SpringCloud Finchley基础教程：3，spring cloud gateway网关

2018年04月20日 11:39:08 卓小洛o 阅读数：18101 更多

1. 引入pom依赖

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

2. 配置文件

2.1 配置eureka，application等信息

```
server:
  port: 30020
  http2:
    enabled: true
  servlet:
    context-path: /i5xforyou

spring:
  application:
    name: i5xforyou-service-gateway

eureka:
  instance:
    prefer-ip-address: true
    status-page-url-path: /actuator/info
    health-check-url-path: /actuator/health
  client:
    register-with-eureka: true
    fetch-registry: false
    service-url:
      defaultZone: http://localhost:50000/eureka/
```

server.servlet.context-path，由于gateway用的是webflux，所以这个设定其实是不生效的，现在还没有一个key来设定webflux的context-path

为了nginx能把请求都统一路由到gateway，所以必须要有一个统一的前缀，这里定义为i5xforyou，nginx可以设置请求前缀为/i5xforyou的请求都转发到gateway服务上。

2.2 配置gateway路由信息

```
spring:
  cloud:
    gateway:
      default-filters:
      routes:
      #---------------------------------------------------------------------
      - id: i5xforyou-biz-auth
        uri: lb://i5xforyou-biz-auth
        predicates:
        - Path= ${server.servlet.context-path}/auth/**
        filters:
        - StripPrefix= 1
      #---------------------------------------------------------------------
      - id: i5xforyou-biz-kanjia-websocket
        uri: lb:ws://i5xforyou-biz-kanjia-websocket
        predicates:
        - Path= ${server.servlet.context-path}/kanjia-websocket/**
        filters:
        - StripPrefix= 1
```

- default-filters: 里面可以定义一些共同的filter，对所有路由都起作用
- routes：具体的路由信息，是一个数组，每一个路由基本包含部分：
  - id：这个路由的唯一id，不定义的话为一个uuid
  - uri：http请求为lb://前缀 + 服务id；ws请求为lb:ws://前缀 + 服务id；表示将请求负载到哪一个服务上
  - predicates：表示这个路由的请求匹配规则，只有符合这个规则的请求才会走这个路由。为一个数组，每个规则为并且的关系。
  - filters：请求转发前的filter，为一个数组。
  - order：这个路由的执行order

2.3 predicates请求匹配规则

predicates：请求匹配规则，为一个数组，每个规则为并且的关系。包含：
1. name：规则名称，目前有10个，有Path，Query，Method，Header，After，Before，Between，Cookie，Host，RemoteAddr
2. args：参数key-value键值对,例：

```
```
predicates:
- name: Query
```
```

```
  args:
    foo: ba
```
等价于

```
predicates:
- Query=foo, ba
```

如果args不写key的，会自动生成一个id，如下会生成一个xxx0的key，值为/foo/*

```
predicates:
- Path=/foo/*
```

3. */代表一层路径，/*代表多层目录
4. 具体详情参照：http://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.0.0.M9/multi/multi_gateway-request-predicates-factories.html

2.4 filters请求过滤filter

filters：请求过滤filter，为一个数组，每个filter都会顺序执行。包含：
1. name：过滤filter名称，常用的有Hystrix断路由，RequestRateLimiter限流，StripPrefix截取请求url
2. args：参数key-value键值对,例：

```
filters:
- name: Hystrix
  args:
    name: fallbackcmd
    fallbackUri: forward:/incaseoffailureusethis
```

如果args不写key的，会自动生成一个id，如下会生成一个xxx0的key，值为1

```
filters:
- StripPrefix= 1
```

3. 具体详情参照：http://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.0.0.M9/multi/multi_gateway-route-filters.html

3. 重要的filter详解

3.1 StripPrefix

```
spring:
  cloud:
    gateway:
      routes:
      - id: nameRoot
        uri: lb://nameservice
        predicates:
        - Path=/name/**
        filters:
        - StripPrefix=1
```

/name/bar/foo的请求会被转发为http://nameserviceip:nameserviceport/bar/foo

3.2 Hystrix断路由

1. 引入pom依赖

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

2. 配置文件：

```
spring:
  cloud:
    gateway:
      default-filters:
      routes:
      #---------------------------------------------------------------------
      - id: i5xforyou-biz-auth
        uri: lb://i5xforyou-biz-auth
        predicates:
        - Path= ${server.servlet.context-path}/auth/**
        filters:
        - StripPrefix= 1
        - name: Hystrix
          args:
            name: authHystrixCommand
            fallbackUri: forward:/hystrixTimeout


#设置断路由的超时时间，毫秒
```

```
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds= 30000
```

name表示HystrixCommand代码的名称，fallbackUri表示触发断路由后的跳转请求url

3. HystrixCommand代码

```
@RestController
public class HystrixCommandController {
    protected final Logger log = LoggerFactory.getLogger(this.getClass());

    @RequestMapping("/hystrixTimeout")
    public JsonPackage hystrixTimeout() {
        log.error("i5xforyou-service-gateway触发了断路由");
        return JsonPackage.getHystrixJsonPackage();
    }

    @HystrixCommand(commandKey="authHystrixCommand")
    public JsonPackage authHystrixCommand() {
        return JsonPackage.getHystrixJsonPackage();
    }

}
```

3.3 Retry重试

```
spring:
  cloud:
    gateway:
      default-filters:
      routes:
      #--------------------------------------------------------------------------
      - id: i5xforyou-biz-auth
        uri: lb://i5xforyou-biz-auth
        predicates:
        - Path= ${server.servlet.context-path}/auth/**
        filters:
        - StripPrefix= 1
        - name: Retry
          args:
            retries: 3 #重试次数，默认3，不包含本次
            status: 404 #重试response code，默认没有
            statusSeries: 500 #重试response code的系列，100（info），200（success），300（redirect），400（client error），500（server error），默认500
            method: GET #重试的request请求，默认GET
```

没有timeout超时重试，并且没有retriesNextServer设置，导致多次重试都是到同一个服务实例。不太实用。

3.4 自定义gateway filter

自定义一个用来检验jwt是否合法的gateway filter为例进行说明。

1. 定义一个JwtCheckGatewayFilterFactory类实现GatewayFilterFactory接口。
   类名一定要为filterName + GatewayFilterFactory，如这里定义为JwtCheckGatewayFilterFactory的话，它的filterName就是JwtCheck

2. 实现gateway filter的业务逻辑

```
@Component
public class JwtCheckGatewayFilterFactory extends AbstractGatewayFilterFactory<Object> {

    @Override
    public GatewayFilter apply(Object config) {
        return (exchange, chain) -> {
            //return chain.filter(exchange);
            String jwtToken = exchange.getRequest().getHeaders().getFirst("Authorization");
            //校验jwtToken的合法性
            if (JwtUtil.verifyToken(jwtToken) != null) {
                //合法
                return chain.filter(exchange);
            }


            //不合法
            ServerHttpResponse response = exchange.getResponse();
            //设置headers
            HttpHeaders httpHeaders = response.getHeaders();
            httpHeaders.add("Content-Type", "application/json; charset=UTF-8");
            httpHeaders.add("Cache-Control", "no-store, no-cache, must-revalidate, max-age=0");
            //设置body
            JsonPackage jsonPackage = new JsonPackage();
            jsonPackage.setStatus(110);
            jsonPackage.setMessage("未登录或登录超时");
            DataBuffer bodyDataBuffer = response.bufferFactory().wrap(jsonPackage.toJSONString().getBytes());

            return response.writeWith(Mono.just(bodyDataBuffer));
        };
    }

}
```

3. 设定配置文件即可

```yaml
spring:
  cloud:
    gateway:
      default-filters:
      routes:
      #--------------------------------------------------------------------------
      - id: i5xforyou-biz-auth
        uri: lb://i5xforyou-biz-auth
        predicates:
        - Path= ${server.servlet.context-path}/auth/**
        filters:
        - StripPrefix= 1
        - JwtCheck
```

3.5 自定义限流gateway filter

gateway自带的RequestRateLimiter可定制的内容太少，真正用的话，需要：
1. 自定义限流后的response返回值
2. 不同的key（即接口）限流数不同
所以需要自定义一个限流的gateway filter

3.5.1 重写RequestRateLimiter

```java
@Component
public class RateCheckGatewayFilterFactory extends AbstractGatewayFilterFactory<RateCheckGatewayFilterFactory.Config> implements ApplicationContextAware {
    private static Logger log = LoggerFactory.getLogger(RateCheckGatewayFilterFactory.class);
    private static ApplicationContext applicationContext;
    private RateCheckRedisRateLimiter rateLimiter;
    private KeyResolver keyResolver;

    public RateCheckGatewayFilterFactory() {
        super(Config.class);
    }

    @Override
    public void setApplicationContext(ApplicationContext context) throws BeansException {
        log.info("RateCheckGatewayFilterFactory.setApplicationContext，applicationContext=" + context);
        applicationContext = context;
    }

    @Override
    public GatewayFilter apply(Config config) {
        this.rateLimiter = applicationContext.getBean(RateCheckRedisRateLimiter.class);
        this.keyResolver = applicationContext.getBean(config.keyResolver, KeyResolver.class);

        return (exchange, chain) -> {
            Route route = exchange.getAttribute(ServerWebExchangeUtils.GATEWAY_ROUTE_ATTR);

            return keyResolver.resolve(exchange).flatMap(key ->
                    // TODO: if key is empty?
                    rateLimiter.isAllowed(route.getId(), key).flatMap(response -> {
                        log.info("response: " + response);
                        // TODO: set some headers for rate, tokens left
                        if (response.isAllowed()) {
                            return chain.filter(exchange);
                        }
                        //超过了限流的response返回值
                        return setRateCheckResponse(exchange);
                    }));
        };
    }

    private Mono<Void> setRateCheckResponse(ServerWebExchange exchange) {
        //超过了限流
        ServerHttpResponse response = exchange.getResponse();
        //设置headers
        HttpHeaders httpHeaders = response.getHeaders();
        httpHeaders.add("Content-Type", "application/json; charset=UTF-8");
        httpHeaders.add("Cache-Control", "no-store, no-cache, must-revalidate, max-age=0");
        //设置body
        JsonPackage jsonPackage = new JsonPackage();
        jsonPackage.setStatus(HttpStatus.TOO_MANY_REQUESTS.value());
        jsonPackage.setMessage("系统繁忙，请稍后重试");
        DataBuffer bodyDataBuffer = response.bufferFactory().wrap(jsonPackage.toJSONString().getBytes());

        return response.writeWith(Mono.just(bodyDataBuffer));
    }

    public static class Config {
        private String keyResolver;//限流id

        public String getKeyResolver() {
            return keyResolver;
        }
        public void setKeyResolver(String keyResolver) {
            this.keyResolver = keyResolver;
        }
    }
}
```

在这里可以自定义限流后的response返回值

3.5.2 重写RedisRateLimiter

```
@Component
@Primary
public class RateCheckRedisRateLimiter extends AbstractRateLimiter<RateCheckRedisRateLimiter.Config> implements ApplicationContextAware {
    public static final String CONFIGURATION_PROPERTY_NAME = "redis-rate-limiter";
    public static final String REDIS_SCRIPT_NAME = "redisRequestRateLimiterScript";

    private static Logger log = LoggerFactory.getLogger(RateCheckGatewayFilterFactory.class);

    private ReactiveRedisTemplate<String, String> redisTemplate;
    private RedisScript<List<Long>> script;
    private AtomicBoolean initialized = new AtomicBoolean(false);
    private Config defaultConfig;

    public RateCheckRedisRateLimiter() {
        super(Config.class, CONFIGURATION_PROPERTY_NAME, null);
    }

// public RateCheckRedisRateLimiter(ReactiveRedisTemplate<String, String> redisTemplate,
//       RedisScript<List<Long>> script, Validator validator) {
//    super(Config.class, CONFIGURATION_PROPERTY_NAME, validator);
//    this.redisTemplate = redisTemplate;
//    this.script = script;
//    initialized.compareAndSet(false, true);
// }
//
// public RateCheckRedisRateLimiter(int defaultReplenishRate, int defaultBurstCapacity) {
//    super(Config.class, CONFIGURATION_PROPERTY_NAME, null);
//    this.defaultConfig = new Config()
//          .setReplenishRate(defaultReplenishRate)
//          .setBurstCapacity(defaultBurstCapacity);
// }

    private Config setConfig(String key) {
        //TODO 根据key（接口）找到对应的限流配置
        int replenishRate = 0;//令牌通流量,每秒
        int burstCapacity = 0;//令牌通容量

        defaultConfig = new Config()
                .setReplenishRate(replenishRate)
                .setBurstCapacity(burstCapacity);
        return defaultConfig;
    }

    @Override
    @SuppressWarnings("unchecked")
    public void setApplicationContext(ApplicationContext context) throws BeansException {
        if (initialized.compareAndSet(false, true)) {
            this.redisTemplate = context.getBean("stringReactiveRedisTemplate", ReactiveRedisTemplate.class);
            this.script = context.getBean(REDIS_SCRIPT_NAME, RedisScript.class);
            if (context.getBeanNamesForType(Validator.class).length > 0) {
                this.setValidator(context.getBean(Validator.class));
            }
        }
    }

    /* for testing */
    Config getDefaultConfig() {
        return defaultConfig;
    }

    /**
    * This uses a basic token bucket algorithm and relies on the fact that Redis scripts
    * execute atomically. No other operations can run between fetching the count and
    * writing the new count.
    */
    @Override
    public Mono<Response> isAllowed(String routeId, String id) {
        if (!this.initialized.get()) {
            throw new IllegalStateException("RedisRateLimiter is not initialized");
        }

        //根据key（接口）找到对应的限流配置
        Config routeConfig = setConfig(id);

        // How many requests per second do you want a user to be allowed to do?
        int replenishRate = routeConfig.getReplenishRate();

        // How much bursting do you want to allow?
        int burstCapacity = routeConfig.getBurstCapacity();

        try {
            List<String> keys = getKeys(id);


            // The arguments to the LUA script. time() returns unixtime in seconds.
            List<String> scriptArgs = Arrays.asList(replenishRate + "", burstCapacity + "",
                    Instant.now().getEpochSecond() + "", "1");
            // allowed, tokens_left = redis.eval(SCRIPT, keys, args)
            Flux<List<Long>> flux = this.redisTemplate.execute(this.script, keys, scriptArgs);
                    // .log("redisratelimiter", Level.FINER);
```

```
            return flux.onErrorResume(throwable -> Flux.just(Arrays.asList(1L, -1L)))
                    .reduce(new ArrayList<Long>(), (longs, l) -> {
                        longs.addAll(l);
                        return longs;
                    }) .map(results -> {
                        boolean allowed = results.get(0) == 1L;
                        Long tokensLeft = results.get(1);

                        Response response = new Response(allowed, tokensLeft);

                        if (log.isDebugEnabled()) {
                            log.debug("response: " + response);
                        }
                        return response;
                    });
        }
        catch (Exception e) {
            /*
         * We don't want a hard dependency on Redis to allow traffic. Make sure to set
         * an alert so you know if this is happening too much. Stripe's observed
         * failure rate is 0.01%.
         */
            log.error("Error determining if user allowed from redis", e);
        }
        return Mono.just(new Response(true, -1));
    }

    static List<String> getKeys(String id) {
        // use `{}` around keys to use Redis Key hash tags
        // this allows for using redis cluster

        // Make a unique key per user.
        String prefix = "request_rate_limiter.{" + id;

        // You need two Redis keys for Token Bucket.
        String tokenKey = prefix + "}.tokens";
        String timestampKey = prefix + "}.timestamp";
        return Arrays.asList(tokenKey, timestampKey);
    }

    @Validated
    public static class Config {
        @Min(1)
        private int replenishRate;

        @Min(0)
        private int burstCapacity = 0;

        public int getReplenishRate() {
            return replenishRate;
        }

        public Config setReplenishRate(int replenishRate) {
            this.replenishRate = replenishRate;
            return this;
        }

        public int getBurstCapacity() {
            return burstCapacity;
        }

        public Config setBurstCapacity(int burstCapacity) {
            this.burstCapacity = burstCapacity;
            return this;
        }

        @Override
        public String toString() {
            return "Config{" +
                    "replenishRate=" + replenishRate +
                    ", burstCapacity=" + burstCapacity +
                    '}';
        }
    }
}
```

在这里可以根据不同的key（接口）来获取不同的限流设置，具体配置文件及映射方法根据自己项目需要自行配置即可。

3.5.3 配置文件设置

```
spring:
  cloud:
    gateway:
      default-filters:
      routes:
      #-----------------------------------------------------------------------
      - id: i5xforyou-biz-auth
        uri: lb://i5xforyou-biz-auth
        predicates:
        - Path= ${server.servlet.context-path}/auth/**
        filters:
        - StripPrefix= 1
```