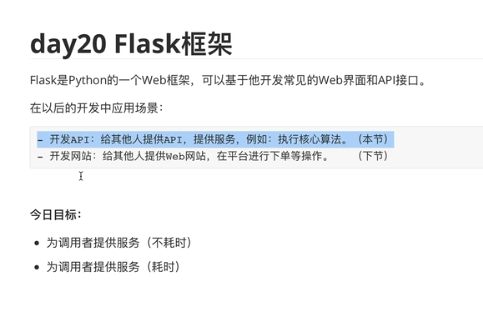
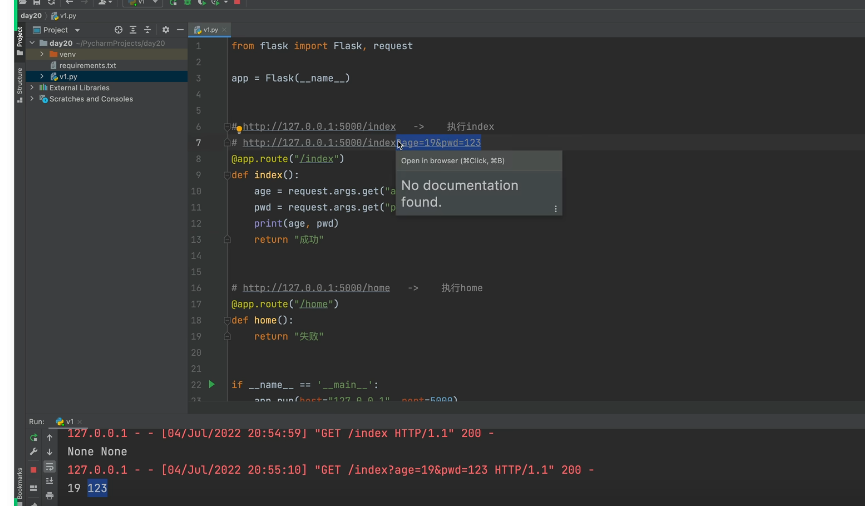
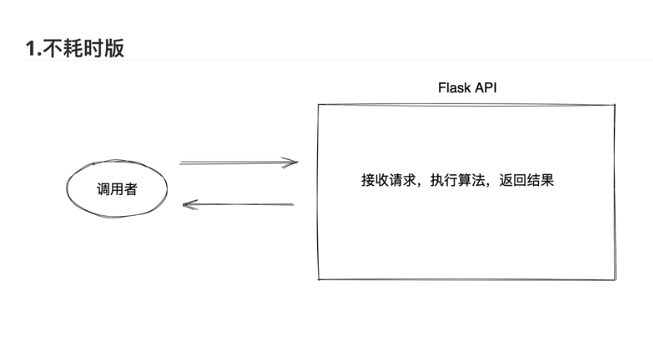
# Flask





接收url的参数

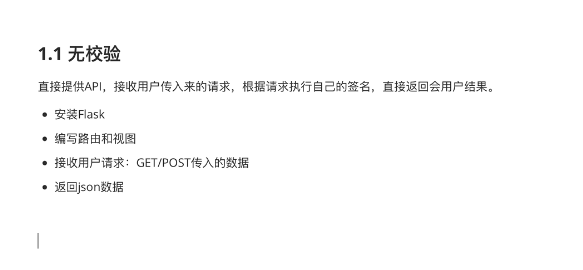


获取请求数据

返回json数据



直接访问API并返回结果



# 官网

https://flask.palletsprojects.com/zh-cn/stable/

# 基本概念

Flask是一个用Python编写的轻量级Web应用框架，被称为"微框架"(microframework)。

# 核心特点

## 1、轻量级设计

Flask的核心非常简洁，只提供Web开发的基础功能，如路由、模板渲染、请求处理等。它不像Django那样"大而全"，而是采用"小而美"的设计理念。

## 2、高度灵活

Flask给开发者很大的自由度，你可以根据项目需求选择合适的组件和架构。它不强制使用特定的数据库、模板引擎或其他工具。

## 3、可扩展性

Flask 的设计允许你通过插件和扩展来添加功能。许多常见的功能，如表单处理、数据库交互和用户认证，都可以通过社区提供的扩展来实现。

## 4、内置开发服务器

Flask 内置了一个开发服务器，方便在本地进行调试和测试。

## 5、RESTful 支持

Flask 支持 RESTful API 的开发，适合构建现代的 Web 服务和应用程序。

# 主要功能

## 1、路由系统

1、支持多种HTTP方法(GET、POST、PUT等)

2、支持URL参数和查询参数

3、灵活的路由规则定义

## 2、模版引擎

1、内置Jinja2模板引擎

2、支持模板继承和包含

3、提供丰富的过滤器和函数

## 3、请求处理

1、简单易用的请求对象

2、支持文件上传、表单处理

3、内置会话(Session)管理

## 扩展生态

Flask通过扩展(Extensions)来增强功能：

1、Flask-SQLAlchemy: 数据库ORM支持

2、Flask-Login: 用户认证管理

3、Flask-WTF: 表单处理和CSRF保护

4、Flask-Mail: 邮件发送功能

5、Flask-Admin: 后台管理界面

## 适用场景

### 1、小型项目

（1）原型开发和快速验证

（2）简单的API服务

（3）个人博客或小型网站

### 2、微服务架构

（1）构建轻量级的微服务

（2）API网关和中间件服务

（3）特定功能的独立服务

### 3、学习和教学

（1）Python Web开发入门

（2）理解Web框架工作原理

（3）快速搭建演示项目

## 优点

### 1、简单易学

Flask的API设计直观，文档清晰，新手容易上手。相比Django，学习曲线更加平缓。

### 2、灵活性强

没有固定的项目结构要求，开发者可以根据需要组织代码，选择合适的工具和库。

### 3、社区活跃

拥有丰富的第三方扩展和活跃的社区支持，遇到问题容易找到解决方案。

## Flask组成

### 1、Flask 应用实例

Flask 的核心是应用实例，通过创建 Flask 对象来初始化应用。

### 2、路由和视图函数

路由将 URL 映射到视图函数，视图函数处理请求并返回响应。

### 3、模板系统

Flask 使用 Jinja2 模板引擎来渲染 HTML 页面，将数据动态插入到页面中。

### 4、请求和响应

Flask 处理 HTTP 请求并生成响应，支持多种 HTTP 方法（如 GET、POST）

## 安装

### 基本说明

Flask 安装还是比较简单的。Flask 是 Python的一个库，所以首先需要确保你的计算机上已经安装了 Python。Flask 需要 Python 3.6 及以上版本，先确保你已安装 Python 3。

可以通过以下命令检查 Python 版本：

python --version

或者，如果你使用 python3 命令：

python3 --version

### 使用pip安装Flask

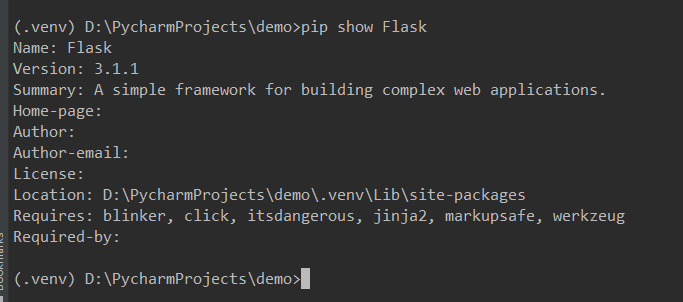
我们可以使用 Python 的包管理器 pip 可以用来安装Flask。打开你的命令行工具（在 Windows 上是命令提示符或 PowerShell，在 macOS 或 Linux 上是终端），然后运行以下命令：

pip install Flask

安装完成后，可以通过以下命令验证 Flask 是否安装成功：

pip show Flask

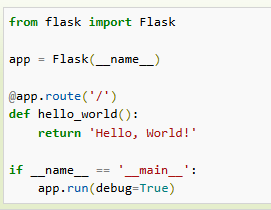
执行以上命令，显示结果类似如下：



这样我们就成功安装了 Flask 包。

## Flask第一个应用示例

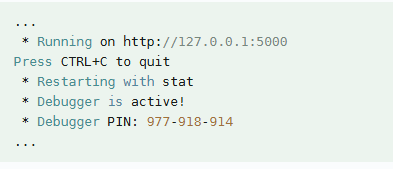
首先，创建一个名为 app.py 的文件，并添加以下内容：



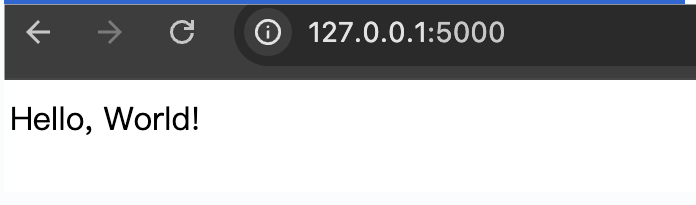
在命令行中运行 Flask 应用：

python app.py

你会看到 Flask 开发服务器启动，并显示类似于以下内容



打开浏览器，访问 http://127.0.0.1:5000/，应该会看到 "Hello, World!" 的消息，表示 Flask 已成功安装并运行。



## Flask框架基本概念

### 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。



**@app.route('/')**：将根 URL / 映射到 home 函数。

**@app.route('/about')**：将 /about URL 映射到 about 函数。

### 视图函数

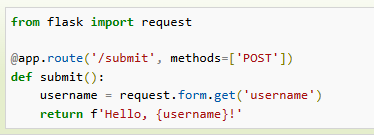
视图函数是处理请求并返回响应的 Python 函数。它们通常接收请求对象作为参数，并返回响应对象。



greet 函数接收 URL 中的 name 参数，并返回一个字符串响应。

### 请求对象

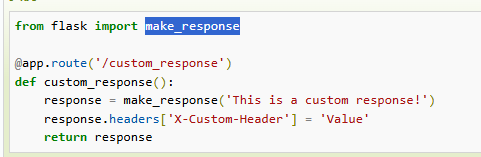
请求对象包含了客户端发送的请求信息，如请求方法、URL、请求头、表单数据等。



request.form.get('username')：获取 POST 请求中表单数据的 username 字段。

### 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。Flask 默认会将字符串、HTML 直接作为响应体。



**make\_response**：创建一个自定义响应对象，并设置响应头 X-Custom-Header。

### 模版

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。

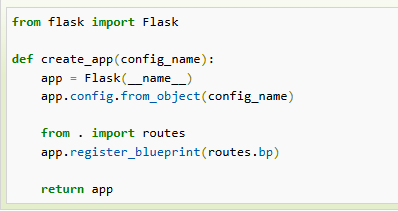


模板文件 (templates/hello.html)：



### 应用工厂

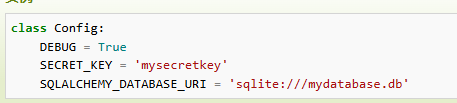
应用工厂是一个 Python 函数，它创建并返回一个 Flask 应用实例。这允许你配置和初始化你的应用，并且可以创建多个应用实例。



create\_app 函数创建一个 Flask 应用实例，并从配置对象中加载配置。

### 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。可以通过直接设置或加载配置文件来配置 Flask 应用。



app.config.from\_object(Config)：将 Config 类中的配置项加载到应用配置中。

### 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。

### 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。

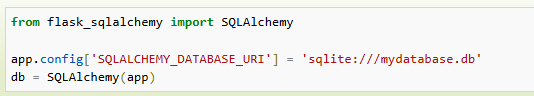
访问静态文件示例：



静态文件目录：将静态文件放在 static 文件夹中，Flask 会自动提供服务。

### 扩展

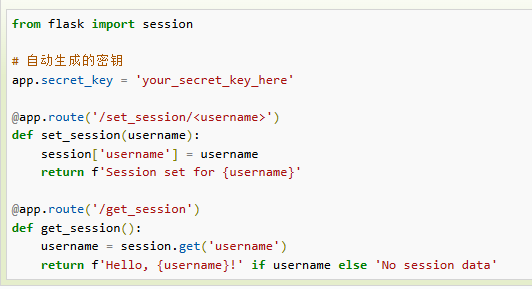
Flask 有许多扩展，可以添加额外的功能，如数据库集成、表单验证、用户认证等。



SQLAlchemy：用于数据库集成的扩展。

### 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。会话数据存储在客户端的 cookie 中，并在服务器端进行签名和加密。

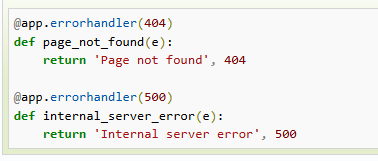


session 对象用于存取会话数据。你可以使用 Python 内置的 secrets 模块生成一个强随机性的密钥。

python3 -c 'import secrets; print(secrets.token\_hex())'

### 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。



@app.errorhandler(404)：定义 404 错误的处理函数，返回自定义错误页面。

## 项目结构

### 基本概念

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。

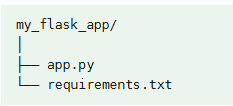
1、简单项目结构：适用于小型应用，所有代码集中在一个文件中。

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。

### 简单项目结构

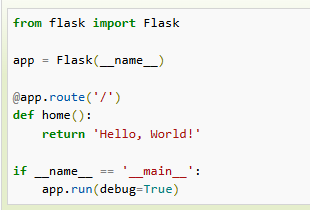
对于一个简单的 Flask 应用，项目结构可以非常简洁：



app.py：主要的 Flask 应用文件，包含路由和视图函数的定义。

requirements.txt：列出项目的依赖库，用于记录 Flask 和其他包的版本信息。

my\_flask\_app 目录下的 app.py 文件代码：

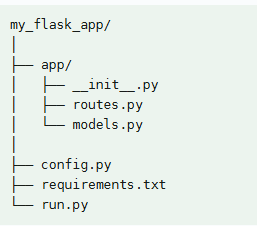


requirements.txt 示例：



### 中型项目结构

对于稍复杂的应用，通常会将项目分为多个模块和目录：



app/：包含 Flask 应用的主要代码。

\_\_init\_\_.py：初始化 Flask 应用和配置扩展。

routes.py：定义应用的路由和视图函数。

models.py：定义应用的数据模型。

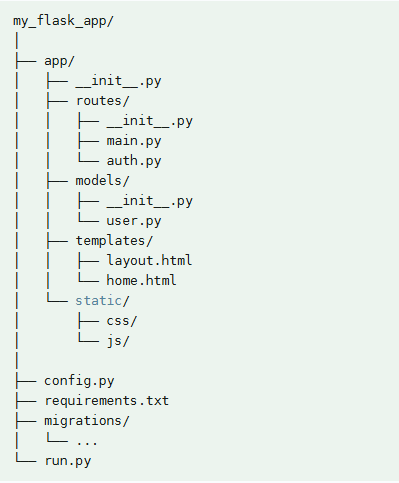
config.py：配置文件，包含应用的配置信息。

requirements.txt：列出项目的依赖库。

run.py：用于启动 Flask 应用。

### 复杂项目结构

对于较大的应用，可能需要更复杂的项目结构，以支持更高的模块化和扩展性



app/routes/：将不同功能模块的路由分开管理。

main.py：主模块的路由。

auth.py：认证相关的路由。

app/models/：管理数据模型，通常与数据库操作相关。

user.py：用户模型。

app/templates/：存放 HTML 模板文件。

app/static/：存放静态文件，如 CSS 和 JavaScript。

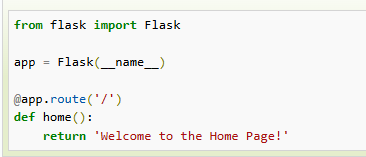
migrations/：数据库迁移文件，通常与 SQLAlchemy 相关。

## 路由

### 基本概念

Flask 路由是 Web 应用程序中将 URL 映射到 Python 函数的机制。Flask 路由是 Flask 应用的核心部分，用于处理不同 URL 的请求，并将请求的处理委托给相应的视图函数。

### 定义路由



@app.route('/')：装饰器，用于定义路由。/ 表示根 URL

def home()：视图函数，当访问根 URL 时，返回 'Welcome to the Home Page!'。

### 路由参数

路由可以包含动态部分，通过在路由中指定参数，可以将 URL 中的部分数据传递给视图函数。



### 路由规则

路由规则支持不同类型的参数和匹配规则。

#### 类型规则

字符串（默认）： 匹配任意字符串。

整数（<int:name>）： 匹配整数值。

浮点数（<float:value>）： 匹配浮点数值。

路径（<path:name>）： 匹配任意字符，包括斜杠 /。

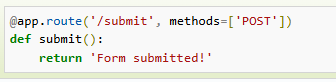


@app.route('/user/<int:user\_id>')：匹配整数类型的 user\_id。

@app.route('/files/<path:filename>')：匹配包含斜杠的路径 filename。

### 请求方法

Flask 路由支持不同的 HTTP 请求方法，如 GET、POST、PUT、DELETE 等。可以通过 methods 参数指定允许的请求方法。



### 路由转换器

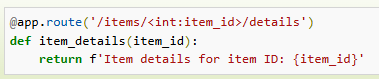
Flask 提供了一些内置的转换器，可以对 URL 中的参数进行特定类型的转换。

常用转换器：

int： 匹配整数。

float： 匹配浮点数。

path： 匹配任意路径，包括斜杠。



<int:item\_id>：将 URL 中的 item\_id 转换为整数。

### 路由函数返回

视图函数可以返回多种类型的响应

#### 字符串

返回纯文本响应。

#### HTML

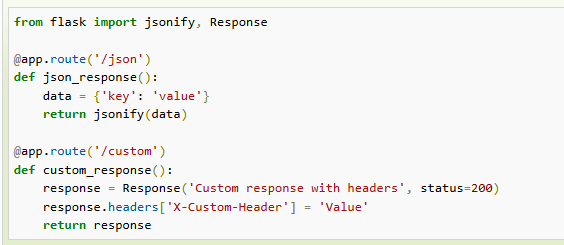
返回 HTML 页面。

#### JSON

返回 JSON 数据。

#### Response 对象

自定义响应。



jsonify(data)：将字典转换为 JSON 响应。

Response('Custom response with headers', status=200)：创建自定义响应对象。

### 静态文件和模版

静态文件（如 CSS、JavaScript、图片）可以通过 static 路由访问。模板文件则通过 templates 文件夹组织，用于渲染 HTML 页面。

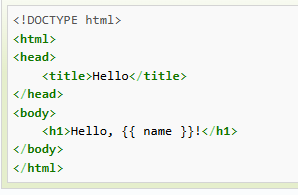
静态文件访问：





模版文件渲染

模板文件 (templates/hello.html)：



### 路由优先级

Flask 按照定义的顺序匹配路由，第一个匹配成功的路由将被处理。确保更具体的路由放在更一般的路由之前。



/user/123 将匹配到 /user/<int:user\_id>，而 /user 将匹配到 user\_list。

## 视图函数

视图函数是 Flask 应用中的核心部分，它负责处理请求并生成响应。视图函数与路由紧密结合，通过路由将 URL 映射到具体的视图函数。

### 定义视图函数

视图函数是一个普通的 Python 函数，它接收请求并返回响应。视图函数通常与路由配合使用，通过装饰器将 URL 映射到视图函数。



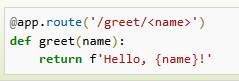
@app.route('/')：将根 URL / 映射到 home 视图函数。

def home()：视图函数，返回字符串 'Hello, World!' 作为响应。

### 接收请求数据

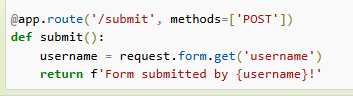
视图函数可以接收不同类型的请求数据，包括 URL 参数、表单数据、查询参数等。

#### 获取URL参数



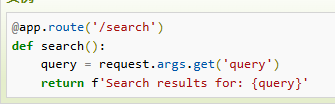
<name> 是一个 URL 参数，传递到视图函数 greet。

#### 获取表单数据



request.form.get('username')：获取 POST 请求中表单数据的 username 字段。

#### 获取查询参数



request.args.get('query')：获取 GET 请求中的查询参数 query。

#### 获取json数据



关键要点：

1、安全获取数据：使用 data.get('key') 而不是 data['key'] 避免KeyError

2、数据验证：始终验证必需字段和数据类型

3、错误处理：提供清晰的错误信息和适当的HTTP状态码

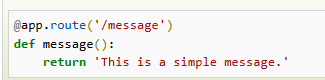
4、内容类型检查：使用 request.is\_json 检查请求格式

5、异常处理：用try-except包装JSON解析和处理逻辑

### 返回响应

视图函数可以返回多种类型的响应，包括字符串、HTML、JSON、或自定义响应对象。

#### 返回字符串

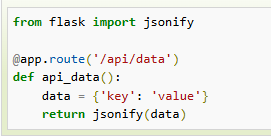


#### 返回HTML模版



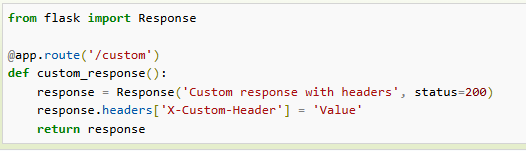
render\_template('hello.html', name=name)：渲染 HTML 模板 hello.html，并将 name 变量传递给模板。

#### 返回JSON数据



jsonify(data)：将 Python 字典转换为 JSON 响应。

#### 返回自定义响应对象

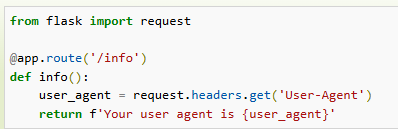


Response('Custom response with headers', status=200)：创建自定义响应对象，并设置响应头。

### 处理请求和响应

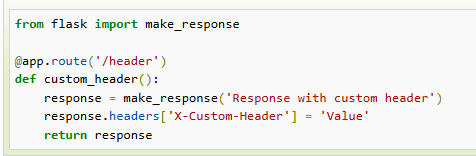
视图函数可以访问请求对象，并根据请求数据生成响应。可以使用 request 对象来获取请求的信息，使用 make\_response 来创建自定义响应。

#### 使用 request 对象



request.headers.get('User-Agent')：获取请求头中的 User-Agent 信息。

#### 使用 make\_response

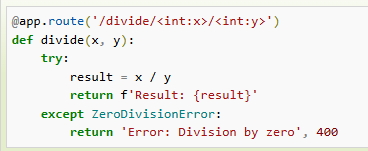


make\_response('Response with custom header')：创建响应对象并设置自定义头信息。

### 处理错误

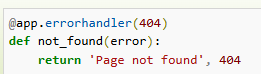
可以在视图函数中处理异常或错误，或者通过 Flask 提供的错误处理机制来处理应用中的错误。

#### 在视图函数中处理错误



使用 try-except 语句处理除零错误，并返回自定义错误消息和状态码。

#### 全局错误处理



@app.errorhandler(404)：定义处理 404 错误的函数。

### 视图函数的装饰器

除了 @app.route，Flask 还支持其他装饰器，用于实现更复杂的功能。

示例：

@app.before\_request：在每个请求处理之前运行的函数。

@app.after\_request：在每个请求处理之后运行的函数。

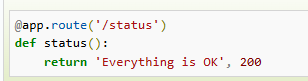
@app.teardown\_request：在请求结束后运行的函数，用于清理工作。

代码示例：



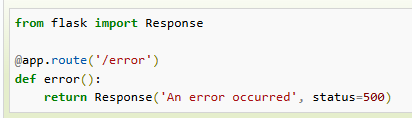
### 视图函数返回的状态码

视图函数不仅可以返回内容，还可以指定 HTTP 状态码。



返回状态码 200 表示请求成功

返回带有状态码的响应对象：



返回状态码 500 表示服务器内部错误。