Simple WebServer

尹玉文东

2025.01.10

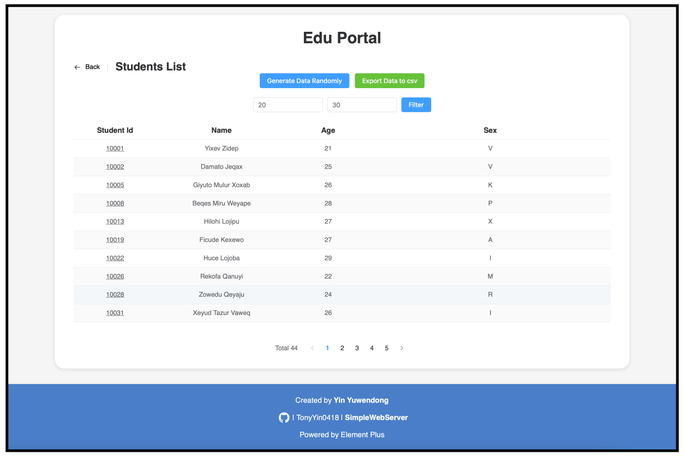
https://github.com/TonyYin0418/SimpleWebServer.git

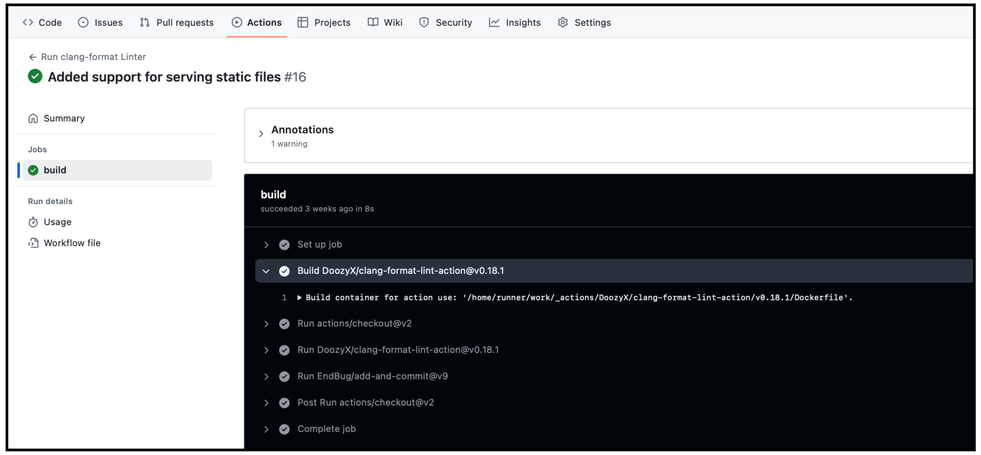
要点A_功能

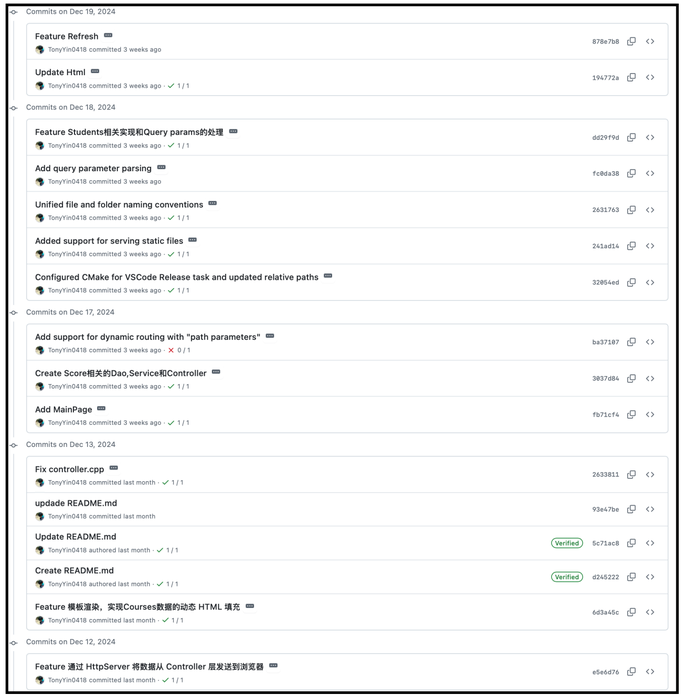
- students列表，courses列表，scores列表
- 学生主页，课程主页
- 按年龄查询学生
- 返回主页
- 长列表翻页，页内排序（未来可扩展为整体排序）
- 随机生成数据的API
- 导出当前数据到csv文件按钮

大作业的复杂性体现在

- C++的Socket库
- 基类、派生类、虚函数
- 较为复杂的 Lambda 表达式，指向函数的指针，tuple元组
- cpp正则表达式 <regex>
- 面向现代cpp的的json库：nlohmann（github.com/nlohmann/json）
- 基于Vue3的前端UI框架：Element Plus (github.com/element-plus/element-plus)
- 版本控制：整个项目用Github进行版本控制，方便提交、回滚、备份
- 使用Github Action配置基于clang-format的代码格式化，在提交时自动格式化代码



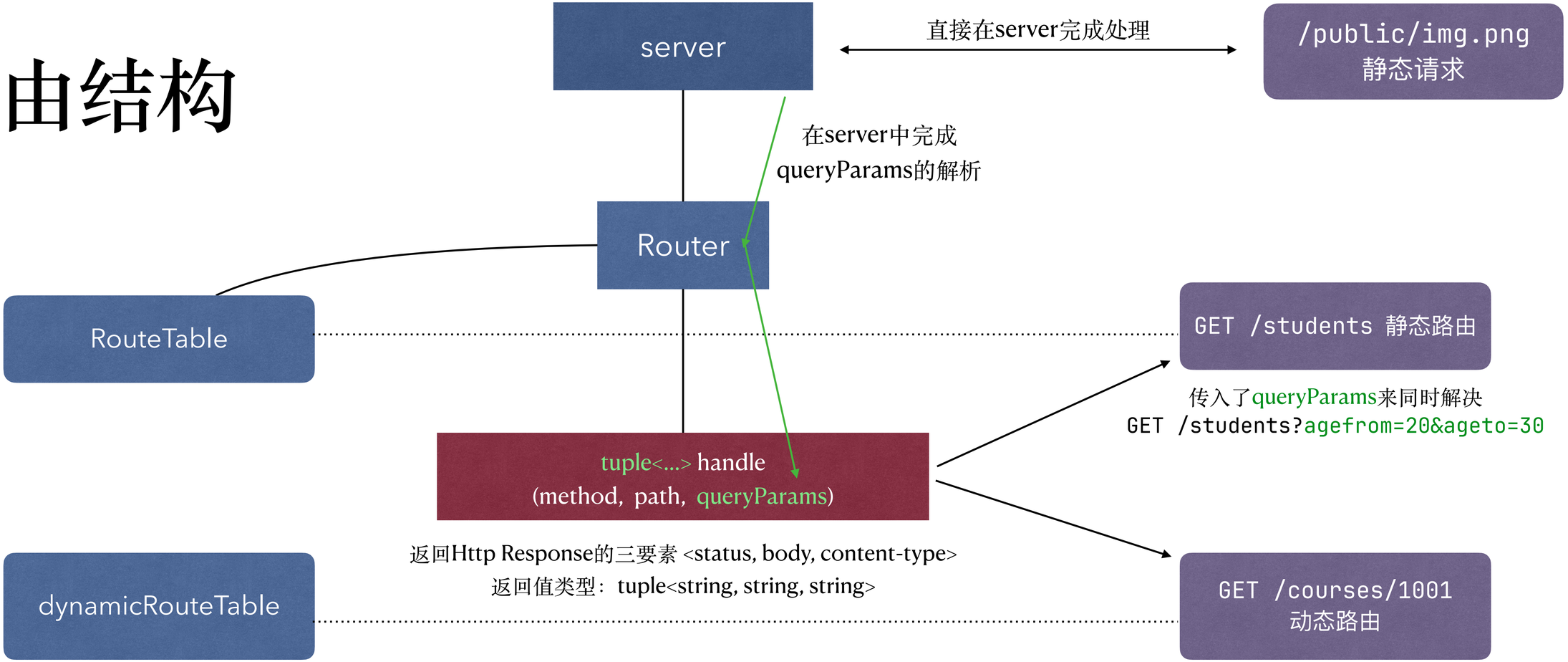




要点B_路由功能概览

- 静态文件访问
  - /public/img.png
- 静态路由
  - /students
  - /students?agefrom=xx&ageto=xx (Query Parameters)
- 动态路由
  - /courses/1003 (Path Parameters)
- 路由表支持扩展POST请求

路由结构




```cpp
using CTRL_FUN = function<tuple<string, string, string>(const smatch&, const map<string, string>&)>;

class Router
{
   public:
    Router();
    virtual void setupRouting() = 0;
    void addRouting(string method, string path, CTRL_FUN handler);
    tuple<string, string, string> handle(string method, string path, map<string, string> queryParams);

   private:
    map<string, map<string, CTRL_FUN> > routeTable;  // method, path, handler
    map<string, vector<pair<regex, CTRL_FUN> > > dynamicRouteTable; // method, path, handler
};
```

动态路由细节

```cpp
//Router.h
//接口有两个可选参数：smatch, map<string, string>

using CTRL_FUN = function<tuple<string, string, string>(const smatch&, const map<string, string>&)>;
```

函数接口有两个可选参数：

smatch(results)存储：pathParams
map<string, string>存储：queryParams

```cpp
//自定义添加动态路由/courses/:courseID
//MyRouter.cpp---SetupRouting():

addRouting("GET", "/courses/:courseID",
    [this](const smatch &match, const map<string, string> &) -> tuple<string, string, string> {
        return scores_controller.getResponse_byCourse(match[1].str());
    });
```

Step A——注册路由

```cpp
//添加到动态路由表
//Router.cpp---addRouting() :
if (path.find(":") != string::npos) {  // 动态路由
    // 下面的regex(":([^/]+)")是匹配模式，第三个参数"([^/]+)"是把通配符替换成的文本内容
    string regexPath = regex_replace(path, regex(":([^/]+)"), "([^/]+)");
    // 将路径中的参数（如 :courseName）替换为文本 ([^/]+) 之后存储。
    dynamicRouteTable[method].push_back({regex(regexPath), handler});  // 存储带有通配符的路径
}
```

Step B——注册路由的实现

使用了cpp<regex>正则表达式库

1. 传入string：/courses/:coursID
2. 变成string：/courses/([^/]+)
3. 转换成regex格式，存进table

```cpp
//处理、转发动态路由
//Router.cpp---handle() :
if (dynamicRouteTable.count(method)) {
    for (auto& item : dynamicRouteTable[method]) {
        // item.first -> 带路径前缀的匹配模式  item.second -> handler
        smatch match;
        if (regex_match(path, match, item.first)) {
            return item.second(match, {});  // queryParams目前是空的，但可以加功能
        }
        // regex_match后，match[0]是整个匹配的字符串(==path),
        // match[1]是第一个括号匹配的字符串，match[2]是第二个括号匹配的字符串
    }
}
```

Step C——路由转发

使用了cpp<regex>正则表达式库

1. 从table中读出/courses/([^/]+)
2. 用这个规则进行正则匹配
3. 根据匹配结果转发

要点C_三层架构

- 底层：数据访问层（DA），封装对数据的增删改查。
  - 面向数据：使用CSV文件
- 中间层：服务层（Service）
- 上层：表示层（Presentation），响应请求，调用服务层的功能，生成响应，controller类属于表示层

架构图

Router

填充html模板

有数据筛选需求的数据流向

/students?agefrom=20&ageto=40

Controller

从service层获得JSON格式数据

加入其他信息

然后填充到HTML模板

Service

vector<Student>转化为JSON

筛选数据

Dao

读取全部数据到vector<Student>

database（student.csv）

nlohmann::json格式:

```
{
    “range” :{“agefrom":"20","ageto":"40"},
    “scores":[
        {“age”:"21","id":"10001","name":"Jack","sex":"M"},
        {"age":"38","id":"10002","name":"Tony","sex":"M"},
        {"age":"23","id":"10004","name":"Mary”,"sex":"F"}
    ]
}
```

```
{
    “scores":[
        {“age”:"21","id":"10001","name":"Jack","sex":"M"},
        {"age":"38","id":"10002","name":"Tony","sex":"M"},
        {"age":"23","id":"10004","name":"Mary”,"sex":"F"}
    ]
}
```

vector<Student>[0]	vector<Student>[1]	vector<Student>[2]
int id=10001 string name=“Jack” int age=21 string sex=“M”	int id=10002 string name=“Tony” int age=38 string sex=“M”	int id=10004 string name=“Mary” int age=23 string sex=“F”

vector<Student>[0]	vector<Student>[1]	vector<Student>[2]	vector<Student>[3]
int id=10001 string name=“Jack” int age=21 string sex=“M”	int id=10002 string name=“Tony” int age=38 string sex=“M”	int id=10003 string name=“ABCD” int age=80 string sex=“M”	int id=10004 string name=“Mary” int age=23 string sex=“F”

csv格式:

```
ID, name, age, sex
10001, Jack, 21, M
10002, Tony, 38, M
10003, ABCD, 80, M
10004, Mary, 23, F
```

Simple WebServer

尹玉文东

2025.01.10

https://github.com/TonyYin0418/SimpleWebServer.git