

API 身份验证

本节核心内容

- 介绍 API 身份验证的常用机制
- 介绍如何进行 API 身份验证

本小节源码下载路径：[demo09](https://github.com/lexkong/apiserver_demos/tree/master)
(https://github.com/lexkong/apiserver_demos/tree/master)

可先下载源码到本地，结合源码理解后续内容，边学边练。

本小节的代码是基于 [demo08](https://github.com/lexkong/apiserver_demos/tree/master)
(https://github.com/lexkong/apiserver_demos/tree/master)
来开发的。

API 身份验证

在典型业务场景中，为了区分用户和安全保密，必须对 API 请求进行鉴权，

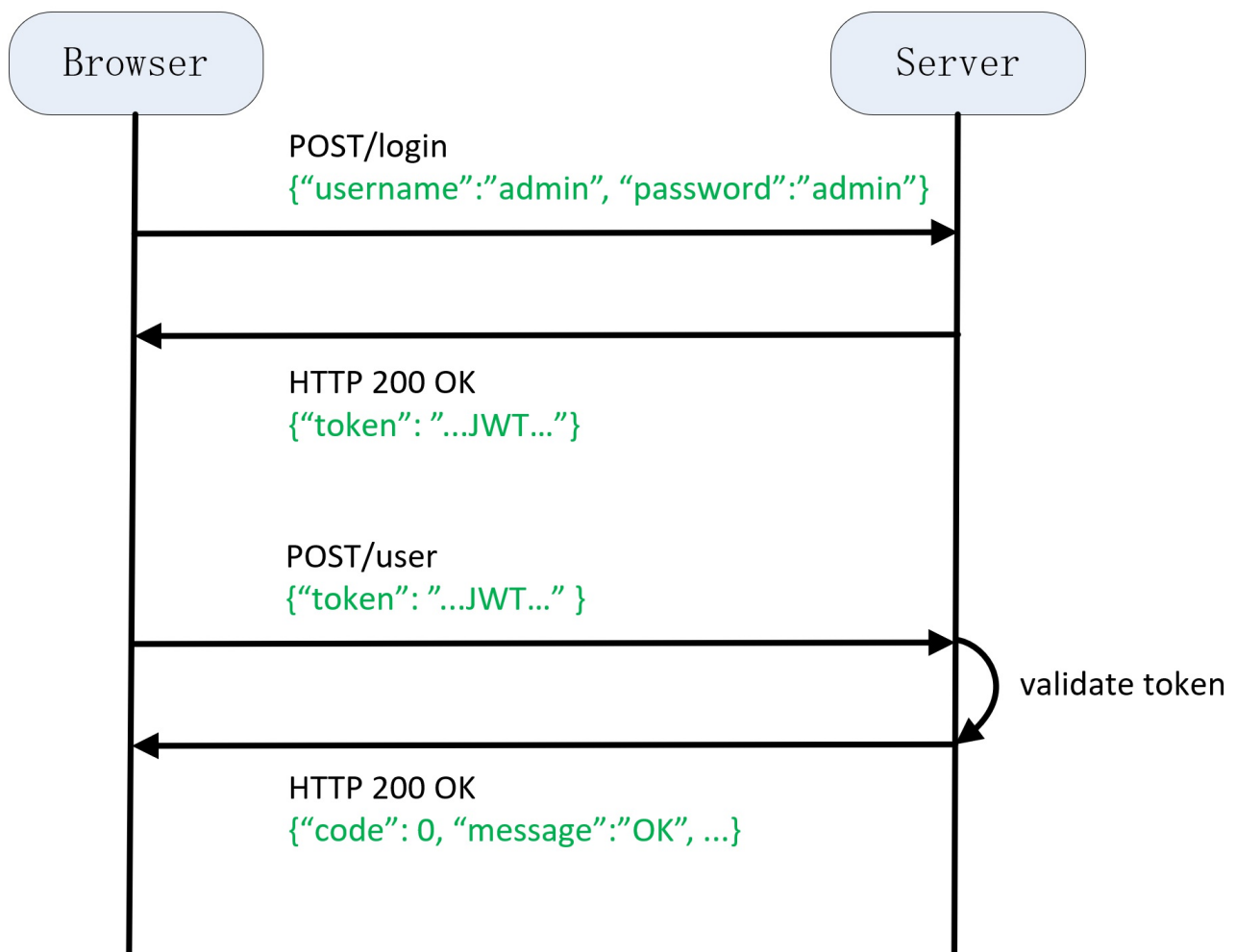
但是不能要求每一个请求都进行登录操作。合理做法是，在第一次登录之后产生一个有一定有效期的 token，并将其存储于浏览器的 Cookie 或 LocalStorage 之中，之后的请求都携带该 token，请求到达服务器端后，服务器端用该 token 对请求进行鉴权。在第一次登录之后，服务器会将这个 token 用文件、数据库或缓存服务器等方法存下来，用于之后请求中的比对。或者，更简单的方法是，直接用密钥对用户信息和时间戳进行签名对称加密，这样就可以省下额外的存储，也可以减少每一次请求时对数据库的查询压力。这种方式，

在业界已经有一种标准的实现方式，该方式被称为 JSON Web Token (JWT, 音同 jot, 详见 [JWT RFC 7519](https://tools.ietf.org/html/rfc7519) (<https://tools.ietf.org/html/rfc7519>)) 。

token 的意思是“令牌”，里面包含了用于认证的信息。这里的 token 是指 JSON Web Token (JWT) 。

JWT 简介

JWT 认证流程



1. 客户端使用用户名和密码请求登录
2. 服务端收到请求后会去验证用户名和密码，如果用户名和密码

跟数据库记录不一致则验证失败，如果一致则验证通过，服务端会签发一个 Token 返回给客户端

3. 客户端收到请求后会将 Token 缓存起来，比如放在浏览器 Cookie 中或者本地存储中，之后每次请求都会携带该 Token
4. 服务端收到请求后会验证请求中携带的 Token，验证通过则进行业务逻辑处理并成功返回数据

在 JWT 中，Token 有三部分组成，中间用 . 隔开，并使用 Base64 编码：

- header
- payload
- signature

如下是 JWT 中的一个 Token 示例：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE1MjgwMTY5MjIsImkiOiJowLCJuYmYiOiE1MjgwMTY5MjIsInVzZXJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-9v43NzWBN7HXsSLfebW92DKd1JQ
```

header 介绍

JWT Token 的 header 中，包含两部分信息：

1. Token 的类型
2. Token 所使用的加密算法

例如：

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

该例说明 Token 类型是 JWT，加密算法是 HS256（alg 算法可以有多种）。

Payload 载荷介绍

Payload 中携带 Token 的具体内容，里面有一些标准的字段，当然你也可以添加额外的字段，来表达更丰富的信息，可以用这些信息来做更丰富的处理，比如记录请求用户名，标准字段有：

- iss: JWT Token 的签发者
- sub: 主题
- exp: JWT Token 过期时间
- aud: 接收 JWT Token 的一方
- iat: JWT Token 签发时间
- nbf: JWT Token 生效时间
- jti: JWT Token ID

本例中的 payload 内容为：

```
{
  "id": 2,
  "username": "kong",
  "nbf": 1527931805,
  "iat": 1527931805
}
```

Signature 签名介绍

Signature 是 Token 的签名部分，通过如下方式生成：

1. 用 Base64 对 header.payload 进行编码
2. 用 Secret 对编码后的内容进行加密，加密后的内容即为 Signature

Secret 相当于一个密码，存储在服务端，一般通过配置文件来配置 Secret 的值，本例中是配置在 `conf/config.yaml` 配置文件中：

```
runmode: debug                                # 开发模式, debug, release, test
addr: :8080                                   # HTTP绑定端口
name: apiserver                               # API Server的名字
url: http://127.0.0.1:8080                   # pingServer函数请求的API服务器的ip:port
max_ping_count: 10                           # pingServer函数try的次数
jwt_secret: Rtg8BPKNEf2mB4mgvKONGPZZQSaJWNLijxR42qRgq0iBb5
log:
  writers: file,stdout
  logger_level: DEBUG
  logger_file: log/apiserver.log
  log_format_text: false
  rollingPolicy: size
  log_rotate_date: 1
  log_rotate_size: 1
  log_backup_count: 7
db:
```

最后生成的 Token 像这样：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE1MjgwMTY5MjIsImlkIjowLCJuYmYiOiE1MjgwMTY5MjIsInVzZSJ0eSwiaWF0eSI6MTYxOTQyODAwfQ.LjxrK9DuAwAzUD8-9v43NzWBN7HXsSLfebw92DKd1JQ
```

签名后服务端会返回生成的 Token，客户端下次请求会携带该 Token，服务端收到 Token 后会解析出 header.payload，然后用相同的加密算法和密码对 header.payload 再进行一次加密，并对比加密后的 Token 和收到的 Token 是否相同，如果相同则验证通过，不相同则返回 HTTP 401 Unauthorized 的错误。

详细的 JWT 介绍参考 [JWT - 基于 Token 的身份验证](https://blog.csdn.net/qg_28098067/article/details/520)
(https://blog.csdn.net/qg_28098067/article/details/520)

如何进行 API 身份验证

API 身份认证包括两步:

1. 签发 token
2. API 添加认证 middleware

签发 token

首先要实现登录接口。在登录接口中采用明文校验用户名密码的方式，登录成功之后再产生 token。在 `router/router.go` 文件中添加登录入口：

```
// api for authentication functionalities  
g.POST("/login", user.Login)
```

在 `handler/user/login.go` (详见 [demo09/handler/user/login.go](https://github.com/lexkong/apiserver_demos/blob/master/handler/user/login.go) (https://github.com/lexkong/apiserver_demos/blob/master/handler/user/login.go)) 中添加 login 的具体实现：

1. 解析用户名和密码
2. 通过 `auth.Compare()` 对比密码是否是数据库保存的密码，如果不是，返回 `errno.ErrPasswordIncorrect` 错误
3. 如果相同，授权通过，通过 `token.Sign()` 签发 token 并返回

`auth.Compare()` 的实现详见 [demo09/pkg/auth/auth.go](https://github.com/lexkong/apiserver_demos/blob/master/pkg/auth/auth.go) (https://github.com/lexkong/apiserver_demos/blob/master/pkg/auth/auth.go)

`token.Sign()` 的实现详见 [demo09/pkg/token/token.go](https://github.com/lexkong/apiserver_demos/blob/master/pkg/token/token.go) (https://github.com/lexkong/apiserver_demos/blob/master/pkg/token/token.go)

API 添加认证 middleware

在 router/router.go 中对 user handler 添加授权 middleware:

```
// api for authentication functionalities
g.POST("/login", user.Login)

// The user handlers, requiring authentication
u := g.Group("/v1/user")
u.Use(middleware.AuthMiddleware())
{
    u.POST("", user.Create)
    u.DELETE("/:id", user.Delete)
    u.PUT("/:id", user.Update)
    u.GET("", user.List)
    u.GET("/:username", user.Get)
}
```

通过该 middleware, 所有对 /v1/user 路径的请求, 都会经过 middleware.AuthMiddleware() 中间件的处理: token 校验。middleware.AuthMiddleware() 函数是通过调用 token.ParseRequest() 来进行 token 校验的。

middleware.AuthMiddleware() 实现详见

<demo09/router/middleware/auth.go>

(https://github.com/lexkong/apiserver_demos/blob/master/demo09/router/middleware/auth.go)

token.ParseRequest() 实现详见

<demo09/pkg/token/token.go>

(https://github.com/lexkong/apiserver_demos/blob/master/demo09/pkg/token/token.go)

编译并测试

1. 下载 apiserver_demos 源码包 (如前面已经下载过, 请忽略此步骤)

```
$ git clone
```

```
https://github.com/lexkong/apiserver\_demos
```

2. 将 apiserver_demos/demo09 复制为 \$GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo09/  
$GOPATH/src/apiserver
```

3. 在 apiserver 目录下编译源码

```
$ cd $GOPATH/src/apiserver  
$ gofmt -w .  
$ go tool vet .  
$ go build -v .
```

上文已经介绍过，API 身份验证首先需要登录，登录成功后会签发 token，之后请求时在 HTTP Header 中带上 token 即可。

1. 用户登录

```
$ curl -XPOST -H "Content-Type: application/json"  
http://127.0.0.1:8080/login -  
d'{"username": "admin", "password": "admin"}'  
  
{  
  "code": 0,  
  "message": "OK",  
  "data": {  
    "token":  
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE1MjgwMTY5MjIsImkiOiJowLCJuYmYiOiE1MjgwMTY5MjIsInVzZXJuYW11IjoieWRtaW4ifQ.LjxrK9DuAwAzUD8-  
9v43NzWBN7HXsSLfebw92DKd1JQ"  
  }  
}
```


返回的 token 为

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOi0jE1MjgwMTY5MjIsImklIjowLCJuYmYiOi0jE1MjgwMTY5MjIsInVzZXJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-9v43NzWBN7HXsSLfebW92DKd1JQ。

2. 请求时如果不携带签发的 token，会禁止请求

```
$ curl -XPOST -H "Content-Type: application/json"
http://127.0.0.1:8080/v1/user -
d '{"username": "user1", "password": "user1234"}'

{
  "code": 20103,
  "message": "The token was invalid.",
  "data": null
}
```

3. 请求时携带 token

```
$ curl -XPOST -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOi0jE1MjgwMTY5MjIsImklIjowLCJuYmYiOi0jE1MjgwMTY5MjIsInVzZXJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-9v43NzWBN7HXsSLfebW92DKd1JQ" -H "Content-Type:
application/json" http://127.0.0.1:8080/v1/user -
d '{"username": "user1", "password": "user1234"}'

{
  "code": 0,
  "message": "OK",
  "data": {
    "username": "user1"
  }
}
```

可以看到携带 token 后验证通过，成功创建用户。通过 HTTP Header `Authorization: Bearer $token` 来携带 token。携带 token 后不需要再次查询数据库核对密码，即可完成授权。

小结

本小节介绍了 API 身份验证的相关知识。apiserver 采用的认证方式为 JWT，小节简单介绍了 JWT 的认证流程，并通过实例展示了具体如何进行 JWT 认证。

通过以上小节的学习，读者已经可以进行基本的 API 开发了，下一节开始介绍 API 开发的进阶内容。