如何优化 Django REST Framework 的性能? (http://blog.oneapm.com/apm-tech/304.html)

王志利 (http://blog.oneapm.com/author-43.html) 2015-12-14

现在，Django REST Framework (http://www.django-rest-framework.org/) 帮助 Django (http://www.oneapm.com/ai/python/django.html) 开发者给自己的应用开发出简单而强大的标准 REST APIs，而且现在我们已经成功的应用在了一些 Django Web 项目上。

然而，看似简单直观的 Django REST Framework 及其嵌套序列化 (http://www.django-rest-framework.org/api-guide/relations/#nested-relationships)可能会大大降低你的 API 端的性能。你的服务器的其他部分的响应能力也会被某一个低效的 REST API 拖后腿。

问题的根源就是 **「N+1 selects problem」**；首先查询数据库一次得到表中的数据（例如， `Customers`），然后每个用户的其他字段又需要循环不止一次地查询数据库（例如 `customer.country.Name`）。使用 Django 的 ORM，很容易造成这个问题，而使用 DRF，同样会造成这个问题。

幸运的是，目前有修复 Django REST Framework 性能问题的解决方法，而且不需要对代码进行重大重组。它只是需要使用未充分利用的 `select_related` 和 `prefetch_related` 方法来执行所谓的「预加载」。

这种方法产生了很好的效果，在我们最近的项目中，有个重要的 API 调用花费了5-10秒钟才返回结果。运用合适的预加载后，同样的调用时间花费都变的低于1秒，**典型的加速20倍以上。**

为什么 Django REST Framework 那么容易造成这个问题

当你建立一个 DRF 视图时，你经常需要从多个相关表中返回相应的数据。写这样的功能是很简单的，DRF文档中有详细的介绍。不过不幸的是，**只要你在序列化中使用嵌套关系，你就在拿你的性能开玩笑**，像很多的性能问题一样，它往往只出现有大型数据集的**真实生产环境**中。

这种情况发生就是因为 Django 的 ORM 是惰性的，它只取出当前查询所需响应最小的数据。它不知道你是否有成百上千的相同或相似的数据也需要取出来。

况且如今，当我们谈到数据库型网站时，一般情况下，最重要的响应指标就是**数据库的访问次数。**

在 DRF 视图中，我们每次序列化有嵌套关系的数据时都会出现问题，如下面的例子：

```
class CustomerSerializer(serializers.ModelSerializer):
    # This can kill performance!
    order_descriptions = serializers.StringRelatedField(many=True)
    # So can this, same exact problem...
    orders = OrderSerializer(many=True, read_only=True) # This can kill performance!
```

CustomerSerializer 函数里面是这么运行的：

- 获取所有的 `customers` (需要往返到数据库)
- 对于第一个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 对于第二个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 对于第三个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 对于第四个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 对于第五个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 对于第六个返回的客户，获取他们的 `orders` (又需要去往返一趟数据库)
- 。。。。终于意识到，**千万不要有更多的用户**

而且这个很快就变的更糟，如果你 `OrderSerializer` 本身又有一个嵌套关系，那你就有了嵌套循环，那么即使数据量很小，你也很快就陷入麻烦之中。这有一个经验，一个拥有适度流量的网站，在进入真正的麻烦前也就可以负担起50次到数据库的访问。

解决 Django 「懒惰」的基本方法

现在我们解决这个问题的方法就是「预加载」。从本质上讲，就是你提前警告 Django ORM 你要一遍又一遍的告诉它同样无聊的指令。在上面的例子中，在 DRF 开始获取前很简单地加上这句话就搞定了：

```
queryset = queryset.prefetch_related('orders')
```

当 DRF 调用上述相同序列化 customers 时，出现的是这种情况：

- 获取所有 `customers` （执行两个往返数据库操作，第一个是获取 customers，第二个获取相关 customers 的所有相关的 orders。）

- 对于第一个返回的 customers，获取其 `order` （不需要访问数据库，我们已经在上一步中获取了所需要的数据）
- 对于第二个返回的 customers，获取其 `order` （不需要访问数据库）
- 对于第三个返回的 customers，获取其 `order` （不需要访问数据库）
- 对于第四个返回的 customers，获取其 `order` （不需要访问数据库）
- 对于第五个返回的 customers，获取其 `order` （不需要访问数据库）
- 对于第六个返回的 customers，获取其 `order` （不需要访问数据库）
- 你又意识到，你可以有了**很多 customers**，已经不需要再继续等待去数据库。

其实 Django ORM 的「预备」是在第1步进行请求，它在本地高速缓存的数据能够提供步骤2+所要求的数据。与之前往返数据库相比从本地缓存数据中读取数据基本上是瞬时的，所以我们在有很多 customers 时就获得了巨大的性能加速。

解决 Django REST Framework 性能问题的标准化模式

我们已经确定了一个优化 Django REST Framework 性能问题的通用模式，那就是每当序列化查询嵌套字段时，我们就添加一个新的 `@staticmethod` 名叫 `setup_eager_loading`，像这样：

```
class CustomerSerializer(serializers.ModelSerializer):
    orders = OrderSerializer(many=True, read_only=True)

    def setup_eager_loading(cls, queryset):
        """ Perform necessary eager loading of data. """
        queryset = queryset.prefetch_related('orders')
        return queryset
```

这样，不管哪里要用到这个序列化，都只需在调用序列化前简单调用 `setup_eager_loading` ，就像这样：

```
customer_qs = Customers.objects.all()
customer_qs = CustomerSerializer.setup_eager_loading(customer_qs)  # Set up eager loading to avoid N+1 selects
post_data = CustomerSerializer(customer_qs, many=True).data
```

或者，如果你有一个 `APIView` 或 `ViewSet`，你可以在 `get_queryset` 方法里调用 `setup_eager_loading`：

```
def get_queryset(self):
    queryset = Customers.objects.all()
    # Set up eager loading to avoid N+1 selects
    queryset = self.get_serializer_class().setup_eager_loading(queryset)
    return queryset
```


那么怎样编写 `setup_eager_loading`

想要解决 Django 的性能问题，最困难的部分就是要熟悉 `select_related` 和它的小伙伴儿们是如何工作的。在这儿，我们将详细介绍它们在 Django ORM 和 Django REST Framework 中怎样使用。

- `select_related`： Django ORM 最简单的预加载工具，对于所有**一对一**或**多对一**的数据关系，你都需要从同一个父对象获取数据，如客户的公司名称。这个会被翻译成 SQL 的 join 操作，这样父对象的数据就和子对象的数据一起取回来了。（参见官方文档 (https://docs.djangoproject.com/en/dev/ref/models/querysets/#django.db.models.query.QuerySet.select_related)）
- `prefetch_related`： 对于更复杂的关系，即每个结果有多行（例如 many=True ），像**多对一**或**多对多**的数据关系，比如上述客户的订单，这转化一个二级 SQL 查询，通常有很长的 `WHERE ... IN` ，从中只选择相关的行。（参见官方文档 (https://docs.djangoproject.com/en/dev/ref/models/querysets/#django.db.models.query.QuerySet.select_related)）
- `Prefetch`： 用于复杂 `prefetch_related` 查询，例如过滤子集。它也可以嵌套 `setup_eager_loading` 进行调用。 （参见官方文档 (https://docs.djangoproject.com/en/dev/ref/models/querysets/#django.db.models.query.QuerySet.select_related)）

一个合适的预加载模型

在本例中，我们优化一个假想的活动规划网站（巧了，我们正在进行的项目 getfetcher.com 就是这样）的 Django REST Framework 相关的性能问题，我们有一个简单的数据库结构：

```
from django.contrib.auth.models import User

class Event:
    """ A single occasion that has many `attendees` from a number of organizations."""
    creator = models.ForeignKey(User)
    name = models.TextField()
    event_date = models.DateTimeField()

class Attendee:
    """ A party-goer who (usually) represents an `organization`, who may attend many `events`."""
    events = models.ManyToManyField(Event, related_name='attendees')
    organization = models.ForeignKey(Organization, null=True)

class Organization:
    name = models.TextField()
```

在这个例子中，为了获取所有的事件，我们的预加载代码长这样：

```
class EventSerializer(serializers.ModelSerializer):
    creator = serializers.StringRelatedField()
    attendees = AttendeeSerializer(many=True)
    unaffiliated_attendees = AttendeeSerializer(many=True)

    @staticmethod
    def setup_eager_loading(queryset):
        """ Perform necessary eager loading of data. """
        # select_related for "to-one" relationships
        queryset = queryset.select_related('creator')

        # prefetch_related for "to-many" relationships
        queryset = queryset.prefetch_related(
            'attendees',
            'attendees__organization')

        # Prefetch for subsets of relationships
        queryset = queryset.prefetch_related(
            Prefetch('unaffiliated_attendees',
                queryset=Attendee.objects.filter(organization__isnull=True))
            )
        return queryset
```

当我们使用 EventSerializer 之前确定调用 `setup_eager_loading`，这样我们只会有两个大的查询，而不是 N+1 个小的查询，而且我们的表现通常会好很多！

结论

预加载是 Django REST Framework 通用的性能优化 (http://www.oneapm.com/brand/apm.html)方式，OneAPM Python Agent (http://www.oneapm.com/ai/python.html?utm_source=Community&utm_medium=Article&utm_term=%E5%A6%82%E4%BD%95%E4%BC%98%E5%8C%96%20Django%20REST) 也是国内通用的性能监控工具，而且支持 Django REST Framework，下面是一个简单 demo 的截图。

任何时候你通过 ORM 查询嵌套关系时，你都应该考虑建立适合的预加载。根据现有经验，这是现在小型和中型 Web 开发中最常见的性能相关问题。

参考

- Django REST Framework (http://www.django-rest-framework.org) 文档
- Github上的问题来自动执行 prefetch_related：Automatically determine `select_related` and `prefetch_related` on ModelSerializer (https://github.com/tomchristie/django-rest-framework/issues/1964)。
- DRF 的作者 Tom Christie，在 DRF 性能这篇博客文章中谈到了我们上面处理的问题，Get your ORM lookups right (http://www.dabapps.com/blog/api-performance-profiling-django-rest-framework/)

原文链接：Optimizing slow Django REST Framework performance (https://ses4j.github.io/2015/11/23/optimizing-slow-django-rest-framework-performance/)