Django信号机制执行过程及其应用

在上一节《简述Django的信号机制》中，我们对 Django 的信号机制概念以及它的适用场景做了讲解，并且了解了 Django 框架中内置的信号，它们在执行某个动作的前后被触发，比如说 HTTP 建立和关闭；Django Model 使用 save 方法保存模型实例前后，这些属于 Django 信号机制的基本内容，在本节我们会通过信号机制的执行过程以及它的实例应用，帮助大家更好的理解它。

1. 内置信号执行过程

1) 信号的发送方法

我们知道 Django 信号的执行过程包括信号发送和信号的接收，在 Signal 中提供了两种发送信号的方法：send 和 send_robust，它们会区别对待 receiver 可能抛出的异常，send 方法不会捕获任何由 receiver 抛出的异常，所以使用 send 方法不能保证所有的 receiver 都会得到信号通知。而 send_robust 则可以捕获抛出的异常，可以保证所有的 receiver 都接收到信号的通知。这两个方法需要的参数相同，以 send_robust 为例：

```
1. def send_robust(self, sender, **named):
2.     pass
```

sender 标识信号的发送者，大多数情况下它是一个类对象；**named 用来指定任意数量的关键字参数，这些参数将会传递给 receiver。

2) 接收信号执行回调函数

接收信号，并执行回调函数，这里需要将回调函数注册到信号上。我们使用 Signal 提供的 connect 方法，定义如下：

```
1. def connect(self, receiver, sender=None, weak=True, dispatch_uid=None):
2.     pass
```

这个方法接受四个参数，但是只有 receiver 参数是必需的，我们分别对这些参数进行讲解：

- receiver：必须要指定的回调函数，信号发送后，就会执行到这个函数。
- sender：信号的发送者，可以不提供。当回调函数只对特定的 sender 时，可以通过提供这个参数实现过滤。
- weak：默认值是 True，代表以弱引用的方式存储信号处理器。当 receiver 是局部变量时，可能会被当做垃圾回收掉。为避免这种情况，可以设置为 False。
- dispatch_uid：用于指定 receiver 的唯一标识符，以防止信号多次发送的情况。

3) 断开信号连接

Sginal 提供了一个与 connect 功能相反的方法 disconnect，用来断开信号的 receiver。函数定义如下：

```
def disconnect(self, receiver=None, sender=None, dispatch_uid=None):
    pass
```

可以看到它的定义和 connect 方法的参数是相似的，它的参数含义如下所示：

- receiver：标识需要断开已注册的信号接收者，若你使用了dispatch_uid 去标识 receiver，那么这个参数可以是 None。
- sender：已注册的信号发送者。
- dispatch_uid：receiver的唯一标识符。

2. 内置信号的应用

1) connect()方法实现信号注册

下面我们对 Django 的内置信号进行简单的应用，场景如下，在视图的处理前后打印一些日志信息。第一步，在 index/views.py 文件中注册信号回调函数：

```
def request_started_callback(sender,**kwargs):
    print("请求开始：%s"%kwargs['environ'])

def request_finished_callback(sender,**kwargs):
    print("请求完成")
```

从上述代码可以看出它们就是普通的 Python 函数，它们接受一个 sender 参数和一个关键字参数 **kwargs，这也是 Django 规定的信号接收者的固定格式，我们可以从 kwargs 中获取到信号发送的关键字参数，如在 request_started_callback 中获取到 environ。定义

完成了回调函数，接下来第二步就需要把相关的信号导入，第三步就是回调函数注册到信号上，如下所示：

```
1. from django.core.signals import request_started, request_finished
2. request_started.connect(request_started_callback)
3. request_finished.connect(request_finished_callback)
```

2) receiver装饰器实现信号注册

上面我们使用了 Sginal 的 connect 方法直接进行注册，当然还有一种方式就是使用装饰器的方式注册信号，如下所示：

```
1. from django.core.signals import request_started, request_finished #与http有关的内置信号
2. from django.dispatch.dispatcher import receiver
3. @receiver(request_started)
4. def request_started_callback(sender,**kwargs):
5.     #获取程序执行的环境信息
6.     print("请求开始：%s"%kwargs['environ'])
7. @receiver(request_finished)
8. def request_finished_callback(sender,**kwargs):
9.     print("请求完成")
```

装饰器 receiver() 的第一个参数是可迭代的对象，可接受一个列表，其中每一个元素都是信号实例。这两种方法选择其一即可，完成了信号的注册以后，我们就可以访问以前定义的任何一个视图函数，在 CMD 命令行可以看到，视图函数的执行前后都打印了对应的内容，这就是内置信号应用的基本流程。

3. 自定义信号应用

上述我们详解介绍了 Django 内置的信号应用过程，即导入内置信号、创建注册函数，信号中导入注册函数，通过以上三个步骤就可以实现内置信号的应用。但是在某些场景下，内置信号无法满足开发者的需求，Django 也充分考虑到这一点，并提供了自定义信号的方法，使用了起来也非常的简单。

场景如下：当你在某个网站成功注册以后，站点通常会给你填写的邮箱发送一封验证邮件，这个场景在 Django 的内置信号并没有涉及，下面我们就模拟这个过程的实现。自定义信号可以定义在项目任何一个位置，但是 Django 推荐我们已单独的文件呈现它，所以为了规范，我们在 user 应用下新建 signal.py 文件，编写如下代码：

```
1. import django.dispatch
2. from django.dispatch import receiver
3. #创建一个信号
4. from user.models import User
```

```
register_signal=django.dispatch.Signal(providing_args=["request","user"])#触发的时候需要传递的参数
#定义回调函数(即信号接收者)并使用装饰器进行注册
@receiver(register_signal,dispatch_uid="register_callback")
def register_callback(sender,**kwargs):
    print("客户端地址：%s, 邮件接收者：%s"%(kwargs['request'].META['REMOTE_ADDR'],kwargs['user'].email))
```

我们在这个文件中实现了信号的创建，信号的接收以及通过 receiver() 完成了信号的注册，然后我们在 user/views.py 中定义一个视图函数：

```
from user.signals import register_signal
def hello_my_signal(request):
    #注意要和回调函数中的**kwargs的参数保持一致
    #参数 sender （信号发送者指函数） **named （**kwargs参数相同）
    register_signal.send(hello_my_signal,request=request,user=User.objects.get(username="admin"))
    print("注册成功已经发送邮件")
    return HttpResponse('Hello signal')
```

通过这个视图函数，我们模拟用户登录后发送邮件的情景。定义好路由映射，然后访问 127.0.0.1:8000/user/signal/，最后在 CMD 命令行输出如下结果：

客户端地址：127.0.0.1，邮件接收者：admin@163.com

这样我们就通过了自定义信号模拟实现了上述场景中的功能。通过对内置信号执行过程介绍以及内置和自定义信号的实例应用，我们对 Django 的信号机制使用有了更加深入的理解，在下一节中我们将从 Python 语言的特性讲解 Django 信号机制的工作原理。