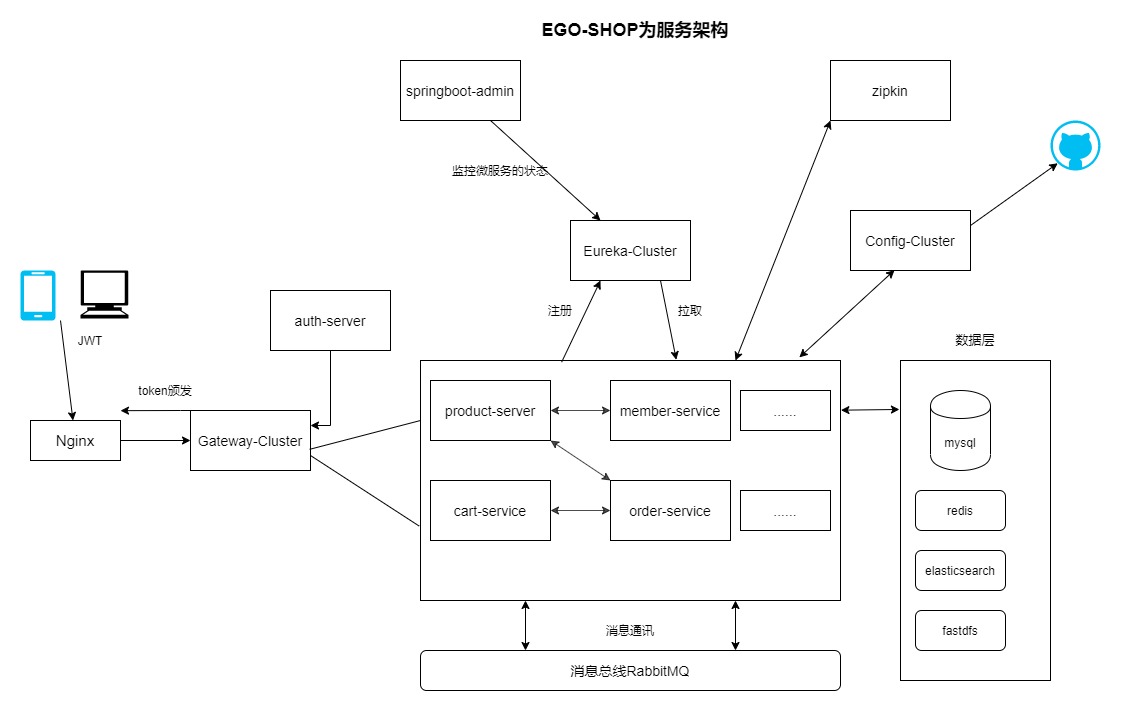
# 项目搭建

## 模块划分



## 架构图（手画出来）



## 搭建和运行

### 有的公司已经搭建完毕，怎么跑起来项目 （会让你重构）注意依赖和jar包一定要下载完成

配置文件的修改即可，注意项目启动顺序：

注册中心->配置文件中心->网关-->授权中心-->微服务--->......

### 有的公司没有搭建出来，你去了让你搭建，主导开发

今天好好看，好好学，把代码搞规范，注意包名，你就可以搭建了，而且可以主导开发

### 复习集群的搭建（面试可能会问到）

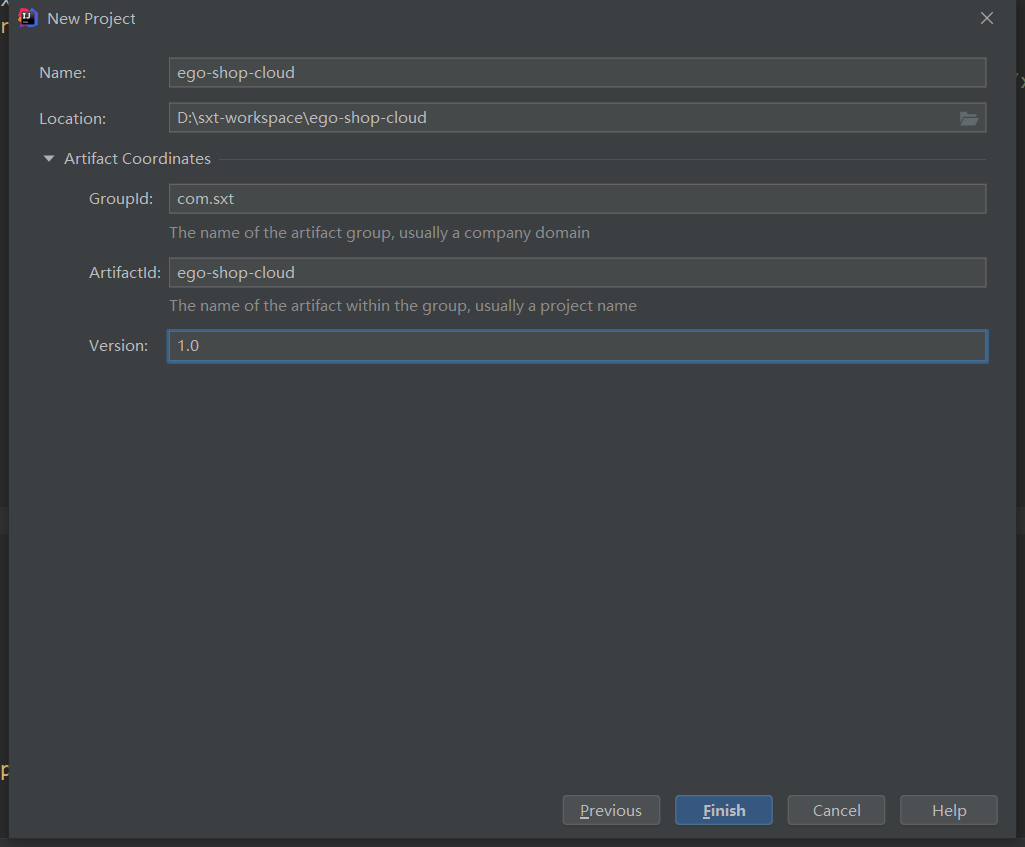
Mysql主写从读 ，两个数据源，aop 注解 master salver， insert，update（@master）

### 准备好工作区间，我搭建好了后，你也搭建一遍，搞好之后，压缩包放在一起

## 开始搭建

### ego-shop-cloud父项目搭建(pom)

#### 新建一个maven项目(ego-shop-cloud)

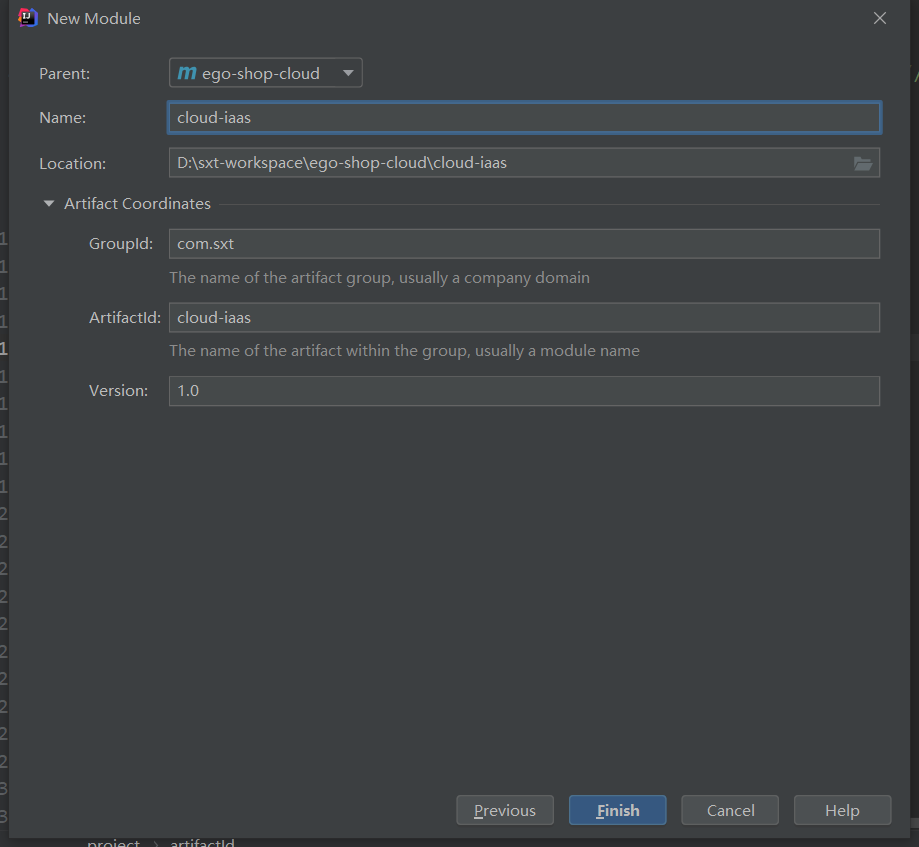


#### 修改pom.xml文件

|  |
| --- |
| <?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">  <modelVersion>4.0.0</modelVersion>  <packaging>pom</packaging>  <!--父项目还是boot项目-->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.3.3.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.sxt</groupId>  <artifactId>ego-shop-cloud</artifactId>  <version>1.0</version>  <!-- 依赖版本管理-->  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>Hoxton.SR8</spring-cloud.version>  <spring-boot-admin.version>2.3.0</spring-boot-admin.version>  <mysql-connector.version>8.0.19</mysql-connector.version>  <mybatis-plus.version>3.2.0</mybatis-plus.version>  <druid.starter.version>1.1.21</druid.starter.version>  <swagger.version>3.0.0</swagger.version>  <hutool.version>5.3.9</hutool.version>  <fastDFS.version>1.27.2</fastDFS.version>  <fastJson.version>1.2.73</fastJson.version>  </properties>  <!-- 全局都需要的依赖-->  <dependencies>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency>  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-boot-starter</artifactId>  <version>${swagger.version}</version>  </dependency>  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-annotation</artifactId>  <version>${mybatis-plus.version}</version>  </dependency>  <dependency>  <groupId>cn.hutool</groupId>  <artifactId>hutool-all</artifactId>  <version>${hutool.version}</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>${fastJson.version}</version>  </dependency>  </dependencies>  <!-- 依赖管理-->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <dependency>  <groupId>de.codecentric</groupId>  <artifactId>spring-boot-admin-dependencies</artifactId>  <version>${spring-boot-admin.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  <version>${druid.starter.version}</version>  </dependency>  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>${mybatis-plus.version}</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql-connector.version}</version>  </dependency>  </dependencies>  </dependencyManagement>  </project> |

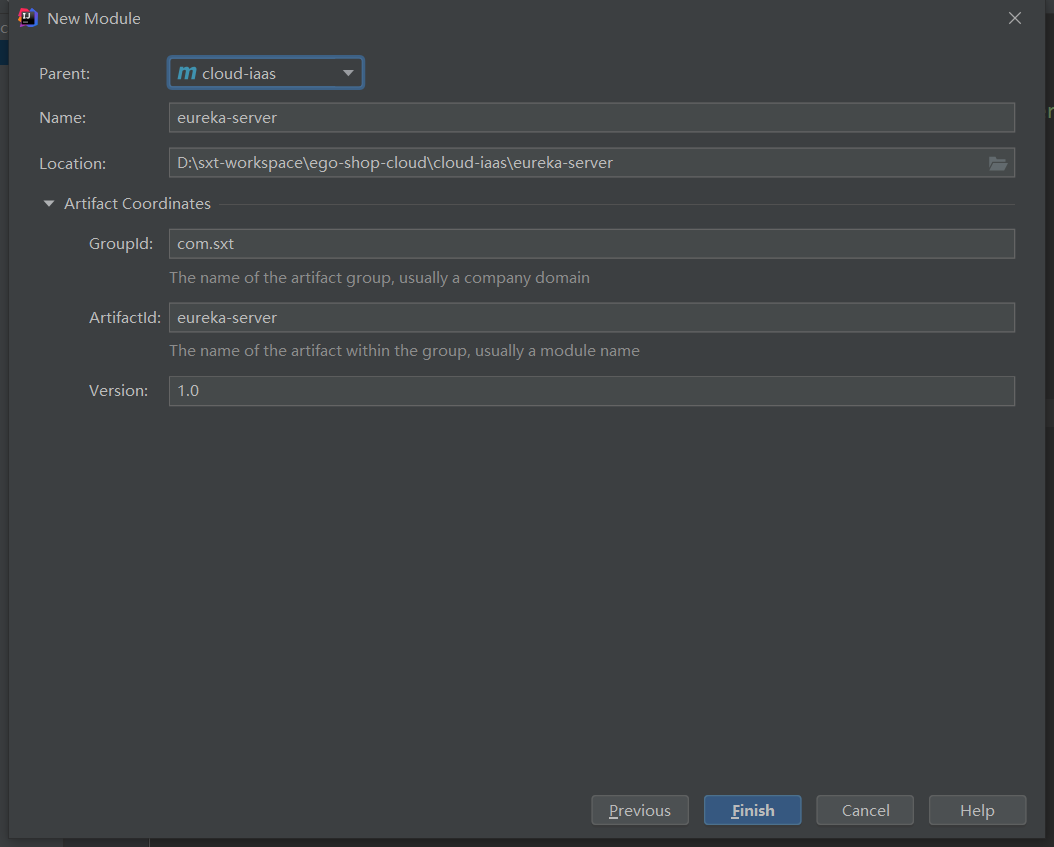
### Spring Cloud基础设施搭建

#### 在ego-shop-cloud里面新建一个cloud-iaas模块（pom）



#### cloud-**iaas**里面搭建eureka-server

##### 新建模块



##### 修改eureka-server的pom.xml

|  |
| --- |
| <?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>cloud-iaas</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>eureka-server</artifactId>  <description>ego商城注册中心</description>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  </dependency>  <!-- 保护起来,访问需要账号密码-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  </dependencies>  <!-- maven的打包插件-->  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

##### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaServer //开启eureka注册中心  public class EurekaServerApplication {  public static void main(String[] args) {  SpringApplication.run(EurekaServerApplication.class, args);  System.out.println(">>>>>>>>>>eureka-server启动完成<<<<<<<<<<<<<<");  }  } |

##### 添加application.yml配置文件

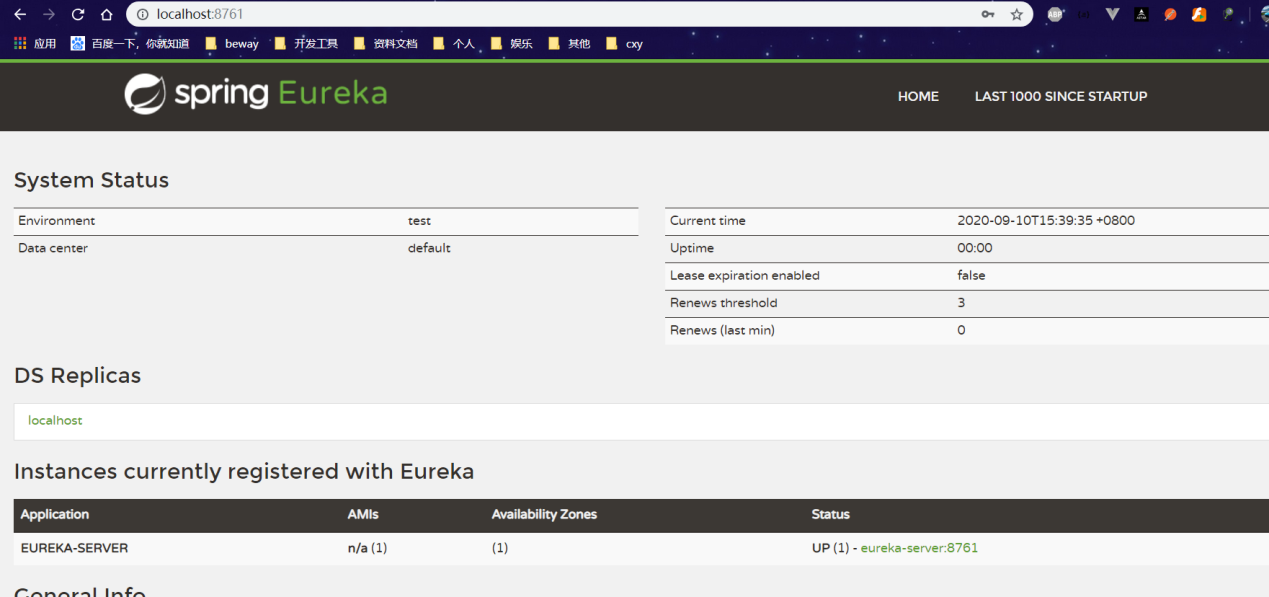
|  |
| --- |
| server:  port: ${APP\_PORT:8761}  spring:  application:  name: eureka-server  security: #配置eureka的访问登录用户  user:  name: admin  password: admin  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  fetch-registry: true  register-with-eureka: true  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-expiration-duration-in-seconds: 30 #30s客户端没发心跳就剔除  lease-renewal-interval-in-seconds: 10 #客户端向服务端发送心跳的时间  management:  endpoints:  web:  exposure:  include: '\*' #暴露的端点放行，例如健康监测的 |

##### 添加配置类

|  |
| --- |
| @Configuration  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  /\*\*  \* http请求的配置  \*  \* @param http  \* @throws Exception  \*/  @Override  protected void configure(HttpSecurity http) throws Exception {  //关闭跨站请求攻击  http.csrf().disable();  //健康监测的请求放行  http.authorizeRequests().antMatchers("/actuator/\*\*").permitAll();  super.configure(http);  }  } |

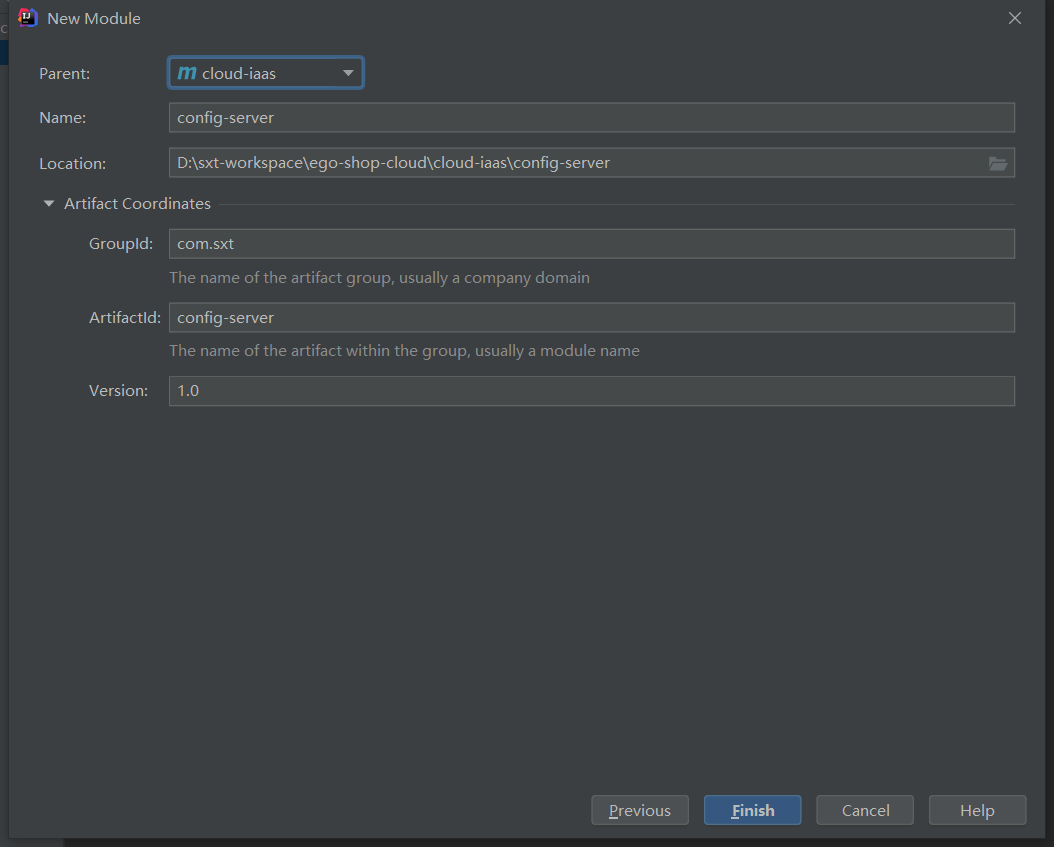
##### 运行测试访问，需要登录

<http://localhost:8761/>



#### cloud-**iaas**里面搭建config-server

##### 新建模块



##### 修改pom.xml

|  |
| --- |
| <?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>cloud-iaas</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>config-server</artifactId>  <description>ego商城的配置文件中</description>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-server</artifactId>  </dependency>  <!-- 添加bus和mq的集成依赖-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

##### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient //eureka的客户端  @EnableConfigServer //开启配置文件中心  public class ConfigServerApplication {  public static void main(String[] args) {  SpringApplication.run(ConfigServerApplication.class, args);  }  /\*\*  \* 发请求  \*  \* @return  \*/  @Bean  public RestTemplate restTemplate() {  return new RestTemplate();  }  } |

##### 添加配置文件

|  |
| --- |
| server:  port: ${APP\_PORT:8888}  spring:  application:  name: config-server  cloud:  config:  discovery:  enabled: true #开启配置文件发现功能  server:  git: #配置远端配置文件地址 用户名和密码以及文件夹名称  uri: https://gitee.com/smiledouble/ego-shop-config.git  username: smiledouble  password: \*\*\*\*\*  search-paths: /ego-shop-cloud/\*\*  rabbitmq: #配置rabbitmq  addresses: 192.168.226.129  username: admin  password: admin  port: 5672  management:  endpoints:  web:  exposure:  include: '\*'  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  fetch-registry: true  register-with-eureka: true  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-expiration-duration-in-seconds: 30 #30s客户端没发心跳就剔除  lease-renewal-interval-in-seconds: 10 #客户端向服务端发送心跳的时间 |

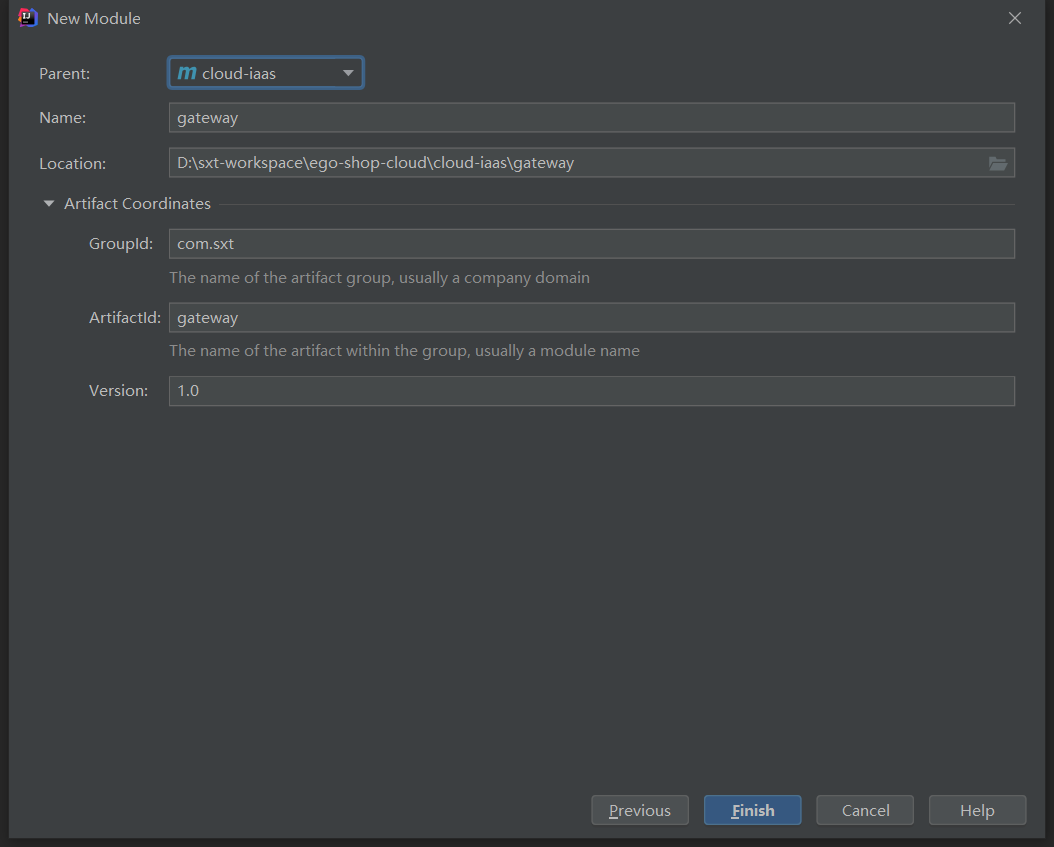
##### 添加动态刷新类

|  |
| --- |
| @RestController  public class ConfigController {  //端口号，实现负载均衡  @Value("${server.port}")  private Integer port;  @Autowired  private RestTemplate restTemplate;  /\*\*  \* 钩子函数会触发我们这个接口，发请求刷新所有的配置文件  \*  \* @return  \*/  @PostMapping("bus-refresh")  public ResponseEntity<String> refresh() {  //组装url  String url = "http://localhost:" + port + "/actuator/bus-refresh";  //创建请求头  HttpHeaders httpHeaders = new HttpHeaders();  httpHeaders.add("content-type", "application/json");  HttpEntity<String> stringHttpEntity = new HttpEntity<>(httpHeaders);  //发post请求  ResponseEntity<String> responseEntity = restTemplate.postForEntity(url, stringHttpEntity, String.class);  if (responseEntity.getStatusCode().equals(HttpStatus.NO\_CONTENT)) {  return ResponseEntity.ok("刷新配置文件成功");  }  return ResponseEntity.badRequest().body("刷新失败");  }  } |

##### 记得添加Gitee的仓库和WebHooks

#### cloud-**iaas**里面搭建gateway

##### 新建模块



##### 修改pom.xml

|  |
| --- |
| <?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>cloud-iaas</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>gateway</artifactId>  <description>ego商城的网关</description>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-gateway</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <!--限流要引入Redis-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

##### 添加启动类

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

##### 添加bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: gateway  cloud:  config: #配置文件的信息  label: master  name: gateway  profile: dev  discovery:  enabled: true  service-id: config-server  gateway:  discovery:  locator:  enabled: true #开启网关的动态路由  lower-case-service-id: true  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  fetch-registry: true  register-with-eureka: true  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-expiration-duration-in-seconds: 30 #30s客户端没发心跳就剔除  lease-renewal-interval-in-seconds: 10 #客户端向服务端发送心跳的时间 |

##### 添加云端gateway/gateway-dev.yml配置文件

|  |
| --- |
| server:  port: ${APP\_PORT:80}  spring:  cloud:  gateway:  routes: #订单服务抢购多，对下订单实行限流即可  - id: order-service-router  uri: lb://order-service  predicates:  - After=2020-10-01T17:42:47.789-07:00[Asia/Shanghai]  filters:  - name: RequestRateLimiter  args:  key-resolver: '#{@hostAddrKeyResolver}'  redis-rate-limiter.replenishRate: 10  redis-rate-limiter.burstCapacity: 20  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*' |

##### 添加限流

|  |
| --- |
| @Configuration  public class RequestRateLimitConfig {  /\*\*  \* ip限流  \*  \* @return  \*/  @Bean("hostAddrKeyResolver")  @Primary  public KeyResolver hostAddrKeyResolver() {  return exchange -> Mono.just(exchange.getRequest().getRemoteAddress().getHostName());  }  /\*\*  \* 用户id限流  \*  \* @return  \*/  @Bean("userIdKeyResolver")  public KeyResolver userIdKeyResolver() {  return exchange -> Mono.just(exchange.getRequest().getQueryParams().getFirst("userId"));  }  /\*\*  \* api接口限流  \*  \* @return  \*/  @Bean("apiKeyResolver")  public KeyResolver apiKeyResolver() {  return exchange -> Mono.just(exchange.getRequest().getPath().value());  }  } |

##### 添加jwt检测

|  |
| --- |
| @Component  public class CheckJwtFilter implements GlobalFilter, Ordered {  @Autowired  private StringRedisTemplate redisTemplate;  @Override  public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  String path = exchange.getRequest().getURI().getPath();  //得到请求的路径 如果是登录之类的就放行，如果不是就检查是否有token  if (GatewayConstant.ALLOW\_PATH.contains(path)) {  //放行  return chain.filter(exchange);  }  //从请求头里面拿到jwt  HttpHeaders headers = exchange.getRequest().getHeaders();  List<String> list = headers.get(GatewayConstant.AUTHORIZATION);  if (!ObjectUtils.isEmpty(list)) {  //如果list不是空  String auth = list.get(0);  String authorization = auth.replaceAll("bearer ", "");  if (!StringUtils.isEmpty(authorization) && redisTemplate.hasKey(GatewayConstant.OAUTH\_PREFIX + authorization)) {  //如果有jwt 就看和redis里面的是不是一样的  return chain.filter(exchange);  }  }  //这里就是没有jwt了，返回401  ServerHttpResponse response = exchange.getResponse();  response.getHeaders().add("content-type", "application/json;charset=utf-8");  Map<String, Object> map = new HashMap<>();  map.put("code", HttpStatus.UNAUTHORIZED.value());  map.put("msg", "非法访问");  ObjectMapper objectMapper = new ObjectMapper();  byte[] bytes = null;  DataBuffer buffer = null;  try {  bytes = objectMapper.writeValueAsBytes(map);  buffer = response.bufferFactory().wrap(bytes);  } catch (JsonProcessingException e) {  e.printStackTrace();  }  return response.writeWith(Mono.just(buffer));  }  /\*\*  \* 执行顺序 越小越先 要比-1小  \*  \* @return  \*/  @Override  public int getOrder() {  return -2;  }  } |

##### 添加授权存储jwt路由

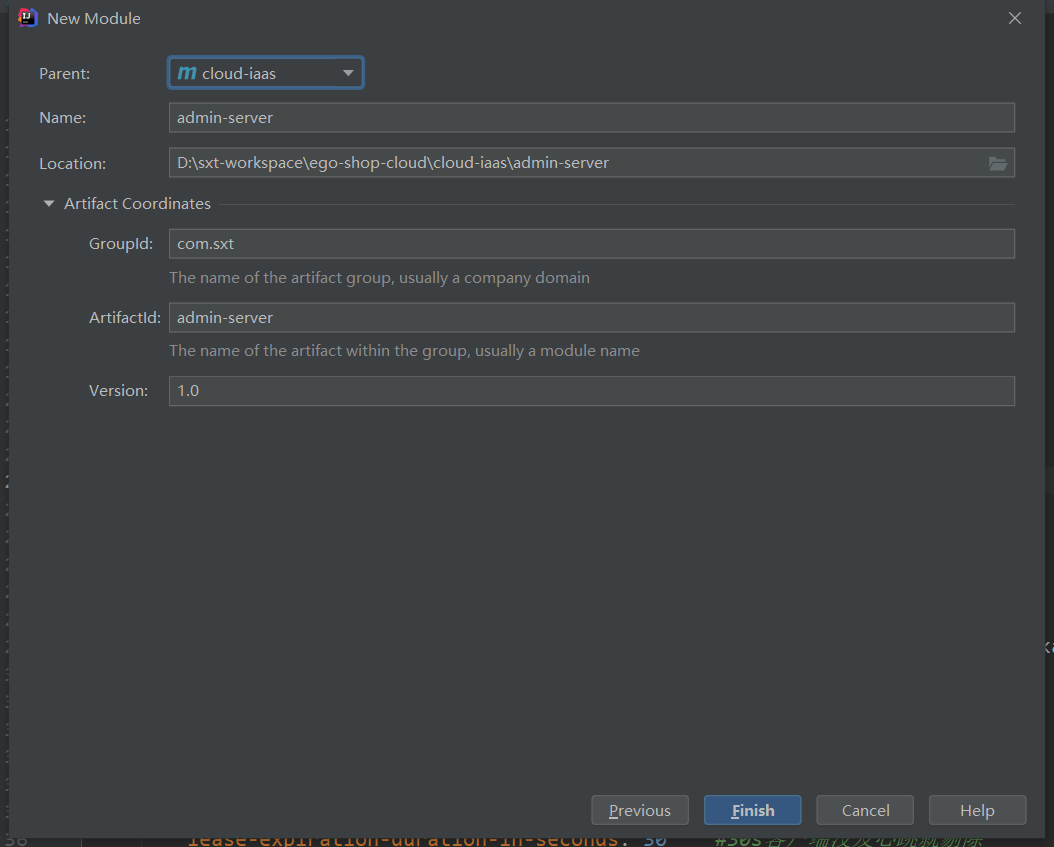
|  |
| --- |
| @Configuration  public class GatewayConfig {  @Autowired  private StringRedisTemplate redisTemplate;  /\*\*  \* 描述: 给授权专门做的存入token路由  \*  \* @param builder:  \* @return org.springframework.cloud.gateway.route.RouteLocator  \*/  @Bean  public RouteLocator routeLocator(RouteLocatorBuilder builder) {  return builder  .routes()  .route("auth-server-router", r -> r.path("/oauth/\*\*")  .filters(f -> f.modifyResponseBody(String.class, String.class, (exchanges, s) -> {  String path = exchanges.getRequest().getURI().getPath();  if ("/oauth/token".equals(path)) {  //如果是登录的请求，那么得到的s就是token的一套  JSONObject tokenObject = JSONUtil.parseObj(s);  if (tokenObject.containsKey("access\_token")) {  //如果包含access\_token就放进redis里面  String access\_token = tokenObject.getStr("access\_token");  Long expires\_in = tokenObject.getLong("expires\_in");  redisTemplate  .opsForValue()  .set(GatewayConstant.OAUTH\_PREFIX + access\_token, "", Duration.ofSeconds(expires\_in)  );  }  }  return Mono.just(s);  })).uri("lb://auth-server"))  .build();  }  } |

##### 添加常量类

|  |
| --- |
| public class GatewayConstant {  public static final String OAUTH\_PREFIX = "oauth:jwt";  public static final String AUTHORIZATION = "Authorization";  public static final List<String> ALLOW\_PATH = Arrays.asList("/oauth/token", "/auth-server/oauth/token");  } |

#### cloud-**iaas**里面搭建admin-server(监控)

##### 新建模块



##### 修改pom.xml

|  |
| --- |
| <?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>cloud-iaas</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>admin-server</artifactId>  <description>ego商城微服务的监控</description>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <dependency>  <groupId>de.codecentric</groupId>  <artifactId>spring-boot-admin-starter-server</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

##### 添加启动类

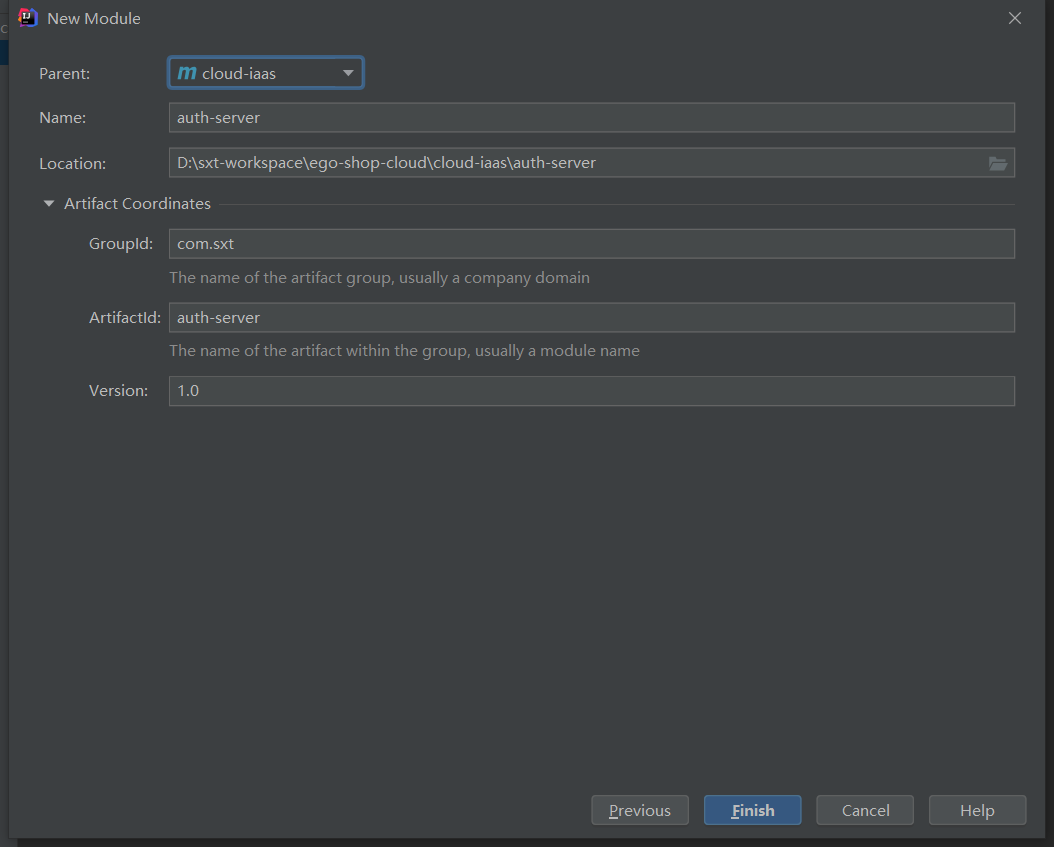
|  |
| --- |
| @SpringBootApplication  @EnableAdminServer //开启监控中心  public class AdminServerApplication {  public static void main(String[] args) {  SpringApplication.run(AdminServerApplication.class, args);  }  } |

##### 添加application.yml配置文件

|  |
| --- |
| server:  port: ${APP\_PORT:10010}  spring:  application:  name: admin-server  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-renewal-interval-in-seconds: 10  lease-expiration-duration-in-seconds: 30  management:  endpoints:  web:  exposure:  include: '\*' |

#### cloud-**iaas**里面搭建auth-server(授权)

##### 新建模块



##### 修改pom.xml

|  |
| --- |
| <?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>cloud-iaas</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>auth-server</artifactId>  <description>ego商城的授权中心</description>  <dependencies>  <!-- 依赖ego-common-->  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

##### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableAuthorizationServer //开启授权中心  @EnableEurekaClient  public class AuthServerApplication {  public static void main(String[] args) {  SpringApplication.run(AuthServerApplication.class, args);  }  /\*\*  \* 加密方式  \*  \* @return  \*/  @Bean  public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder();  }  } |

##### 添加bootstrap.yml配置文件

|  |
| --- |
| eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka/}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-renewal-interval-in-seconds: 10  lease-expiration-duration-in-seconds: 30  spring:  application:  name: auth-server  cloud:  config:  label: master  name: auth-server  profile: dev  discovery:  enabled: true  service-id: config-server |

##### 添加云端的配置文件auth-server/auth-server.yml

|  |
| --- |
| server:  port: ${APP\_PORT:9999}  spring:  datasource:  druid:  url: jdbc:mysql://localhost:3306/ego-shop?useSSL=false&serverTimezone=GMT%2B8  username: root  password: root  driver-class-name: com.mysql.cj.jdbc.Driver  max-active: 30  min-idle: 3  initial-size: 3  validation-query: select 1  max-wait: 5000  stat-view-servlet:  enabled: true  login-username: admin  login-password: admin  allow:  deny:  url-pattern:  - /druid/\*  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  management:  endpoints:  web:  exposure:  include: '\*'  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl |

##### 添加WebSecurityConfig类

|  |
| --- |
| @Configuration  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  @Autowired  private UserDetailServiceImpl userDetailService;  /\*\*  \* 认证管理器  \*  \* @return  \* @throws Exception  \*/  @Bean  public AuthenticationManager authenticationManager() throws Exception {  return super.authenticationManager();  }  @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  //走自己的登录  auth.userDetailsService(userDetailService);  }  } |

##### 添加AuthorizationConfig类

|  |
| --- |
| @Configuration  public class AuthorizationConfig extends AuthorizationServerConfigurerAdapter {  @Autowired  private PasswordEncoder passwordEncoder;  @Autowired  private UserDetailServiceImpl userDetailService;  @Autowired  private AuthenticationManager authenticationManager;  /\*\*  \* 使用jwt存放token  \*  \* @return  \*/  // @Bean  public JwtTokenStore jwtTokenStore() {  return new JwtTokenStore(jwtAccessTokenConverter());  }  /\*\*  \* 我们使用非对称加密的方式  \*  \* @return  \*/  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  //将文件加载进来  ClassPathResource resource = new ClassPathResource("cxs-jwt.jks");  //得到内容  KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(resource, "cxs123".toCharArray());  //转换  KeyPair keyPair = keyStoreKeyFactory.getKeyPair("cxs-jwt");  //放入转换器  jwtAccessTokenConverter.setKeyPair(keyPair);  return jwtAccessTokenConverter;  }  /\*\*  \* 描述: 一个web平台，用于第三方访问  \* 一个sxt平台，内部访问的，例如mq里发起远程调用  \*  \* @param clients:  \* @return void  \*/  @Override  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  clients.inMemory()  .withClient("web")  .secret(passwordEncoder.encode("web-secret"))  .scopes("read")  .authorizedGrantTypes("password")  .accessTokenValiditySeconds(7200)  .and()  .withClient("sxt")  .secret(passwordEncoder.encode("sxt-secret"))  .scopes("write", "read")  .authorizedGrantTypes("client\_credentials")  .accessTokenValiditySeconds(Integer.MAX\_VALUE);  super.configure(clients);  }  /\*\*  \* 端点配置，使用自己的UserService，使用jwt  \*  \* @param endpoints  \* @throws Exception  \*/  @Override  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  endpoints.userDetailsService(userDetailService)  .authenticationManager(authenticationManager)  .tokenStore(jwtTokenStore())  .accessTokenConverter(jwtAccessTokenConverter());  super.configure(endpoints);  }  } |

##### 添加UserDetailServiceImpl类

|  |
| --- |
| @Service  public class UserDetailServiceImpl implements UserDetailsService {  @Autowired  private UserDetailMapper userDetailMapper;  @Autowired  private SysUserDetailMapper sysUserDetailMapper;  @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = requestAttributes.getRequest();  String loginType = request.getHeader(AuthConstant.LOGIN\_TYPE);  if (StringUtils.isEmpty(loginType)) {  return null;  }  switch (loginType) {  case AuthConstant.USER\_TYPE:  //如果是前台用户 直接插数据库    break;  case AuthConstant.SYS\_USER\_TYPE:  //如果是后台用户，要查询权限  SysUserDetail sysUserDetail = sysUserDetailMapper.getOne(new LambdaQueryWrapper<SysUserDetail>()  .eq(SysUserDetail::getUsername, username)  .eq(SysUserDetail::getStatus, 1)  );  if (!ObjectUtils.isEmpty(sysUserDetail)) {  //查询权限  List<String> authorization = getUserPermission(sysUserDetail.getUserId());  if (!ObjectUtils.isEmpty(authorization)) {  sysUserDetail.setPermissions(authorization);  }  }  return sysUserDetail;  default:  }  return null;  }  /\*\*  \* 查询权限的方法  \*  \* @param userId  \* @return  \*/  private List<String> getUserPermission(Long userId) {  HashSet<String> actualPermissions = new HashSet<>();  //得到权限  List<String> permission = sysUserDetailMapper.getPermissionByUserId(userId);  if (ObjectUtils.isEmpty(permission)) {  return Collections.emptyList();  }  //处理逗号分隔  permission.forEach(p -> {  String[] complexPermission = p.split(",");  for (String s : complexPermission) {  actualPermissions.add(s);  }  });  return new ArrayList<>(actualPermissions);  }  } |

##### 添加AuthConstant类

|  |
| --- |
| public class AuthConstant {  //登录类型  public static final String LOGIN\_TYPE = "login\_type";  //前台用户  public static final String USER\_TYPE = "user";  //后台用户  public static final String SYS\_USER\_TYPE = "sysUser";  } |

##### 逆向生成user表和sys\_user表（用于登录的）

因为登录只需要user表里面的id和状态字段，所以创建一个UserDetail类

因为登录只需要sys\_user表里面的id和状态以及权限，所以创建一个SysUserDetail类

但是很多地方需要获得当前的用户信息，所以我们将UserDetail类和SysUserDetail类提升到ego-common里面，我在ego-common模块中详细贴出来代码

因为需要用到mybatisplus查询，所以将生成的mapper，service和serviceImpl中的继承泛型修改为对应的detail类即可，这里贴出分包细节

##### 贴出UserDetailMapper

|  |
| --- |
| public interface UserDetailMapper extends BaseMapper<UserDetail> {  } |

##### 贴出SysUserDetailMapper

|  |
| --- |
| public interface SysUserDetailMapper extends BaseMapper<SysUserDetail> {  //查询后台用户的权限  @Select(" select DISTINCT perms from sys\_user\_role t1 join sys\_role\_menu t2 on(t1.role\_id=t2.role\_id) join sys\_menu t3 on(t2.menu\_id=t3.menu\_id) where t1.user\_id = #{userId} and t3.type = 2 ")  List<String> getPermissionByUserId(Long userId);  } |

##### 贴出UserDetailService

|  |
| --- |
| public interface UserDetailService extends IService<UserDetail>{  } |

##### 贴出SysUserDetailService

|  |
| --- |
| public interface SysUserDetailService extends IService<SysUserDetail> {  } |

##### 贴出SysUserDetailServiceImpl

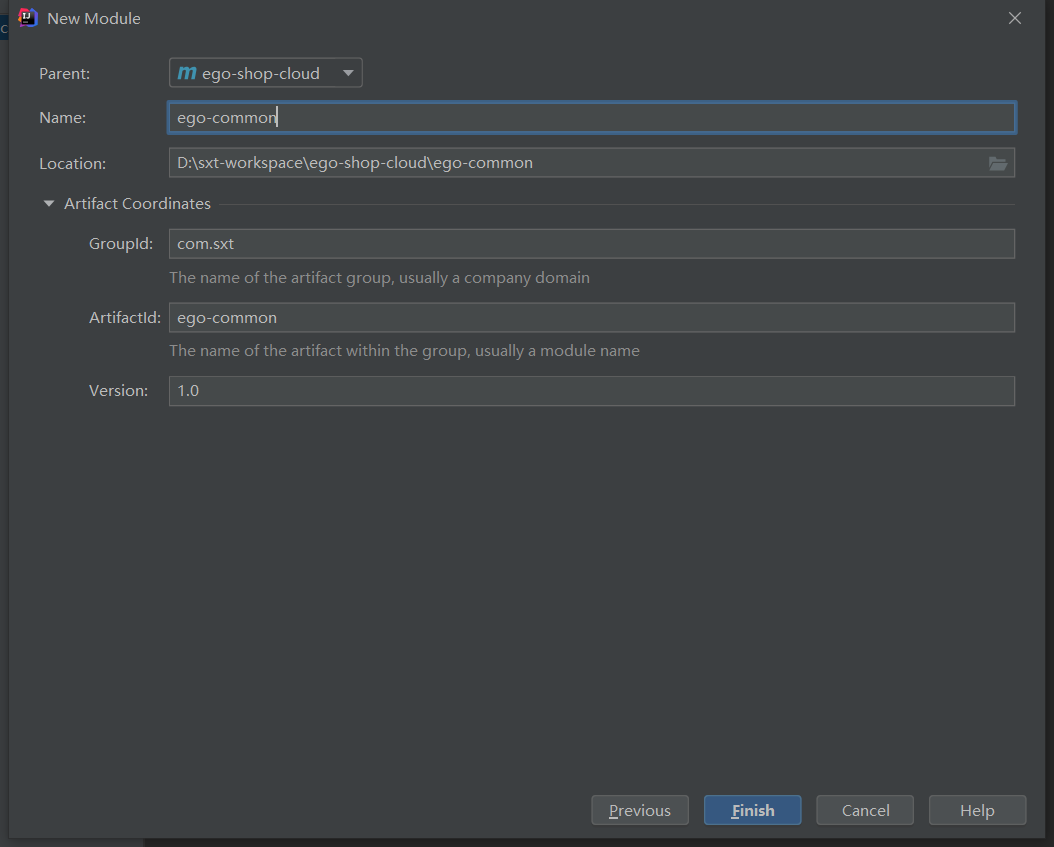
|  |
| --- |
| @Service  public class SysUserDetailServiceImpl extends ServiceImpl<SysUserDetailMapper, SysUserDetail> implements SysUserDetailService {  } |

##### 将生成的jwt私钥放在resources里面

#### ego-shop-cloud里面搭建ego-common(公共模块)

比如一些公用的配置或者一些工具 放在common里面 增强代码的复用性和可维护性

##### 新建模块



##### 修改pom.xml

|  |
| --- |
| <?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>ego-shop-cloud</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>ego-common</artifactId>  <description>ego的公共模块，重要的一点是资源服务器的统一验证jwt</description>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency>  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>${mybatis-plus.version}</version>  </dependency>  </dependencies>  </project> |

##### 添加ResourceConfig类（统一解析token）

|  |
| --- |
| @Configuration  @EnableGlobalMethodSecurity(prePostEnabled = true) //开启方法级别的验证  @EnableResourceServer //开启资源服务器授权  public class ResourceConfig extends ResourceServerConfigurerAdapter {  @Bean  public JwtTokenStore tokenStore() {  return new JwtTokenStore(jwtAccessTokenConverter());  }  @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  //得到文件  ClassPathResource resource = new ClassPathResource("jwt-publicKey.txt");  try {  String publicKey = FileUtil.readString(resource.getFile(), Charset.defaultCharset());  //验证jwt  jwtAccessTokenConverter.setVerifierKey(publicKey);  } catch (IOException e) {  e.printStackTrace();  }  return jwtAccessTokenConverter;  }  @Override  public void configure(ResourceServerSecurityConfigurer resources) throws Exception {  resources.tokenStore(tokenStore());  super.configure(resources);  }  @Override  public void configure(HttpSecurity http) throws Exception {  http.csrf().disable()  .sessionManagement().disable(); // 基于token，所以不需要session  http.authorizeRequests().antMatchers(  "/v2/api-docs",  "/v3/api-docs",  "/swagger-resources/configuration/ui", //用来获取支持的动作  "/swagger-resources", //用来获取api-docs的URI  "/swagger-resources/configuration/security",//安全选项  "/webjars/\*\*",  "/swagger-ui/\*\*",  "/druid/\*\*",  "/actuator/\*\*"  ).permitAll()  .antMatchers("/\*\*").authenticated()  .and().headers().cacheControl();  }  } |

##### 添加UserDetail类（auth-server中提到的）

|  |
| --- |
| @Data  @AllArgsConstructor  @NoArgsConstructor  @TableName(value = "`user`")  public class UserDetail implements UserDetails, Serializable {  @TableId(value = "user\_id", type = IdType.INPUT)  private String userId;  @TableField(value = "status")  private Integer status;  @Override  public Collection<? extends GrantedAuthority> getAuthorities() {  return Collections.EMPTY\_LIST;  }  //WECHAT字符串的加密后  @Override  public String getPassword() {  return "$2a$10$xM/zITVckVuQBALYT4XqhOHS/n18u8tW8CEynDoYQJCmt2PsH6Rnq";  }  @Override  public String getUsername() {  return this.userId;  }  @Override  public boolean isAccountNonExpired() {  return this.status == 1;  }  @Override  public boolean isAccountNonLocked() {  return this.status == 1;  }  @Override  public boolean isCredentialsNonExpired() {  return this.status == 1;  }  @Override  public boolean isEnabled() {  return this.status == 1;  }  } |

##### 添加SysUserDetail类（auth-server中提到的）

|  |
| --- |
| @Data  @AllArgsConstructor  @NoArgsConstructor  @TableName(value = "sys\_user")  public class SysUserDetail implements UserDetails, Serializable {  @TableId(value = "user\_id", type = IdType.AUTO)  private Long userId;  /\*\*  \* 用户名  \*/  @TableField(value = "username")  private String username;  /\*\*  \* 密码  \*/  @TableField(value = "password")  private String password;  /\*\*  \* 状态 0：禁用 1：正常  \*/  @TableField(value = "status")  private Integer status;  @TableField(exist = false)  List<String> permissions = new ArrayList<>(0);  @Override  public Collection<? extends GrantedAuthority> getAuthorities() {  ArrayList<SimpleGrantedAuthority> authorities = new ArrayList<>();  if (!ObjectUtils.isEmpty(permissions.size())) {  permissions.forEach(p -> {  authorities.add(new SimpleGrantedAuthority(p));  });  }  return authorities;  }  //因为username不常用 直接把数据库的用户id放进jwt里面  @Override  public String getUsername() {  return String.valueOf(userId);  }  @Override  public boolean isAccountNonExpired() {  return this.status == 1;  }  @Override  public boolean isAccountNonLocked() {  return this.status == 1;  }  @Override  public boolean isCredentialsNonExpired() {  return this.status == 1;  }  @Override  public boolean isEnabled() {  return this.status == 1;  }  } |

##### 添加MybatisPlusConfig类

|  |
| --- |
| @Configuration  @MapperScan(basePackages = {"com.sxt.mapper"})  public class MybatisPlusConfig {  /\*\*  \* mybatis的分页插件  \*  \* @return  \*/  @Bean  public PaginationInterceptor paginationInterceptor() {  return new PaginationInterceptor();  }  } |

##### 添加Swagger配置类

|  |
| --- |
| @Data  @ConfigurationProperties(prefix = "swagger2")  public class SwaggerProperties {  /\*\*  \* 要对那个包的Controller 做文档的生成  \*/  private String basePackage;  /\*\*  \* 作者的名称  \*/  private String name ;  /\*\*  \* 主页  \*/  private String url;  /\*\*  \* 邮箱  \*/  private String email ;  /\*\*  \* 标题  \*/  private String title ;  /\*\*  \* 描述  \*/  private String description ;  /\*\*  \* 服务的团队  \*/  private String termsOfServiceUrl;  /\*\*  \* 授权信息  \*/  private String license ;  /\*\*  \* 授权的url  \*/  private String licenseUrl ;  } |

##### 添加SwaggerAutoConfiguration

|  |
| --- |
| @Configuration  @EnableOpenApi // 开启swagger的功能  @EnableConfigurationProperties(SwaggerProperties.class)  public class SwaggerAutoConfiguration {  @Autowired  private SwaggerProperties swaggerProperties;  /\*\*  \* 创建swagger文档  \*  \* @return  \*/  @Bean  public Docket docket() {  return new Docket(DocumentationType.OAS\_30).  apiInfo(getApiInfo()).  select().apis(RequestHandlerSelectors.basePackage(swaggerProperties.getBasePackage())).build();  }  private ApiInfo getApiInfo() {  Contact contact = new Contact(swaggerProperties.getName(), swaggerProperties.getUrl(), swaggerProperties.getEmail());  return new ApiInfoBuilder().  contact(contact).  title(swaggerProperties.getTitle()).  description(swaggerProperties.getDescription()).  termsOfServiceUrl(swaggerProperties.getTermsOfServiceUrl()).  license(swaggerProperties.getLicense()).  licenseUrl(swaggerProperties.getLicenseUrl()).  build();  }  } |

##### 添加OauthFeignConfig（重点）

|  |
| --- |
| @Configuration  public class OauthFeignConfig implements RequestInterceptor {  /\*\*  \* 自己设置feign拦截器，在请求的时候设置token传递  \* （这是有前台请求由token 的时候，但是还有一种，例如内部直接调用，或者mq里面去调用，就没有token了）  \* 所以我们要搞一个全局的token，在前台没有token 的时候 带上这个token  \*  \* @param requestTemplate  \*/  @Override  public void apply(RequestTemplate requestTemplate) {  //得到请求  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  if (requestAttributes != null) {  //说明是前端请求，我们做token的传递  HttpServletRequest request = requestAttributes.getRequest();  String authorization = request.getHeader("Authorization");  if (!StringUtils.isEmpty(authorization)) {  //如果请求头有token，就带上  requestTemplate.header("Authorization", request.getHeader("Authorization"));  } else {  //如果是支付宝的回调，也是有请求的 但是没有token  requestTemplate.header("Authorization", "bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzY29wZSI6WyJ3cml0ZSIsInJlYWQiXSwiZXhwIjozNzQ4NjUzMzQ1LCJqdGkiOiJiNzliMDc4YS1lMGQzLTRjOTAtYTY2Ni01YmRkZjcyOWM1OWQiLCJjbGllbnRfaWQiOiJzeHQifQ.kWHxG0oNCJICpeAHn\_WWCAaJL4dzR8FDgt1De\_oM31iHMW29JK3g1BBQZIbJJ5H1FCbawBAH3sHzGKY2VeEiKxvud-1f-W9\_vNv6k5wq9wpG\_R8CRK-w3f3t\_B6QkKnmv4GtdVhPInjLS3A9hM5zQeTCURmZEggHXifAIPYfaffqrzJcrpXBAkvhoCywFQUp3XvfjCU86\_Q1TGkMkjQp9MZ4xnyCn2gP7H06-sAaphMMoz15nB-23oHLjKz\_hgbdLkt3QW63TIkwt26o1OChCh9FjG6P9EYkVXTCXUscPL4hEvXrHUmxKCsCWM6WEkl\_Dn2wPVOya5CFfZ3WkS009w");  }  } else {  //如果请求为空，说明是内部自己调用的，我们设置自己的永久token，这个token还需要放在redis里面  //我们用请求的方式拿到这个永久的token  requestTemplate.header("Authorization", "bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzY29wZSI6WyJ3cml0ZSIsInJlYWQiXSwiZXhwIjozNzQ4NjUzMzQ1LCJqdGkiOiJiNzliMDc4YS1lMGQzLTRjOTAtYTY2Ni01YmRkZjcyOWM1OWQiLCJjbGllbnRfaWQiOiJzeHQifQ.kWHxG0oNCJICpeAHn\_WWCAaJL4dzR8FDgt1De\_oM31iHMW29JK3g1BBQZIbJJ5H1FCbawBAH3sHzGKY2VeEiKxvud-1f-W9\_vNv6k5wq9wpG\_R8CRK-w3f3t\_B6QkKnmv4GtdVhPInjLS3A9hM5zQeTCURmZEggHXifAIPYfaffqrzJcrpXBAkvhoCywFQUp3XvfjCU86\_Q1TGkMkjQp9MZ4xnyCn2gP7H06-sAaphMMoz15nB-23oHLjKz\_hgbdLkt3QW63TIkwt26o1OChCh9FjG6P9EYkVXTCXUscPL4hEvXrHUmxKCsCWM6WEkl\_Dn2wPVOya5CFfZ3WkS009w");  }  }  } |

##### 创建FastDFS工具类

|  |
| --- |
| public class FastDFSUtil {  /\*\*  \* 解析fastDFS的路径参数  \*  \* @return  \*/  public static String parseGroup(String fullPath) {  String[] paths = fullPath.split("/");  if (paths.length <= 1) {  throw new RuntimeException("参数错误，无法解析");  }  return paths[0];  }  } |

##### 将jwt的公钥放到resources中

## 代码管理，版本控制

### 在Gitee上创建仓库，将代码推上去