MICROSERVICIOS ALTAMENTE PARALELOS PARA TODOS

CÓMO MONTAR UN SERVIDOR WEB ASÍNCRONO CON STARLETTE

Por Alberto Fernández Valiente

ayesa

MICROSERVICIOS ALTAMENTE PARALELOS PARA TODOS

- Django vs Flask
- API con una única responsabilidad
- Solo dependencias indispensables
- Desplegables independientemente

MICROSERVICIOS ALTAMENTE PARALELOS PARA TODOS

- Muchas peticiones a la vez
- Latencia de las respuestas
- Escalabilidad y ráfagas de peticiones

MICROSERVICIOS ALTAMENTE PARALELOS PARA TODOS

- Costes controlados
- Infraestructura sencilla

ASYNCIO

ASYNCIO VS THREADING

- Corrutinas vs hilos
- Gestión de la ejecución
- Cuellos de botella
- Primitivas de sincronización

STARLETTE

- Framework ligero para crear servicios web asíncronos
- Alto rendimiento en producción
- Diseño modular con componentes reusables
- Soporta WebSocket
- Código 100% con anotaciones de tipado
- Usable sin dependencias externas

EJEMPLO BÁSICO

```
import uvicorn
from starlette.applications import Starlette
from starlette.responses import JSONResponse

app = Starlette(debug=True)

@app.route('/')
async def homepage(request):
    return JSONResponse({'hello': 'world'})

if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

UVICORN

```
if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

- Servidor ASGI(Asynchronous Server Gateway Interface)
- Compatible con WSGI
- Basado en uvloop y httptools
- Soporta HTTP/1 y WebSockets

APPLICATIONS

```
from starlette.applications import Starlette

app = Starlette(debug=True)
```

- Enrutamiento
- Eventos
- Montar aplicaciones ASGI
- Manejo de excepciones
- Gestión del estado común

ENRUTAMIENTO SIMPLE

```
@app.route('/')
async def homepage(request):
    return JSONResponse({'hello': 'world'})
```

Se usa un decorador sobre una función asíncrona que solo tiene un único parámetro de entrada, la petición del cliente.

REQUEST

```
@app.route('/')
async def homepage(request):
    return JSONResponse({'hello': 'world'})
```

- Método, parametros y URL de la petición
- Cabeceras y cookies
- Cuerpo de la petición: formulario, JSON o datos binarios
- Ficheros enviados en la petición
- Estado asociado

RESPONSE

```
Response(content, status_code, headers, media_type)
```

- Permite asignar y borrar Cookies
- PlainTextResponse, HTMLResponse, JSONResponse
- RedirectResponse
- StreamingResponse
- FileResponse

ENRUTAMIENTO AVANZADO

```
from starlette.routing import Router, Route, Mount

app.mount('', Router([
    Route('/', endpoint=Homepage, methods=['GET']),
    Mount('/users', app=Router([
        Route('/', endpoint=Users, methods=['GET', 'POST']),
        Route('/{username}', endpoint=User, methods=['GET']),
    ]))
]))
```

Router es una aplicación ASGI que se encarga de enviar las peticiones entrantes a los endpoints o aplicaciones ASGI anidadas correspondientes.

ENDPOINTS

```
from starlette.endpoints import HTTPEndpoint

class CustomEndpoint(HTTPEndpoint):
    async def dispatch(self):
        request = Request(self.scope, receive=self.receive)
        handler = getattr(self, request.method.lower())
        response = await handler(request)
        await response(self.scope, self.receive, self.send)
```

Permiten crear vistas basadas en clases.

HTTPEndpoint se utiliza para para peticiones HTTP y WebSocketEndpoint para sesiones WebSocket.

MIDDLEWARE

```
from starlette.middleware.base import BaseHTTPMiddleware

class SessionMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        response = await call_next(request)
        return response

app.add_middleware(SessionMiddleware)
```

Clases usadas para añadir un comportamiento que será aplicado en toda la aplicación.

Se incluyen: CORSMiddleware, SessionMiddleware, HTTPSRedirectMiddleware, TrustedHostMiddleware, GZipMiddleware.

PLANTILLAS

```
from starlette.templating import Jinja2Templates

templates = Jinja2Templates(directory='templates')

@app.route('/')
async def homepage(request):
    context = {'request': request}
    return templates.TemplateResponse('index.html', context)
```

No hay ningún motor de plantillas predeterminado, aunque se proporciona un mecanismo sencillo para usar Jinja2.

EVENTOS

```
@app.on_event('startup')
async def setup_pool():
    await RedisPool.initialize()

async def teardown_pool():
    if RedisPool.pool:
        RedisPool.pool.close()

app.add_event_handler('shutdown', teardown_pool)
```

Permite registrar código que se ejecutará antes de que la aplicación arranque o cuando se cierre.

TAREAS EN SEGUNDO PLANO

```
from starlette.background import BackgroundTask

async def send_welcome_email(to_address):
    pass

@app.route('/user/signup', methods=['POST'])
async def signup(request):
    task = BackgroundTask(send_welcome_email, to_address=email)
    message = {'status': 'Signup successful'}
    return JSONResponse(message, background=task)
```

Permite crear tareas que se ejecutarán una vez se haya mandado la respuesta al cliente.

OTROS DETALLES

- Configuración por variables de entorno
- Integración con bases de datos
- GraphQL
- Autenticación
- Generación de esquemas OpenAPI
- Soporte de server push en HTTP/2 y HTTP/3

PUESTA EN PRODUCCIÓN

- Circus + Uvicorn
- Nginx + Supervisor + Gunicorn con UvicornWorker

FRAMEWORKS BASADOS EN STARLETTE

- FastAPI
- Responder
- Bocadillo

REFERENCIAS

- Documentación Starlette
- Librería asyncio

GRACIAS A TODOS

Código de la aplicación de ejemplo disponible en
https://github.com/voodmania/pycones2019-demo