基于 Gin 的 HTTP 服务端实现

Kascas

2021 年 3 月 8 日

目录

1 服务端结构

1.1 C/S 模型

客户端与服务器通信采用 C/S 架构。为了使 Web 业务与数据存取分离，采用三层结构的 C/S 模型，即服务端由**云服务器**与**云数据库**组成。云数据库只能被云服务器访问，而云服务器服务器可以被任意用户访问。客户端与服务端沟通的大致流程为：云服务器处理客户端发送的请求，向云数据库的请求数据并处理，最后将请求的结果返还给客户端。

1.2 环境配置

云服务器 云服务器具有公网 IP、私网 IP、防火墙等若干重要属性。**公网 IP** 即为云服务器在公网中的地址，外部设备通过该地址向云服务器发送请求；**私网 IP** 即云服务器网卡所绑定的地址，服务程序应监听私网 IP 而非公网 IP；**防火墙**定义了若干访问规则，通常开放 80(http)、443(https)、22(ssh) 端口，也可根据需要自定义开放端口。对于云服务器来说，如果需要提供相关服务，则应当将程序监听的地址与私网 IP 和端口相匹配，且在防火墙设置中设定相应的访问规则（例如开放指定端口）。

云数据库 Mysql 默认使用 3306 端口。如果云服务器与云数据库不在同一私网下，则需要获取公网 IP 从而使得云服务器能够与云数据库建立连接，然后在防火墙设置安全分组，指定仅云服务器可访问（IP 限制）。

2 程序结构

2.1 配置文件 (.conf)

网络配置 (net.json) 网络配置较为简单，仅包含 Host 和 Port 两个字段.

```
{
    "Host": "192.179.116.130",
    "Port": 443
}
```

数据库配置 (db.json) 数据库配置较为复杂，主要包括 SQL（数据库类型）、IP（数据库网络地址）、Port（数据库端口）、User（数据库用户）、Passwd（数据库用户对应的密码）、DBName（数据库名称）、MaxOpenConns、MaxIdleConns。

```
{
    "SQL": "mysql",
    "IP": "192.168.116.129",
    "Port": 3306,
    "User": "root",
    "Passwd": "qwdqwd",
    "DBName": "demo",
    "MaxOpenConns": 10,
    "MaxIdleConns": 10
}
```

2.2 配置初始化 (confs)

TLS 证书生成 (cert.go) 为了支持 https，需要具备相应的证书。为了方便起见，使用 OpenSSL 生成 key.pem 和 cert.pem。

```
func CertInit()
```

数据库配置 (db.go) 定义结构体 DBConf，存储数据库相关配置。其内容与 db.json 一致。设置一个可导出的全局变量 DBInfo。

```
type DBConf struct {
    IP      string
    Port    uint32
    User    string
    Passwd string
    DBName string
    MaxOpenConns int
    MaxIdleConns int
}
var DBInfo *DBConf
```

数据库设置默认值初始化与 db.json 信息覆写。

```
func DBInit()
func (s *DBConf) Reload()
```

数据库连接 (dbConn.go) 根据初始化后 DBInfo 的信息与数据库建立连接。

```
func DBConn()
```

网络初始化 (net.go) 定义结构体 NetConf，存储网络相关配置，其内容与 net.json 一致。设置一个可导出的全局变量 NetInfo。

```
type NetConf struct {
    Host string
    Port uint32
}
var NetInfo *NetConf
```

网络默认值初始化与 net.json 信息覆写。

```
func NetInit()
func (n *NetConf) Reload()
```

工作空间初始化 (cwd.go) 获取并进入可执行文件所在的路径。

```
func WdInit()
```

2.3 中间件 (middleware)

JWT 认证 (myjwt/jwt.go) 定义若干 Token 认证异常。

```
var (
    TokenExpired     = errors.New("Token已过期")
    TokenNotValidYet = errors.New("Token认证错误")
    TokenMalformed   = errors.New("Token格式错误")
    TokenInvalid     = errors.New("Token不合法")
    SignKey          = "httpserver"
)
```

结构体 CustomClaims 存储 Token 的载荷，JWT 存储签名结构。

```
type CustomClaims struct {
    User string `json:"user"`
    jwt.StandardClaims
}
type JWT struct {
    SigningKey []byte
}
```

关于 Signkey 的 Get 和 Set。

```
func GetSignKey() string
func SetSignKey(key string) string
```

解析、生成、更新 Token。

```
func (j *JWT) ParseToken(tokenString string) (*CustomClaims, error)
func (j *JWT) CreateToken(claims CustomClaims) (string, error)
func (j *JWT) RefreshToken(tokenString string) (string, error)
```

中间件入口函数。

```
func JWTAuth() gin.HandlerFunc
```

TLS(tls/tls.go) 中间件入口函数。

```
func TLS() gin.HandlerFunc
```

2.4 路由处理 (handler)

注册 (signup.go) 提取请求中的 user-passwd，如果符合要求则存入数据库。

```
func SignUp(c *gin.Context)
```

登录 (signin.go) 定义结构体 userInfo，存储用户信息（用户名-密码）。

```
type userInfo struct {
    User   string `json:"user"`
    Passwd string `json:"passwd"`
}
```

定义结构体 auth，继承 userInfo，增加 Token 字段。

```
type auth struct {
    Token string
    userInfo
}
```

根据数据库的 authtable 对 user-passwd 进行验证。

```
func check(user string, passwd string) error
```

生成一个 Token。

```
func generateToken(c *gin.Context, u userInfo)
```

登录的入口函数。如果登录成功，则返回一个 Token。

```
func SignIn(c *gin.Context)
```

上传 (upload.go) 提取 Body 中的数据，并存入数据库的 datatable 表。

```
func Upload(c *gin.Context)
```

下载 (download.go) 向用户提供文件。

```
func Download(c *gin.Context)
```

3 快速使用

3.1 配置文件

在运行程序前，需要按当前网络环境修改./.conf/文件夹下的 net.json 和 db.json 文件。需注意，net.json 下的 IP 字段应绑定私网 IP 而非公网 IP。

3.2 nginx 设置

```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

worker_processes auto;
pid /run/nginx.pid;
```

```
events {
    worker_connections 1024;
}

http {
    sendfile            on;
    keepalive_timeout   65;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /root/nginx-access.log  main;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    #Setting for a TLS enabled server.
    server {
        listen       443 ssl;
        server_name  dcyztech.xyz;

        error_log /root/https-nginx-error.log;

        ssl_certificate "/root/httpserver/cert.pem";
        ssl_certificate_key "/root/httpserver/key.pem";
        location /{
            proxy_pass      https://172.24.9.190:4433;
        }
    }

}
```

3.3 路由访问

当前程序中共设置了 4 个路由：/signup, /signin, /auth/upload, /auth/download。

/signup [POST] /signup 为注册功能，需要使用 POST 请求。在请求的 Body 中，需要提供 JSON 格式的数据，键为 user 和 passwd。示例如下：

```
{
    "user":"abc",
    "passwd":"123",
}
```

/signin [GET] /signin 为登录功能，需要使用 GET 请求。在请求的 Body 中，需要提供 JSON 格式的数据，格式与/signup 一致。在收到的 Response 中，Token 数据为 JWT 认证的令牌，在 upload 和 download 路由中有重要作用，需要保存。Response 内容示例如下：

```
{
    "auth": {
        "Token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
            eyJ1c2VyIjoiYWJjIiwiZXhwIjoxNjE0NzgxNjYzLCJpc3MiOiJodHRwc2Vy
dmVyIiwibmJmIjoxNjE0Nzc3MDYzfQ.gbYkPVfYUNwptHYqIrj4oX8CWW2067hTRuYI0M50DYs",
        "user": "abc",
        "passwd": "123"
    },
    "msg": "登录成功",
    "status": 0
}
```

/auth/upload [POST] /auth/upload 为上传功能，需要使用 POST 请求。在请求 Headers 的 Authorization 字段内，需要填写 signin 阶段获取的 Token (形式为 Authorization: Bearer <Token>)：

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJ1c2VyIjoiYWJjIiwiZXhwIjoxNjE0NzgxNjYzLCJpc3MiOiJodHRwc2VydmVyIiwibmJmIjox
NjE0Nzc3MDYzfQ.gbYkPVfYUNwptHYqIrj4oX8CWW2067hTRuYI0M50DYs
```

/auth/download [GET] /auth/download 为下载功能，需要使用 GET 请求。请求的 Headers 设置与 upload 相同。