

# Включаем кэширование данных

Смотреть материал на видео

Архив проекта: lesson-22-coolsite.zip

На предыдущем занятии мы с вами увидели, сколько и какие SQL-запросы выполняются для формирования HTML-страницы. Здесь важно понимать, что эта операция повторяется для каждого запроса от клиента. И представьте, что ваш сайт посещают тысячи человек в сутки, значит, каждый час таких запросов может приходить сотни. А это еще не самая большая посещаемость. Обычным делом сейчас для сайтов является нагрузка в несколько сот тысяч, а то и миллионы запросов в сутки. Поэтому, те 3-5 SQL-запросов, что мы видим в Debug Toolbar можно смело умножать на тысячи, если имеем дело с высоконагруженным сайтом.

Как еще (помимо оптимизации самих запросов) можно разгрузить сервер и еще больше уменьшить нагрузку на СУБД? Для этого был придуман такой механизм как кэширование страниц. Идея, в общем-то очевидна. Если мы строим какой-либо информационный сайт, то содержимое его страниц обновляется не часто (включая дизайн). Поэтому при многочисленных запросах к одной и той же такой странице, было бы логично первый раз сформировать ее, а при повторных

обращениях, выдавать уже сформированный HTML-документ. Это намного снизит нагрузку на СУБД и сервер. Давайте посмотрим, как это делается во фреймворке Django.

Подробную информацию можно почитать на странице русскоязычной документации:

https://djbook.ru/rel3.0/topics/cache.html

Здесь же отмечено, что кэш в Django можно реализовать либо на уровне памяти – это самый быстрый тип кэша, либо на уровне БД – наименее распространенный способ, либо на уровне файловой системы – наиболее частый вид кэша. Его мы и рассмотрим.

Вначале нужно настроить сам механизм кэширования. В документации (для файловой системы) написано, что для этого нужно в файле settings.py прописать специальный словарь CACHES, причем он отличается в зависимости от ОС сервера. Так как я работают под Windows, то мне подойдет второй вариант, где путь к корню каталога кэша прописывается с указанием имени диска:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.c
        'LOCATION': os.path.join(
    }
}
```

Здесь я воспользовался переменной BASE_DIR для формирования пути к кэшу, то есть, в рабочем каталоге нашего проекта будет использоваться папка coolsite_cache для хранения кэша. Соответственно, мы ее должны добавить в проекте.

Теперь, посмотрим по документации порядок кэширования. Здесь мы видим список различных параметров кэша, то есть, настраивать кэш и для страниц сайтов целиком и отдельно для шаблонов и устанавливать время хранения кэша и так далее. Советую со всем этим ознакомиться.

## Кэширование на уровне представлений

Далее, видим раздел «Кэширование на уровне представлений». И здесь сразу представлен пример реализации кэша через декоратор cache_page, у которого в круглых скобках указывается время хранения кэша в секундах.

Но, мы используем не функции, а классы представлений, поэтому кэширование будем делать на уровне URLconf. Опять же в документации представлен пример. Нам здесь вначале нужно импортировать декоратор cache_page, указать время хранения кэша и представление, связанное с конкретным URL. Давайте это сделаем. Откроем файл women/urls.py и вначале пропишем импорт:

```
from django.views.decorators.cach
```

И, в качестве примера, закэшируем главную страницу:

```
path('', cache_page(60)(WomenHome
```

Все, теперь в течение 60 секунд главная страница будет использовать кэш, то есть, любые изменения, которые будут вноситься в нее в течение этого времени, не будут отображаться в браузере. Это следует учитывать, используя кэширование.

Перейдем в браузер. Сейчас для формирования главной страницы используется 386 мс и четыре SQL-запроса. Обновим ее, все без изменений, так как сейчас было выполнено формирование кэша страницы. И, если посмотреть каталог coolsite_cache, то увидим в нем файлы кэша. Обновим главную страницу еще раз и видим примерно 80 мс время формирования страницы и нулевое число SQL-запросов, то есть, был использован кэш. Через минуту страница снова будет полностью строиться по алгоритму приложения, но затем, снова браться из кэша.

## Кэширование на уровне шаблонов

Вот так выполняется кэширование отдельных типов страниц на уровне представлений. Кроме того, довольно полезным является делать кэширование на уровне шаблонов. Например, мы можем выполнить кэширование отдельно сайдбара, а не всей страницы. Для этого откроем базовый шаблон base.html и вначале (или в любом удобном месте) импортируем специальный тег для реализации кэша:

```
{% load cache %}
```

И, затем, мы его используем для сайдбара:

```
{% cache 60 sidebar %}
...
{% endcache %}
```

Если теперь обновить главную страницу, то увидим всего два SQL-запроса для ее формирования. Возможно, у вас здесь возник вопрос, а почему число SQL-запросов уменьшилось, если выборка категорий прописана в классе DataMixin и команда

```
cats = Category.objects.annotate(
```

все равно выполняется? Все верно, эта строчка будет выполняться и при кешировании сайдбара. Но, всегда следует помнить, что это «ленивый» (отложенный) запрос, то есть, он фактически выполняется только в момент обращения к данным. Здесь же, при включенном кэше обращения не происходит, сайдбар берется уже сформированным из файла кэша, поэтому и SQL-запрос не выполняется.

## Функции API низкого уровня для кэширования

Для более тонкой настройки и использования механизма кэширования в Django имеются весьма полезные функции, которые составляют уровень API для кэширования. Основные из них, следующие:

- cache.set() – сохранение произвольных данных в кэш по ключу;
- cache.get() – выбор произвольных данных из кэша по ключу;
- cache.add() – заносит новое значение в кэш, если его там еще нет (иначе данная операция игнорируется);
- cache.get_or_set() – извлекает данные из кэша, если их нет, то автоматически заносится значение по умолчанию;
- cache.delete() – удаление данных из кэша по ключу;
- cache.clear() – полная очистка кэша.

Давайте воспользуемся этим функционалом и закэшируем данные рубрик. Откроем файл women/utils.py, где сначала импортируем модуль:

```
from django.core.cache import cac
```

А, затем, в классе DataMixin будем делать выборку категорий из кэша с помощью функции get():

```
        cats = cache.get('cats')
        if not cats:
            cats = Category.objec
            cache.set('cats', cat
```

Далее, мы проверяем, если данные из кэша не были получены (это значит, что они туда либо не были помещены, либо истекло время кэша), то выполняем чтение из БД и заносим данные в кэш с помощью функции set().

Вот так, достаточно просто можно использовать кэширование страниц, их фрагментов и отдельных данных во фреймворке Django. Но, имейте в виду, этот инструмент следует включать на конечном этапе разработки сайта, чтобы в процессе программирования мы могли отслеживать все нагрузки, которые происходят при формировании ответов к серверу. Кэширование может скрывать от разработчика какие-либо SQL-запросы и вводить в заблуждение. Чтобы этого не было, кэш подключается в самую последнюю очередь.

## Видео по теме


#1. Django - что это такое, порядок установки


#2. Модель MTV. Маршрутизация. Функц представления

← Предыдущая Следующая →