Signals

2017.02.08 flora5

Django 包括一个“signal dispatcher”，它帮助 允许解耦的应用程序当 action 在框架的其他地方发生时得到通知。 简而言之，signal 允许特定的发送者通知一组接收者发生了某些行为。 当许多代码对同一事件感兴趣时，特别有用。

Django 提供了一系列内置信号可以让用户代码获得通知，通过 Django 本身的一些特定操作。这包括一些有用的通知：

django.db.models.signals.pre__save&django.db.models.signals.post__save

在 Save()方法调用之前或者之后发送

django.db.models.signals.pre__delete & django.db.models.signals.post__delete

在 model 的 delete()或者 queryset 的 delete()方法调用之前或者之后发送

django.db.models.signals.m2m__changed

当一个 model 的 ManyToMany 字段被修改的时候发送

django.core.signals.request__started & django.core.signals.request__finished

当 Django 开始或者完成一个 HTTP 请求的时候发送

查看完整的信号列表和信号介绍见文档： built-in signal documentation

你也可以定义和发送自己定制的信号： 见如下：

监听信号

要接收信号，用 Signal.connect（）方法注册接收函数。 信号被发送时接收函数被调用。

Signal.connect(receiver, sender=None, weak=True, dispatch_uid=None)

参数:

- **receiver** – 连接到此信号的回调函数 详见 Receiver functions
- **sender** – 指定信号的发送者
- 详见： Connecting to signals sent by specific senders
- **weak** – 默认，Django 将信号处理者存储为若引用
-
- 如果你的接受者是一个本地函数，它可能会被垃圾收集；
- 要避免被垃圾收集，在调用信号的 connect()方法的时候传递 weak = False
- **dispatch_uid** – 可能发送重复信号的情况下，对信号接受者的，唯一标识符，
  ： Preventing duplicate signals

让我们通过注册每个 HTTP 请求完成后调用的信号来看看这是如何工作的。 我们将连接到 request_finished 信号。

接受函数

首先我们需要定义一个接受函数， 一个 receiver 可以是任何 python 函数或者方法。

```
def my_callback(sender, **kwargs):
    print("Request finished!")
```

注意，函数接受 sender 参数，以及关键字参数（** kwargs）；所有 signal handlers 程序必须接收这些参数。

稍后再看 sender，但现在看看** kwargs 参数。所有信号都发送关键字参数，并且可以随时更改这些关键字参数。在 request_finished 中， 文档记录为不发送任何参数，意味着我们可能试图将信号处理写为 my_callback(sender)

这将是错误的 - 事实上，如果你这样做，Django 会抛出一个错误。 这是因为在任何时候参数可以被添加到信号，并且你的 receiver 必须能够处理这些新的参数。

连接 receiver 函数

有两种方法可以将 receiver 连接到信号。 可以采取手动连接路由：

```
from django.core.signals import request_finished

request_finished.connect(my_callback)
```

或者，可以用 **receiver()** 装饰器

```
receiver(signal)
```

参数:

signal – 连接函数的信号或者一个信号列表

如何用装饰器连接：

```
from django.core.signals import request_finished
from django.dispatch import receiver


    @receiver(request_finished)
    def my_callback(sender, **kwargs):
        print("Request finished!")
```

现在，my_callback 函数将会在每次 request 完成后被调用

Where should this code live?

严格来说，信号处理和注册代码可以在任何你喜欢的地方存在，尽管建议避免应用程序的根模块和它的模型模块，以最小化导入代码的副作用。

实际上，signal handler 通常在它们相关的 application 的 signals submodule 中定义。 信号 reciver 连接在你的应用程序 configuration class 的 ready（）方法中。 如果你使用 receiver（）装饰器，只需在 ready（）中 import 信号子模块。

Note

在测试中 ready()方法可能会执行不止一次， 你可能希望保护你的信号免于重复，特别是如果你打算在测试中发送它们。

连接到由特定发件人发送的信号

一些信号被发送多次，但你只有兴趣接收这些信号的某个子集。例如，考虑在保存模型之前发送的 django.db.models.signals.pre_save 信号。 大多数时候，你不需要在任何任何模型被保存的时候都得到通知 - 只需某一个特定模型被保存的时候接受信号。
这种情况下，可以注册接收特定 sender 发送的信号。 比如： django.db.models.signals.pre_save，sender 是要保存的 model class，因此可以指示你只想要某些模型发送的信号：

```
from django.db.models.signals import pre_save

from django.dispatch import receiver

from myapp.models import MyModel

    @receiver(pre_save, sender=MyModel)

    def my_handler(sender, **kwargs):

        ...
```

my_handler 函数只会在 MyModel 实例被 save 时调用。
不同的信号使用不同的对象作为其 sender; 你需要参考内置的信号文档了解每个特定信号的详细信息。

防止重复信号

在一些情况下，将 receiver 连接到信号的代码可以运行多次。 这可能导致你的 receiver 函数被注册多次，因此，对于单个信号事件被多次调用。

如果此行为会引起问题（例如，当使用信号在每次保存模型时发送电子邮件），请传递唯一标识符作为 dispatch_uid 参数以标识 receiver 函数。 这个标识符通常是一个字符串，虽然任何 hashable 对象就足够。最终结果是，对于每个唯一的 dispatch_uid 值，receiver 函数只会绑定到信号一次：

```
from django.core.signals import request_finished

request_finished.connect(my_callback, dispatch_uid="my_unique_identifier")
```

定义和发送信号

您的应用程序可以利用信号基础设施（signal infrastructure）并提供自己的信号。

定义信号

class Signal(providing_args=list)

所有信号都是 django.dispatch.Signal 的实例。 provide_args 是信号将提供给 listeners 的参数名称的列表。文档中虽然这样写，但是，没有机制会检查信号是不是提供了这些参数给它的 listeners。

例如：

```
import django.dispatch

pizza_done = django.dispatch.Signal(providing_args=["toppings", "size"])
```

这声明了一个 pizza_done 信号，它将为 receiver 提供 topings 和 size 参数。

请记住，你可以随时更改此参数列表，so getting the API right on the first try isn’t necessary.

发送信号

有两种发送信号的方式

Signal.send(sender, **kwargs)

Signal.send_robust(sender, **kwargs)

要发送信号，请调用 Signal.send（） （所有内置信号使用这种方式）或 Signal.send_robust（）。 必须提供 sender 参数（通常是一个 class），并且可以提供你需要的其他参数。

例如，以下是发送我们的 pizza_done 信号的方式：

```
class PizzaStore(object):
    ...
    def send_pizza(self, toppings, size):
        pizza_done.send(sender=self.__class__, toppings=toppings, size=size)
        ...
```

send（）和 send_robust（）都返回一个元组对[（receiver，response），...]的列表，表示被调用的 receiver 函数列表及其响应值。

send（）与 send_robust（）在处理 receiver 函数引发的异常方面有所不同。 send（）不捕获 receiver 引发的任何异常; 它只是允许错误传播。 因此，并不是所有 receiver 都可以在信号面对错误时被通知。

send_robust（）捕获从 Python 的 Exception 类派生的所有错误，并确保所有 receiver 都被该信号通知。 如果发生错误，引起错误的 error instance 以 tuple pair 的形式被返回给 receiver.

当调用 send_robust（）时 ，Tracebacks 呈现在错误返回的__traceback__ 属性中

断开信号

Signal.disconnect(receiver=None, sender=None, dispatch_uid=None)[source]

要断开接收器与信号的连接，请调用 Signal.disconnect（）。 参数如 Signal.connect（）中所述。 如果接收器断开连接，该方法返回 True，否则返回 False。

receiver 参数指示 要断开的 注册的 receiver 。 可能是 None，如果使用 dispatch_uid 来标识 receiver

从版本 1.9 开始弃用：

弱参数已被弃用，因为它没有效果。 它将在 Django 2.0 中删除。