Обробка запиту в Django

Центральним елементом будь-якого веб-додатка є обробка запиту, який відправляє користувач. У Django за обробку запиту відповідають представлення (**views**) та маршрутизація. Представлення — це функції або класи, які отримують об'єкт запиту і повертають відповідь. Маршрутизація визначає, які представлення будуть викликані для конкретних URL-адрес.

Давайте розглянемо основні поняття.

Представлення (Views)

Представлення зазвичай розташовуються в додатку у файлі views.py.

1. Функціональні представлення:
   Функціональні представлення є простими функціями, які приймають запит і повертають відповідь. Наприклад:

```
from django.http import HttpResponse

def my_view(request):
    return HttpResponse("Hello, World!")
```

Детально:

```
# файл: views.py
from django.http import HttpResponse

def index(request):
    return HttpResponse("Головна")

def about(request):
    return HttpResponse("Про сайт")

def contact(request):
    return HttpResponse("Контакти")
```

Цей код наразі не обробляє запити, він лише імпортує функцію render(), яка може використовуватися для обробки.

2. Клас-представлення:
   Клас-представлення — це клас, який має методи для різних HTTP-запитів (GET, POST тощо). Наприклад:

```python
from django.http import HttpResponse
from django.views import View

class MyView(View):
    def get(self, request):
        return HttpResponse("Hello, World!")
```

Маршрутизація

Маршрутизація в Django визначається у файлі urls.py для кожного додатка.

```python
from django.urls import path
from .views import my_view, MyView

urlpatterns = [
    path('function-view/', my_view, name='function-view'),
    path('class-view/', MyView.as_view(), name='class-view'),
]
```

У цьому прикладі:

- Для функціонального представлення my_view, ми вказуємо URL-адресу function-view/.
- Для клас-представлення MyView, ми вказуємо URL-адресу class-view/.
- Зауважте, що ми також вказуємо ім'я (name) для кожного маршруту, що дозволяє звертатися до них в шаблонах або в іншому коді Django.

Реєстрація маршрутів

Для того, щоб Django знала, які маршрути використовувати, їх потрібно зареєструвати. Це можна зробити в головному urls.py проєкту:

```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
]
```

У цьому прикладі, ми вказуємо, що всі URL-адреси, що починаються з myapp/, повинні оброблятися маршрутами, визначеними у файлі myapp.urls.

Це базовий огляд того, як в Django відбувається обробка запитів через представлення та маршрутизацію.

Детально про обробку запиту

Центральним елементом будь-якого веб-додатка є обробка запиту, який відправляє користувач. У Django за обробку запиту відповідають представлення або views. По суті, views представляють функції обробки, які отримують дані запиту у вигляді об'єкта HttpRequest з пакету django.http і генерують якийсь результат, який потім відправляється користувачеві.

Зазвичай views розташовуються в додатку у файлі views.py.
При створенні нового проекту файл views.py має наступний вміст:

```
from django.shortcuts import render
# Create your views here.
```

Генерувати результат можна різними способами. Один із них - використання класу HttpResponse з пакету django.http, який дозволяє надсилати текстовий вміст.
Отже, змінимо файл views.py наступним чином:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Головна")

def about(request):
    return HttpResponse("Про сайт")

def contact(request):
    return HttpResponse("Контакти")
```

Щоб ці функції відповідали за запити, потрібно визначити для них маршрути в проекті у файлі urls.py. Змінимо цей файл наступним чином:

```
from django.urls import path
from hello import views

urlpatterns = [
    path('', views.index),
    path('about', views.about),
    path('contact', views.contact),
]
```

Обробка запиту в Django і Python. Змінна urlpatterns визначає набір відповідностей функцій обробки конкретним рядкам запиту.
Відповідно до цього ми можемо відправляти не простий текст, а, наприклад, HTML-код, який потім браузер інтерпретує. Отже, змінимо файл views.py наступним чином:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Головна</h2>")

def about(request):
    return HttpResponse("<h2>Про сайт</h2>")

def contact(request):
    return HttpResponse("<h2>Контакти</h2>")
```

Path і re_path

В Django, для визначення маршрутів, використовуються функції path та re_path у модулі django.urls.

1. path: Ця функція використовується для визначення статичних маршрутів.
   Наприклад:
   from django.urls import path
   from . import views

```
urlpatterns = [
    path('home/', views.home),
    path('about/', views.about),
    path('contact/', views.contact),
]
```

2. re_path: Ця функція використовується для визначення динамічних маршрутів за допомогою регулярних виразів.
   Наприклад:

```
from django.urls import re_path
from . import views

urlpatterns = [
    re_path(r'^articles/(\\\d{4})/$', views.article_year),
    re_path(r'^articles/(\\\d{4})/(\\\d{2})/$', views.article_month),
]
```

Таким чином, path використовується для статичних маршрутів, тоді як re_path дозволяє вам визначати більш складні, динамічні маршрути за допомогою регулярних виразів.

Детально про маршрути та параметри

Центральним моментом будь-якого веб-додатка є обробка запиту, який відправляє користувач. У Django за обробку запиту відповідають представлення або views. Зазвичай views представляють собою функції обробки, які приймають дані запиту у вигляді об'єкта HttpRequest з пакету django.http і генерують який-небудь результат, який потім відсилається користувачеві.

В файлі urls.py проекту вони асоціюються з URL-адресами за допомогою функції path():

```
from django.urls import path
from hello import views

urlpatterns = [
    path('', views.index),
    path('about', views.about),
    path('contact', views.contact),
]
```

Функція path() приймає чотири параметри: route, view, kwargs, name.

Функція re_path() використовується для визначення динамічних маршрутів за допомогою регулярних виразів.
Наприклад:

```
from django.urls import path, re_path
from hello import views
```

```
urlpatterns = [
    path('', views.index),
    re_path(r'^about', views.about),
    re_path(r'^contact', views.contact),
]
```

Тут ^about вказує, що адреса повинна починатися з "about".
Очередність визначення маршрутів важлива: більш конкретні маршрути повинні бути визначені раніше, ніж більш загальні. Наприклад:

```
from django.urls import path, re_path
from hello import views

urlpatterns = [
    re_path(r'^about/contact/', views.contact),
    re_path(r'^about', views.about),
    path('', views.index),
]
```

Також можна використовувати функцію re_path() для передачі значень у функцію. Наприклад, у файлі views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Головна</h2>")

def about(request, name, age):
    return HttpResponse(f"""
            <h2>Про користувача</h2>
            <p>Ім'я: {name}</p>
            <p>Вік: {age}</p>
    """)
```

У файлі urls.py:

```
from django.urls import path
from hello import views

urlpatterns = [
    path('', views.index),
    path('about', views.about, kwargs={"name": "Tom", "age": 38}),
]
```

Тут через параметр kwargs у функцію about передається словник зі значеннями.

Отримання даних запиту: HttpRequest

Обробка запиту в Django розпочинається з об'єкта HttpRequest. HttpRequest зберігає усю інформацію, пов'язану із поточним HTTP-запитом. Він передається в функції-представлення як параметр і містить важливі дані про запит.
Основні атрибути об'єкта HttpRequest включають:

- request.GET: словник, який містить параметри GET-запиту.
- request.POST: словник, який містить параметри POST-запиту.
- request.method: рядок, який містить метод HTTP-запиту.
- request.path: рядок, який містить шлях у URL без параметрів.
- request.headers: словник, який містить заголовки запиту.
  Наприклад, розглянемо функцію-представлення, яка виводить інформацію про отримані параметри GET-запиту:

```
from django.http import HttpResponse

def show_params(request):
    get_params = request.GET
    response = f"Параметри GET-запиту: {get_params}"
    return HttpResponse(response)
```

Цю функцію слід додати до списку маршрутів у файлі urls.py. Наприклад:

```
from django.urls import path
from hello import views

urlpatterns = [
    path('show_params', views.show_params),
```

```
]
```

Тепер, якщо ви відправите GET-запит за адресою http://127.0.0.1:8000/show_params?name=John&age=25, ви побачите відповідь: Параметри GET-запиту: {'name': 'John', 'age': '25'}

Детально про HttpRequest

Функції-представлення, як обов'язковий параметр, отримують об'єкт HttpRequest, який містить інформацію про запит. HttpRequest визначає ряд атрибутів, які зберігають інформацію про запит. Виділимо наступні з них:

- scheme: схема запиту (http або https)
- body: представляє тіло запиту у вигляді рядка байтів
- path: представляє шлях запиту
- method: метод запиту (GET, POST, PUT і т.д.)
- encoding: кодування
- content_type: тип вмісту запиту (значення заголовка CONTENT_TYPE)
- GET: об'єкт у вигляді словника, який містить параметри запиту GET
- POST: об'єкт у вигляді словника, який містить параметри запиту POST
- COOKIES: відправлені клієнтом куки
- FILES: відправлені клієнтом файли
- META: містить усі доступні заголовки HTTP у вигляді словника.
- headers: заголовки запиту у вигляді словника
  Також HttpRequest визначає ряд методів. Зазначимо кілька з них:
- get_full_path(): повертає повний шлях запиту, включаючи рядок запиту
- get_host(): повертає хост клієнта
- get_port(): повертає номер порта
  Наприклад, отримаємо деяку інформацію про запит. Для цього у файлі views.py:

```
from django.http import HttpResponse

def index(request):
    host = request.META["HTTP_HOST"] # отримуємо адресу сервера
    user_agent = request.META["HTTP_USER_AGENT"]    # отримуємо дані
браузера
    path = request.path     # отримуємо запитаний шлях

    return HttpResponse(f"""
        <p>Host: {host}</p>
        <p>Path: {path}</p>
        <p>User-agent: {user_agent}</p>
    """)
```

У файлі urls.py зареєструємо дану функцію:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("index", views.index),
]
```

HttpResponse та відправлення відповіді

Коли функція-представлення обробляє запит, вона повинна повернути об'єкт відповіді, який вказує Django, що відправити назад клієнту. Одним з таких об'єктів є HttpResponse. Об'єкт HttpResponse створюється з вмістом, який потрібно відправити клієнту. Це може бути текст, HTML, JSON або будь-який інший тип вмісту.
В основному, об'єкт HttpResponse має такий синтаксис:

```
from django.http import HttpResponse

def my_view(request):
    content = "Hello, this is the response content!"
    response = HttpResponse(content, content_type="text/plain")
    return response
```

Також, якщо вам потрібно вказати код статусу HTTP, ви можете зробити це так:

```
response = HttpResponse(content, content_type="text/plain", status=200)
```

Крім того, ви можете використовувати різні методи об'єкта HttpResponse для додавання заголовків, налаштування кешування тощо. Наприклад:

```
response = HttpResponse()
response['Custom-Header'] = 'Some value'
response.set_cookie('my_cookie', 'cookie_value')
```

Детально про HttpResponse

Для відправки відповіді клієнту в Django використовується клас HttpResponse з пакету

django.http. Узагальнено для відправки деяких даних достатньо передати ці дані в конструктор HttpResponse. Наприклад, припустимо, у файлі views.py є проста функція-представлення, яка відправляє відповідь клієнту:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Привіт")
```

І в файлі urls.py цю функцію асоціюють із певним маршрутом:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("", views.index),
]
```

Функціональність HttpResponse не обмежується цим. Функція ініціалізації класу визначає кілька параметрів:

```
HttpResponse.__init__(content=b'', content_type=None, status=200,
reason=None, charset=None, headers=None)
```

Також визначає кілька атрибутів для зберігання відправлених даних. Деякі з них:

- content: вміст відповіді у вигляді байтового рядка.
- headers: відправлені заголовки у вигляді словника.
- charset: кодування відповіді у вигляді рядка.
- status_code: HTTP-код стану відповіді.
- reason_phrase: повідомлення, яке відправляється разом із кодом стану.
  Давайте розглянемо деякі можливості. Наприклад, змінимо визначення функції в файлі views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Привіт", headers={"SecretCode": "21234567"})
```

У цьому випадку також встановлюється заголовок "SecretCode". Хоча в реальності в HTTP немає такого заголовка, ми можемо визначати власні заголовки, щоб передавати через них клієнту яку-небудь інформацію.

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Сталася помилка", status=400, reason="Невірні
дані")
```

Встановлення вмісту та кодування:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Привіт</h1>", content_type="text/plain",
charset="utf-8")
```

Хоча у вмісті відповіді використовуються теги HTML (<h1>), браузер тепер буде розглядати цей вміст як звичайний текст, оскільки встановлений заголовок "text/plain".

Параметри представлень

В Django параметри представлень визначаються в функції-представленні, яка обробляє певний запит.

1. request: Це обов'язковий параметр, який представляє об'єкт HttpRequest.
   Приклад:
   from django.http import HttpResponse

```
def example_view(request):
    method = request.method
    path = request.path
    return HttpResponse(f"Метод: {method}, Шлях: {path}")
```

2. args та kwargs: Ці параметри дозволяють отримати доступ до позиційних і ключових аргументів.
   Приклад:

```python
from django.http import HttpResponse

def example_view(request, *args, **kwargs):
    arg1 = args[0] if args else None
    kwarg1 = kwargs.get('param1', None)
    return HttpResponse(f"Позиційний аргумент: {arg1}, Ключовий аргумент: {kwarg1}")
```

3. pk або інші названі параметри:
   В urls.py:

```python
from django.urls import path
from .views import example_view

urlpatterns = [
    path('example/<int:pk>/', example_view, name='example-detail'),
]
```

В views.py:

```python
from django.http import HttpResponse

def example_view(request, pk):
    return HttpResponse(f"Отримано значення pk: {pk}")
```

Детально про параметри представлень

В Django параметри представлень передаються через адресу URL. Наприклад, у запиті:
http://127.0.0.1:8000/index/Tom/38/
останні два сегменти Tom/38/ можуть представляти параметри URL.
Визначення параметрів через функцію path:
Давайте визначимо наступні функції в файлі views.py:

```python
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Головна</h2>")
```

```
def user(request, name):
    return HttpResponse(f"<h2>Ім'я: {name}</h2>")
```

Далі в файлі urls.py ми визначимо такий код:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("", views.index),
    path("user/<str:name>", views.user),
]
```

За замовчуванням Django надає наступні специфікатори: str, int, slug, uuid, path. Аналогічно можна визначити і більше параметрів. Наприклад, додамо другий параметр у функцію user в views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Головна</h2>")

def user(request, name, age):
    return HttpResponse(f"<h2>Ім'я: {name}  Вік:{age}</h2>")
```

А в файлі urls.py додамо до маршруту параметр age:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("", views.index),
    path("user/<name>/<int:age>", views.user),
]
```

У цьому випадку ми можемо звертатися до функції user, наприклад, за допомогою запиту:

```
http://127.0.0.1:8000/user/Tom/38
```

У цьому випадку сегмент /Tom/ буде відповідати параметру name, а /38 - параметру age.
Значення для параметрів за замовчуванням:
У вищеперечисленому прикладі використовувалися два параметри, але що, якщо ми не передамо значення для одного або обох параметрів?
У цьому випадку ми отримаємо помилку. Однак ми можемо задати значення за замовчуванням для параметрів маршруту на випадок, якщо значення не передаються через рядок запиту. Так, для функції user в views.py ми визначимо значення за замовчуванням для параметрів:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Головна</h2>")

def user(request, name="Undefined", age=0):
    return HttpResponse(f"<h2>Ім'я: {name}  Вік: {age}</h2>")
```

У цьому випадку для функції user в файлі urls.py треба визначити додаткові маршрути, які не враховують необов'язкові параметри:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("", views.index),
    path("user", views.user),
    path("user/<name>", views.user),
    path("user/<name>/<int:age>", views.user),
]
```

Визначення параметрів через функцію re_path:
Аналогічно можна використовувати функцію re_path для визначення параметрів.
Визначимо в файлі views.py такі функції:

```
from django.http import HttpResponse
```

```
def index(request):
    return HttpResponse("<h2>Головна</h2>")

def user(request, name, age):
    return HttpResponse(f"<h2>Ім'я: {name}  Вік: {age}</h2>")
```

Тепер змінимо файл urls.py, щоб він міг відповідати цим функціям запитів:

```
from django.urls import path, re_path
from hello import views

urlpatterns = [
    path("", views.index),
    re_path(r"^user/(?P<name>\\\D+)/(?P<age>\\\d+)", views.user),
]
```

Для представлення параметра в шаблоні адреси використовується вираз ?P<>. Загальне визначення параметру відповідає формату (?P<ім'я_параметра>регулярний_вираз). Між вугловими дужками вказується назва параметра.
Також ми можемо вказати значення за замовчуванням для певних параметрів:

```
def user(request, name="Undefined", age=0):
    return HttpResponse(f"<h2>Ім'я: {name}  Вік: {age}</h2>")
```

У цьому випадку потрібно додатково визначити ще маршрути в файлі urls.py для тих запитів, в яких значення для маршрутів не передаються:

```
from django.urls import path, re_path
from hello import views

urlpatterns = [
    path("", views.index),
    re_path(r"^user/(?P<name>\\\D+)/(?P<age>\\\d+)", views.user),
    re_path(r"^user/(?P<name>\\\D+)", views.user),
    re_path(r"^user", views.user),
```

```
]
```

Зверніть увагу на порядок розташування маршрутів: на відміну від випадку з функцією path, тут спочатку розміщуються більш конкретні маршрути з більшою кількістю параметрів.

Приклади для закріплення

Завдання 1: Простий калькулятор

- **views.py:** Створіть функцію calculator(request, operation, num1, num2).
- **urls.py:** Створіть маршрут path('calc/<str:operation>/<int:num1>/<int:num2>/', views.calculator).
- operation може бути add, sub, mul або div.
- На основі operation, функція має повернути результат відповідної математичної операції.

Завдання 2: Динамічна сторінка новин

- **views.py:** Створіть функцію news_article(request, year, month, day, slug).
- **urls.py:** Створіть маршрут re_path(r'^news/(?P<year>\\\\d{4})/(?P<month>\\\\d{2})/(?P<day>\\\\d{2})/(?P<slug>[a-zA-Z0-9-]+)/$', views.news_article).
- slug – це унікальна частина URL, що складається з букв, цифр і дефісів.
- Функція має повернути HTML-сторінку з інформацією про статтю, включаючи дату та заголовок з slug.

Відповіді до завдань

Завдання 1: Простий калькулятор

- **Файл** `views.py`

```
# myapp/views.py
from django.http import HttpResponse

def calculator(request, operation, num1, num2):
    result = None
    error_message = ""

    if operation == 'add':
        result = num1 + num2
    elif operation == 'sub':
        result = num1 - num2
    elif operation == 'mul':
        result = num1 * num2
    elif operation == 'div':
        if num2 != 0:
            result = num1 / num2
        else:
            error_message = "Ділення на нуль неможливе!"
    else:
        error_message = "Невідома операція. Використовуйте 'add', 'sub',
'mul' або 'div'."

    if error_message:
        return HttpResponse(f"<h1>Помилка: {error_message}</h1>")
    else:
        return HttpResponse(f"<h1>Результат: {result}</h1>")
```

- **Файл** `urls.py`

```
# myapp/urls.py
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('calc/<str:operation>/<int:num1>/<int:num2>/', views.calculator),
]
```

Завдання 2: Динамічна сторінка новин

- **Файл** views.py

```
# myapp/views.py
from django.http import HttpResponse

def news_article(request, year, month, day, slug):
    return HttpResponse(f"""
        <h1>{slug.replace('-', ' ').capitalize()}</h1>
        <p>Дата публікації: {day}.{month}.{year}</p>
        <p>Це динамічна сторінка для новини з URL-адреси.</p>
    """)
```

- **Файл** urls.py

```
# myapp/urls.py
from django.urls import re_path
from . import views

urlpatterns = [

re_path(r'^news/(?P<year>\\\\d{4})/(?P<month>\\\\d{2})/(?P<day>\\\\d{2})/(?
P<slug>[a-zA-Z0-9-]+)/$', views.news_article),
]
```