第 5 章

ASGI

ASGI(Asynchronous Server Gateway Interface)为异步服务网关接口，秉承 WSGI 统一网关接口原则，在异步服务、框架和应用之间提供一个标准接口，同时兼容 WSGI。

学习一个新知识之前，我们首先需要明白它为什么会出现，那么为什么 ASGI 和 WSGI 这类的统一网关接口会出现呢？

Python Web 开发大致可以分为两大层，协议层(HTTP 逻辑层，后简称协议层)和应用层，协议层主要根据 HTTP 协议解码与编码数据，应用层主要实现具体的业务逻辑，分层的好处是应用层可以根据需要很方便地替换协议层，要做到这一点，只需为协议层和应用层设计一个统一的接口，这就是 WSGI 与 ASGI。

有了这个统一的接口，协议层不仅仅只有 Python 才能实现，其他的语言例如 C 语言如果按这个接口去实现协议层，那么也可以和 Python 应用层对接，所以出现的 WSGI 可让 Python 运行在 Apache 服务器上，这样可以让 Apache 强大的静态文件请求处理功能发挥作用，避免 Python 去处理静态请求，从而大大提高整个网站的运行速度。

6min

5.1 WSGI

WSGI 可以与传统阻塞型 IO 服务器配合使用。

在这里需要明确一点，传统阻塞型 IO 服务器就一定是低效的吗？当然不是，我们已知异步 IO 编程模型的目的是为了合理利用 IO 资源，而传统的阻塞型 IO 编程也可以通过复杂的方案实现合理利用 IO 资源，例如优秀的 Apache 服务器。当然最新的 Apache 服务器也采用了异步 IO 编程模型，毕竟是操作系统已经实现了的功能，直接使用能够让项目的维护成本降低。

在传统阻塞型 IO 编程模型中，通常会把 Python 应用服务与 Apache 服务器结合使用，仅把与业务逻辑处理相关的任务交给 Python 语言去完成，而其他资源的处理(例如：静态文件请求处理、代理等)任务交给强大的 Apache 服务器去完成，从而实现负载均衡，提高项目整体的运行效率。

接下来我们以搭建一个 Apache + WSGI 的 Docker 环境为例演示如何在项目中使用 WSGI。项目结构如图 5-1 所示。

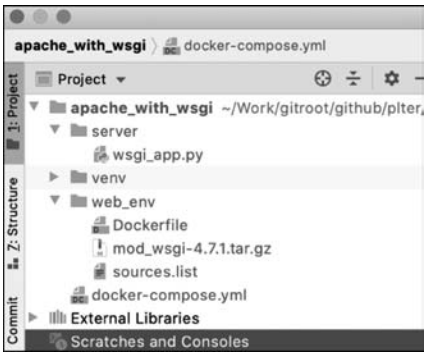

图 5-1　项目结构

先从编写 Docker 构建脚本开始介绍，我们计划使用的基础镜像是 httpd:2.4。因为需要重新编译 mod_wsgi 模块，所以需要下载庞大的 C/C++编译环境，而官方镜像服务器在国外，下载速度较慢，所以这里使用清华的 Debian 镜像。

首先需要准备一个 sources.list 文件，其内容如下：

```
#第 5 章/apache_with_wsgi/web_env/sources.list

deb http://mirrors.tuna.tsinghua.edu.cn/debian/ buster main contrib non-free
#deb-src https://mirrors.tuna.tsinghua.edu.cn/debian/ buster main contrib
#non-free
deb http://mirrors.tuna.tsinghua.edu.cn/debian/ buster-updates main contrib non-free
#deb-src https://mirrors.tuna.tsinghua.edu.cn/debian/ buster-updates main
#contrib non-free
deb http://mirrors.tuna.tsinghua.edu.cn/debian/ buster-backports main
contrib non-free
#deb-src https://mirrors.tuna.tsinghua.edu.cn/debian/ buster-backports
#main contrib non-free
deb http://mirrors.tuna.tsinghua.edu.cn/debian-security buster/updates
main contrib non-free
#deb-src https://mirrors.tuna.tsinghua.edu.cn/debian-security
#buster/updates main contrib non-free
```

mod_wsgi 的源码可从 https://GitHub.com/GrahamDumpleton/mod_wsgi 下载，在这里使用的 mod_wsgi-4.7.1，如果使用其他版本，需要注意修改相应的目录。Dockerfile 文件的内容如下：

```
#第 5 章/apache_with_wsgi/web_env/Dockerfile
FROM httpd:2.4.41

RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak
#替换软件源为清华镜像站
COPY sources.list /etc/apt/
#更新软件源
RUN apt-get update
#安装 Python 运行环境及 C 语言编译器用以编译 WSGI
RUN apt-get install -y python3 python3-dev gcc g++ make
#将 WSGI 源码复制到镜像中
COPY mod_wsgi-4.7.1.tar.gz /opt
#切换镜像中的工作目录
WORKDIR /opt
#解压 wsgi 源码包
RUN tar -xzvf ./mod_wsgi-4.7.1.tar.gz
#切换工作目录到 WSGI 源码目录
WORKDIR /opt/mod_wsgi-4.7.1
#配置编译环境
RUN ./configure --with-apxs=/usr/local/apache2/bin/apxs \
    --with-python=/usr/bin/python3
#编译并安装
```

```
RUN make&&make install

#这里对 Apache 服务器的配置文件进行修改,添加 WSGI 相关的配置项
#包括加载 WSGI 动态库、配置脚本文件、配置脚本文件所在目录的权限
RUN echo 'LoadModule wsgi_module modules/mod_wsgi.so' >>\
    /usr/local/apache2/conf/httpd.conf
RUN echo 'WSGIScriptAlias /app /opt/server/wsgi_app.py' >>\
    /usr/local/apache2/conf/httpd.conf
RUN echo '<Directory /opt/server/>' >>\
    /usr/local/apache2/conf/httpd.conf
RUN echo ' Require all granted' >> \
    /usr/local/apache2/conf/httpd.conf
RUN echo '</Directory>' >> /usr/local/apache2/conf/httpd.conf
```

脚本文件 wsgi_app.py 的内容如下：

```
"""
第 5 章/apache_with_wsgi/server/wsgi_app.py
"""


def application(environ, start_response):
    status = '200 OK'
    output = b'Hello World!'

    response_headers = [
        ('Content-type', 'text/plain'),
        ('Content-Length', str(len(output)))
    ]
    start_response(status, response_headers)
    return [output]
```

这是一个最简单的 WSGI 脚本，其中声明了一个 application 函数，当有前端请求时，该函数会被调用以处理请求，这个示例中向前端返回'Hello World!'。

对应的 docker-compose.yml 文件内容如下：

```
#第 5 章/apache_with_wsgi/docker-compose.yml
version: "3"

services:
  web:
    build: ./web_env
    tty: true
    ports:
      - "80:80"
    volumes:
      - "./server:/opt/server"
```

在终端里执行命令 docker-compose up-d 以构建并启动该服务器，待服务器启动完成后，通过浏览器访问 http://127.0.0.1/app，显示效果如图 5-2 所示。

图 5-2 Apache 集成 WSGI 示例页面

用同样的方式，还可以集成第三方框架如 web.py、web2py、Django 等以提高开发效率。但这些都已经是过时的技术了，讲解 WSGI 是为了让读者拥有维护旧项目的能力，并对 ASGI 有一个初步的认知，也因为目前 Apache 和 Nginx 平台均未出现 ASGI 模块，无法演示 ASGI 模块的编译与配置，但构建 ASGI 与构建 WSGI 类似，所以只能以构建 WSGI 环境来演示构建过程。

5.2 ASGI

8min

ASGI 是根据统一接口的思想重新设计的新标准，你可能会有疑问，为什么不直接升级 WSGI 而去创造新的标准呢？

WSGI 是基于 HTTP 短连接的网关接口，一次调用请求必须尽快处理完毕并返回结果，这种模式并不适用于长连接，例如 HTML 5 新标准中的技术 SSE(Server-Sent Events)和 WebSocket，WSGI 及传统阻塞型 IO 编程模型并不擅长处理这类请求。就算强行升级 WSGI 以支持异步 IO，可是如果配套的技术(如 Apache 服务器)没有提供相应的支持也是没有意义的。既然 Python 异步 IO 编程模型已经走在了前面，那就制定一个全新的标准 ASGI 以最优雅的方式支持并使用最新的技术。

ASGI 接口是一个异步函数，它要求传入 3 个参数，分别为 scope、receive 和 send，示例代码如下：

```
async def app(scope, receive, send):
    pass
```

其中 scope 是一个字典(dict)，包括连接相关的信息，图 5-3 所示是一个请求中的 scope 所包括信息的断点调试截图。

receive 是一个异步函数，用于读取前端发来的信息，一条读取到的信息结构如下：

```
{
    'type': 'http.request',
    'body': b"",
    'more_body': False
}
```

该信息中包括 3 个字段，分别为类型(type)、内容(body)和是否还有更多内容(more_body)，其中通过 type 可以用来判断该信息是什么类型，如 HTTP 请求、生命周期、WebSocket 请求等，body 是该信息中包括的数据，此数据采用二进制格式，more_body 指明当前数据是否已经发送完毕，如果发送完毕，则 more_body 的值为 False，这样便可以用来

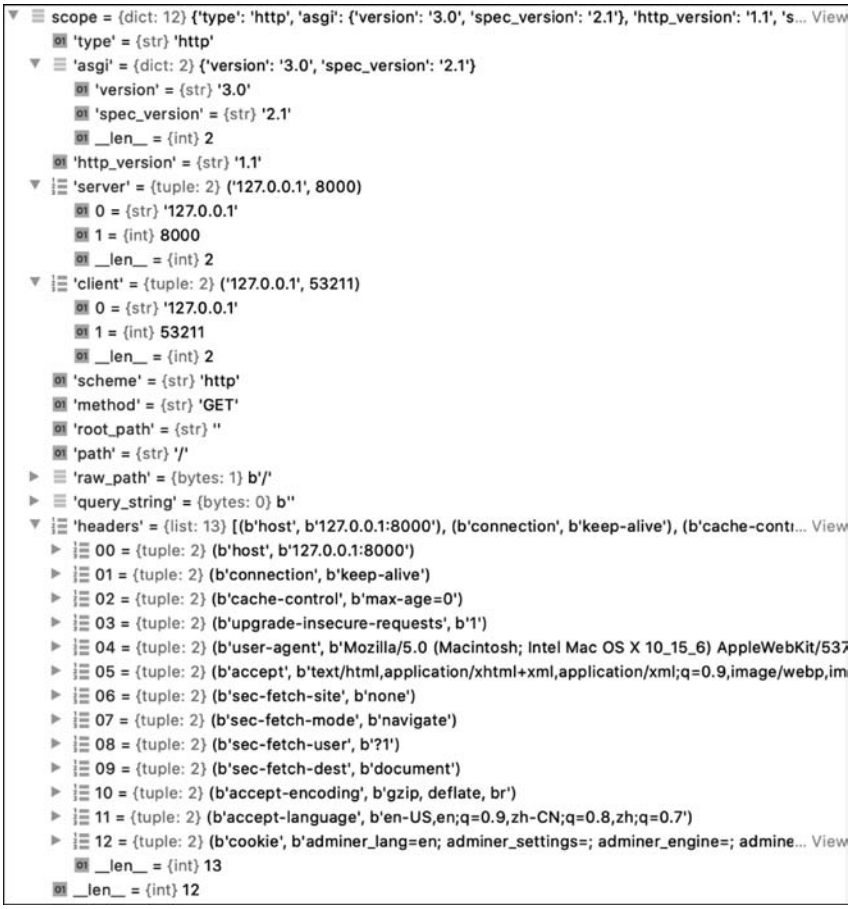

图 5-3　断点截图

分段传输大文件。

send 也是一个异步函数，用于向前端发送信息，所发送的信息结构与从前端接收的信息结构类似。一个向前端发送简单信息的示例代码如下：

```
"""
第 5 章/asgi_app/simple_asgi.py
"""



async def app(scope, receive, send):
    # 向前端发送 HTTP 协议头，包括了 HTTP 状态与协议头
    await send({
        'type': 'http.response.start',
        'status': 200,
        'headers': [
            [b'content-type', b'text/html'],
        ]
    })
```

```
    #向前端发送数据,如果数据庞大,则可以分段发送
    await send({
        'type': 'http.response.body',
        'body': b"Hello World",
        'more_body': False
    })
```

除了常规数据通信外,ASGI 还规定了生命周期管理接口,可以用于侦听服务器的启动与关闭。在实际开发工作中,这非常有用,可以用来执行初始化工作与收尾工作,生命周期管理的运用代码如下:

```
"""
第 5 章/asgi_app/asgi_lc.py
"""


async def app(scope, receive, send):
    request_type = scope['type']
    if request_type == 'lifespan':
        while True:
            message = await receive()
            if message['type'] == 'lifespan.startup':
                await send({'type': 'lifespan.startup.complete'})
            elif message['type'] == 'lifespan.shutdown':
                await send({'type': 'lifespan.shutdown.complete'})
                break
```

当 scpoe['type']的类型是 lifespan 时,意味着该请求的类型是生命周期,该请求会在服务器启动之初发生,接下来应该实现对生命周期的管理。

通过无限循环不断侦听请求状态的变化,当读到 message['type']是 lifespan.startup 时执行初始化操作,在操作完成后向前端(协议层)发送 lifespan.startup.complete 信息,协议层可理解为服务器已经启动完成,可以正常接受浏览器请求了。

当读到 message['type']是 lifespan.shutdown 时,意味服务要关闭,可能是由于服务器管理员执行了关闭指令,那么在这里就需要执行收尾工作,例如释放相应资源等。在收尾完成后向协议层发送 lifespan.shutdown.complete 信息,表明此时协议层可以放心地关闭服务器。

一个完整的基于 ASGI 的 Hello World 示例代码如下:

```
"""
第 5 章/asgi_app/asgi.py
"""


async def app(scope, receive, send):
    request_type = scope['type']
    if request_type == 'http':
        await send({
```

```
            'type': 'http.response.start',
            'status': 200,
            'headers': [
                [b'content-type', b'text/html'],
            ]
        })
        await send({
            'type': 'http.response.body',
            'body': b"Hello World",
            'more_body': False
        })
    elif request_type == 'lifespan':
        while True:
            message = await receive()
            if message['type'] == 'lifespan.startup':
                await send({'type': 'lifespan.startup.complete'})
            elif message['type'] == 'lifespan.shutdown':
                await send({'type': 'lifespan.shutdown.complete'})
                break
    else:
        raise NotImplementedError()
```

5.3 Uvicorn

Uvicorn是ASGI的一个协议层实现，一个轻量级的ASGI服务器，基于uvloop和httptools实现，运行速度极快。

uvloop是一个高效的基于异步IO的事件循环框架，底层实现由libuv承载。libuv是一个使用C语言开发的支持高并发的异步IO库，由Node.js的作者开发，作为Node.js的底层IO库实现，如今已经发展得相当成熟稳定。

要使用Uvicorn需要先通过命令pip install uvicorn安装该依赖项，项目结构如图5-4所示。

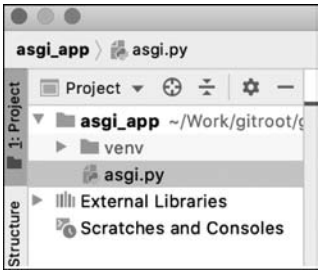

图5-4 ASGI项目文件结构

其中asgi.py文件即第5章/asgi_app/asgi.py。

接下来在终端输入uvicorn asgi:app以启动该服务器，效果如图5-5所示。

```
Terminal:  Local  +
(venv) yunp@yunps-MBP asgi_app % uvicorn asgi:app
INFO:     Started server process [8574]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

图5-5 启动ASGI服务器

服务器启动后，可使用浏览器通过 http://127.0.0.1:8000 访问该站点，结果如图 5-6 所示。

图 5-6 页面访问结果

为了向用户提供更加安全的服务，现代网站都需要支持 HTTPS，Uvicorn 也提供了对 HTTPS 的支持，使用起来也相当方便。

首先准备好 HTTPS 证书文件，如图 5-7 所示。

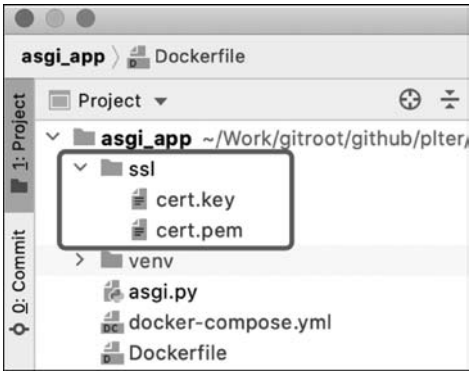

图 5-7 证书文件所在目录

接下来通过命令 uvicorn --ssl-certfile ./ssl/cert.pem --ssl-keyfile ./ssl/cert.key asgi:app 来启动该服务器，如图 5-8 所示。

图 5-8 以 HTTPS 方式启动服务器

容器化对应的 Dockerfile 文件内容如下：

```
#第 5 章/asgi_app/Dockerfile
FROM python:3 - slim

RUN pip3 install  - i https://pypi.tuna.tsinghua.edu.cn/simple uvicorn
```

容器化对应的 docker-compose.yml 内容如下：

```
#第 5 章/asgi_app/docker - compose.yml
version: "3"

services:
  web:
    build: .
    restart: always
    tty: true
```

```
    ports:
      - "8000:8000"
    volumes:
      - ".:/opt/"
    working_dir: "/opt/"
    command: uvicorn -- host 0.0.0.0 asgi:app
```

5.4 Daphne

Daphne是一个功能强大的ASGI协议层实现，支持HTTP、HTTP2和WebSocket协议，由Django团队开发，用于支持DjangoChannels(Channels是一个由Django团队开发的可让Django支持WebSocket、HTTP长连接和其他异步编程的框架)的功能。

如果要使用Daphne，需要先通过命令pip install daphne进行安装，然后通过命令daphne asgi:app启动服务器，如图5-9所示。

```
Terminal:  Local  +
(venv) yunp@yunps-MBP daphne_asgi_app % daphne asgi:app
2020-09-11 09:44:47,972 INFO     Starting server at tcp:port=8000:interface=127.0.0.1
2020-09-11 09:44:47,973 INFO     HTTP/2 support not enabled (install the http2 and tls Twisted extras)
2020-09-11 09:44:47,973 INFO     Configuring endpoint tcp:port=8000:interface=127.0.0.1
2020-09-11 09:44:47,975 INFO     Listening on TCP address 127.0.0.1:8000
```

图5-9 启动Daphne服务器

此时可以使用浏览器通过http://127.0.0.1:8000进行访问，结果如图5-10所示。

Daphne也支持HTTPS，需要先准备好HTTPS证书，如图5-11所示。

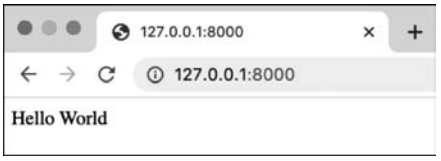

图5-10 页面访问结果

图5-11 证书文件所在目录

接下来用命令daphne-e ssl:8443:privateKey=./ssl/cert.key:certKey=./ssl/cert.pem asgi:app启动服务器，如图5-12所示。

```
daphne_asgi_app % daphne -e ssl:8443:privateKey=./ssl/cert.key:certKey=./ssl/cert.pem asgi:app
01 INFO     Starting server at ssl:8443:privateKey=./ssl/cert.key:certKey=./ssl/cert.pem
01 INFO     HTTP/2 support not enabled (install the http2 and tls Twisted extras)
01 INFO     Configuring endpoint ssl:8443:privateKey=./ssl/cert.key:certKey=./ssl/cert.pem
06 INFO     Listening on TCP address 0.0.0.0:8443
```

图5-12 启动Daphne HTTPS服务器

此时可以使用浏览器通过 https://127.0.0.1:8443 进行访问，结果如图 5-13 所示。

图 5-13 访问 HTTPS 页面

容器化对应的 Dockerfile 文件内容如下：

```
#第 5 章/daphne_asgi_app/Dockerfile
FROM python:3 - slim

#因为安装 Daphne 的过程中可能需要编译原生代码，所以需要安装 GCC 编译器
RUN apt update&&apt install - y gcc make
RUN pip3 install - i https://pypi.tuna.tsinghua.edu.cn/simple daphne
```

容器化对应的 docker-compose.yml 文件内容如下：

```
#第 5 章/daphne_asgi_app/docker - compose.yml
version: "3"

services:
  web:
    build: .
    restart: always
    tty: true
    ports:
      - "8000:8000"
    volumes:
      - ".:/opt/"
    working_dir: "/opt/"
    command: daphne - b 0.0.0.0 asgi:app
```

5.5 Django 搭配 ASGI

Django 是一个非常强大的 Python Web 开发框架，也是目前 Python 语言使用最广泛的 Web 开发框架，很多人在入门 Python Web 开发中所学习的第一个技术框架就是 Django，此框架具有易学、易用、功能强大、文档优秀、技术支持完整等优点，这些优点使 Django 在近十几年来一直都是 Python 语言中最受欢迎的 Web 开发框架。

随着 ASGI 技术的快速发展，Django 率先对 ASGI 提供了支持，目前官方已经发行了支持 ASGI 的稳定版本框架。

与 AIOHTTP 不同，Django 对于技术的分层非常明确，Django 属于应用层实现，协议层可以自由切换。AIOHTTP 同时实现了协议层与应用层，并将所有的技术混合在一起，

虽然使用起来相当方便，但是如果你期望后期随着技术的迁移和项目架构的改动而更换协议层进行实现，将会变得非常麻烦。而 Django 框架是单纯的应用层实现，无须担心协议层的技术实现，可以通过 WSGI 技术将协议层实现交给 Apache 服务器或者 Nginx 服务器实现，也可以通过 ASGI 技术将协议层实现交给 Uvicorn 或者 Daphne 等实现。

接下来我们一步一步来演示如何将 Django 部署在 ASGI 服务器上。

使用 PyCharm 创建一个新项目，名为 django_proj，并基于 Python 3.8 创建一个运行环境，如图 5-14 所示。

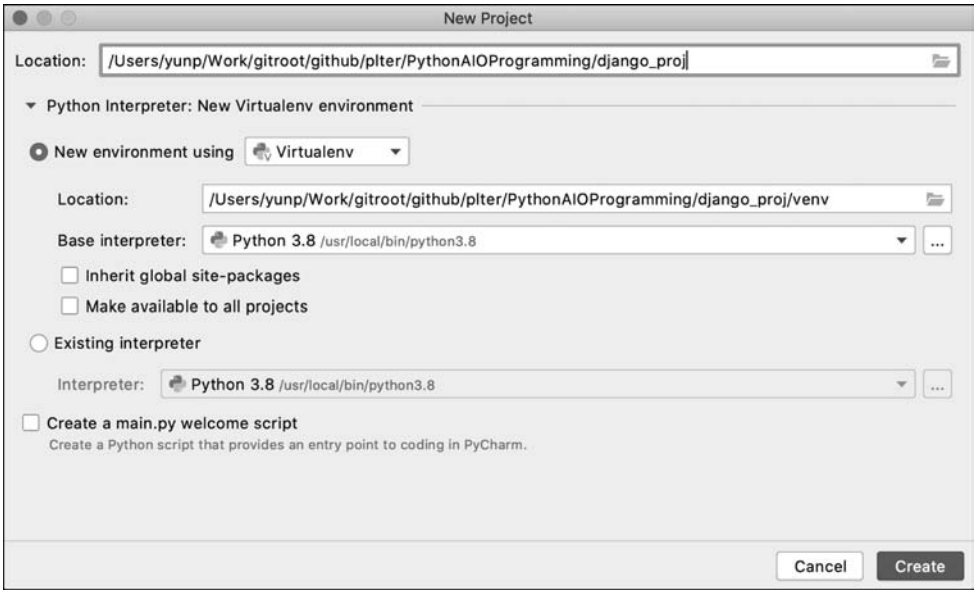

图 5-14　创建项目

项目创建完成后通过命令 pip install Django 安装 Django，如图 5-15 所示。

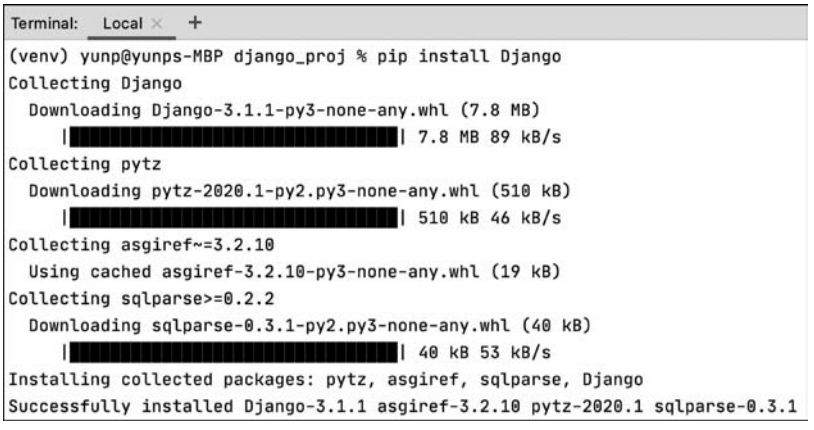

图 5-15　安装 Django

通过命令 django-admin startproject web 创建一个 Django 项目，如图 5-16 所示。

通过命令 pip install uvicorn 安装 Uvicorn 环境，如图 5-17 所示。

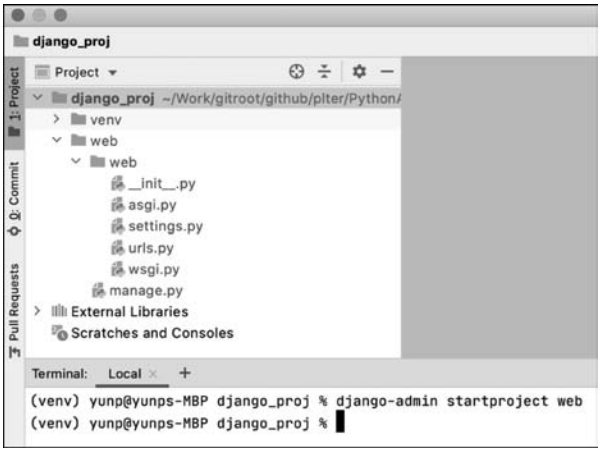

图 5-16 创建 Django 项目

```
Terminal:  Local  +
(venv) yunp@yunps-MBP django_proj % pip install uvicorn
Collecting uvicorn
  Downloading uvicorn-0.11.8-py3-none-any.whl (43 kB)
     |████████████████████████████████| 43 kB 124 kB/s
Collecting h11<0.10,>=0.8
  Downloading h11-0.9.0-py2.py3-none-any.whl (53 kB)
     |████████████████████████████████| 53 kB 72 kB/s
Collecting websockets==8.*
  Downloading websockets-8.1-cp38-cp38-macosx_10_9_x86_64.whl (64 kB)
     |████████████████████████████████| 64 kB 96 kB/s
Collecting uvloop>=0.14.0; sys_platform != "win32" and sys_platform != "cygwin" and platf
  Downloading uvloop-0.14.0-cp38-cp38-macosx_10_11_x86_64.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 59 kB/s
Collecting click==7.*
  Downloading click-7.1.2-py2.py3-none-any.whl (82 kB)
     |████████████████████████████████| 82 kB 111 kB/s
Collecting httptools==0.1.*; sys_platform != "win32" and sys_platform != "cygwin" and pla
  Downloading httptools-0.1.1-cp38-cp38-macosx_10_13_x86_64.whl (102 kB)
     |████████████████████████████████| 102 kB 68 kB/s
Installing collected packages: h11, websockets, uvloop, click, httptools, uvicorn
Successfully installed click-7.1.2 h11-0.9.0 httptools-0.1.1 uvicorn-0.11.8 uvloop-0.14.0
(venv) yunp@yunps-MBP django_proj %
```

图 5-17 安装 Uvicorn

使用命令 cd web 将工作目录切换到 web 目录，如图 5-18 所示。

```
Terminal:  Local  +
(venv) yunp@yunps-MBP django_proj % cd web
(venv) yunp@yunps-MBP web %
```

图 5-18 切换工作目录

使用命令 uvicorn web.asgi:application 启动服务器，如图 5-19 所示。

```
Terminal:  Local  +
(venv) yunp@yunps-MBP django_proj % cd web
(venv) yunp@yunps-MBP web % uvicorn web.asgi:application
INFO:     Started server process [3977]
INFO:     Waiting for application startup.
INFO:     ASGI 'lifespan' protocol appears unsupported.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

图 5-19　启动 Django 服务器

此时可以使用浏览器通过地址 http://127.0.0.1:8000 访问，效果如图 5-20 所示。

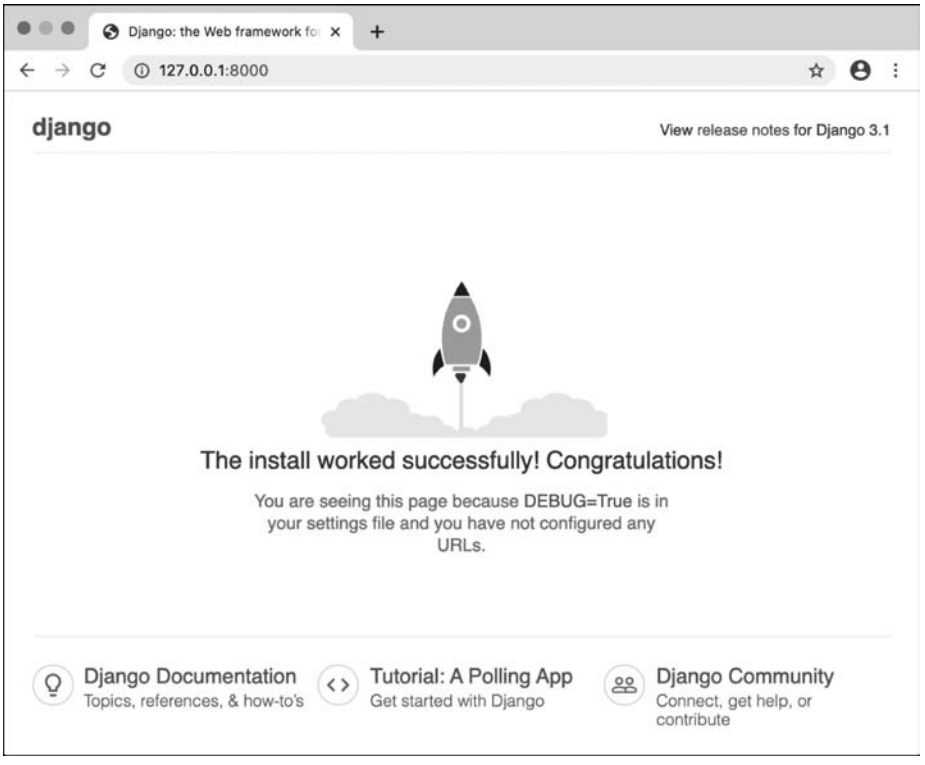

图 5-20　Django 站点首页

5.6　Quart

Quart 是一个基于 asyncio 的 Python Web 开发框架，提供了一种简单的在 Web 开发中使用 asyncio 的方式。

这里需要说明一下，为什么我们要了解这么多框架，因为目前 Python asyncio 生态正处于发展阶段，各种框架如雨后春笋般出现，呈现百家争鸣的状态。对于开发者来讲，不能押宝于任何一个框架，只能尝试学习更多的框架，才能在各种框架之间有一个实际的对比能力，在进行技术选型的时候能够更加准确，而不至于给项目后期带来不利影响。同时更应该

去学习和掌握框架实现技术，从而拥有解决框架本身问题的能力，或者直接在项目中采用自己所开发的框架，以杜绝依赖第三方技术所带来的不稳定因素。

使用 Quart 之前需要先使用命令 pip install quart 安装该依赖项，之后创建一个文件名为 server.py 的文件，实现一个最简页面请求处理功能，代码如下：

```
"""
第 5 章/quart_app/server.py
"""
from quart import Quart

#创建一个 Quart 应用
app = Quart(__name__)


#处理对站点根路径的请求
@app.route('/')
async def hello():
    #向前端返回字符串
     return 'hello'

app.run()
```

通过命令 python server.py 启动该服务器，如图 5-21 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP quart_app % python server.py
Running on http://127.0.0.1:5000 (CTRL + C to quit)
[2020-09-11 23:01:28,790] Running on 127.0.0.1:5000 over http (CTRL + C to quit)
```

图 5-21 启动 Quart 服务器

此时可以通过地址 http://127.0.0.1:5000 访问该站点，如图 5-22 所示。

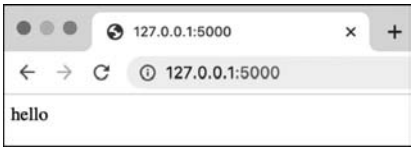

图 5-22 页面访问结果

此时可以发现，Quart 是可以独立运行的，就像 AIOHTTP 一样，这是因为 Quart 的依赖项中有 Hypercorn（一个 ASGI 协议层实现，与 Uvicorn、Daphne 是同类型竞品，用法类似），在安装 Quart 时会自动安装 Hypercorn，而当使用 app.run 函数启动服务器时，会默认使用 Hypercorn 来启动服务器。

也可以使用单独的 Hypercorn 去启动服务器，只需删除 app.run() 这行代码，源码如下：

```
"""
第 5 章/quart_app/asgi.py
"""
```

```
from quart import Quart

#创建一个 Quart 应用
app = Quart(__name__)


#处理对站点根路径的请求
@app.route('/')
async def hello():
    #向前端返回字符串
    return 'hello'
```

接下来通过命令 hypercorn server:app 启动服务器,如图 5-23 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP quart_app % hypercorn server:app
Running on 127.0.0.1:8000 over http (CTRL + C to quit)
```

图 5-23　使用 hypercorn 命令启动服务器

在安装 Uvicorn 之后也可以使用命令 uvicorn server:app 启动服务器,如图 5-24 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP quart_app % uvicorn server:app
INFO:     Started server process [5943]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

图 5-24　使用 uvicorn 命令启动服务器

同样在安装 Daphne 之后也可以使用命令 daphne server:app 启动服务器,如图 5-25 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP quart_app % daphne server:app
2020-09-11 23:26:23,761 INFO     Starting server at tcp:port=8000:interface=127.0.0.1
2020-09-11 23:26:23,761 INFO     HTTP/2 support enabled
2020-09-11 23:26:23,761 INFO     Configuring endpoint tcp:port=8000:interface=127.0.0.1
2020-09-11 23:26:23,762 INFO     Listening on TCP address 127.0.0.1:8000
```

图 5-25　使用 daphne 命令启动服务器

5.7　Starlette

Starlette 是一个轻量级的高性能应用层框架,构建于 ASGI 之上,框架本身不带 ASGI 协议层,所以运行时需要手动安装协议层依赖项,例如 uvicorn。

要使用 Starlette,需要先用命令 pip install starlette 安装该框架,如图 5-26 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP starlette_app % pip install starlette
Requirement already satisfied: starlette in ./venv/lib/python3.8/site-packages (0.13.8)
(venv) yunp@yunps-MBP starlette_app %
```

图 5-26 安装 Starlette

创建一个名为 server.py 的文件，在其中输入 Hello World 示例代码如下：

```
"""
第 5 章/starlette_app/server.py
"""
from starlette.applications import Starlette
from starlette.responses import HTMLResponse
from starlette.routing import Route


async def homepage(request):
    return HTMLResponse("Hello World")


app = Starlette(debug=True, routes=[
    Route('/', homepage),
])
```

若要运行该服务器，需要先使用命令 pip install uvicorn 安装 Uvicorn，也可以尝试使用其他的协议层实现，如 Daphne 或者 Hypercorn。在安装 Uvicorn 之后需要使用命令 uvicorn server:app 启动服务器，如图 5-27 所示。

```
Terminal:  Local ×  +
(venv) yunp@yunps-MBP starlette_app % ucivorn server:app
zsh: command not found: ucivorn
(venv) yunp@yunps-MBP starlette_app % uvicorn server:app
INFO:     Started server process [9962]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

图 5-27 启动 Starlette 服务器