程序设计训练 Rust 课堂
OJ 大作业入门

游宇凡 2025.9.2

Web 服务器

这次的 OJ 大作业需要写一个 Web 服务器。

- 命令行程序：通过标准输入输出交互
- GUI 程序：通过键盘输入、鼠标点击移动等方式交互
- Web 服务器：通过 HTTP 请求交互

HTTP 请求

HTTP 是一种特殊的文本格式，用来在客户端（例如浏览器或其他工具）和 Web 服务器之间进行通信。客户端向服务器发送 HTTP 请求，服务器处理后回复 HTTP 响应。

例如，客户端请求访问网页 https://example.com，服务器收到请求，返回网页内容。

HTTP 请求

HTTP 是一种特殊的文本格式，用来在客户端（例如浏览器或其他工具）和 Web 服务器之间进行通信。客户端向服务器发送 HTTP 请求，服务器处理后回复 HTTP 响应。

例如，客户端请求访问网页 https://example.com，服务器收到请求，返回网页内容。

```
GET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1256

<!doctype html>
<html>
.....
</html>
```

```
GET /foo HTTP/1.1
Host: example.com
```

```
HTTP/1.1 404 Not Found
Content-Type: text/html
Content-Length: 1256

<!doctype html>
<html>
.....
</html>
```

```
POST / HTTP/1.1
Host: example.com
Content-Length: 4

data
```

```
HTTP/1.1 403 Forbidden
Content-Type: text/html
Content-Length: 361

.....
```

HTTP 请求

```
POST /foo HTTP/1.1
Host: example.com
Content-Length: 4

data
```

```
Method URI Version
HeaderName: HeaderValue
.....
HeaderName: HeaderValue

Body
```

```
HTTP/1.1 403 Forbidden
Content-Type: text/html
Content-Length: 361

.....
```

```
Version StatusCode StatusText
Header Name: Header Value
.....
Header Name: Header Value

Body
```

URI: 路径和参数

简单来说，HTTP 中的 URI 就是网址里位于域名后面的部分，例如 `https://example.com/foo` 中的 `/foo`

URI 中可以包含路径和询问参数，例如 `/foo/2?bar=test&baz=42`

一般来说，路径用于定位访问的资源，参数用于进一步地传递信息。

在大作业框架所使用的 actix-web 中，可以方便地使用 `web::Path`、`web::Query` 等方式获取请求中的路径参数和询问参数。

```rust
#[derive(Debug, Deserialize)]
struct FooQuery { bar: String, baz: Option<i32> }

#[get("/foo/{id}")]
async fn get_foo(id: web::Path<i32>, query: web::Query<FooQuery>) -> impl Responder {
    format!("id: {id}, query: {query:?}")
}
```

HTTP 方法，JSON 请求体

HTTP 支持多种请求方法，最常用的两种是 GET 和 POST，简单来说，它们分别用于读取和发送数据。在本次大作业中还会用到 PUT 和 DELETE 方法，遵循 API 文档实现即可。

POST 请求的 body 可以是任意数据，而在本次大作业中主要以 JSON 格式发送数据。

在 actix-web 中通过 `#[get(path)]` 和 `#[post(path)]` 等来标记一个函数对应的 HTTP 方法和 API 路径，可以用 `web::Json` 读取请求体中的 JSON 数据：

```
POST /foo HTTP/1.1
Host: example.com
Content-Length: 26
Content-Type: application/json

{"bar":[2,3,5],"baz":true}
```

```rust
#[derive(Debug, Deserialize)]
struct FooData { bar: Vec<i32>, baz: bool }

#[post("/foo")]
async fn post_foo(data: web::Json<FooData>) -> impl Responder {
    format!("data: {data:?}")
}
```

HTTP 响应

和请求体类似，响应体也可以是任意数据，而在本次大作业中主要发送 JSON。

```
#[derive(Serialize)]
struct ResponseData { ... }

return web::Json(ResponseData { ... });
```

HTTP 响应有状态码，例如 200 OK、404 Not Found、400 Bad Request 等，本次大作业中规定了出错时需要返回的状态码。

```
return HttpResponse::NotFound().json(Error { ... });
```

```
HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 68

{"code":3,"reason":"ERR_NOT_FOUND","message":"Problem 2 not found."}
```

Actix Web 框架

本次大作业默认使用 Actix Web 框架，前面已经演示过，可以方便地在各个函数中实现各个 API，自动实现参数和响应的解析转换。（如果愿意自学，也可以使用其他框架）

每个 API 路径需要分别实现为一个函数，然后在 `main` 函数中的 `App::new()` 处使用 `.service(function_name)` 注册路由。

更多框架功能和用法可以参考文档 https://actix.rs/docs/ 以及 https://docs.rs/actix-web

HTTP 调试工具

开发 Web 服务器时，一般需要使用工具发送 HTTP 请求来进行调试。

- VS Code REST Client 扩展：在 IDE 内发送测试 HTTP 请求
- Postman、Apifox：图形界面的 API 设计测试工具
- HTTPie、curl：命令行的 HTTP 请求发送工具

Rust 文件操作 & 运行外部程序

执行 OJ 评测任务需要读写输入输出文件，运行编译器以及编译结果等外部程序，可以使用 `std::fs::File` 和 `std::process::Command`。

```
let in_file = File::open("wordle.in")?;
let out_file = File::create("wordle.out")?;
let status = Command::new("wordle")
                     .args(["--random", "-t"])
                     .stdin(Stdio::from(in_file))
                     .stdout(Stdio::from(out_file))
                     .stderr(Stdio::null())
                     .status()?;
```

本次大作业使用异步框架 tokio，也可以使用 `tokio::process::Command`、`tokio::fs::File` 来异步进行操作。

代码模块划分

由于课堂上已经讲解了 Rust 代码的模块组织方式，且本次大作业的代码会更加复杂，涉及多个不同模块，在本次大作业中，代码模块划分的合理性是评分项之一。

- 代码逻辑结构
  - 干同一件事的代码放在同一个模块里
- 控制代码复杂程度（代码过长考虑拆分）
  - 一个模块内的代码不要太长
  - 单个函数内的代码不要太长

示例（仅供参考，鼓励自行设计）：

- 命令行参数解析
- 配置文件解析
- 评测任务执行
- 错误处理（自定义错误类型）
- API 路由
  - 评测任务路由
  - 用户路由
  - 比赛路由
- 数据库（如果实现了提高要求）
  - 评测任务表
  - 用户表
  - 比赛表
- ……