Python装饰器（Decorator）[Python小技巧：装饰器(Decorator)_哔哩哔哩_bilibili]

```
import time

def is_prime(num):
    '''
    判断素数
    '''
    if num < 2:
        return False
    elif num == 2:
        return True
    else:
        for i in range(2, num):
            if num % i == 0:
                return False
        return True

def prime_nums():
    '''
    给函数增加一个计时功能
    '''
    t1 = time.time()
    for i in range(10000):
        if is_prime(i):
            print(i)
    t2 = time.time()
    print(t2-t1)

prime_nums()
```

函数prime_nums原先是没有计时功能的，在内部添加t1,t2后相减得到运行时间，但如果其他函数也有计时需求，显然在每个函数内部添加这三行代码，效率不高而且代码冗余

```
import time

def display_time(func):
    t1 = time.time()
    func()
    t2 = time.time()
    print(t2 - t1)

def is_prime(num):
    ...

def prime_nums():
    for i in range(10000):
        if is_prime(i):
            print(i)

display_time(prime_nums)
```

一种土办法是再写一个display_time作为辅助函数，将prime_nums作为参数传递进去，执行display_time即可增加时间功能。而这个辅助函数对于使用者来说应该隐藏起来，提升代码的阅读性，因此该方法也不够完美

- **装饰器模式**（被装饰的函数：没有返回值，没有参数）

```
import time

def display_time(func):
    def wrapper():
        t1 = time.time()
        func()
        t2 = time.time()
        print(t2 - t1)
    return wrapper

def is_prime(num):
    ...

@display_time
def prime_nums():
    for i in range(10000):
        if is_prime(i):
            print(i)

#prime_nums = display_time(prime_nums)
prime_nums()
```

对上面的土方法进行修改，display_time内部再定义一个wrapper函数，最后返回wrapper，在prime_nums前添加@display_time对它进行修饰，之后直接调用prime_nums就是带有计时功能的，它接受的就是display_time返回的wrapper，因此@display_time的作用与注释的那一行一样

- **装饰器模式**（被装饰的函数：有返回值，没有参数）

```
import time

def display_time(func):
    def wrapper():
        t1 = time.time()
        ret = func()
        t2 = time.time()
        print(t2 - t1)
        return ret
    return wrapper

def is_prime(num):
    ...

@display_time
def prime_nums():
    count = 0
    for i in range(10000):
        if is_prime(i):
            count = count + 1
    return count

count = prime_nums()
print(count)
```

将prime_nums修改为统计素数个数的函数，带有返回值，它对应的就是display_time中的func函数以及wrapper函数，因此也需要有返回值记为ret，在返回前先接收func的返回值，有时也可以直接return func()

- **装饰器模式**（被装饰的函数：有返回值，有参数）

```
import time

def display_time(func):
    def wrapper(*args):
        t1 = time.time()
        ret = func(*args)
        t2 = time.time()
        print(t2 - t1)
        return ret
    return wrapper

def is_prime(num):
    ...

@display_time
def prime_nums(maxnum):
    count = 0
    for i in range(maxnum):
        if is_prime(i):
            count = count + 1
    return count

count = prime_nums(10000)
print(count)
```

为prime_nums增加一个参数，对应的display_time中wrapper和func都添加对应参数即可，但若不知道传入参数是什么，可以使用可变参数和关键字参数即可，这里添加两处*arg，其余不变