一个基于 http 服务的环境参数监测系统

福建中学 2024-2025 届 6B 班 22 号 钟岱霖

1. 目录

2. 介紹

2.1. 背景

舒適的環境是閱讀的第一要素，在合適的環境中閱讀可以事半功倍，在不合適的環境中讀書將事倍功半，我發現學校禮堂的燈光屏閃問題嚴重，對於會使眼敏感群衆感到惡心反胃，圖書館的某些自習位在正午可能會比較炎熱，就此，我希望設計一套系統將環境數據收集整理並以表格及顏色標記顯示各種參數，幫助同學們篩選合適的讀書地點，例如顯示人體舒適度指數讓同學可快速瞭解天氣冷熱狀況，酷熱指數作為判斷熱傷害的依據 (台灣衛生福利部國民健康署, 2021)。

表一、酷熱指數

酷熱指數(℃)													
	相對濕度(%)												
溫度(℃)	40	45	50	55	60	65	70	75	80	85	90	95	100
47													
43													
41													
40													
39													
38													
37													
36													
34													
33													
32													
31													
30													
29													
28													
27													

2.2. 目的

製作一套系統用於收集環境數據，包括濕度，溫度及光照程度，並存儲至數據庫，以輸出圖表的方式為用戶分析各環境是否舒適及適合讀書，运行用户借助该系统快速对周围环境做出判断。

3. 分析

3.1. 已有產品分析

- 奥斯恩 OSEN-QX
    - 价格：16500RMB
    - 功能：收集空气温度，风速，风向，雨量，大气压力，光照
    - 适用环境：户外
    - (深圳市奥斯恩净化技术有限公司, 2024)
- 防爆一体化气象仪
    - 功能：风速、风向、空气温度、湿度、大气压力
    - 适用环境：爆炸危险环境
    - (山东竞道光电科技有限公司, 2024)
- 校园气象站
    - 功能：温度、湿度、风向、风速、降雨量、气压
    - 适用环境：户外
    - (山东天合环境科技有限公司, 2024)

根据已有产品分析,可见市面上成熟的环境参数收集设备都与前述需求不符合,多是面向户外,且多数未有面向用户(而非开发者)的界面,且大多价格昂贵,因此,有必要针对这个空缺开发系统

3.2. 舒适环境的要素

體感溫度介於 16 到 20 度时感到舒适,
介於 11 到 15 度感到寒冷,
介於 5 到 10 度体温快速流失，非常危险
(謝玠揚, 2021)

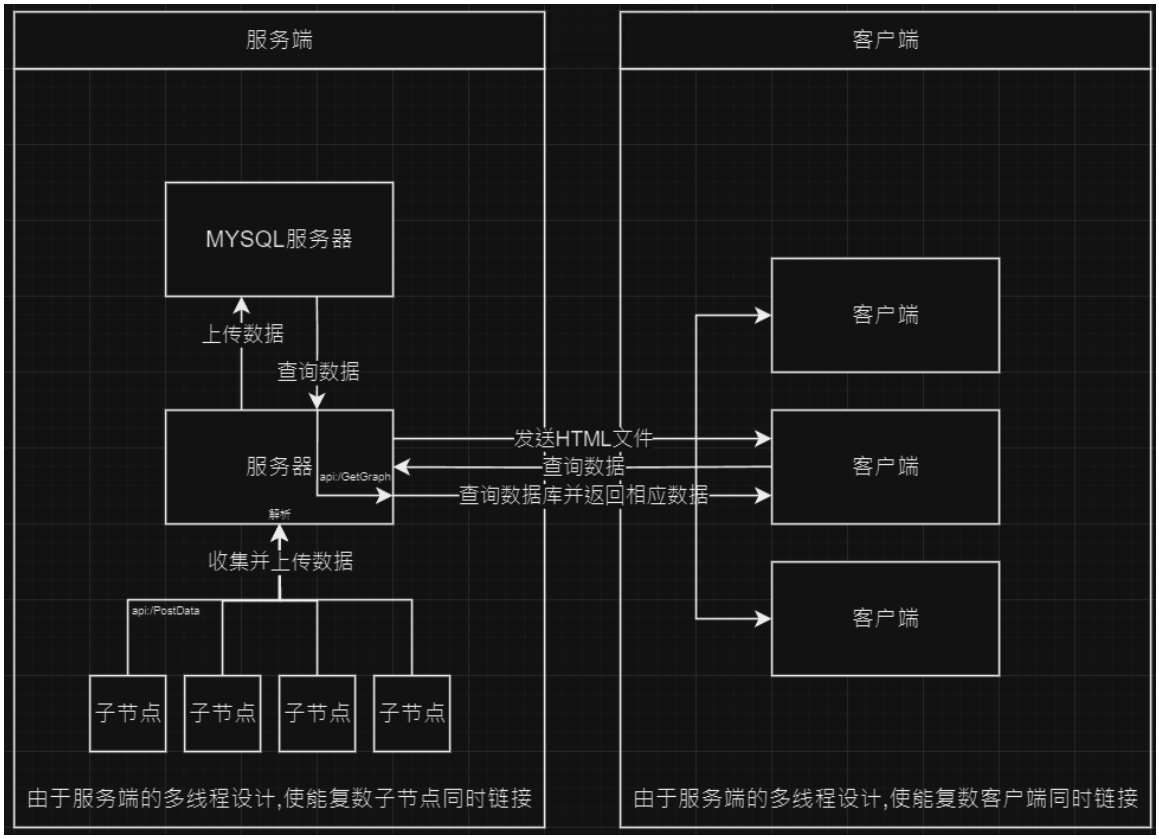
酷热指数共四级，按颜色区分
第一級—綠色：舒适
第二級—黃色：需預防潛在熱浪。
第三級—橘色：高危險族群（老人、小孩、慢性病者）需尽量避免
第四級—紅色：不僅高危險族群，健康成年人亦有生命危險
(台灣衛生福利部國民健康署, 2021)

3.3. 目標用戶

以学校,工厂等地方为例,这些地方有必要对环境参数进行监控,以此保障设施使用者的安全,因此这类环境参数敏感用户会需要一套室内环境监控系统

4. 設計與執行

4.1. 系統總覽



該系統為服務端-客戶端架構，其中：

- 可擴充數量的子节点將連接到網絡，收集數據並上傳到服務器，其中，子节点應當在服務器注冊后提供功能
- 服務器接收已注冊子节点的數據，在 html 頁面將信息總結並展示給用戶查看

其中，子节点預期用具有網絡功能的 esp8266 實現，服務器實現與 windows 平臺上

4.2. 系统设计

4.2.1. 系統功能

子节点:

- 收集數據
- 上傳數據

服務器

- 接受數據
- 總結數據
- 注冊子节点
- 子机管理

用户界面

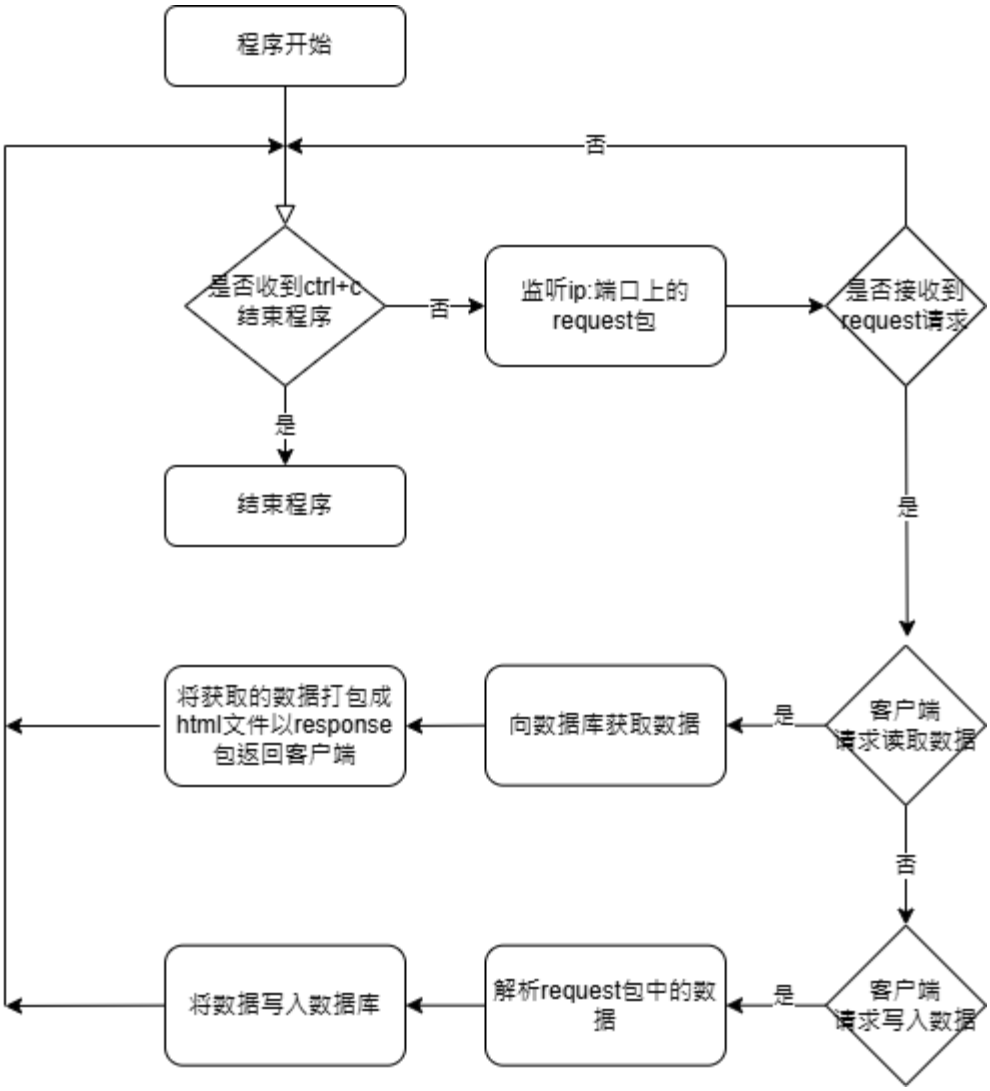
- 展示总体數據
- 展示子节点的环境参数
- 展示总体的环境参数
- 展示图标

4.2.2. api 设计

/ (主页)
发送一个 html 文件以用于客户操作
/PostData
接受子节点数据并解析 json,并通过 MYSQL C API 上传至早前已连接的 MYSQL 服务器
/setup.html
用于允许用户以表格方式设定 config.txt
/postSetup
用于接受来自/setup.htmld 的表格信息
/resetGraph
重置 mysql 数据库里的数据(测试用)
/GetNodeNameList
用于获取节点名字列表
/index.css
用于获取 index.css 用于 index.html
/favicon.ico
用于获取 index.html 的图标(由于未设计,将返回 404)
/status.html
用于获取总体信息,将引用/GetstatusLeft 及/GetstatusRight 的来源
/GetstatusLeft
用于获取总体信息(左部分)
/GetstatusRight
用于获取总体信息(右部分)
/GetGraph
从 MYSQL 服务器请求数据,生成图表 html 并返回到客户端
/NewName
从 MYSQL 服务器请求最新的 NodeName,以用于绑定子节点名字
/BindName
向 MYSQL 服务器注册一个子节点名字,需要输入/BindKey 所获取的秘钥
/GetBindKey
获取秘钥以用于注册子节点名字

4.2.3. 流程设计

本项目为传统服务端-客户端结构,通过 http 协议的 request 包和 response 包沟通



如流程图所示,服务端软件主要功能是写入/读取数据库数据

4.2.4. 初始化

4.2.4.1. 子节点端

子节点的初始化程序并未完成,用户需要向子节点硬编码以下内容

- Wifi 名
- Wifi 密码
- 服务端地址
- 服务端端口

服务端初始化时,管理员需填写必要的信息至 config.txt,信息如下

- 服务器端口
- MYSQL 服务器地址
- MYSQL 服务器端口
- MYSQL 服务器账号
- MYSQL 服务器密码

4.2.4.2. MYSQL 服务器端

MYSQL 服务器初始化时,管理员需导入 sbaSQL.sql 至 MYSQL 服务器内

4.2.5. 维护

4.2.5.1. 服务端

维护服务器端需要拥有基本网络知识,服务器需要开放 80 端口(可自定义)以允许用户访问及服务器需要公网 ip 才可运行(即非 NAT 转换后的 ip)

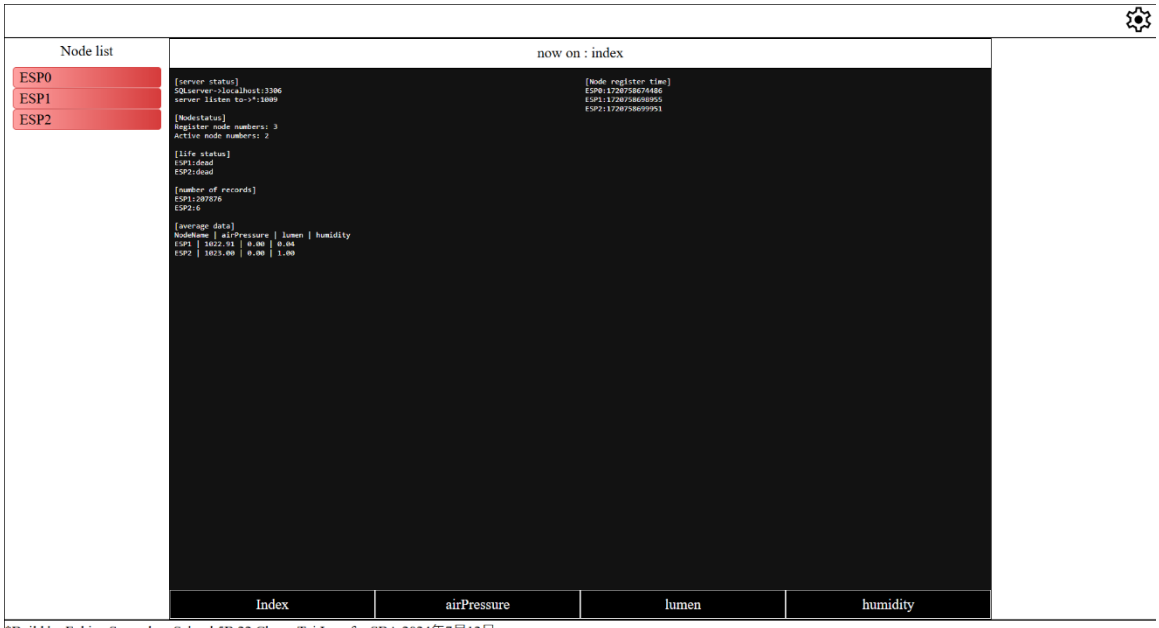
当需要以 nginx 为例的中间件服务以增加网络纵深时,可以修改 nginx.conf 的 path 元素

4.2.5.2. 子节点端,MYSQL 服务器端

一旦初始化完成则无需后续维护

4.2.6. 用戶界面及使用流程

用户界面将~/提供管理员界面,样式如下



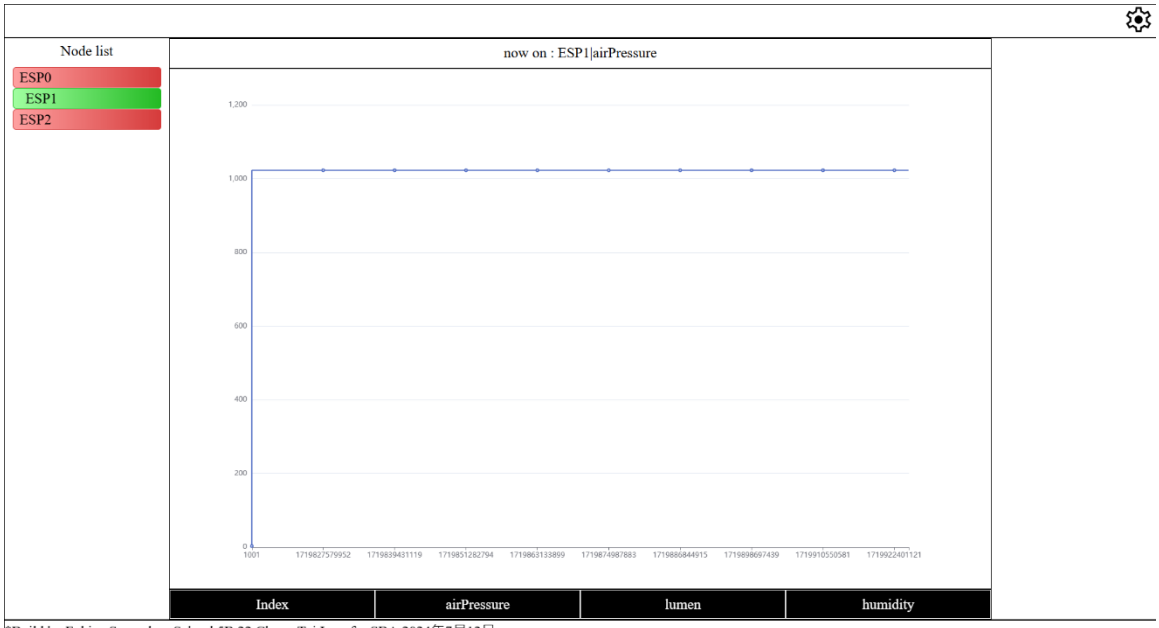
*Build by Fukien Secondary School 5B 22 Chung Toi Lam for SBA,2024年7月12日

其中,选中 airPressure 及 ESP1 将请求 ESP1 的 airPressure 图表,

同理,选中 lumen 及 ESP1 将请求 ESP1 的 lumen 图表,

在 nodelist 选择其子节点将同样改变图表,

以 ESP1|airPressure 为例,将请求 ESP1|airPressure 图表



*Build by Fukien Secondary School 5B 22 Chung Toi Lam for SBA,2024年7月12日

点击右上角的小齿轮将跳转至/setup.html

服务器聆听端口:
SQL服务器域名:
SQL服务器端口:
SQL服务器账号:
SQL服务器密码:
SQL服务器表格:
提交 返回主页

点击提交将表格内容写入 config.txt,并返回相应提示,此处需手动返回主页

```
received
Setupfile:serverPort=asd&SQLserverIP=asd&SQLserverPort=asd&username=asd&password=asd&table=asd
The setup will be enable in next initialization
please manually return to main page
```

4.2.7. 數據處理

服务器共有两个接受 post 方法的 api 接口

分别是/postData 和/postSetup

其中,postData 将允许子节点上传数据,数据将包括:

- 上传时间
- 子节点名字
- 气压
- 温度
- 湿度

(当然,这套系统的参数并不局限于我所设定的这些,是可增可减的,具体将由开发者设计)

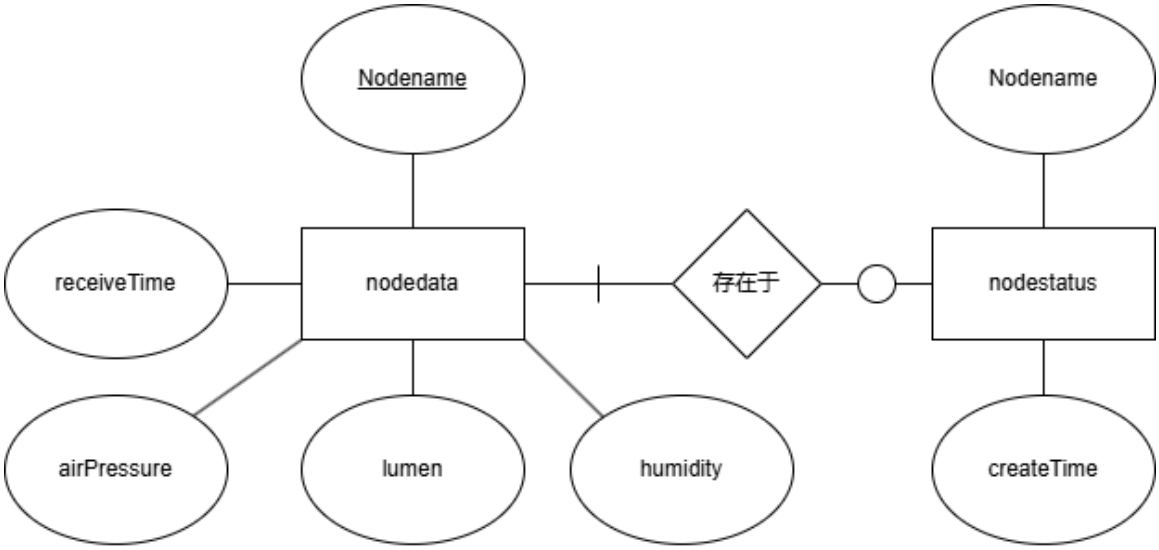
这些数据将通过 http 协议从子节点传送到服务器,经过整理后再通过 MYSQL C API (mysql.h) 传送到 MYSQL 服务器上,最终写入数据库

postSetup 将允许用户以 html 表格的方式修改 config.txt,数据将包括:

- 服务器聆听端口
- SQL 服务器域名
- SQL 服务器端口
- SQL 服务器账号
- SQL 服务器密码
- SQL 服务器表格

这些数据将通过 http 协议从用户的浏览器上传送到服务器,最终写入 config.txt

4.2.8. 数据库设计



该数据库共有两个表,nodedata 和 nodestatus

其中,Nodename 是 nodedata 的主键和 nodestatus 的外键码

4.3. 应用

4.3.1. 算法应用

由于本软件数据繁复，需要不同算法以进行数据处理，现列出涉及算法及用途

- 数据库连接池
    - 用于复用和管理数据库的链接
- 基于 std::map 的 Json 格式数据处理
    - 用于转换 json 格式数据至 map
- 基于迭代器的迭代方法
    - 用于迭代

-

4.3.2. 模组化方法

基于模组化思想，对多次复用的代码进行了函数封装如下

```
std::string getIndexHTML(std ::string Nodelist);
std::string getGraphContent(std::string x, std::string y);
std::string getIndexCSS();
std::string getStatusHTML();
std::string getSetupHTML();
int cti(const char *input);
std::string initCheck(json &configdata);
long long getTimeStamp(timeb &t);
std::string getTimeStampinString(timeb &t);
bool writeSetup(std::string SetupDATA);
void SQLWrite(SQLconnectPool &pool, std::map<std::string, std::string> rec_c, bool &isbusy);
void SQLClear(SQLconnectPool &pool, bool &isbusy);
std::string SQLNewName(SQLconnectPool &pool, bool &isbusy);
void SQLBindName(SQLconnectPool &pool, bool &isbusy, long long createtime, std::string name);
void SQLGetData(SQLconnectPool &pool, std::string DATAname, std::string ESPname, std::string &receiveTimeLists, std::string &dataLists,
bool &isbusy);
std::string GetstatusLeft(SQLconnectPool &pool, bool &isbusy, const char *SQLserverip, const char *SQLserverPort, const char *serverPort);
std::string GetstatusRight(SQLconnectPool &pool, bool &isbusy, const char *SQLserverip, const char *SQLserverPort, const char *serverPort);
std::string SQLGetNodeNameList(SQLconnectPool &pool, bool &isbusy);
json::json_to_map(std::string data);
SQLconnectPool::SQLconnectPool(const char *HOST, const char *SQL_PORT, const char *USER, const char *PASSWORD, const char *TABLE);
SQLconnectPool::~SQLconnectPool();
SQLconnectPool::applySQLConnect(bool &isbusy);
SQLconnectPool::unapplySQLConnect(MYSQL *input);
```

通过函数进行模组化编程,有以下优点:

- 简化重复代码
- 降低心智负担

以 cti 为例,它将 char 转换为 int 类型并返回

```
int cti(const char *input)
{
   std::string _input = input;
   int __input = std::stoi(_input);
   return __input;
}
```

这一功能在程序中可以简化三行的代码,并且允许写在一行表达式内,使得代码更加精简和优雅

4.3.3. 错误处理及健壮性

错误处理基于 c++11 中新增的异常处理机制,简单来说,当错误时,程序会退出,并报错

为了提高健壮性,降低程序异常退出的可能性,我们需要通过 catch 处理异常,或者从源头解决错误的产生

以 std::stoi 为例,这一函数的用途是将 string 转化成 int

这一函数要求 string 内容必须是一个不包含任何字母,标点符号的数字,否则就会报错 std::invalid_argument,使程序强制退出,为了保证健壮性,在使用 stoi 之前对数据进行验证是一个不错的解决方法,当然,更好的解决方案是确保输入来源安全提供一个以不包含任何字母,标点符号的数字为内容的 string

4.3.4. 程序解释

4.3.4.1. 库解释

本程序由 c++写成,共包含 4 个自定义库,2 个外来库和 5 个标准库,如下所示:

外来库:

- #include "httplib.h"
- #include "mysql.h"

标准库:

- #include <string>
- #include <map>
- #include <sys/timeb.h>
- #include <iostream>
- #include <fstream>

自定义库:

- #include "html.h"
- #include "json.h"
- #include "yoursql.h"
- #include "tools.h"

其中:

Cpphttplib(httplib.h)是一个轻量级的 c++库,用于创建和处理 http 请求和响应,他的设计使人可以简单建立一个 http 服务器,而不用从 socket 层重复造轮子,同时也允许我们无需提前直面多线程程序设计的复杂性

MYSQL C API(mysql.h)允许我们的 c++程序与 MYSQL 数据库通信,执行 SELECT,UPDATE,INSERT 一系列常用命令,然而,该库的连接不允许在释放连接之前同时进行两次 query 操作,否则会立即导致崩溃,这一点我们会在后面讲到如何处理

Std::string 提供了标准库的 string 类,该类提供了比传统 char*更加灵活的字符串操作

Std::map 提供了标准库的 map 容器,该 stl 容器允许我们存储键值对,如 Nodename:ESP1,该特性与 json 相当吻合,因此我们选用他来存储来自 html 表格的 json 内容

sys/timeb.h 提供了一系列方法以获取时间戳

iostream 提供了标准输入输出,允许我们在命令行窗口输出日志

fstream 与 iostream 共同提供了读写文件的能力,允许我们从 config.txt 中获取配置信息

自定义库的函数定义及其实现如下:

<html.h>

<html.cpp>

<json.h>
<json.cpp>
<tools.h>
<tools.cpp>
<yoursql.h>
<yoursql.cpp>
“main.cpp”

4.3.4.2. 整体解释

对于这些函数的作用,我们暂且不谈,因为孤例并不能很好得帮助读者理解函数作用,我希望在讲到了其实现的时候再讲解,那么让我们从 main.cpp 开始讲解,我们首先定义了四个变量:

```
timeb t;
bool isbusy = false;
json configJson;
httplib::Server server;
```

这四个变量将贯通整个程序,这也是他们是全局变量的其中一个原因

```
std::string errormessage = initCheck(configJson);
```

我们建立了一个 string 用来装着报错信息(errormessage),这个 errormessage 是由 initCheck 传入的,initCheck 是 tools.h 下定义的函数,其用意是用于检测 config.txt 是否被正确填写,顺带一提,这个库之所以叫 tools.h 是因为我想不到他叫什么好.

```
    std::cout << "[error]errormessage:" << errormessage << std::endl;
    return 0;
```

如果 config.txt 出现问题,则退出程序并打印错误信息

当我们检测到 config.txt 无误的时候,就可以转入主要循环了

```
class json
{
private:
  std::string key, value;
  bool key_done = 0;
public:
  std::map<std::string, std::string> record;
  void json_to_map(std::string data);
};
```

我们的 configJson 是一个 json 类型的类,该自定义类型在 json.h 下定义,其用意是将 json 内容的字符串转做 map 以便使用,上面提到,map 是处理 json 数据的最佳选择,他允许我们建立一对一的对应关系,且支持多种类型,这是 c++的多态特性所导致的,我们举一对例子作例,当我们在 map test<std::string,std::string>内塞入 name:mumu 时可以 name[“mumu”]这种形似数组的方式读取(事实上和数组一点关系都没有,这只是一个[ ]运算符的重载)

```
  std::string errormessage = initCheck(configJson);
```

回到话题上,我们在 std::string initCheck(json &configdata);内传入了一个 json 的引用,用引用来写函数是一个非常方便的做法,这意味着我们无需被”只能返回一个值”的设计给限制住,

直接传入引用并修改该值能允许我们灵活地传出更多参数,甚至有些程序员主张”返回值只用于返回成功与否的信号,其余的返回值一概用引用返回” 这种相当大胆的设想.总而言之,我们的 configJson 传入 initCheck()之后被赋予了 configdata 的别名,在 initCheck 里,我们将读取 config.txt 的值,并通过 configdata.json_to_map()将其反序列化(此时是 char*)后塞进 configdata.record 里(configdata.Record 是一个 std::map<std::string, std::string>类型的容器(当然,你管他叫变量也可以))(json::json_to_map 是一个我实现的简易 json 反序列化算法,仅支持单行和键值对,不过在当前场景下已经完全够用)

```
const char *serverPort = configJson.record["serverPort"].c_str();
const char *SQLserverip = configJson.record["SQLserverip"].c_str();
const char *SQLserverPort = configJson.record["SQLserverPort"].c_str();
const char *username = configJson.record["username"].c_str();
const char *password = configJson.record["password"].c_str();
const char *table = configJson.record["table"].c_str();
```

从 initCheck 返回了 configJson.record 后,我们就可以从中取值赋予变量,这些来着 config.txt 的值随后便会给各种函数连接用

```
std::cout << "*********************************************************" << std::endl;
std::cout << "[Server log]Server begin!" << std::endl;
std::cout << "[Server log]Server listen on *:" << serverPort << std::endl;
std::cout << "*********************************************************" << std::endl;
std::cout << "[TIPS]Please open localhost:" << serverPort << " to view the website" << std::endl;
std::cout << "*********************************************************" << std::endl;
```

30 到 35 行是打印日志,不必多说

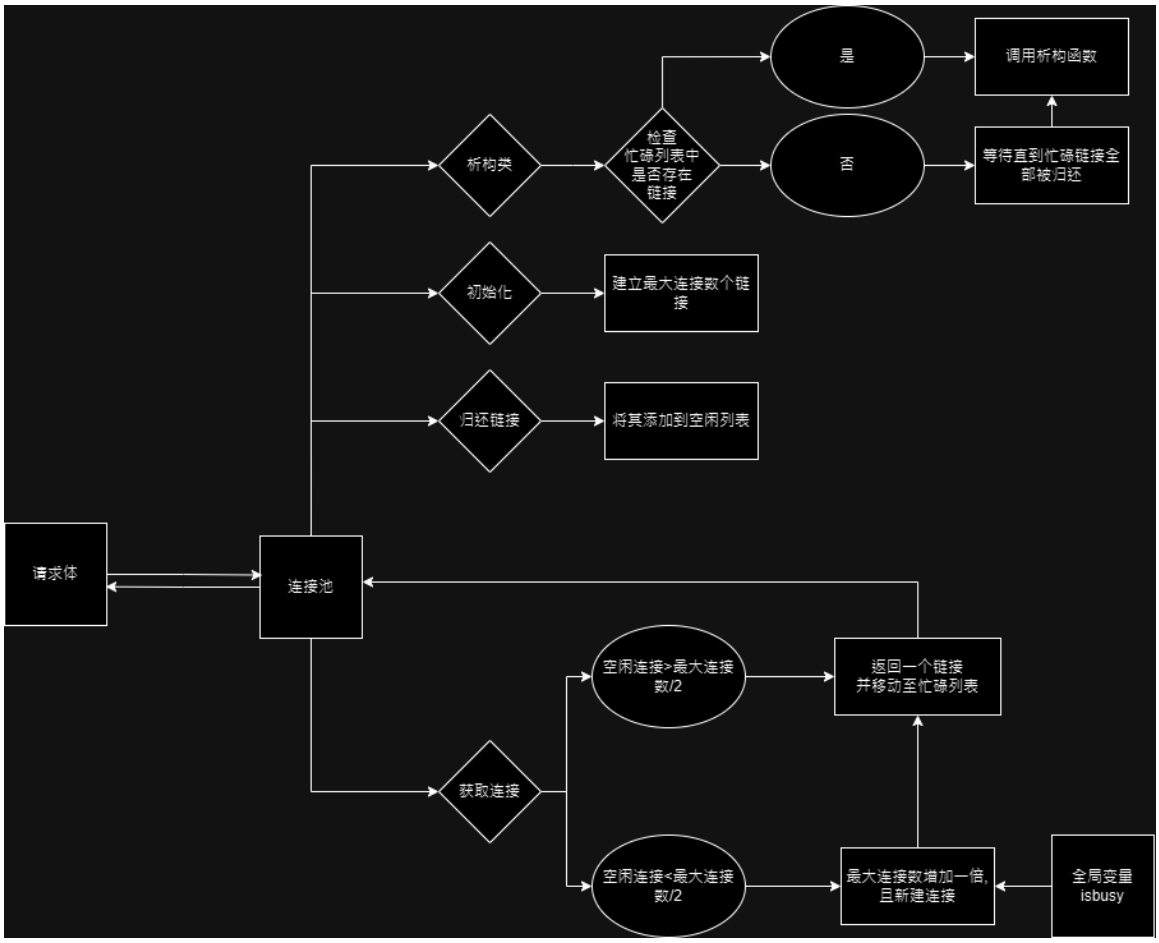
```
SQLconnectPool dataPool(SQLserverip, SQLserverPort, username, password, table);
```

第 36 行,我们定义了一个自定义类型类 SQLconnectPool,如名,这是一个给 MYSQL 用的连接池,这里就不得不提到,由于服务器天然存在的多线程特性,因此我不得不进行多线程编程,其中包括线程危险,资源竞争之类的多线程独有的问题需要被考虑到,且每个链接需要各分配一个 mysql 连接,而为每个线程都建立链接并在线程结束后关闭链接是一个相当消耗资源的操作,因此,我引入了连接池的概念并设计了一个简易的链接池:

连接池的意义
资源复用
由于数据库连接得到复用，避免了频繁的创建、释放连接引起的性能开销，在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片以及数据库临时进程/线程的数量）
更快的系统响应速度
数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了从数据库连接初始化和释放过程的开销，从而缩减了系统整体响应时间。
统一的连接管理，避免数据库连接泄露
在较为完备的数据库连接池实现中，可根据预先的连接占用超时设定，强制收回被占用连接。从而避免了常规数据库连接操作中可能出现的资源泄露。 (wsg_blog, 2023)

在此感谢前辈 (wsg_blog, 2023)提供了具有相当启发性的文章

以下是简易连接池的逻辑:



连接池这个概念的出现可谓是为我扫尽了眼前的迷雾,一举解决了 MYSQL 连接冲突问题和性能问题.最初的设计里,整个软件共用一个 MYSQL 链接,以全局变量 isbusy 调度(即互斥锁),以此要求多个线程同一时间只能处理一个 SQL 查询,但这引发了及其严重的性能问题,访问首页时将消耗足足 3.5 秒钟来等待响应,在采取连接池之后,现在只需要 0.9 秒,简直是丝绸般流畅

```
class SQLconnectPool
{
public:
   SQLconnectPool(const char *HOST, const char *SQL_PORT, const char *USER, const char *PASSWORD, const char *TABLE);
   ~SQLconnectPool();
   MYSQL *applySQLConnect(bool &isbusy);
   void unapplySQLConnect(MYSQL *input);

private:
   std::unordered_set<MYSQL *> SQLconnectBusySet;
   std::stack<MYSQL *> SQLconnectPoolStack;
   int Connectsize = 8;
   const char *HOST;
   const char *USER;
   const char *PASSWORD;
   const char *TABLE;
   const char *SQL_PORT;
```

```
};
```

这个连接池在初始化之后会把其初始化参数(SQLserverip, SQLserverPort, username, password, table)保存到类内的变量里,以此初始化连接.

Std::stack 提供了一个漂亮的堆实现,和我们在 ict 课上学到的相似,push,top,pop 的功能也一应俱全,这将允许我依照顺序安排连接的分配(SQLconnectPoolStack).

当连接被分配时,他将被转移到一个无序的集(SQLconnectBusySet),该无序集由 std::unordered_set 提供,其底层由哈希表实现,因此具有相当高的搜索性能,因此,我们能高速转入转出连接,为什么不用早前声明过的 std::map 呢?这是因为无序集只需要存储 MYSQL 连接这么一段信息,而 std::map 必须储存一个键值对,无序集天生就比 std::map 少了一半的储存需求.

```
MYSQL *applySQLConnect(bool &isbusy);
```

从上图可见,在增加连接数和获取连接的时候会需要一个全局变量引用 isbusy,这是什么情况呢,是这样的,由于我们的服务器是多线程的,因此,同时出现两个线程获取连接是可能出现的,这种时候就会导致同链接重复查询,导致程序崩溃,即使没有出现重复获取连接,也会导致重复增加连接数的情况,这种情况在最大连接数提高之后是尤为明显的,因为建立十个链接约需要 0.1 秒的时间,而当你疯狂点击刷新键,又或者有两个幸运儿在同一时间访问服务器时,就会导致连接数的多次倍增,例如从 32 连接一次性飞跃到 4096 个连接,这会显著拖累服务器的性能,导致用户体验下降.这个问题在业界被称作资源竞争

因此,我们需要一个全局变量来调度,这个方法在业界被称作互斥锁,也就是说,同一时间仅有一个线程被允许执行任务,知道互斥锁被释放,互斥锁用在高频操作上是及其浪费资源的,不过我们获取 MYSQL 连接的操作相对消耗时间较少,因此该性能消耗并不明显,唯有在高并发的时候才会显现出来,为简单考量,我们暂时不要考虑到这么高的高度先

第 39 行到第 124 行做的是相似的事情,我们在最开始声明了一个 httplib::Server 类型的类,该类是由 cpphttplib 提供的,有相当强大的功能,其中之一便是提供 http 响应,我们所提供的 15 个 api 也是由 httplib::Server 类所承担的,在此为了避免缺乏必要知识,让我们回顾下 http 协议可用的方法:

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
3	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。

http 协议共有 8 种请求方法,其中最常用的就是 get 和 post 这两种,httplib::Server 也提供了相对应的方法,以 api ~/ 为例,

```
//----------------------------------------------------------------------------
server.Get("/", [&](const httplib::Request &req, httplib::Response &res)
        { res.set_content(getIndexHTML(SQLGetNodeNameList(dataPool, isbusy)), "text/html"); });
//----------------------------------------------------------------------------
```

我们看似向 server.get 函数输入了看似一大堆参数,但其实事实上东西很少,是这样的,在 c++11 或之后存在一个叫 lambda 表达式的特性,其用意是让我们临时创造一个函数,他将是一次性的,专门为一个只用一次的函数命名是一件极其痛苦的事情,因此 lambda 函数出现了,这将允许我们将一个函数以参数的方法输入到函数里面,这可能听着有些绕,以上面的 server.get 为例,我们向 server.get()传入了两个参数,表示路径的"/"和表示处理方法的 lambda 表达式:

```
[&](const httplib::Request &req, httplib::Response &res)
{ res.set_content(getIndexHTML(SQLGetNodeNameList(dataPool, isbusy)), "text/html"); }
```

让我们再把 lambda 表达式拆开看看

```
[&]
```

上图定义了将以引用的方法捕获全局的变量为 lambda 内部使用

也相当于,在 lambda 的参数栏内引用了全局的每个变量

```
const httplib::Request &req, httplib::Response &res
```

可见,上图是 lambda 的两个参数,

```
res.set_content(getIndexHTML(SQLGetNodeNameList(dataPool, isbusy)), "text/html");
```

上图为 lambda 内部调用的函数,其用意是向 SQLGetNodeNameList()获取子节点列表,并将其返回值作为参数输入 getIndexHTML ()获取主页的 html 文件,再将其返回值作为参数输入 res.set_content(),res 是我们在 lambda 表达式的参数栏内声明的 httplib::Response 类型的类,

他将代表服务器对客户端的响应,也就是说,我们获取并对客户端响应了主页的 html 文件,抽丝剥茧,你会发现 lambda 的核心内容其实是由[](){}这三个运算符组成的,是不是很有趣.

在声明了 15 个 api 之后,我们需要指定 server 所聆听的端口和 ip,聆听端口默认是 0.0.0.0 且不在 config.txt 内可改,因为 0.0.0.0 代表聆听所有来源的 ip,以 0.0.0.0:80 为例,在浏览器中输入 localhost 就可以访问网页了,当然,将其公布到公共网络上需要一些额外的努力.

```
    server.listen("0.0.0.0", cti(serverPort));
```

第 125 行,我们指定了 server 去聆听来自 0.0.0.0,端口为不久前从 config.txt 里读来的 serverPort.我们可以见到中间加了个 cti()函数,这是因为从 config.txt 里读来的是 char*内容,而 char*内容想要转换成 int 内容并没有那么容易,我们知道 std::string 提供了 std::stoi()函数,用于将 std::string 内容转换成 int 内容(其全称其实就是 string to int),而 char*内容可以被 std::string 所转换,因此,如果我们想要优雅地将 char*转换到 int,就需要打包从 char*到 std::string,再到 int 的过程,也就是 cti().

到这里,我们已经讲清楚这个服务器是如何运行起来的了.

不过考虑到我写这么多 api 不能只讲其中一个,我决定挑俩最特别的出来讲讲

来讲讲/postData 和/GetGraph

```
//------------------------------------------------------------------------------
    server.Post("/postData", [&](const httplib::Request &req, httplib::Response &res)
            {
        json reqdata;
        reqdata.json_to_map(req.body);
        reqdata.json_to_map("receiveTime:" + getTimeStampinString(t));
        std::string tempContent;
        tempContent += std::to_string(reqdata.record.size()) + " records had received\n";
        for (auto it = reqdata.record.begin(); it != reqdata.record.end(); it++)
        {
           tempContent += R"(    [)" + it->first + ':' + it->second + "]" + "\n";
        }
        SQLWrite(dataPool, reqdata.record, isbusy);
        reqdata.record.clear();
        res.set_content(tempContent, "text/plain"); });
    //------------------------------------------------------------------------------
```

/postData 的代码是这样的,我们从请求中获取数据(req.body)(这是自动完成的)并把他通过 json_to_map()函数存储到 reqdata.record 里面,接着调用 SQLWrite 函数把获取到的数据存储到 MYSQL 数据库内,让我们再看看 SQLWrite 是怎么定义的:

```
void SQLWrite(SQLconnectPool &pool, std::map<std::string, std::string> rec_c, bool &isbusy)
{
   MYSQL *_SQLconnect = pool.applySQLConnect(isbusy);
   std::string tempSQL = "INSERT INTO sba_test.nodedata (receiveTime,Nodename,airPressure,lumen,humidity) VALUES (" +
rec_c["receiveTime"] + ",\"" + rec_c["Nodename"] + "\",\"" + rec_c["airPressure"] + "\",\"" + rec_c["lumen"] + "\",\"" + rec_c["humidity"] +
"\")";
   mysql_query(_SQLconnect, tempSQL.c_str());
   pool.unapplySQLConnect(_SQLconnect);
}
```

我们一共输入了三个参数,

```
SQLconnectPool &pool, std::map<std::string, std::string> rec_c, bool &isbusy
```

这三个参数分别是:线程池,std::map,互斥锁

其中,线程池会在开始和结束使用

```
MYSQL *_SQLconnect = pool.applySQLConnect(isbusy);
```

我们定义了一个 MYSQL 指针对象,并从线程池里申请了一个连接句柄,将新定义的对象指向该句柄,随后这个句柄(在线程池之内的位置)就会被转移到标记繁忙的无序集

```
pool.unapplySQLConnect(_SQLconnect);
```

这个函数会在线程结束前调用,以释放被占用的 MYSQL 连接句柄,此时,该句柄在线程池内的位置会从标记繁忙的无序集(std::unordered_set)转移到标记空闲的堆(std::stack)

在申请连接句柄之后,我们就可以用他来连接 MYSQL 服务器,我们定义了一个字符串,存储了一条 SQL 指令

```
std::string tempSQL = "INSERT INTO sba_test.nodedata (receiveTime,Nodename,airPressure,lumen,humidity) VALUES (" +
rec_c["receiveTime"] + ",\"" + rec_c["Nodename"] + "\",\"" + rec_c["airPressure"] + "\",\"" + rec_c["lumen"] + "\",\"" + rec_c["humidity"] +
"\")";
```

这指令可能的值是:

```
INSERT INTO sba_test.nodedata (receiveTime,Nodename,airPressure,lumen,humidity) VALUES ( 10000000,"ESP1"," 1000  "," 1000"," 1000"
```

这将在 nodedata 里插入一条记录,用于记 receiveTime,Nodename,airPressure,lumen,humidity 这五条数据

定义了字符串之后,我们再把这字符串转换作 char*以供 mysql_query()所需

```
mysql_query(_SQLconnect, tempSQL.c_str());
```

mysql_query()是用于执行 mysql 命令的,执行完命令之后,我们就可以释放申请过的连接句柄,并且退出函数了

退出函数之后,我们又声明了一个字符串

```
std::string tempContent
```

这个字符串是用来暂时装着要用于响应(response)的数据的

```
tempContent += std::to_string(reqdata.record.size()) + " records had received\n";
```

接下来,我们先往字符串里装下一条提示信息”xx records had received”

Std::tostring()是一个很好用的函数,他允许我们把数字转换成 std::string,与 std::stoi()正正相反,需要注意的是,这两个函数在遇到无法接受的输入类型的时候会直接报错并导致程序退出,因此如果不能确定数据类型,那就有必要使用 try{catch()}和 typeid()来预防未知的错误

```
        for (auto it = reqdata.record.begin(); it != reqdata.record.end(); it++)
        {
           tempContent += R"(    [)" + it->first + ':' + it->second + "]" + "\n";
        }
```

您可能看到这段代码会有些迷糊,auto 是什么,为什么我要给 it 赋值 reqdata.record.begin()

是这样的,auto 也是 c++11 的时候规范的关键字,其目的是让编译器依照初始赋值来给变量一个类型,这个特性也许会在多态或者其他未知类型编程中非常好用,不过我在这里使用他单纯是因为这个变量的变量类型太长了,事实上,it 是一个 std::map<std::__cxx11::string,

std::__cxx11::string>::iterator 类型的变量,这个变量名已经可以占满我屏幕的一半左右了,用 auto 来少打几个字是非常省心的应用.

您可能刚刚看到那个超长的变量类型感到不解, iterator 是什么?为什么不用 int i=0 来遍历?

是这样的,并非每个容器在内存上都是连续的,也并非每个容器都提供[ ]运算符,因此, iterator(迭代器)就有其存在的意义了,通过各个容器自己提供的迭代器,我们可以在不同的容器上用同样的迭代操作,就好像 c++新手时期迭代数组一样轻松惬意.

聪明的你可能已经隐约猜到了, iterator 是基于内存来实现遍历的,也就是说,其本质是一个指针,因此我们需要给他赋值一个指针才能让他跑起来,因此,我们从 reqdata.record 取来 reqdata.record.begin()和 reqdata.record.end()两个指针,就可以轻轻松松,无需了解底层实现地遍历一个容器啦.

```
tempContent += R"(    [)" + it->first + ':' + it->second + "]" + "\n";
```

看明白外面的循环在做什么之后,您可能对这一串看起来像是咒语的玩意感到害怕,无须担心,让我娓娓道来

Std::string 已经对+,+=这两个运算符做了重载,允许我们用这个看起来像是数学公式的玩意来组成一个字符串并赋值 std::string

R"()"的名字是原始字符串,这也是 c++11 中添加的特性,c++11 添加的好用特性还真是多,

> 新的 C++标准可以在代码里嵌入一段原始字符串，该原始字符串不作任何转义，所见即所得，这个特性对于编写代码时要输入多行字符串，或者含引号的字符串提供了巨大方便。原始字符串的定义形式为：**R"xxx(**raw string text**)xxx"**
> 其中，原始字符串必须用括号 **()** 括起来，括号的前后可以加任意其它相同的字符串，所加的字符串会被编译器忽略。 (大飞飞鱼, 2022)

也就是说呢,在 R"()"里添加的那几个空格也会被添加进字符串里

顺带一提,html.cpp 里面就大量用到了 R"()",众所周知,html 里也会大量地用到单引号和双引号,这种时候如果如传统般使用转义符指定要累到手抽筋,而且还不方便修改,但是使用原始字符串就不用思考那么多有的没的,直接把 html 代码复制到 c++里就好.

有些人可能会说:啊我这个 html 的)”被 c++识别作原始字符串的终结段了怎么办啊

看看引文,原始字符串括号的前后是可以添加其他任意相同字符串的,例如 R"a()a"或者 R"aaa()aaa"都是一样的,因此这个特性是极大地利好跨语言用户啊

再讲到 it->first,前面讲到,it 是一个指针,众所周知当我们调用一个类下面的变量时,以 reqdata 为例,要用小数点运算符

```
.
```

那如果我们手上是一个类指针时,要怎么做呢?

有些人可能说我们可以先使用间接寻址运算符把手上的指针提升为变量,也就是说:

```
*it.first
```

虽然也可以,不过,这未免有些不优雅了

```
it->first
```

这样子看着就更优雅一些,一眼看上去就知道这玩意是指针,也不用考虑寻址运算符和取地址运算符运算的困难,毕竟有些人一看到&*这俩符号脑子就转不过来

讲到 it,我们就得回想起 it 是一个指向 reqdata.record 内存地址的迭代器,也就是说 it->first 就是 reqdata.record.first,而 reqdata.record 是一个 std::map,也就是说作为键值对,reqdata.record.first 就是其键,而 reqdata.record.first 就是其值

```
tempContent += R"(    [)" + it->first + ':' + it->second + "]" + "\n";
```

讲到这里,您应该能看懂上面的神秘咒语了,依旧是以 name:mumu 为例,我们这次的可能值是:" [name]:mumu"

让我们再往回倒倒带,回忆下前面到底在讲什么

```
//------------------------------------------------------------------------------
    server.Post("/postData", [&](const httplib::Request &req, httplib::Response &res)
            {
        json reqdata;
        reqdata.json_to_map(req.body);
        reqdata.json_to_map("receiveTime:" + getTimeStampinString(t));
        std::string tempContent;
        tempContent += std::to_string(reqdata.record.size()) + " records had received\n";
        for (auto it = reqdata.record.begin(); it != reqdata.record.end(); it++)
        {
           tempContent += R"(    [)" + it->first + ':' + it->second + "]" + "\n";
        }
        SQLWrite(dataPool, reqdata.record, isbusy);
        reqdata.record.clear();
        res.set_content(tempContent, "text/plain"); });
    //------------------------------------------------------------------------------
```

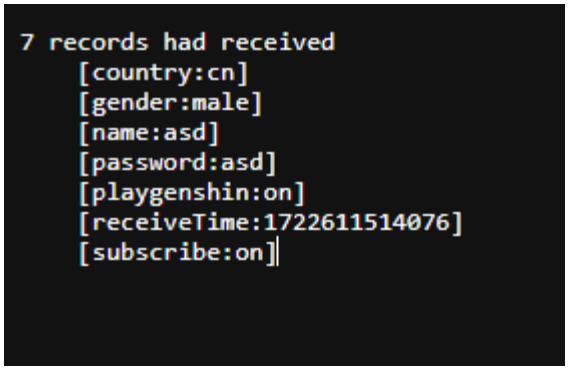
在遍历了一次整个 reqdata.record 之后,我们获得了一个装着很多数据的 tempContent ,接下来就是要调用 res.set_content() ,把这串数据响应到用户的浏览器上了,您可以看到,在 res.set_content()的参数栏里,除了我们刚刚获取的 tempContent ,还有一串 const char*,指向 "text/plain",让我们再复习下 http 的内容类型(content-type):

```
Content-Type（内容类型），一般是指网页中存在的 Content-Type，用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件，这就是经常看到一些 PHP 网页点击的结果却是下载一个文件或一张图片的原因。
Content-Type 标头告诉客户端实际返回的内容的内容类型。

常见的媒体格式类型如下:
text/html： HTML 格式
text/plain： 纯文本格式
text/xml： XML 格式
image/gif： gif 图片格式
image/jpeg： jpg 图片格式
image/png：png 图片格式
以 application 开头的媒体格式类型:
application/xhtml+xml： XHTML 格式
application/xml： XML 数据格式
application/atom+xml： Atom XML 聚合格式
application/json： JSON 数据格式
application/pdf：pdf 格式
application/msword： Word 文档格式
application/octet-stream： 二进制流数据（如常见的文件下载）
```

application/x-www-form-urlencoded： <form encType="">中默认的 encType，form 表单数据被编码为 key/value 格式发送到服务器（表单默认的提交数据的格式）
另外一种常见的媒体格式是上传文件之时使用的：
multipart/form-data： 需要在表单中进行文件上传时，就需要使用该格式 (菜鸟教程, 无日期)

也就是说,我们的响应会让浏览器按照纯文本格式的方法来打开,也就是如下图黑乎乎的页面



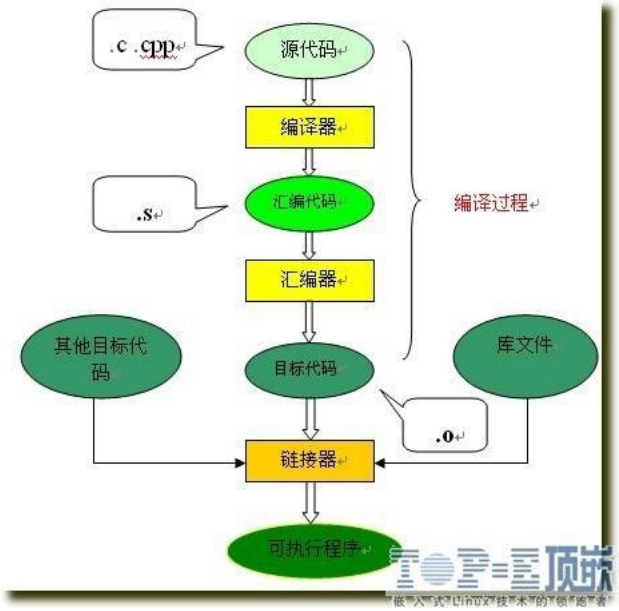
在本程序中,status 也是用这种方法处理的,因为他真的很省事,不用手动画个 ui

关于 const char*,这是一个很有意思的玩意,如您所见,这是一个指针,那么请问 const char*指的是指针不能修改还是数据不能修改呢?答案是数据不能修改,而 char * const 才是"指针不能修改,而数据可以"不过一般也没人用 char * const 这种冷门玩意就是了

关于 cmake:

在工程文件的主目录里可以看到一个 cmakelist.txt,这是向 cmake 程序声明如何编译程序的文件,其优点是能够自定义编译过程,例如,我可以要求他先编译 httplib,然后编译其他的库,最后再将这些库与主程序链接,为什么我要这样子做呢?是因为 httplib 编译起来实在是太过折磨了,长达 50 秒的编译时间在早期调整阶段简直是度日如年,把 httplib 单独编译出来的话就可以避免了编译 httplib 所花费的时间,这是效果显著的.

顺便聊聊编译过程吧,从 c 代码到 exe 可执行文件间大致可分为三步:编译,汇编和连接



(mickole, 2014)

源代码经过编译变成.s 文件

.s 文件经过汇编变成.o 文件

.o 文件再经过连接才最终变成可执行程序

至于 cmake,他本身是没有编译和汇编能力的,这些工作都由 g++之类的编译器承担了,cmake 只是负责编写编译脚本的工具,如果没有 cmake,可能今天我们都要学习怎么用 g++编译软件了

參考資料

mickole. (2014 年 4 月 10 日). C/C++程序编译过程详解 - mickole - 博客园. 检索来源: 博客园: https://www.cnblogs.com/mickole/articles/3659112.html

wsg_blog. (2023 年 3 月 18 日). 3.1.2 连接池. 检索来源: 博客园: https://www.cnblogs.com/go-ahead-wsg/p/17230995.html

菜鸟教程. (无日期). HTTP content-type | 菜鸟教程. 检索来源: 菜鸟教程: https://www.runoob.com/http/http-content-type.html

大飞飞鱼. (2022 年 05 月 02 日). C++ R 原始字符串 R 表示方法，R 是原始字符串-CSDN 博客. 检索来源: CSDN 博客: https://blog.csdn.net/abababab12345/article/details/124539021

山东竞道光电科技有限公司. (2024 年 02 月 05 日). 防爆一体化气象仪-一款安全引爆，平安归来的防爆气象站-仪表网. 检索来源: 仪表网: https://www.ybzhan.cn/solution/show-3252.html

山东天合环境科技有限公司. (2024 年 2 月 27 日). 科普校园气象站系统交流总结会-公司动态-山东天合环境科技有限公司. 检索来源: 智慧城市网: https://www.afzhan.com/st243065/news_433743.html

深圳市奥斯恩净化技术有限公司. (2024). 奥斯恩 OSEN-QX 气象站气象传感器,数据采集器,智能控制器系统. 检索来源: 阿里 1688: https://detail.1688.com/offer/533945053327.html

台灣衛生福利部國民健康署. (2021 年 06 月 10 日). 衛生福利部國民健康署 - 高溫不代表感覺熱！?讓舒適度指數告訴你答案！. 擷取自 衛生福利部國民健康署: https://www.hpa.gov.tw/Pages/Detail.aspx?nodeid=577&pid=10742

謝玠揚. (2021 年 02 月 08 日). 不用管寒流了，先搞清楚「體感溫度」！台灣最常見的溫度是…台大化工博士告訴你：毛衣、發熱衣、羽絨衣這樣搭才對 - 謝玠揚 - 良醫健康網 - 商業周刊（百大良醫）. 检索来源: 良醫健康網: https://health.businessweekly.com.tw/article/ARTL003004286