第3章 微服务网关限流&鉴权

课程目标

- 掌握微服务网关Gateway的系统搭建
- 掌握网关限流的实现
- 能够使用BCrypt实现对密码的加密与验证
- 了解加密算法
- 能够使用JWT实现微服务鉴权

1.微服务网关Gateway

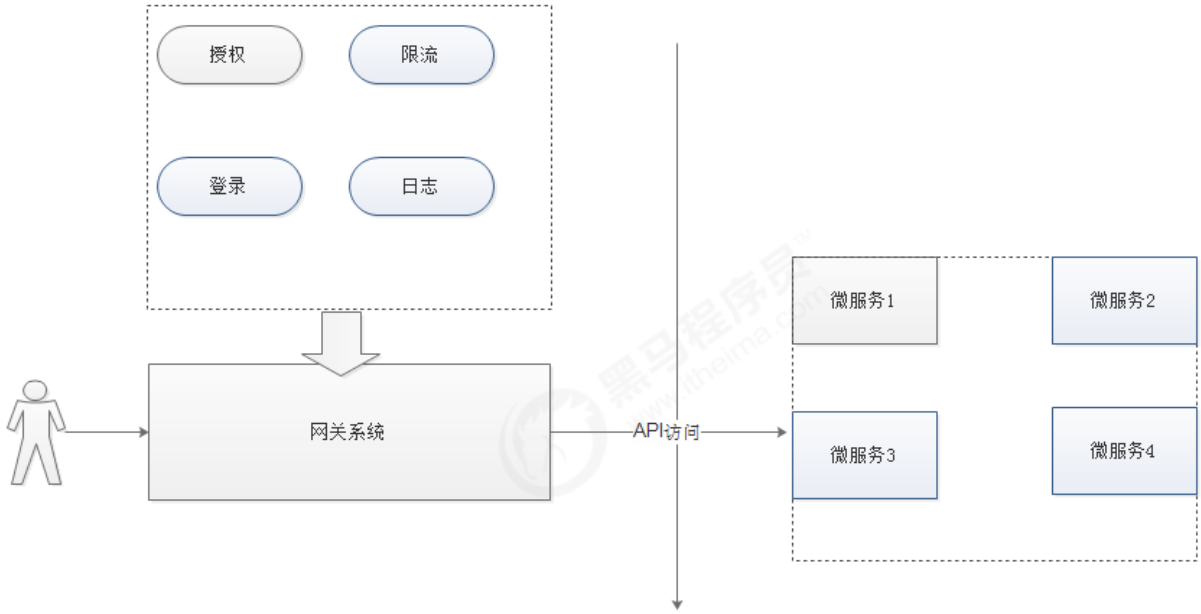
1.1 微服务网关概述

不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求，如果让客户端直接与各个微服务通信，会有以下的问题：

- 客户端会多次请求不同的微服务，增加了客户端的复杂性
- 存在跨域请求，在一定场景下处理相对复杂
- 认证复杂，每个服务都需要独立认证
- 难以重构，随着项目的迭代，可能需要重新划分微服务。例如，可能将多个服务合并成一个或者将一个服务拆分成多个。如果客户端直接与微服务通信，那么重构将会很难实施

以上这些问题可以借助网关解决。

网关是介于客户端和服务器端之间的中间层，所有的外部请求都会先经过 网关这一层。也就是说，API 的实现方面更多的考虑业务逻辑，而安全、性能、监控可以交由 网关来做，这样既提高业务灵活性又不缺安全性，典型的架构图如图所示：



优点如下：

- 安全，只有网关系统对外进行暴露，微服务可以隐藏在内网，通过防火墙保护。
- 易于监控。可以在网关收集监控数据并将其推送到外部系统进行分析。
- 易于统一认证授权。可以在网关上进行认证，然后再将请求转发到后端的微服务，而无须在每个微服务中进行认证。
- 减少了客户端与各个微服务之间的交互次数

总结：微服务网关就是一个系统，通过暴露该微服务网关系统，方便我们进行相关的鉴权，安全控制，日志统一处理，易于监控的相关功能。

实现微服务网关的技术有很多，

- nginx Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务
- zuul ,Zuul 是 Netflix 出品的一个基于 JVM 路由和服务端的负载均衡器。
- spring-cloud-gateway, 是spring 出品的 基于spring 的网关项目，集成断路器，路径重写，性能比Zuul好。

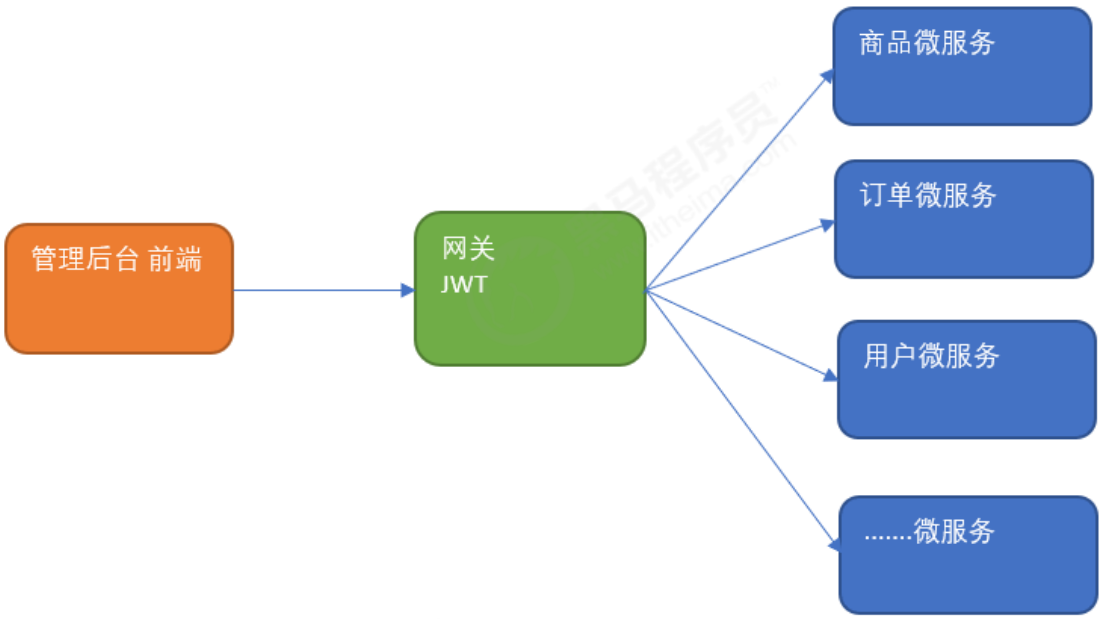
我们使用gateway这个网关技术，无缝衔接到基于spring cloud的微服务开发中来。

gateway官网：

https://spring.io/projects/spring-cloud-gateway

1.2 微服务网关微服务搭建

由于我们开发的系统 有包括前台系统和后台系统，后台的系统给管理员使用。那么也需要调用各种微服务，所以我们针对管理后台搭建一个网关微服务。分析如下：



搭建步骤：

（1）在changgou_gateway工程中，创建changgou_gateway_system工程，pom.xml:

```
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
        </dependency>
```

（2）创建包com.changgou.system, 创建引导类：GatewayApplication

```
@SpringBootApplication
@EnableEurekaClient
public class GatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }
}
```

（3）在resources下创建application.yml

```
spring:
  application:
    name: sysgateway
  cloud:
    gateway:
      routes:
      - id: goods
        uri: lb://goods
        predicates:
        - Path=/goods/**
        filters:
        - StripPrefix= 1
      - id: system
        uri: lb://system
        predicates:
        - Path=/system/**
        filters:
        - StripPrefix= 1
server:
  port: 9101
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:6868/eureka
  instance:
    prefer-ip-address: true
```

参考官方手册：

https://cloud.spring.io/spring-cloud-gateway/spring-cloud-gateway.html#_stripprefix_gatewayfilter_factory

1.3 微服务网关跨域

修改application.yml ,在spring.cloud.gateway节点添加配置，

```
      globalcors:
        cors-configurations:
          '[/**]': # 匹配所有请求
            allowedOrigins: "*" #跨域处理 允许所有的域
            allowedMethods: # 支持的方法
            - GET
            - POST
            - PUT
            - DELETE
```

最终配置文件如下：

```yaml
spring:
  application:
    name: sysgateway
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]': # 匹配所有请求
            allowedOrigins: "*" #跨域处理 允许所有的域
            allowedMethods: # 支持的方法
            - GET
            - POST
            - PUT
            - DELETE
      routes:
      - id: goods
        uri: lb://goods
        predicates:
        - Path=/goods/**
        filters:
        - StripPrefix= 1
server:
  port: 9101
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:6868/eureka
  instance:
    prefer-ip-address: true
```

1.4 微服务网关过滤器

我们可以通过网关过滤器，实现一些逻辑的处理，比如ip黑白名单拦截、特定地址的拦截等。下面的代码中做了两个过滤器，并且设定的先后顺序，只演示过滤器与运行效果。（具体逻辑处理部分学员实现）

（1）changgou_gateway_system创建IpFilter

```
@Component
public class IpFilter implements GlobalFilter, Ordered {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {

        System.out.println("经过第1个过滤器IpFilter");
        ServerHttpRequest request = exchange.getRequest();
        InetSocketAddress remoteAddress = request.getRemoteAddress();
        System.out.println("ip:"+remoteAddress.getHostName());
        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return 1;
    }
}
```

（2）changgou_gateway_system创建UrlFilter

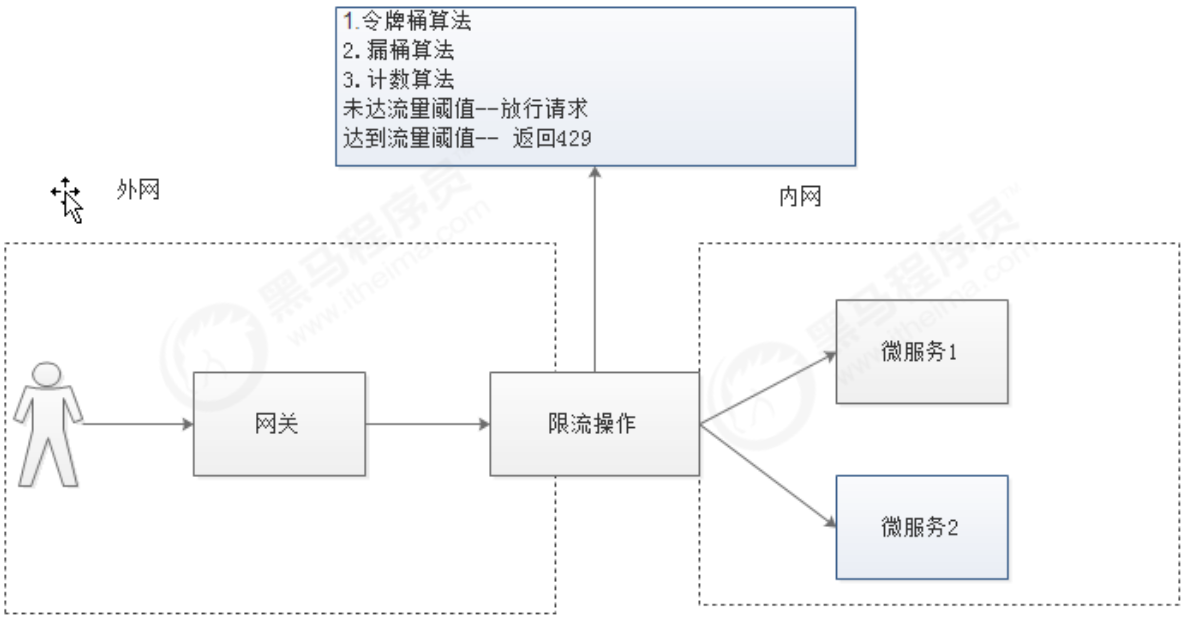
```
@Component
public class UrlFilter implements GlobalFilter, Ordered {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {
        System.out.println("经过第2个过滤器UrlFilter");
        ServerHttpRequest request = exchange.getRequest();
        String url = request.getURI().getPath();
        System.out.println("url:"+url);
        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return 2;
    }

}
```

测试，观察控制台输出。

2 网关限流

我们之前说过，网关可以做很多的事情，比如，限流，当我们的系统被频繁的请求的时候，就有可能将系统压垮，所以为了解决这个问题，需要在每一个微服务中做限流操作，但是如果有了网关，那么就可以在网关系统做限流，因为所有的请求都需要先通过网关系统才能路由到微服务中。

2.1 思路分析



2.2 令牌桶算法

令牌桶算法是比较常见的限流算法之一，大概描述如下：

1）所有的请求在处理之前都需要拿到一个可用的令牌才会被处理；

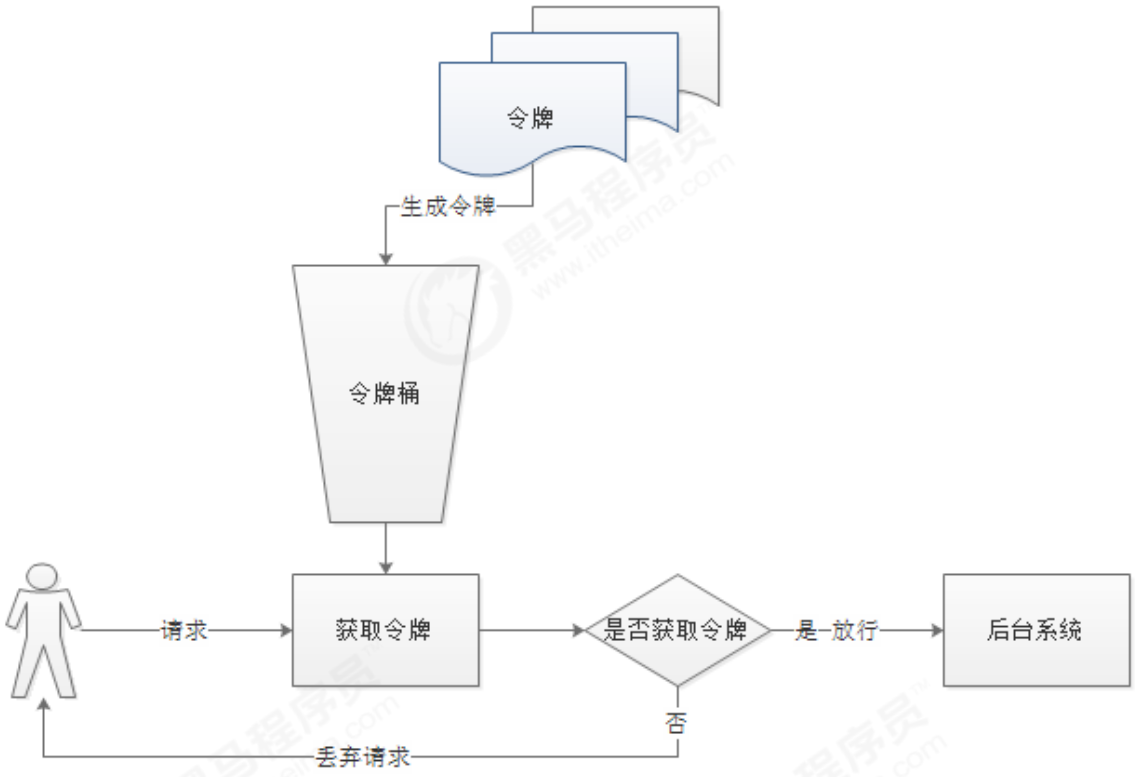
2）根据限流大小，设置按照一定的速率往桶里添加令牌；

3）桶设置最大的放置令牌限制，当桶满时、新添加的令牌就被丢弃或者拒绝；

4）请求达到后首先要获取令牌桶中的令牌，拿着令牌才可以进行其他的业务逻辑，处理完业务逻辑之后，将令牌直接删除；

5）令牌桶有最低限额，当桶中的令牌达到最低限额的时候，请求处理完之后将不会删除令牌，以此保证足够的限流

如下图：



这个算法的实现，有很多技术，Guava(读音: 瓜哇)是其中之一，redis客户端也有其实现。

2.3 网关限流代码实现

需求：每个ip地址1秒内只能发送1次请求，多出来的请求返回429错误。

代码实现：

（1）spring cloud gateway 默认使用redis的RateLimter限流算法来实现。所以我们要使用首先需要引入redis的依赖

```
<!--redis-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>
```

（2）定义KeyResolver

在GatewayApplicatioin引导类中添加如下代码，KeyResolver用于计算某一个类型的限流的KEY也就是说，可以通过KeyResolver来指定限流的Key。

```
    //定义一个KeyResolver
    @Bean
    public KeyResolver ipKeyResolver() {
        return new KeyResolver() {
            @Override
            public Mono<String> resolve(ServerWebExchange exchange) {
                return
Mono.just(exchange.getRequest().getRemoteAddress().getHostName());
            }
        };
    }
```

（3）修改application.yml中配置项，指定限制流量的配置以及REDIS的配置，修改后最终配置如下：

```yaml
spring:
  application:
    name: sysgateway
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]': # 匹配所有请求
            allowedOrigins: "*" #跨域处理 允许所有的域
            allowedMethods: # 支持的方法
            - GET
            - POST
            - PUT
            - DELETE
        routes:
        - id: goods
          uri: lb://goods
          predicates:
          - Path=/goods/**
          filters:
          - StripPrefix= 1
          - name: RequestRateLimiter #请求数限流 名字不能随便写
            args:
              key-resolver: "#{@ipKeyResolver}"
              redis-rate-limiter.replenishRate: 1 #令牌桶每秒填充平均速率
              redis-rate-limiter.burstCapacity: 1 #令牌桶总容量
        - id: system
          uri: lb://system
          predicates:
          - Path=/system/**
          filters:
          - StripPrefix= 1
  # 配置Redis 127.0.0.1可以省略配置
  redis:
    host: 192.168.200.128
    port: 6379
server:
  port: 9101
eureka:

  client:
```

```
    service-url:
      defaultZone: http://127.0.0.1:6868/eureka
  instance:
    prefer-ip-address: true
```

解释：

- burstCapacity：令牌桶总容量。
- replenishRate：令牌桶每秒填充平均速率。
- key-resolver：用于限流的键的解析器的 Bean 对象的名字。它使用 SpEL 表达式根据#{@beanName}从 Spring 容器中获取 Bean 对象。

通过在`replenishRate`和中设置相同的值来实现稳定的速率`burstCapacity`。设置`burstCapacity`高于时，可以允许临时突发`replenishRate`。在这种情况下，需要在突发之间允许速率限制器一段时间（根据`replenishRate`），因为2次连续突发将导致请求被丢弃（`HTTP 429 - Too Many Requests`）

key-resolver: "#{@userKeyResolver}" 用于通过SPEL表达式来指定使用哪一个KeyResolver.

如上配置：

表示 一秒内，允许 一个请求通过，令牌桶的填充速率也是一秒钟添加一个令牌。

最大突发状况 也只允许 一秒内有一次请求，可以根据业务来调整 。

（4）测试

启动redis

启动注册中心

启动商品微服务

启动gateway网关

打开浏览器 http://localhost:9101/goods/brand

快速刷新，当1秒内发送多次请求，就会返回429错误。

3. BCrypt密码加密

3.1 BCrypt快速入门

在用户模块，对于用户密码的保护，通常都会进行加密。我们通常对密码进行加密，然后存放在数据库中，在用户进行登录的时候，将其输入的密码进行加密然后与数据库中存放的密文进行比较，以验证用户密码是否正确。 目前，MD5和BCrypt比较流行。相对来说，BCrypt比MD5更安全。因为其内部引入的加盐机制

BCrypt 官网http://www.mindrot.org/projects/jBCrypt/

（1）新建测试类，main方法中编写代码，实现对密码的加密

```
public class TestBcrypt {

    public static void main(String[] args) {
        /**
         * 得到盐
         * 盐是一个随机生成的含有29个字符的字符串,并且会与密码一起合并进行最终的密文生成
         * 并且每一次生成的盐的值都是不同的
         */
        for(int i=0;i<10;i++){
            String gensalt = BCrypt.gensalt();
            System.out.println("salt:"+gensalt);
            String saltPassword = BCrypt.hashpw("123456", gensalt);
            System.out.println("本次生成的密码:"+saltPassword);
        }
    }
}
```

（2）main方法中编写代码，实现对密码的校验。BCrypt不支持反运算，只支持密码校验。

```
//校验密码
boolean checkpw = BCrypt.checkpw("123456", saltPassword);
System.out.println("密码校验结果:"+checkpw);
```

3.2 新增管理员密码加密

3.2.1 需求与表结构分析

新增管理员，使用BCrypt进行密码加密

id	int	主键id
login_name	varchar	登录名
password	varchar	密码
status	char	状态

3.2.2 代码实现

（1）修改changgou_service_system项目的AdminServiceImpl

```
    /**
     * 增加
     * @param admin
     */
    @Override
    public void add(Admin admin){
        String password = BCrypt.hashpw(admin.getPassword(),
BCrypt.gensalt());
        admin.setPassword(password);
        adminMapper.insert(admin);
    }
```

3.3 管理员登录密码验证

3.3.1 需求分析

系统管理用户需要管理后台，需要先输入用户名和密码进行登录，才能进入管理后台。

思路：

用户发送请求，输入用户名和密码

后台管理微服务controller接收参数，验证用户名和密码是否正确，如果正确则返回用户登录成功结果

3.3.2 代码实现

（1）AdminService新增方法定义

```
    /**
     * 登录验证密码
     * @param admin
     * @return
     */
    boolean login(Admin admin);
```

（2）AdminServiceImpl实现此方法

```
    @Override
    public boolean login(Admin admin) {
        //根据登录名查询管理员
        Admin admin1=new Admin();
        admin1.setLoginName(admin.getLoginName());
        admin1.setStatus("1");
        Admin admin2 = adminMapper.selectOne(admin1);//数据库查询出的对象
        if(admin2==null){
            return false;
        }else{
            //验证密码，Bcrypt为spring的包，第一个参数为明文密码，第二个参数为密文密码
            return
BCrypt.checkpw(admin.getPassword(),admin2.getPassword());
        }
    }
```

（3）AdminController新增方法

```
    /**
     * 登录
     * @param admin
     * @return
     */
    @PostMapping("/login")
    public Result login(@RequestBody Admin admin){
        boolean login = adminService.login(admin);
        if(login){
            return new Result();
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

4.加密算法(了解)

由于在学习**JWT**的时候会涉及使用很多加密算法, 所以在这里做下扫盲, 简单了解就可以

加密算法种类有:

4.1.可逆加密算法

解释: 加密后, 密文可以反向解密得到密码原文.

4.1.1. 对称加密

【文件加密和解密使用相同的密钥，即加密密钥也可以用作解密密钥】

解释: 在对称加密算法中，数据发信方将明文和加密密钥一起经过特殊的加密算法处理后，使其变成复杂的加密密文发送出去，收信方收到密文后，若想解读出原文，则需要使用加密时用的密钥以及相同加密算法的逆算法对密文进行解密，才能使其回复成可读明文。在对称加密算法中，使用的密钥只有一个，收发双方都使用这个密钥，这就需要解密方事先知道加密密钥。

优点: 对称加密算法的优点是算法公开、计算量小、加密速度快、加密效率高。

缺点: 没有非对称加密安全.

用途： 一般用于保存用户手机号、身份证等敏感但能解密的信息。

常见的对称加密算法有: `AES、DES、3DES、Blowfish、IDEA、RC4、RC5、RC6、HS256`

4.1.2. 非对称加密

【两个密钥：公开密钥（**publickey**）和私有密钥，公有密钥加密，私有密钥解密】

*解释: * 同时生成两把密钥：私钥和公钥，私钥隐秘保存，公钥可以下发给信任客户端.

加密与解密:

- 私钥加密，持有私钥或公钥才可以解密
- 公钥加密，持有私钥才可解密

签名:

- 私钥签名，持有公钥进行验证是否被篡改过.

*优点: * 非对称加密与对称加密相比，其安全性更好；

缺点: 非对称加密的缺点是加密和解密花费时间长、速度慢，只适合对少量数据进行加密。
用途： 一般用于签名和认证。私钥服务器保存, 用来加密，公钥客户拿着用于对于令牌或者签名的解密或者校验使用.

常见的非对称加密算法有： `RSA、DSA（数字签名用）、ECC（移动设备用）、RS256 (采用 SHA-256 的 RSA 签名)`

4.2.不可逆加密算法

解释: 一旦加密就不能反向解密得到密码原文.

种类: Hash加密算法, 散列算法, 摘要算法等

用途：一般用于效验下载文件正确性，一般在网站上下载文件都能见到；存储用户敏感信息，如密码、 卡号等不可解密的信息。

常见的不可逆加密算法有： `MD5、SHA、HMAC`

4.3.Base64编码

Base64是网络上最常见的用于传输8Bit字节代码的编码方式之一。Base64编码可用于在HTTP环境下传递较长的标识信息。采用Base64编码解码具有不可读性，即所编码的数据不会被人用肉眼所直接看到。注意：**Base64**只是一种编码方式，不算加密方法。

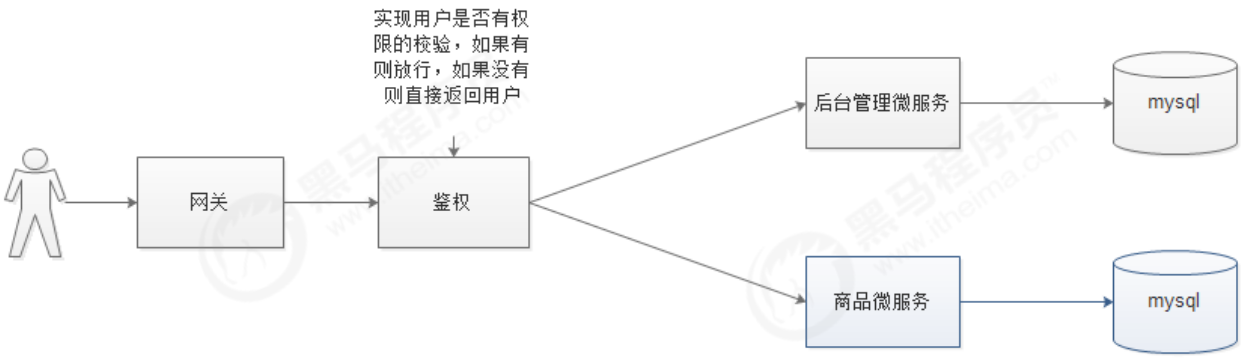
在线编码工具：

http://www.jsons.cn/img2base64/

5. JWT 实现微服务鉴权

5.1 什么是微服务鉴权

我们之前已经搭建过了网关，使用网关在系统中比较适合进行权限校验。



那么我们可以采用JWT的方式来实现鉴权校验。

5.2 JWT

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息。

一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名。

头部（**Header**）

头部用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。这也可以被表示成一个JSON对象。

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

在头部指明了签名算法是HS256算法。 我们进行BASE64编码http://base64.xpcha.com/，编码后的字符串如下：

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

载荷（**playload**）

载荷就是存放有效信息的地方。

定义一个payload:

```
{"sub":"1234567890","name":"John Doe","admin":true}
```

然后将其进行base64加密，得到Jwt的第二部分。

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
```

签证（**signature**）

jwt的第三部分是一个签证信息，这个签证信息由三部分组成：

> header (base64后的)
>
> payload (base64后的)
>
> secret

这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符串，然后通过header中声明的加密方式进行加盐secret组合加密，然后就构成了jwt的第三部分。

```
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

将这三部分用.连接成一个完整的字符串,构成了最终的jwt:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

5.3 JJWT签发与验证token

JJWT是一个提供端到端的JWT创建和验证的Java库。永远免费和开源(Apache License，版本2.0)，JJWT很容易使用和理解。它被设计成一个以建筑为中心的流畅界面，隐藏了它的大部分复杂性。

官方文档：

https://github.com/jwtk/jjwt

5.3.1 创建token

（1）新建项目jwtTest中的pom.xml中添加依赖：

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

(2) 创建测试类，代码如下

```
JwtBuilder builder= Jwts.builder()
    .setId("888")   //设置唯一编号
    .setSubject("小白")//设置主题  可以是JSON数据
    .setIssuedAt(new Date())//设置签发日期
    .signWith(SignatureAlgorithm.HS256,"itcast");//设置签名 使用HS256算法，
并设置SecretKey(字符串)
//构建 并返回一个字符串
System.out.println( builder.compact() );
```

运行打印结果：

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJpYXQiOjE1NTc5M
DQxODF9.ThecMfgYjtoys3JX7dpx3hu6pUm0piZ0tXXreFU_u3Y
```

再次运行，会发现每次运行的结果是不一样的，因为我们的载荷中包含了时间。

5.3.2 解析token

我们刚才已经创建了token，在web应用中这个操作是由服务端进行然后发给客户端，客户端在下次向服务端发送请求时需要携带这个token（这就好像是拿着一张门票一样），那服务端接到这个token 应该解析出token中的信息（例如用户id）,根据这些信息查询数据库返回相应的结果。

```
String
compactJwt="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJp
YXQiOjE1NTc5MDQxODF9.ThecMfgYjtoys3JX7dpx3hu6pUm0piZ0tXXreFU_u3Y";

Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJwt).getBody(
);

System.out.println(claims);
```

运行打印效果：

```
{jti=888, sub=小白, iat=1557904181}
```

试着将token或签名秘钥篡改一下，会发现运行时就会报错，所以解析token也就是验证token.

5.3.3 设置过期时间

有很多时候，我们并不希望签发的token是永久生效的，所以我们可以为token添加一个过期时间。

（1）创建token 并设置过期时间

```
//当前时间
long currentTimeMillis = System.currentTimeMillis();
Date date = new Date(currentTimeMillis);
JwtBuilder builder= Jwts.builder()
    .setId("888")   //设置唯一编号
    .setSubject("小白")//设置主题  可以是JSON数据
    .setIssuedAt(new Date())//设置签发日期
    .setExpiration(date)
    .signWith(SignatureAlgorithm.HS256,"itcast");//设置签名 使用HS256算法，
并设置SecretKey(字符串)
//构建 并返回一个字符串
System.out.println( builder.compact() );
```

解释：

```
.setExpiration(date)//用于设置过期时间 ， 参数为Date类型数据
```

运行，打印效果如下：

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJpYXQiOjE1NTc5M
DUzMDgsImV4cCI6MTU1NzkwNTMwOH0.4q5AaTyBRf8SB9B3Tl-I53PrILGyicJC3fgR3gWbvUI
```

（2）解析TOKEN

```
String
compactJwt="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJp
YXQiOjE1NTc5MDUzMDgsImV4cCI6MTU1NzkwNTMwOH0.4q5AaTyBRf8SB9B3Tl-
I53PrILGyicJC3fgR3gWbvUI";

Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJwt).getBody(
);

System.out.println(claims);
```

打印效果：

```
ava\jdk1.8.0_144\bin\java" ...

:dJwtException: JWT expired at 2019-05-15T15:28:28Z. Current time: 2019-05-15T15:29:47Z, a difference of 79842 milliseconds.

ı.impl.DefaultJwtParser.parse(DefaultJwtParser.java:385)
ı.impl.DefaultJwtParser.parse(DefaultJwtParser.java:481)
ı.impl.DefaultJwtParser.parseClaimsJws(DefaultJwtParser.java:541)
:st.JWTTest.parseJWT(JWTTest.java:40) <22 internal calls>
```

当前时间超过过期时间，则会报错。

5.3.4 自定义claims

我们刚才的例子只是存储了id和subject两个信息，如果你想存储更多的信息（例如角色）可以定义自定义claims。

创建测试类，并设置测试方法：

创建token:

```
@Test
public void createJWT(){
    //当前时间
    long currentTimeMillis = System.currentTimeMillis();
    currentTimeMillis+=1000000L;
    Date date = new Date(currentTimeMillis);
    JwtBuilder builder= Jwts.builder()
        .setId("888")   //设置唯一编号
        .setSubject("小白")//设置主题  可以是JSON数据
        .setIssuedAt(new Date())//设置签发日期
        .setExpiration(date)//设置过期时间
        .claim("roles","admin")//设置角色
        .signWith(SignatureAlgorithm.HS256,"itcast");//设置签名 使用HS256算法，并设置SecretKey(字符串)
    //构建 并返回一个字符串
    System.out.println( builder.compact() );
}
```

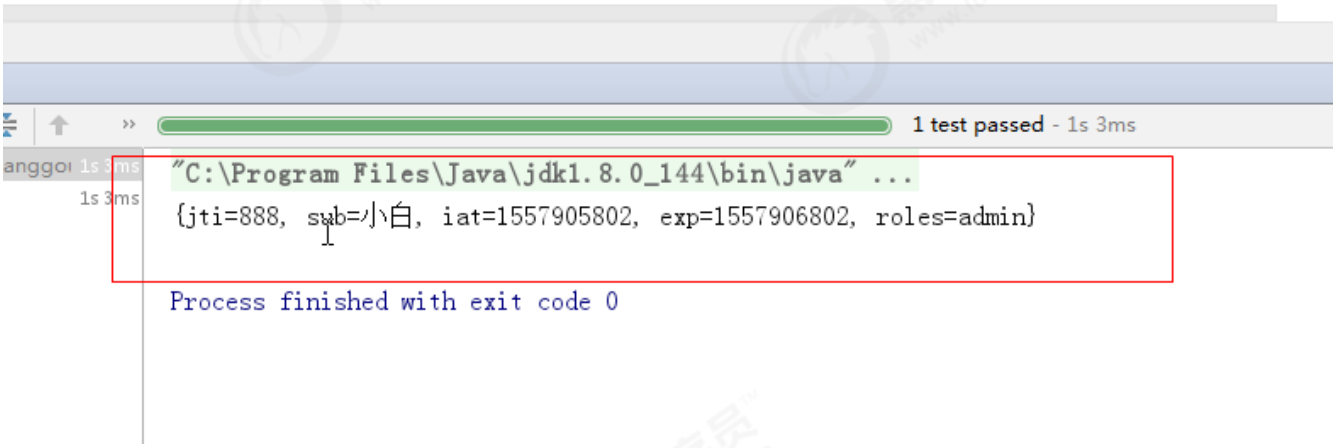
运行打印效果：

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJpYXQiOjE1NTc5M
DU4MDIsImV4cCI6MTU1NzkwNjgwMiwicm9sZXMiOiJhZG1pbiJ9.AS5Y2fNCwUzQQxXh_QQWMp
aB75YqfuK-2P7VZiCXEJI
```

解析TOKEN:

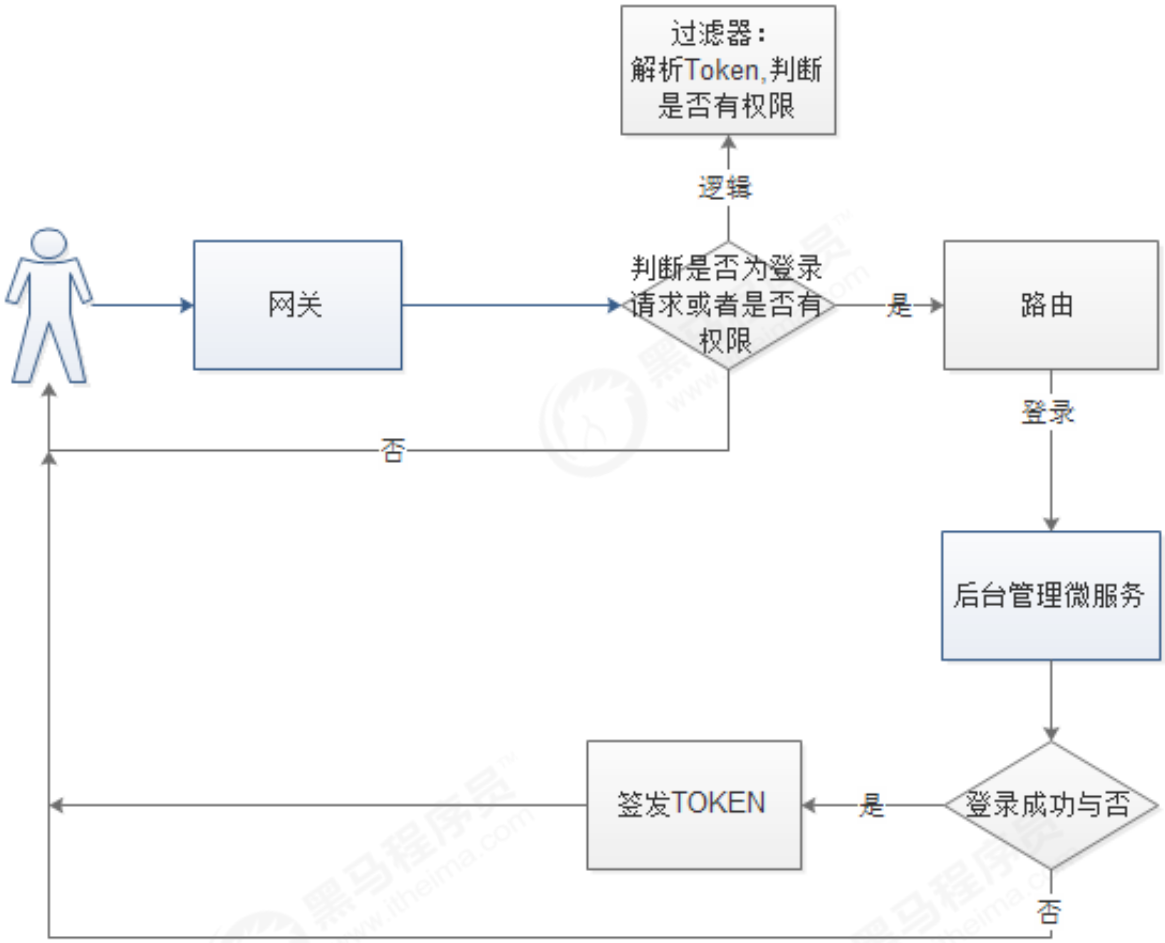
```
//解析
@Test
public void parseJWT(){
    String
compactJwt="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJp
YXQiOjE1NTc5MDU4MDIsImV4cCI6MTU1NzkwNjgwMiwicm9sZXMiOiJhZG1pbiJ9.AS5Y2fNC
wUzQQxXh_QQWMpaB75YqfuK-2P7VZiCXEJI";
    Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJwt).getBody(
);
    System.out.println(claims);
}
```

运行效果：



5.4 畅购微服务鉴权代码实现

5.4.1 思路分析



1．用户进入网关开始登陆，网关过滤器进行判断，如果是登录，则路由到后台管理微服务进行登录
2．用户登录成功，后台管理微服务签发JWT TOKEN信息返回给用户
3．用户再次进入网关开始访问，网关过滤器接收用户携带的TOKEN
4．网关过滤器解析TOKEN ，判断是否有权限，如果有，则放行，如果没有则返回未认证错误

5.4.2 系统微服务签发token

（1）在changgou_service_system添加依赖

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.0</version>
</dependency>
```

（2）在changgou_service_system中创建类： JwtUtil

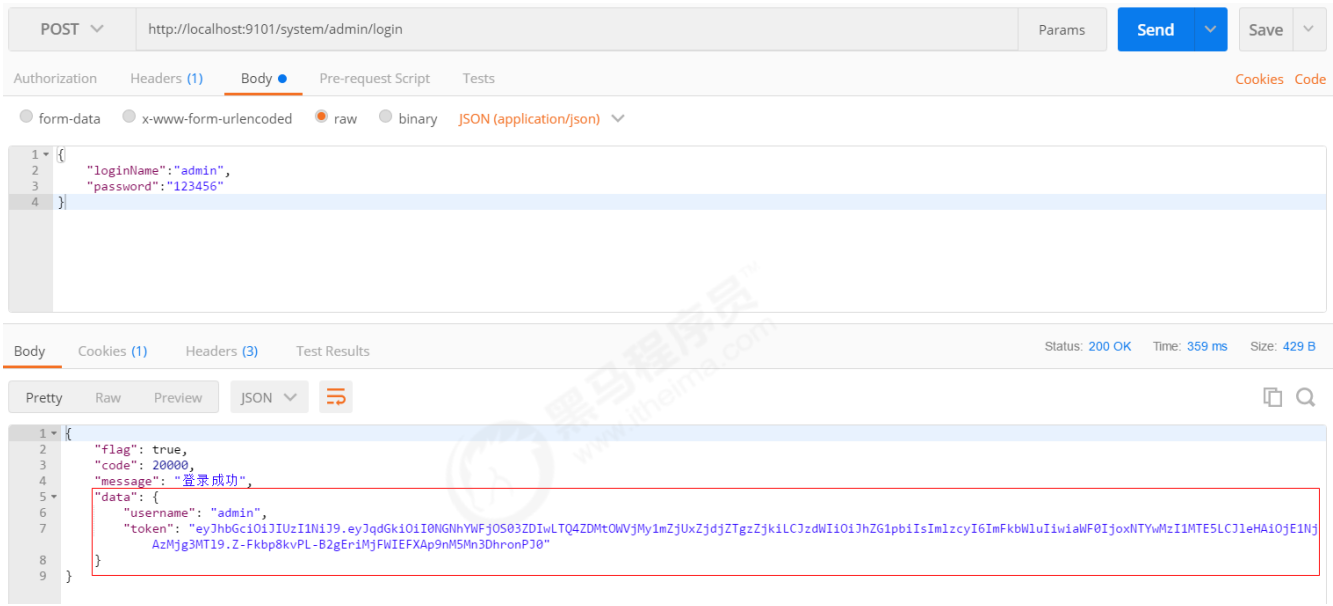
```
package com.changgou.system.util;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Date;
/**
 * JWT工具类
 */
public class JwtUtil {

    //有效期为
    public static final Long JWT_TTL = 3600000L;// 60 * 60 *1000  一个小时
    //设置秘钥明文
    public static final String JWT_KEY = "itcast";

    /**
     * 创建token
     * @param id
     * @param subject
     * @param ttlMillis
     * @return
     */
    public static String createJWT(String id, String subject, Long
ttlMillis) {

        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);
        if(ttlMillis==null){
            ttlMillis=JwtUtil.JWT_TTL;
        }
        long expMillis = nowMillis + ttlMillis;
        Date expDate = new Date(expMillis);
        SecretKey secretKey = generalKey();

        JwtBuilder builder = Jwts.builder()

                .setId(id)                //唯一的ID
```

```
                .setSubject(subject)   // 主题  可以是JSON数据
                .setIssuer("admin")     // 签发者
                .setIssuedAt(now)      // 签发时间
                .signWith(signatureAlgorithm, secretKey) //使用HS256对称加
密算法签名，第二个参数为秘钥
                .setExpiration(expDate);// 设置过期时间
        return builder.compact();
    }

    /**
     * 生成加密后的秘钥 secretKey
     * @return
     */
    public static SecretKey generalKey() {
        byte[] encodedKey = Base64.getDecoder().decode(JwtUtil.JWT_KEY);
        SecretKey key = new SecretKeySpec(encodedKey, 0,
encodedKey.length, "AES");
        return key;
    }
}
```

（3）修改AdminController的login方法，用户登录成功 则 签发TOKEN

```
    /**
     * 登录
     * @param admin
     * @return
     */
    @PostMapping("/login")
    public Result login(@RequestBody Admin admin){
        boolean login = adminService.login(admin);
        if(login){  //如果验证成功
            Map<String,String> info = new HashMap<>();
            info.put("username",admin.getLoginName());
            String token =
JwtUtil.createJWT(UUID.randomUUID().toString(), admin.getLoginName(),
null);
            info.put("token",token);
            return new Result(true, StatusCode.OK,"登录成功",info);
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

使用postman 测试



5.4.3 网关过滤器验证token

（1）在changgou_gateway_system网关系统添加依赖

```
<!--鉴权-->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

（2）创建JWTUtil类

```
package com.changgou.gateway.util;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

/**
 * jwt校验工具类
 */
public class JwtUtil {

    //有效期为
    public static final Long JWT_TTL = 3600000L;// 60 * 60 *1000  一个小时
    //设置秘钥明文
    public static final String JWT_KEY = "itcast";

    /**
     * 生成加密后的秘钥 secretKey
     *
     * @return
     */
    public static SecretKey generalKey() {
        byte[] encodedKey = Base64.getDecoder().decode(JwtUtil.JWT_KEY);
        SecretKey key = new SecretKeySpec(encodedKey, 0,
encodedKey.length, "AES");
        return key;
    }

    /**
     * 解析
     *
     * @param jwt
     * @return
     * @throws Exception
     */
    public static Claims parseJWT(String jwt) throws Exception {
        SecretKey secretKey = generalKey();
        return Jwts.parser()

                .setSigningKey(secretKey)
```

```
                .parseClaimsJws(jwt)
                .getBody();
    }
}
```

（3）创建过滤器，用于token验证

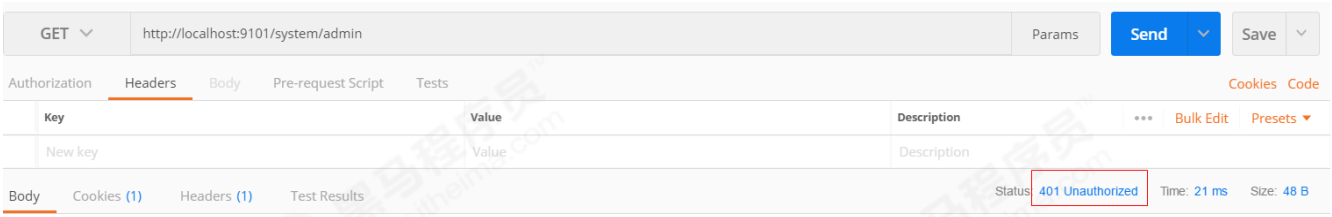
```
/**
 * 鉴权过滤器 验证token
 */
@Component
public class AuthorizeFilter implements GlobalFilter, Ordered {
    private static final String AUTHORIZE_TOKEN = "token";

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {
        //1. 获取请求
        ServerHttpRequest request = exchange.getRequest();
        //2. 则获取响应
        ServerHttpResponse response = exchange.getResponse();
        //3. 如果是登录请求则放行
        if (request.getURI().getPath().contains("/admin/login")) {
            return chain.filter(exchange);
        }
        //4. 获取请求头
        HttpHeaders headers = request.getHeaders();
        //5. 请求头中获取令牌
        String token = headers.getFirst(AUTHORIZE_TOKEN);

        //6. 判断请求头中是否有令牌
        if (StringUtils.isEmpty(token)) {
            //7. 响应中放入返回的状态吗，没有权限访问
            response.setStatusCode(HttpStatus.UNAUTHORIZED);
            //8. 返回
            return response.setComplete();
        }

        //9. 如果请求头中有令牌则解析令牌
        try {
            JwtUtil.parseJWT(token);
        } catch (Exception e) {
            e.printStackTrace();
            //10. 解析jwt令牌出错，说明令牌过期或者伪造等不合法情况出现
            response.setStatusCode(HttpStatus.UNAUTHORIZED);
            //11. 返回

            return response.setComplete();
```

```
            }
            //12. 放行
            return chain.filter(exchange);
        }

        @Override
        public int getOrder() {
            return 0;
        }
    }
```

（4）测试：

注意: 数据库中管理员账户为 : **admin** ，密码为 : **123456**

如果不携带token直接访问，则返回401错误



如果携带正确的token，则返回查询结果