

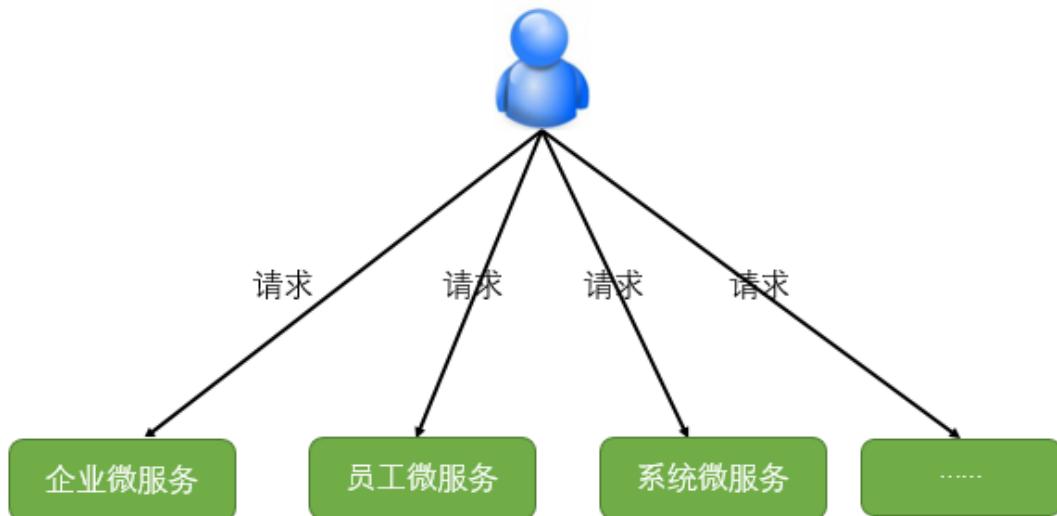


# 第14章 API网关与社保模块

- 理解zuul网关的作用
- 完成zuul网关的搭建
- 实现社保模块的代码开发

## 1 zuul网关

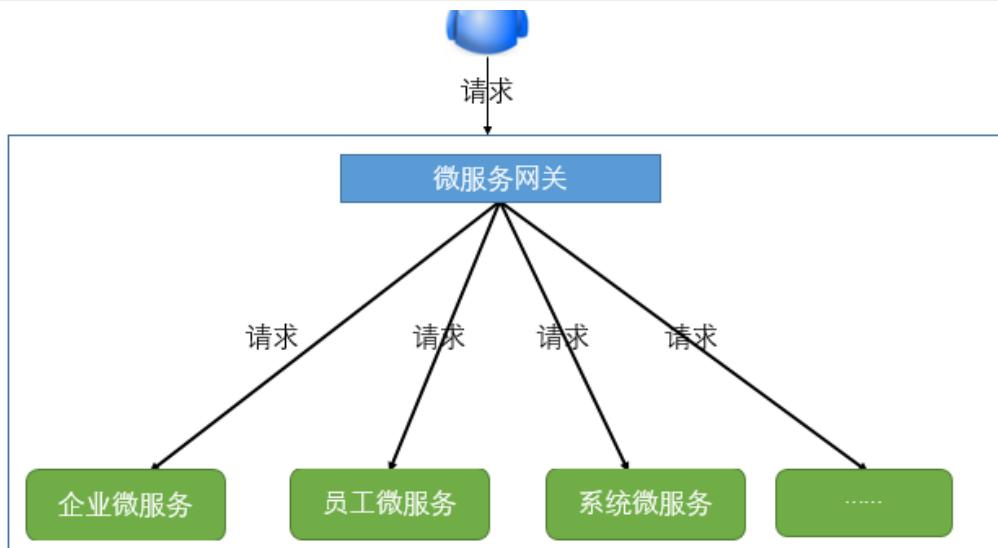
在学习完前面的知识后，微服务架构已经初具雏形。但还有一些问题：不同的微服务一般会有不同的网络地址，客户端在访问这些微服务时必须记住几十甚至几百个地址，这对于客户端方来说太复杂也难以维护。如下图：



如果让客户端直接与各个微服务通讯，可能会有很多问题：

- 客户端会请求多个不同的服务，需要维护不同的请求地址，增加开发难度
- 在某些场景下存在跨域请求的问题
- 加大身份认证的难度，每个微服务需要独立认证

因此，我们需要一个微服务网关，介于客户端与服务器之间的中间层，所有的外部请求都会先经过微服务网关。客户端只需要与网关交互，只知道一个网关地址即可，这样简化了开发还有以下优点：1、易于监控 2、易于认证 3、减少了客户端与各个微服务之间的交互次数



API网关是一个服务器，是系统对外的唯一入口。API网关封装了系统内部架构，为每个客户端提供一个定制的API。API网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。通常，网关也是提供REST/HTTP的访问API。服务端通过API-GW注册和管理服务。

## 1.1 什么是zuul网关

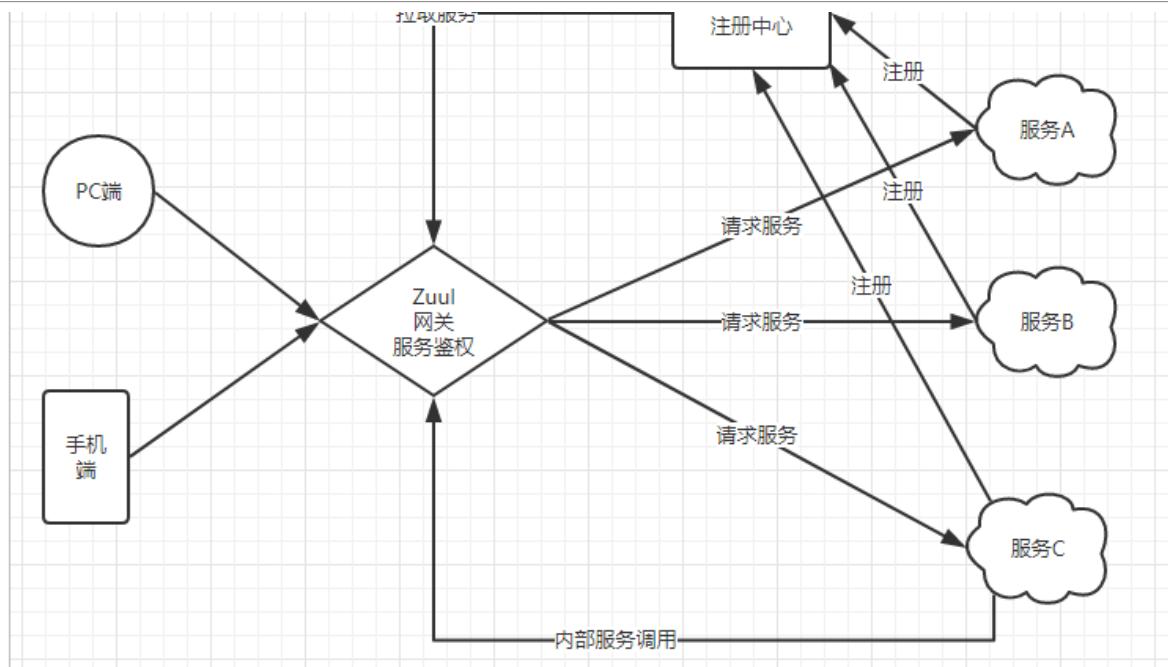


ZUUL是Netflix开源的微服务网关，它可以和Eureka、Ribbon、Hystrix等组件配合使用，Zuul组件的核心是一系列的过滤器，这些过滤器可以完成以下功能：

- 动态路由：动态将请求路由到不同后端集群
- 压力测试：逐渐增加指向集群的流量，以了解性能
- 负载分配：为每一种负载类型分配对应容量，并弃用超出限定值的请求
- 静态响应处理：边缘位置进行响应，避免转发到内部集群
- 身份认证和安全：识别每一个资源的验证要求，并拒绝那些不符的请求。Spring Cloud对Zuul进行了整合和增强。

Spring Cloud对Zuul进行了整合和增强

## 1.2 Zuul加入后的架构



不管是来自于客户端（PC或移动端）的请求，还是服务内部调用。一切对服务的请求都会经过Zuul这个网关，然后再由网关来实现 鉴权、动态路由等等操作。Zuul就是我们服务的统一入口。

## 1.3. 快速入门

### 1.3.1 工程搭建

创建工程 `ihrm_gate`，并导入zuul网关的响应依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

### 1.3.2 编写启动类

在网关工程中的 `com.ihrm.gate` 下创建启动类 `GateApplication`

```
/**
 * 网关启动类-
 */
@SpringBootApplication
@EnableZuulProxy
public class GateApplication {
    public static void main(String[] args) {
        SpringApplication.run(GateApplication.class);
    }
}
```

- `@EnableZuulProxy`注解开启Zuul网关功能

### 1.3.3 编写配置文件



```
spring:  
  application:  
    name: api-gateway #指定服务名
```

### 1.3.4 配置路由规则

通过zuul网关请求企业微服务，需要在application.xml中配置路由转发规则如下

```
zuul:  
  routes:  
    ihrm-company: # 这里是路由id，随意写  
      path: /ihrm-company/** # 这里是映射路径  
      url: http://127.0.0.1:9001 # 映射路径对应的实际url地址
```

## 1.4 路由配置

在刚才的路由规则中，我们把路径对应的服务地址写死了！如果同一服务有多个实例的话，这样做显然就不合理了。我们应该根据服务的名称，去Eureka注册中心查找服务对应的所有实例列表，然后进行动态路由才对！

### 1.4.1 添加Eureka客户端发现功能

(1) 添加eureka依赖

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

(2) 修改启动类，开启服务发现功能

```
@SpringBootApplication  
@EnableZuulProxy // 开启zuul的网关功能  
@EnableDiscoveryClient  
public class GateApplication {  
  public static void main(String[] args) {  
    SpringApplication.run(GateApplication.class, args);  
  }  
}
```

### 1.4.2 修改配置文件

(1) 在application.xml中添加eureka服务发现的相关配置信息



```
registry-fetch-interval-seconds: 5 # 获取服务列表的周期: 5s
service-url:
  defaultZone: http://127.0.0.1:10086/eureka
instance:
  prefer-ip-address: true
  ip-address: 127.0.0.1
```

zuul网关从eureka注册中心中获取服务提供者的所有数据

### ( 2 ) 修改zuul路由的映射规则

因为已经有了Eureka客户端，我们可以从Eureka获取服务的地址信息，因此映射时无需指定IP地址，而是通过服务名称来访问，而且Zuul已经集成了Ribbon的负载均衡功能。

```
zuul:
  routes:
    ihrm-company: # 这里是路由id, 随意写
      path: /ihrm-company/** # 这里是映射路径
      serviceId: ihrm-company # 指定服务名称
```

## 1.4.3 简化配置

在刚才的配置中，我们的规则是这样的：

- `zuul.routes.<route>.path=/xxx/**`：来指定映射路径。`<route>`是自定义的路由名
- `zuul.routes.<route>.serviceId=/user-service`：来指定服务名。

而大多数情况下，我们的`<route>`路由名称往往和 服务名会写成一样的。因此Zuul就提供了一种简化的配置语法：`zuul.routes.<serviceId>=<path>`

比方说上面我们关于user-service的配置可以简化为一条(省去了对服务名称的配置)：

```
zuul:
  routes:
    ihrm-company: /ihrm-company/** # 这里是映射路径
```

在使用Zuul的过程中，上面讲述的规则已经大大的简化了配置项。但是当服务较多时，配置也是比较繁琐的。因此Zuul就指定了默认的路由规则：

- 默认情况下，一切服务的映射路径就是服务名本身。
  - 例如服务名为：`ihrm-company`，则默认的映射路径就是：`/ihrm-company/**`

也就是说，刚才的映射规则我们完全不配置也是可以的

## 1.5 前端修改

由于使用了zuul网关统一对所有微服务进行转发，那么我们在前端系统调用的时候，只需要向网关服务器发送请求即可。在前端框架中的`config/index.js`中修改请求URL



```
// target: 'https://www.easy-
mock.com/mock/5ab213e33666166110a94928/admin',
// target: 'http://172.17.0.58:7999',
// target: 'http://172.16.43.141:8080',
// target: 'http://172.16.43.86:8080',
target: 'http://localhost:8888',
changeOrigin: true,
pathRewrite: {
  '^/api': ''
},
},
},
```

## 2 基于Zuul的统一鉴权

spring cloud Zuul包含了对请求的路由和过滤2个功能。路由功能负责将请求转发到具体的微服务上，而过滤器负责对请求的处理过程进行干预，是实现权限校验、服务聚合等功能的基础。

### 2.1 Zuul的过滤器

#### 2.1.1 ZuulFilter

ZuulFilter是过滤器的顶级父类。在这里我们看一下其中定义的4个最重要的方法：

```
public abstract ZuulFilter implements IZuulFilter{

    abstract public String filterType();

    abstract public int filterOrder();

    boolean shouldFilter(); // 来自IZuulFilter

    Object run() throws ZuulException; // IZuulFilter
}
```

- `shouldFilter`：返回一个 Boolean 值，判断该过滤器是否需要执行。返回true执行，返回false不执行。
- `run`：过滤器的具体业务逻辑。
- `filterType`：返回字符串，代表过滤器的类型。包含以下4种：
  - `pre`：请求在被路由之前执行
  - `routing`：在路由请求时调用
  - `post`：在routing和error过滤器之后调用
  - `error`：处理请求时发生错误调用
- `filterOrder`：通过返回的int值来定义过滤器的执行顺序，数字越小优先级越高。

#### 2.1.2 自定义过滤器

接下来我们来自定义一个过滤器，用于深入理解zuul过滤器的执行过程

```

@Override
public String filterType() {
    // 前置过滤器
    return "pre";
}

@Override
public int filterOrder() {
    //优先级为0，数字越大，优先级越低
    return 0;
}

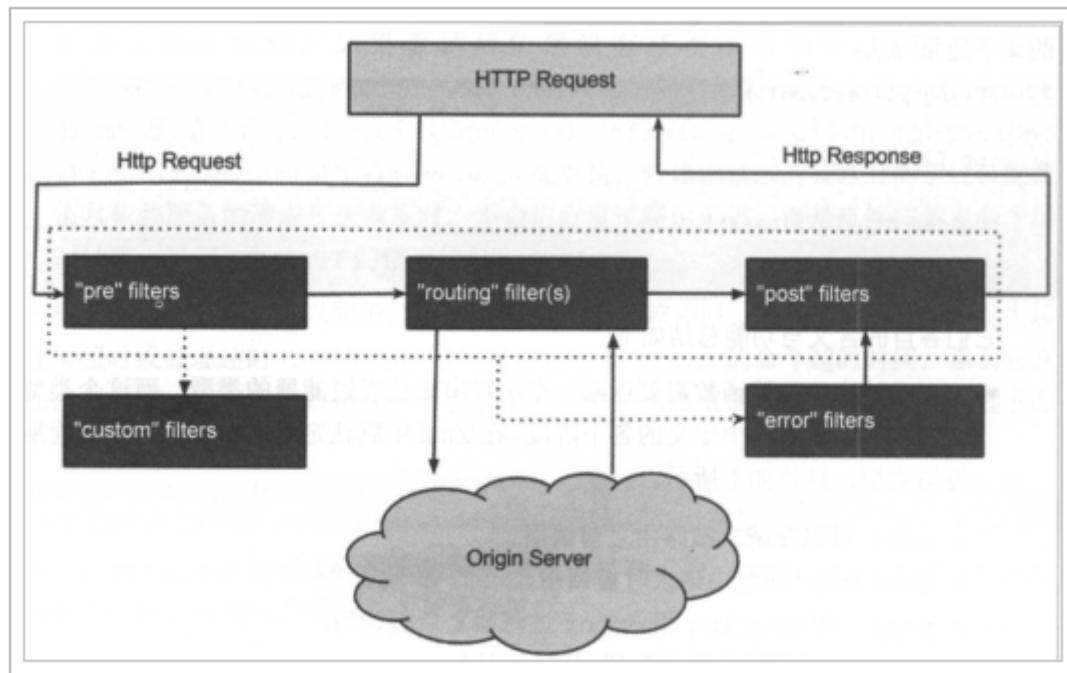
@Override
public boolean shouldFilter() {
    // 是否执行该过滤器，此处为true，说明需要过滤
    return true;
}

//过滤器执行的具体逻辑
@Override
public Object run() throws zuulException {
    System.out.println("zuul过滤器...");
    return null;
}
}

```

### 2.1.3 过滤器执行生命周期

这张是Zuul官网提供的请求生命周期图，清晰的表现了一个请求在各个过滤器的执行顺序。

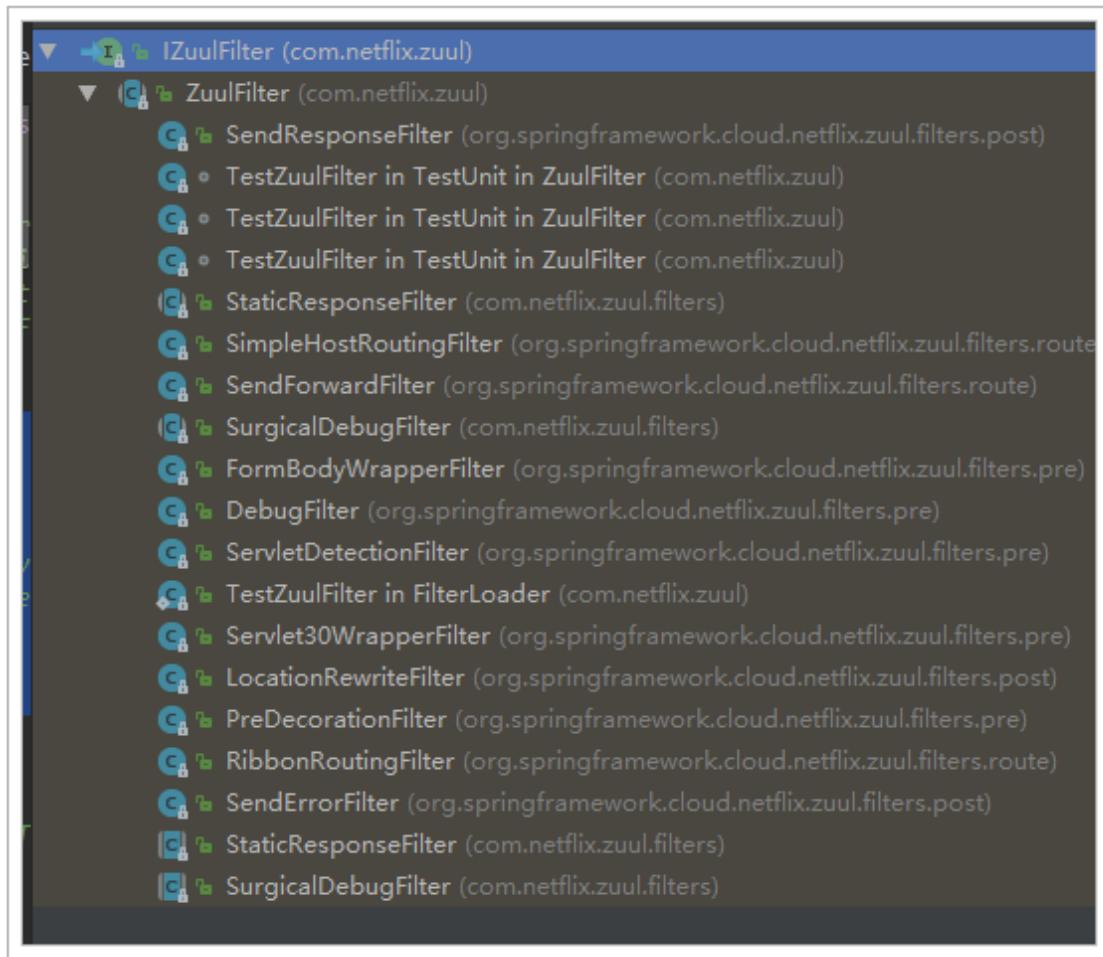


- 正常流程：
  - 请求到达首先会经过pre类型过滤器，而后到达routing类型，进行路由，请求就到达真正的服务提供者，执行请求，返回结果后，会到达post过滤器。而后返回响应。
- 异常流程：



- 如果是error过滤器自己出现异常，最终也会进入POST过滤器，而后返回。
- 如果是POST过滤器出现异常，会跳转到error过滤器，但是与pre和routing不同的时，请求不会再到达POST过滤器了。

所有内置过滤器列表：



## 2.1.4 使用场景

场景非常多：

- 请求鉴权：一般放在pre类型，如果发现没有访问权限，直接就拦截了
- 异常处理：一般会在error类型和post类型过滤器中结合起来处理。
- 服务调用时长统计：pre和post结合使用。

## 2.2 统一鉴权

### 2.2.1 基于JWT的统一鉴权

在某些应用中，往往使用JWT的形式进行无状态的用户鉴权。对于JWT的鉴权，只需要使用Zuul的自定义过滤器，在过滤器中判断是否携带JWT的token信息即可。

```
@Component
public class AuthFilter extends zuulFilter{
    @Override
    public String filterType() {
        // 登录校验，肯定是在前置拦截
    }
}
```



```
@Override
public int filterOrder() {
    // 顺序设置为1
    return 1;
}

@Override
public boolean shouldFilter() {
    // 返回true，代表过滤器生效。
    return true;
}

@Override
public Object run() throws ZuulException {
    // 登录校验逻辑。
    // 1) 获取zuul提供的请求上下文对象
    RequestContext ctx = RequestContext.getCurrentContext();
    // 2) 从上下文中获取request对象
    HttpServletRequest req = ctx.getRequest();
    // 3) 从请求中获取token
    String token = req.getHeader("Authorization");
    // 4) 判断
    if(token == null || "".equals(token.trim())){
        // 没有token，登录校验失败，拦截
        ctx.setSendZuulResponse(false);
        // 返回401状态码。也可以考虑重定向到登录页。
        ctx.setResponseStatus(HttpStatus.UNAUTHORIZED.value());
    }
    // 校验通过，可以考虑把用户信息放入上下文，继续向后执行
    return null;
}
}
```

## 2.2.2 基于Shiro的统一鉴权

由于我们的系统使用shiro结合自定义session的形式，相当于将用户数据存储到了分布式缓存redis中。  
那么只需要再zuul中使用shiro即可完成统一用户权限校验

### (1) 引入shiro依赖

```
<dependency>
<groupId>com.ihrm</groupId>
<artifactId>ihrm_common</artifactId>
<version>1.0-SNAPSHOT</version>
</dependency>
```

### (2) 配置shiro

```
@Configuration
public class ShiroConfiguration {

    //1. 创建realm
    @Bean
```



```
}

//2.创建安全管理器
@Bean
public SecurityManager getSecurityManager(IhrmRealm realm) {
    DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager();
    securityManager.setRealm(realm);

    //将自定义的会话管理器注册到安全管理器中
    securityManager.setSessionManager(sessionManager());
    //将自定义的redis缓存管理器注册到安全管理器中
    securityManager.setCacheManager(cacheManager());

    return securityManager;
}

//3.配置shiro的过滤器工厂

/**
 * 在web程序中，shiro进行权限控制全部是通过一组过滤器集合进行控制
 *
 */
@Bean
public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
    //1.创建过滤器工厂
    ShiroFilterFactoryBean filterFactory = new ShiroFilterFactoryBean();
    //2.设置安全管理器
    filterFactory.setSecurityManager(securityManager);
    //3.通用配置（跳转登录页面，未授权跳转的页面）
    filterFactory.setLoginUrl("/autherror?code=1");//跳转url地址
    filterFactory.setUnauthorizedUrl("/autherror?code=2");//未授权的url
    //4.设置过滤器集合
    Map<String, String> filterMap = new LinkedHashMap<>();
    //anon -- 匿名访问
    filterMap.put("/sys/login", "anon");
    filterMap.put("/autherror", "anon");
    //注册
    //authc -- 认证之后访问（登录）
    filterMap.put("/**", "authc");
    //perms -- 具有某中权限（使用注解配置授权）
    filterFactory.setFilterChainDefinitionMap(filterMap);

    return filterFactory;
}

@Value("${spring.redis.host}")
private String host;
@Value("${spring.redis.port}")
private int port;

/**
 * 1.redis的控制器，操作redis
 */
public RedisManager redisManager() {
    北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090
}
```



```
redisManager.setPort(port);
    return redisManager;
}

/**
 * 2.sessionDao
 */
public RedisSessionDAO redisSessionDAO() {
    RedisSessionDAO sessionDAO = new RedisSessionDAO();
    sessionDAO.setRedisManager(redisManager());
    return sessionDAO;
}

/**
 * 3.会话管理器
 */
public DefaultWebSessionManager sessionManager() {
    CustomSessionManager sessionManager = new CustomSessionManager();
    //sessionManager.setSessionIdCookieEnabled(false);
    sessionManager.setSessionIdUrlRewritingEnabled(false);
    sessionManager.setSessionDAO(redisSessionDAO());
    return sessionManager;
}

/**
 * 4.缓存管理器
 */
public RedisCacheManager cacheManager() {
    RedisCacheManager redisCacheManager = new RedisCacheManager();
    redisCacheManager.setRedisManager(redisManager());
    return redisCacheManager;
}

//开启对shiro注解的支持
@Bean
public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
    AuthorizationAttributeSourceAdvisor advisor = new
    AuthorizationAttributeSourceAdvisor();
    advisor.setSecurityManager(securityManager);
    return advisor;
}
}
```

由于再zuul网关中已经进行了统一的权限校验，那么其它微服务的权限校验就可以关闭了。

## 2.2.3 传递敏感header

经过测试可以发现在网关中明明已经具备了权限，可以在具体的微服务中还是会告知权限不足或者没有找到相关用户。这是因为在Zuul进行请求转发的时候，会把header清空，为了传递原始的header信息到最终的微服务，在配置加上：



# 3 社保管理

## 3.1 需求分析

序号	姓名	入职时间	手机号	身份证号码	学历	开户行	一级部门	二级部门
1	itcast	2018-11-02	13800000002		本科		test1	
2	zpz	2018-11-04	13800000003				测试部	
3	ll	2018-12-02	13800000004				测试部	
4	a01	2018-01-01	13400000001				开发部	
5	a02	2018-01-01	13400000002		初中		开发部	
6	test001	2018-01-01	13500000001				开发部	
7	test002	2018-01-01	13500000002				开发部	
8	test003	2018-01-01	13500000003				开发部	

完成社保模板相关代码开发：

- 企业员工参保设置
- 企业月度社保明细
- 企业社保归档数据

## 3.2 数据库表

### (1) 社保归档表

```
CREATE TABLE `ss_archive` (
  `id` varchar(40) NOT NULL COMMENT 'id',
  `company_id` varchar(40) NOT NULL COMMENT '企业id',
  `years_month` varchar(255) NOT NULL COMMENT '年月',
  `creation_time` date NOT NULL COMMENT '创建时间',
  `enterprise_payment` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '企业缴纳',
  `personal_payment` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '个人缴纳',
  `total` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '合计',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-归档表'
```

### (2) 社保-归档详情表

```
CREATE TABLE `ss_archive_detail` (
  `id` varchar(40) NOT NULL COMMENT 'id',
  `archive_id` varchar(40) NOT NULL COMMENT '归档id',
  `user_id` varchar(40) DEFAULT NULL COMMENT '用户id',
  `username` varchar(255) DEFAULT NULL COMMENT '用户名',
  `time_of_entry` varchar(255) DEFAULT NULL COMMENT '入职时间',
  `mobile` varchar(255) DEFAULT NULL COMMENT '手机号',
  `id_number` varchar(255) DEFAULT NULL COMMENT '身份证号',
  北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090
)
```



```
`bank_card_number` varchar(255) DEFAULT NULL COMMENT '银行卡号',
`first_level_department` varchar(255) DEFAULT NULL COMMENT '一级部门',
`two_level_department` varchar(255) DEFAULT NULL COMMENT '二级部门',
`working_city` varchar(255) DEFAULT NULL COMMENT '工作城市',
`social_security_computer_number` varchar(255) DEFAULT NULL COMMENT '社保电脑号',
`provident_fund_account` varchar(255) DEFAULT NULL COMMENT '公积金账号',
`leave_date` varchar(255) DEFAULT NULL COMMENT '离职时间',
`household_registration_type` varchar(255) DEFAULT NULL COMMENT '户籍类型',
`participating_in_the_city` varchar(255) DEFAULT NULL COMMENT '参保城市',
`social_security_month` varchar(255) DEFAULT NULL COMMENT '社保月份',
`social_security_base` decimal(10,2) DEFAULT NULL COMMENT '社保基数',
`social_security` decimal(10,2) DEFAULT NULL COMMENT '社保合计',
`social_security_enterprise` decimal(10,2) DEFAULT NULL COMMENT '社保企业',
`social_security_individual` decimal(10,2) DEFAULT NULL COMMENT '社保个人',
`provident_fund_city` varchar(255) DEFAULT NULL COMMENT '公积金城市',
`provident_fund_month` varchar(255) DEFAULT NULL COMMENT '公积金月份',
`provident_fund_base` decimal(10,2) DEFAULT NULL COMMENT '公积金基数',
`accumulation_fund_enterprise_base` decimal(10,2) DEFAULT NULL COMMENT '公积金企业基数',
`proportion_of_provident_fund_enterprises` decimal(10,2) DEFAULT NULL COMMENT
'公积金企业比例',
`individual_base_of_provident_fund` decimal(10,2) DEFAULT NULL COMMENT '公积金个人基数',
`personal_ratio_of_provident_fund` decimal(10,2) DEFAULT NULL COMMENT '公积金个人比例',
`total_provident_fund` decimal(10,2) DEFAULT NULL COMMENT '公积金合计',
`provident_fund_enterprises` decimal(10,2) DEFAULT NULL COMMENT '公积金企业',
`provident_fund_individual` decimal(10,2) DEFAULT NULL COMMENT '公积金个人',
`pension_enterprise_base` decimal(10,2) DEFAULT NULL COMMENT '养老企业基数',
`proportion_of_pension_enterprises` decimal(10,2) DEFAULT NULL COMMENT '养老企业比例',
`pension_enterprise` decimal(10,2) DEFAULT NULL COMMENT '养老企业',
`personal_pension_base` decimal(10,2) DEFAULT NULL COMMENT '养老个人基数',
`personal_pension_ratio` decimal(10,2) DEFAULT NULL COMMENT '养老个人比例',
`old_age_individual` decimal(10,2) DEFAULT NULL COMMENT '养老个人',
`unemployment_enterprise_base` decimal(10,2) DEFAULT NULL COMMENT '失业企业基
数',
`proportion_of_unemployed_enterprises` decimal(10,2) DEFAULT NULL COMMENT '失业企业比例',
`unemployed_enterprise` decimal(10,2) DEFAULT NULL COMMENT '失业企业',
`the_number_of_unemployed_individuals` decimal(10,2) DEFAULT NULL COMMENT '失业个人基数',
`percentage_of_unemployed_individuals` decimal(10,2) DEFAULT NULL COMMENT '失业个人比例',
`unemployed_individual` decimal(10,2) DEFAULT NULL COMMENT '失业个人',
`medical_enterprise_base` decimal(10,2) DEFAULT NULL COMMENT '医疗企业基数',
`proportion_of_medical_enterprises` decimal(10,2) DEFAULT NULL COMMENT '医疗企
业比例',
`medical_enterprise` decimal(10,2) DEFAULT NULL COMMENT '医疗企业',
`medical_personal_base` decimal(10,2) DEFAULT NULL COMMENT '医疗个人基数',
`medical_personal_ratio` decimal(10,2) DEFAULT NULL COMMENT '医疗个人比例',
`medical_individual` decimal(10,2) DEFAULT NULL COMMENT '医疗个人',
`base_of_industrial_injury_enterprises` decimal(10,2) DEFAULT NULL COMMENT '工
伤企业基数',
```



```
`industrial_injury_enterprise` decimal(10,2) DEFAULT NULL COMMENT '工伤企业',
`fertility_enterprise_base` decimal(10,2) DEFAULT NULL COMMENT '生育企业基数',
`proportion_of_fertility_enterprises` decimal(10,2) DEFAULT NULL COMMENT '生育企业比例',
`childbearing_enterprise` decimal(10,2) DEFAULT NULL COMMENT '生育企业',
`base_of_serious_illness` decimal(10,2) DEFAULT NULL COMMENT '大病企业基数',
`proportion_of_seriously_ill_enterprises` decimal(10,2) DEFAULT NULL COMMENT
'大病企业比例',
`big_disease_enterprise` decimal(10,2) DEFAULT NULL COMMENT '大病企业',
`personal_base_of_serious_illness` decimal(10,2) DEFAULT NULL COMMENT '大病个人
基数',
`personal_proportion_of_serious_illness` decimal(10,2) DEFAULT NULL COMMENT
'大病个人比例',
`a_person_of_great_disease` decimal(10,2) DEFAULT NULL COMMENT '大病个人',
`provident_fund_notes` text COMMENT '公积金备注',
`social_security_notes` text COMMENT '社保备注',
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-归档详情'
```

### (3) 社保-城市与缴费项目关联表

```
CREATE TABLE `ss_city_payment_item` (
`id` varchar(40) NOT NULL,
`city_id` varchar(40) NOT NULL COMMENT '城市id',
`payment_item_id` varchar(40) NOT NULL COMMENT '缴费项目id',
`switch_company` tinyint(1) NOT NULL COMMENT '企业是否缴纳开关, 0为禁用, 1为启用',
`scale_company` decimal(6,2) DEFAULT NULL COMMENT '企业比例',
`switch_personal` tinyint(1) NOT NULL COMMENT '个人是否缴纳开关, 0为禁用, 1为启用',
`scale_personal` decimal(6,2) DEFAULT NULL COMMENT '个人比例',
PRIMARY KEY (`id`),
UNIQUE KEY `UK_CID_PIID` (`city_id`, `payment_item_id`) USING BTREE COMMENT '城
市id与缴费项目id组合唯一'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-城市与缴费项目关联表'
```

### (4) 社保-企业设置信息

```
CREATE TABLE `ss_company_settings` (
`company_id` varchar(40) NOT NULL COMMENT '企业id',
`is_settings` tinyint(1) NOT NULL DEFAULT '0' COMMENT '0是未设置, 1是已设置',
`data_month` varchar(40) NOT NULL COMMENT '当前显示报表月份',
PRIMARY KEY (`company_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-企业设置信息'
```

### (5) 社保-缴费项目



```
    `name` varchar(255) NOT NULL COMMENT '缴费项目名称',
    `switch_company` tinyint(1) NOT NULL DEFAULT '0' COMMENT '企业是否缴纳开关, 0为禁用, 1为启用',
    `scale_company` decimal(6,2) NOT NULL DEFAULT '0.00' COMMENT '企业比例',
    `switch_personal` tinyint(1) NOT NULL DEFAULT '0' COMMENT '个人是否缴纳开关, 0为禁用, 1为启用',
    `scale_personal` decimal(6,2) NOT NULL DEFAULT '0.00' COMMENT '个人比例',
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-缴费项目'
```

## ( 6 ) 社保-用户社保信息表

```
CREATE TABLE `ss_user_social_security` (
    `user_id` varchar(40) NOT NULL COMMENT '用户id',
    `enterprises_pay_social_security_this_month` tinyint(1) NOT NULL DEFAULT '0'
COMMENT '本月是否缴纳社保 0为不缴纳 1为缴纳',
    `enterprises_pay_the_provident_fund_this_month` tinyint(1) NOT NULL DEFAULT
'0' COMMENT '本月是否缴纳公积金 0为不缴纳 1为缴纳',
    `participating_in_the_city_id` varchar(40) NOT NULL COMMENT '参保城市id',
    `social_security_type` tinyint(1) NOT NULL DEFAULT '2' COMMENT '参保类型 1为首次开户 2为非首次开户',
    `household_registration_type` tinyint(1) NOT NULL COMMENT '户籍类型 1为本市城镇 2
为本市农村 3为外埠城镇 4为外埠农村',
    `social_security_base` int(8) NOT NULL COMMENT '社保基数',
    `industrial_injury_ratio` decimal(6,2) DEFAULT NULL COMMENT '工伤比例',
    `social_security_notes` varchar(300) DEFAULT NULL COMMENT '社保备注',
    `provident_fund_city_id` varchar(40) NOT NULL COMMENT '公积金城市id',
    `provident_fund_base` int(8) NOT NULL COMMENT '公积金基数',
    `enterprise_proportion` decimal(6,2) NOT NULL COMMENT '公积金企业比例',
    `personal_proportion` decimal(6,2) NOT NULL COMMENT '公积金个人比例',
    `enterprise_provident_fund_payment` decimal(8,2) NOT NULL COMMENT '公积金企业缴
纳数额',
    `personal_provident_fund_payment` decimal(8,2) NOT NULL COMMENT '公积金个人缴纳数
额',
    `provident_fund_notes` varchar(300) DEFAULT NULL COMMENT '公积金备注',
    `create_time` datetime NOT NULL COMMENT '创建时间',
    `last_modify_time` datetime NOT NULL COMMENT '最后修改时间',
    `social_security_switch_update_time` datetime NOT NULL COMMENT '社保是否缴纳变更
时间',
    `provident_fund_switch_update_time` datetime NOT NULL COMMENT '公积金是否缴纳变更
时间',
PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='社保-用户社保信息表'
```

## 3.3 搭建环境

( 1 ) 使用代码生成工具，根据数据库表生成最基本的实体类，dao接口和service层代码。

略

( 2 ) 创建社保管理模块ihrm\_social\_securities，并将自动生成的代码依次copy到响应的包下。

( 3 ) 修改zuul网关，添加社保相关的请求转发



```
serviceId: ihrm-social-securitys #指定Eureka注册中心中的服务id
strip-prefix: false
sentiviteHeaders:
customSensitiveHeaders: true
```

## 3.4 社保列表

The screenshot shows a web-based application interface for managing social security data. At the top, there is a search bar with placeholder text '请输入内容' and several buttons: '导入' (Import), '历史归档' (History Archiving), and '八月报表' (August Report). Below the search bar, there are several filter options:

- 部门 :** Includes checkboxes for Research Department, Testing Department, Finance Department, Admin Center, Development Department, test1, Research Institute, Product Information Group, R&D Department, President's Office, and HR Department.
- 社保城市 :** Includes a checkbox for Beijing.
- 公积金城市 :** Includes a checkbox for Beijing.

The main content area displays a table of employee data:

序号	姓名	手机	工号	部门	入职时间	离职时间	社保城市	公积金城市	社保基数	公
1	毕鹏	1380000 0002	9002	开发部	2018-11- 02	--	北京	北京	10000	1
2	zbz	1380000 0003	111	测试部	2018-11- 04	--	北京	北京	3388	2
3	ll	1380000 0004	1111	测试部	2018-12- 02	--				

### 3.4.1 企业社保设置

企业第一次进入社保页面，会要求输入制作社保记录的日期

```
/***
 * 获取企业是否设置社保
 * @return
 */
@RequestMapping(value = "/settings", method = RequestMethod.GET)
public Result getSettings() throws Exception {
    CompanySettings companySettings =
companySettingsService.findById(companyId);
    return new Result(ResultCode.SUCCESS, companySettings);
}

/***
 * 保存企业社保设置
 */
@RequestMapping(value = "/settings", method = RequestMethod.POST)
public Result saveSettings(@RequestBody CompanySettings companySettings){
    companySettings.setCompanyId(companyId);
    companySettingsService.save(companySettings);
    return new Result(ResultCode.SUCCESS);
}
```

### 3.4.2 查询所有参保人员数据列表



```
 */
@RequestMapping(value = "/list", method = RequestMethod.POST)
public Result list(@RequestBody SearchListvo
searchListVo,@RequestParam(defaultValue = "1") int
page,@RequestParam(defaultValue = "1") int pageSize) throws Exception {
    Page<UserSocialSecurityItem> itemPage = userSocialService.findAll(page,
pageSize, companyId, searchListvo.getDepartmentChecks(),
searchListvo.getSocialSecurityChecks(), searchListvo.getProvidentFundChecks());
    PageResult pageResult = new PageResult(itemPage.getTotalElements(),
itemPage.getContent());
    return new Result(ResultCode.SUCCESS, pageResult);
}
```

## 3.5 社保设置

### 3.5.1 获取不同城市的社保缴费项目

```
/**
 * 根据城市id获取社保缴费项目
 */
@RequestMapping(value = "/payment_item/{cityId}", method = RequestMethod.GET)
public Result findPaymentItemByCityId(@PathVariable(value = "cityId") String
cityId) {
    List<CityPaymentItem> cityPaymentItemList =
paymentItemService.findAllByCityId(cityId);
    return new Result(ResultCode.SUCCESS, cityPaymentItemList);
}
```

### 3.5.2 展示员工社保数据

```
/**
 * 获取社保信息
 */
@RequestMapping(value = "/{userId}", method = RequestMethod.GET)
public Result findById(@PathVariable(value = "userId") String userId) throws
Exception {
    //获取用户数据
    User user = (User) systemFeignClient.findById(userId).getData();
    //查询用户的社保数据
    UserSocialSecurity userSocialSecurity =
userSocialService.findByUserId(userId);
    Map map = new HashMap<>();
    map.put("user",user);
    map.put("userSocialSecurity",userSocialSecurity);
    return new Result(ResultCode.SUCCESS, map);
}
```

### 3.5.3 设置员工参保数据



```
    * 保存用户社保信息
    */
@ApiOperation(value="保存用户社保信息",httpMethod = "PUT")
@RequestMapping(value = "/{userId}", method = RequestMethod.PUT)
public Result save(@RequestBody UserSocialSecurity userSocialSecurity) {
    userSocialService.save(userSocialSecurity);
    return new Result(ResultCode.SUCCESS);
}
```

### 3.5.4 批量导入员工参保数据

```
@RequestMapping(value = "/import", method = RequestMethod.POST)
public Result importSocialSecurity(@RequestParam(name = "file") MultipartFile file) throws Exception {
    List<UserSocialSecurity> list = new
    ExcelImportUtil(UserSocialSecurity.class).readExcel(file.getInputStream(), 1,
    0);
    for (UserSocialSecurity item : list) {
        UserSocialSecurity us =
        userSocialService.findByUserId(item.getUserId());
        if (us == null) {
            userSocialService.save(item);
        }
    }
    return new Result(ResultCode.SUCCESS);
}
```

## 3.6 月报表

### 3.6.1 展示月报表数据

```
@RequestMapping(value = "/histories/{yearMonth}", method = RequestMethod.GET)
public Result histories(@ApiParam(value = "年月", required = true)
@PathVariable(value = "yearMonth") String yearMonth, @RequestParam(value =
"opType") Integer opType) {
    List<ArchiveDetail> reportVoList = new ArrayList<>();
    if (opType == 1) {
        reportVoList.addAll(archiveService.getReports(yearMonth, companyId));
    } else {
        Archive archive = archiveService.findArchive(companyId, yearMonth);
        if (archive != null) {
            reportVoList =
        archiveService.findAllDetailByArchiveId(archive.getId());
        }
    }
    return new Result(ResultCode.SUCCESS, reportVoList);
}
```

### 3.6.2 导出月报表



```
let getName = "社保报表"
let xxxx = XLSX.utils.table_to_book(
    document.querySelector("#item"),
    xlsxParam
);
getBlob(getName, xxxx, XLSX.write, FileSaver.saveAs);
this.$message.success("导出报表成功！");
this.$forceUpdate()
},
```

## 3.7 历史归档

归档：同义词为存档，指将处理完并且具有保存价值的事情或文件经系统整理后交档案室（馆）保存备案(备查)的过程。

### 3.7.1 归档月报表数据

```
@RequestMapping(value = "/historys/{yearMonth}/archive", method =
RequestMethod.POST)
public Result archive(@PathVariable(value = "yearMonth") String yearMonth)
throws Exception {
    //构造归档详情数据列表
    List<ArchiveDetail> reportVoList =
archiveservice.getReports(yearMonth,companyId);
    //企业费用
    BigDecimal enterprisePayment = new BigDecimal(0);
    //个人费用
    BigDecimal personalPayment = new BigDecimal(0);
    //循环计算企业与个人部分费用总和
    for (ArchiveDetail vo : reportVoList) {
        enterprisePayment =
enterprisePayment.add(vo.getProvidentFundEnterprises().add(vo.getSocialSecurityEnterprise()));
        personalPayment =
personalPayment.add(vo.getSocialSecurityIndividual().add(vo.getProvidentFundIndividual()));
    }

    Archive archive = archiveservice.findArchive(companyId, yearMonth);

    if (archive == null) {
        archive = new Archive(companyId,yearMonth);
    }

    archive.setEnterprisePayment(enterprisePayment);
    archive.setPersonalPayment(personalPayment);
    archive.setTotal(enterprisePayment.add(personalPayment));

    archiveservice.save(archive);
    archiveservice.batchSaveDetail(archive.getId(), reportVoList);
    return new Result(ResultCode.SUCCESS);
}
```



```
@RequestMapping(value = "/historys/{year}/list", method = RequestMethod.GET)
public Result historiesList(@PathVariable(value = "year") String year) throws
Exception {
    List<Archive> archiveVoList = new ArrayList<>();
    List<String> yearMonths = Dateutil.getYearMonths();
    for (String yearMonth : yearMonths) {
        Archive archive = archiveService.findArchive(companyId, yearMonth);
        if (archive == null) {
            archive = new Archive("0", companyId, yearMonth, null, new
BigDecimal(0), new BigDecimal(0), new BigDecimal(0));
        }
        archiveVoList.add(archive);
    }
    return new Result(ResultCode.SUCCESS, archiveVoList);
}
```

### 3.7.3 历史归档详情

```
@RequestMapping(value = "/historys/{yearMonth}", method = RequestMethod.GET)
public Result histories(@PathVariable(value = "yearMonth") String
yearMonth,@RequestParam(value = "opType") Integer opType) {
    List<ArchiveDetail> reportVoList = new ArrayList<>();
    if (opType == 1) {
        reportVoList.addAll(archiveService.getReports(yearMonth,companyId));
    } else {
        Archive archive = archiveService.findArchive(companyId, yearMonth);
        if (archive != null) {
            reportVoList =
archiveservice.findAllDetailByArchiveId(archive.getId());
        }
    }
    return new Result(ResultCode.SUCCESS, reportVoList);
}
```