# Flask 基本概念

以下是 Flask 的主要基本概念的详细解析：

　　1.路由：路由是 URL 到 Python 函数的映射。Flask 允许你定义路由，这样当特定的 URL 被访问时，就会调用相应的函数。

　　2.视图函数：视图函数是处理请求并返回响应的 Python 函数。它们通常接收请求对象作为参数，并返回响应对象。

　　3.请求对象：请求对象包含了客户端发送的请求信息，如请求方法、URL、请求头、表单数据等。

　　4.响应对象：响应对象包含了发送给客户端的响应信息，如状态码、响应头、响应体等。

　　5.模板：Flask 使用 Jinja2 模板引擎来渲染 HTML 模板。模板允许你将 Python 代码嵌入到 HTML 中，从而动态生成网页。

　　6.应用工厂：应用工厂是一个 Python 函数，它创建并返回一个 Flask 应用实例。这允许你配置和初始化你的应用，并且可以创建多个应用实例。

　　7.配置对象：Flask 应用有一个配置对象，你可以使用它来设置各种配置选项，如数据库连接字符串、调试模式等。

　　8.蓝图：蓝图是 Flask 中的一个组织代码的方式，它允许你将相关的视图函数、模板和静态文件组织在一起，并且可以在多个应用中重用。

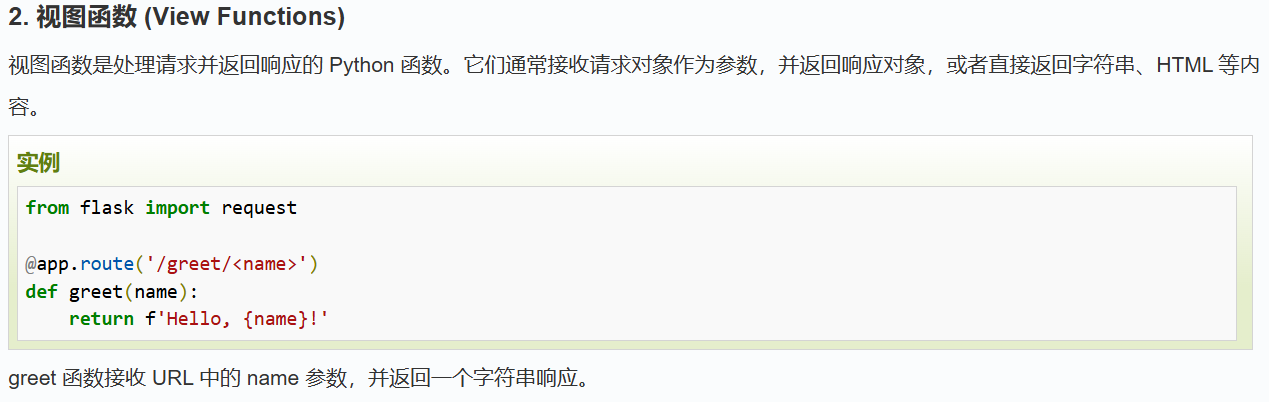
　　9.静态文件：静态文件是不会被服务器端执行的文件，如 CSS、JavaScript 和图片文件。Flask 提供了一个简单的方法来服务这些文件。

　　10.扩展：Flask 有许多扩展，可以添加额外的功能，如数据库集成、表单验证、用户认证等。

　　11.会话：Flask 使用客户端会话来存储用户信息，这允许你在用户浏览你的应用时记住他们的状态。

　　12.错误处理：Flask 允许你定义错误处理函数，当特定的错误发生时，这些函数会被调用。

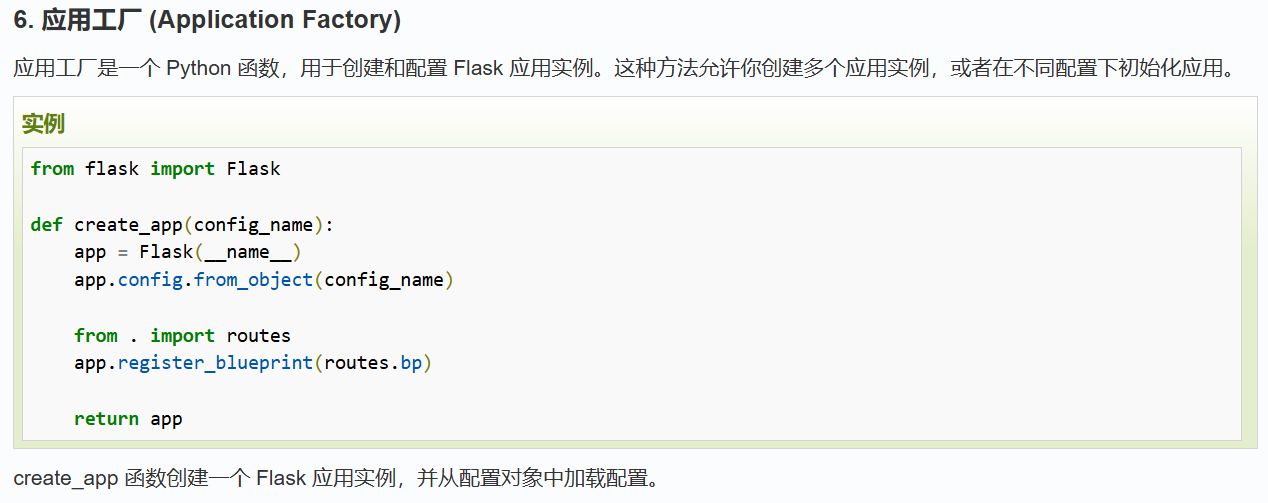












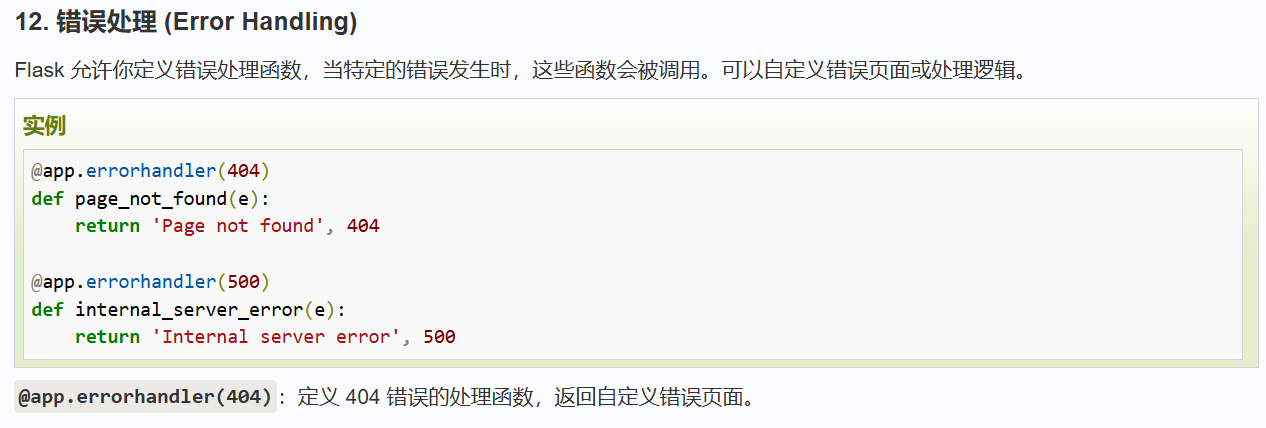












# Flask项目简介

以下是几种常见的 Flask 项目结构，分别适用于简单应用和较复杂的应用：

1.简单项目结构：适用于小型应用，所有代码集中在一个文件中。

2.中型项目结构：将应用分为多个模块，便于管理和扩展。

3.复杂项目结构：支持更高的模块化，适用于大型应用，包含路由、模型、模板和静态文件目录。











# Flask 路由

以下是关于 Flask 路由的说明，包括路由的定义、参数、方法和规则等：

　　1.定义路由：使用@app.route('/path')装饰器定义 URL 和视图函数的映射。

　　2.路由参数：通过动态部分在 URL 中传递参数。

　　3.路由规则：使用类型转换器指定 URL 参数的类型。

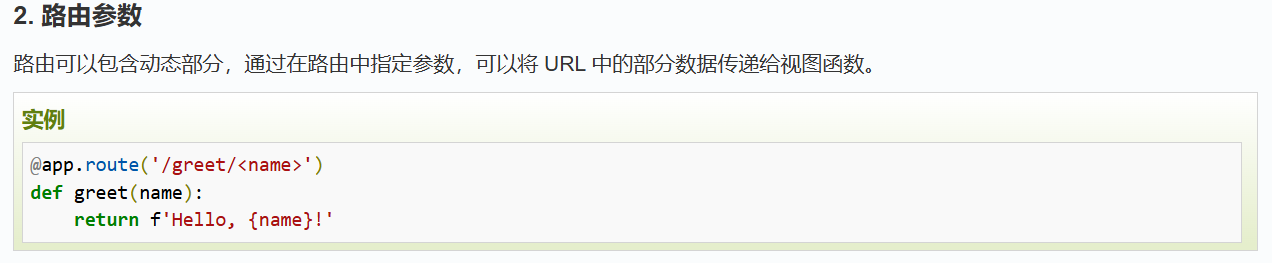
　　4.请求方法：指定允许的 HTTP 请求方法。

　　5.路由函数返回：视图函数可以返回不同类型的响应。

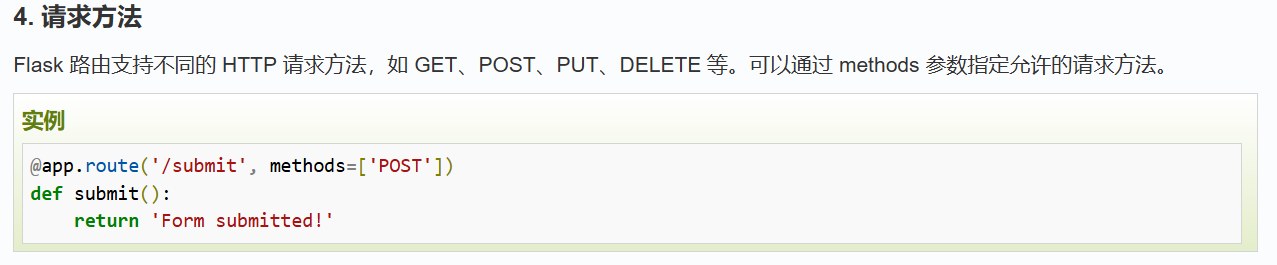
　　6.静态文件和模板：管理静态文件和动态渲染 HTML 模板。

7.路由优先级：确保路由顺序正确，以避免意外地匹配结果。





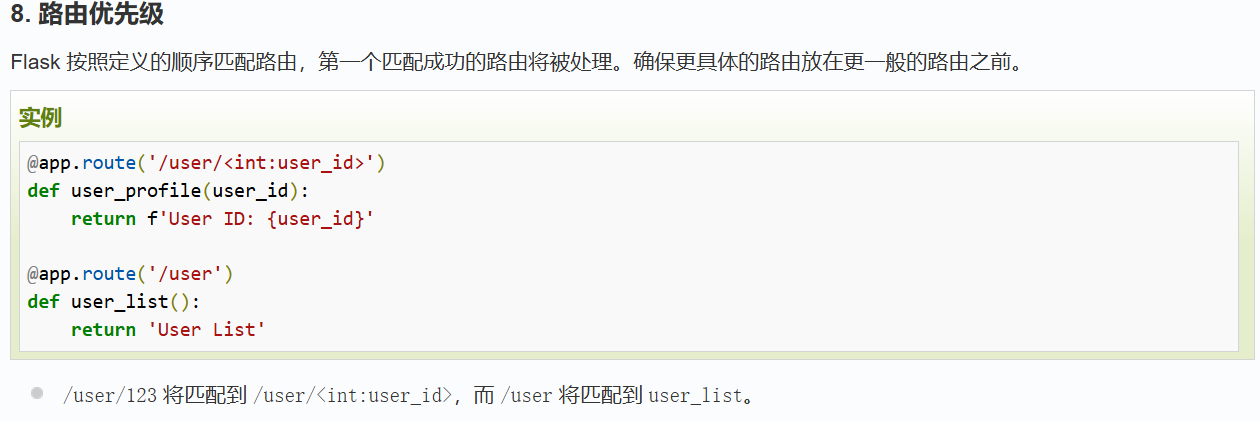












# Flask视图函数

以下是对 Flask 视图函数的详细说明，包括如何定义、使用请求数据、返回响应、以及如何处理错误等：

1.定义视图函数：视图函数是处理请求并返回响应的核心功能。

2.接收请求数据：使用 request 对象获取 URL 参数、表单数据、查询参数等。

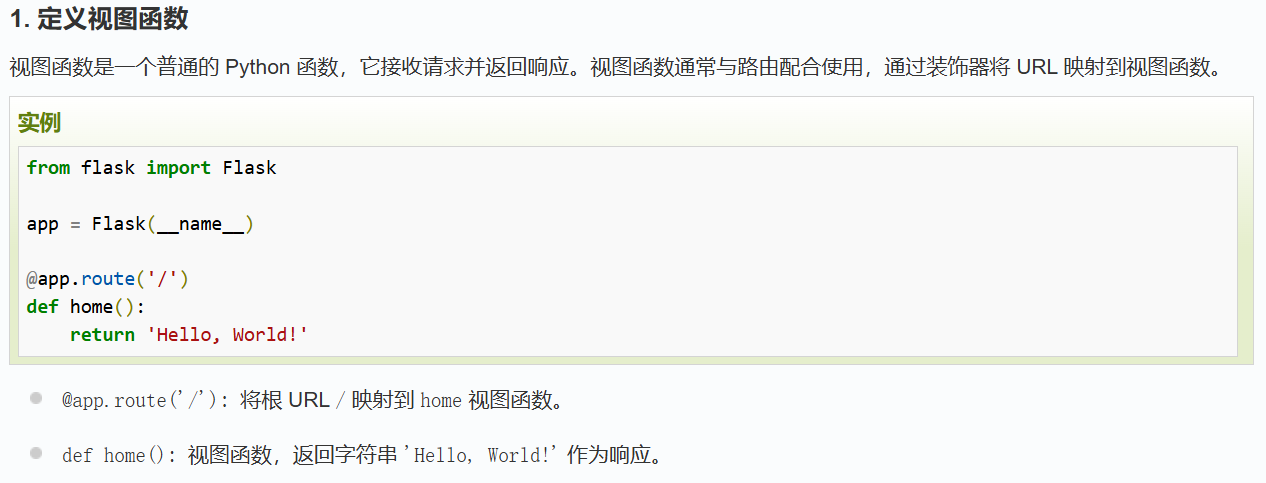
3.返回响应：可以返回字符串、HTML、JSON 或自定义响应对象。

4.处理请求和响应：使用 request 对象和 make\_response 来处理请求和生成自定义响应。

5.处理错误：视图函数内处理异常或使用 Flask 的错误处理机制。

6.视图函数的装饰器：使用 @app.before\_request、@app.after\_request 等装饰器处理请求前后逻辑。

7.视图函数返回的状态码：可以指定 HTTP 状态码来表示请求的处理结果。

















# Flask模板渲染

以下是关于 Flask 模板渲染的详细说明，包括如何创建和使用模板、模板继承、控制结构等：

1.基本概念：Flask 使用 Jinja2 模板引擎来处理模板渲染。模板渲染允许你将动态内容插入到 HTML 页面中，使得应用能够生成动态的网页内容。

2.创建模板：将 HTML 文件放在 templates 文件夹中，使用 Jinja2 占位符。

3.渲染模板：使用 render\_template 函数在视图函数中渲染模板。

4.模板继承：创建基础模板，允许其他模板继承和扩展。

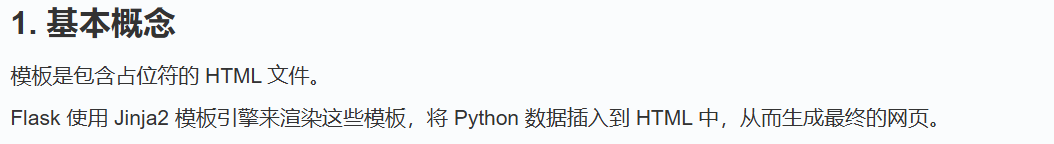
5.控制结构：使用条件语句和循环在模板中控制逻辑。

6.过滤器：使用过滤器格式化变量数据。

7.宏和模板包含：创建和使用宏以及模板包含，提高模板的复用性。

8.安全性：Jinja2 默认对模板变量进行自动转义以防止 XSS 攻击。

9.模板上下文：将数据传递给模板，并在模板中使用这些数据。

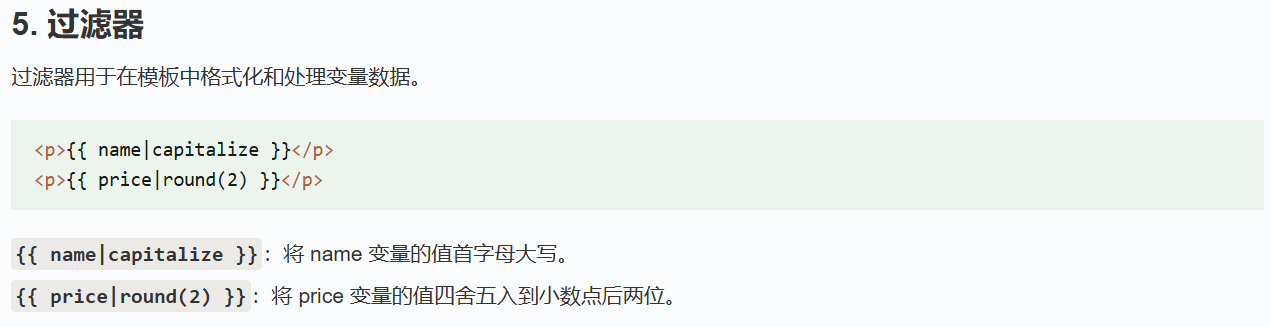




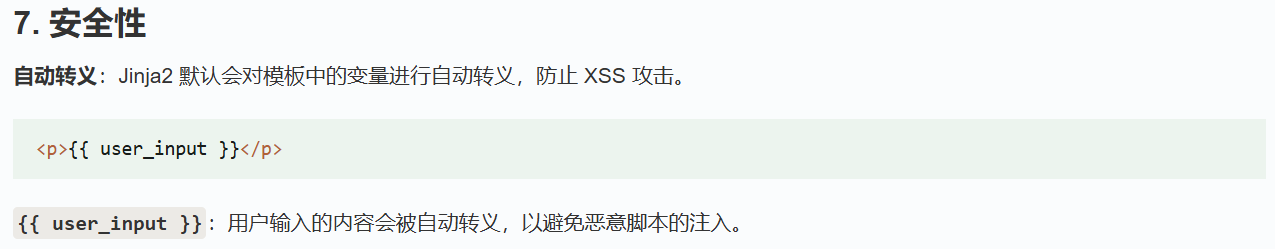














# Flask 表单处理

在 Flask 中，表单处理是构建 Web 应用时一个常见的需求。处理表单数据涉及到接收、验证和处理用户提交的表单。Flask 提供了基本的表单处理功能，但通常结合 Flask-WTF 扩展来简化表单操作和验证：

　　1.基本表单处理：使用 request.form 获取表单数据。

　　2.使用 Flask-WTF：结合 WTForms 进行表单处理和验证，简化表单操作。

　　3.表单验证：使用验证器确保表单数据的有效性。

　　4.文件上传：处理文件上传和保存文件。

　　5.CSRF 保护：确保表单免受跨站请求伪造攻击。

















# Flask数据库操作

Flask 提供了多种方式来与数据库进行交互，包括直接使用 SQL 和利用 ORM（对象关系映射）工具，如 SQLAlchemy。以下是对 Flask 数据库操作的详细说明，包括使用 SQLAlchemy 的基本操作和 SQL 语句的直接执行：

1.使用 SQLAlchemy：定义模型，配置数据库，执行基本的 CRUD 操作。

2.创建和管理数据库：使用 db.create\_all() 创建表。

3.CRUD 操作：添加、读取、更新和删除记录。

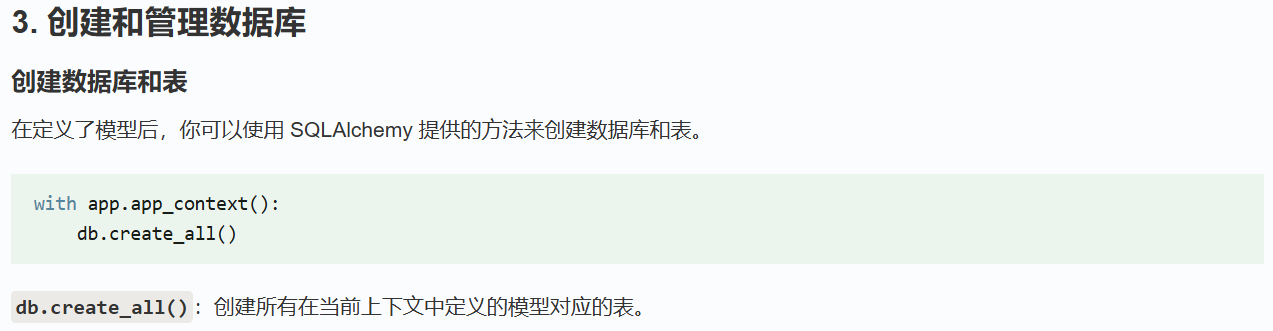
4.查询操作：执行基本和复杂查询，包括排序和分页。

5.Flask-Migrate：使用 Flask-Migrate 管理数据库迁移。

1. 执行原始 SQL：使用原始 SQL 语句进行数据库操作。













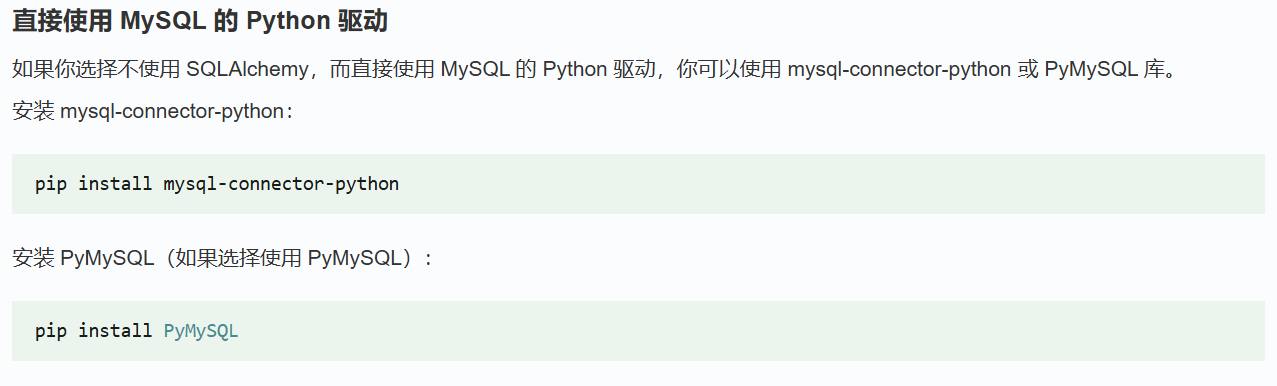


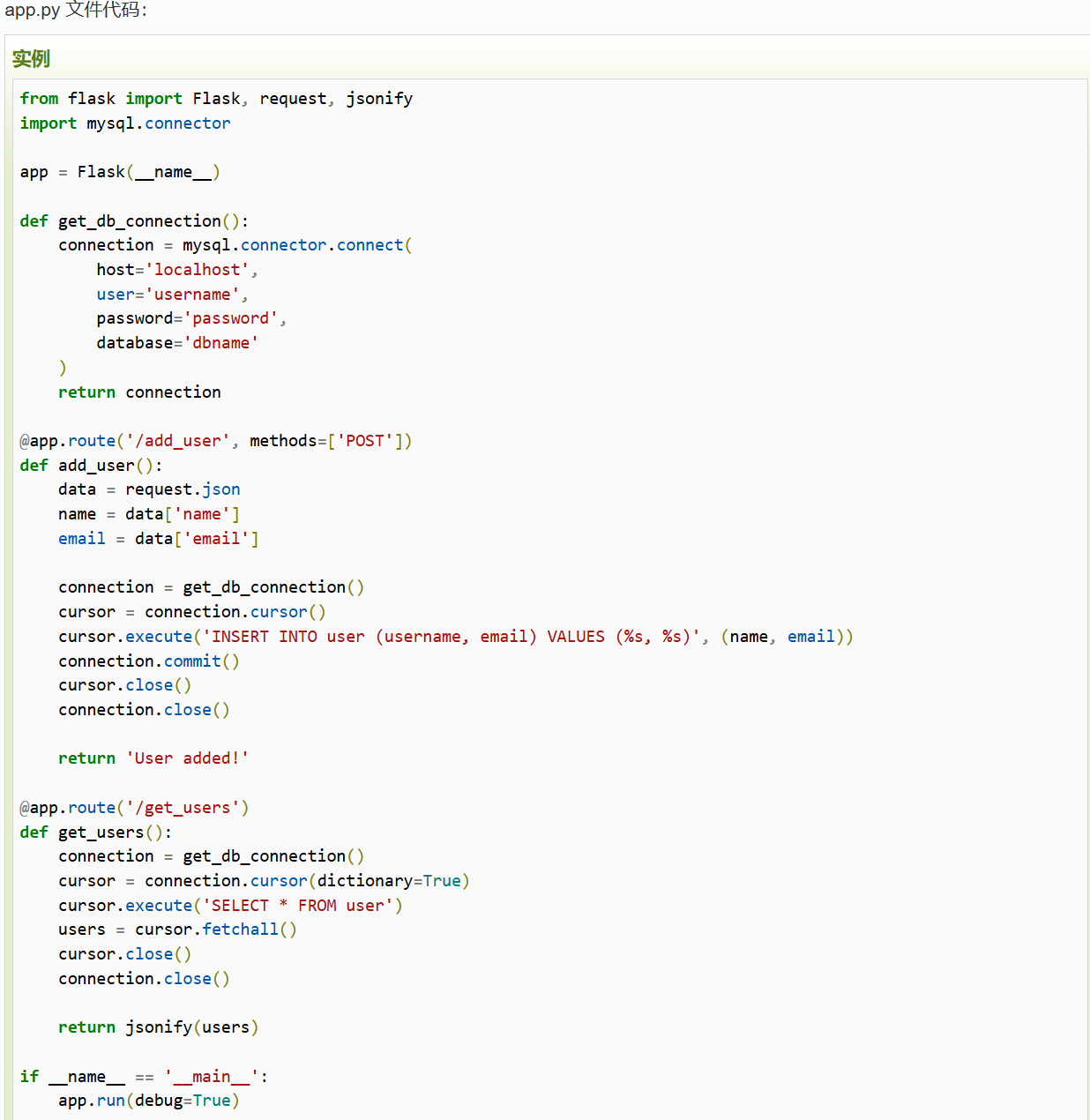


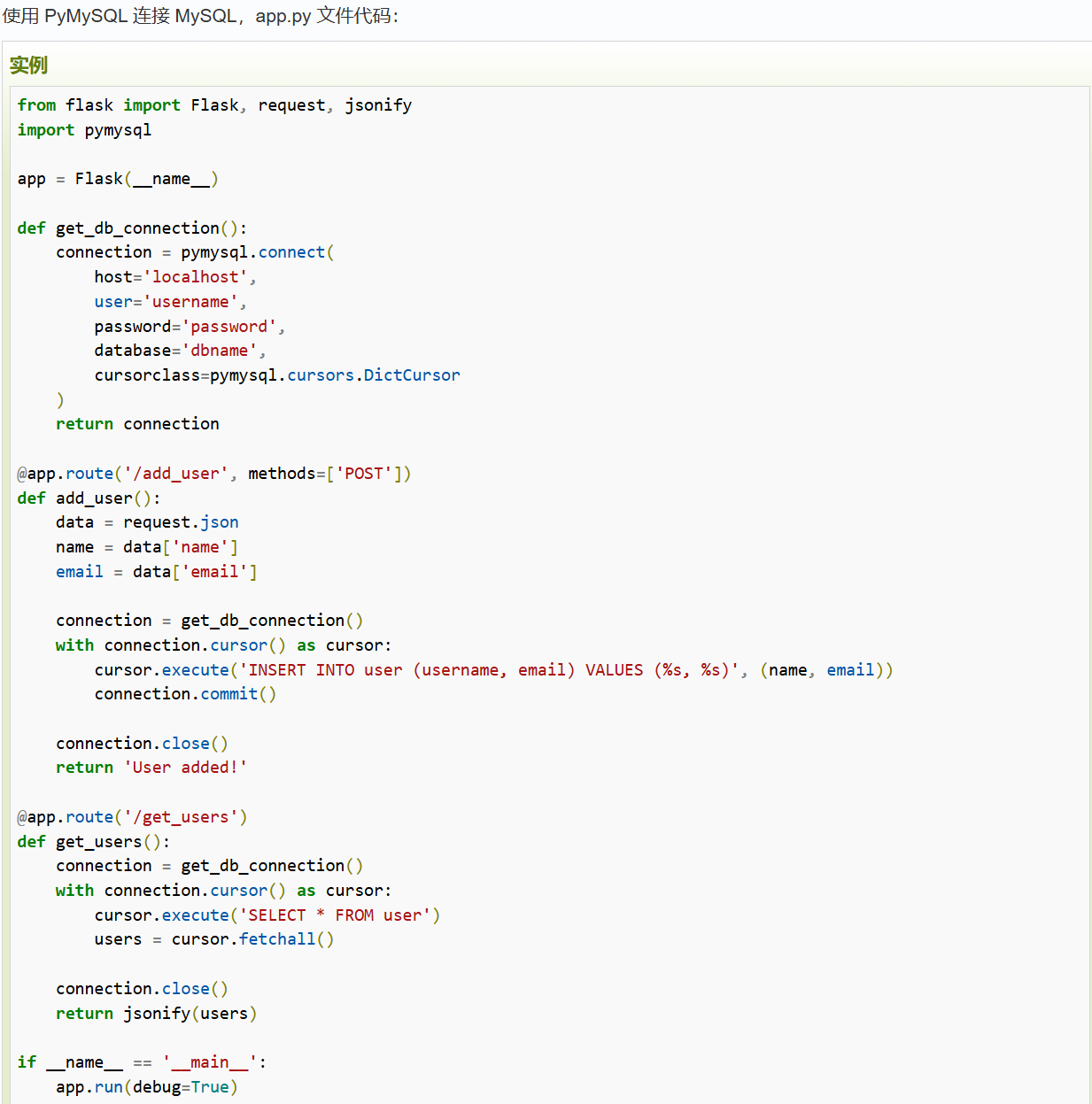












# Flask 蓝图 (Blueprints)

Flask 的蓝图（Blueprints）是一种组织代码的机制，允许你将 Flask 应用分解成多个模块。每个蓝图可以有自己的路由、视图函数、模板和静态文件，这样可以将相关的功能分组。通过使用蓝图，你可以将 Flask 应用拆分成多个模块，每个模块处理相关的功能，使得代码更加清晰和易于管理：

1.创建蓝图：在独立的模块中定义蓝图，并指定路由和视图函数。

2.注册蓝图：在主应用中注册蓝图，并设置路由前缀。

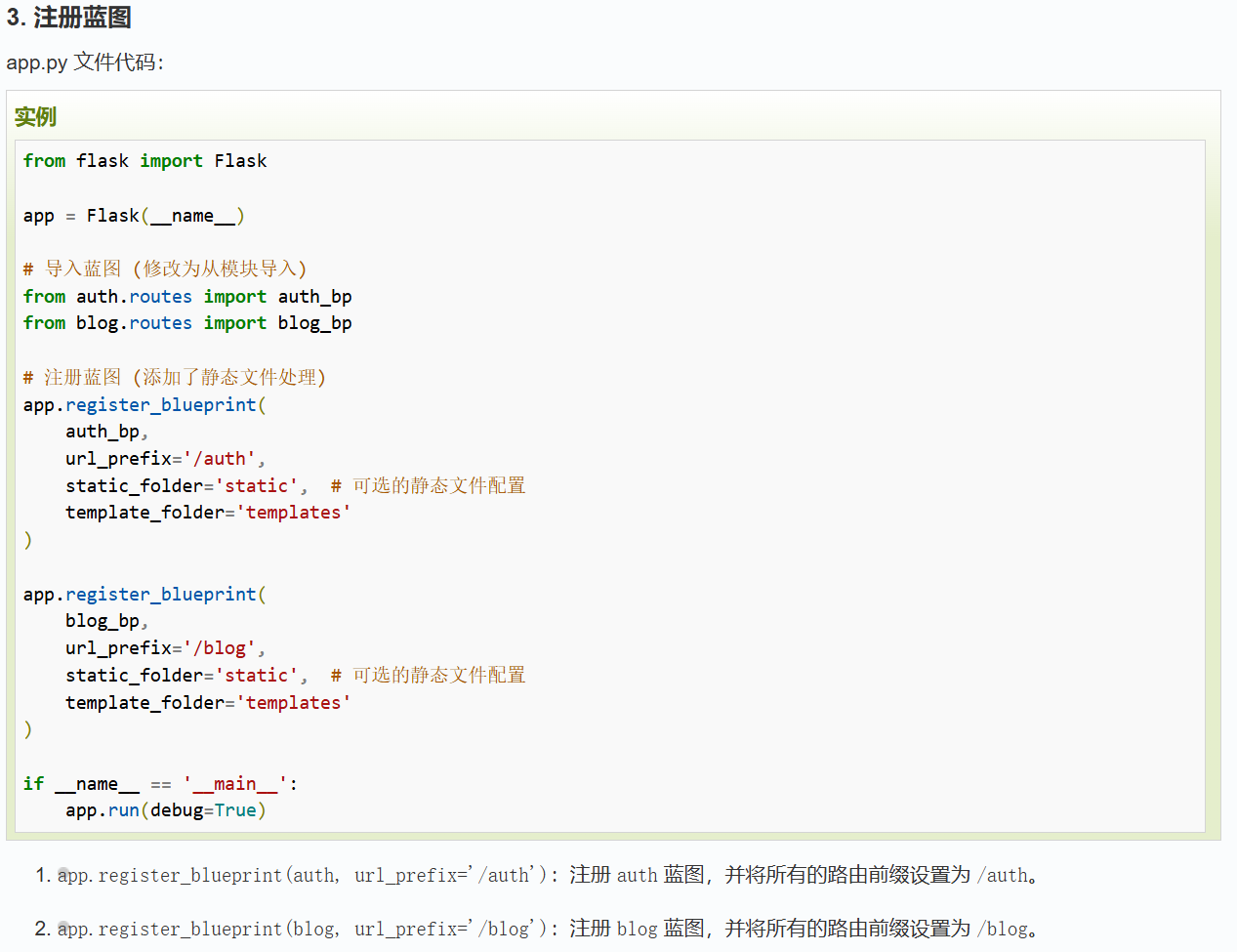
3.使用蓝图中的模板和静态文件：将模板和静态文件放在蓝图的 templates 和 static 文件夹中。

4.使用请求钩子和错误处理：在蓝图中定义请求钩子和错误处理函数。



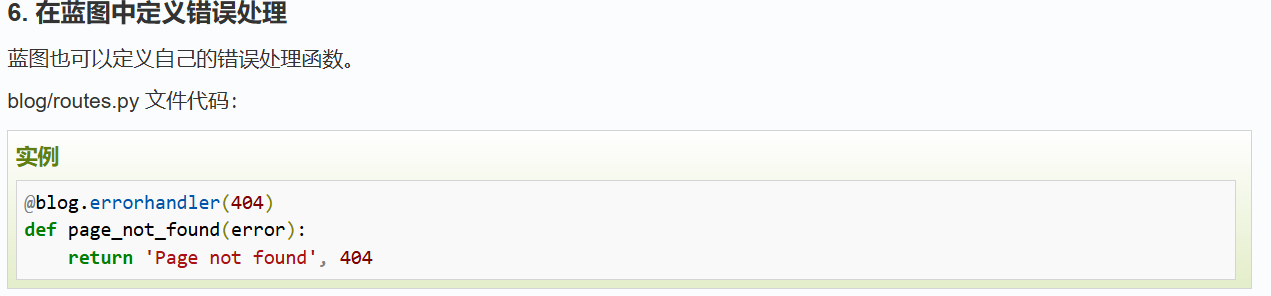












# Flask 错误处理

Flask 提供了灵活的错误处理机制，可以捕获并处理应用中的各种错误。以下是详细的说明，涵盖了如何定义和处理错误，如何处理 HTTP 状态码以及如何处理自定义错误：

1.处理 HTTP 错误：使用 @app.errorhandler 装饰器定义针对特定 HTTP 状态码的错误处理函数。

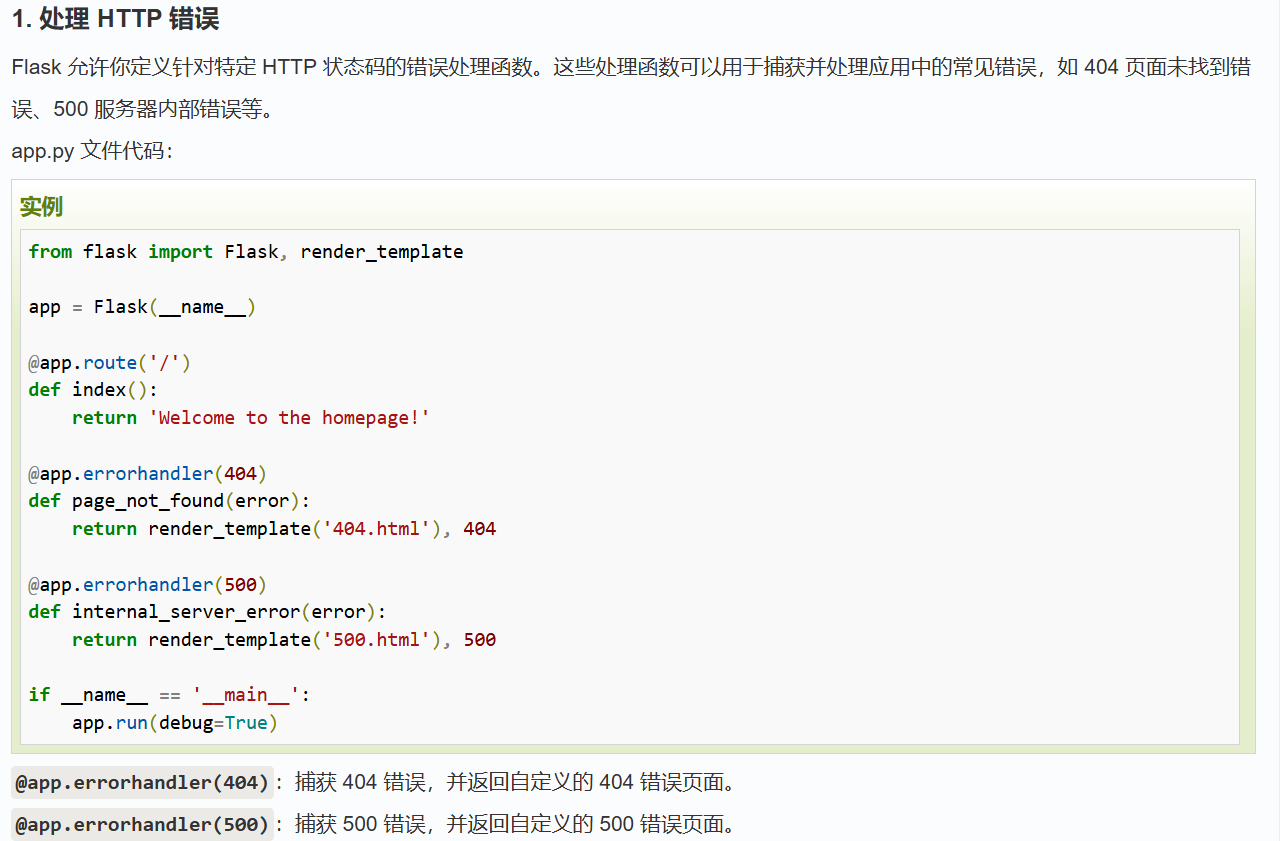
2.蓝图中的错误处理：在蓝图中定义错误处理函数，允许模块化的错误处理。

3.自定义错误：定义自定义异常类，并在应用中捕获和处理这些异常。

4.全局错误处理：使用全局错误处理函数捕获所有未处理的异常。

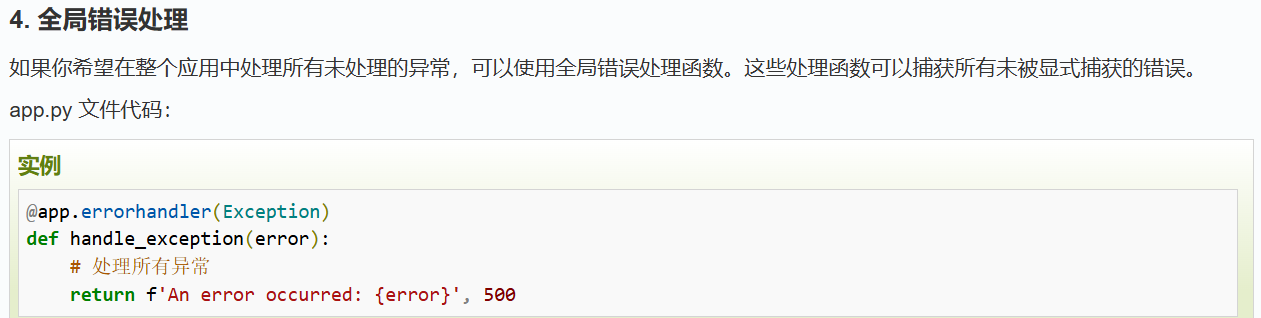
5.使用 abort 函数：在视图函数中主动触发 HTTP 错误。

6.渲染自定义错误页面：为每个错误码创建自定义的 HTML 错误页。













# Flask 中间件和扩展

Flask 中间件：

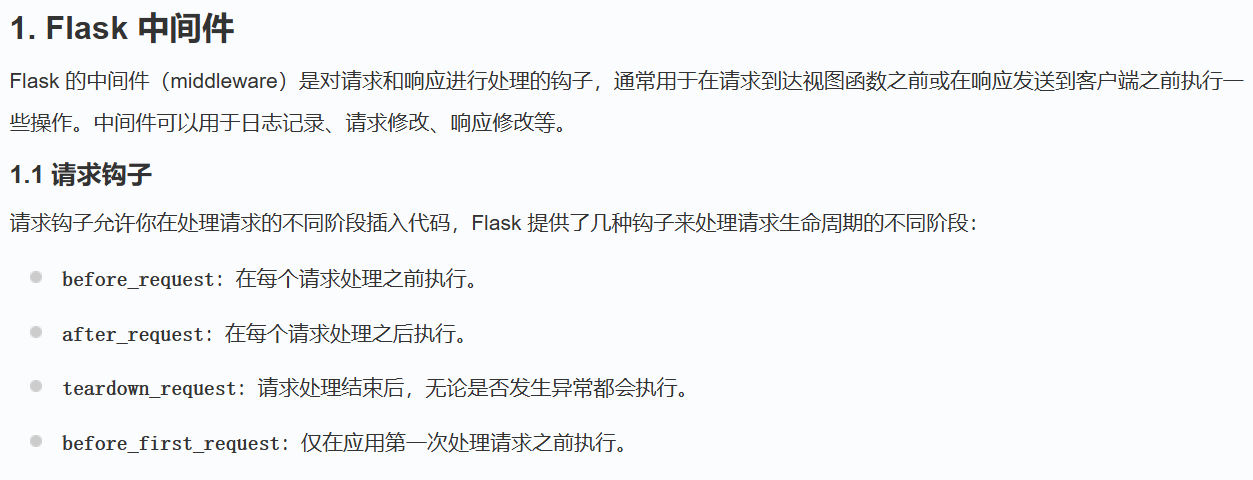
1.使用请求钩子在请求处理的不同阶段插入代码。

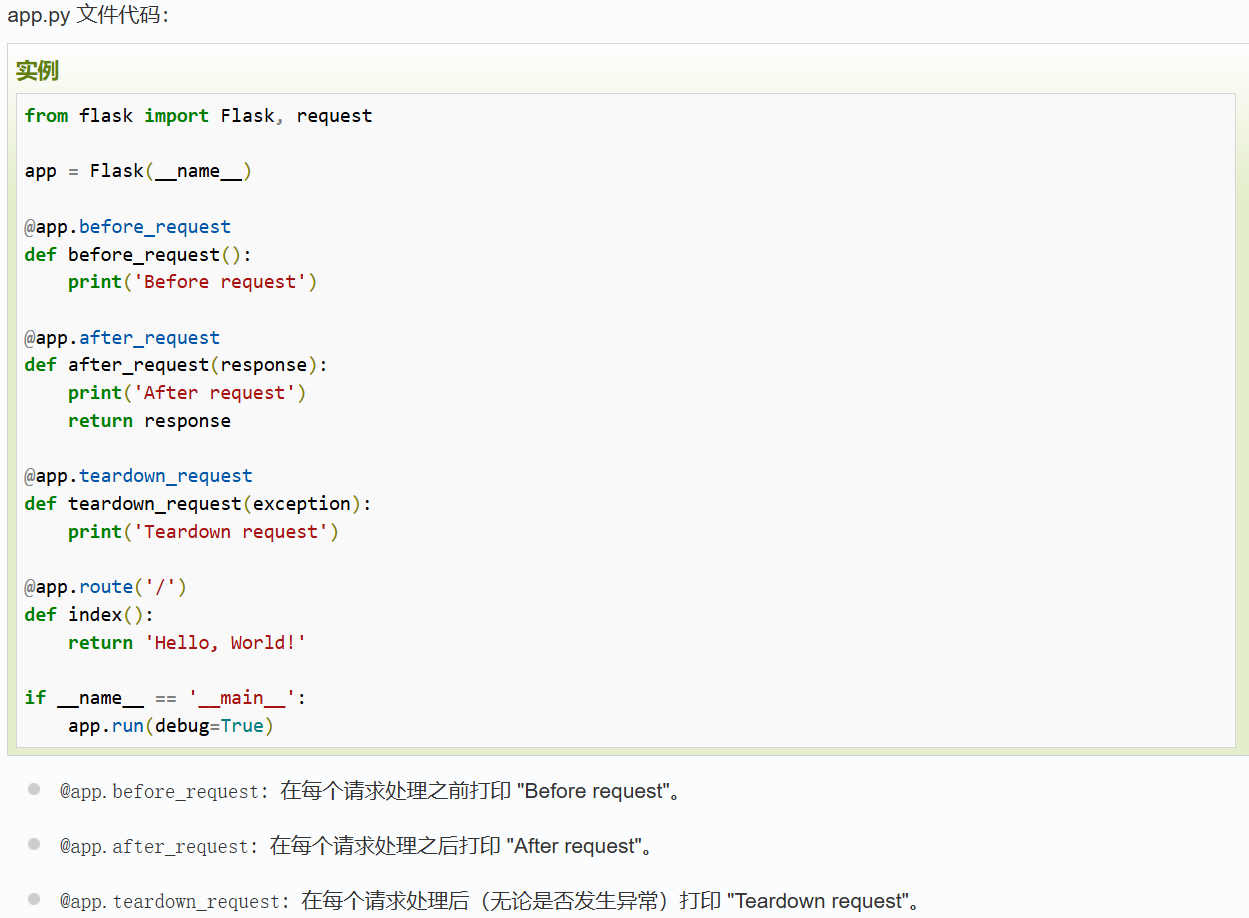
2.创建自定义中间件类来处理请求和响应。

Flask 扩展：

1.使用现有的扩展（如 Flask-SQLAlchemy、Flask-WTF）来添加功能。

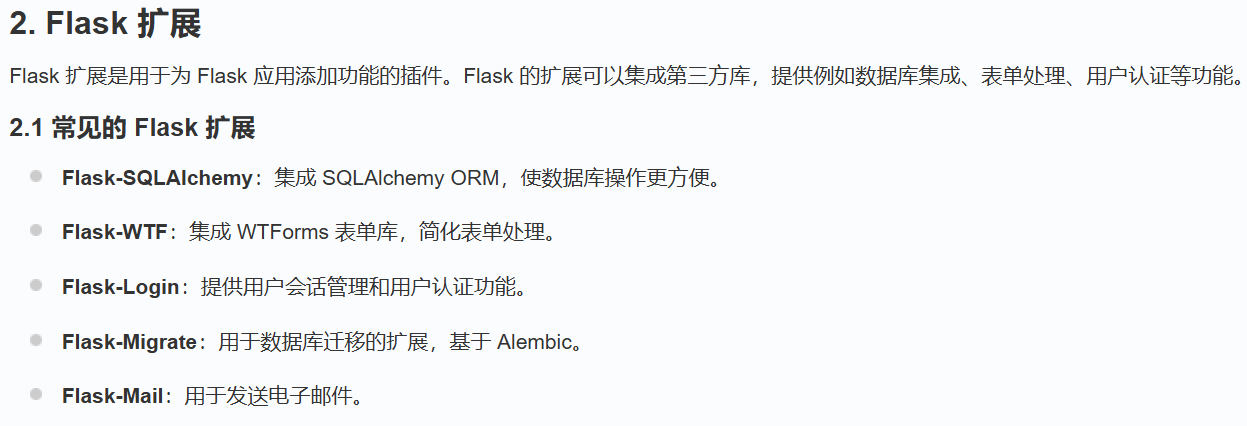
1. 创建自定义扩展来满足特定需求。





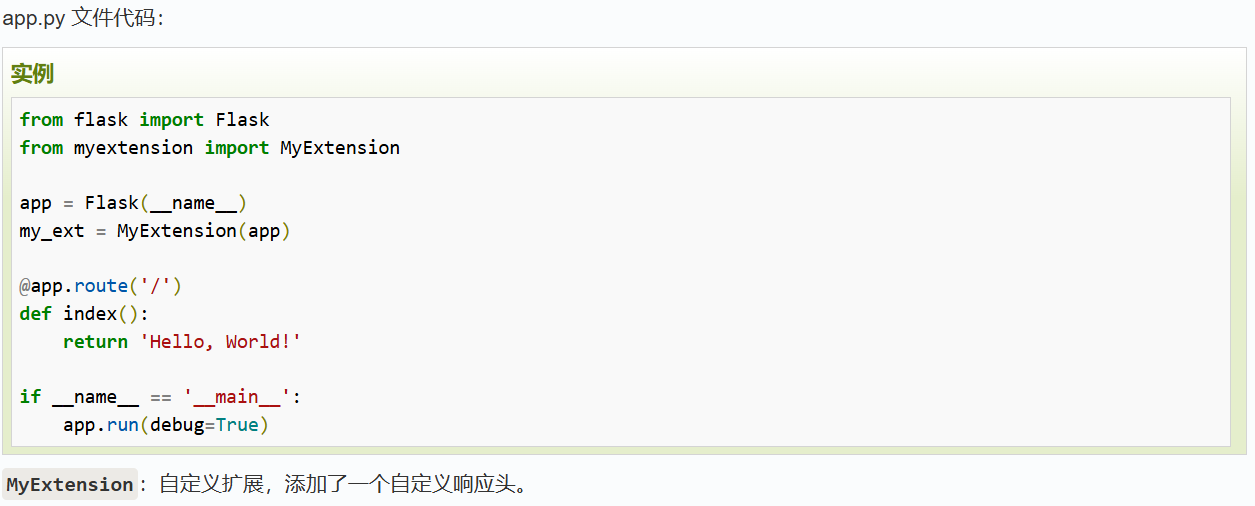












# Flask 部署

Flask 部署是将你的 Flask 应用程序发布到生产环境中的过程，使其可以被用户访问。部署 Flask 应用涉及选择合适的服务器和环境配置。以下是常见的 Flask 部署方法和步骤，包括使用 WSGI 服务器和 Web 服务器，以及如何在不同平台上部署 Flask 应用。

选择部署方式：

1.使用 Gunicorn、uWSGI 或 Waitress 作为 WSGI 服务器。

2.使用 Nginx 或 Apache 作为反向代理服务器。

3.可以选择在 Heroku 或 Docker 上部署。

部署步骤：

1.安装并配置 WSGI 服务器。

2.配置 Web 服务器（如 Nginx）。

3.可选：在 Heroku 或 Docker 上部署。

监控和维护：

1.监控应用的性能和日志。

1. 定期更新依赖和应用代码。

