**C++ Crow**

包含头文件：

#include<crow.h>

最简单的Crow程序：

**crow::SimpleApp** app;//创建应用对象

//定义路由

**CROW\_ROUTE**(app,”/api”)([](){

return “HelloWorld!”;

});

app**.port(8080).multithreaded().run()**;//监听8080端口,多线程

然后在**http://localhost:8080/api**就能看到”HelloWorld!”.

动态路由：在路径中用<...>包含参数，并在lambda表达式处理参数，如：

CROW\_ROUTE(app,”/greet/**<**string**>**)([](std::string name){return “Hi,”+name+”!”;});

可以访问http://localhost:8080/greet/Jack，出现对Jack的问候语。

定义路由时，默认为GET请求，其他请求需要加一个HTTPMethod参数，如**.methods(“...”\_method)**:

CROW\_ROUTE(app,”/post”)**.methods(“POST”\_method)**(...lambda...);//响应POST请求

请求对象：在lambda表达式中传入**crow::request**对象，进行操作：

CROW\_ROUTE(app, "/submit").methods("POST"\_method)

([](**const crow::request&** req){

// ......读取用户发送的数据......

std::string name = req.body;

return "收到：" + name;

});

在crow::request中包含用户的请求，request的成员如下：

req.**method** ->HTTPMethod // 请求类型（GET/POST等）

req.**body** ->std::string // 请求体（POST提交的数据）

req.**url** ->std::string // 请求的url

req.**headers** ->std::unordered\_map // 请求头键值对(大小写不敏感)

req.**get\_header\_value**(“User-Agent”) //获取指定请求头的值(std::string)

响应对象：在lambda表达式中也可以返回**crow::response**对象，如：

**crow::responce** res;

res.body=...;

return res;

在crow::response中包含后端对客户端的响应，成员如下：

res.**code** ->int //HTTP状态码

状态码：200(成功)、404(未找到)、500(服务器错误)

res.**body** ->std::string //响应体

res.**headers** ->std::unordered\_map //响应头键值对(大小写不敏感)

res.**set\_header**(key,value) //设置响应头

res.**add\_header**(key,value) //添加响应头，不覆盖原有值

res.**write**(body) //追加响应体内容

res.**redirect**(url) //重新定向到指定URL

res.**set\_static\_file\_info**(path) //发送静态文件

直接返回response的构造函数：

crow::response(code,std::string) //返回状态码和文本

crow::response(code,json) //返回状态码和JSON

Crow库的JSON文件操作：

CROW\_ROUTE(app,”/data”)([](){

**crow::json::wvalue** x;//创建JSON对象

x[“name”]=”Richard”;

x[“hobbies”]={“Draw”,”Sing”};

return x;//返回JSON对象

});

=====>在http://localhost:8080/data中：

{

“name”:”Richard”,

“hobbies”:[“Draw”,”Sing”]

}

在crow::json子库中的常用操作：

crow::json::**wvalue** x;//创建json结构体

if(!x) //判断结构体是否为空

if(x**.has**(“food”)) //判断键存不存在

x[‘name’] //访问键的值(引用)

x[‘name’]**.s()** //返回一个字符串

x[‘name’].**i()**  //返回int

x[‘info’][“age”]=25;//嵌套JSON, {"info":{"age":25}}

crow::json::wvalue{

{"message","Hello"},

{"status", "success"}

}; //直接构造

解析JSON数据：

auto x = crow::json::**load(**req.body**)**;//解析json字符串为json结构体

解析HTML表单：

对于类似于

<form method="POST">

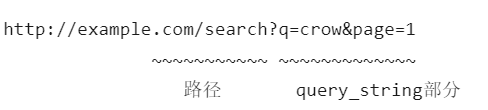
<input name="username" value="张三">

<input name="age" value="20">

</form>

的HTML表单文件(实际发送数据为username=张三&age=20），

或类似于URL的’?’后的参数：



可以使用**crow::query\_string**(req.body)解析数据，用**.get**(key)方法获取返回值指针，如：

auto form = crow::query\_string(req.body);// 假设收到数据：name=李雷&age=25

// 获取字段值

const char\* name = form**.get**("name"); // "李雷"

const char\* age = form.get("age"); // "25"

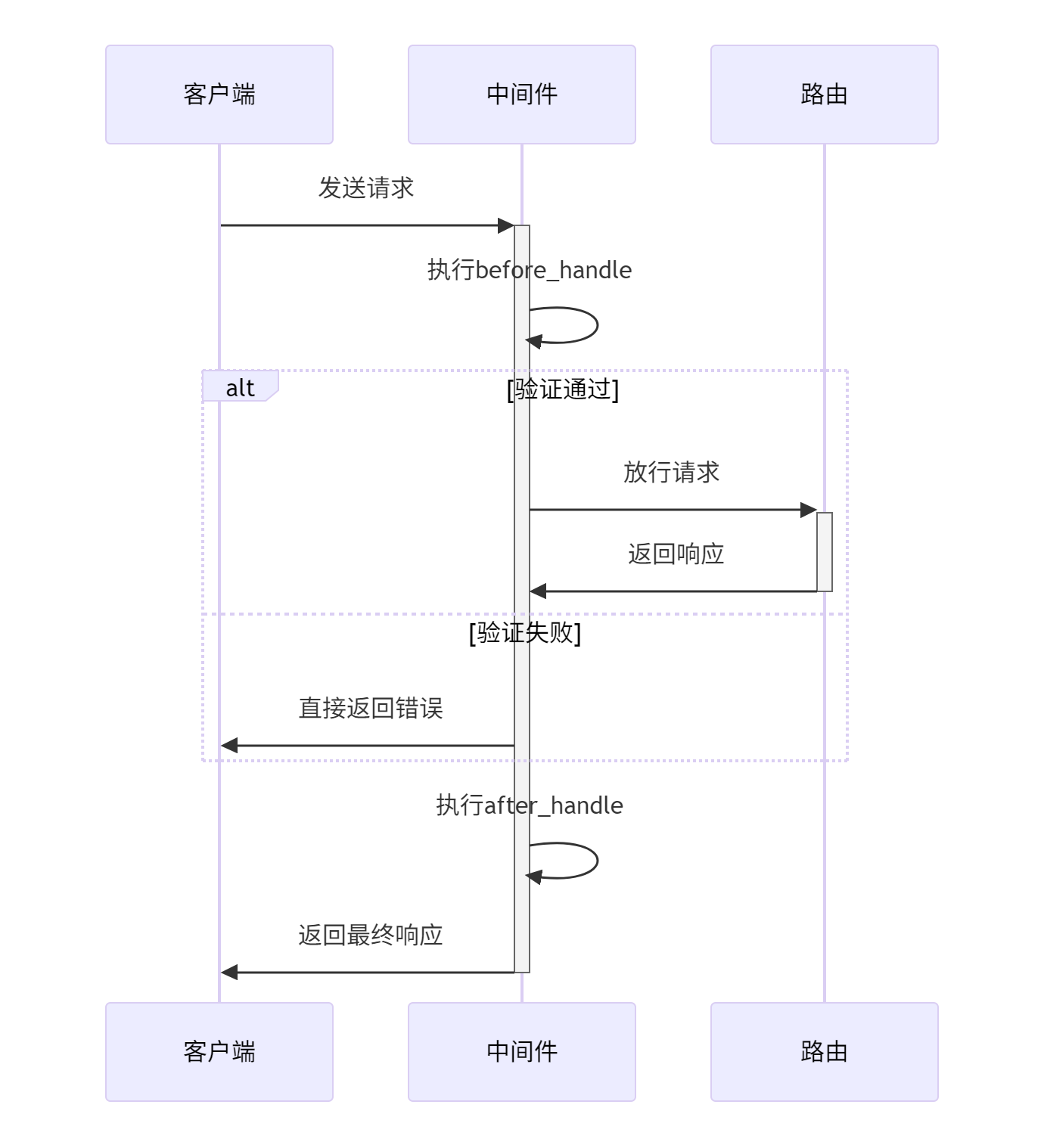
// 检查字段是否存在

if (!form.get("name")) {

return "名字不能为空";

}

中间件：



中间件最常用的用处:解决CORS跨域请求问题.

//CORS跨域请求中间件

struct CORSMiddleWare

{

**struct context {}**; //context结构体,必须要写

**void before\_handle(crow::request& req, crow::response& res, context&)** {

// 对于 OPTIONS 请求，提前返回 200

if (req.method == "OPTIONS"\_method) {

res.code = 200;

res.**set\_header**("Access-Control-Allow-Origin", "\*");

res.set\_header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");

res.set\_header("Access-Control-Allow-Headers", "Content-Type");

**res.end()**;

}

}

**void after\_handle(crow::request& req, crow::response& res, context&)** {

// 给所有响应添加 CORS 头

res.set\_header("Access-Control-Allow-Origin", "\*");

}

};

//然后在main函数中,创建的app就应该是:

int main(){

**crow::App<CORSMiddleWare> app**;//启用中间件

CROW\_ROUTE(app,"/").method("POST")([](const crow::request&req){

//...

});

//...其他路由

app.port(18080).multithreaded().run();

return 0;

}