6-10 | 引入 Gateway 网关

为什么要使用 Gateway 网关？

1. 使用网关后，可以对下游的 Web 服务做负载均衡
1. 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。
1. 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。
1. API Gateway 可以方便的实现蓝绿部署，金丝雀发布或 A/B 发布。
1. 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
```