**6-10｜引入Gateway网关**

**为什么要使用Gateway网关？**

* 使用网关后，可以对下游的Web服务做负载均衡
* 采用API Gateway可以与微服务注册中心连接，实现微服务无感知动态扩容。
* API Gateway对于无法访问的服务，可以做到自动熔断，无需人工参与。
* API Gateway可以方便的实现蓝绿部署，金丝雀发布或A/B发布。
* API Gateway做为系统统一入口，我们可以将各个微服务公共功能放在API Gateway中实现，以尽可能减少各服务的职责。

**如何引入网关**

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-gateway</artifactId>  <description>网关</description>    <version>1.0.1</version>   <properties>  <spring-cloud-starter-gateway.version>4.0.6</spring-cloud-starter-gateway.version>  <spring-cloud-starter-loadbalancer.version>4.0.3</spring-cloud-starter-loadbalancer.version>  <spring-cloud-starter-bootstrap.version>3.0.2</spring-cloud-starter-bootstrap.version>  <alibaba-fastjson.version>2.0.10</alibaba-fastjson.version>  </properties>   <dependencies>  <!--gateway 内部引入了webflux-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-gateway</artifactId>  <version>${spring-cloud-starter-gateway.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-loadbalancer</artifactId>  <version>${spring-cloud-starter-loadbalancer.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>${spring-cloud-starter-bootstrap.version}</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>${alibaba-fastjson.version}</version>  </dependency>  <dependency>  <groupId>org.idea</groupId>  <artifactId>qiyu-live-common-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>   <build>  <finalName>${artifactId}-docker</finalName>  <plugins>  <plugin>  <groupId>com.spotify</groupId>  <artifactId>docker-maven-plugin</artifactId>  <version>1.2.0</version>  <executions>  <!-- 当mvn执行install操作的时候，执行docker的build -->  <execution>  <id>build</id>  <phase>install</phase>  <goals>  <goal>build</goal>  </goals>  </execution>  </executions>  <configuration>  <imageTags>  <imageTag>${project.version}</imageTag>  </imageTags>  <imageName>${docker.registry.address}/${docker.registry.namespace}/${project.build.finalName}</imageName>  <!--指定Dockerfile文件的位置-->  <dockerDirectory>${project.basedir}/docker</dockerDirectory>  <!-- 指定jar包路径，这里对应Dockerfile中复制 jar 包到 docker 容器指定目录配置，也可以写到 Docokerfile 中 -->  <resources>  <resource>  <targetPath>/</targetPath>  <!-- 将下边目录的内容，拷贝到docker镜像中 -->  <directory>${project.build.directory}</directory>  <include>${project.build.finalName}.jar</include>  </resource>  <resource>  <targetPath>/</targetPath>  <directory>${arthus.zip.address}</directory>  <include>arthas-bin.zip</include>  </resource>  </resources>  </configuration>  </plugin>  <!-- 将springboot应用打包成jar-->  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> </project> |

启动类代码：

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

引入logback日志管理文件：

|  |
| --- |
| 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> |

bootstrap.yml配置：

|  |
| --- |
| YAML server:  port: 80 spring:  application:  name: qiyu-live-gateway  cloud:  nacos:  username: qiyu  password: qiyu  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-gateway.yaml  logging:  level:  org.springframework.cloud.gateway: DEBUG  reactor.netty.http.client: DEBUG |

nacos上配置网关的信息：

|  |
| --- |
| YAML spring:  cloud:  gateway:  discovery:  locator:  enabled: true  routes:  - id: qiyu-live-api  uri: lb://qiyu-live-api  predicates:  - Path=/live/api/\*\* |

关于网关的Dockerfile和docker-compose文件内容如下：

Dockerfile：

|  |
| --- |
| YAML FROM openjdk:17-jdk-alpine VOLUME /tmp ADD qiyu-live-gateway-docker.jar app.jar COPY /arthas-bin.zip /opt/arthas/arthas-bin.zip ENV JAVA\_OPTS="\ -server \ -Xmx1g \ -Xms1g \ -Xmn256m" ENTRYPOINT java ${JAVA\_OPTS} -Djava.security.egd=file:/dev/./urandom --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.math=ALL-UNNAMED -jar app.jar |

docker-compose：

|  |
| --- |
| YAML version: '3' services:  qiyu-live-gateway-docker:  container\_name: qiyu-live-gateway-docker  image: 'registry.baidubce.com/qiyu-live-test/qiyu-live-gateway-docker:1.0.1'  ports:  - "80:80"  environment:  - JAVA\_OPTS=-XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xms512m -Xmx512m -Xmn128m -Xss256k |