

# 第8章 微服务网关和Jwt令牌

---

## 学习目标

---

- 搜索页域名配置
- 静态页域名配置
- 静态页发布
- 对接搜索页
- 掌握微服务网关的系统搭建-作用
- 了解什么是微服务网关以及它的作用[回顾微服务网关的路由过滤规则]
- 掌握系统中心微服务的搭建[用户微服务]
- 掌握用户密码加密存储bcrypt
- 了解JWT鉴权的介绍
- 掌握JWT的鉴权[权限认证]的使用
- 使用Jwt令牌来存储用户登录信息，在微服务网关中识别登录信息(用户的身份)
- 掌握网关使用JWT进行校验
- 掌握网关限流-[令牌桶]

## 1 搜索页详情页对接

---

### 目标

- 实现搜索页与详情页对接

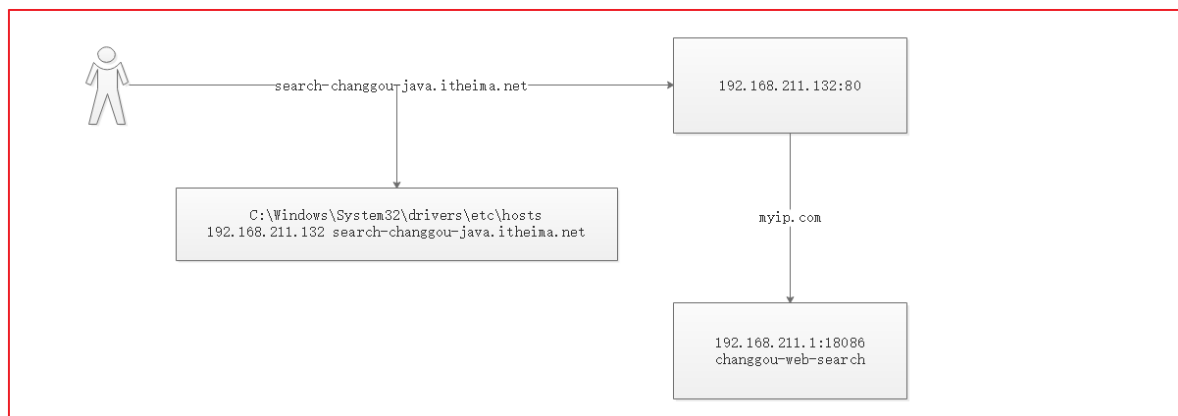
### 路径

- 搜索页域名配置
- 用Nginx发布商品详情页
- 搜索页与详情页对接

### 讲解

#### 1.1 搜索域名配置

(1)搜索域名配置分析



用户搜索域名的时候，先经过虚拟机的nginx，然后将请求转发到myip.com对应的服务器的18086端口进行处理。

我们首先在 `C:\windows\System32\drivers\etc\hosts` 添加域名映射

```
1 | 192.168.211.132 search-changgou-java.itheima.net
```

## (2)nginx修改

搜索的域名为 `search-changgou-java.itheima.net`，我们可以在虚拟机的nginx中配置该域名的解析。修改虚拟机中的nginx，添加如下配置：

```
server {
    listen 80;
    server_name search-changgou-java.itheima.net;

    location / {
        proxy_pass http://myip.com:18086;
    }
}
```

```
1 server {
2     listen 80;
3     server_name search-changgou-java.itheima.net;
4
5     location / {
6         proxy_pass http://myip.com:18086;
7     }
8 }
```

## (3)启动搜索微服务

如果启动的时候出现如下错误，原因是因为bean对象重复了，我们需要配置下配置文件，让程序中的对象覆盖重复的对象。

Description:

The bean 'goods.FeignClientSpecification', defined in null, could not be registered. A bean with that name has already been defined in null and overriding is disabled.

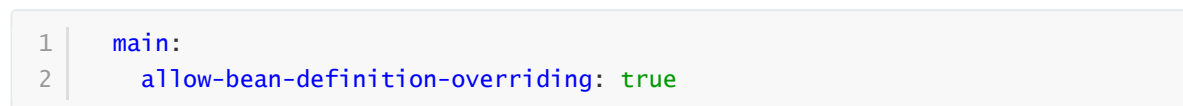
Action:

Consider renaming one of the beans or enabling overriding by setting `spring.main.allow-bean-definition-overriding=true`

修改配置文件，添加如下配置：

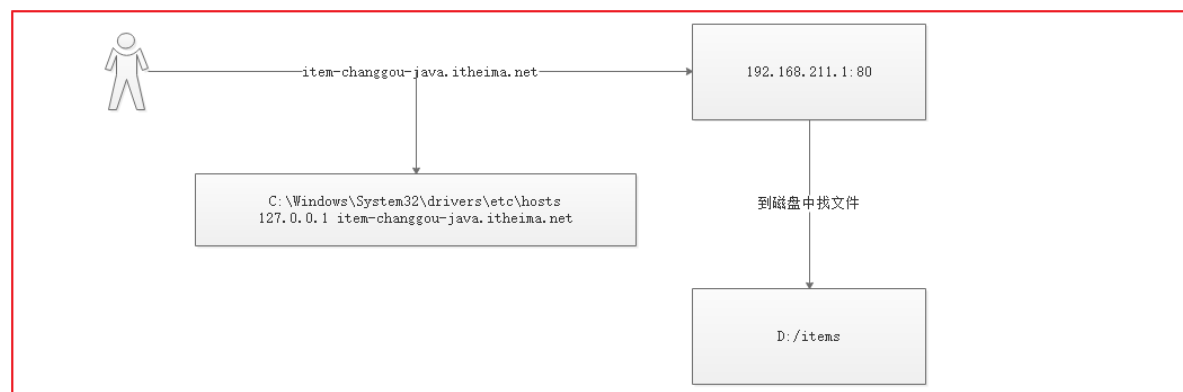


配置如下：

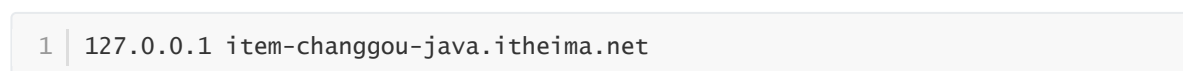


用域名访问 `http://search-changgou-java.itheima.net/search/list`

## 1.2 静态页发布



修改 `C:\windows\System32\drivers\etc\hosts` 添加 `item-changgou-java.itheima.net` 的映射解析：



添加Nginx配置，在本地将nginx解压，并且修改配置文件nginx.conf，添加 `item-changgou-java.itheima.net`，配置如下：

```

1      #windows的静态页面部署
2      server {
3          listen      80;
4          server_name  item-changgou-java.itheima.net;
5
6          location / {
7              root     D:/items;
8          }
9      }
10     #linux的静态页面部署
11     server {
12         listen      80;
13         server_name  item-changgou-java.itheima.net;
14
15         #charset koi8-r;
16
17         #access_log  logs/host.access.log  main;
18
19         location / {
20             root     /usr/local/items;
21         }
22     }
23

```

## 1.3 搜索页对接静态页

### (1)详情页跳转

修改搜索详情页跳转地址，代码如下：

```

<div class="p-img">
  <a th:href="@{'http://item-changgou-java.itheima.net/' + ${sku.spuId} + '.html' (id=${sku.id})}" target="_blank">
    
  </a>
</div>

```

### (2)选中指定Sku

在详情页中选择当前用户选中的Sku

```

//获取请求路径中的指定参数
getUrlKey:function(name){
    return decodeURIComponent((new RegExp('[?|&]' + name + '=' + '([^&]+?)(&|#|;|$)').exec(location.href) || [, ""])[1].r
},
//初始化加载
loadDefault:function () {
    //获取请求地址中id
    let id = this.getUrlKey('id');
    if(id!=null){
        for(var i=0;i<this.skuList.length;i++){
            //当前的sku
            let csku = this.skuList[i];
            if(csku.id==id){
                /**
                 * 默认选中第1个
                 */
                this.sku = JSON.parse(JSON.stringify(csku));
                //默认的规格
                this.spec=JSON.parse(csku.spec);
            }
        }
    }
}

created:function () {
    //加载
    this.loadDefault();
}

```

上图代码如下:

```

1 //获取请求路径中的指定参数
2 getUrlKey:function(name){
3     return decodeURIComponent((new RegExp('[?|&]' + name + '=' + '([^&]+?)(&|#|;|$)').exec(location.href) || [, ""])[1].replace(/\+/g, '%80')) || null
4 },
5
6 //初始化加载
7 loadDefault:function () {
8     //获取请求地址中id
9     let id = this.getUrlKey('id');
10    if(id!=null){
11        for(var i=0;i<this.skuList.length;i++){
12            //当前的sku
13            let csku = this.skuList[i];
14            if(csku.id==id){
15                /**
16                 * 默认选中第1个
17                 */
18                this.sku = JSON.parse(JSON.stringify(csku));
19                //默认的规格
20                this.spec=JSON.parse(csku.spec);
21            }
22        }
23    }
24 }

```

访问地址: <http://item-changgou-java.itheima.net/No1210166544772890624.html?id=No1210166545255235584>

← → ↺

① 不安全 | item-changgou-java.itheima.net/No121016654472890624.html?id=No1210166545255235584

☆

🔍

📄

📱

📺

🌐

🔗

🔒

青橙欢迎您! 请登录 | 免费注册

我的订单 | 我的购物车 | 我的青橙 | 青橙会员 | 企业采购 | 关注青橙 | 合作招商 | 商家后台 | 网站导航

畅购

CHANG GOU

全部商品分类

服装城

美妆馆

青橙超市

全球购

闪购

团购

有趣

秒杀

教育/培训 / 计算机培训 / 人工智能

OpenCV3

编程入门

京东自营

1

2

3

4

OpenCV 5G未来市场 项目实战 项目经理 资料

推荐选择下方[移动优惠购],手机套餐齐搞定,不用换号,每月还有花惠返

价 格

¥ 3245

降价通知

累计评价 1111111

促 销

加价购

满999.00另加20.00元,或满1999.00另加30.00元,或满2999.00另加40.00元,即可在购物车换购热销商品

支 持

以旧换新,闲置手机回收 4G套餐超值抢 礼品购

配 送 至

满999.00另加20.00元,或满1999.00另加30.00元,或满2999.00另加40.00元,即可在购物车换购热销商品【“学习内容”:“项目实战”,“包装内容”:“资料”,“课程分类”:“项目经理”】

学习内容

基础知识

框架学习

项目实战

课程分类

少儿编程

产品经理

项目经理

包装内容

源码

视频

资料

1

+

-

加入购物车

## 2 微服务网关

### 目标

- 微服务网关回顾

### 路径

- 微服务网关介绍
- 微服务网关技术应用

### 讲解

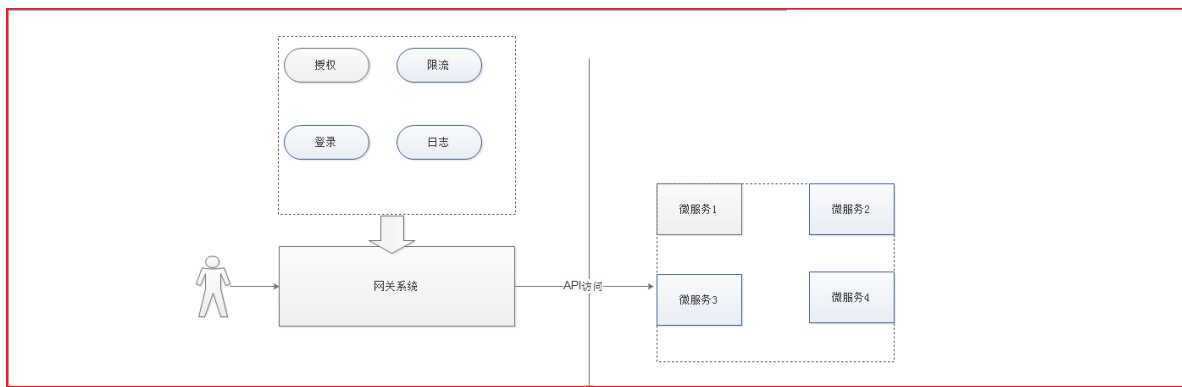
#### 2.1 微服务网关的概述

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
- 某些微服务可能使用了防火墙 / 浏览器不友好的协议，直接访问会有一些的困难

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。
- 易于认证。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
- 易于统一授权。

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。

## 2.2 微服务网关技术

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>

## 3 网关系统使用

### 目标

- 微服务网关使用回顾

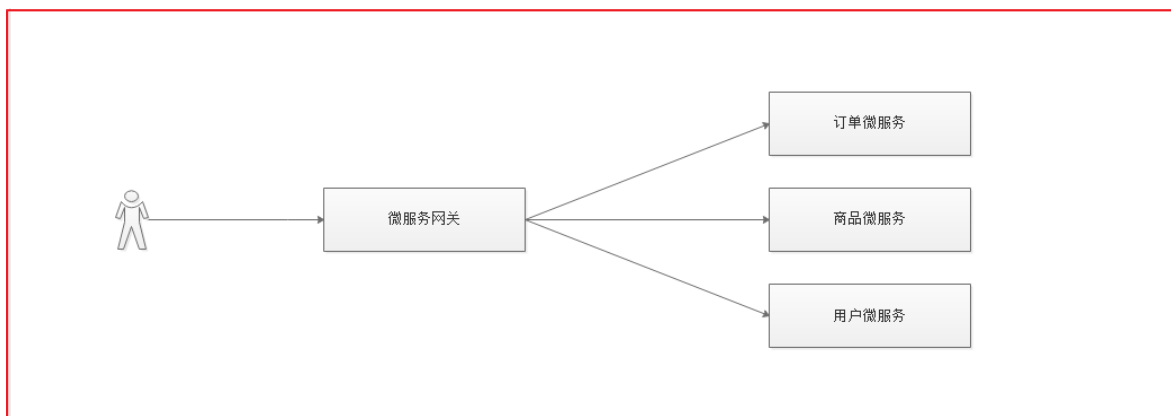
### 路径

- 搭建微服务网关(用于路由)
- 跨域配置
- 微服务网关的过滤器
- 微服务网关限流

# 讲解

## 3.1 需求分析

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 系统管理搭建一个网关系统。分析如下：



## 3.2 搭建后台网关系统

### 3.2.1 搭建分析

由上可知道，由于 需要多个网关，所以为了管理方便。我们新建一个项目，打包方式为pom,在里面建立各种网关系统模块即可。如图所示：

```
▼ changgou-gateway
  > changgou-gateway-web
    pom.xml
```

### 3.2.2 工程搭建

(1)引入依赖

修改changgou-gateway工程，打包方式为pom

pom.xml如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>changgou-parent</artifactId>
8         <groupId>com.changgou</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>changgou-gateway</artifactId>
13    <packaging>pom</packaging>
14    <modules>
15        <module>changgou-gateway-web</module>
16    </modules>
```



```

17      <!--网关依赖-->
18      <dependencies>
19          <dependency>
20              <groupId>org.springframework.cloud</groupId>
21              <artifactId>spring-cloud-starter-gateway</artifactId>
22          </dependency>
23
24          <dependency>
25              <groupId>org.springframework.cloud</groupId>
26              <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
27          </dependency>
28          <dependency>
29              <groupId>org.springframework.cloud</groupId>
30              <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
31          </dependency>
32      </dependencies>
33
34  </project>

```

在changgou-gateway工程中，创建 changgou-gateway-web工程，该网关主要用于对后台微服务进行一个调用操作，将多个微服务串联到一起。

pom.xml:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>changgou-gateway</artifactId>
7          <groupId>com.changgou</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11     <artifactId>changgou-gateway-web</artifactId>
12     <description>
13         普通web请求网关
14     </description>
15 </project>

```

## (2)引导类

在changgou-gateway-web中创建一个引导类com.changgou.GatewayWebApplication，代码如下：

```

1  @SpringBootApplication
2  @EnableEurekaClient
3  public class GatewayWebApplication {
4
5      public static void main(String[] args) {
6          SpringApplication.run(GatewayWebApplication.class, args);
7      }
8  }

```

### (3) application.yml配置

在changgou-gateway-web的resources下创建application.yml,代码如下:

```

1  spring:
2      application:
3          name: gateway-web
4  server:
5      port: 8001
6  eureka:
7      client:
8          service-url:
9              defaultZone: http://127.0.0.1:7001/eureka
10     instance:
11         prefer-ip-address: true
12 management:
13     endpoint:
14         gateway:
15             enabled: true #Actuator 监控
16     web:
17         exposure:
18             include: true #暴露所有的监控点info,health,beans,env

```

## 3.3 跨域配置

有时候,我们需要对所有微服务跨域请求进行处理,则可以在gateway中进行跨域支持。修改application.yml,添加如下代码:

```

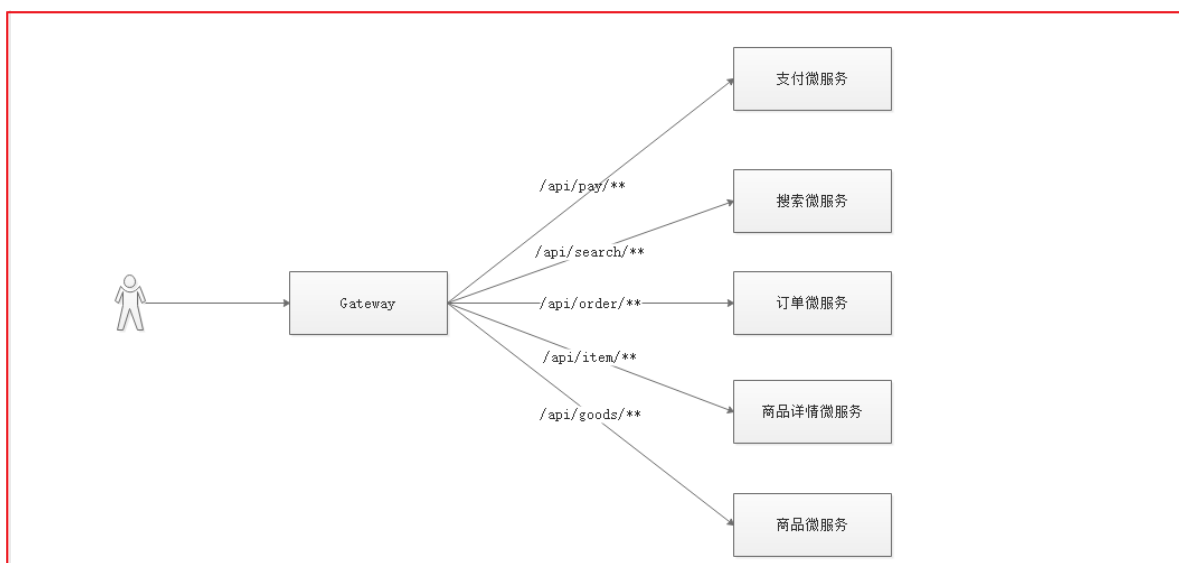
1  spring:
2      cloud:
3          gateway:
4              globalcors:
5                  cors-configurations:
6                      '["/**"]': # 匹配所有请求
7                          allowedOrigins: "*" #跨域处理 允许所有的域
8                          allowedMethods: # 支持的方法
9                              - GET
10                             - POST
11                             - PUT
12                             - DELETE

```

最终文件如下：

```
1  spring:
2    cloud:
3      gateway:
4        globalcors:
5          cors-configurations:
6            '["/**"]': # 匹配所有请求
7              allowedOrigins: "*" #跨域处理 允许所有的域
8              allowedMethods: # 支持的方法
9                - GET
10               - POST
11               - PUT
12               - DELETE
13      application:
14        name: gateway-web
15  server:
16    port: 8001
17  eureka:
18    client:
19      service-url:
20        defaultZone: http://127.0.0.1:7001/eureka
21    instance:
22      prefer-ip-address: true
23  management:
24    endpoint:
25      gateway:
26        enabled: true
27    web:
28      exposure:
29        include: true
```

### 3.4 网关过滤配置



路由过滤器允许以某种方式修改传入的HTTP请求或传出的HTTP响应。路径过滤器的范围限定为特定路径。Spring Cloud Gateway包含许多内置的GatewayFilter工厂。如上图，根据请求路径路由到不同微服务去，这块可以使用Gateway的路由过滤功能实现。

过滤器有 20 多个实现类，包括头部过滤器、路径类过滤器、Hystrix 过滤器和变更请求 URL 的过滤器，还有参数和状态码等其他类型的过滤器。

### 3.4.1 路径匹配过滤配置

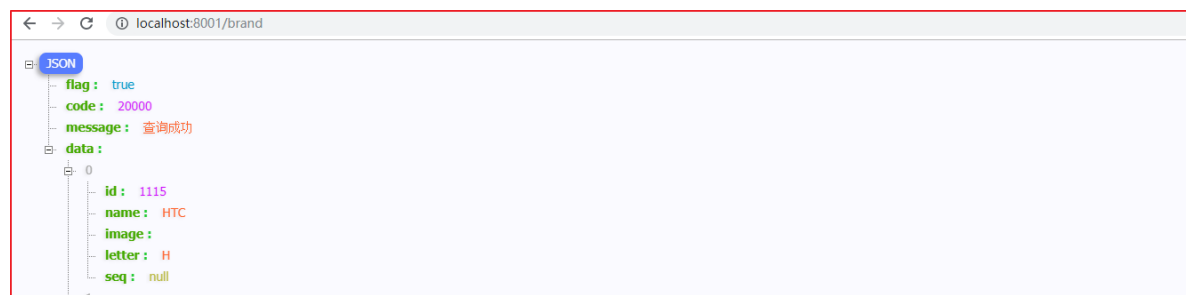
我们还可以根据请求路径实现对应的路由过滤操作，例如请求中以 `/brand/` 路径开始的请求，都直接交给 `http://localhost:180801` 服务处理，如下配置：



上图配置如下：



测试请求 `http://localhost:8001/brand`，效果如下：



### 3.4.2 PrefixPath 过滤配置

用户每次请求路径的时候，我们可以给真实请求加一个统一前缀，例如用户请求

`http://localhost:8001` 的时候我们让它请求真实地址 `http://localhost:8001/brand`，如下配置：

```

spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]': # 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: changgou_goods_route
          uri: http://localhost:18081
          predicates:
            #- Host=cloud.itheima.com**
            - Path=/**
          filters:
            - PrefixPath=/brand

```

→ 用户所有请求路径中都加一个/brand 前缀, 例如请求 http://localhost:8001-->真实请求 http://localhost:8001/brand

上图配置如下:

```

1 routes:
2   - id: changgou_goods_route
3     uri: http://localhost:18081
4     predicates:
5       - Path=/**
6     filters:
7       - PrefixPath=/brand

```

测试请求 http://localhost:8001/ 效果如下:

真实请求 http://localhost:18081/brand 的结果

```

{
  "flag": true,
  "code": 20000,
  "message": "查询成功",
  "data": {
    "id": 1115,
    "name": "HTC",
    "image": "",
    "letter": "H",
    "seq": null
  }
}

```

### 3.4.3 StripPrefix 过滤配置

很多时候也会有这么一种请求, 用户请求路径是 /api/brand, 而真实路径是 /brand, 这时候我们需要去掉 /api 才是真实路径, 此时可以使用 StripPrefix 功能来实现路径的过滤操作, 如下配置:

```

spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]': # 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: changgou_goods_route
          uri: http://localhost:18081
          predicates:
            #- Host=cloud.itheima.com**
            - Path=/**
          filters:
            #- PrefixPath=/brand
            - StripPrefix=1

```

会将请求路径中的第一个路径过滤掉，例如请求 `http://localhost:8001/api/brand` --> 真实请求 `http://localhost:8001/brand`

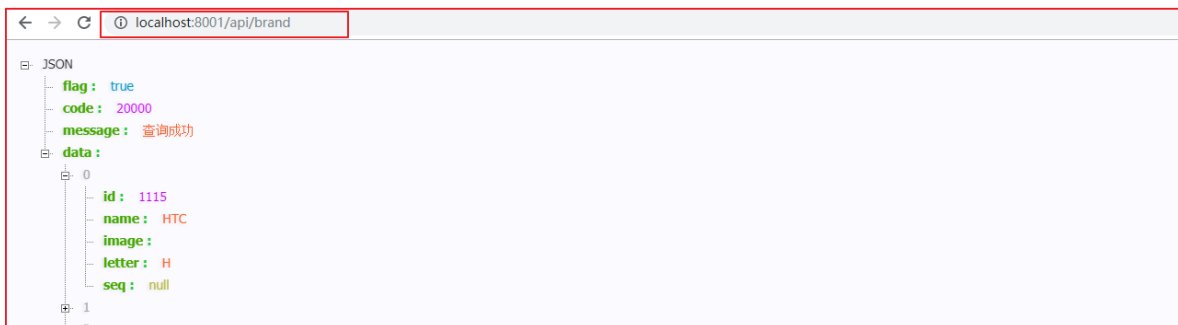
上图配置如下：

```

1      routes:
2          - id: changgou_goods_route
3            uri: http://localhost:18081
4            predicates:
5              - Path=/**
6            filters:
7              #- PrefixPath=/brand
8              - StripPrefix=1

```

测试请求 `http://localhost:8001/api/brand` ,效果如下：



```

{
  "flag": true,
  "code": 20000,
  "message": "查询成功",
  "data": [
    {
      "id": 1115,
      "name": "HTC",
      "image": "",
      "letter": "H",
      "seq": null
    }
  ]
}

```

### 3.4.4 LoadBalancerClient 路由过滤器(客户端负载均衡)

上面的路由配置每次都会将请求给指定的 URL 处理，但如果在以后生产环境，并发量较大的时候，我们需要根据服务的名称判断来做负载均衡操作，可以使用 `LoadBalancerClientFilter` 来实现负载均衡调用。`LoadBalancerClientFilter` 会作用在 url 以 lb 开头的路由，然后利用 `LoadBalancer` 来获取服务实例，构造目标 `requestUrl`，设置到 `GATEWAY_REQUEST_URL_ATTR` 属性中，供 `NettyRoutingFilter` 使用。

修改 `application.yml` 配置文件，代码如下：

```

spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]': # 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: changgou_goods_route
          #uri: http://localhost:18081
          uri: lb://goods → 所有的请求都交给goods服务处理
          predicates:
            #- Host=cloud.itheima.com**
            - Path=/**
          filters:
            #- PrefixPath=/brand
            - StripPrefix=1

```

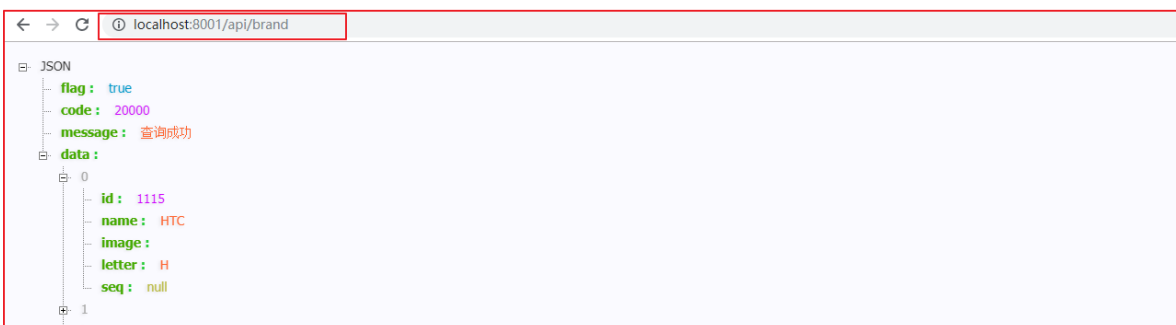
上图配置如下：

```

1      routes:
2          - id: changgou_goods_route
3            #uri: http://localhost:18081
4            uri: lb://goods
5            predicates:
6              #- Host=cloud.itheima.com**
7              - Path=/**
8            filters:
9              #- PrefixPath=/brand
10             - StripPrefix=1

```

测试请求路径 `http://localhost:8001/api/brand`



```

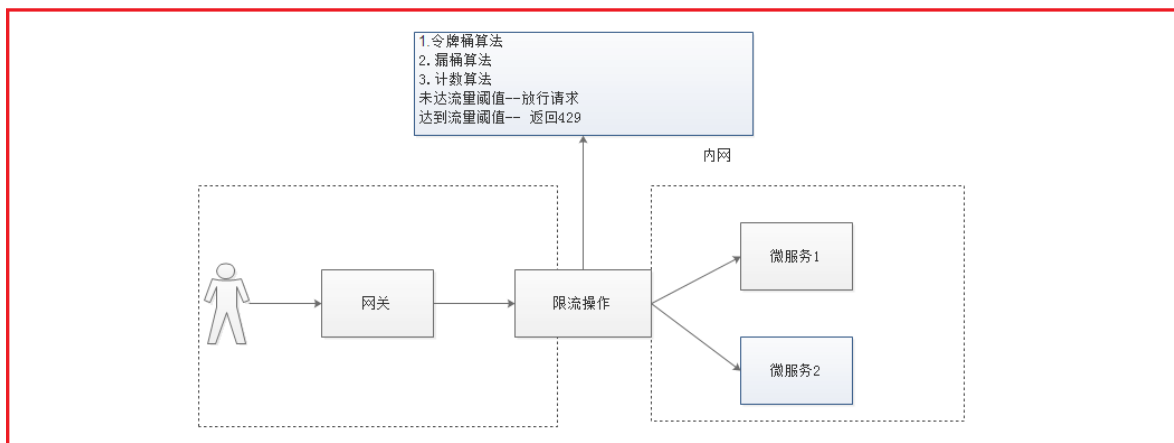
{
  "flag": true,
  "code": 20000,
  "message": "查询成功",
  "data": [
    {
      "id": 1115,
      "name": "HTC",
      "image": "",
      "letter": "H",
      "seq": null
    }
  ]
}

```

## 3.5 网关限流

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。

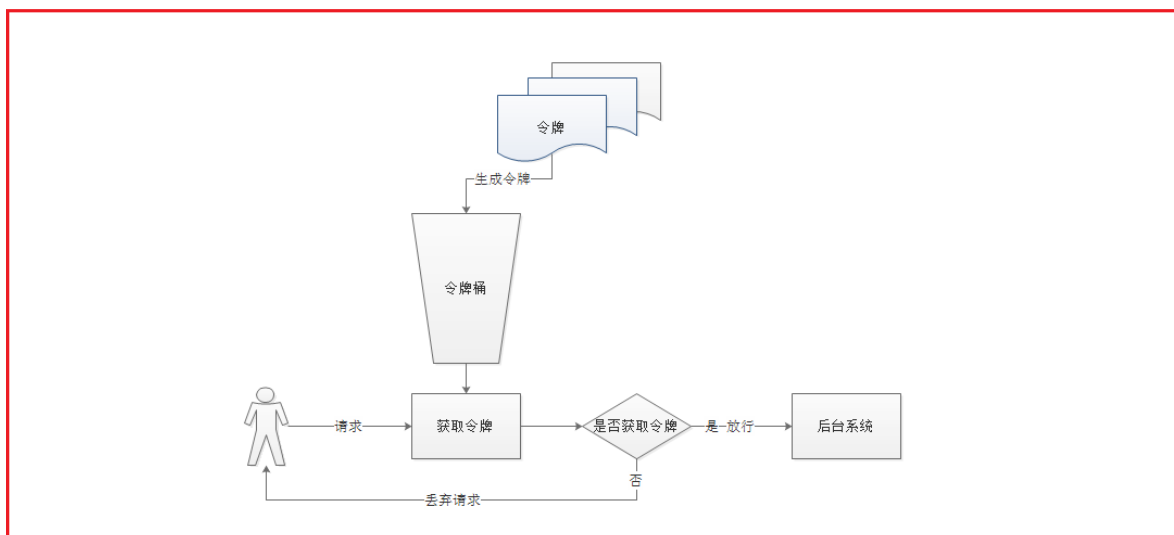
### 3.5.1 思路分析



### 3.5.2 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是其中之一，redis客户端也有其实现。

### 3.5.3 使用令牌桶进行请求次数限流

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

(1)引入redis依赖

在changgou-gateway的pom.xml中引入redis的依赖



```

1 <!--redis-->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
5     <version>2.1.3.RELEASE</version>
6 </dependency>

```

## (2)定义KeyResolver

在Application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。

我们可以根据IP来限流，比如每个IP每秒钟只能请求一次，在GatewayWebApplication定义key的获取，获取客户端IP，将IP作为key，如下代码：

```

1  /**
2   * IP限流
3   * @return
4   */
5  @Bean(name="ipKeyResolver")
6  public KeyResolver userKeyResolver() {
7      return new KeyResolver() {
8          @Override
9          public Mono<String> resolve(ServerWebExchange exchange) {
10              //获取远程客户端IP
11              String hostName =
12              exchange.getRequest().getRemoteAddress().getAddress().getHostAddress();
13              System.out.println("hostName:"+hostName);
14              return Mono.just(hostName);
15          }
16      };
17  }

```

## (3)修改application.yml中配置项，指定限制流量的配置以及REDIS的配置，如图

修改如下图：

```
spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]': # 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: changgou_goods_route
          uri: lb://goods
          predicates:
            - Path=/api/brand**
          filters:
            - StripPrefix=1
            - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的facatory
              args:
                key-resolver: "#{@ipKeyResolver}"
                redis-rate-limiter.replenishRate: 1
                redis-rate-limiter.burstCapacity: 1

  application:
    name: gateway-web

#Redis配置
redis:
  host: 192.168.211.132
  port: 6379
```

配置代码如下：

```
1  spring:
2    cloud:
3      gateway:
4        globalcors:
5          corsConfigurations:
6            '[/**]': # 匹配所有请求
7              allowedOrigins: "*" #跨域处理 允许所有的域
8              allowedMethods: # 支持的方法
9                - GET
10               - POST
11               - PUT
12               - DELETE
13
14          routes:
15            - id: changgou_goods_route
16              uri: lb://goods
17              predicates:
18                - Path=/api/brand/**
19              filters:
20                - StripPrefix=1
21                - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的
22                  facatory
23                  args:
24                    key-resolver: "#{@ipKeyResolver}"
25                    redis-rate-limiter.replenishRate: 1
26                    redis-rate-limiter.burstCapacity: 1
27
28          application:
29            name: gateway-web
30
31          #Redis配置
32          redis:
33            host: 192.168.211.132
34            port: 6379
```

```

33 server:
34     port: 8001
35 eureka:
36     client:
37         service-url:
38             defaultZone: http://127.0.0.1:7001/eureka
39     instance:
40         prefer-ip-address: true
41 management:
42     endpoint:
43         gateway:
44             enabled: true
45     web:
46         exposure:
47             include: true

```

解释:

`redis-rate-limiter.replenishRate` 是您希望允许用户每秒执行多少请求, 而不会丢弃任何请求。这是令牌桶填充的速率

`redis-rate-limiter.burstCapacity` 是指令牌桶的容量, 允许在一秒钟内完成的请求数, 将此值设置为零将阻止所有请求。

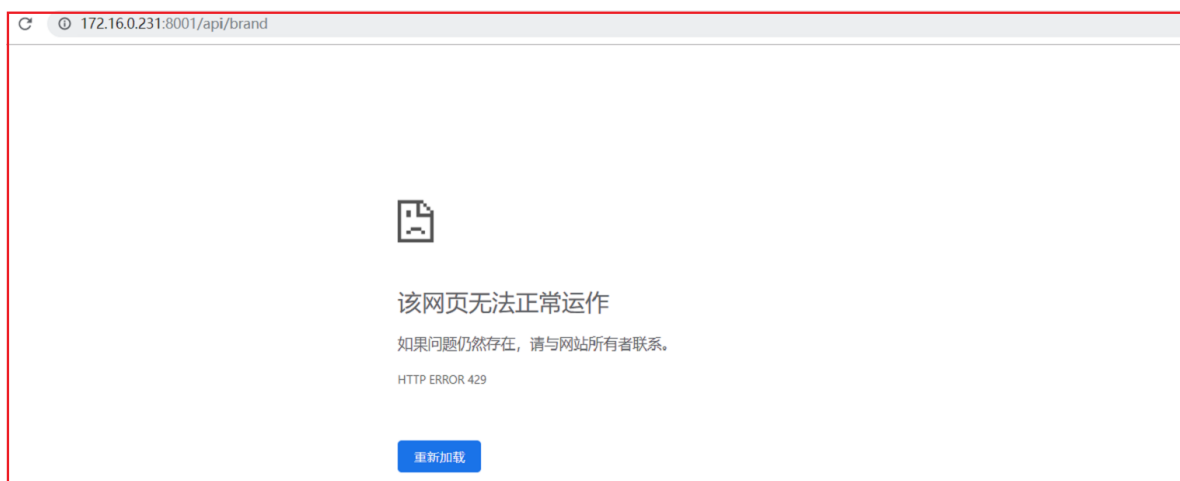
`key-resolver: "#{@ipKeyResolver}"` 用于通过SPEL表达式来指定使用哪一个KeyResolver。

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。

多次请求会发生如下情况



## 4 用户登录

项目中有2个重要角色, 分别为管理员和用户, 下面几章我们将实现购物下单和支付, 用户如果没登录是没法下单和支付的, 所以我们这里需要实现一个登录功能。

## 4.1 表结构介绍

changgou\_user表如下:

oauth_client_details		数据库: changgou_user	16 KB	InnoDB	2
tb_address	66		16 KB	InnoDB	7
tb_areas			176 KB	InnoDB	3144
tb_cities			48 KB	InnoDB	345
tb_provinces			16 KB	InnoDB	34
tb_user			16 KB	InnoDB	23
undo_log	1		16 KB	InnoDB	0

用户信息表tb\_user

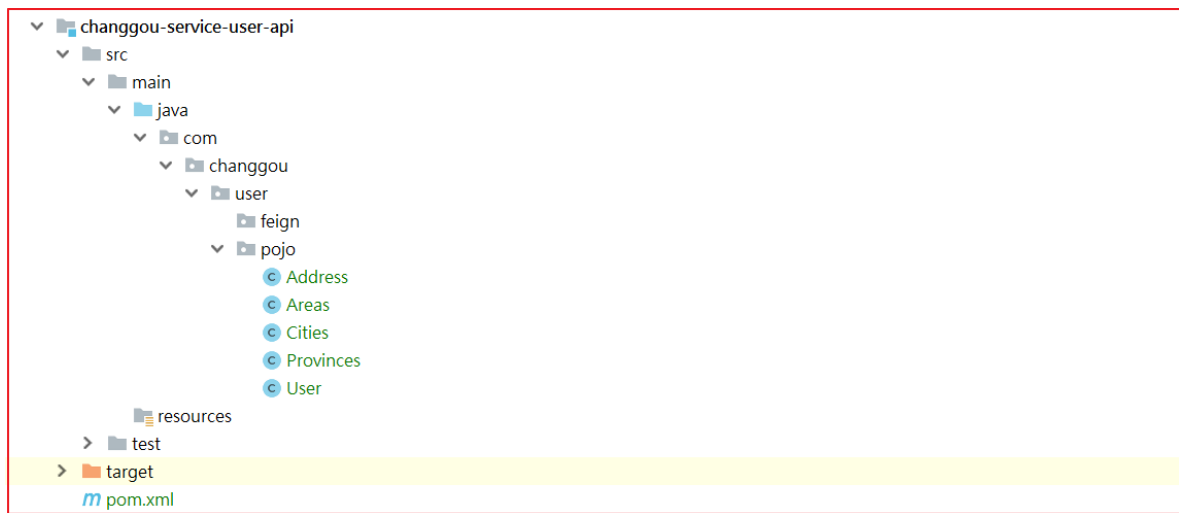
```
1 CREATE TABLE `tb_user` (  
2   `username` varchar(50) NOT NULL COMMENT '用户名',  
3   `password` varchar(100) NOT NULL COMMENT '密码, 加密存储',  
4   `phone` varchar(20) DEFAULT NULL COMMENT '注册手机号',  
5   `email` varchar(50) DEFAULT NULL COMMENT '注册邮箱',  
6   `created` datetime NOT NULL COMMENT '创建时间',  
7   `updated` datetime NOT NULL COMMENT '修改时间',  
8   `source_type` varchar(1) DEFAULT NULL COMMENT '会员来源: 1:PC, 2: H5, 3:  
Android, 4: IOS',  
9   `nick_name` varchar(50) DEFAULT NULL COMMENT '昵称',  
10  `name` varchar(50) DEFAULT NULL COMMENT '真实姓名',  
11  `status` varchar(1) DEFAULT NULL COMMENT '使用状态 (1正常 0非正常)',  
12  `head_pic` varchar(150) DEFAULT NULL COMMENT '头像地址',  
13  `qq` varchar(20) DEFAULT NULL COMMENT 'QQ号码',  
14  `is_mobile_check` varchar(1) DEFAULT '0' COMMENT '手机是否验证 (0否 1是)',  
15  `is_email_check` varchar(1) DEFAULT '0' COMMENT '邮箱是否检测 (0否 1是)',  
16  `sex` varchar(1) DEFAULT '1' COMMENT '性别, 1男, 0女',  
17  `user_level` int(11) DEFAULT NULL COMMENT '会员等级',  
18  `points` int(11) DEFAULT NULL COMMENT '积分',  
19  `experience_value` int(11) DEFAULT NULL COMMENT '经验值',  
20  `birthday` datetime DEFAULT NULL COMMENT '出生年月日',  
21  `last_login_time` datetime DEFAULT NULL COMMENT '最后登录时间',  
22  PRIMARY KEY (`username`),  
23  UNIQUE KEY `username` (`username`) USING BTREE  
24 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='用户表';
```

## 4.2 用户微服务创建

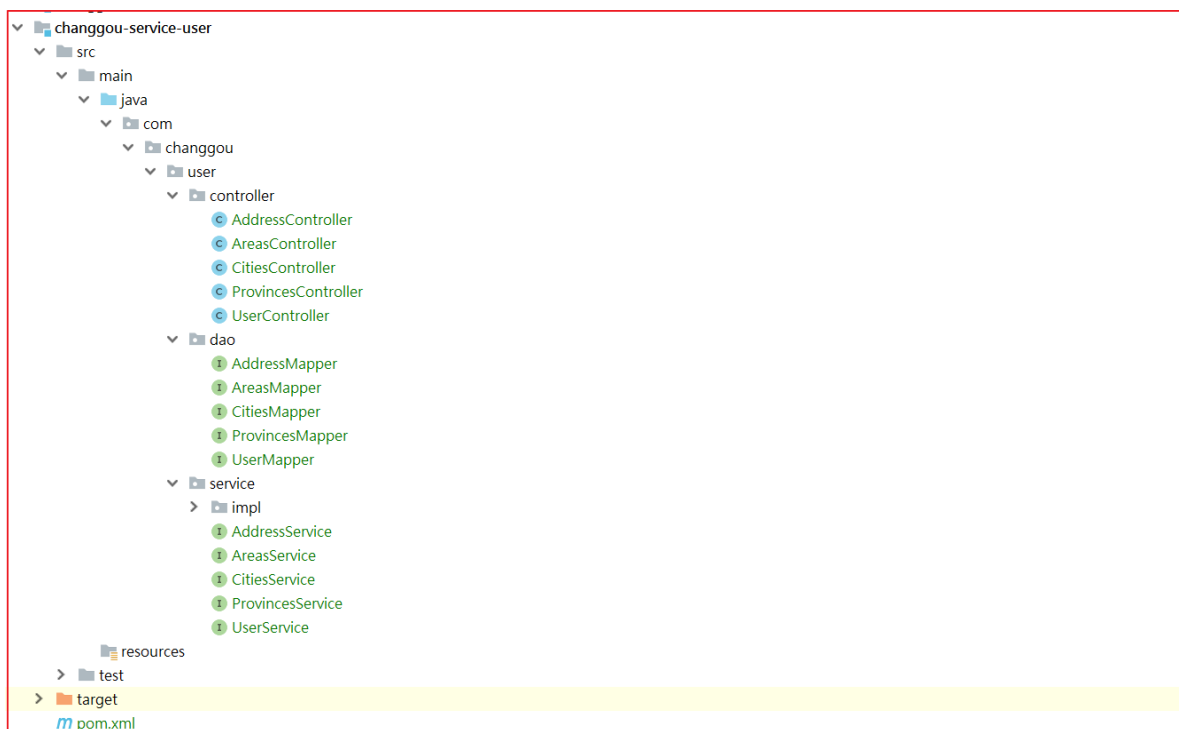
创建工程之前, 先使用代码生成器生成对应的业务代码。

(1)公共API创建

在changgou-service-api中创建changgou-service-user-api, 并将pojo拷贝到工程中, 如下图:



在changgou-service中创建changgou-service-user微服务,并引入生成的业务逻辑代码, 如下图:



## (2)依赖

在changgou-service-user的pom.xml引入如下依赖:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>changgou-service</artifactId>
8         <groupId>com.changgou</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>changgou-service-user</artifactId>
```

```

13     <!--依赖-->
14     <dependencies>
15         <dependency>
16             <groupId>com.changgou</groupId>
17             <artifactId>changgou-service-user-api</artifactId>
18             <version>1.0-SNAPSHOT</version>
19         </dependency>
20     </dependencies>
21 </project>

```

### (3)启动类创建

在changgou-service-user微服务中创建启动类com.changgou.UserApplication，代码如下：

```

1  @SpringBootApplication
2  @EnableEurekaClient
3  @MapperScan("com.changgou.user.dao")
4  public class UserApplication {
5
6      public static void main(String[] args) {
7          SpringApplication.run(UserApplication.class,args);
8      }
9  }

```

### (4)application.yml配置

在changgou-service-user的resources中创建application.yml配置，代码如下：

```

1  server:
2      port: 18088
3  spring:
4      application:
5          name: user
6      datasource:
7          driver-class-name: com.mysql.cj.jdbc.Driver
8          url: jdbc:mysql://192.168.211.132:3306/changgou_user?
          useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
9          username: root
10         password: 123456
11  eureka:
12      client:
13          service-url:
14              defaultZone: http://127.0.0.1:7001/eureka
15      instance:
16          prefer-ip-address: true

```

## 4.3 登录

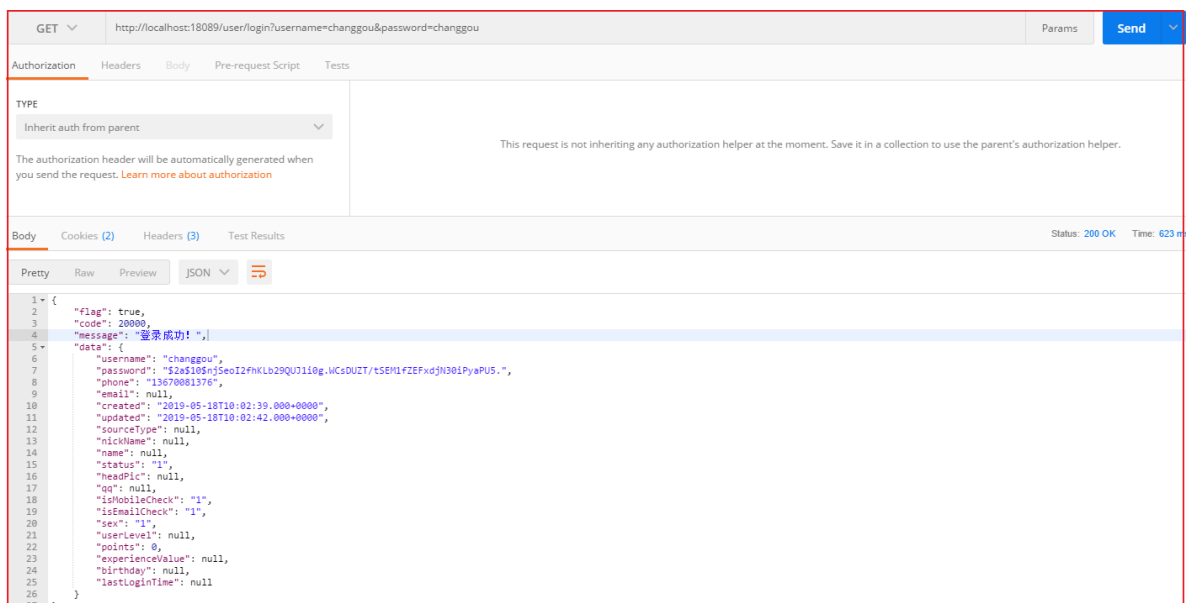
登录的时候，需要进行密码校验，这里采用了BCryptPasswordEncoder进行加密，需要将资料中的BCrypt导入到common工程中，其中BCrypt.checkpw("明文","密文")用于对比密码是否一致。

修改changgou-service-user的com.changgou.user.controller.UserController添加登录方法，代码如下：

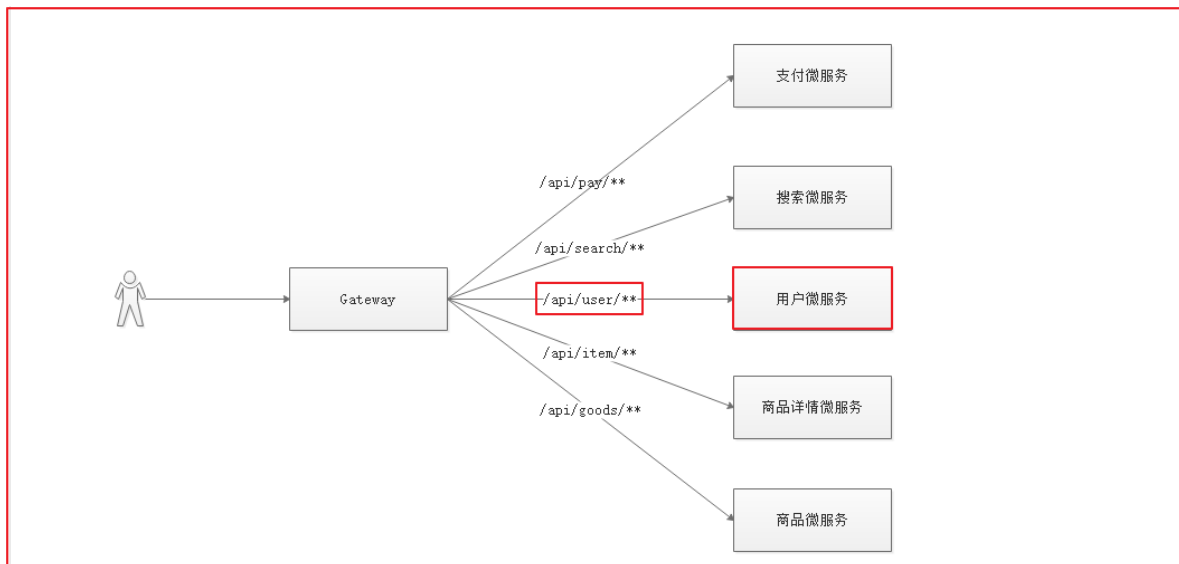
```
1  /**
2   * 用户登录
3   */
4  @RequestMapping(value = "/login")
5  public Result login(String username,String password){
6      //查询用户信息
7      User user = userService.findById(username);
8
9      if(user!=null && BCrypt.checkpw(password,user.getPassword())){
10         return new Result(true,StatusCode.OK,"登录成功!",user);
11     }
12     return new Result(false,StatusCode.LOGINERROR,"账号或者密码错误!");
13 }
```

注意：这里密码进行了加密。

使用Postman测试如下：



## 4.4 网关关联



在我们平时工作中，并不会直接将微服务暴露出去，一般都会使用网关对接，实现对微服务的一个保护作用，如上图，当用户访问 `/api/user/` 的时候我们再根据用户请求调用用户微服务的指定方法。当然，除了 `/api/user/` 还有 `/api/address/`、`/api/areas/`、`/api/cities/`、`/api/provinces/` 都需要由user微服务处理，修改网关工程 `changgou-gateway-web` 的 `application.yml` 配置文件，如下代码：

```

spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]': # 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: changgou_goods_route
          uri: lb://goods
          predicates:
            - Path=/api/goods/**
          filters:
            - StripPrefix=1
            - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的factory
              args:
                key-resolver: "#{@ipKeyResolver}"
                redis-rate-limiter.replenishRate: 1
                redis-rate-limiter.burstCapacity: 1

        #用户微服务
        - id: changgou_user_route
          uri: lb://user
          predicates:
            - Path=/api/user/**,/api/address/**,/api/areas/**,/api/cities/**,/api/provinces/**
          filters:
            - StripPrefix=1
  
```

上图代码如下：

```

1  spring:
2    cloud:
3      gateway:
4        globalcors:
5          corsConfigurations:
6            '[/**]': # 匹配所有请求
7              allowedOrigins: "*" #跨域处理 允许所有的域
8              allowedMethods: # 支持的方法
9                - GET
10               - POST
  
```

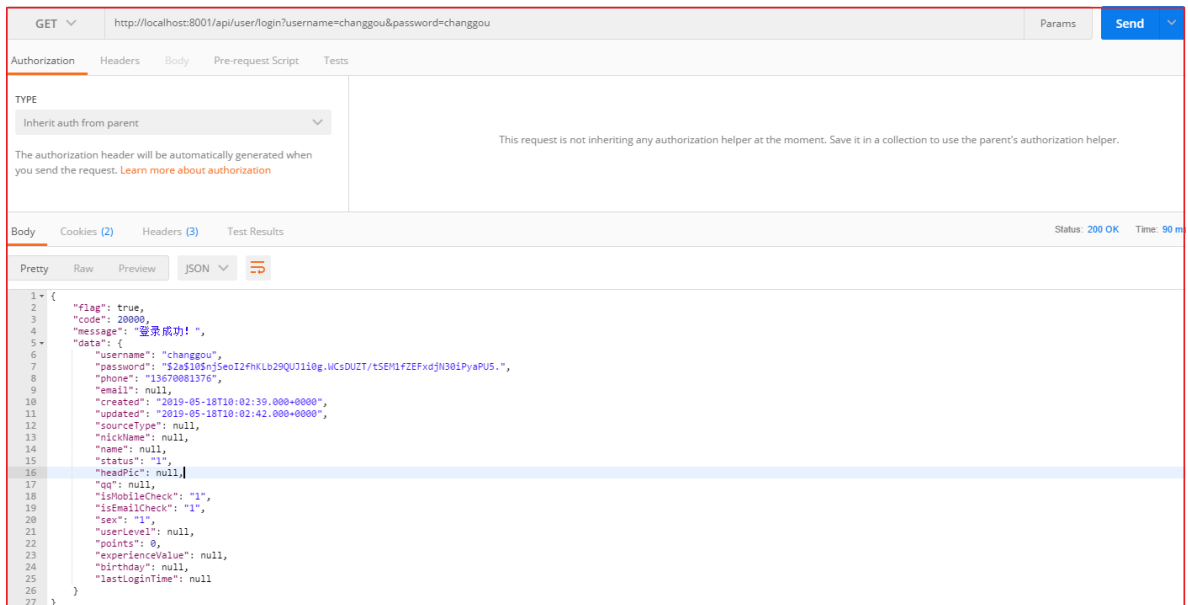


```

11         - PUT
12         - DELETE
13     routes:
14         - id: changgou_goods_route
15           uri: lb://goods
16           predicates:
17             - Path=/api/goods/**
18           filters:
19             - StripPrefix=1
20             - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的
facactory
21           args:
22             key-resolver: "#{@ipKeyResolver}"
23             redis-rate-limiter.replenishRate: 1
24             redis-rate-limiter.burstCapacity: 1
25           #用户微服务
26         - id: changgou_user_route
27           uri: lb://user
28           predicates:
29             -
Path=/api/user/**,/api/address/**,/api/areas/**,/api/cities/**,/api/province
s/**
30           filters:
31             - StripPrefix=1
32
33     application:
34       name: gateway-web
35     #Redis配置
36     redis:
37       host: 192.168.211.132
38       port: 6379
39
40     server:
41       port: 8001
42     eureka:
43       client:
44         service-url:
45           defaultZone: http://127.0.0.1:7001/eureka
46       instance:
47         prefer-ip-address: true
48     management:
49       endpoint:
50         gateway:
51           enabled: true
52       web:
53         exposure:
54           include: true

```

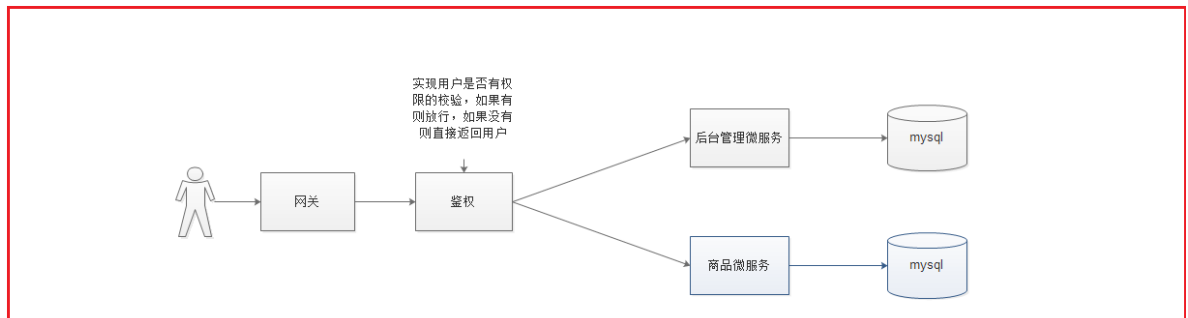
使用Postman访问 `http://localhost:8001/api/user/login?username=changgou&password=changgou`，效果如下：



## 5 JWT讲解

### 5.1 需求分析

我们之前已经搭建过了网关，使用网关在网关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 什么是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。

JWT总结：

- 1 **JWT**是用于微服务之间传递用户信息的一段加密字符串，该字符串是一个**JSON**格式，各个微服务可以根据该**JSON**字符串识别用户的身份信息，也就是该**JSON**字符串中会封装用户的身份信息。

### 5.3 JWT的构成

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
2 eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWRtaW4iOnRydWV9.  
3 TJVA95OrM7E2cBab30RMHrHDCEfXjoYZgeFONFh7HgQ
```

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。

```
1 {"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：

```
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

小知识：Base64是一种基于64个可打印字符来表示二进制数据的表示方法。由于2的6次方等于64，所以每6个比特为一个单元，对应某个可打印字符。三个字节有24个比特，对应于4个Base64单元，即3个字节需要用4个可打印字符来表示。JDK 中提供了非常方便的 **BASE64Encoder** 和 **BASE64Decoder**，用它们可以非常方便的完成基于 BASE64 的编码和解码

头部：指定了令牌类型和令牌签名的算法。

加密方式：Base64

## 载荷 (payload)

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

### (1) 标准中注册的声明（建议但不强制使用）

```
1 iss: jwt签发者  
2 sub: 当前令牌的描述说明  
3 aud: 接收jwt的一方  
4 exp: jwt的过期时间，这个过期时间必须要大于签发时间  
5 nbf: 定义在什么时间之前，该jwt都是不可用的。  
6 iat: jwt的签发时间  
7 jti: jwt的唯一身份标识，主要用来作为一次性token，从而回避重放攻击。
```

### (2) 公共的声明

公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密。

### (3) 私有的声明

私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

这个指的就是自定义的claim。比如下面面结构举例中的admin和name都属于自定的claim。这些claim跟JWT标准规定的claim区别在于：JWT规定的claim，JWT的接收方在拿到JWT之后，都知道怎么对这些标准的claim进行验证(还不知道是否能够验证)；而private claims不会验证，除非明确告诉接收方要对这些claim进行验证以及规则才行。

定义一个payload:

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

然后将其进行base64加密，得到jwt的第二部分。

```
1 | eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRTaw4iOnRydWV9
```

载荷总结：

主要包含三部分：1:标准中注册的声明

2:公共的声明 (不参与令牌校验)

3:私有声明 (不参与令牌校验)

1+2+3->Base64加密

## 签证 (signature[签名])

jwt的第三部分是一个签证信息(校验数据是否被篡改)，这个签证信息由三部分组成：

header (base64后的)

payload (base64后的)

secret(秘钥->盐)

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。

```
1 | 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:

```
1 | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
2 | eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRTaw4iOnRydWV9.  
3 | TJVA95OrM7E2cBab30RMHrHDCEfXjoYZgeFONFh7HgQ  
4 |  
5 |  
6 | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRTaw4iOnRydWV9java99
```

**注意：**secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。

签名总结：Base64(头)+Base64(载荷)+秘钥(盐)->加密:采用头中指定的算法进行加密->密文->签名。

签名的作用：用于校验令牌是否被串改





```
1 eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiI1IiwiaWF0IjE5MjUsImV4cCI6MTU2MjA2MjkynX0._vs4METaPkCza52LuN0-2NGGWII07v51xt40DHY1U1Q
```

### 5.4.3.2 解析TOKEN

```
1  /**
2   * 解析Jwt令牌数据
3   */
4  @Test
5  public void testParseJwt(){
6      String
compactJwt="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiI1IiwiaWF0IjE5MjUsImV4cCI6MTU2MjA2MjkynX0._vs4METaPkCza52LuN0-2NGGWII07v51xt40DHY1U1Q";
7      Claims claims = Jwts.parser().
8          setSigningKey("itcast").
9          parseClaimsJws(compactJwt).
10         getBody();
11      System.out.println(claims);
12  }
```

打印效果：

```
io.jsonwebtoken.ExpiredJwtException: JWT expired at 2019-07-02T18:22:05Z. Current time: 2019-07-02T18:22:30Z a difference of 25973 milliseconds. Allowed clock skew: 0 milliseconds.
    at io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:385)
    at io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:481)
    at io.jsonwebtoken.impl.DefaultJwtParser.parseClaimsJws(DefaultJwtParser.java:541)
    at com.itheima.test.JwtTest.testParseJwt(JwtTest.java:41)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
```

当前时间超过过期时间，则会报错。

### 5.4.4 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：

```

/**
 * 创建Jwt令牌
 */
@Test
public void testCreateJwt() {
    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(new Date())//过期时间设置
        .signWith(SignatureAlgorithm.HS256, "itcast");//设置签名 使用HS256算法, 并设置SecretKey(字符串)

    //自定义数据
    Map<String, Object> userInfo = new HashMap<String, Object>();
    userInfo.put("name", "王五");
    userInfo.put("age", 27);
    userInfo.put("address", "深圳黑马训练营程序员中心");
    builder.addClaims(userInfo);

    //构建 并返回一个字符串
    System.out.println( builder.compact() );
}

```

自定义数据

运行打印效果:

```

1 eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiI1IiwiaWF0IjE5ODU0MjY0Lm51LCJpYXQiOiE1NjIwNjMyOTIsImFkZHIjOiJlc3MiOiJmLT7HlnLPPu5Hpqazorq3nu4PokKXnqIvluo_lkzjkuk3lv4MiLCJuYWl1Ijo1546L5LQUIwiYwdlIjoyn30.ZSbHt5qrxz0F1Ma9rvHHAiy4jMCBGIHoNaaPQxxv_dk

```

解析TOKEN:

```

1 /**
2  * 解析Jwt令牌数据
3  */
4  @Test
5  public void testParseJwt(){
6      String
compactJwt="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiI1IiwiaWF0IjE5ODU0MjY0Lm51LCJpYXQiOiE1NjIwNjMyOTIsImFkZHIjOiJlc3MiOiJmLT7HlnLPPu5Hpqazorq3nu4PokKXnqIvluo_lkzjkuk3lv4MiLCJuYWl1Ijo1546L5LQUIwiYwdlIjoyn30.ZSbHt5qrxz0F1Ma9rvHHAiy4jMCBGIHoNaaPQxxv_dk";
7      Claims claims = Jwts.parser().
8          setSigningKey("itcast").
9          parseClaimsJws(compactJwt).
10         getBody();
11      System.out.println(claims);
12  }

```

运行效果:

```

1 test passed - 1s 29ms

"C:\Program Files\Java\jdk\bin\java" ...
{iti=888, sub=小白, iat=1562063292, address=深圳黑马训练营程序员中心, name=王五, age=27}

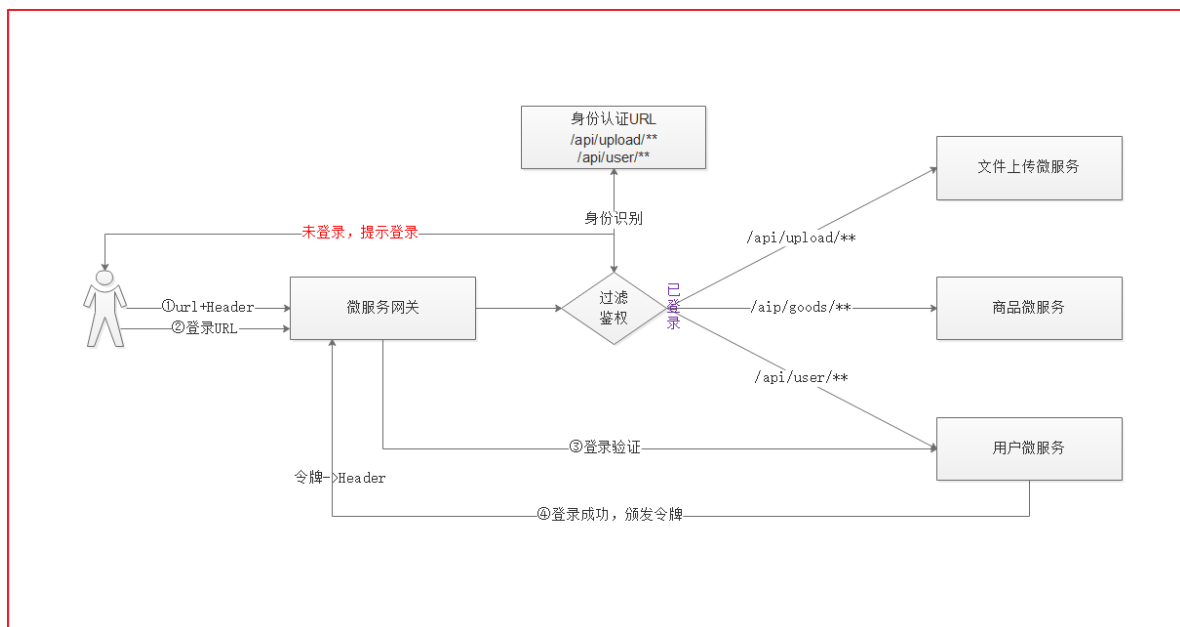
Process finished with exit code 0

```

## 5.5 鉴权处理

### 5.5.1 思路分析





- 1 1. 用户通过访问微服务网关调用微服务，同时携带头文件信息
- 2 2. 在微服务网关这里进行拦截，拦截后获取用户要访问的路径
- 3 3. 识别用户访问的路径是否需要登录，如果需要，识别用户的身份是否能访问该路径[这里可以基于数据库设计一套权限]
- 4 4. 如果需要权限访问，用户已经登录，则放行
- 5 5. 如果需要权限访问，且用户未登录，则提示用户需要登录
- 6 6. 用户通过网关访问用户微服务，进行登录验证
- 7 7. 验证通过后，用户微服务会颁发一个令牌给网关，网关会将用户信息封装到头文件中，并响应用户
- 8 8. 用户下次访问，携带头文件中的令牌信息即可识别是否登录

## 5.5.2 用户登录签发TOKEN

### (1) 生成令牌工具类

在changgou-common中创建类com.changgou.util.JwtUtil，主要辅助生成Jwt令牌信息，代码如下：

```

1 public class JwtUtil {
2
3     //有效期为
4     public static final Long JWT_TTL = 3600000L; // 60 * 60 * 1000 一个小时
5
6     //Jwt令牌信息
7     public static final String JWT_KEY = "itcast";
8
9     public static String createJWT(String id, String subject, Long
10     ttlMillis) {
11         //指定算法
12         SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
13
14         //当前系统时间
15         long nowMillis = System.currentTimeMillis();
16         //令牌签发时间
17         Date now = new Date(nowMillis);
18
19         //如果令牌有效期为null，则默认设置有效期1小时
20         if (ttlMillis == null) {
21             ttlMillis = JwtUtil.JWT_TTL;
22         }
23     }
24 }

```

```

21     }
22
23     //令牌过期时间设置
24     long expMillis = nowMillis + ttlMillis;
25     Date expDate = new Date(expMillis);
26
27     //生成密钥
28     SecretKey secretKey = generalKey();
29
30     //封装Jwt令牌信息
31     JwtBuilder builder = Jwts.builder()
32         .setId(id) //唯一的ID
33         .setSubject(subject) // 主题 可以是JSON数据
34         .setIssuer("admin") // 签发者
35         .setIssuedAt(now) // 签发时间
36         .signWith(signatureAlgorithm, secretKey) // 签名算法以及密钥
37         .setExpiration(expDate); // 设置过期时间
38     return builder.compact();
39 }
40
41 /**
42  * 生成加密 secretKey
43  * @return
44  */
45 public static SecretKey generalKey() {
46     byte[] encodedKey =
Base64.getEncoder().encode(JwtUtil.JWT_KEY.getBytes());
47     SecretKey key = new SecretKeySpec(encodedKey, 0, encodedKey.length,
"AES");
48     return key;
49 }
50
51
52 /**
53  * 解析令牌数据
54  * @param jwt
55  * @return
56  * @throws Exception
57  */
58 public static Claims parseJWT(String jwt) throws Exception {
59     SecretKey secretKey = generalKey();
60     return Jwts.parser()
61         .setSigningKey(secretKey)
62         .parseClaimsJws(jwt)
63         .getBody();
64 }
65 }

```

(2) 用户登录成功 则 签发TOKEN，修改登录的方法：

```

/**
 * 用户登录
 */
@RequestMapping(value = "/login")
public Result login(String username,String password){
    //查询用户信息
    User user = userService.findById(username);

    if(user!=null && BCrypt.checkpw(password,user.getPassword())){
        //设置令牌信息
        Map<String,Object> info = new HashMap<String,Object>();
        info.put("role","USER");
        info.put("success","SUCCESS");
        info.put("username",username);
        //生成令牌
        String jwt = JwtUtil.createJWT(UUID.randomUUID().toString(), JSON.toJSONString(info),ttlMillis: null);
        return new Result( flag: true,StatusCode.OK, message: "登录成功!", jwt);
    }
    return new Result( flag: false,StatusCode.LOGINERROR, message: "账号或者密码错误!");
}

```

代码如下:

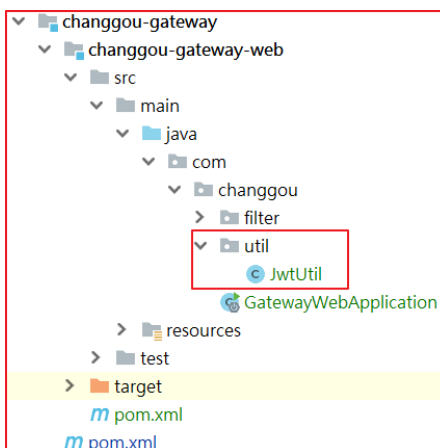
```

1  /**
2   * 用户登录
3   */
4  @RequestMapping(value = "/login")
5  public Result login(String username,String password){
6      //查询用户信息
7      User user = userService.findById(username);
8
9      if(user!=null && BCrypt.checkpw(password,user.getPassword())){
10         //设置令牌信息
11         Map<String,Object> info = new HashMap<String,Object>();
12         info.put("role","USER");
13         info.put("success","SUCCESS");
14         info.put("username",username);
15         //生成令牌
16         String jwt = JwtUtil.createJWT(UUID.randomUUID().toString(),
17         JSON.toJSONString(info),null);
18         return new Result(true,StatusCode.OK,"登录成功!",jwt);
19     }
20     return new Result(false,StatusCode.LOGINERROR,"账号或者密码错误!");
21 }

```

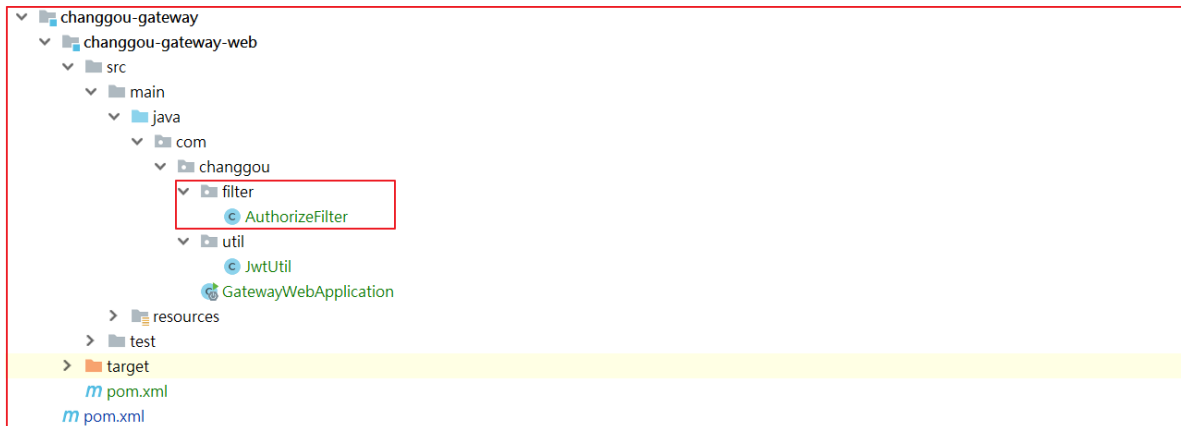
### 5.5.3 网关过滤器拦截请求处理

拷贝JwtUtil到changgou-gateway-web中



## 5.5.4 自定义全局过滤器

创建 过滤器类，如图所示：



AuthorizeFilter代码如下：

```
1 public class AuthorizeFilter implements GlobalFilter,Ordered {
2
3     //令牌头名字
4     private static final String AUTHORIZE_TOKEN = "Authorization";
5
6
7     /**
8      * 全局过滤器
9      * @param exchange
10     * @param chain
11     * @return
12     */
13     @Override
14     public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
15         //获取request和response
16         ServerHttpRequest request = exchange.getRequest();
17         ServerHttpResponse response = exchange.getResponse();
18
19         //获取uri
20         String path = request.getURI().getPath();
21
22         //判断如果是/api/user/login，则放行
23         if(path.startsWith("/api/user/login")){
24             //放行
25             return chain.filter(exchange);
26         }
27
28         //获取头文件中的令牌
29         String token = request.getHeaders().getFirst(AUTHORIZE_TOKEN);
30
31         //如果头文件中的token为空，则从请求参数获取
32         if(StringUtils.isEmpty(token)){
33             token = request.getQueryParams().getFirst(AUTHORIZE_TOKEN);
34         }
35     }
```

```

36         //如果此时参数还未空，则不允许访问
37         if(StringUtils.isEmpty(token)){
38             //401无权访问
39             response.setStatus(HttpStatus.UNAUTHORIZED);
40             //调用该方法结束访问
41             return response.setComplete();
42         }
43
44         //解析令牌
45         try {
46             Claims claims = JwtUtil.parseJWT(token);
47             System.out.println("令牌数据: "+claims.toString());
48         } catch (Exception e) {
49             //401无权访问
50             response.setStatus(HttpStatus.UNAUTHORIZED);
51             //调用该方法结束访问
52             return response.setComplete();
53         }
54         return chain.filter(exchange);
55     }
56
57     /**
58     * 排序
59     * @return
60     */
61     @Override
62     public int getOrder() {
63         return 0;
64     }
65 }

```

### 5.5.5 配置过滤规则

修改网关系统的yml文件:

```

spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/*]': # 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: changgou_goods_route
            uri: lb://goods
            predicates:
              - Path=/api/album/**,/api/brand/**,/api/cache/**,/api/categoryBrand/**,/api/category/**,/api/para/**,/api/pref/**,/api/sku/**,/api/spec/**,/api/spu/**,/api/stockBack/**,/api/template/**
            filters:
              - StripPrefix=1
              - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的facatory
                args:
                  key-resolver: "#{@ipKeyResolver}"
                  redis-rate-limiter.replenishRate: 1
                  redis-rate-limiter.burstCapacity: 1

#用户微服务
          - id: changgou_user_route
            uri: lb://user
            predicates:
              - Path=/api/user/**,/api/address/**,/api/areas/**,/api/cities/**,/api/provinces/**
            filters:
              - StripPrefix=1

application:
  name: gateway-web
#Redis配置
redis:
  host: 192.168.211.132
  port: 6379

server:
  port: 8001
ureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:7001/eureka
  instance:
    prefer-ip-address: true
management:
  endpoint:
    gateway:
      enabled: true
web:
  exposure:
    include: true

```

上述代码如下：

```

1  spring:
2    cloud:
3      gateway:
4        globalcors:
5          corsConfigurations:
6            '[/*]': # 匹配所有请求
7              allowedOrigins: "*" #跨域处理 允许所有的域
8              allowedMethods: # 支持的方法
9                - GET
10               - POST
11               - PUT
12               - DELETE
13
14         routes:
15           - id: changgou_goods_route
16             uri: lb://goods
17             predicates:
18               - Path=/api/album/**,/api/brand/**,/api/cache/**,/api/categoryBrand/**,/api/category/**,/api/para/**,/api/pref/**,/api/sku/**,/api/spec/**,/api/spu/**,/api/stockBack/**,/api/template/**
19             filters:
20               - StripPrefix=1
21               - name: RequestRateLimiter #请求数限流 名字不能随便写，使用默认的
22                 facatory
23                 args:
24                   key-resolver: "#{@ipKeyResolver}"
25                   redis-rate-limiter.replenishRate: 1
26                   redis-rate-limiter.burstCapacity: 1
27
28             #用户微服务

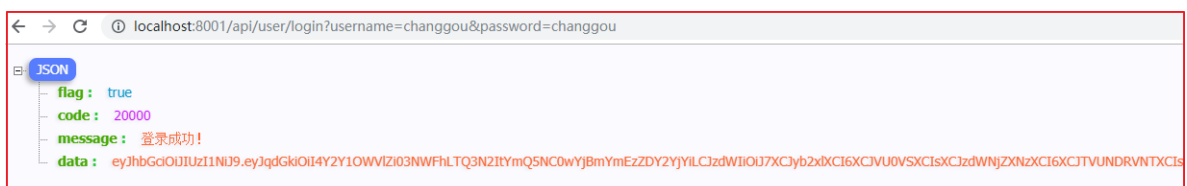
```

```

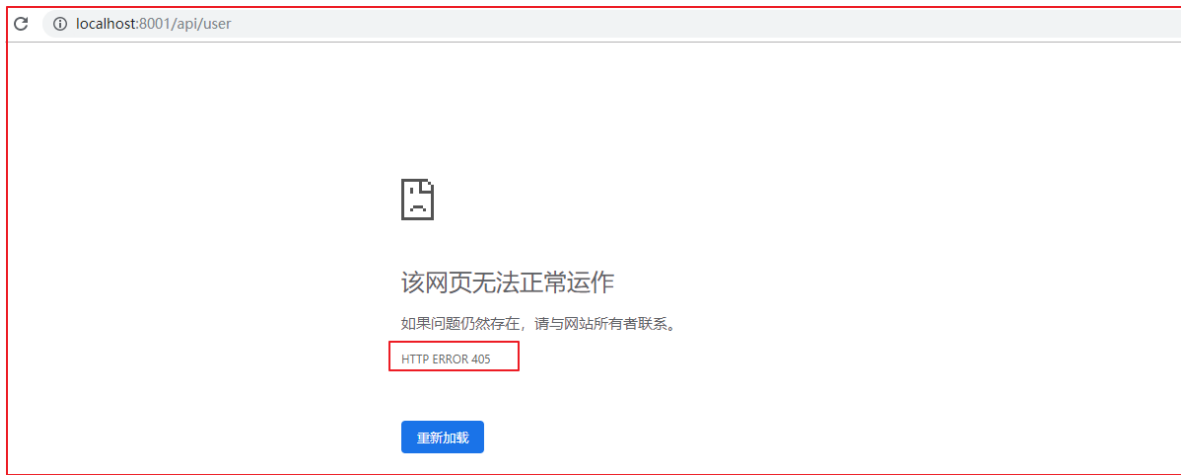
26         - id: changgou_user_route
27           uri: lb://user
28           predicates:
29             -
Path=/api/user/**,/api/address/**,/api/areas/**,/api/cities/**,/api/province
s/**
30           filters:
31             - StripPrefix=1
32
33     application:
34       name: gateway-web
35     #Redis配置
36     redis:
37       host: 192.168.211.132
38       port: 6379
39
40     server:
41       port: 8001
42     eureka:
43       client:
44         service-url:
45           defaultZone: http://127.0.0.1:7001/eureka
46       instance:
47         prefer-ip-address: true
48     management:
49       endpoint:
50         gateway:
51           enabled: true
52       web:
53         exposure:
54           include: true

```

测试访问 `http://localhost:8001/api/user/login?username=changgou&password=changgou` ,  
效果如下:



测试访问 `http://localhost:8001/api/user` , 效果如下:



参考官方手册：

[https://cloud.spring.io/spring-cloud-gateway/spring-cloud-gateway.html#\\_stripprefix\\_gatewayfilter\\_factory](https://cloud.spring.io/spring-cloud-gateway/spring-cloud-gateway.html#_stripprefix_gatewayfilter_factory)

## 课后要求:

---

- 1.搭建网关微服务,测试加前缀 减前缀
- 2.实现ip限流配置,测试一下
- 3.搭建用户微服务,实现用户的登录
- 4.生成和解析jwt令牌
- 5.网关中通过全局过滤器实现以下url和header中的token获取和验证,放行登录请求
- 6.在用户登录成功后,颁发jwt令牌
- 7.将搜索页面和详情的静态页面的域名进行配置发布
- 8.将静态页面发布到linux虚拟机的nginx中去
- 9.将搜索页面和详情页面进行对接(需要使用域名进行访问)