# python进阶-装饰器

## 实验背景

经常可以看到在一些python的函数申明前，添加了@的标识，如下图@tf.function和@app.route

这个@就是python中的装饰器（decorater），在实验内容中WedO实验君就详细说下python的这个特性。

## 实验内容

### 什么是装饰器

python中的所有的东西都是对象，连函数也是对象，也能return； 这个是实验君最喜欢python的地方，一个函数可以返回任意你想要返回的东西（不像c/c++/java想要多个返回值，还需要废些功夫。

言归正传，装饰器是对函数的一种装饰。换句话说就是在不改变原来函数的功能基础上，给函数添加一些额外的功能，比如公共的处理，日志打印等。

举个简单的例子: 为函数的执行进行耗时统计，代码如下

```
def func():
    start_time = time.time()
    print('some function is running')
    print('the running time is %s s' % (time.time()- start_time))
```

如果想给不同的函数计算耗时，又不想重复的写start_time,要怎么办呢？ 一种解决方案是把函数作为一个参数传递给计算耗时的函数里（这似乎只有在python里可以办到吧）

```
def real_func1():
    print('some function is running1')

def real_func2():
    print('some function is running2')

def time_func(func):
    start_time = time.time()
    func()
    print('the running time is %s s' % (time.time()- start_time))

# 执行
    time_func(real_func1)
    time_func(real_func2)
```

装饰器其实就是上面这种解决方案的一种简写。

### 如何构建装饰器

上面的例子怎么改写成装饰器？

```
def log_time()
   def time_func(func):
      start_time = time.time()
      func()
      print('the running time is %s s' % (time.time()- start_time))
   return time_func

@log_time
def real_func1():
   print('some function is running1')
@log_time
def real_func2():
   print('some function is running2')

# 执行
   real_func1()
   real_func2()
```

装饰器log_time就是给函数real_func1和real_func2外包了一个统计耗时的功能。

我们再看几个装饰的例子

- 有参数的装饰器

```
def log_time(func):
 def make_decorater(*args, **kwargs):
     print('outer print1')
     test_func = func(*args, **kwargs)
     print('outer print2')
     return test_func
 return make_decorater

@log_time
def test(num):
   print('inner print 3')
   return num+1
num = test(2)
print(num)
# outer print1
# inner print 3
# outer print2
# 3
```

- 多个装饰器(注意执行顺序)

```
def log_time1(func):
 def make_decorater():
     print('decorater1: outer print1')
     func()
     print('decorater1: outer print2')
 return make_decorater

def log_time2(func):
   def make_decorater():
      print('decorater2: outer print1')
      func()
      print('decorater2: outer print2')
   return make_decorater

@log_time1
@log_time2
def test():
   print('inner print 3')
test()

# decorater1: outer print1
# decorater2: outer print1
# inner print 3
# decorater2: outer print2
# decorater1: outer print2
```

### 装饰器的作用

了解了装饰器的功能，聪明的你一定想到了很多装饰器的用处。就如开头提到的

- `@tf.function`是把普通的函数改造成tf的op操作的函数
- `app.route` 是`flask`中普通函数封装成可以接收http请求的函数

总结下, 装饰器可以用来：

- 公共日志的打印
- 公共的模块封装，比如授权， 异常处理, 函数转换

## 实验结语

本实验和大家简单描述了`python`装饰器的使用，希望对大家有帮助。