

call()魔术方法

```
1 作用：实现了__call__()魔术方法的类的实例对象会变成一个可调用对象，对实例对象进行调用，实际上是调用了实例对象的__call__()魔术方法
2
3  class Maker():
4      name="hello"
5
6      def __call__(self):
7          print("当对象象函数一样调用时,我被调用")
8      def myfunc(self):
9          print("myfunc")
10
11 m=Maker()
12 m()#当对象象函数一样调用时,我被调用
```

使用类作为装饰器

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #====#====#====#====
4  #Author:
5  #CreatDate:
6  #Version:
7  #====#====#====#====
8  import time
9
10 class Maker():
11     def __init__(self,func):
12         self.func=func
13
14     def __call__(self,*arg):
15         ret=self.func(*arg)
16         return ret+1000
17
18 #这里的Maker(myfunc)是生成一个对象
19 @Maker#myfunc=Maker(myfunc)
20 def myfunc(a,b):
21     time.sleep(3)
22     return a+b
23
24 #这里使用对象调用,那么类中就必须有__call__函数
25 print(myfunc(10,20))
```

使用装饰器来实现单例模式

通过装饰器实现

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #====#====#====#====
4  #Author:
5  #CreatDate:
6  #Version:
7  #====#====#====#====
8
9  # class Maker():
10     # def myfunc(self):
11     #     print("Maker myfunc")
12     #
13     # m=Maker()
14     # m2=Maker()
15     # print(id(m))
16     # print(id(m2))
17
18
19 #1. 定义一个全局的字典变量, 用于保存单例模式下实例化出的对象, 键是类名, 值是对象
20 instances={}
21
22
23 def myfunc(cls):
24     def mytest(*args, **kwargs):
25         if cls.__name__ not in instances.keys():
26             #如果这个类名不在字典的键里, 那么生成对象
27             instances[cls.__name__]=cls(*args, **kwargs)
28             return instances[cls.__name__]
29         return mytest
30
31 @myfunc    #Maker=myfunc(Maker)
32 class Maker():
33     pass
34
35
36 t=Maker()
37 t2=Maker()
38 print(id(t))
39 print(id(t2))
```

通过new()魔术方法实现单例模式

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
```

```

3  #====#====#====#====
4  #Author:
5  #CreatDate:
6  #Version:
7  #====#====#====#====
8
9  class Maker():
10     __instance=None
11
12     def __new__(cls, *args, **kwargs):
13         if cls.__instance is None:
14             #如果__instance没有值,那么就使用object的来实例化对象
15             cls.__instance=super().__new__(cls)
16             return cls.__instance
17
18 t=Maker()
19 t2=Maker()
20 print(id(t))
21 print(id(t2))
22

```

类属性

```

1  实例属性： 在__init__魔术方法中初始化的属性，或者通过实例对象添加的属性
2  类属性：   在类名称下面定义的属性
3
4  class Maker():
5      name="mytest"#类属性
6      def myfunc(self):
7          self.age=18#实例属性
8
9  #!/usr/bin/env python
10 # -*- coding:utf-8 -*-
11 #====#====#====#====
12 #Author:
13 #CreatDate:
14 #Version:
15 #====#====#====#====
16
17 class Maker():
18     name="张安"
19
20     def mytest(self):
21         print(self.name)#在成员函数中使用self来访问类属性
22         print(id(self.name))
23
24 m=Maker()
25 m.mytest()
26 m.name="李四"#这里不是修改类属性,而是增加一个实例属性
27 m.mytest()

```

```
28
29 Maker.name="王五"#修改类属性的值
30 print(m.name)
31 print(Maker.name)
```

方法

实例方法

第一个参数是self, 代表当前实例对象本身

类方法

第一个参数是cls,代表当前的类对象

需要在方法的上面加上@classmethod

类方法可以通过实例对象和类对象来访问

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #====#====#====#====
4  #Author:
5  #CreatDate:
6  #Version:
7  #====#====#====#====
8
9  class Maker():
10     def myfunc(self):
11         print("我是普通的成员函数")
12
13     @classmethod
14     def mytest(cls):
15         print("我是类方法")
16
17
18 m=Maker()
19 m.myfunc()
20
21 # Maker.myfunc()#报错,不能使用类名调用普通的成员函数
22
23 Maker.mytest()
24
25
26 作用:方便别人直接使用类名来访问类方法,而不需要再去实例化对象
```

静态方法

不需要额外的参数

通过@staticmethod来进行装饰

静态方法可以通过类对象和实例对象来进行访问

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #====#====#====#====
4  #Author:
5  #CreateDate:
6  #Version:
7  #====#====#====#====
8
9  class Maker():
10
11      name="hello"
12      @staticmethod
13      def mytest():
14          #静态方法中不能使用类的属性和方法
15          a=10
16          a+=1
17          print("我是静态方法",a)
18
19  m=Maker()
20  m.mytest()
21  Maker.mytest()
22
23
24  静态方法的作用:
25  #!/usr/bin/env python
26  # -*- coding:utf-8 -*-
27  #====#====#====#====
28  #Author:
29  #CreateDate:
30  #Version:
31  #====#====#====#====
32  #静态方法在类中是独立的,单纯的函数,只是托管在类的空间,增强代码的逻辑性,简化代码的维护
33  import time
34  class Maker():
35      def __init__(self,h,m,s):
36          self.h=h
37          self.m=m
38          self.s=s
39
40      @staticmethod
41      def showtime():
42          return time.strftime("%H:%M:%S",time.localtime())
43
44  print(Maker.showtime())
45  m=Maker(21,37,56)
46  kk=m.showtime()
47  print(kk)
```

进程的概念

- 1 进程是什么：进程是操作系统进行资源分配的基本单位
- 2 比如，我们在操作系统上运行一个应用程序，其实对操作系统来说你就开启了一个进程
- 3
- 4 单核的CPU,同一时刻只能运维单个进程，虽然可以同时运行多个程序，但进程之间是通过轮流占用CPU来执行的
- 5
- 6 现在的CPU核数越来越多，有4核，8核，28核等等，为了充分发挥多核优势，提高程序的并发度，我们要使用多进程
- 7

创建进程的类Process

```
1 Multiprocessing模块提供了一个创建进程的类 Process,所以你使用Process类之前要引入
  Multiprocessing模块
2 创建进程有以下两种方法:
3 1.创建一个Process类的实例，并指定目标任务函数
4 2.自定义一个类，并继承Process类，重写__init__()方法和run()方法
5
6 一.使用Process类的实例创建进程
7 #!/usr/bin/env python
8 # -*- coding:utf-8 -*-
9 #====#====#====#====
10 #Author:
11 #CreatDate:
12 #Version:
13 #====#====#====#====
14 #引入模块
15 from multiprocessing import Process
16 #为了获取进程pid
17 import os
18 import time
19 def mytest(d):
20     num=0
21     for i in range(d*1000000):
22         num+=i
23     print(f"进程的pid为{os.getpid()}")
24
25 #在该代码块内写的代码只能在本文件有,不能被别的文件引用
26 if __name__=="__main__":
27     print("父进程PID为%s"%os.getpid())
28     #创建子进程,target的值是子进程要执行的函数,args的值是函数的参数
29     p1=Process(target=mytest,args=(3,))
30     t0=time.time()#记录当前时间
31     #激活子进程
32     p1.start()
33     p1.join()#阻塞主进程,让子进程完成任务或子进程被终止
34     t1 = time.time() # 记录当前时间
35     print(t1-t0)
36
37
```

```

38 #注意,创建进程必须在__main__里面,因为windows创建进程相当于导入该文件,由于创建进程在main函数之前
39 # ,会无限递归导入模块创建进程。所以在windows创建进程应放在main函数之后,因为main函数不会执行被导入模块。
40
41 二.使用类创建子进程
42 #!/usr/bin/env python
43 # -*- coding:utf-8 -*-
44 #====#====#====#====
45 #Author:
46 #CreateDate:
47 #Version:
48 #====#====#====#====
49 #引入模块
50 from multiprocessing import Process
51 #为了获取进程pid
52 import os
53 import time
54
55 class Maker(Process):
56     def __init__(self,d):
57         self.d=d
58         super().__init__()
59
60     #子进程要执行任务函数
61     def run(self):
62         num = 0
63         for i in range(self.d * 1000000):
64             num += i
65         print(f"进程的pid为{os.getpid()}")
66
67
68
69 #在该代码块内写的代码只能在本文件有,不能被别的文件引用
70 if __name__=="__main__":
71     print("父进程PID为%s "%os.getpid())
72     #创建子进程,使用类
73     p1=Maker(3)
74     t0=time.time()#记录当前时间
75     #激活子进程
76     p1.start()
77     p1.join()#阻塞主进程,让子进程完成任务或子进程被终止
78     t1 = time.time() # 记录当前时间
79     print(t1-t0)
80
81
82
83
84
85
86
87

```