2.8. 信号

0.5 版中的新功能。

笔记

优秀的blinker库提供了信号支持。如果您希望启用信号支持，则必须安装此库，尽管 MongoEngine 不需要它来运行。

2.8.1. 概述

信号位于*mongoengine.signals*模块中。除非指定信号不接收*发送者*类和 *文档*实例之外的其他参数。只有在处理相关功能期间没有引发异常时，才会调用后置信号。

可用信号包括：

预初始化

`Document`在创建新的或 实例的过程中`EmbeddedDocument`，在收集构造函数参数之后但在对它们进行任何其他处理之前调用。 *（即默认值的分配。）此信号的处理程序使用*values 关键字参数传递参数字典，并且可以在返回之前修改此字典。

post_init

在完成对新实例`Document`或 实例的所有处理后调用。`EmbeddedDocument`

预存

`save()`在执行任何操作之前调用。

pre_save_post_validation

`save()`在验证发生之后但在保存之前调用。

post_save

`save()`在大多数操作（验证、插入/更新和级联，但不清除脏标志）成功完成后调用。*传递了创建的*附加布尔关键字参数 ，以指示保存是插入还是更新。

预删除

`delete()`在尝试删除操作之前调用。

删除后

`delete()`成功删除记录后调用。

预批量插入

在验证要插入的文档之后但在写入任何数据之前调用。在这种情况下，*文档参数*将替换为表示要插入的文档列表的文档参数。

post_bulk_insert

在成功的批量插入操作后调用。根据*pre_bulk_insert*，*文档*参数被省略并替换为*文档*参数。一个附加的布尔参数*loaded将文档*的内容标识 为*True* `Document` 时的实例，或者如果为*False*则只是插入记录的主键值列表。

2.8.2. 附加事件

编写如下处理函数后：

```
import logging
from datetime import datetime

from mongoengine import *
from mongoengine import signals

def update_modified(sender, document):
    document.modified = datetime.utcnow()
```

您将事件处理程序附加到您的 `Document` 或 `EmbeddedDocument` 子类：

```
class Record(Document):
    modified = DateTimeField()

signals.pre_save.connect(update_modified)
```

虽然这不是最详尽的文档模型，但它确实展示了所涉及的概念。作为更完整的演示，您还可以在子类中定义处理程序：

```
class Author(Document):
    name = StringField()

    @classmethod
    def pre_save(cls, sender, document, **kwargs):
        logging.debug("Pre Save: %s" % document.name)

    @classmethod
    def post_save(cls, sender, document, **kwargs):
        logging.debug("Post Save: %s" % document.name)
        if 'created' in kwargs:
            if kwargs['created']:
                logging.debug("Created")
            else:
                logging.debug("Updated")

signals.pre_save.connect(Author.pre_save, sender=Author)
signals.post_save.connect(Author.post_save, sender=Author)
```

警告

请注意，EmbeddedDocument 仅支持 pre/post_init 信号。pre/post_save 等应仅附加到文档类。将 pre_save 附加到 EmbeddedDocument 会被静默忽略。

最后，您还可以使用这个小装饰器快速创建一些信号并将它们作为类装饰器附加到您的 `Document` 或 子类中：`EmbeddedDocument`

```
def handler(event):
    """Signal decorator to allow use of callback functions as class decorators."""

    def decorator(fn):
        def apply(cls):
            event.connect(fn, sender=cls)
            return cls

        fn.apply = apply
        return fn

    return decorator
```

使用更新修改时间的第一个示例，代码现在看起来更清晰，同时仍然允许手动执行回调：

```
@handler(signals.pre_save)
def update_modified(sender, document):
    document.modified = datetime.utcnow()

@update_modified.apply
class Record(Document):
    modified = DateTimeField()
```