我们经常编写的Python代码程序通常在服务器（后端环境）上运行，处理数据、执行业务逻辑、与数据库等后端服务交互。这种情况下，它是典型的“后端服务”。Python在网络编程领域的发展历程大致经历了从基本的HTTP服务器和CGI处理，到WSGI标准的制定，再到现在广泛使用的成熟Web框架。具体来看：

1. 早期的HTTP服务器和CGI脚本

最开始，Python与前端的交互方式比较原始且不方便。不过，也存在一些方法和技术可以实现前后端的交互，比如我们在前期会尝试使用：

- **基本的CGI（Common Gateway Interface）**

Web服务器与后端程序的交互主要通过CGI实现。Python脚本可以作为CGI程序运行，处理HTTP请求数据，执行所需的逻辑，并直接向客户端输出HTML或其他格式的响应。这种方式直接、简单，但效率较低，因为每次HTTP请求都需要启动一个新的进程来处理。

- **自定义Python服务器**

Python的后端开发者会使用Python标准库中的模块（如http.server或更早的SimpleHTTPServer和BaseHTTPServer）来创建简单的HTTP服务器。这些服务器能够接收HTTP请求，然后通过编写Python代码来处理这些请求并返回响应。这种方法提供了很大的灵活性，但通常缺乏用于生产环境的必要特性和安全性。

- **Python模块与库的使用**

进一步地，Python可以利用各种库来处理Web请求，如使用`urllib`和`xmlrpc.client`等库编写客户端应用程序。这些库支持与HTTP服务的交互，但主要用于客户端编程，不适合作为服务器端处理HTTP请求。

> 客户端编程通常指的是在客户端设备上运行的程序的编写和开发，这些设备可以是用户的计算机、手机或任何其他终端设备。客户端程序主要负责与用户直接交互，处理用户界面和用户输入，并可能与远端服务器通信来获取数据或执行任务。

上述方法基本上都是处理HTTP请求并返回响应的简单机制，但通常效率不高、功能有限，且不易于扩展。

为了解决这些问题，并统一Python Web应用与各种Web服务器之间的接口，PEP 333（Python Enhancement Proposal 333）提出了WSGI（Web Server Gateway Interface）。

2. WSGI的出现

WSGI的引入是一个重大转变，它定义了一个标准化的接口，使得Web服务器和应用之间的通信更加规范和高效。WSGI允许开发者编写可以与任何支持WSGI的服务器配合工作的应用，从而使Python Web开发更加灵活和强大。WSGI（Web Server Gateway Interface）的设计初衷是为了解决Python Web服务器和应用之间的兼容问题，提供一个标准化的接口。WSGI的主要目的包括：

1. **标准化接口**：在WSGI标准出现之前，Python的Web开发环境相对碎片化，不同的Web框架和Web服务器之间很难直接交互。WSGI提供了一个标准的接口，允许Web服务器和Web应用之间有一个统一的通信协议。这样，开发者就可以选择任何支持WSGI的服务器来部署他们的Web应用，而不用担心底层的兼容性问题。

2. **简化开发**：WSGI使得开发人员可以更专注于应用的开发而不是底层的细节。开发者可以使用任何WSGI兼容的Web框架来开发应用，而无需关心服务器的细节，反之亦然。
3. **中间件支持**：WSGI还支持所谓的“中间件”组件，这些组件可以执行请求和响应的处理任务，如会话管理、认证、数据压缩等。这意味着可以插入额外的功能，而无需改变应用本身的代码。
4. **可扩展性和可插拔性**：WSGI的接口设计允许多种组件（服务器、应用、中间件）之间的灵活组合和匹配，从而使得Web应用的部署和扩展变得更加灵活。

基于WSGI标准，开始出现了更加复杂和功能丰富的Web框架。这些框架如Django、Flask等不仅实现了WSGI接口，还提供了许多开箱即用的功能，如ORM（对象关系映射）、表单处理、模板渲染和会话管理等。这些框架极大地简化了Web应用的开发过程，让开发者可以更专注于业务逻辑而不是底层细节。在WSGI出现之前，不同的Python Web框架（如Django, Flask等）和Web服务器（如Apache, Nginx等）之间的交互通常需要特定的适配器或者中间件，这导致了许多兼容性和效率问题。

- Django makes it easier to build better web apps more quickly and with less code：https://www.djangoproject.com/
- Flask provides configuration and conventions, with sensible defaults, to get started：https://flask.palletsprojects.com/en/3.0.x/

WSGI 专注于后端的部分，即如何在 Python 应用程序和 Web 服务器之间建立一个清晰、标准化的接口。这使得开发者可以选择适合他们需求的框架和服务器，而无需担心两者之间的兼容性问题。WSGI应用可以完全后端驱动，也可以仅仅作为前后端分离模式中后端部分的实现。但WSGI（Web Server Gateway Interface）是Python中传统的、同步的Web服务器与应用程序之间的标准接口。框架如Flask和Django（在Django 3.0之前）都使用WSGI。它适合处理不需大量并发处理的应用，但在高并发和实时数据处理方面表现不佳。

3. 异步编程和现代框架

随着网络应用对性能和并发的需求日益增长，出现了支持异步编程的框架，典型框架就是：Starlette和FastAPI。这些框架基于ASGI（异步服务器网关接口），是WSGI的异步版本。它们可以更高效地处理并发请求，允许服务器和应用程序非阻塞方式通信，从而处理异步程序。适合构建需要高性能和大规模并发处理的现代Web应用。

总的来说，Python的网络编程从基本的HTTP服务器和CGI脚本发展到了现在的成熟Web框架，这个过程中WSGI的制定起到了关键的桥梁作用。现代的Web框架不仅提供了丰富的功能和组件，还极大地提高了开发效率和应用性能。

4. Starlette

Starlette 是一个轻量级的 ASGI（异步服务器网关接口）框架，它提供了构建异步Web应用的基础设施。Starlette 可以单独使用来构建应用程序，也可以作为其他异步Web框架（如 FastAPI）的基础。它提供了许多有用的功能，如请求和响应类、路由系统、中间件支持等。此外，Starlette 也支持WebSocket，是构建实时应用程序的一个好选择。其官方文档：https://www.starlette.io/

Starlettet可以单独使用，用于构建Web应用和服务，包括但不限于：

- 异步请求处理
- WebSocket 支持
- 事件生命周期管理

- .....

Starlette 提供了构建现代Web应用所需的基本工具，但它的设计更注重灵活性和性能，不包含一些高层次的抽象。但需要注意的是：Starlette仅仅是一个提供了构建异步 Web 应用所需的各种工具和组件，本身不包含服务器的功能，它需要一个 ASGI 兼容的服务器来运行应用程序。

我们还需要启动Uvicorn、Daphne 和 Hypercorn 这些 ASGI 兼容的服务器，它们用来接收客户端（如 Web 浏览器）的网络请求，将这些请求转发给后端的 ASGI 应用（如用 Starlette 构建的应用），并将应用的响应返回给客户端。

Starlettet可以单独使用，用于构建Web应用和服务，包括但不限于：

- 异步请求处理
- WebSocket 支持
- 事件生命周期管理
- .....

Starlette 提供了构建现代Web应用所需的基本工具，但它的设计更注重灵活性和性能，不包含一些高层次的抽象。但需要注意的是：Starlette仅仅是一个提供了构建异步 Web 应用所需的各种工具和组件，本身不包含服务器的功能，它需要一个 ASGI 兼容的服务器来运行应用程序。

我们还需要启动Uvicorn、Daphne 和 Hypercorn 这些 ASGI 兼容的服务器，它们用来接收客户端（如 Web 浏览器）的网络请求，将这些请求转发给后端的 ASGI 应用（如用 Starlette 构建的应用），并将应用的响应返回给客户端。

运行测试请查看课件代码。

5. FastAPI

FastAPI 和 Starlette 的关系并不是网络编程接口协议与基于该协议的框架的关系，而是两个不同层次的Web框架之间的关系。具体来说，FastAPI 是基于 Starlette 构建的，可以看作是 Starlette 的一个高级封装，提供了更多易用的功能和更丰富的工具集，特别是在创建API服务方面。

Starlette 提供了构建现代Web应用所需的基本工具，但它的设计更注重灵活性和性能，不包含一些高层次的抽象。FastAPI 是一个现代、快速（高性能）的Web框架，用于构建API，基于Python 3.6+ 的类型提示特性，主要优势包括：

- 自动生成交互式API文档（基于Swagger和ReDoc）
- 基于Python类型提示的请求参数和响应体的验证
- 易于使用的依赖注入系统
- 对异步和同步代码的支持
- 易于扩展和高性能

FastAPI 在 Starlette 的基础上提供了更多的高级功能，尤其是在自动数据验证、序列化和API文档生成方面。这些功能使得开发复杂的API服务变得更简单、更快速，同时保持了高性能。

因此，可以说 FastAPI 是在 Starlette 的基础上增加了额外的功能和更高层次的抽象，以便更专注于快速API开发。Starlette 提供了底层的异步处理能力和Web功能，而 FastAPI 则利用这些基础构建了更完整的开发框架，特别是为了更好地服务于API开发。这种关系类似于 Flask 和 Flask-RESTful 的关系，后者在前者的基础上增加了便于创建RESTful API的工具和功能。