API Gateway Service-3

작성 일시	@2022년 11월 3일 오후 9:16
최종 편집 일시	@2022년 11월 3일 오후 11:15
유형	Spring Cloud로 개발하는 마이크로서비스 애플리케이션(MSA)
작성자	
참석자	

Spring Cloud Gateway - Global Filter (공통필터)

```
spring:
  application:
    name: apigateway-service
#  yml filter방식
cloud:
    gateway:
      default-filters:
        - name: GlobalFilter
          args:
            baseMessage: Spring Cloud Gateway Global Filter
            preLogger: true
            postLogger: true
      routes:
```

```
@Component
@Slf4j
public class GlobalFilter extends AbstractGatewayFilterFactory<GlobalFilter.Config> {
    public GlobalFilter() {
        super(Config.class);
    }

    @Override
    public GatewayFilter apply(Config config) {
        // Custom pre Filter
        return (exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest();
```

```
            ServerHttpResponse response = exchange.getResponse();

log.info("Global Filter baseMessage: {}", config.getBaseMessage());

            if (config.isPreLogger()) {
log.info("Global Filter Start: request id -> {}", request.getId());
            }
            // Custom Post Filter
            // webflux(spring 5) Mono 단일값 전달 한다는 의미이며, 비동기적 netty 서버에서 지원
함.
            return chain.filter(exchange).then(Mono.fromRunnable(() -> {
                if (config.isPostLogger()) {
log.info("Global Filter End: response code -> {}", response.getStatusCode());
                }
            }));
        };
    }

    @Data
    public static class Config {
        private String baseMessage;
        private boolean preLogger;
        private boolean postLogger;
    }
}
```

Spring Cloud Gateway - Logging Filter

filter 클래스에서 바로 람다식으로 return 하는 방식이 아닌 gatewayFilter 를 상속받아 사용되는 클래스인 OrderedGatewayFilter를 사용해 return 한다.

```
spring:
  application:
    name: apigateway-service
#  yml filter방식
cloud:
    gateway:
      default-filters:
        - name: GlobalFilter
          args:
            baseMessage: Spring Cloud Gateway Global Filter
            preLogger: true
            postLogger: true
      routes:
        - id: first-service
          uri: http://localhost:8081/
          predicates:
            - Path=/first-service/**
          filters:
#            -  AddRequestHeader=first-request, first-request-header2
#            -  AddResponseHeader=first-response, first-response-header2
            - name: CustomFilter
            - name: LoggingFilter
              args:
```

```
                baseMessage: Hi i am logging
                preLogger: true
                postLogger: true
```

```
@Component
@Slf4j
public class LoggingFilter extends AbstractGatewayFilterFactory<LoggingFilter.Config>
 {
    public LoggingFilter() {
        super(Config.class);
    }

    @Override
    public GatewayFilter apply(Config config) {
        GatewayFilter filter = new OrderedGatewayFilter((exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest();
            ServerHttpResponse response = exchange.getResponse();

log.info("Logging Filter baseMessage: {}", config.getBaseMessage());

            if (config.isPreLogger()) {
log.info("Logging PRE Filter: request id -> {}", request.getId());
            }

            return chain.filter(exchange).then(Mono.fromRunnable(() -> {
                if (config.isPostLogger()) {
log.info("Logging POST Filter: response code -> {}", response.getStatusCode());
                }
            }));
        }, Ordered.LOWEST_PRECEDENCE);

        return filter;
    }

    @Data
    public static class Config {
        private String baseMessage;
        private boolean preLogger;
        private boolean postLogger;
    }
}
```

Spring Cloud Gateway - Load Balancer

first service, second service, apigateway service 들을 eureka client 로 등록해준다.

```
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:42580/eureka
```

기존의 apigateway service 에서 yml 내용을 변경한다.

- lb → load Balancer
- Eureka url 이동후 인스턴스 항목중에 Application 이름 을 지정 lb:// 뒤에 지정

```
      routes:
        - id: first-service
#          uri: http://localhost:8081/
          uri: lb://MY-FIRST-SERVICE
```

한 서비스에 여러개의 인스턴스를 있는 경우 어떤 인스턴스를 탔는지 확인을 위한 코드

```
@GetMapping("/check")
public String check(HttpServletRequest request) {
    log.info("Server port= {}", request.getServerPort());
    return String.format("Hi, There. This is a message from Second Service on Port %
s", env.getProperty("local.server.port"));
}
```

RR (round - robin) 방식으로 나눠서 동작하게 된다.