**6-21 | 接口鉴权模块的开发**

**账户服务的职责**

负责对外界请求的token进行校验，需要频繁访问redis

**构建步骤**

新建账户服务

* qiyu-live-account-provider
* qiyu-live-account-interface

配置文件：

qiyu-live-account-provider引入的maven依赖：

|  |
| --- |
| XML <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/maven-v4\_0\_0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.idea</groupId>  <artifactId>qiyu-live-app</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>  <artifactId>qiyu-live-account-provider</artifactId>  <version>1.0.1</version>    <properties>  <qiyu-live-redis-starter.version>1.0-SNAPSHOT</qiyu-live-redis-starter.version>  <qiyu-live-account-interface.version>1.0-SNAPSHOT</qiyu-live-account-interface.version>  <qiyu-live-common-interface.version>1.0-SNAPSHOT</qiyu-live-common-interface.version>  </properties>   <dependencies>  <dependency>  <groupId>org.idea</groupId>  <artifactId>qiyu-live-account-interface</artifactId>  <version>${qiyu-live-account-interface.version}</version>  </dependency>  <dependency>  <groupId>org.idea</groupId>  <artifactId>qiyu-live-common-interface</artifactId>  <version>${qiyu-live-common-interface.version}</version>  </dependency>  <dependency>  <groupId>org.idea</groupId>  <artifactId>qiyu-live-framework-redis-starter</artifactId>  <version>${qiyu-live-redis-starter.version}</version>  </dependency>  <dependency>  <groupId>org.apache.dubbo</groupId>  <artifactId>dubbo-spring-boot-starter</artifactId>  <version>${dubbo.version}</version>  </dependency>  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bootstrap</artifactId>  <version>${spring-cloud-boostrap.version}</version>  </dependency>  </dependencies>   </project> |

新建bootstrap.yml配置文件：

|  |
| --- |
| YAML spring:  cloud:  nacos:  username: ${NACOS\_USER}  password: ${NACOS\_PWD}  discovery:  server-addr: qiyu.nacos.com:8848  namespace: qiyu-live-test  config:  import-check:  enabled: false  # 当前服务启动后去nacos中读取配置文件的后缀  file-extension: yaml  # 读取配置的nacos地址  server-addr: qiyu.nacos.com:8848  # 读取配置的nacos的名空间  namespace: qiyu-live-test  config:  import:  - optional:nacos:qiyu-live-account-provider.yaml |

nacos配置文件：

|  |
| --- |
| YAML spring:  application:  name: qiyu-live-account-provider  data:  redis:  port: 8801  host: cloud.db  password: qiyu  lettuce:  pool:  min-idle: 10  max-active: 100  max-idle: 10  dubbo:  application:  name: ${spring.application.name}  registry:  #docker启动的时候，注入host的配置  address: nacos://qiyu.nacos.com:8848?namespace=qiyu-live-test&&username=qiyu&&password=qiyu  protocol:  name: dubbo  port: 9090  threadpool: fixed  dispatcher: execution  threads: 500  accepts: 500 |

新建logback.xml配置文件：

|  |
| --- |
| XML <?xml version="1.0" encoding="UTF-8"?> <configuration>  <springProperty name="APP\_NAME" scope="context" source="spring.application.name" defaultValue="undefined"/>  <!-- 用于生成一个标识，防止多个Docker容器映射到同一台宿主机上出现目录名重复问题-->  <define name="index" class="org.qiyu.live.common.interfaces.utils.IpLogConversionRule"/>  <property name="LOG\_HOME" value="/tmp/logs/${APP\_NAME}/${index}"/>  <property name="LOG\_PATTERN" value="[%d{yyyy-MM-dd HH:mm:ss.SSS} -%5p] %-40.40logger{39} :%msg%n"/>   <!-- 控制台标准继续输出内容 -->  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">  <!-- 日志输出的格式 -->  <layout class="ch.qos.logback.classic.PatternLayout">  <pattern>${LOG\_PATTERN}</pattern>  </layout>  </appender>   <!-- info级别的日志，记录到对应的文件内 -->  <appender name="INFO\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  <file>${LOG\_HOME}/${APP\_NAME}.log</file>  <!-- 滚动策略，日志生成的时候会按照时间来进行分类，例如2023-05-11日的日志，后缀就会有2023-05-11，每天的日志归档后的名字都不一样 -->  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>${LOG\_HOME}/${APP\_NAME}.log.%d{yyyy-MM-dd}</fileNamePattern>  <!-- 日志只保留1个月 -->  <maxHistory>1</maxHistory>  </rollingPolicy>  <!-- 日志输出的格式 -->  <layout class="ch.qos.logback.classic.PatternLayout">  <pattern>${LOG\_PATTERN}</pattern>  </layout>  </appender>   <!-- error级别的日志，记录到对应的文件内 -->  <appender name="ERROR\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  <file>${LOG\_HOME}/${APP\_NAME}\_error.log</file>  <!-- 滚动策略，日志生成的时候会按照时间来进行分类，例如2023-05-11日的日志，后缀就会有2023-05-11，每天的日志归档后的名字都不一样 -->  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>${LOG\_HOME}/${APP\_NAME}\_error.log.%d{yyyy-MM-dd}</fileNamePattern>  <!-- 日志只保留1个月 -->  <maxHistory>1</maxHistory>  </rollingPolicy>  <!-- 日志输出的格式 -->  <layout class="ch.qos.logback.classic.PatternLayout">  <pattern>${LOG\_PATTERN}</pattern>  </layout>  <!-- 值记录error级别的日志 -->  <filter class="ch.qos.logback.classic.filter.LevelFilter">  <level>error</level>  <onMismatch>DENY</onMismatch>  </filter>  </appender>   <!-- 根输出级别为INFO，控制台中将出现包含info及以上级别的日志-->  <!-- 日志输出级别 -->  <root level="INFO">  <!-- ref值与上面的appender标签的name相对应 -->  <appender-ref ref="CONSOLE"/>  <appender-ref ref="INFO\_FILE"/>  <appender-ref ref="ERROR\_FILE"/>  </root> </configuration> |

定义账户服务的token校验接口；

|  |
| --- |
| Java public interface IAccountTokenRPC {    /\*\*  \* 创建一个登录token  \*  \* @param userId  \* @return  \*/  String createAndSaveLoginToken(Long userId);   /\*\*  \* 校验用户token  \*  \* @param tokenKey  \* @return  \*/  Long getUserIdByToken(String tokenKey);  } |

rpc层实现：

|  |
| --- |
| Java package org.qiyu.live.account.provider.rpc;  import jakarta.annotation.Resource; import org.apache.dubbo.config.annotation.DubboService; import org.qiyu.live.account.interfaces.IAccountTokenRPC; import org.qiyu.live.account.provider.service.IAccountTokenService;  /\*\*  \* @Author idea  \* @Date: Created in 10:18 2023/6/20  \* @Description  \*/ @DubboService public class AccountTokenRPCImpl implements IAccountTokenRPC {   @Resource  private IAccountTokenService accountTokenService;   @Override  public String createAndSaveLoginToken(Long userId) {  return accountTokenService.createAndSaveLoginToken(userId);  }   @Override  public Long getUserIdByToken(String tokenKey) {  return accountTokenService.getUserIdByToken(tokenKey);  }  } |

service接口：

|  |
| --- |
| Java package org.qiyu.live.account.provider.service;  /\*\*  \* @Author idea  \* @Date: Created in 10:21 2023/6/20  \* @Description  \*/ public interface IAccountTokenService {   /\*\*  \* 创建一个登录token  \*  \* @param userId  \* @return  \*/  String createAndSaveLoginToken(Long userId);   /\*\*  \* 校验用户token  \*  \* @param tokenKey  \* @return  \*/  Long getUserIdByToken(String tokenKey); } |

service层实现：

|  |
| --- |
| Java package org.qiyu.live.account.provider.service.impl;  import jakarta.annotation.Resource; import org.idea.qiyu.live.framework.redis.starter.key.AccountProviderCacheKeyBuilder; import org.qiyu.live.account.provider.service.IAccountTokenService; import org.springframework.data.redis.core.RedisTemplate; import org.springframework.stereotype.Service;  import java.util.UUID; import java.util.concurrent.TimeUnit;  /\*\*  \* @Author idea  \* @Date: Created in 10:21 2023/6/20  \* @Description  \*/ @Service public class AccountTokenServiceImpl implements IAccountTokenService {   @Resource  private RedisTemplate<String, String> redisTemplate;  @Resource  private AccountProviderCacheKeyBuilder accountProviderCacheKeyBuilder;   @Override  public String createAndSaveLoginToken(Long userId) {  String tokenKey = UUID.randomUUID().toString();  String loginTokenKey = accountProviderCacheKeyBuilder.buildUserLoginToken(tokenKey);  redisTemplate.opsForValue().set(loginTokenKey, String.valueOf(userId), 30, TimeUnit.DAYS);  return tokenKey;  }   @Override  public Long getUserIdByToken(String tokenKey) {  String loginTokenKey = accountProviderCacheKeyBuilder.buildUserLoginToken(tokenKey);  Object value = redisTemplate.opsForValue().get(loginTokenKey);  if (value == null) {  return null;  }  return Long.valueOf(redisTemplate.opsForValue().get(loginTokenKey));  }  } |

springboot启动类：

|  |
| --- |
| Java package org.qiyu.live.account.provider;  import org.apache.dubbo.config.spring.context.annotation.EnableDubbo; import org.springframework.boot.SpringApplication; import org.springframework.boot.WebApplicationType; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  /\*\*  \* @Author idea  \* @Date: Created in 10:13 2023/6/20  \* @Description  \*/ @SpringBootApplication @EnableDubbo @EnableDiscoveryClient public class AccountProviderApplication {   public static void main(String[] args) {  SpringApplication springApplication = new SpringApplication(AccountProviderApplication.class);  springApplication.setWebApplicationType(WebApplicationType.NONE);  springApplication.run(args);  } } |