

< Teach
Me
Skills />

Занятие 25. Кэширование

1

Кэширование

Кэширование

Кэширование позволяет сохранить результат операции или данные в кэше, чтобы при последующих запросах на ту же операцию или данные можно было быстро получить из кэша, минуя долгий процесс выполнения операции или обращения к базе данных. Это ускоряет время отклика и снижает нагрузку на сервер

На каких уровнях можно реализовать кэширование:

- уровень БД
- уровень HTTP запросов
- уровень шаблонов
- уровень сри-bound вычислений

Проблемы:

- прогревание кэша
- устаревшие данные
- приватность

Частный и приватный кэш

- Пользователь обычно сталкивается с двумя типами кешей: их собственный кэш браузера (частный кэш) и кэш поставщика (общедоступный кэш). Публичный кэш используется несколькими пользователями и контролируется кем-то другим. Это создает проблемы с конфиденциальными данными - например, вы не хотите, чтобы номер вашего банковского счета хранился в общедоступном кэше. Таким образом, веб-приложениям необходим способ сообщать кэшем, какие данные являются частными, а какие общедоступными.

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

@cache_control(private=True)
def my_view(request):
    pass
```

Частный и приватный кэш

```
from django.views.decorators.cache import patch_cache_control
from django.views.decorators.vary import vary_on_cookie

@vary_on_cookie
def list_blog_entries_view(request):
    if request.user.is_anonymous:
        response = render_only_public_entries()
        patch_cache_control(response, public=True)
    else:
        response = render_private_and_public_entries(request.user)
        patch_cache_control(response, private=True)

    return response
```


Кэширование в Django - страницы и фрагменты

1. Кэширование страниц: Django предоставляет декоратор **cache_page**, который можно применять к представлениям для кэширования всей страницы. Это позволяет кэшировать результаты представления и возвращать их напрямую из кэша при последующих запросах, минуя выполнение представления и базы данных.
2. Кэширование фрагментов: Кэширование фрагментов представляет собой кэширование только части страницы, а не всей страницы целиком. В Django вы можете использовать теги шаблонов для кэширования отдельных фрагментов шаблона, таких как боковые панели, списки последних записей и т.д. Это позволяет более гибко управлять кэшированием и обновлять только измененные фрагменты страницы.

Кэширование в Django - шаблоны

Django предоставляет возможность кэшировать отрендеренные шаблоны. Это позволяет сохранить результаты рендеринга сложных шаблонов и избежать их повторного выполнения при каждом запросе

```
{% extends "base.html" %}

{% block content %}
    {% load cache %}

    {% cache 600 posts_cache %}
        {% for post in posts %}
            {% include "includes/post_card.html" %}
        {% endfor %}
    {% endcache %}
{% endblock %}
```

Кэширование в Django - API

Если ваше Django-приложение предоставляет API, вы можете использовать кэширование для хранения результатов запросов к API. Это позволяет уменьшить нагрузку на сервер и ускорить ответы на повторные запросы. Может быть реализовано при помощи **django-rest-framework-cache**

```
class MyAPIView(APIView):  
  
    12 usages (12 dynamic)  
    @cache_response()  
    def get(self, request):  
        ...
```


Кэширование в Django - DB

Django поддерживает использование пакетов, таких как **cacheops**, которые позволяют кэшировать результаты запросов базы данных. Это полезно, когда запросы базы данных занимают значительное время, и результаты этих запросов могут быть закэшированы для повторного использования без необходимости выполнения запроса снова

```
@cached(timeout=60 * 15)  
~~~~~  
class Todo(models.Model):  
    ...
```

```
Todo.objects.all().cache()
```

Бэкенды кэширования

- `django.core.cache.backends.memcached.MemcachedCache`
- `django.core.cache.backends.filebased.FileBasedCache`
- `django.core.cache.backends.db.DatabaseCache`
- `django.core.cache.backends.dummy.DummyCache`
- `django.core.cache.backends.redis.RedisCache`

ПЛЮСЫ И МИНУСЫ

Кэширование cru-bound functions

```
from functools import lru_cache
```

2 usages

```
@lru_cache(maxsize=128)
```

```
def expensive_function(n):
```

```
    print("Performing expensive calculations...")
```

```
    return n * n
```

```
result = expensive_function(5)
```

```
print(result)    You, 6/16/23, 9:27 PM • Uncommitted c
```

```
result = expensive_function(5)
```

```
print(result)
```

Кэширование `cpu-bound functions`

`lru_cache` использует алгоритм **LRU (Least Recently Used)** для управления кэшем. Если размер кэша превышает `maxsize`, то самые редко используемые элементы будут удалены из кэша, чтобы освободить место для новых результатов.

Обратите внимание, что `lru_cache` не предоставляет возможность управления временем жизни кэша или инвалидации кэша при изменении данных. Если вам требуются такие функции, вам может потребоваться использовать сторонние библиотеки, такие как `cachetools` или `dogpile.cache`, которые предоставляют расширенные возможности для управления кэшированием функций

кастомное кэширование при помощи `redis`

Кэширование стилей в браузере

Когда клиентский браузер впервые запрашивает стили для вашего веб-приложения, сервер Django отправляет стили со специальными заголовками, указывающими на необходимость кэширования.

Когда браузер получает стили, он сохраняет их в своем кэше и использует их для последующих запросов, вместо отправки запроса на сервер. Это позволяет ускорить загрузку страницы, поскольку стили могут быть загружены из кэша браузера, а не с сервера

- Заголовок **Cache-Control** указывает браузеру, как долго стили могут быть сохранены в кэше. Например, **Cache-Control: max-age=3600**
- Заголовок **Last-Modified** содержит дату и время последнего изменения стилей на сервере. При следующем запросе браузер может отправить заголовок If-Modified-Since, содержащий значение Last-Modified, чтобы проверить, изменились ли стили с момента последнего запроса. Если стили не изменились, сервер может ответить с кодом статуса 304 "Not Modified", и браузер может использовать стили из кэша

Кэширование стилей в браузере

```
from django.views.decorators.cache import cache_control  
  
@cache_control(max_age=3600)  
def styles(request):  
    content = {}  
    return HttpResponse(content, content_type='text/css')
```


Redis (Remote Dictionary Server)

Это система управления данными с открытым исходным кодом, работающая в оперативной памяти и предназначенная для хранения, кэширования и обмена сообщениями. Она часто называется "сервером структур данных", так как позволяет хранить и манипулировать различными структурами данных, такими как строки, списки, множества, хэши и другие.

Основные особенности Redis:

1. **Хранение данных в оперативной памяти:** Redis основным образом хранит данные в памяти, что обеспечивает быстрые операции чтения и записи. Он также предлагает варианты сохранения данных на диск периодически или по требованию.
2. **Структуры данных:** Redis предоставляет богатый набор структур данных, с которыми можно взаимодействовать непосредственно на сервере, обеспечивая эффективное и гибкое моделирование данных.
3. **Механизмы кэширования и обмена сообщениями:** Redis может использоваться как кэш для ускорения доступа к данным, а также как система обмена сообщениями между компонентами приложения.

Вопросы

1. Что такое кэширование и зачем его использовать в веб-приложениях?
2. Какие механизмы кэширования доступны в Django?
3. Как настроить кэширование на уровне представлений (views) в Django?
4. Какие директивы кэширования вы знаете и как они влияют на поведение браузера и сервера?
5. Каким образом можно кэшировать данные из базы данных в Django?
6. Какие параметры можно использовать для управления кэшированием в Django?
7. Какие инструменты и библиотеки вы использовали для расширенного кэширования в Django?
8. Как обрабатывать ситуации, когда данные в кэше устарели или изменились?
9. Как можно проверить эффективность кэширования в Django?
10. Какие могут быть проблемы при использовании кэширования и как их избежать?

2

Домашнее задание

Домашнее задание

- Настроить в вашем Django приложении кэширование при помощи Redis
 - `cache_page`
 - `cache_response`
- Проверить что оно работает

The background is a vibrant yellow color, covered with a dense, repeating pattern of various geometric shapes. These shapes include circles, squares, triangles, and lines, some of which are filled with a fine grid pattern. The shapes are scattered across the entire surface, creating a dynamic and modern aesthetic.

Спасибо за внимание!