6. django配置缓存

一. 为什么使用缓存?
    在Django中, 请求到达视图后, 视图会从数据库取数据放到模板中进行动态渲染, 渲染后的结果就是用户看到的html页面. 但是如果每次请求都从数据库取数据并渲染, 将极大降低性能, 不仅服务器压力大, 而且客户端也无法即时获得响应. 如果能将渲染后的结果放到速度更快的缓存中, 每次有请求过来, 先检查缓存中是否有对应的资源, 如果有那么久直接从缓存中取出来返回响应, 节省取数据和渲染的时间, 不仅能大大提高系统性能, 还能提高用户体验; 另外, 缓存只是一类统称, 一般其介质是速度很快的内存, 但也可以是能加快数据读取的其它方式. 对页面实时性要求不高的页面, 可以用缓存; 比如博客文章, 假设作者一天更新一篇文章, 那么可以为博客服务器设置1天的缓存, 一天后会刷新.

二. django提供的缓存方式:
Django中提供了多种常用的缓存方式, 如(内存 本地磁盘 memcache), 如果要使用缓存, 需要现在settings.py中进行配置, 然后再应用. 我们分别以内存和文件的形式来定义缓存.

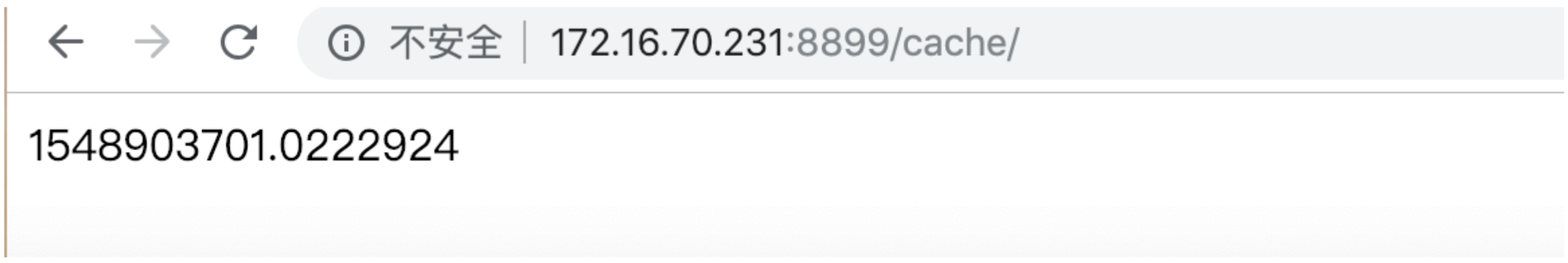
2.1 内存

2.1.1 修改项目settings.py文件
```
   CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
        }
    }
```

2.2.2 定义views视图函数
```
    import time
    def cache(request):
        x = time.time()
        return HttpResponse(x)
```

2.2.3 将views视图函数与urls相关联
```
    urlpatterns = [
        path('admin/', admin.site.urls),
        path('', include('login.urls')),
        path('addclass/', login_views.AddClass.as_view()),
        path('upload/', login_views.file_upload),
        path('cache/', login_views.cache),  #新增
        ]
```

2.2.4 验证



未对视图应用缓存时, 每刷新一次页面, 页面中显示的时间戳就变化一次.

2.2.5 为视图函数添加缓存装饰器. 将views视图函数变更为如下:
```
    import time
    from django.views.decorators.cache import cache_page
    @cache_page(5)
    def cache(request):
        x = time.time()
        return HttpResponse(x)
```

2.2.6 刷新网页验证会发现, 网页每5秒更新一次, 5秒之内对URL的请求返回的内容都是相同的.

2.2 使用本地磁盘作为缓存, 项目settings.py中的配置变为.
```
    CACHES = {
        'default': {
            'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
            'LOCATION': '/tmp/cache',
            }
    }
```

三. 在CBV中应用缓存, 写法为
```
    from django.views.decorators.cache import cache_page
    from django.utils.decorators import method_decorator
    class c1:
        @method_decorator(cache_page(300))
        def get(self, request):
            pass
```

四. 注意.

django只缓存http的get与head请求, 查看django源码如下:

源代码路径为:
~lib/python3.6/site-packages/django/middleware/cache.py
```
class FetchFromCacheMiddleware(MiddlewareMixin):
    """
    Request-phase cache middleware that fetches a page from the cache.

    Must be used as part of the two-part update/fetch cache middleware.
    FetchFromCacheMiddleware must be the last piece of middleware in MIDDLEWARE
    so that it'll get called last during the request phase.
    """
    def __init__(self, get_response=None):
        self.key_prefix = settings.CACHE_MIDDLEWARE_KEY_PREFIX
        self.cache_alias = settings.CACHE_MIDDLEWARE_ALIAS
        self.cache = caches[self.cache_alias]
        self.get_response = get_response

    def process_request(self, request):
        """
        Check whether the page is already cached and return the cached
        version if available.
        """
        if request.method not in ('GET', 'HEAD'):
            request._cache_update_cache = False
            return None  # Don't bother checking the cache.

        # try and get the cached GET response
        cache_key = get_cache_key(request, self.key_prefix, 'GET', cache=self.cache)
        if cache_key is None:
            request._cache_update_cache = True
            return None  # No cache information available, need to rebuild.
        response = self.cache.get(cache_key)
        # if it wasn't found and we are looking for a HEAD, try looking just for that
```

```
        if response is None and request.method == 'HEAD':
            cache_key = get_cache_key(request, self.key_prefix, 'HEAD', cache=self.cache)
            response = self.cache.get(cache_key)

        if response is None:
            request._cache_update_cache = True
            return None  # No cache information available, need to rebuild.

        # hit, return cached response
        request._cache_update_cache = False
        return response
```