1.1 fastapi介绍

简介：

- FastAPI 是一个**高性能，易于学习，高效编码，生产可用**的Python **Web异步框架**
- FastAPI 适合来写API，使用 Python 3.6+ 并基于标准的 Python 类型提示

核心特性：

- 速度快：可与 **NodeJS** 和 **Go** 比肩的极高性能（归功于 Starlette 和 Pydantic），asyncio
- 简单：易于学习，编码简单，文档学习简单，类型提示
- 生产可用：生产可用级别的代码。还有自动生成的交互式文档。
- OpenAPI文档：自带api文档，便于前后端对接（swagger和redoc）
- FastAPI适合前后端分离项目，轻量级项目
- 目前生态还在不断完善中

文档：

- 官方文档： https://fastapi.tiangolo.com
- 源码： https://github.com/tiangolo/fastapi

课程内容：

1. 学习Web开发的原理和方式
2. 学习FastAPI 框架特性，相对同步框架（如Django/Flask） 的优势
3. 学习各种请求参数和验证，如：路径参数、查询参数、请求体、cookie、header
4. 学习FastAPI 的表单数据处理、响应模型、文件处理、路径操作配置等
5. 学习 FastAPI 的依赖注入系统
6. 学习FastAPI 集成数据库，ORM的使用
7. 学习登录认证相关的基本原理和实现
8. 学习FastAPI 异步框架的使用方式
9. 学习FastAPI工程化应用，如何拆分文件
10. 学习如何部署FastAPI开发的web项目
11. 学习FastAPI的高级特性：中间件、后台任务、WewSocket、时间钩子等等

课程前置基础：

- python基础：https://www.51zxw.net/List.aspx?cid=929
- python进阶：https://www.51zxw.net/List.aspx?cid=1072
- python异步编程：https://www.51zxw.net/List.aspx?cid=1054
- web基础：网络协议/HTTP协议/前端基础等等（本课程中会补充相关基础知识点）

1.2 准备环境

系统环境

- Win10系统 + Python3.10
- 使用pycharm社区版做编辑器

准备环境

- 安装python3.10（https://www.python.org/downloads/windows/）
- 安装pycharm社区版编辑器（https://www.jetbrains.com/pycharm/download/#section=windows）

下载第三方包

- pip换源：

```
清华：https://pypi.tuna.tsinghua.edu.cn/simple/
阿里云：http://mirrors.aliyun.com/pypi/simple/
中国科技大学 ：https://pypi.mirrors.ustc.edu.cn/simple/
华中科技大学：http://pypi.hustunique.com/
豆瓣：http://pypi.douban.com/simple/
```

```
# 方式1:临时使用
pip3 install fastapi -i https://pypi.tuna.tsinghua.edu.cn/simple/


# 方式2：全局配置一次
pip3 config global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

- 下载FastAPI

```
pip3 install fastapi
```

- 下载uvicorn，用来启动fastapi项目。官网：https://www.uvicorn.org/

```
pip3 install uvicorn
```

1-3 Windows上使用虚拟环境

- 创建虚拟环境

```
#方式1：  直接使用pycharm中提供的虚拟环境工具


#方式2：  cmd窗口手动新建
python3 -m venv venv_name        # 创建虚拟环境venv_name
cd venv_name/Scripts             # 进入文件夹
activate                         # 激活虚拟环境
deactivate                       # 退出虚拟环境


# 初学者不推荐在powershell中操作
```

结论：

- 学习阶段，我们就直接使用全局的python解释器（虚拟环境）
- 如果你电脑上安装过其他版本的python，比如，python3.7等等。你再安装一个python3.10（多版本共存，设置好环境变量即可变方便使用），多版本共存设置参考视频：https://www.51zxw.net/Show.aspx?cid=1072&id=123521

1-4 pycharm环境使用问题

占位

1-5 第一个fastapi

编写第一个fastapi

```python
# main.py
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def root():
    return {"message": "Hello World"}
```

使用uvicorn启动项目

- 方式1：在cmd窗口中输入指令，启动fastapi项目（推荐）

```
uvicorn main:app --reload --host 127.0.0.1 --port 8080
```

> main:app：启动main这个模块下的app这个可执行对象
>
> --reload：表示代码修改会自动加载重启服务

> --host：表示绑定的ip地址
>
> --port：表示监听的端口号

- 方式2：执行python脚本文件也是可以启动fastapi项目（不推荐）

```python
# main.py
from fastapi import FastAPI
import uvicorn

app = FastAPI()


@app.get("/")
def root():
    return {"message": "Hello World"}

if __name__ == "__main__":
    uvicorn.run("main:app", host="127.0.0.1", port=8080, reload=True)
```

四个核心知识点

- FastAPI实例对象：是整个项目所有api的入口，有了它才能在它的基础上构建api

```python
app = FastAPI()
```

- 路径操作装饰器：`@app.get("/")`，在这里可以定义接口的url，并指定访问该URL使用的请求方式。比如这个例子中，我们定义了一个URL：`/`，也就是跟路由，访问这个URL时需要使用GET方式。后面简称：路径装饰器

> URL：用来标识某一个具体的网站地址的
>
> GET请求方式：是HTTP协议中标识请求的一种方式，除此之外还有：
> POST\PUT\PATCH\DELETE等，且fastapi也有对应的路径操作装饰器

- 路径操作函数：`def root()`，这个就是我们定义的路径操作函数，或者可以把它叫做为api接口。当别人在浏览器上访问URL`/`的时候，fastapi就会执行这个接口(也就是这个函数)里面的代码。后面简称：路径函数或API接口

- 接口返回值：`return {"message": "Hello World"}`，这个是api接口的返回值，也就是当我们访问URL`/`后在浏览器上看到的响应结果。

> HTTP协议基本内容：当客户端访问(或者说请求)某一个URL后，服务端会响应结果。

1-6 路径参数

需求场景

我们开发一个购物网站，设计不同的URL来表示不同的产品，比如：

- 访问 `/apple`：表示获取苹果的信息
- 访问 `/huawei`：表示获取华为手机的信息
- 访问 `/books`：表示获取所有的图书
- 访问 `/books/2`：表示获取编号id=2的图书
- 访问 `/books/3`：表示获取编号id=3的图书

FastAPI的解决方式

对于上面的需求，我们把这些URL称之为路径参数。**不同的URL表示不同的路径，不同的含义，因此称之为路径参数。**

在FastAPI中可以非常方便的定义和设计这些URL(或者说路径参数)，并且同时支持静态路径参数和动态路径参数

- **静态路径**：写死不变的URL
- **动态路径**：一组格式相同，但包含动态可变的部分。比如：`/books/2`和`/books/3`等等，编号是可变的。

示例：main.py

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/apple")
def get_apple():
    return {"name": "apple", "price": 9.9}


@app.get("/huawei")
def get_huawei():
    return {"name": "huawei", "price": 99}


@app.get("/books/{id}")
def get_huawei(id):
    return {"id": id, "price": "name": f"图书{id}"}
```

动态路径参数的使用！！！

上面的例子中我们定义了动态路径(动态路由)：`/books/{id}`，被{}包起来的参数`id`就是一个动态路径参数，并且可以直接在下面被装饰的路径函数中定义一个同名的形参 `id`，这样的好处：

- 当我们访问URL是 `/books/1`，此时就会把`1`这个路径参数当实参赋值给 `get_huawei(id)`中的形参id
- 当我们访问URL是 `/books/10`，此时就会把`10`这个路径参数当实参赋值给 `get_huawei(id)`中的形参id

路径参数类型转化和校验！！！

上面的例子中我们定义了动态路径：`/books/{id}`， 此时我们在路径函数内拿到的形参`id`的值默认都是 `str` 类型

但是，我们一般使用的id都是 `int` 类型，所以，我们需要有一个类型转化的功能。

- 手动的方式：`id = int(id)`
- Fastapi提供的方式：`def get_huawei(id: int)`，只要给这个参数做一个int的类型提示，就可实现自动转化为`int`
- 如果这个参数不支持强制转化为int，则FastAPI会校验失败，直接报错。

1-7 接口顺序问题

需求场景

我们开发的购物网站，有下面几个URL

- 根据Id获取图书： `/books/{id}`
- 最畅销的图书：`/books/most_popular`

问题来了，当我们在FastAPI中写了下面两个接口，

```
from fastapi import FastAPI


app = FastAPI()


@app.get("/books/{id}")
def get_books_by_id(id: int):
    return {"id": id, "name": f"图书{id}"}


@app.get("/books/most_popular")
def get_most_popular_books():
    return "this is most popular book"
```

启动服务后，浏览器中访问 `/books/most_popular` 就会遇到报错的响应：

```
{"detail":[{"loc":["path","id"],"msg":"value is not a valid
integer","type":"type_error.integer"}]}
```

FastAPI的解决方式

- 问题出现的原因：当请求的URL来了之后，FastAPI会从上往下开始匹配我们定义的接口。
- 所以当我们我们请求的URL是 `/books/most_popular` 的时候，从上往下第一个要匹配的接口是 `/books/{id}`，满足格式要求，此时就会把 most_popular这个值赋值给 id，然后因为我们定义了id这个形参的类型为int，所以FastAPI会帮我们做强制类型转化，即 `int("most_popular")`，此时就报错了。

- 解决这里问题的思路非常简单，就是把 **具体的/写死的/静态的路由放在动态路由前面**，这样在匹配时就会先匹配。

```python
from fastapi import FastAPI


app = FastAPI()


# 静态路由放在动态路由前面
@app.get("/books/most_popular")
def get_most_popular_books():
    return "this is most popular book"


@app.get("/books/{id}")
def get_books_by_id(id: int):
    return {"id": id, "name": f"图书{id}"}
```

补充JSON显示插件

- 在谷歌浏览器中访问URL地址返回的json数据能否按照json格式展现出来。
- 谷歌浏览器安装JOSN扩展插件即可（教程素材会提供插件，大家自己网上下载也可以的哈）

1-8 查询参数

需求场景

我们开发的购物网站，有一个图书列表页面，该页面的图书数据都是从数据库取出来通过图书接口 `/books` 返回前浏览器展示的。

但是数据库中有成千上万本图书，浏览器页面显示的位置有限。通常此时会**使用分页的方式展示数据**。

分页操作时，前端在请求接口`/books`时带上**查询字符串参数**，比如：要第3页的数据，每页数据10条等等。

此时URL可能长这个样子：`/books?page=3&size=10`，page=3表示第三页，size=10表示每页10条数据。

> 补充：HTTP协议规定，URL中问号后面的参数为查询字符串参数，按照键值对的形式存在，可以有多个，用&连接

FastAPI的解决方式

- 直接在路径函数内定义两个普通形参：`page: int, size: int`，
- 这样写好之后FastAPI就会自动帮我们从URL中把`page=3&size=10`解析出来，把3和10赋值给形参page和size
- 需要注意的点：URL中查询参数默认都是字符串格式，所以默认`page`和`size`都是字符串
- 开心的是：当我们在定义形参时加上类型提示`def get_books(page: int, size: int)`，此时FastAPI会帮我们自动做类型转化，并且因此还具备了类型校验的功能。

```
from fastapi import FastAPI


app = FastAPI()


books = [
    {"id": 1, "name": "图书1"},
    {"id": 2, "name": "图书2"},
    {"id": 3, "name": "图书3"},
    {"id": 4, "name": "图书4"},
    {"id": 5, "name": "图书5"},
    {"id": 6, "name": "图书6"},
    {"id": 7, "name": "图书7"},
    {"id": 8, "name": "图书8"},
]


@app.get("/books")
def get_books(page: int, size: int):
    return books[(page - 1) * size:page * size]
```

总结

- 在路径函数中定义的普通形参，就表示要从URL中解析查询字符串参数
- 查询字符串参数默认是字符串类型，使用类型提示可以实现类型转化和类型校验
- 技术点：使用页表切片的方式实现分页逻辑

1-9 OpenAPI文档

OpenAPI文档也可以说是接口文档，用来做接口展示和管理的

FastAPI自带API文档且功能强大。

启动服务后，可以访问两个URL，

- swagger：`/docs`
- redoc：`/redoc`

API文档的功能

- 接口展示，请求格式和响应格式
- 接口调试，可以直接反问后端接口
- 参数校验

设置API文档

```
import typing
from fastapi import FastAPI


app = FastAPI(title="图书项目")      # 设置项目标题


books = [
    {"id": 1, "name": "图书1"},
    {"id": 2, "name": "图书2"},
    {"id": 3, "name": "图书3"},
    {"id": 4, "name": "图书4"},
    {"id": 5, "name": "图书5"},
    {"id": 6, "name": "图书6"},
    {"id": 7, "name": "图书7"},
    {"id": 8, "name": "图书8"},
]


@app.get("/books", tags=["Book"], description="设置描述信息", summary='设置概括信息')
def get_books(page: int, size: int):
    return books[(page - 1) * size:page * size]
```

- tags 设置接口标签
- description 设置接口描述信息
- summary 设置接口概括信息

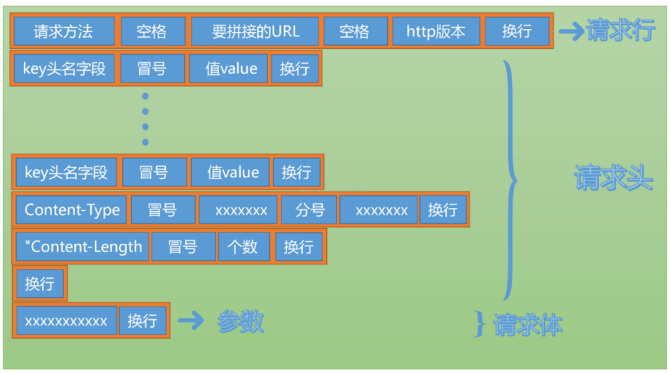
1-10 补充HTTP协议

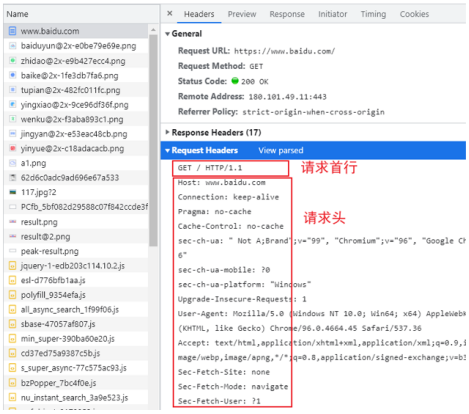
HTTP协议

- HTTP是超文本传输协议，用于客户端和服务端通信
- HTTP协议规定，先有客户端发出请求，再有服务端响应
- HTTP是不保存状态的，即无状态协议。对于服务端来说，每个请求都是新的，不知道你是不是之前请求过。这样做，有好处也有坏处。
- 推荐：图解HTTP（见素材包）
- 免费教程：https://www.51zxw.net/List.aspx?cid=1003

HTTP请求部分

- HTTP请求格式：请求首行、请求头、请求体

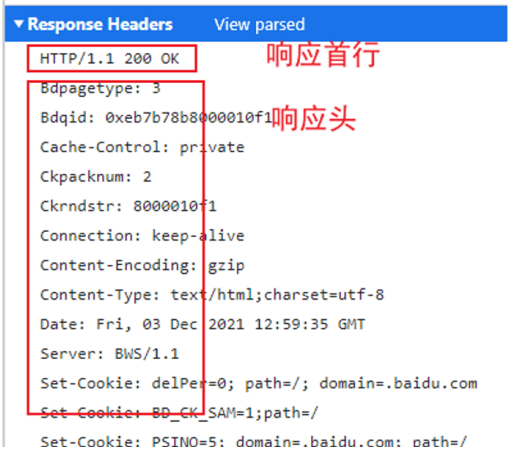




- 请求方式：GET/POST/PUT/PATCH/DELETE等等
- URL：简单理解就是网址。
- 请求头：key-value键值对
- 请求体：像服务端提交数据时，一般把数据放在请求体上

HTTP响应部分

- HTTP响应格式：响应首行、响应头、响应正文



- 状态码：一串表示复杂状态或者描述信息的数字

  > 2xx系列：表示成功
  >
  > 3xx系列：表示重定向
  >
  > 4xx系列：表示客户端请求错误
  >
  > 5xx系列：表示服务端出错

- 响应头：键值对
- 响应数据：响应给客户端的数据