《Django Web框架教学笔记》

目录

缓存

什么是缓存？

缓存是一类可以更快的读取数据的介质统称，也指其它可以加快数据读取的存储方式。一般用来存储临时数据，常用介质的是读取速度很快的内存

为什么使用缓存？

视图渲染有一定成本，对于低频变动的页面可以考虑使用缓存技术，减少实际渲染次数

案例分析

```
from django.shortcuts import render

def index(request):
    # 时间复杂度极高的渲染
    book_list = Book.objects.all()  #-> 此处假设耗时2s
    return render(request, 'index.html', locals())
```

优化思想

```
given a URL, try finding that page in the cache
if the page is in the cache:
    return the cached page
else:
    generate the page
    save the generated page in the cache (for next time)
    return the generated page
```

使用缓存场景：

1，博客列表页

2，电商商品详情页

3，缓存导航及页脚

Django中设置缓存

Django中提供多种缓存方式，如需使用需要在settings.py中进行配置

1,数据库缓存

Django可以将其缓存的数据存储在您的数据库中

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
        'TIMEOUT': 300,  #缓存保存时间 单位秒，默认值为300,
        'OPTIONS':{
            'MAX_ENTRIES': 300, #缓存最大数据条数
            'CULL_FREQUENCY': 2,#缓存条数达到最大值时 删除1/x的缓存数据
        }
    }
}
```

创建缓存表

```
python manage.py createcachetable
```

2,文件系统缓存

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',#这个是文件夹的路径
        #'LOCATION': 'c:\test\cache',#windows下示例
    }
}
```

3, 本地内存缓存

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake'
    }
}
```

Django中使用缓存

- 在视图View中使用
- 在路由URL中使用
- 在模板中使用

在视图View中使用cache

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

@cache_page(30)  -> 单位s
def my_view(request):
    ...
```

在路由中使用

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

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

在模板中使用

```
{% load cache %}
{% cache 500 sidebar request.user.username %}
    .. sidebar for logged in user ..
{% endcache %}
```

- 缓存api

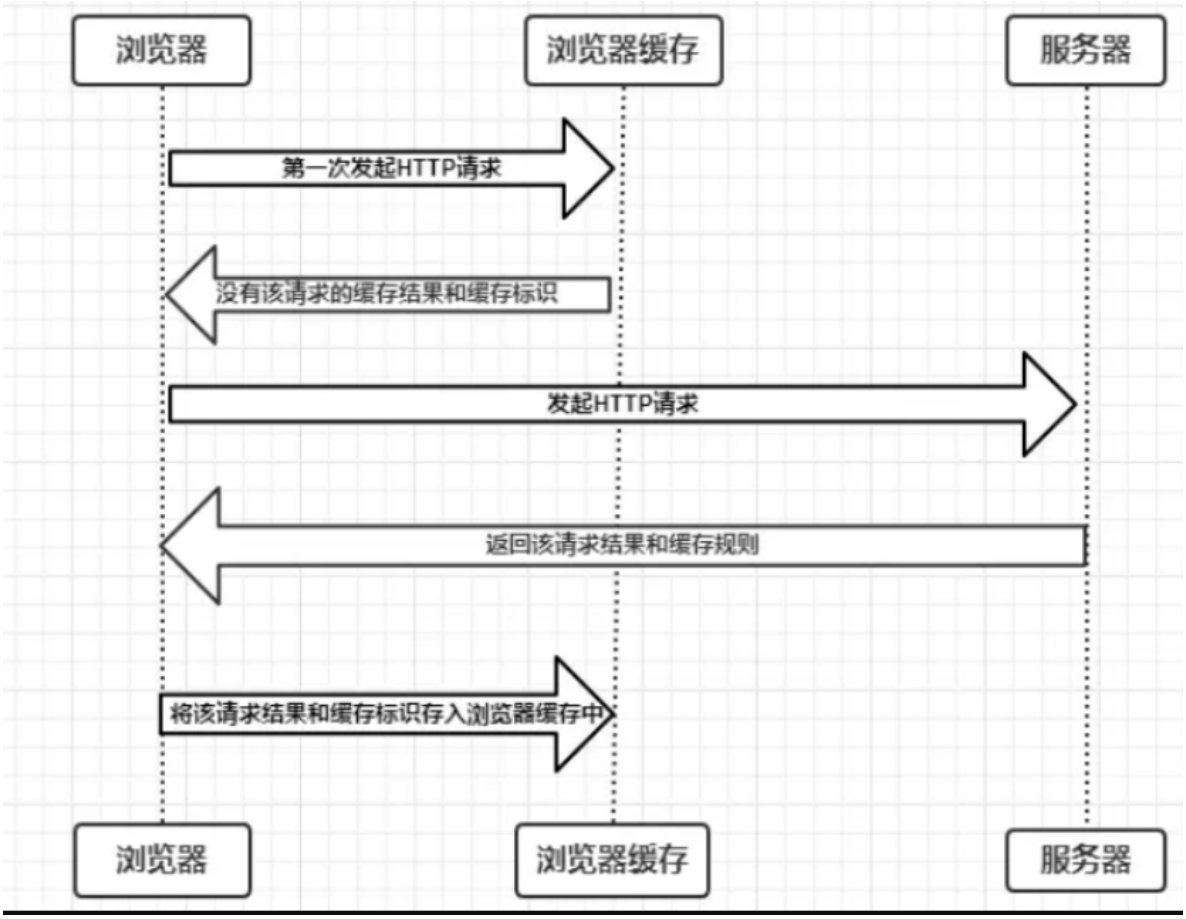
  作用：局部缓存部分结果

  使用：

  ```
  #指定配置引入
  from django.core.cache import caches
  cache1 = caches['myalias']
  cache2 = caches['myalias_2']

  #默认配置引入【指的配置中的default项】 等同于 caches['default']
  from django.core.cache import cache

  #常规命令 set
  #key: 字符串类型
  #value: Python对象
  ```

```
#timeout：缓存存储时间  默认值为settings.py CACHES对应配置的TIMEOUT
#返回值：True
cache.set('my_key', 'myvalue', 30)

#常规命令 get
#返回值：为key的具体值，如果没有数据，则返回None
cache.get('my_key')
#可添加默认值，如果没取到返回默认值
cache.get('my_key', 'default值')

#常规命令 add 只有在key不存在的时候 才能设置成功
#返回值 True or None
cache.add('my_key', 'value') #如果my_key已经存在，则此次赋值失效

#常规命令 get_or_set 如果未获取到数据 则执行set操作
#返回值 key的值
cache.get_or_set('my_key', 'value', 10)

#常规命令 get_many() set_many()
#返回值  set: None
#       get: OrderedDict
>>> cache.set_many({'a': 1, 'b': 2, 'c': 3})
>>> cache.get_many(['a', 'b', 'c'])
{'a': 1, 'b': 2, 'c': 3}

#常规命令 delete
#返回值  成功删除 1   删除失败0
cache.delete('my_key')

#常规命令 delete_many
#返回值  成功删除的数据条数
cache.delete_many(['a', 'b', 'c'])
```

浏览器中的缓存



浏览器缓存分类：

强缓存

不会向服务器发送请求，直接从缓存中读取资源

1，Expires

缓存过期时间，用来指定资源到期的时间，是服务器端的具体的时间点

Expires=max-age + 请求时间

Expires 是 **HTTP/1** 的产物，受限于本地时间，如 果修改了本地时间，可能会造成缓存失效

2, Cache-Control

在HTTP/1.1中，Cache-Control主要用于控制网页缓存。比如当`Cache-Control:max-age=120`代表请求创建时间后的120秒，缓存失效

横向对比 Expires VS Cache-Control

协商缓存

**协商缓存就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程

1，Last-Modified和If-Modified-Since

第一次访问时，服务器会返回

Last-Modified: Fri, 22 Jul 2016 01:47:00 GMT

浏览器下次请求时 携带If-Modified-Since这个header , 该值为 Last-Modified

服务器接收请求后，对比结果，若资源未发生改变，则返回304， 否则返回200并将新资源返回给浏览器

缺点：只能精确到秒，容易发生单秒内多次修改，检测不到

2，ETag和If-None-Match

Etag是服务器响应请求时，返回当前资源文件的一个唯一标识(由服务器生成)，只要资源有变化，Etag就会重新生成

流程同上

对比 Last-Modified VS ETag

1，精度不一样 - Etag 高

2，性能上 - Last-Modified 高

3，优先级 - Etag 高

中间件 **Middleware**

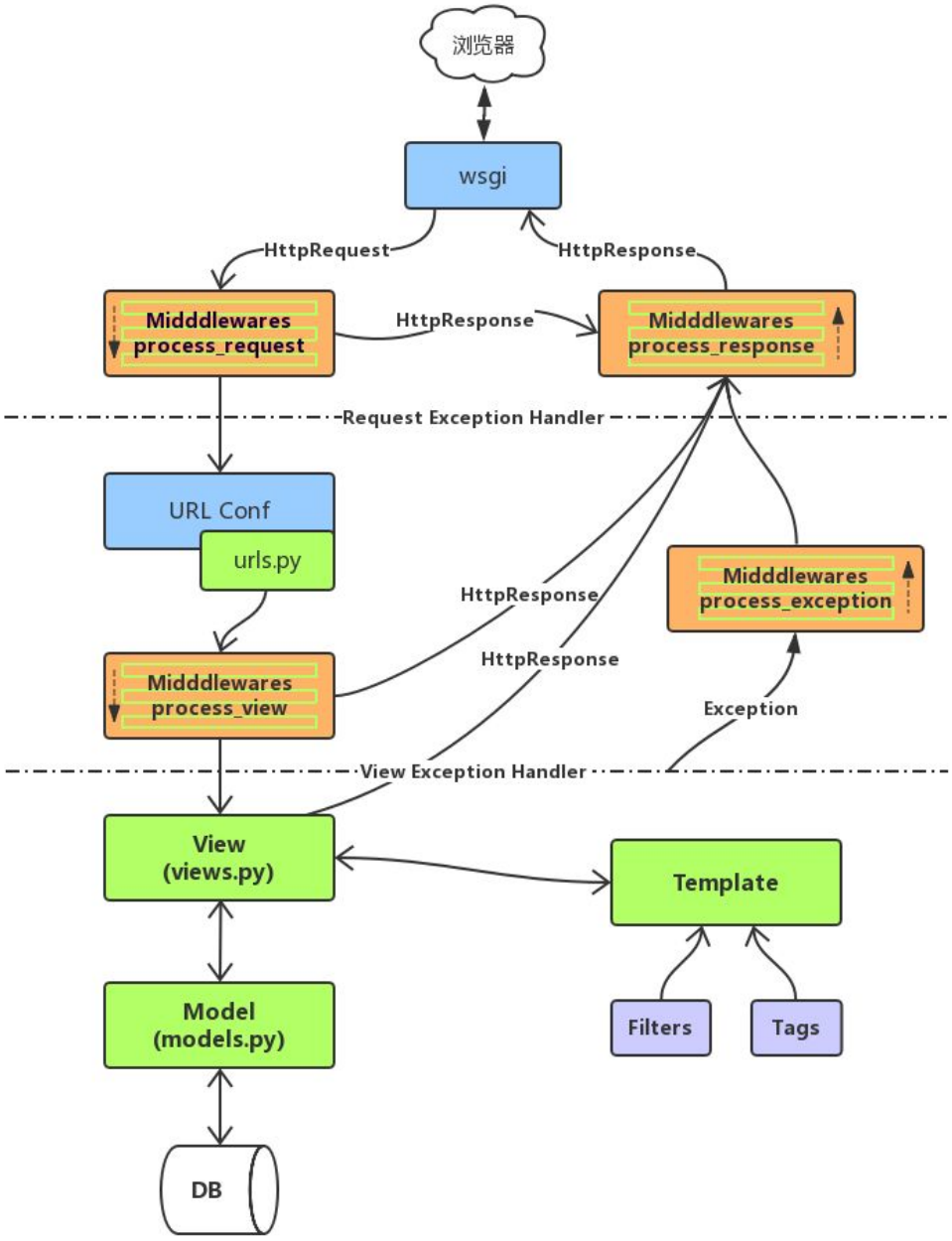
- 中间件是 Django 请求/响应处理的钩子框架。它是一个轻量级的、低级的“插件”系统，用于全局改变 Django 的输入或输出。
- 每个中间件组件负责做一些特定的功能。例如，Django 包含一个中间件组件 AuthenticationMiddleware，它使用会话将用户与请求关联起来。
- 他的文档解释了中间件是如何工作的，如何激活中间件，以及如何编写自己的中间件。Django 具有一些内置的中间件，你可以直接使用。它们被记录在 built-in middleware reference 中。
- 中间件类:
  - 中间件类须继承自 `django.utils.deprecation.MiddlewareMixin` 类
  - 中间件类须实现下列五个方法中的一个或多个:
    - `def process_request(self, request):` 执行路由之前被调用，在每个请求上调用，返回None或HttpResponse对象
    - `def process_view(self, request, callback, callback_args, callback_kwargs):` 调用视图之前被调用，在每个请求上调用，返回None或HttpResponse对象
    - `def process_response(self, request, response):` 所有响应返回浏览器 被调用，在每个请求上调用，返回HttpResponse对象
    - `def process_exception(self, request, exception):` 当处理过程中抛出异常时调用，返回一个HttpResponse对象
    - `def process_template_response(self, request, response):` 在视图刚好执行完毕之后被调用，在每个请求上调用，返回实现了render方法的响应对象
  - 注： 中间件中的大多数方法在返回None时表示忽略当前操作进入下一项事件，当返回HttpResponse对象时表示此请求结束，直接返回给客户端
- 编写中间件类:

```python
# file : middleware/mymiddleware.py
from django.http import HttpResponse, Http404
from django.utils.deprecation import MiddlewareMixin

class MyMiddleWare(MiddlewareMixin):
    def process_request(self, request):
        print("中间件方法 process_request 被调用")

    def process_view(self, request, callback, callback_args, callback_kwargs):
        print("中间件方法 process_view 被调用")

    def process_response(self, request, response):
        print("中间件方法 process_response 被调用")
        return response

    def process_exception(self, request, exception):
        print("中间件方法 process_exception 被调用")

    def process_template_response(self, request, response):
        print("中间件方法 process_template_response 被调用")
        return response
```

- 注册中间件:

```python
# file : settings.py
MIDDLEWARE = [
    ...
       ]
```

- 中间件的执行过程



- 练习
  - 用中间件实现强制某个IP地址只能向/test 发送 5 次GET请求
  - 提示:
    - request.META['REMOTE_ADDR'] 可以得到远程客户端的IP地址
    - request.path_info 可以得到客户端访问的GET请求路由信息
  - 答案:

```
```

跨站请求伪造保护 CSRF

- 跨站请求伪造攻击
  - 某些恶意网站上包含链接、表单按钮或者JavaScript，它们会利用登录过的用户在浏览器中的认证信息试图在你的网站上完成某些操作，这就是跨站请求伪造(CSRF，即Cross-Site Request Forgey)。
- 说明:

- CSRF中间件和模板标签提供对跨站请求伪造简单易用的防护。
- 作用:
  - 不让其它表单提交到此 Django 服务器
- 解决方案:
  1. 取消 csrf 验证(不推荐)
     - 删除 settings.py 中 MIDDLEWARE 中的 `django.middleware.csrf.CsrfViewMiddleware` 的中间件
  2. 通过验证 csrf_token 验证

```
需要在表单中增加一个标签
{% csrf_token %}
```

分页

- 分页是指在web页面有大量数据需要显示，为了阅读方便在每个页页中只显示部分数据。
- 好处:
  1. 方便阅读
  2. 减少数据提取量，减轻服务器压力。
- Django提供了Paginator类可以方便的实现分页功能
- Paginator类位于`django.core.paginator`模块中。

Paginator对象

- 对象的构造方法
  - Paginator(object_list, per_page)
  - 参数
    - object_list 需要分类数据的对象列表
    - per_page 每页数据个数
  - 返回值:
    - 分页对象
- Paginator属性
  - count：需要分类数据的对象总数
  - num_pages：分页后的页面总数
  - page_range：从1开始的range对象, 用于记录当前面码数
  - per_page 每页数据的个数
- Paginator方法
  - Paginator.page(number)
    - 参数 number为页码信息(从1开始)
    - 返回当前number页对应的页信息
    - 如果提供的页码不存在，抛出InvalidPage异常
- Paginator异常exception
  - InvalidPage：当向page()传入一个无效的页码时抛出
  - PageNotAnInteger：当向page()传入一个不是整数的值时抛出
  - EmptyPage：当向page()提供一个有效值，但是那个页面上没有任何对象时抛出

Page对象

- 创建对象
  Paginator对象的page()方法返回Page对象，不需要手动构造
- Page对象属性
  - object_list：当前页上所有数据对象的列表
  - number：当前页的序号，从1开始
  - paginator：当前page对象相关的Paginator对象
- Page对象方法
  - has_next()：如果有下一页返回True
  - has_previous()：如果有上一页返回True
  - has_other_pages()：如果有上一页或下一页返回True
  - next_page_number()：返回下一页的页码，如果下一页不存在，抛出InvalidPage异常
  - previous_page_number()：返回上一页的页码，如果上一页不存在，抛出InvalidPage异常
  - len()：返回当前页面对象的个数
- 说明:
  - Page 对象是可迭代对象,可以用 for 语句来 访问当前页面中的每个对象
- 参考文档https://docs.djangoproject.com/en/1.11/topics/pagination/
- 分页示例:
  - 视图函数

```
from django.core.paginator import Paginator
def book(request):
    bks = models.Book.objects.all()
    paginator = Paginator(bks, 10)
    print('当前对象的总个数是:', paginator.count)
    print('当前对象的面码范围是:', paginator.page_range)
    print('总页数是: ', paginator.num_pages)
    print('每页最大个数:', paginator.per_page)

    cur_page = request.GET.get('page', 1)  # 得到默认的当前页
    page = paginator.page(cur_page)
    return render(request, 'bookstore/book.html', locals())
```

  - 模板设计

```
<html>
<head>
    <title>分页显示</title>
</head>
<body>
{% for b in page %}
    <div>{{ b.title }}</div>
{% endfor %}

{# 分页功能 #}
{# 上一页功能 #}
{% if page.has_previous %}
<a href="{% url 'book' %}?page={{ page.previous_page_number }}">上一页</a>
{% else %}
上一页
{% endif %}
```

```
{% for p in paginator.page_range %}
    {% if p == page.number %}
        {{ p }}
    {% else %}
        <a href="{% url 'book' %}?page={{ p }}">{{ p }}</a>
    {% endif %}
{% endfor %}

{#下一页功能#}
{% if page.has_next %}
<a href="{% url 'book' %}?page={{ page.next_page_number }}">下一页</a>
{% else %}
下一页
{% endif %}
</body>
</html>
```

文件上传

- 文件上传必须为POST提交方式
- 表单`<form>`中文件上传时必须有带有`enctype="multipart/form-data"`时才会包含文件内容数据。
- 表单中用`<input type="file" name="xxx">`标签上传文件
  - 名字`xxx`对应`request.FILES['xxx']`对应的内存缓冲文件流对象。可通能过`request.FILES['xxx']`返回的对象获取上传文件数据
  - `file=request.FILES['xxx']` file 绑定文件流对象，可以通过文件流对象的如下信息获取文件数据
    file.name 文件名
    file.file 文件的字节流数据
- 上传文件的表单书写方式

```
<!-- file: index/templates/index/upload.html -->
<html>
<head>
    <meta charset="utf-8">
    <title>文件上传</title>
</head>
<body>
    <h3>上传文件</h3>
    <form method="post" action="/upload" enctype="multipart/form-data">
        <input type="file" name="myfile"/><br>
        <input type="submit" value="上传">
    </form>
</body>
</html>
```

- 在setting.py 中设置一个变量MEDIA_ROOT 用来记录上传文件的位置

```
# file : settings.py
...
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/files')
```

- 在当前项目文件夹下创建 `static/files` 文件夹

```
$ mkdir -p static/files
```

- 添加路由及对应的处理函数

```
# file urls.py
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^upload', views.upload_view)
]
```

- 上传文件的视图处理函数

```
# file views.py
from django.http import HttpResponse, Http404
from django.conf import settings
import os

def upload_view(request):
    if request.method == 'GET':
        return render(request, 'index/upload.html')
    elif request.method == "POST":
        a_file = request.FILES['myfile']
        print("上传文件名是:", a_file.name)

        filename =os.path.join(settings.MEDIA_ROOT, a_file.name)
        with open(filename, 'wb') as f:
            data = a_file.file.read()
            f.write(data)
        return HttpResponse("接收文件:" + a_file.name + "成功")
    raise Http404
```

- 访问地址: <http://127.0.0.1:8000/static/files/upload.html

文件下载

Django可直接在视图函数中生成**csv**文件 并响应给浏览器

```
import csv
from django.http import HttpResponse
from .models import Book

def make_csv_view(request):
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="mybook.csv"'
    all_book = Book.objects.all()
    writer = csv.writer(response)
    writer.writerow(['id', 'title'])
    for b in all_book:
        writer.writerow([b.id, b.title])

    return response
```

- 响应获得一个特殊的MIME类型*text / csv*。这告诉浏览器该文档是CSV文件，而不是HTML文件
- 响应会获得一个额外的`Content-Disposition`标头，其中包含CSV文件的名称。它将被浏览器用于“另存为...”对话框
- 对于CSV文件中的每一行，调用`writer.writerow`，传递一个可迭代对象，如列表或元组。