Django中的缓存机制及其实现方法

在本教程的《Django Cache缓存系统》一节中，我们对 Django 缓存系统做了基本的介绍，那么它在 Django 中是如何进行应用呢，在本节将对 Django 的缓存系统展开详细的描述，从它的缓存机制、使用场景以及如何进行配置，在本节中你将认识到什么是强缓存、什么是协商缓存，最后我们通过实际举例对缓存的使用进行说明，下面我们就正式开始本节的内容。

1. 缓存机制的分类

我们已经了解了 Django 为什么使用缓存的原因，即避免服务器通对高频率的同样请求，反复的进行计算并去数据库获取相应的数据，这样容易导致服务器过载，并浪费服务器资源。Django 的缓存很好的节解决了这一问题。

1) MemCache缓存实现流程

缓存将一个某个 views 视图函数的返回值保存至内存或者 memcached 中，若在时间约定范围内该用户又对此视图发起了请求，则不再去执行 views 中的操作，而是直接从内存或者 Redis 中获取之前已经缓存的数据，并将其返回给浏览器，这也是动态网站使用缓存的常用流程。

这里须需要大家注意，Memcached 不是 Django 自带的模块，而是需要你自己安装、配置和启动服务；Memcached 安装后，用 Python 操作 Memcached 的依赖库，最常用的是 python-memcached 和 pylibmc。但是现在也多了另外一种选择也就是 Redis，它是基于内存的缓存型数据库，同时支持数据序列化相比而言更加安全，我们推荐使用。

Redis 的工作流程总结如下：先检查客户端的请求数据是否在 Redis 中，如有，直接把请求数据返回，不再对数据库进行任何操作；如果请求的数据不在 Redis 中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到 Redis 中；每次更新数据库的同时更新 Redis 中的数据，保证与数据库的一致性。

2) Django中的缓存机制

Django 中提供多种缓存机制，如需使用需要在 settings.py 文件中进行配置，Django 提供了六种常用的缓存机制，如下所示：

- 开发调试缓存
- 本地内存缓存
- 数据库缓存
- 文件缓存
- Memcache缓存（使用python-memcached模块）
- Memcache缓存（使用pylibmc模块）

在这里我们对几种缓存机制进行介绍，分别是数据库缓存、文件缓存、本地内存缓存，下面我们介绍它们在 settings.py 文件中是如何进行配置的，数据库缓存的配置如下所示：

```
#数据缓存机制
CACHES = {
    'default': {
        'BACKEND':'django.core.cache.backends.db.DatabaseCache', #数据库引擎
        'TIMEOUT':300, #缓存超时时间（默认300秒，None表示永不过期，0表示立即过期）
        'LOCATION': 'my_cache_table',
        'MAX_ENTRIES': 3,  # 当前最大缓存数
        'CULL_FREQUENCY': 3,  # 缓存到达最大个数之后，剔除缓存个数的比例，即 1/CULL_FREQUENCY（默认3）
    }
}
```

这里还有最红要的一步操作就是创建缓存数据表，使用如下命令：

python manage.py createcachetable my_cache_table

基于本教程中使用数据库进行缓存其实是非常鸡肋的，因为我们之所以使用缓存就是为了减少数据库的查询，但是企业环境下的数据库如果非常高速、高效，那么你可以使用这种机制，我们在这里简单讲解一下它的基本配置。大家注意各种缓存机制在 settings.py 文件的配置都是相差无几的。文件系统缓存的配置如下所示：

```
#基于window
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': 'c:/foo/bar', #若是 Linux 路径写为 /home/cnet/cachetest
    }
}
```

文件缓存机制，也是 Django 为我们提供的一种执行缓存的方法，不过这种方法相较数据库而言更慢，不过有总比没有强，在你无路可走的时候，为了实现达到缓存的目的也可以使用。

下面我们介绍最后一种也就是基于本地内存的缓存机制，大家知道内存的交互读写速度非常之快，所以内存资源也相当的珍贵，使用这种缓存机制也再合适不过了。

```
# 此缓存将内容保存至内存的变量中
CACHES = {
    'default':{
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    'LOCATION': 'unique-snowflake',#给缓存放置的内存区设置一个名字
    }
}
```

2. Django中使用缓存的方式

在 Django 中我们可以使用如下方式直接对缓存进行应用，前两者是通过装饰器实现的，最后一个是通过模板标签实现的：

- 在视图View中使用
- 在路由URL中使用
- 在模板中使用

它们的使用方式分别如下所示：

```
from django.views.decorators.cache import cache_page
@cache_page(60*30)#以秒为单位
def my_view(request):
    ...
```

在 django.views.decorators.cache 定义了一个自动缓存视图响应结果的装饰器 cache_page，这个装饰器接受一个参数：timeout 以秒为单位。在上例中，my_view() 视图的结果将被缓存 30 分钟，也可以写为 60*30。在路由中使用缓存的格式如下：

```
from django.views.decorators.cache import cache_page

urlpatterns = [
    path('foo/', cache_page(60)(my_view)),
]
```

在模板中也可以使用缓存，也叫模板碎片缓存，如下所示：

```
1. {% load cache %}
2. {% cache 500 bar request.user.username %}
3.     .. bar for logged in user ..
4. {% endcache %}
```

我们要在模版的顶部位置添加 {% load cache %}，同样它也需要 {% endcache %} 结尾。模板标签 {% cache %} 将在设定的时间内，缓存标签块中包含的内容。它最少需要两个参数：依次为缓存时间（以秒为单位）以及要被缓存的片段起的名称（bar）。

3. Django缓存的实例应用

1) 视图**函数中使用装**饰**器**

通过上面的讲解我们对 Django 缓存机制与实现方法有了深入的了解，下面我们通过简单的例子进行测试，看看它是如何进行应用的。我们通过获取当前时间戳，来证明缓存机制的存在作用。

```
1. #在缓存有效时间内不会阻塞，直到缓存过期重新阻塞3秒
2. @cache_page(60) #缓存有效时间60s
3. def test_cache(request):
4.     t1=time.time() #得到当前时间戳
5.     time.sleep(3) #阻塞三秒
6.     html='t1 is %s'%(t1)
7.     return HttpResponse(html)
```

配置好路由映射关系，然后访问 127.0.0.1:8000/index/test_cache ，发现第一次请求时会阻塞三秒，然后再缓存期间请求页面则不需要阻塞，直到缓存过期重新阻塞。

我们可以通过谷歌浏览器 F12 查看它的响应头，其中的 Cache-Control: max-age=60 代表了最大的过期时间为 60s，它是 HTTP1.1 的产物，而另外一个 Expires 同样代表过期时间，只不过它属于 HTTP1.0 时代，可以把前者看做是后者的补充。

2) 在path中使用cache_page()

同样我们也可以在 url 中使用缓存的装饰器，方法如下所示：

```
1. from django.views.decorators.cache import cache_page
```

```
2. path('test_cache/',cache_page(15)(views.test_cache))
```

使用这种范式的时候，记得要把视图函数中的装饰器取消。

3) 实现**模板中碎片化**缓**存**

编写如下视图函数以及 test_cache.html 页面，如下所示:

```
1. #视图函数
2. def test_time(request):
3.     if request.method=='GET':
4.         return render(request,'index/test_cache.html')
5.     elif request.method=='POST':
6.         t1 = time.time()  # 得到当前时间戳
7.         time.sleep(3)  # 阻塞三秒
8.         return render(request,'index/test_cache.html',locals())
```

test_cache.html 模板页面如下所示：

```
1. <form action="/index/test_time/" method="post">
2.     {% csrf_token %}
3. {% load cache %}
4. {% cache 30 test %}
5. <p>我是缓存的 {{ t1 }}</p>
6. {% endcache %}
7. <h4>
8.     我是没有缓存的{{ t1 }}
9. </h4>
10. <input type="submit" value="提交">
11. </form>
```

配置好路由后，访问 127.0.0.1:8000/index/test_time，点击提交按钮后，你会发现有缓存的 t1 时间戳只有在 30s 以后才会改变，因为在这 30s 内它读取的都是已缓存模板片段内容。