,
                                                                               err_msg=str(e)
                                                                               )
                result += t + "\n"
        sys.stdout.write(result)

    def lst_plugins(self, plugin_name=None):
        if not plugin_name:
            return self.plugins
        else:
            return {plugin_name: self.plugins[plugin_name]}

    def remove_plugins(self, plugin_name):
        if plugin_name in self.lst_plugins():
            self.lst_plugins().pop(plugin_name)

    def get_plugins_result(self):
        while not self.q.empty():
            print(self.q.get())

if __name__=="__main__":
    x1 = PluginOperator()
    x2 = PluginOperator()
    x3 = PluginOperator()
    x1.start_plugins()
    x1.get_plugins_result()
```

```
1) import module相当于__import__("module")
2) from module import func相当于__import__("module", fromlist=["func"])
3) import package.module相当于__import__("package.module", fromlist=["module"])
```

示例代码：

In [ ]:
```
#coding:utf8
import sys
from abc import ABCMeta, abstractmethod

class AbcPlugins(metaclass=ABCMeta):

    @classmethod
    @abstractmethod
    def run(cls):
        pass

    @classmethod
    def start(cls):
        s = 'plugins {className} is starting..'.format(className=cls.__name__)
        sys.stdout.write(s + "\n")

    @classmethod
    def stop(cls):
        s = 'plugins {className} is stop..'.format(className=cls.__name__)
        sys.stdout.write(s + "\n")

def GetPluginClass():
    pass
```

示例代码__init__.py

In [ ]:
```
import os,sys
BASE_DIR=os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, BASE_DIR)
```

示例代码Plugins1:

In [ ]:
```
#coding:utf8
from AbcBase import *
class Plugins1(AbcPlugins):
    @classmethod
    def run(cls):
        print("hello")
        return {cls.__name__: {"a": 1}}

def GetPluginClass():
    return Plugins1
```

示例代码Plugins2:

In [ ]:
```
from AbcBase import *
class Plugins2(AbcPlugins):
    @classmethod
    def run(cls):
        print("hello p2")
        return {cls.__name__: {"b": 2}}

def GetPluginClass():
    return Plugins2
```

运行结果:

```
plugins Plugins3 is starting..hello
plugins Plugins2 is starting..hello p2
plugins Plugins1 is starting..hello
{'Plugins3': {}}
{'Plugins2': {'b': 2}}
{'Plugins1': {'a': 1}}
```

In [ ]: