**畅购电商系统开发**

**第8天**

传智播客.黑马程序员.深圳

# 学习目标

* 掌握微服务网关的系统搭建
* 了解什么是微服务网关以及它的作用
* 掌握系统中心微服务的搭建
* 掌握用户密码加密存储bcrypt
* 了解JWT鉴权的介绍
* 掌握JWT的鉴权的使用
* 掌握网关使用JWT进行校验
* 掌握网关限流

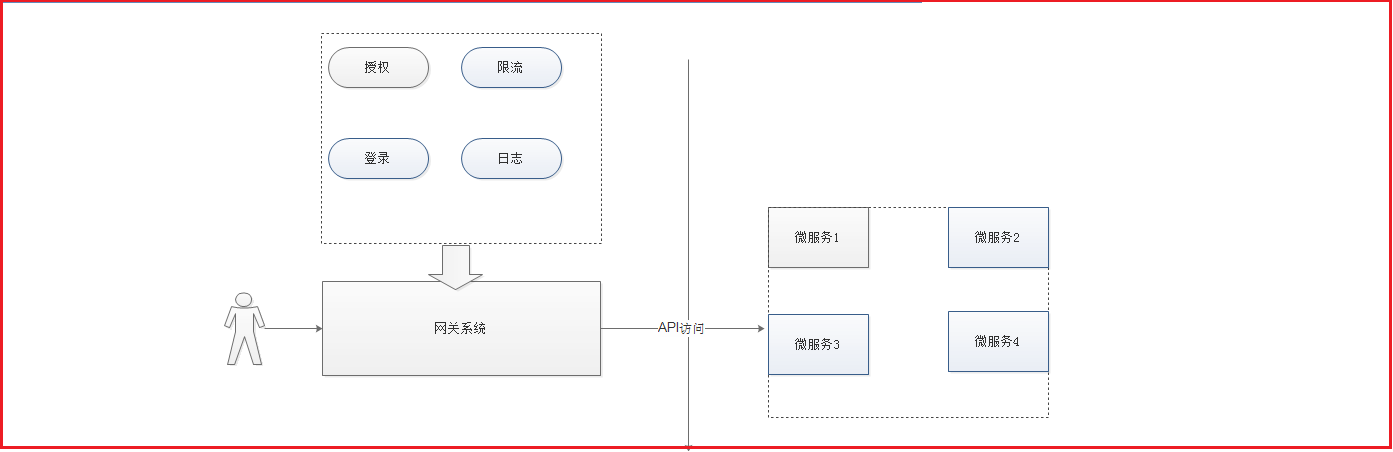
# 微服务网关

## 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，直接访问会有一定的困难
* 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。

## 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。

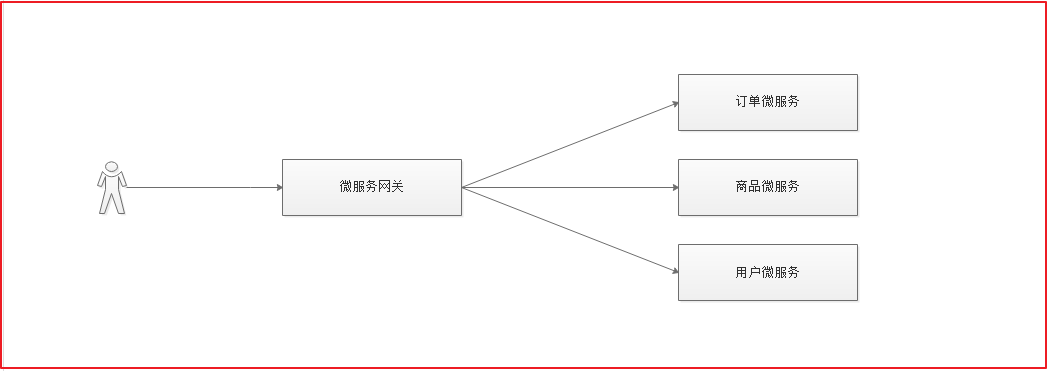
gateway官网：

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

# 网关系统使用

## 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：



## 搭建后台网关系统

### 搭建分析

由上可知道，由于 需要有多个网关，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：



### 工程搭建

#### 网关父工程

修改changgou-gateway工程，添加依赖

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_gateway</**artifactId**>  
  
 *<!--网关依赖-->* <**dependencies**>  
 <**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**>  
 </**dependencies**>  
</**project**>

#### 创建 changgou-gateway-web工程

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。

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_gateway</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_gateway\_web</**artifactId**>  
  
</**project**>

#### 启动引导类

在changgou\_gateway\_web中创建一个引导类com.changgou.GatewayWebApplication，代码如下：

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

#### application.yml配置

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

**spring**:  
 **application**:  
 **name**: gateway-web  
**server**:  
 **port**: 8001  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true**

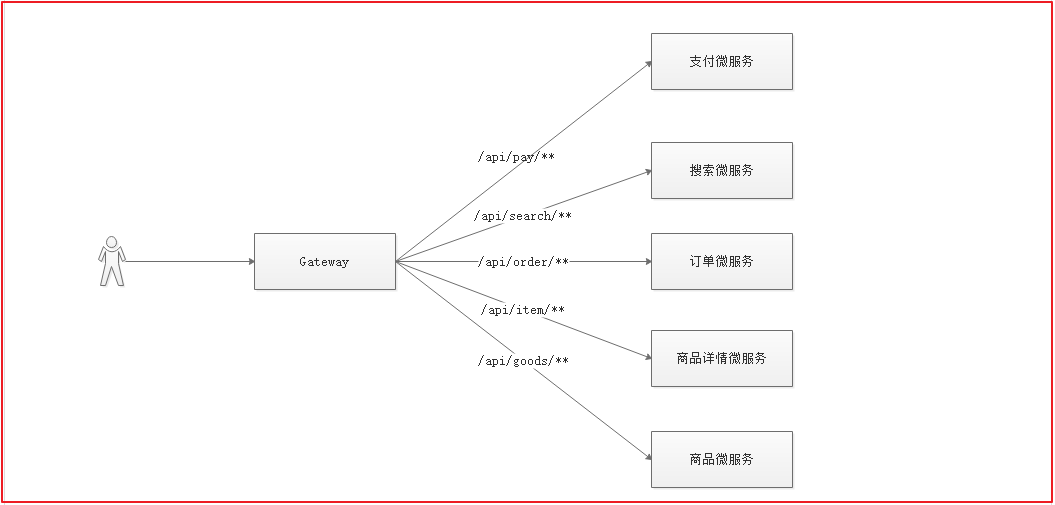
## 跨域配置

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：

**spring**:  
 **application**:  
 **name**: gateway-web  
 **cloud**:  
 **gateway**:  
 *#添加此全局跨域过滤器后，Controller中就不再需要@CrossOrigin注解* **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  
**server**:  
 **port**: 8001  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true**

## 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。

内置的过滤器工厂有22个实现类，包括 头部过滤器、路径过滤器、Hystrix 过滤器 、请求URL 变更过滤器，还有参数和状态码等其他类型的过滤器。根据过滤器工厂的用途来划分，可以分为以下几种：Header、Parameter、Path、Body、Status、Session、Redirect、Retry、RateLimiter和Hystrix。

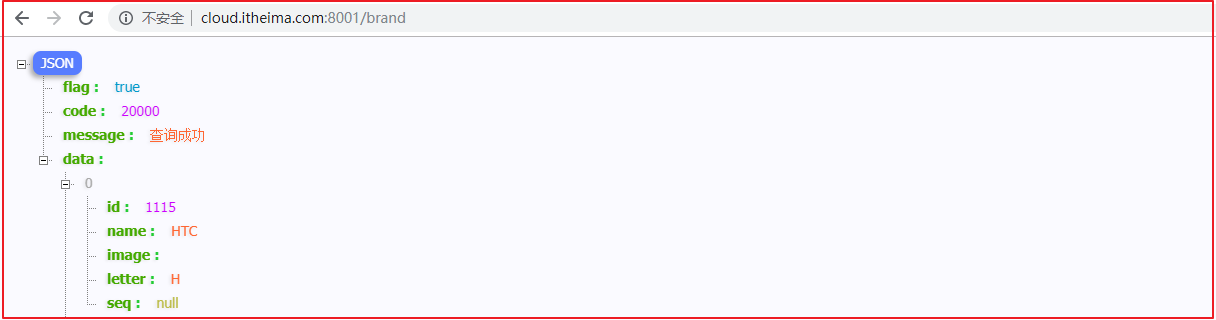
### Host 路由

换掉的是域名 127.0.0.1:8080

比如用户请求cloud.itheima.com的时候，可以将请求路由给http://localhost:18081服务处理，如下配置：

**spring**:  
 **application**:  
 **name**: gateway-web  
 **cloud**:  
 **gateway**:  
 *#添加此全局跨域过滤器后，Controller中就不再需要@CrossOrigin注解* **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**: changgou\_goods\_route *#路由唯一标识* **uri**: http://localhost:18081 *#路由转发地址  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求* - Host=cloud.itheima.com\*\*

测试请求<http://cloud.itheima.com:8001/brand> ,效果如下：



注意：此时要想让cloud.itheima.com访问本地计算机，要配置C:\Windows\System32\drivers\etc\hosts文件,映射配置如下：

127.0.0.1 cloud.itheima.**com**

### 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:18081服务处理，如下配置：

**routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识* **uri**: http://localhost:18081 *#路由转发地址  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求* - Path=/brand/\*\*

测试请求http://localhost:8001/brand,效果如下：



### 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：

**routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识* **uri**: http://localhost:18081 *#路由转发地址  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求  
 #- Path=/brand/\*\** - Path=/\*\*  
 **filters**:  
 *# 自动添加请求前缀* - PrefixPath=/brand

测试请求http://localhost:8001/效果如下：



### 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SttripPrefix功能来实现路径的过滤操作，如下配置：

**routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识* **uri**: http://localhost:18081 *#路由转发地址  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求  
 #- Path=/brand/\*\** - Path=/\*\*  
 **filters**:  
 *#- PrefixPath=/brand # 自动添加请求前缀* - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径*

### LoadBalancerClient 路由过滤器(客户端负载均衡)-动态路由

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：

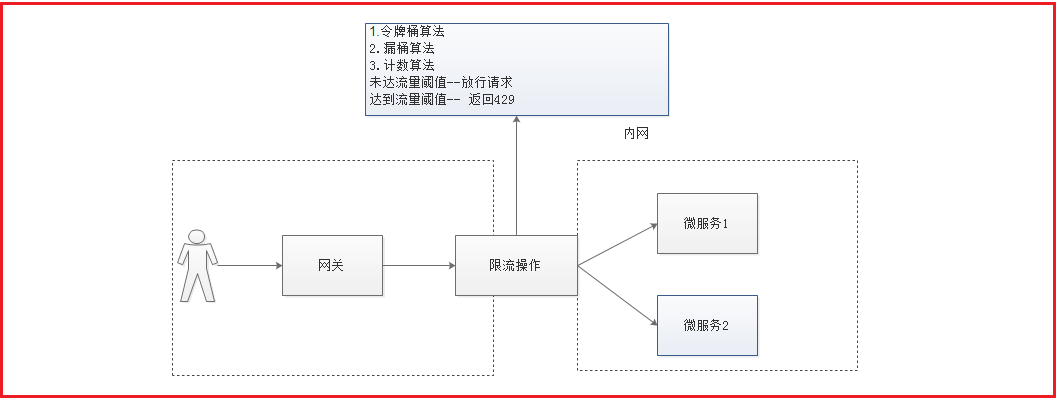
**routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识  
 #uri: http://localhost:18081 #路由转发地址* **uri**: lb://goods *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求  
 #- Path=/brand/\*\** - Path=/\*\*  
 **filters**:  
 *#- PrefixPath=/brand # 自动添加请求前缀* - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径*

测试请求路径http://localhost:8001/api/brand



## 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。



### 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；

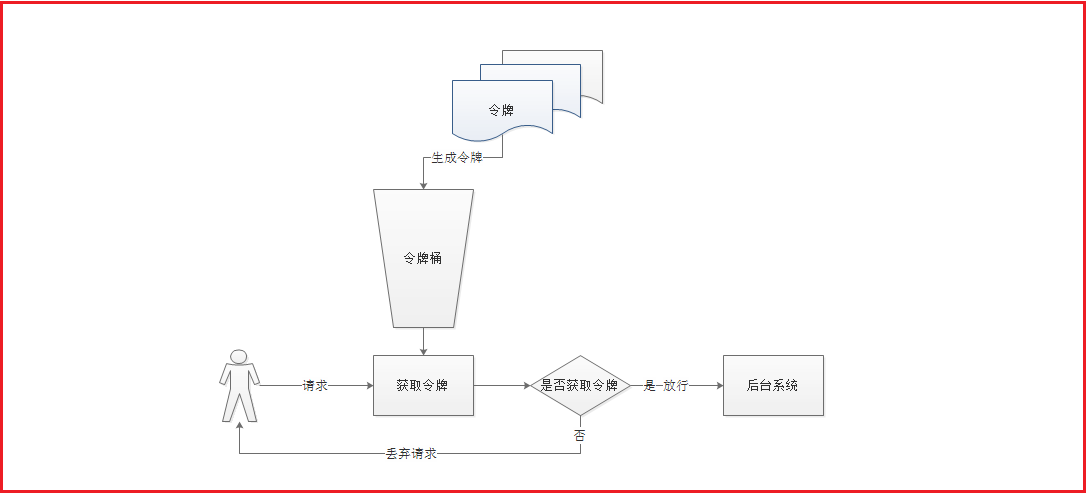
2）根据限流大小，设置按照一定的速率往桶里添加令牌；

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，Guaua是其中之一，redis客户端也有其实现。

### 使用令牌桶进行请求次数限流

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

#### 父工程加依赖

在changgou-gateway的pom.xml中引入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**>

#### 定义KeyResolver

在Applicatioin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：

@SpringBootApplication  
@EnableEurekaClient  
**public class** GatewayWebApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(GatewayWebApplication.**class**,args);  
 }  
 @Bean(name = **"ipKeyResolver"**)  
 **public** KeyResolver getKeyResolver(){  
 **return new** KeyResolver() {  
 @Override  
 **public** Mono<String> resolve(ServerWebExchange exchange) {  
 *//获取请求客户端ip* String ip = exchange.getRequest().getRemoteAddress().getHostString();  
 *//以ip限流* **return** Mono.*just*(ip);  
 }  
 };  
 }  
}

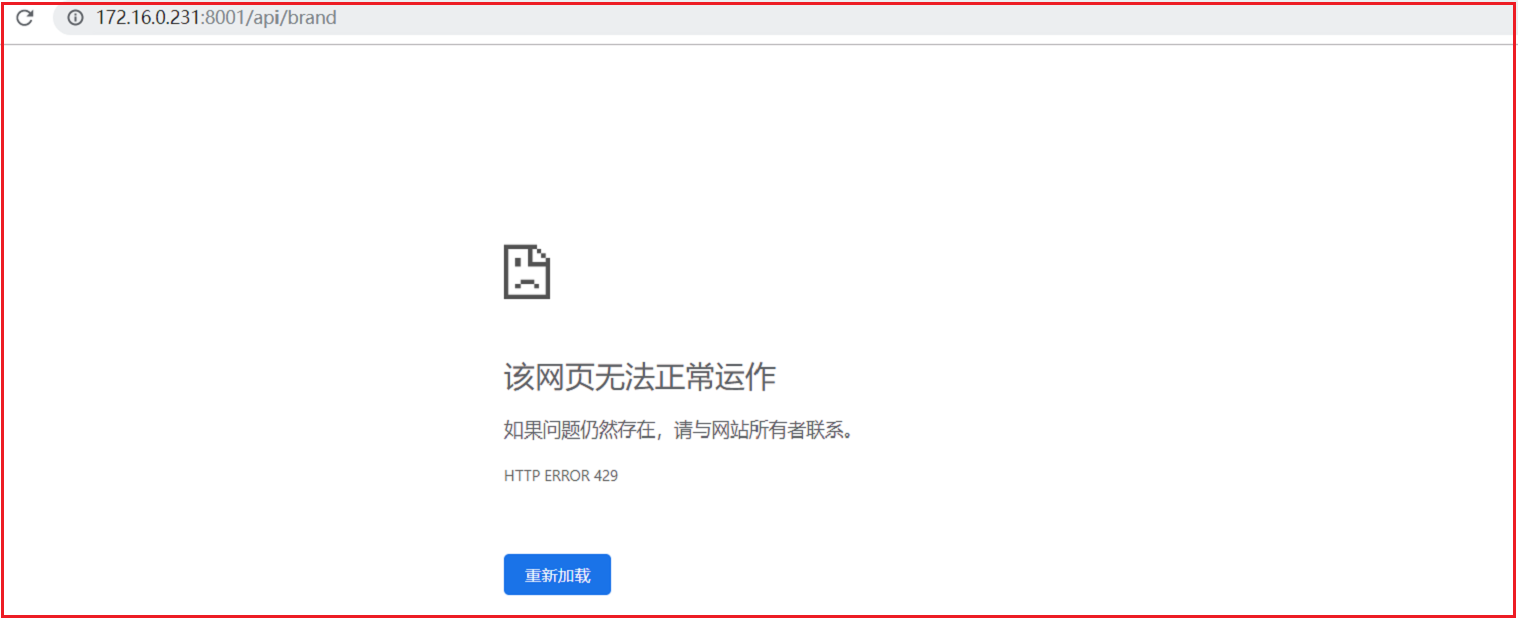
#### 修改application.yml

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

**routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识  
 #uri: http://localhost:18081 #路由转发地址* **uri**: lb://goods *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求  
 #- Path=/brand/\*\** - Path=/\*\*  
 **filters**:  
 *#- PrefixPath=/brand # 自动添加请求前缀* - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径* - **name**: RequestRateLimiter *#请求数限流 名字不能随便写 ，使用默认的facatory* **args**:  
 *# 用于通过SPEL表达式来指定使用哪一个KeyResolver，需要先在启动引导类中定义* **key-resolver**: **"#{@ipKeyResolver}"** *# 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.replenishRate**: 1  
 *# 是指令牌桶的容量，允许在一秒钟内完成的最大请求数,将此值设置为零将阻止所有请求。* **redis-rate-limiter.burstCapacity**: 2

多次请求会发生如下情况

注意请求时要连接redis，所以大家要么准备一个本地redis，要么修改yml连容器准备好的Redis



# 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。

## 表结构介绍

changgou\_user表如下：



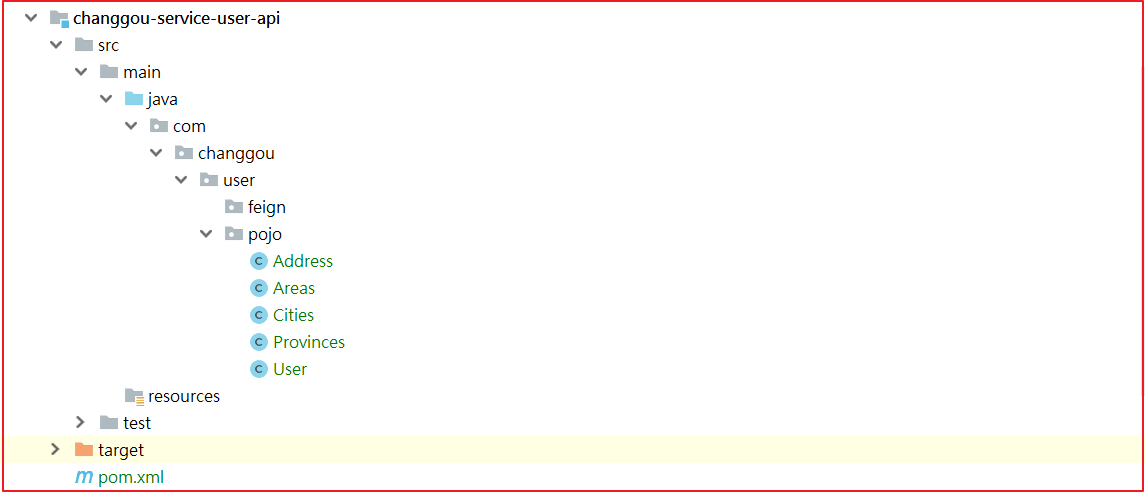
## 用户微服务创建

创建工程之前，先使用代码生成器生成对应的业务代码。

### 公共API创建

#### 创建changgou\_service\_user\_api

在changgou-service-api中创建changgou\_service\_user\_api，并将pojo拷贝到工程中，如下图：



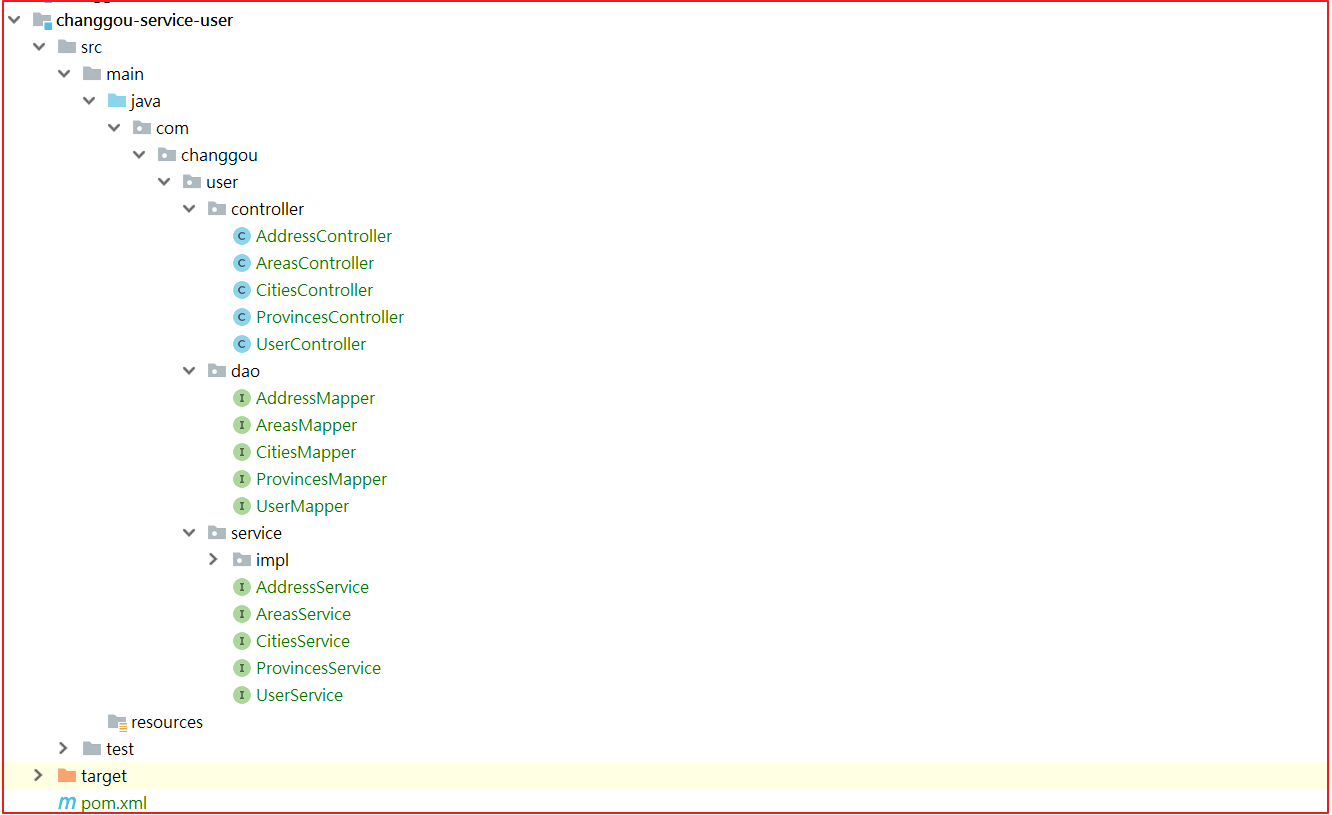
**添加依赖，参考goods\_api**

#### 创建changgou\_service\_user

在changgou-service中创建changgou\_service\_user微服务，加入依赖

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_service</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**artifactId**>changgou\_service\_user</**artifactId**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_user\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

并引入生成的业务逻辑代码，如下图：



#### 启动引导类创建

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

@SpringBootApplication  
@EnableEurekaClient  
@MapperScan(**"com.changgou.user.dao"**)  
**public class** UserApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(UserApplication.**class**,args);  
 }  
}

#### application.yml配置

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

**server**:  
 **port**: 18088  
**spring**:  
 **application**:  
 **name**: user  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_user?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true**

## 登录

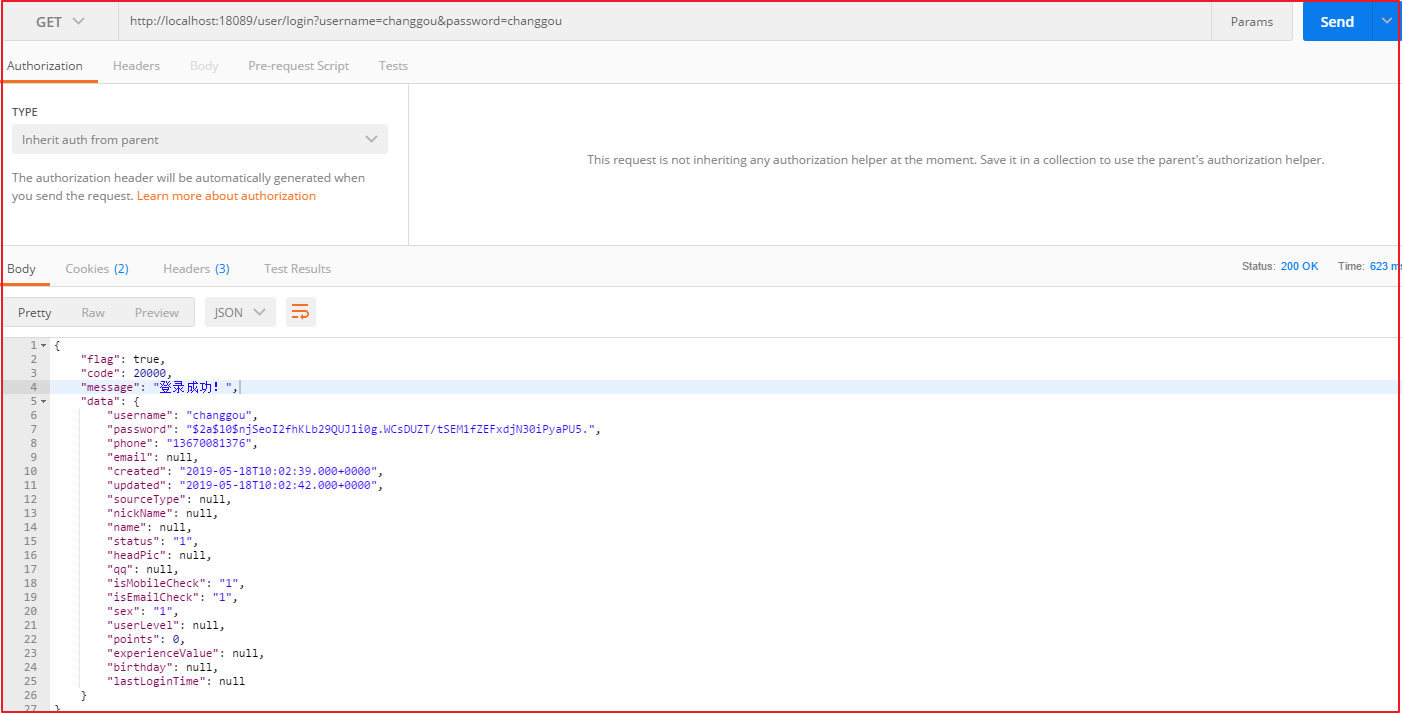
登录的时候，需要进行密码校验，这里采用了BCryptPasswordEncoder进行加密，需要将资料中的BCrypt导入到common工程中，其中BCrypt.checkpw(“明文”,“密文”)用于对比密码是否一致。

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

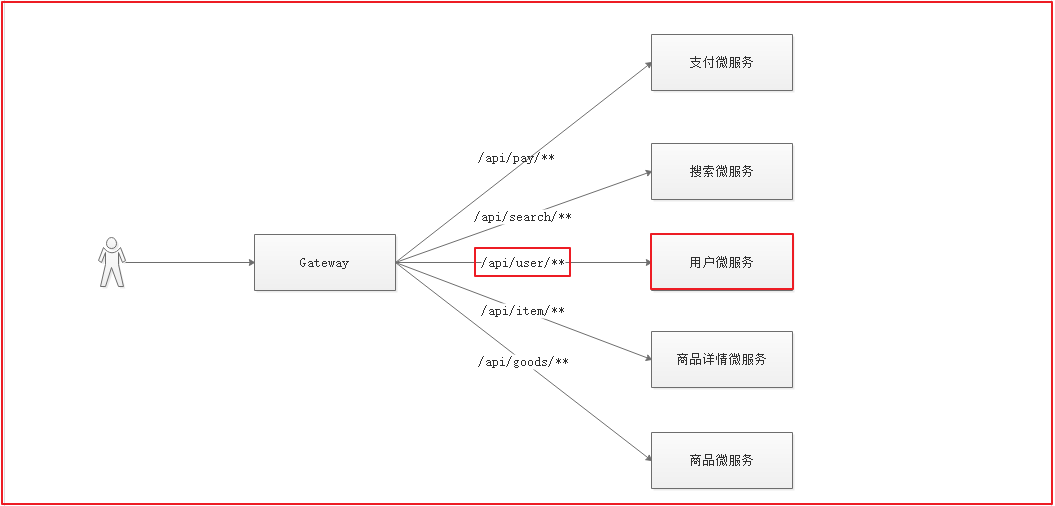
*/\*\*  
 \* 用户登录  
 \** ***@return*** *\*/*@RequestMapping(**"login"**)  
**public** Result login(String username,String password){  
 User user = **userService**.findById(username);  
 **if**(user == **null**){  
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"用户名不存在！"**);  
 }**else**{  
 *//验证加密的密码-checkpw(明文，密文)* **if**(BCrypt.*checkpw*(password,user.getPassword())){  
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"登录成功！"**,user);  
 }**else**{  
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"输入的密码错误！"**);  
 }  
 }  
}

注意：这里密码进行了加密。

使用Postman测试如下：



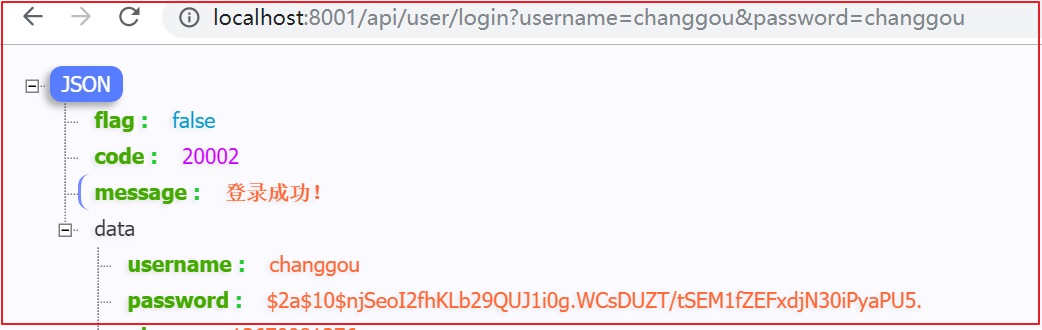
## 网关关联



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：

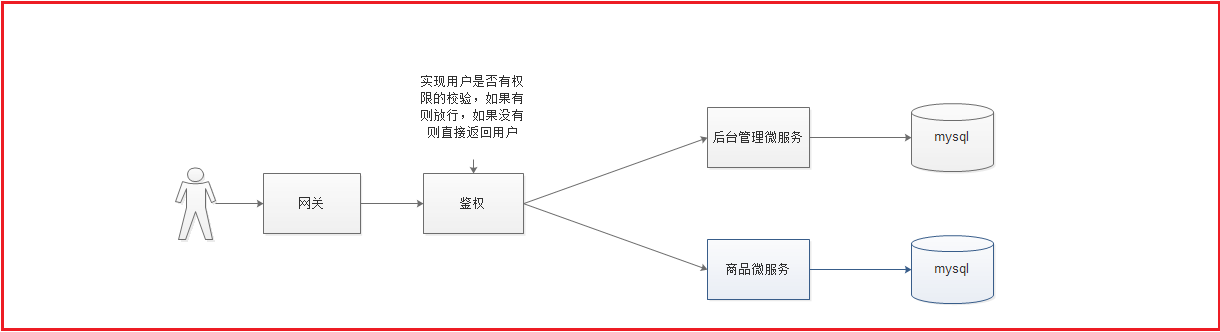
**spring**:  
 **application**:  
 **name**: gateway-web  
 **cloud**:  
 **gateway**:  
 *#添加此全局跨域过滤器后，Controller中就不再需要@CrossOrigin注解* **globalcors**:  
 **cors-configurations**:  
 **'[/\*\*]'**: *# 匹配所有请求* **allowedOrigins**: **"\*"** *#跨域处理 允许所有的域* **allowedMethods**: *# 支持的方法* - GET  
 - POST  
 - PUT  
 - DELETE  
 **routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识  
 #uri: http://localhost:18081 #路由转发地址* **uri**: lb://goods *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 # 拦截路径为/brand的请求* - Path=/brand/\*\*  
 **filters**:  
 *#- PrefixPath=/brand # 自动添加请求前缀* - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径* - **name**: RequestRateLimiter *#请求数限流 名字不能随便写 ，使用默认的facatory* **args**:  
 *# 用于通过SPEL表达式来指定使用哪一个KeyResolver，需要先在启动引导类中定义* **key-resolver**: **"#{@ipKeyResolver}"** *# 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.replenishRate**: 1  
 *# 是指令牌桶的容量，允许在一秒钟内完成的最大请求数,将此值设置为零将阻止所有请求。* **redis-rate-limiter.burstCapacity**: 2  
  
 - **id**: changgou\_user\_route *#路由唯一标识-user* **uri**: lb://user *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 - Path=/api/user/\*\*,/api/address/\*\*,/api/areas/\*\*,/api/cities/\*\*,/api/provinces/\*\*  
 **filters**:  
 - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径* **redis**:  
 **host**: 192.168.211.132  
**server**:  
 **port**: 8001  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true**

访问<http://localhost:8001/api/user/login?username=changgou&password=changgou> ，效果如下：



# JWT讲解

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。

## 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可以保证安全传输的前提下传送一些基本的信息，以减轻对外部存储的依赖，减少了分布式组件的依赖，减少了硬件的资源。

可实现无状态、分布式的Web应用授权，jwt的安全特性保证了token的不可伪造和不可篡改。

本质上是一个独立的身份验证令牌，可以包含用户标识、用户角色和权限等信息，以及您可以存储任何其他信息（自包含）。任何人都可以轻松读取和解析，并使用密钥来验证真实性。

## JSON Web Token工作原理

1. 初次登录：用户初次登录，输入用户名密码
2. 密码验证：服务器从数据库取出用户名和密码进行验证
3. 生成JWT：服务器端验证通过，根据从数据库返回的信息，以及预设规则，生成JWT
4. 返还JWT：服务器的将token放在(cookie、请求头、请求参数)中将JWT返还
5. 带JWT的请求：以后客户端发起请求，带上(cookie、请求头、请求参数)中的token信息。

## JWT的构成

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。

A.B.C

A由JWT头部信息header加密得到

B由JWT用到的身份验证信息json数据加密得到

C由A和B加密得到，是校验部分

**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alg描述的是签名算法。默认值是HS256。

将header用base64加密，得到A。<http://www.ofmonkey.com/encrypt/base64>，编码后的字符串如下：

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（就是只需要6比特就表示Base64对应的字符,然后还剩2比特）

**载荷（playload）**

**载荷就是存放有效信息的内容。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）

**iss**: jwt签发者

**sub**: jwt所面向的用户

**aud**: 接收jwt的一方

**exp**: jwt的过期时间，这个过期时间必须要大于签发时间

**nbf**: 定义在什么时间之前，该jwt都是不可用的.

**iat**: jwt的签发时间

**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:

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

然后将其进行base64加密，得到Jwt的第二部分。

eyJzdWIiOiIxMj**M0**NTY3ODkwIiwibmFtZSI6Ikpva**G4**gR**G9**lIiwiYWRtaW4iOnRydWV9

**签证（signature）**

**签名的目的是用来验证头部和载荷是否被非法篡改。**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。

TJVA95**OrM7**E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

将这三部分用.连接成一个完整的字符串,构成了最终的jwt:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX**VCJ9**.eyJzdWIiOiIxMj**M0**NTY3ODkwIiwibmFtZSI6Ikpva**G4**gR**G9**lIiwiYWRtaW4iOnRydWV9.TJVA95**OrM7**E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

**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。

## JJWT的介绍和使用

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>

### 创建TOKEN

#### 依赖引入

在changgou-parent项目中的pom.xml中添加依赖：

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

#### 创建测试

在changgou-common的/test/java下创建测试类，并设置测试方法

**Api步骤**

//1、创建Jwt构建器-jwtBuilder = Jwts.builder()

//2、设置唯一编号-setId

//3、设置主题，可以是JSON数据-setSubject()

//4、设置签发日期-setIssuedAt

//5、设置签发人-setIssuer

//6、设置签证

//7、生成令牌-compact()

//8、输出结果

**实现**

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.test  
 \*/***public class** JwtTest {  
 */\*\*  
 \* 创建令牌  
 \*/* @Test  
 **public void** testCreateJwt(){  
 *//1、创建Jwt构建器-jwtBuilder = Jwts.builder()* JwtBuilder jwtBuilder = Jwts.*builder*();  
 *//2、设置唯一编号-setId* jwtBuilder.setId(**"007"**);  
 *//3、设置主题，可以是JSON数据-setSubject()* jwtBuilder.setSubject(**"测试主题"**);  
 *//4、设置签发日期-setIssuedAt* jwtBuilder.setIssuedAt(**new** Date());  
 *//5、设置签发人-setIssuer* jwtBuilder.setIssuer(**"www.itheima.com"**);  
 *//6、设置签证* jwtBuilder.signWith(SignatureAlgorithm.***HS256***, **"itheima.steven"**);  
 *//7、生成令牌-compact()* String token = jwtBuilder.compact();  
 *//8、输出结果* System.***out***.println(token);  
 }  
}

运行打印结果：

eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIwMDciLCJzdWIiOiLmtYvor5XkuLvpopgiLCJpYXQiOjE1NjgxNzAwMjgsImlzcyI6Ind3dy5pdGhlaW1hLmNvbSJ9.yIO3b9WjmBXcvmQYRHCTgpknuI-QX8Y5bhXD\_vAmss4

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。

### TOKEN解析

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。

*/\*\*  
 \* 解析令牌  
 \*/*@Test  
**public void** testParseJwt(){  
 String token = **"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIwMDciLCJzdWIiOiLmtYvor5XkuLvpopgiLCJpYXQiOjE1NjgxNzAwMjgsImlzcyI6Ind3dy5pdGhlaW1hLmNvbSJ9.yIO3b9WjmBXcvmQYRHCTgpknuI-QX8Y5bhXD\_vAmss4"**;  
 *//1、创建Jwt解析器-jwtParser = Jwts.parser();* JwtParser jwtParser = Jwts.*parser*();  
 *//2、设置签名-密钥* jwtParser.setSigningKey(**"itheima.steven"**);  
 *//3、设置要解析的密文，并读取结果* Claims claims = jwtParser.parseClaimsJws(token).getBody();  
 *//4、输出结果* System.***out***.println(claims);  
}

运行打印效果：

{jti=007, sub=测试主题, iat=1568170028, iss=www.itheima.com}

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。

### 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。

*/\*\*  
 \* 创建令牌  
 \*/*@Test  
**public void** testCreateJwt() {  
 *//1、创建Jwt构建器-jwtBuilder = Jwts.builder()* JwtBuilder jwtBuilder = Jwts.*builder*();  
 *//2、设置唯一编号-setId* jwtBuilder.setId(**"007"**);  
 *//3、设置主题，可以是JSON数据-setSubject()* jwtBuilder.setSubject(**"测试主题"**);  
 *//4、设置签发日期-setIssuedAt* jwtBuilder.setIssuedAt(**new** Date());  
 *//5、设置签发人-setIssuer* jwtBuilder.setIssuer(**"www.itheima.com"**);  
  
 *//设置过期时间-30秒后* Date exp = **new** Date(System.*currentTimeMillis*() + 30000);  
 jwtBuilder.setExpiration(exp);  
  
 *//6、设置签证* jwtBuilder.signWith(SignatureAlgorithm.***HS256***, **"itheima.steven"**);  
 *//7、生成令牌-compact()* String token = jwtBuilder.compact();  
 *//8、输出结果* System.***out***.println(token);  
}

运行，测试时，如果在30秒内不会报错，超过了就会报错。

### 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。

修改创建令牌的方法：

@Test  
**public void** testCreateJwt() {  
 *//1、创建Jwt构建器-jwtBuilder = Jwts.builder()* JwtBuilder jwtBuilder = Jwts.*builder*();  
 *//2、设置唯一编号-setId* jwtBuilder.setId(**"007"**);  
 *//3、设置主题，可以是JSON数据-setSubject()* jwtBuilder.setSubject(**"测试主题"**);  
 *//4、设置签发日期-setIssuedAt* jwtBuilder.setIssuedAt(**new** Date());  
 *//5、设置签发人-setIssuer* jwtBuilder.setIssuer(**"www.itheima.com"**);  
 *//设置过期时间-30秒后  
 /\*Date exp = new Date(System.currentTimeMillis() + 30000);  
 jwtBuilder.setExpiration(exp);\*/  
  
 //自定义claims* Map<String, Object> user = **new** HashMap<>();  
 user.put(**"name"**, **"steven"**);  
 user.put(**"age"**, **"18"**);  
 user.put(**"address"**, **"深圳市.黑马程序员"**);  
 *//注意这里使用add方法，代表追加内容* jwtBuilder.addClaims(user);  
  
 *//6、设置签证* jwtBuilder.signWith(SignatureAlgorithm.***HS256***, **"itheima.steven"**);  
 *//7、生成令牌-compact()* String token = jwtBuilder.compact();  
 *//8、输出结果* System.***out***.println(token);  
}

运行输出令牌信息如下：

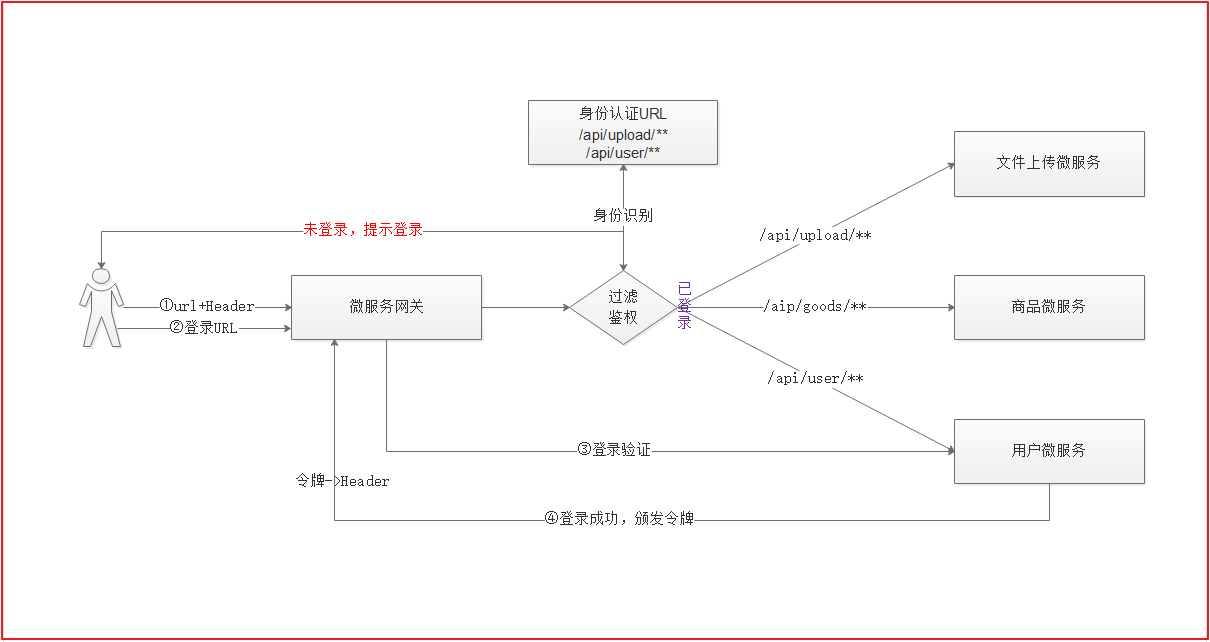
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIwMDciLCJzdWIiOiLmtYvor5XkuLvpopgiLCJpYXQiOjE1NjgxNzIxNjEsImlzcyI6Ind3dy5pdGhlaW1hLmNvbSIsImFkZHJlc3MiOiLmt7HlnLPluIIu6buR6ams56iL5bqP5ZGYIiwibmFtZSI6InN0ZXZlbiIsImFnZSI6IjE4In0.tsc98OiECra7aH4MS4H7f1-OB0K4W8wmaneZRFty3BA

解析令牌后，信息如下：

{jti=007, sub=测试主题, iat=1568172161, iss=www.itheima.com, address=深圳市.黑马程序员, name=steven, age=18}

## 鉴权处理

☒没有的话，就提示要的登录，在登陆成功之后，就将令牌保存；在使用微服务时，网关验证令牌，看是否含有令牌，有的话，就放行访问，没有，提示登录



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.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.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.如果需要权限访问，用户已经登录，则放行

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.用户通过网关访问用户微服务，进行登录验证

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.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

### 用户登录签发TOKEN

#### 生成令牌工具类

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

*/\*\*  
 \* 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  
  
 //Jwt令牌信息* **public static final** String ***JWT\_KEY*** = **"itcast"**;  
  
 */\*\*  
 \* 生成令牌  
 \** ***@param id*** *唯一id  
 \** ***@param subject*** *主题  
 \** ***@param ttlMillis*** *令牌有效时间(毫秒)，默认1小时  
 \** ***@return*** *令牌  
 \*/* **public static** String createJWT(String id, String subject, Long ttlMillis) {  
 *//指定算法* SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.***HS256***;  
  
 *//当前系统时间* **long** nowMillis = System.*currentTimeMillis*();  
 *//令牌签发时间* Date now = **new** Date(nowMillis);  
 *//如果令牌有效期为null，则默认设置有效期1小时* **if** (ttlMillis == **null**) {  
 ttlMillis = JwtUtil.***JWT\_TTL***;  
 }  
 *//令牌过期时间设置* **long** expMillis = nowMillis + ttlMillis;  
 Date expDate = **new** Date(expMillis);  
  
 *//生成秘钥* SecretKey secretKey = *generalKey*();  
  
 *//封装Jwt令牌信息* 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) *// 签名算法以及密匙* .setExpiration(expDate); *// 设置过期时间* **return** builder.compact();  
 }  
  
 */\*\*  
 \* 生成加密 secretKey  
 \** ***@return*** *\*/* **public static** SecretKey generalKey() {  
 **byte**[] encodedKey = Base64.*getEncoder*().encode(JwtUtil.***JWT\_KEY***.getBytes());  
 SecretKey key = **new** SecretKeySpec(encodedKey, 0, encodedKey.**length**, **"AES"**);  
 **return** key;  
 }  
  
 */\*\*  
 \* 解析令牌数据  
 \** ***@param jwt*** *密文  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** Claims parseJWT(String jwt) **throws** Exception {  
 SecretKey secretKey = *generalKey*();  
 **return** Jwts.*parser*()  
 .setSigningKey(secretKey)  
 .parseClaimsJws(jwt)  
 .getBody();  
 }  
}

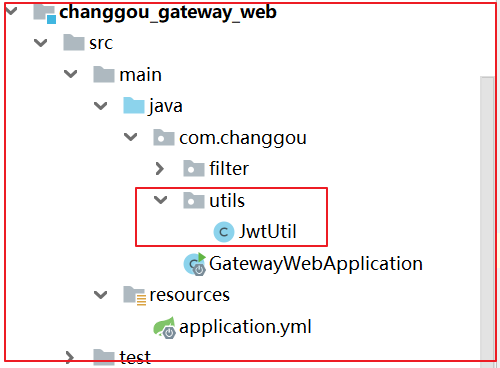
#### 用户登录成功 则 签发TOKEN，修改登录的方法：

修改

*/\*\*  
 \* 用户登录  
 \** ***@return*** *\*/*@RequestMapping(**"login"**)  
**public** Result login(String username, String password, HttpServletResponse response){  
 User user = **userService**.findById(username);  
 **if**(user == **null**){  
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"用户名不存在！"**);  
 }**else**{  
 *//验证加密的密码-checkpw(明文，密文)* **if**(BCrypt.*checkpw*(password,user.getPassword())){  
 *//封装令牌信息* Map<String, Object> map = **new** HashMap<>();  
 map.put(**"role"**,**"USER"**);  
 map.put(**"flag"**,**true**);  
 map.put(**"user"**,user);  
  
 *//创建令牌* String token = JwtUtil.*createJWT*(UUID.*randomUUID*().toString(), JSON.*toJSONString*(map), **null**);  
 *//把令牌传给客户端  
 //方式1-响应头中* response.addHeader(**"Authorization"**, token);  
 *//方式2-cookie中* Cookie cookie = **new** Cookie(**"Authorization"**, token);  
 response.addCookie(cookie);   
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"登录成功！"**,user);  
 }**else**{  
 **return new** Result(**false**, StatusCode.***LOGINERROR***, **"输入的密码错误！"**);  
 }  
 }  
}

#### 拷贝JwtUtil到changgou-gateway-web中

拷贝JwtUtil到changgou-gateway-web中，因为gateway与common工程存在Jar冲突，所以不有依赖。



#### 自定义全局拦截请求过滤器

在网关工程changgou-gateway-web中，创建com.changgou.filter.AuthorizeFilter过滤器类：

**步骤思路：**

//1、获取Request、Response对象-exchange.get...

//2、获取请求的URI-request.getURI().getPath()

//3、如果是登录请求-uri.startsWith，放行-chain.filter

//4、如果是非登录请求

//4.1 获取前端传入的令牌-从请求头中获取-request.getHeaders().getFirst

//4.2 如果头信息中没有，从请求参数中获取-request.getQueryParams().getFirst

//4.3 如果请求参数中没有，从cookie中获取-request.getCookies()-取值前先判断不为空-getFirst

//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete

// 4.5 如果获取到了令牌，解析令牌-JwtUtil.parseJWT，放行-chain.filter(exchange)

**代码实现：**

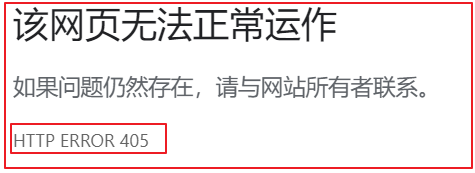
*/\*\*  
 \* 网关全局权限过滤器  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.filter  
 \*/*@Component  
**public class** AuthorizeFilter **implements** GlobalFilter, Ordered {  
  
 *//令牌的key* **private static final** String ***AUTHORIZE\_TOKEN*** = **"Authorization"**;  
  
 */\*\*  
 \* 拦截请求过滤规则逻辑  
 \** ***@param exchange*** *\** ***@param chain*** *\** ***@return*** *\*/* @Override  
 **public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  
 *//1、获取Request、Response对象-exchange.get...* ServerHttpRequest request = exchange.getRequest();  
 ServerHttpResponse response = exchange.getResponse();  
 *//2、获取请求的URI-request.getURI().getPath()* String uri = request.getURI().getPath();  
 *//3、如果是登录请求-uri.startsWith，放行-chain.filter* **if**(uri.startsWith(**"/api/user/login"**)){  
 **return** chain.filter(exchange);  
 }**else**{ *//4、如果是非登录请求  
 //4.1 获取前端传入的令牌-从请求头中获取-request.getHeaders().getFirst* String token = request.getHeaders().getFirst(***AUTHORIZE\_TOKEN***);  
 *//4.2 如果头信息中没有，从请求参数中获取-request.getQueryParams().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 token = request.getQueryParams().getFirst(***AUTHORIZE\_TOKEN***);  
 }  
 *//4.3 如果请求参数中没有，从cookie中获取-request.getCookies().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 HttpCookie first = request.getCookies().getFirst(***AUTHORIZE\_TOKEN***);  
 **if**(first != **null**){  
 token = first.getValue();  
 }  
 }  
 *//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete* **if**(StringUtils.*isEmpty*(token)){  
 *//返回405，错误表示方法不允许访问* response.setStatusCode(HttpStatus.***METHOD\_NOT\_ALLOWED***);  
 **return** response.setComplete();  
 }**else**{ *// 4.5 如果获取到了令牌，解析令牌-JwtUtil.parseJWT，放行-chain.filter(exchange)* **try** {  
 Claims claims = JwtUtil.*parseJWT*(token);  
 *//4.5.1解析成功-把令牌返回-request.mutate().header(key,value)* request.mutate().header(***AUTHORIZE\_TOKEN***, claims.toString());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//无效的认证* response.setStatusCode(HttpStatus.***UNAUTHORIZED***);  
 **return** response.setComplete();  
 }  
 **return** chain.filter(exchange);  
 }  
 }  
 }  
  
 */\*\*  
 \* 过滤器执行顺序设置  
 \** ***@return*** *\*/* @Override  
 **public int** getOrder() {  
 **return** 0;  
 }  
}

#### 配置过滤规则

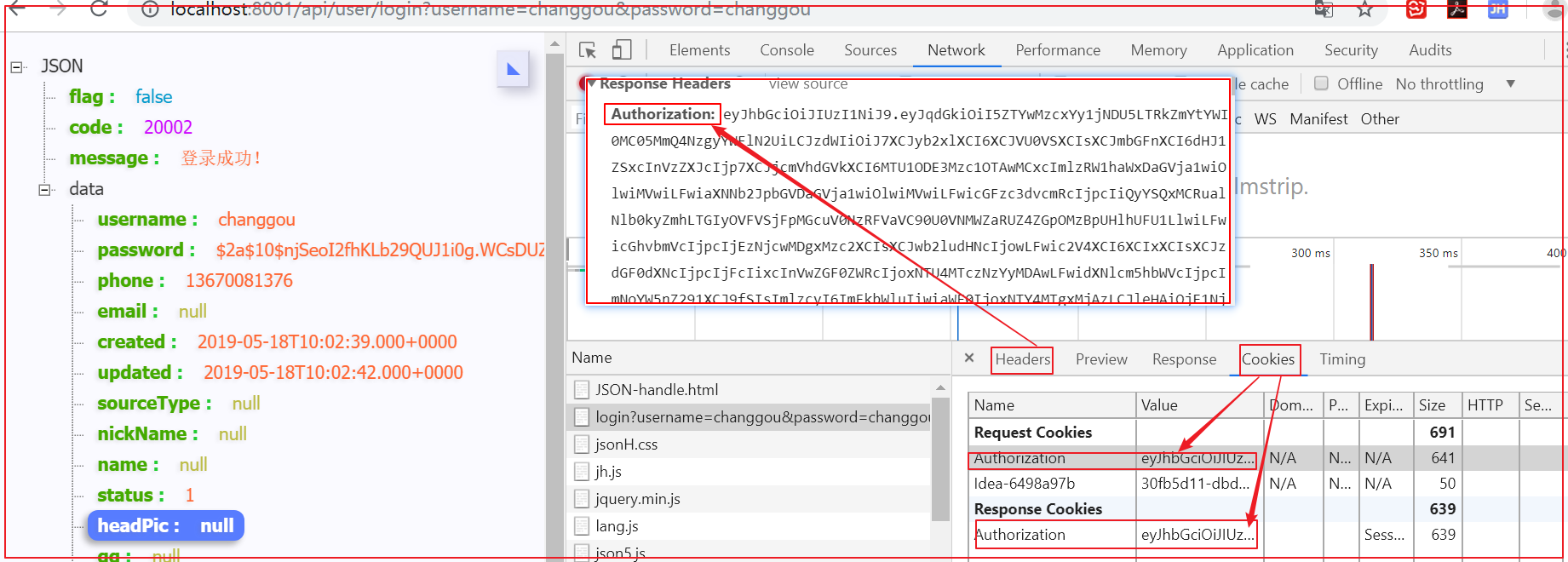
**spring**:  
 **application**:  
 **name**: gateway-web  
 **cloud**:  
 **gateway**:  
 *#添加此全局跨域过滤器后，Controller中就不再需要@CrossOrigin注解* **globalcors**:  
 **cors-configurations**:  
 **'[/\*\*]'**: *# 匹配所有请求* **allowedOrigins**: **"\*"** *#跨域处理 允许所有的域* **allowedMethods**: *# 支持的方法* - GET  
 - POST  
 - PUT  
 - DELETE  
 **routes**:  
 - **id**: changgou\_goods\_route *#路由唯一标识  
 #uri: http://localhost:18081 #路由转发地址* **uri**: lb://goods *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 *# 拦截请求是cloud.itheima.com域名的请求  
 #- Host=cloud.itheima.com\*\*  
 # 拦截路径为/brand的请求  
 #- Path=/brand/\*\** - 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**:  
 *#- PrefixPath=/brand # 自动添加请求前缀* - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径* - **name**: RequestRateLimiter *#请求数限流 名字不能随便写 ，使用默认的facatory* **args**:  
 *# 用于通过SPEL表达式来指定使用哪一个KeyResolver，需要先在启动引导类中定义* **key-resolver**: **"#{@ipKeyResolver}"** *# 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.replenishRate**: 1  
 *# 是指令牌桶的容量，允许在一秒钟内完成的最大请求数,将此值设置为零将阻止所有请求。* **redis-rate-limiter.burstCapacity**: 2  
  
 - **id**: changgou\_user\_route *#路由唯一标识-user* **uri**: lb://user *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 - Path=/api/user/\*\*,/api/address/\*\*,/api/areas/\*\*,/api/cities/\*\*,/api/provinces/\*\*  
 **filters**:  
 - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径* **redis**:  
 **host**: 192.168.211.132  
**server**:  
 **port**: 8001  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true**

#### 测试

没登录时，直接访问<http://localhost:8001/api/user/findAll>



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



登录后，再次访问<http://localhost:8001/api/user/findAll> ,就可以取到数据了

