Django信号机制工作原理

在上一节《Django信号机制执行过程及其应用》一节，我们详解的介绍了 Django 信号机制的执行过程及其应用。通过上一节知识的学习，其实我们可以认识到信号的工作原理其实就是 Signal 对象的实现过程，在不同的阶段调用了 Signal 相应的方法，但是有些地方你可能还是会感到迷茫，比如什么是观察者模式？什么是 Python 语言的弱引用等，在本节我们将从 Python 语言的相关特性出发，介绍 Django 信号机制的工作原理。

1. 观察者设计模式简述

1)观察者设计模式理解

观察者模式它定义了对象之间一对多的依赖关系，当一对象的状态发生改变的时候，所有依赖于它的对象都获取到通知并发生相应的改变。观察者模式还有一个别名叫做发布订阅模式，这个名字就非常的形象的说明了这种设计模式的理念，当订阅者订阅了某系列杂志，当杂志有了新的状态即发布者，比如更新了，那么此时就会给所有的订阅者发送一条消息，那么所有的订阅者就会收到此消息做出购买或不购买的选择。

观察者模式的两个重要角色即目标和观察者，为了形象的理解，我们把目标定义为杂志，把观察者定义为订阅了杂志的用户，当目标状态发生改变的时候，所有的观察者都会收到通知，并做出相应的动作。比如在微博或者论坛中，你发布了一个话题，那么关注你的粉丝就会收到一个通知，粉丝收到通知后可以相应的做出回馈。这也是观察者模式的适应场景。

从对观察者设计模式的说明与理解，我们可以得知 Django 信号机制就是观察者模式的实现。这种模式的优点非常明显，它在目标与观察者之间建立了轻度的关联关系，对于它们各自的扩展就会非常容易。在运行时，观察者可以动态地添加或删除（取关操作），对目标（话题发布者）不会有任何影响，反过来也是一样，所以它们是抽象耦合的。

2. Python语言特性弱引用

学习过 Python 语言的小伙伴知道，Python 的垃圾回收由引用计数、标记清理和分代回收等方式构成。其中大部分对象的生命周期由对象的引用计数来管理。在 Python 语言中一切皆对象（还是 Python 好从不缺对象），每一个对象都会维护一个叫做 obrefcnt 的属性，也就是引用计数，当一个对象有了新的引用时，obrefcnt 就会加 1 ；当对象的引用被删除时就会就会减 1；当其为 0 的时候表示当前对象没有被使用，我们可以使用如下方式查看某个对象的引用计数值：

```
1. import sys
2. sys.getrefcount()#接受一个参数对象
```

但是引用计数有一个明显的缺点就是无法解决循环引用的问题即 a 引用 b， b 引用 a，导致其引用计数永远不为 0，这样就会导致内存无法释放也浪费了系统内存，从而造成内存泄露的问题。为了避免这个问题 Python 提供了 weakref 即弱引用。它的效果是：当对一个对象创建了弱引用时，对象的引用计数不会增加。示例如下所示：

```
1. In [1]: import sys,weakref
2. In [2]: class A:
3.    ...:     def hello(self):
4.    ...:         return "c语言中文网"
5.    ...:
6. In [3]: a=A()
7. In [4]: sys.getrefcount(a)
8. Out[4]: 4
9. In [5]: ref=weakref.ref(a)
10. In [6]: sys.getrefcount(a)
11. Out[6]: 4
```

weakref 的 ref 方法用于创建弱引用对象，它的返回值是引用指向的对象。函数定义如下：

```
weakref.ref(object[callback,])
```

其中 object 即为被引用的对象，而 callback 是一个可选的回调函数。当引用的对象被删除的时候，回调函数就会被执行调用。通过 weakref.ref 创建的弱引用，在使用时需要使用 ref() 去获取 object：

```
1. In [1]: ref().hello()
2. Out[2]: 'c语言中文网'
```

weakref 提供了 finalize 来定义引用对象被删除时执行的清理函数，它的定义如下：

```
finaiize(object,func,*args,**kwargs)
```

其中，object 是引用的对象；func 是清理函数，obj 被删除时自动调用；*args 和 **kwargs 将会作为参数传递给清理函数。所以使用它的时候需要预先定义清理函数，示例如下：

```
1. In [9]: def func(obj):
2.    ...:  print("%s被删除"%obj)
3.    ...:
```

```
In [10]: weakref.finalize(a,func,str(hex(id(a))))
Out[10]: <finalize object at 0x508bd98; for 'A' at 0x85a7930>
In [11]: del a
0x85a7930被删除
```

这里还有一种特殊的情况即引用对象是对象实例的方法，此时不可以直接使用 weakref.ref 去创建弱引用，而是使用 weakref.WeakMethod()，示例如下：

```
In [14]: a=A()
In [15]: ref=weakref.WeakMethod(a.hello)#引用对象a的hello方法
In [16]: ref()()
Out[16]: 'c语言中文网'
In [17]: ref()
Out[17]: <bound method A.hello of <__main__.A object at 0x05B64F30>>
```

因为弱引用不会改变对象的引用计数，所以，Django 信号机制采用弱引用的方式对信号回调函数进行引用，以此来避免内存泄露的问题。

Django 信号机制涉及了诸多方面的知识，本节我们只是讲了你可能没有涉猎过的知识点，比如，Django 信号机制还应用了 Python 线程的同步机制，也就是使用锁即 Lock 来避免多线程程序中对共享资源的竞争导致的错误。在这里就不进行介绍了，有兴趣的同学可以自己探索。