一个由Django的save方法引发的bug

03-11 11:36
1.84k

bug的产生

我已经很久没碰到让人比较头疼的bug了，前两天的时候碰到了一个。写个文章记录下来。 希望看过文章的同学下次碰到类似bug就不用被烦恼到。

出现问题的这段代码简化过的逻辑大概是这个样子的：

```
from app.models import User
from celery.task import task

... ...

# 假设我们取到的user的age和name都是None
user = User.objects.get(pk=1)
# 开启两个task来执行任务
update_user_age.delay(user)
update_user_name.delay(user)


# Celery tasks

@task
def update_user_age(user):
    """
    接受一个User实例，然后修改它的年龄
    """
    user.age = 38
    user.save()


@task
def update_user_name(user):
    """
    接受一个User实例，然后修改它的名字
    """
    user.name = 'piglei'
    user.save()
```

这段代码的执行结果是，user的age和name字段永远 不会都被赋上值 ，要么是有name没有age，要么有age没有name。

怎么样？能一眼能看出来问题在什么地方吗？如果能看出来的话，可以直接跳到最后一段啦。 不过当时笔者碰到的代码比这个要复杂不少，有一些task是通过django的signals触发 调用的，所以给排

查产生了很大的麻烦，花了一些功夫才找到问题。

对于那些不能看出来问题的同学，且往下看。

问题是如何产生的？

所有学过django的人都知道，在django中修改单个Model对象，一般是这样做的：

```
from app.models import User

user = User.objects.get(pk=1)
user.age = 38
user.save()
```

我们先获取一个实例，然后修改这个实例的某个属性，最后调用save方法去操作数据库，大功告成。 那我们的改动是怎么被回写到数据库的呢？是在最后调用save方法的时候，django最后会生成并执行一条这样的UPDATE语句：

```
UPDATE "user_table_name" set age = 38, name = 'name_value' ... ... WHERE "user
```

我们可以注意到， 不光是你修改过的**age**字段被提交到了数据库，其他你并没有修改过的字段，也被放在**update**的参数里面一并被提交过去了。 当你的所有操作都是串行，没有什么并发同时操作同一个model的时候，这样的处理方式，一般不会给你带来任何麻烦。

但是像我们前面所描述的使用环境，就很可能会产生问题。当我们往celery队列里面丢一个task过去的时候，所有的参数会先经过序列化，然后丢进队列里，这样当worker拿到参数开始执行的时候，传过来的这个**Model**实例里面很有可能会有某些字段是 ``已经过时`` 的。

比如上面例子中的 `user` 对象，在作为参数传到 `update_user_age` 和 `update_user_name` 任务的同时。它的name和age参数都是None，所以在这两个task里面，修改了 `user` 的某一个字段后，调用save方法保存时。另外一个没被修改的字段也被当做None放在了UPDATE语句里回写到了数据库。

这样便导致了问题。

解决办法

知道了这个问题的原因之后，要解决起来其实也是比较简单的。最简单的解决办法只要让celery处理的时候不要调用save方法即可。

我使用了一个django snippet:

```
def update(instance, **kwargs):
    "Atomically update instance, setting field/value pairs from kwargs"
    # fields that use auto_now=True should be updated corrected, too!
    for field in instance._meta.fields:
        if hasattr(field, 'auto_now') and field.auto_now and field.name not in kwa
```

```
            kwargs[field.name] = field.pre_save(instance, False)

    rows_affected = instance.__class__._default_manager.filter(pk=instance.pk).updat

    # apply the updated args to the instance to mimic the change
    # note that these might slightly differ from the true database values
    # as the DB could have been updated by another thread. callers should
    # retrieve a new copy of the object if up-to-date values are required
    for k,v in kwargs.iteritems():
        if isinstance(v, ExpressionNode):
            v = resolve_expression_node(instance, v)
        setattr(instance, k, v)

    # If you use an ORM cache, make sure to invalidate the instance!
    #cache.set(djangocache.get_cache_key(instance=instance), None, 5)
    return rows_affected
```

这个update方法主要是实现了原子级别的update。用法很简单，把你需要修改的字段和内容作为**kwargs传进去调用即可。这样就避免了使用save时产生的一些不必要的修改。

把前面例子中的task里面的save方法改写成update，问题就能得到解决了。

```
update(user, age=age)
update(user, name=name)
```

Django新版本对这样的情景提供的支持

其实在Django 1.5以后的版本，已经对这样的情景提供了支持：

https://docs.djangoproject.com/en/1.6/ref/models/instances/#specifying-which-fields-to-save

在save方法里面新增加了一个update_fields参数。这样就可以只修改特定字段了：

```
user.name = name
user.save(update_fields=['name'])
```

这样便有效避免了并行save产生的数据冲突。不过笔者目前的生产环境还是1.3，所以还是暂时用之前的update方法顶着了。

思考：为什么Django不自动追踪哪些字段被修改过？

看过以上的这些内容，可能有很多人和我一样有一个疑问，为什么Django不自动追踪每一个instance被修改过的字段，然后在调用save方法的时候，只把这些被修改过的字段组合到UPDATE语句里去呢？这样就解决了并发修改数据冲突的问题。

通过一些搜索，我在django的ticket历史里面找到了一些讨论：

https://code.djangoproject.com/ticket/4102

这个ticket从最早的patch讨论到现在已经七年了！！！！主要是关于为save方法提供自定义的update_fields字段的。在最后的两个comment有提到另外一个项目 django-save-the-change ，这个项目实现了我们想要的自动只save修改过的字段过的功能。

下面这个ticket里面，这个项目作者对于为什么这个项目没有被加入django作为默认的feature发表了一些看法。

https://github.com/karanlyons/django-save-the-change/issues/1

有兴趣的同学可自行阅读。