使用缓存

本篇主要介绍如何在在Django REST framework 中使用缓存，以及为什么要使用缓存、有什么优势。

一、为什么要使用缓存?

在C/S架构中，用户通过浏览器向服务器发送请求，前端会向后端请求数据进行页面展示。

而对于经常要使用到的数据，而同时这部分数据可能是不经常发生改变的数据，前端会不断的向后端发送请求，后端再不断的对数据库进行查询。

这样的话，查询的过程中会消耗一定的时间，同时也会加大数据库的性能压力，那么有没有更好的办法呢?

那就是**使用缓存。**

我们将前端第一次请求响应的数据，放置到缓存中，那么这一定的时间内前端再次进行请求数据时，后端直接将缓存中的数据返回给前端。

这样做的优点:

1. 减少数据库的查询次数，提交查询效率
2. 从缓存中读取数据的速度，要比从数据库中读取数据快得多。

二、使用Redis作为缓存对象

为什么使用Redis作为缓存对象呢?

1、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

2、Redis支持master-slave(主-从)模式应用，可以将数据复制到任意数量的从机中。

3、Redis支持数据持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。（断电的情况等。）

4、Redis的读写速度异常快。

5、Redis支持设置存储的过期时间，避免数据存储的冗余。

如何在Django中将redis设置为默认的缓存?

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1::6379/1",,
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    "session": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1::6379/1",,
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
}
```

可以在Django的配置文件的CACHES项中指明：

1. 缓存后端使用的是何种缓存 -- "BACKEND": "django_redis.cache.RedisCache"，这里指明为Redis缓存。
2. 指明缓存的地址即Redis的地址： "LOCATION": "redis://127.0.0.1::6379/1"。
3. 选项 -- 指明作为redis客户端连接redis服务器所使用的类："CLIENT_CLASS": "django_redis.client.DefaultClient"。

注：我们使用 **django-redis** 提供了**get_redis_connection**的方法，通过调用get_redis_connection方法传递redis的配置名称可获取到redis的连接对象，通过redis连接对象可以执行redis命令。

这里不是我们今天介绍的主题。

三、Django中将数据设置在缓存中

若要将经常被请求的数据且不经常更改的数据存储在缓存中，可以引入DRF框架的一个扩展：

```
pip install drf-extensions
```

其使用方法有以下两种：

1. 直接添加装饰器
2. 使用drf-extensions提供的扩展类

一、直接添加装饰器：

可以在使用rest_framework_extensions.cache.decorators中的cache_response装饰器来装饰返回数据的类视图的对象方法，如

```
class CityView(views.APIView):
    @cache_response()
    def get(self, request, *args, **kwargs):
        ...
```

注：cache_response装饰器可以接受两个参数，若不传，则默认会使用配置文件中的默认配置。

```
@cache_response(timeout=60*60, cache='default')
```

- timeout 缓存时间 （对于缓存而言肯定需要设置过期时间）
- cache 缓存使用的Django缓存后端（即CACHES配置中的键名称），上面提到的。

如果在使用cache_response装饰器时未指明timeout或者cache参数，则会使用配置文件中的默认配置，可以通过如下方法指明：

```
# DRF扩展
REST_FRAMEWORK_EXTENSIONS = {
    # 缓存时间
    'DEFAULT_CACHE_RESPONSE_TIMEOUT': 60 * 60,
    # 缓存存储
    'DEFAULT_USE_CACHE': 'default',
}
```

- DEFAULT_CACHE_RESPONSE_TIMEOUT --- 缓存有效期，单位秒

- DEFAULT_USE_CACHE ---- 缓存的存储方式，与配置文件中的`CACHES`的键对应

注：cache_response装饰器既可以装饰在类视图中的get方法上，也可以装饰在REST framework扩展类提供的list或retrieve方法上。使用cache_response装饰器无需使用method_decorator进行转换。

二、使用drf-extensions提供的扩展类

以下三个扩展类都是在`rest_framework_extensions.cache.mixins`中。

- ListCacheResponseMixin

  用于缓存返回列表数据的视图，与ListModelMixin扩展类配合使用，实际是为list方法添加了cache_response装饰器

- RetrieveCacheResponseMixin

  用于缓存返回单一数据的视图，与RetrieveModelMixin扩展类配合使用，实际是为retrieve方法添加了cache_response装饰器

- CacheResponseMixin

  为视图集同时补充List和Retrieve两种缓存，与ListModelMixin和RetrieveModelMixin一起配合使用。

例如：

```
from rest_framework_extensions.cache.mixins import CacheResponseMixin

class AreasViewSet(CacheResponseMixin, ReadOnlyModelViewSet):
    '''给list/retrieve视图返回的数据添加到缓存'''
    pass


class AddressViewSet(ListCacheResponseMixin,viewsets.ViewSet):
      '''给list视图添加cache_response，即用于返回的列表数据 添加到缓存'''

    def list(self, request):
        ...

    def retrieve(self, request, pk=None):
        ...
```

over~~~，本篇主要提供一个思想，若常用的数据且不常发生变更的数据，为了减少数据库的查询次数和提升查询速度 可以将第一次响应的数据添加到缓存中，而对于这类数据作为缓存对象的最好选择便是--- Redis。

在我们web制作过程中进场使用到缓存技术，那么今天python中的一种简单便捷的缓存技术与大家分享；
以下是在省/市/区三级联动的地址查询做的缓存处理

使用缓存
省市区的数据是经常被用户查询使用的，而且数据基本不变化，所以我们可以将省市区数据进行缓存处理，减少数据库的查询次数。

在Django REST framework中使用缓存，可以通过drf-extensions扩展来实现。

关于扩展使用缓存的文档，可参考链接http://chibisov.github.io/drf-extensions/docs/#caching

安装
pip install drf-extensions
使用方法
1） 直接添加装饰器

可以在使用rest_framework_extensions.cache.decorators中的cache_response装饰器来装饰返回数据的类视图的对象方法，如

class CityView(views.APIView):
@cache_response()
def get(self, request, *args, **kwargs):
...
cache_response装饰器可以接收两个参数

@cache_response(timeout=60*60, cache='default')
timeout 缓存时间
cache 缓存使用的Django缓存后端（即CACHES配置中的键名称）
如果在使用cache_response装饰器时未指明timeout或者cache参数，则会使用配置文件中的默认配置，可以通过如下方法指明：

DRF扩展
REST_FRAMEWORK_EXTENSIONS = {

缓存时间

'DEFAULT_CACHE_RESPONSE_TIMEOUT': 60 * 60,

缓存存储

'DEFAULT_USE_CACHE': 'default',
}
DEFAULT_CACHE_RESPONSE_TIMEOUT 缓存有效期，单位秒
DEFAULT_USE_CACHE 缓存的存储方式，与配置文件中的CACHES的键对应。
注意，cache_response装饰器既可以装饰在类视图中的get方法上，也可以装饰在REST framework扩展类提供的list或retrieve方法上。使用cache_response装饰器无需使用method_decorator进行转换。

2）使用drf-extensions提供的扩展类

drf-extensions扩展对于缓存提供了三个扩展类：

ListCacheResponseMixin

用于缓存返回列表数据的视图，与ListModelMixin扩展类配合使用，实际是为list方法添加了cache_response装饰器

RetrieveCacheResponseMixin

用于缓存返回单一数据的视图，与RetrieveModelMixin扩展类配合使用，实际是为retrieve方法添加了cache_response装饰器

CacheResponseMixin

为视图集同时补充List和Retrieve两种缓存，与ListModelMixin和RetrieveModelMixin一起配合使用。

三个扩展类都是在rest_framework_extensions.cache.mixins中。

为省市区视图添加缓存
因为省市区视图使用了视图集，并且视图集中有提供ListModelMixin和RetrieveModelMixin的扩展（由ReadOnlyModelViewSet提供），所以可以直接添加CacheResponseMixin扩展类。

修改返回省市区信息的视图

```
from rest_framework_extensions.cache.mixins import CacheResponseMixin

class AreasViewSet(CacheResponseMixin, ReadOnlyModelViewSet):
    """
    行政区划信息
    """
    pagination_class = None # 区划信息不分页

    def get_queryset(self):
        """
        提供数据集
        """
        if self.action == 'list':
            return Area.objects.filter(parent=None)
        else:
            return Area.objects.all()

    def get_serializer_class(self):
        """
        提供序列化器
        """
        if self.action == 'list':
            return AreaSerializer
        else:
            return SubAreaSerializer
```

缓存数据保存位置与有效期的设置
我们想把缓存数据保存在redis中，且设置有效期，可以通过在配置文件中定义的方式来实现。

在配置文件中增加

```
缓存配置
REST_FRAMEWORK_EXTENSIONS = {

    缓存时间

    'DEFAULT_CACHE_RESPONSE_TIMEOUT': 60 * 60,

    缓存存储

    'DEFAULT_USE_CACHE': 'default',
}
```

Django REST framework 缓存技术

对于经常被用户查询使用的，而且数据基本不变化的数据，我们可以使用缓存处理，减少数据库的查询次数。
在Django REST framework中使用缓存，可以通过drf-extensions扩展来实现。

安装
pip install drf-extensions

使用方法
1、直接添加装饰器
使用rest_framework_extensions.cache.decorators中的cache_response装饰器来装饰返回数据的类视图的对象方法，如：

```
class CityView(APIView):
    @cache_response()
    def get(self, request, *args, **kwargs):
    ...
```

cache_response装饰器可以接收两个参数：

timeout 缓存时间
cache 缓存使用的Django缓存后端（即CACHES配置中的键名称）
方式一：局部使用：

```
@cache_response(timeout=60*60, cache='default')
```

方式二：全局配置

```
REST_FRAMEWORK_EXTENSIONS = { # DRF扩展
'DEFAULT_CACHE_RESPONSE_TIMEOUT': 60 * 60, # 缓存时间
'DEFAULT_USE_CACHE': 'default', # 缓存存储
}
```

cache_response装饰器中就无需再添加参数了

2、使用drf-extensions提供的扩展类

drf-extensions扩展对于缓存提供了三个扩展类：
三个扩展类都是在rest_framework_extensions.cache.mixins中。

ListCacheResponseMixin
用于缓存返回列表数据的视图，与ListModelMixin扩展类配合使用，实际是为list方法添加了cache_response装饰器

RetrieveCacheResponseMixin
用于缓存返回单一数据的视图，与RetrieveModelMixin扩展类配合使用，实际是为retrieve方法添加了cache_response装饰器

CacheResponseMixin
为视图集同时补充List和Retrieve两种缓存，与ListModelMixin和RetrieveModelMixin一起配合使用

使用方式

```
class CityView(CacheResponseMixin，APIView):
    def get(self, request, *args, **kwargs):
        ...
```