

# 7장. 마이크로 서비스 통신



Order 마이크로서비스 구축  
ecommerce\_order 프로젝트

# 1. Http Client용 자바 라이브러리

## 개요 및 종류

분산 환경에서 일반적으로 많이 사용되는 Http Client 자바 라이브러리 종류는 다음과 같다.

### 가. HttpURLConnection

- HTTP Connection 연결
- HTTP 요청 방식(메서드) 설정 및 Header 설정
- Connection 타임아웃 설정 및 응답 Content Type 설정
- 호출 및 응답 데이터 처리

### 나. Apache Commons HttpClient

### 다. RestTemplate

Spring 3.0부터 지원하는 Restful 서비스에 최적화된 Http Client 라이브러리로서 손쉽게 JSON을 요청/응답을 처리 할 수 있으나 코드가 복잡해진다.

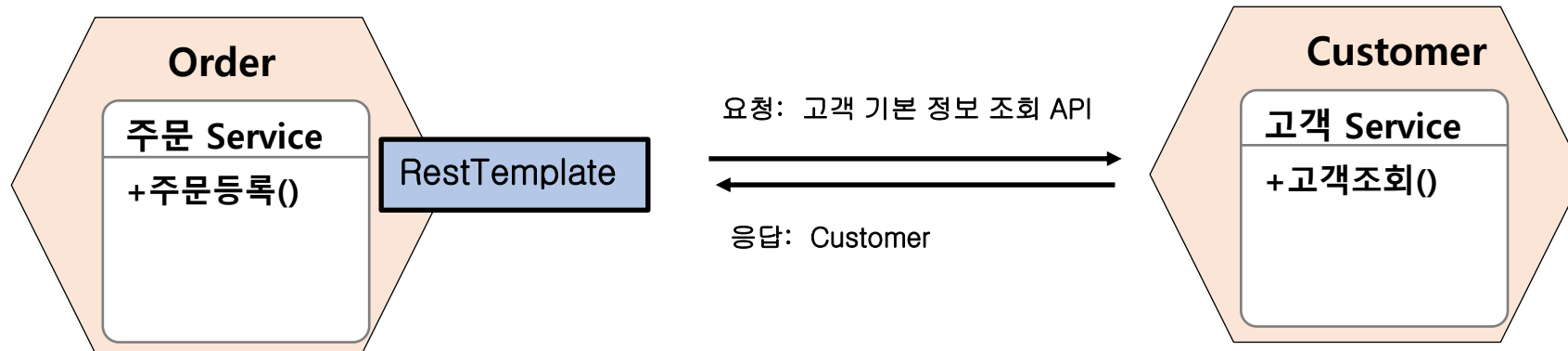
### 라. Feign Client

Netflix에서 만든 인터페이스 기반의 선언적 Rest Client 라이브러리이다. RestTemplate을 대체할 수 있는 기능으로서 Spring Cloud 프로젝트에서 Spring Cloud OpenFeign 이름으로 제공해주며 매우 간단하게 작업할 수 있다.

## 2. Order 와 Customer 서비스 통신(RestTemplate)

### 1) 개요

Order 서비스에서 주문등록시 고객기본정보를 Customer서비스에서 조회하여 주문 등록하는 코드를 구현한다.



## 2. Order 와 Customer 서비스 통신 (RestTemplate)

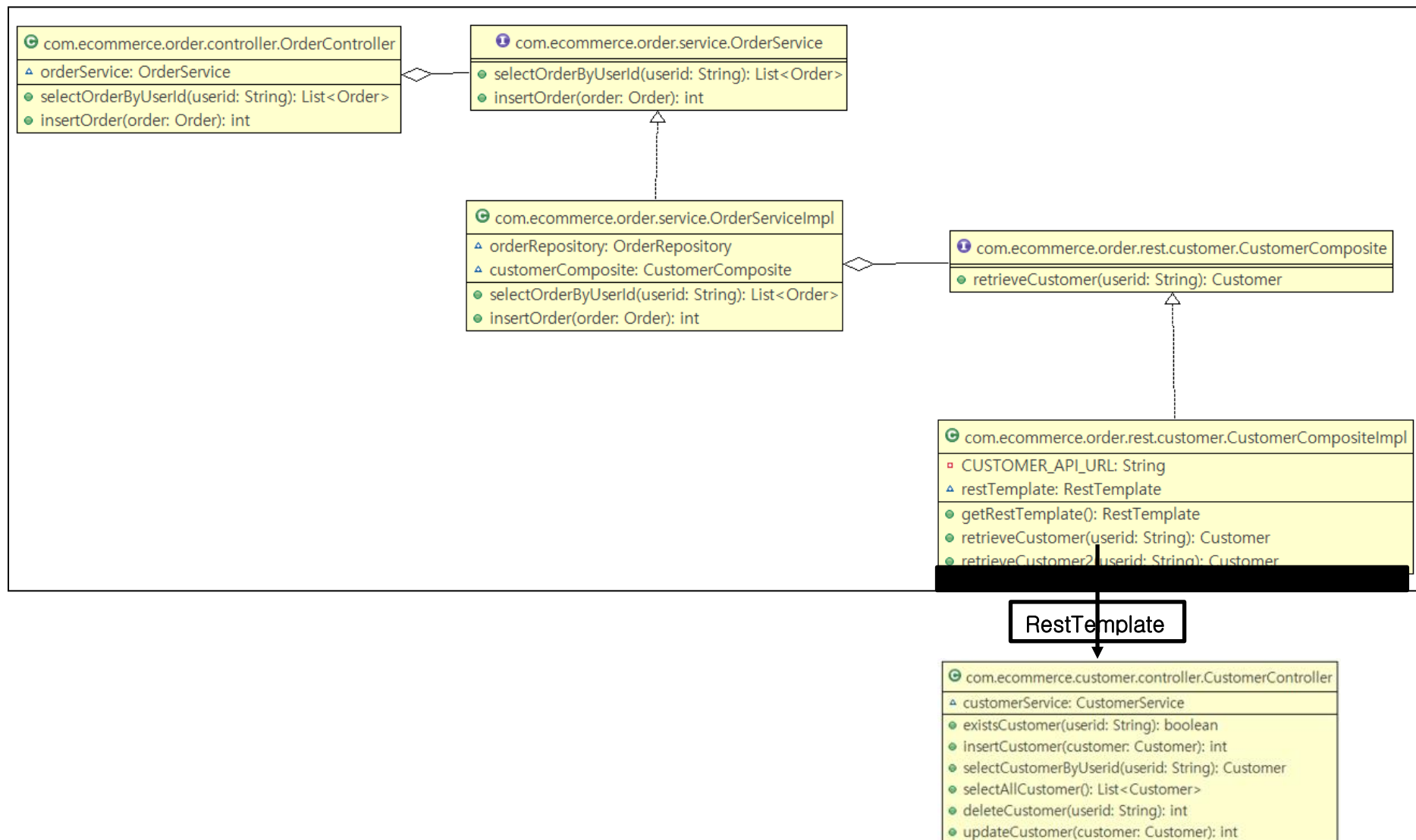
### 2) 요구사항

아래의 요구사항에 만족하도록 API를 개발한다.

API 명	URL	HTTP method	Annotation
주문조회	/rest/orders/{userid}	GET	@PathVariable
주문등록	/insertOrder	POST	@RequestBody

## 2. Order 와 Customer 서비스 통신 (RestTemplate)

### 3) 클래스 다이어그램



## 2. Order 와 Customer 서비스 통신 (RestTemplate)

### 4) Order 서비스 수정

```
@ApiOperation(value = "주문 등록", httpMethod = "POST", notes = "주문 등록")
@PostMapping("/insertOrder")
public int insertOrder(@RequestBody Order order) throws Exception{
    order.setOrderId(UUID.randomUUID().toString());

    return orderService.insertOrder(order);
}
```

```
public interface OrderService {

    public List<Order> selectOrderByUserId(String userid) throws Exception;
    public int insertOrder(Order order) throws Exception;
}
```

```
@Service("orderService")
public class OrderServiceImpl implements OrderService {

    @Autowired
    OrderRepository orderRepository;

    @Autowired
    CustomerComposite customerComposite;

    @Override
    public int insertOrder(Order order) throws Exception {

        Customer customer = customerComposite.retrieveCustomer(order.getUserId());
        order.setName(customer.getName());
        System.out.println("Customer >>" + customer);

        return orderRepository.insertOrder(order);
    }
}
```

## 2. Order 와 Customer 서비스 통신 (RestTemplate)

### 5) CustomerComposite 인터페이스 구현

```
public interface CustomerComposite {  
  
    public Customer retrieveCustomer(String userid) throws Exception;  
  
}
```

### 6) CustomerCompositeImpl 인터페이스 구현 (\*)

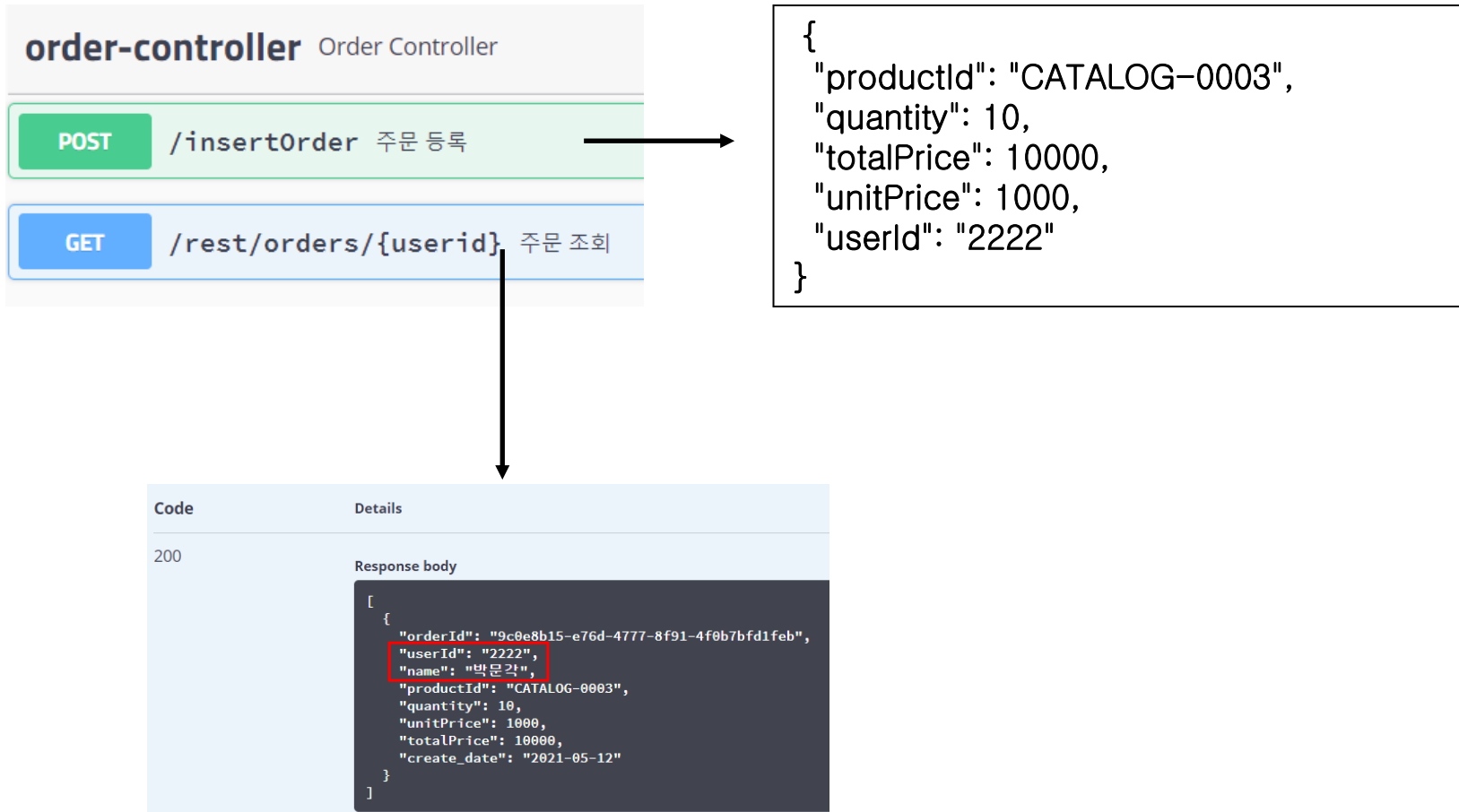
```
@Component  
public class CustomerCompositeImpl implements CustomerComposite {  
  
    @Value("${customer.api.url}")  
    private String CUSTOMER_API_URL;  
  
    @Bean  
    public RestTemplate getRestTemplate() {  
        return new RestTemplate();  
    }  
  
    @Autowired  
    RestTemplate restTemplate;  
  
    @Override  
    public Customer retrieveCustomer(String userid) throws Exception {  
        return restTemplate.getForObject(CUSTOMER_API_URL+"/rest/customers/"+userid,  
            Customer.class);  
    }  
  
}
```

```
#RestTemplate  
customer.api.url=http://${CUSTOMER}:8076/ecommerce/customer
```

## 2. Order 와 Customer 서비스 통신 (RestTemplate)

### 7) 실행 및 Swagger UI 요청

<http://localhost:8074/ecommerce/order/swagger-ui.html>





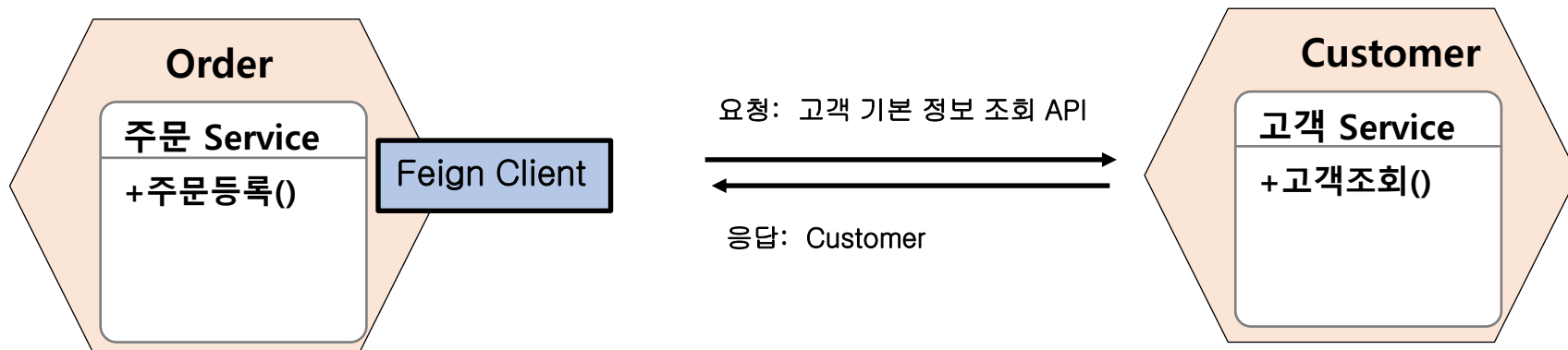
### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### Feign Client API 이용

- Rest Call을 추상화한 Spring Cloud Netflix 라이브러리
- 사용방법
  - 가. 의존성 설정
  - 나. 호출하려는 HTTP Endpoint에 대한 interface 작성 및 @FeignClient 선언
  - 다. Application 빈에 @EnableFeignClients 선언
- Load balanced 지원 및 예외발생시 처리하는 fallback 지원

#### 1) 개요

Order 서비스에서 주문등록시 고객기본정보를 Customer서비스에서 조회하여 주문 등록하는 코드를 구현한다.



### 3. Order 와 Customer 서비스 통신 (Feign Client)

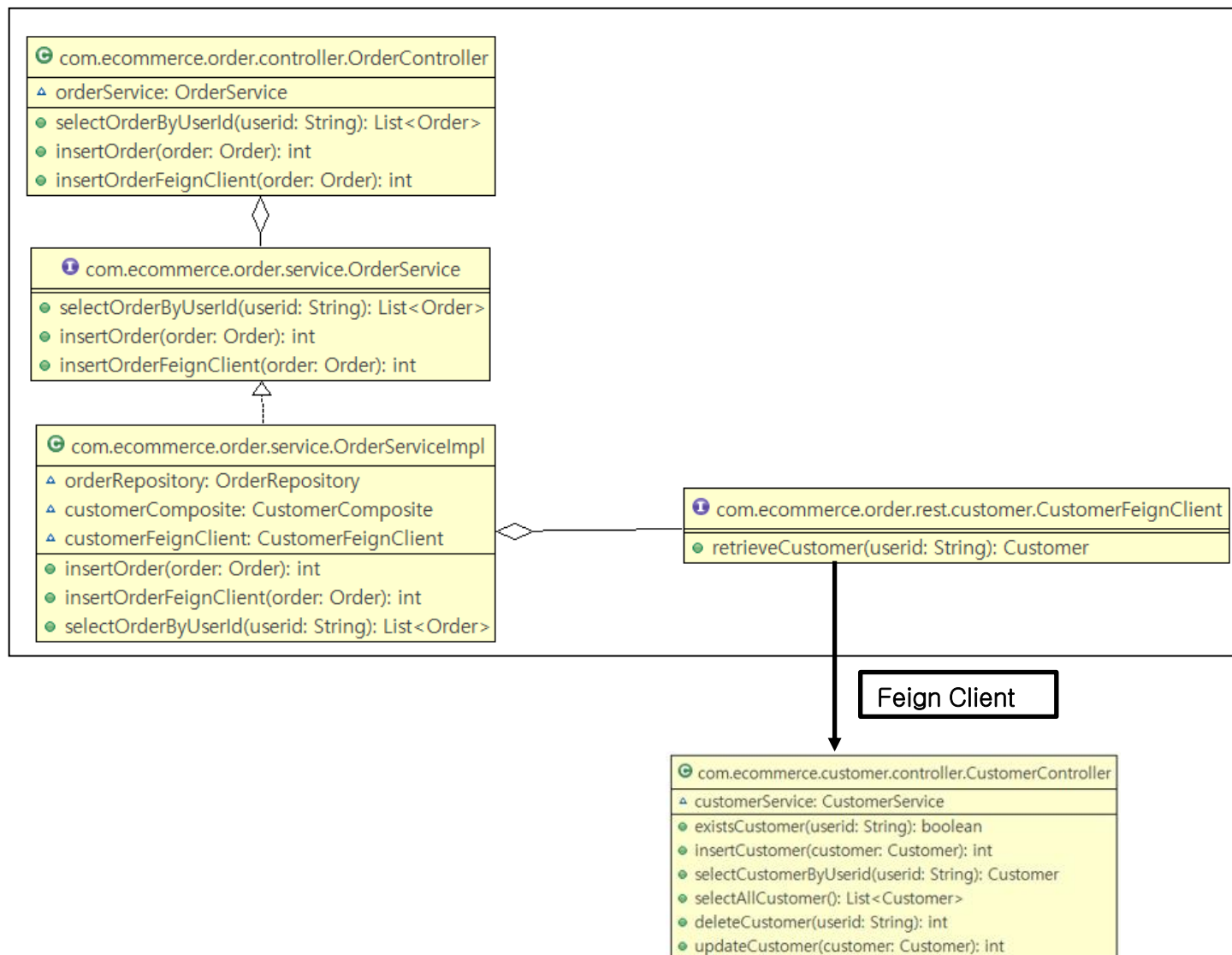
#### 2) 요구사항

아래의 요구사항에 만족하도록 API를 개발한다.

API 명	URL	HTTP method	Annotation
주문조회	/rest/orders/{userid}	GET	@PathVariable
주문등록	/insertOrder	POST	@RequestBody
주문등록	/insertOrderFeignClient	POST	@RequestBody

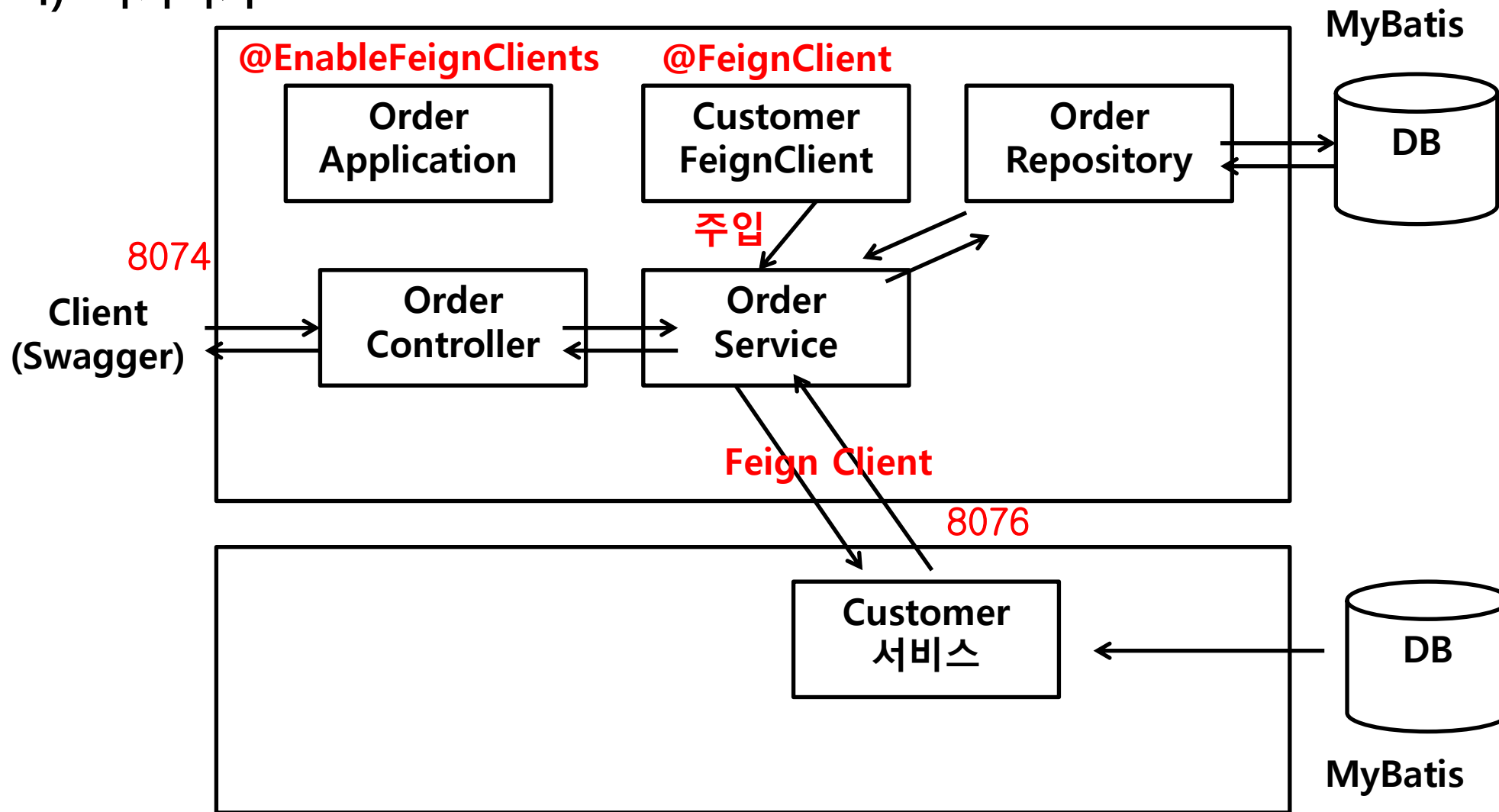
### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### 3) 클래스 다이어그램



### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### 4) 아키텍처



### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### 5) Order 서비스 수정

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

```
@RestController
public class OrderController {

    @Autowired
    OrderService orderService;

    @ApiOperation(value = "주문 등록, Feign Client", httpMethod = "POST", notes = "주문 등록, Feign Client")
    @PostMapping("/insertOrderFeignClient")
    public int insertOrderFeignClient(@RequestBody Order order) throws Exception{
        order.setOrderId(UUID.randomUUID().toString());

        return orderService.insertOrderFeignClient(order);
    }
}
```

```
public interface OrderService {

    public List<Order> selectOrderByUserId(String userid) throws Exception;
    public int insertOrder(Order order) throws Exception;
    public int insertOrderFeignClient(Order order) throws Exception;
}
```

### 3. Order 와 Customer 서비스 통신 (Feign Client)

```
@Service("orderService")
public class OrderServiceImpl implements OrderService {

    @Autowired
    OrderRepository orderRepository;

    @Autowired
    CustomerComposite customerComposite;

    @Autowired
    CustomerFeignClient customerFeignClient;

    //Feign Client
    @Override
    public int insertOrderFeignClient(Order order) throws Exception {
        Customer customer = customerFeignClient.retrieveCustomer(order.getUserId());
        order.setName(customer.getName());
        System.out.println("Customer Feign Client >>" + customer);

        return orderRepository.insertOrder(order);
    }
}
```

### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### 6) CustomerFeignClient 인터페이스 구현

```
@FeignClient(name="ecommerce-customer",
            url = "http://localhost:8076/ecommerce/customer"
            )
public interface CustomerFeignClient {

    @GetMapping("/rest/customers/{userid}")
    public Customer retrieveCustomer(@PathVariable String userid)throws Exception;

}
```

#### 7) Application 빈 수정

```
@SpringBootApplication
@EnableFeignClients
public class EcommerceOrderApplication {

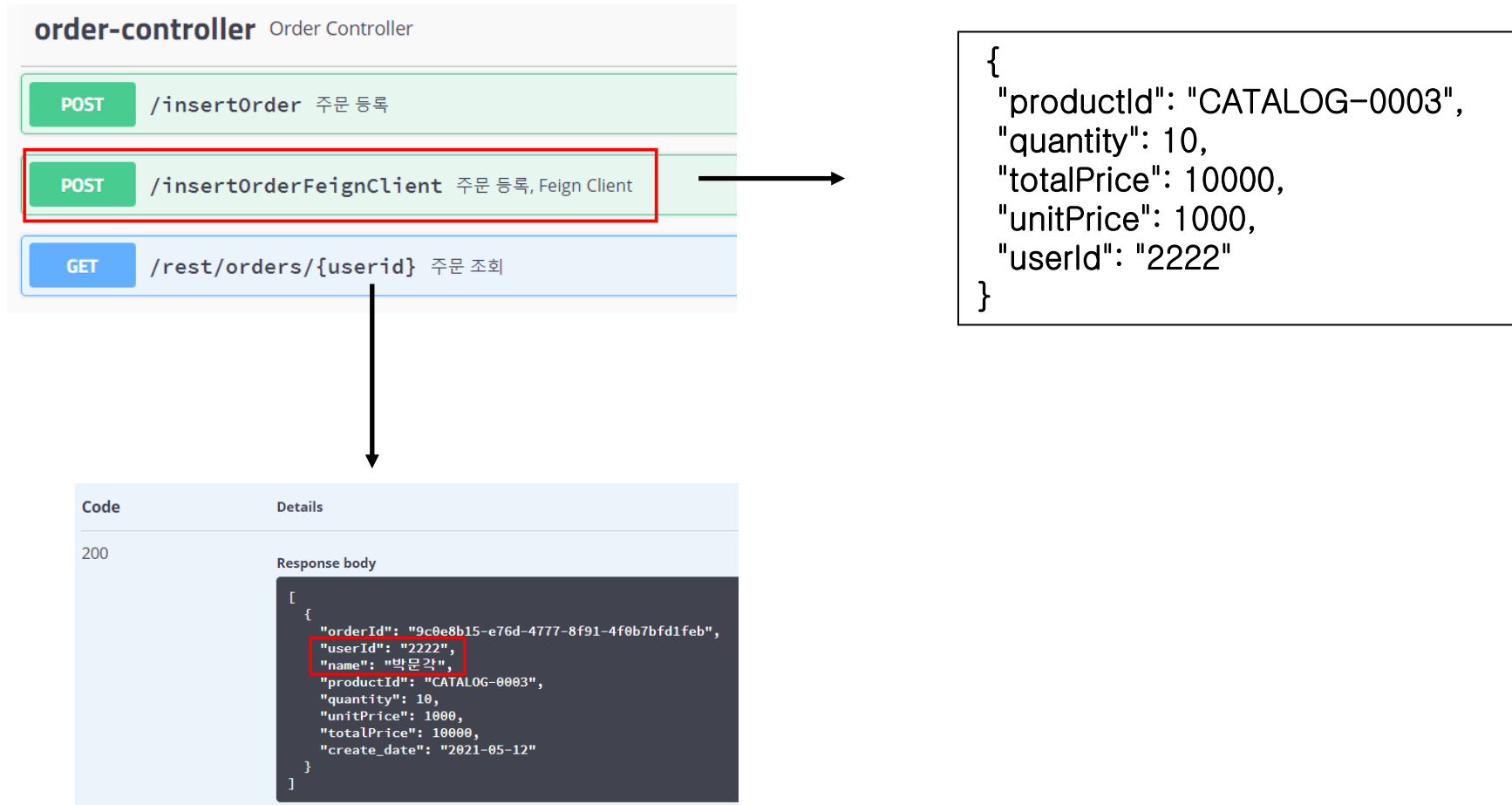
    public static void main(String[] args) {
        SpringApplication.run(EcommerceOrderApplication.class, args);
    }

}
```

### 3. Order 와 Customer 서비스 통신 (Feign Client)

#### 8) 실행 및 Swagger UI 요청

<http://localhost:8074/ecommerce/order/swagger-ui.html>





# 10장. Circuit Breaker



적용 시나리오

Hystrix 개요 및 역할

Hystrix 적용방법 및 설정값

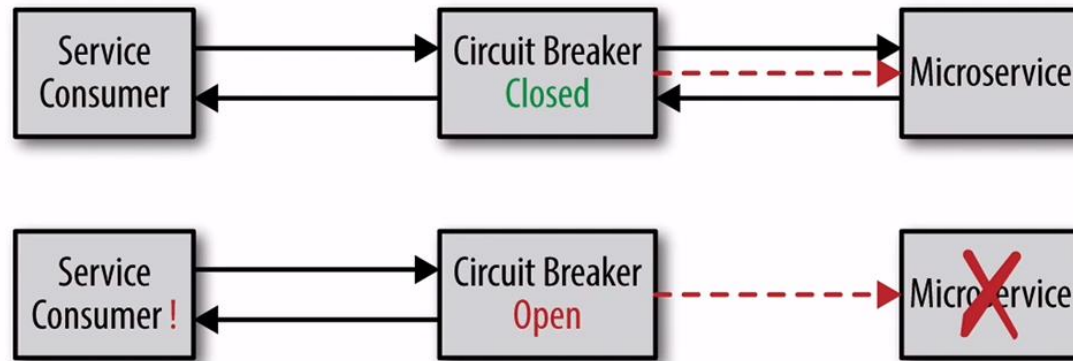
# 1. Circuit Breaker – Hystrix 개요

## 1. Hystrix 개요 <https://github.com/Netflix/hystrix>

Hystrix는 분산환경을 위한 장애 및 지연 내성(Latency and Fault Tolerance)을 갖도록 도와주는 라이브러리로서, Circuit Breaker Pattern 디자인을 적용하여 MSA 어플리케이션의 장애 전파를 방지 할 수 있다.

Hystrix와 같은 Circuit Breaker 서비스는 기존의 모놀리식 아키텍처에서는 고려하지 않았던 모듈간 또는 메서드간의 호출 실패가 MSA에서는 발생할 수 있음을 의미한다.

또한 MSA에서는 각각의 서비스들이 독립적이지만, 장애가 전파될 수 있는 성질이 있어 이를 미연에 방지하기 위한 Circuit Breaker가 필수이다.



<https://martinfowler.com/bliki/CircuitBreaker.html>

## 2. Circuit Breaker – Hystrix 역할

### 2. Hystrix 핵심 역할 3가지

#### 가. 장애 및 지연 내성 ( Fault and Latency Tolerance)

분산환경에서 한 개의 서비스가 실패하는 경우, 해당 서비스의 실패로 의존성이 있는 타 서비스까지 장애가 전파될 수 있다.

이렇게 외부에서 호출하는 서비스를 Hystrix로 Wrapping하면 실패가 전파되는 것을 Fallback를 활용하여 미연에 방지하고 빠르게 복구할 수 있도록 도와준다.

또한, 각 서비스의 HystrixCommand는 Circuit Breaker Pattern으로 외부에 영향을 받지 않도록 쓰레드 또는 세마포어 방식으로 분리(Isolation)되어 있다.

#### 나. 실시간 구동 모니터링 ( Realtime Operations)

Hystrix를 사용하면 실시간 모니터링과 설정 변경을 지원한다. 서비스와 설정값의 변경이 어떻게 시스템에 적용되는지 대쉬보드를 통해서 확인할 수 있다.

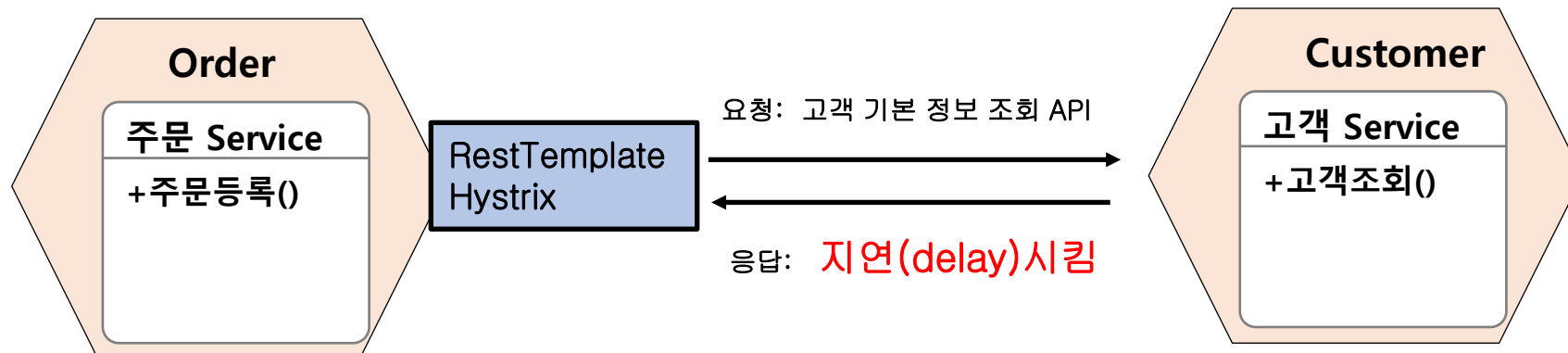
#### 다. 병행성

Parallel execution을 제공하여 여러 설정 값에 대한 변경을 지원하고 중복되는 request 처리를 줄이기 위하여 Request Caching 기능을 제공한다.

### 3. Circuit Breaker 적용 시나리오

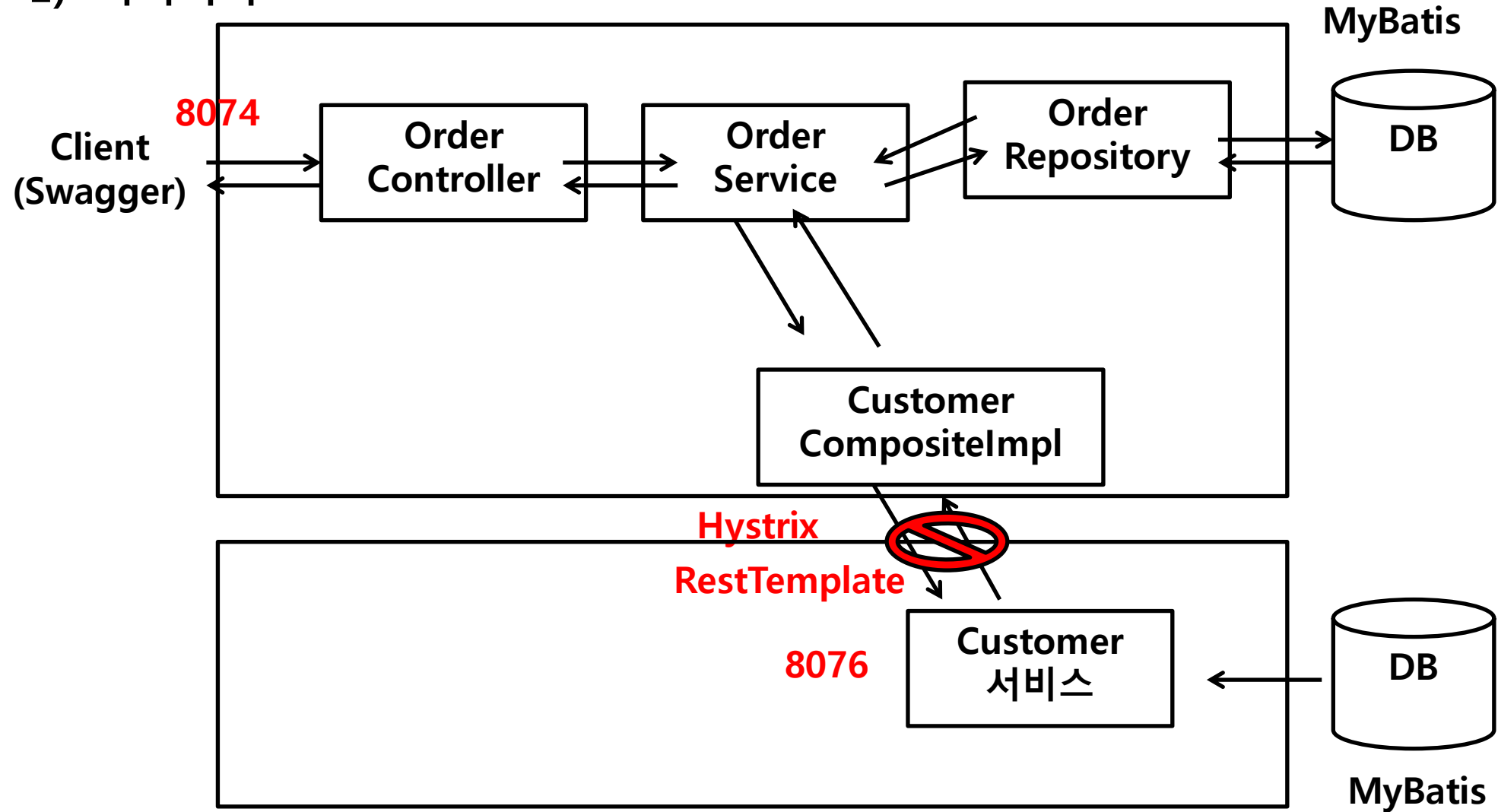
#### 1) 시나리오

Order 서비스에서 주문등록시 고객기본정보를 Customer 서비스에서 조회하는 코드를 구현해 본다. 그리고 Customer 서비스에서 응답 지연 시 Order 서비스로 장애가 전파되는 것을 방지하는 Circuit Breaker의 구현체로서 Hystrix를 적용해 본다.



### 3. Circuit Breaker 적용 시나리오

#### 2) 아키텍처



## 4. Circuit Breaker – Hystrix 적용

### Hystrix 라이브러리 적용

#### 가. 라이브러리 등록

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

#### 나. Service에 @HystrixCommand와 Fallback 메서드 추가

```
@Override
@HystrixCommand(fallbackMethod = "getCustomerFallback", commandKey = "retrieveCustomer")
public Customer retrieveCustomer(String userid) throws Exception {
    return restTemplate.getForObject(CUSTOMER_API_URL+"/rest/customers/"+userid,
        Customer.class);
}

public Customer getCustomerFallback(String userid) throws Exception {
    String msg = "Error: " + userid + "에 해당하는 고객 정보 조회가 지연되고 있습니다.";
    System.out.println(msg);
    throw new Exception();
}
```

@HystrixCommand 메서드와 fallbackMethod의 signature는 일치해야 된다.

## 4. Circuit Breaker – Hystrix 적용

### 다. application.properties 에 hystrix 속성값 설정

```
# hystrix
# If you do not assign command key, Hystrix use 'default' as a key.
#TODO : hystrix 설정 추가
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=10000
hystrix.command.retrieveCustomer.execution.isolation.thread.timeoutInMilliseconds=5000
```

### 라. Application.java 에 @EnableCircuitBreaker 어노테이션 추가

```
@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
public class EcommerceOrderApplication {

    public static void main(String[] args) {
        SpringApplication.run(EcommerceOrderApplication.class, args);
    }
}
```

### 마. CustomerController 에 인위적인 지연 코드 설정

```
@ApiOperation(value = "고객 조회", httpMethod = "GET", notes = "고객 조회")
@GetMapping(value="/rest/customers/{userid}")
public Customer selectCustomerByUserId(@PathVariable("userid") String userid) {

    // 응답지연 코드 추가 ( 10 초 지연 )
    Thread.sleep(10000);

    return customerService.selectCustomerByUserId(userid);
}
```

## 5. Circuit Breaker – Hystrix 속성값

### Hystrix 설정값 변경

@HystrixCommand에 @HystrixProperty를 추가하거나 application.properties 파일을 이용하여 설정값을 변경할 수 있다.

```
@Service
public class MyService {
    @HystrixCommand(fallbackMethod = "defaultDoSomething",
        commandProperties = {
            @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "2"),
            @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "500"),
            @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "1"),
            @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "1000")
        })
    public void doSomething(int input) {
        System.out.println("output: " + 10 / input);
    }

    public void defaultDoSomething(int input, Throwable throwable) {
        System.out.printf("fallback, input:%s, exception:%s%n", input, throwable);
    }
}
```



## 5. Circuit Breaker – Hystrix 속성값

<https://github.com/Netflix/Hystrix/wiki/Configuration>



### Hystrix 주요 설정

Hystrix 기본 설정은 다음과 같이 설정한다.

```
hystrix.commad.default.{property}
```

commandKey를 사용하여 Hystrix 별로 설정을 다르게 하는 경우 다음과 같이 설정한다.

```
hystrix.commad.{commandKey}.{property}
```

## 5. Circuit Breaker – Hystrix 속성값

`execution.isolation.thread.timeoutInMilliseconds` (기본값은 1초, 1000)

➔ Hystrix가 적용된 메서드의 타임아웃을 지정한다. 이 타임아웃내에 메서드가 완료되지 못하면

Fallback 메서드가 호출된다.

`metrics.rollingStats.timeInMilliseconds` (기본: 10초, 10000)

=> 서킷브레이커가 열리기 위한 조건을 체크할 시간. 아래 조건들과 함께 조건을 지정한다.

예를 들어 “10초간 50% 실패하면 서킷브레이커 발동” 이라는 조건이면 여기서 10초를 말한다.

`circuitBreaker.errorThresholdPercentage`

=> 서킷브레이커가 발동할 에러 퍼센트 지정. 기본값은 50

`circuitBreaker.requestVolumeThreshold` ( 기본값은 20 )

=> 서킷브레이커가 열리기 위한 최소 요청조건이다.

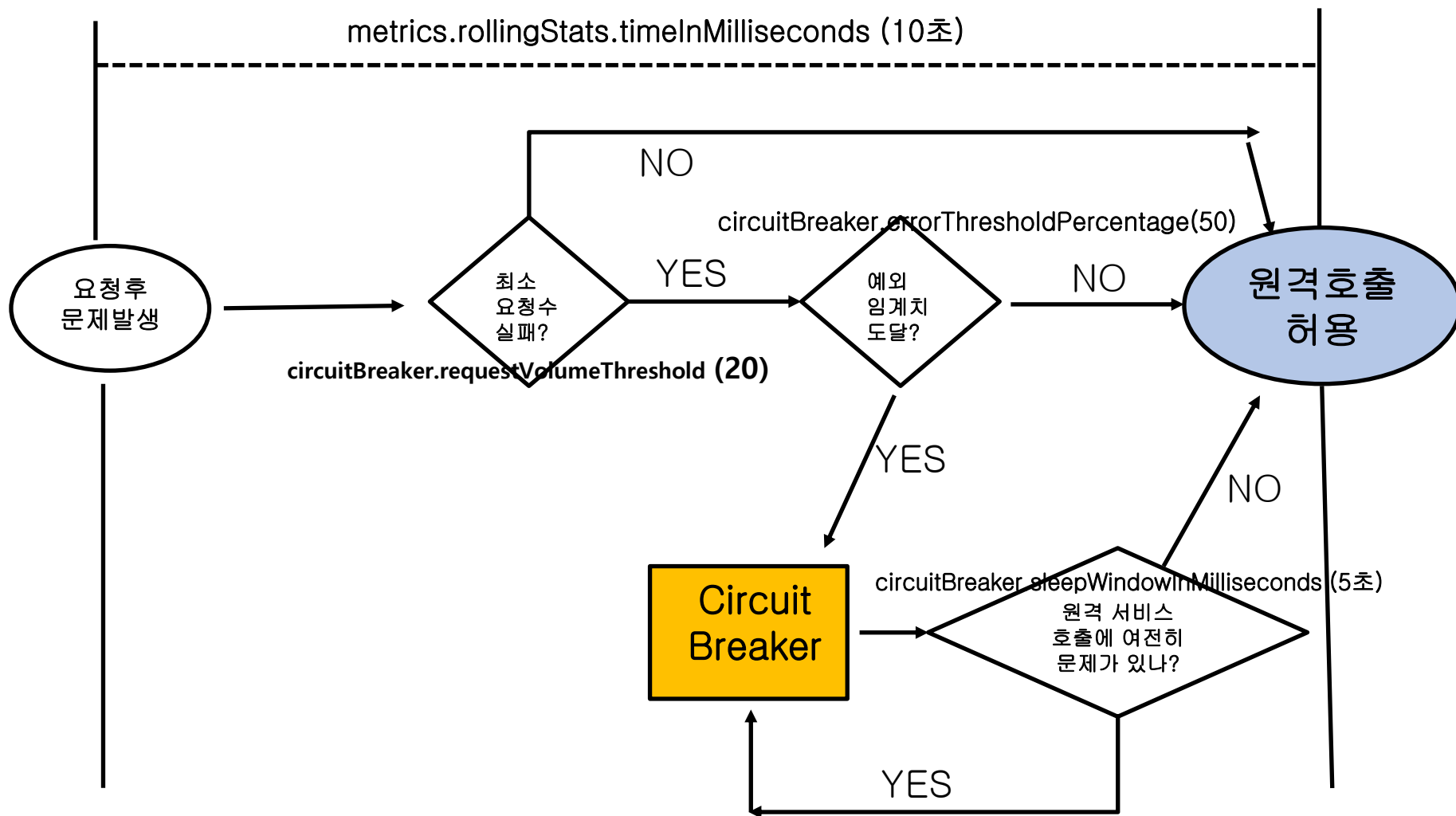
만약 이 값이 20으로 설정되어 있다면 10초간 19개의 요청이 들어와서 19개가 전부 실패하더라도

서킷브레이커는 열리지 않는다.

`circuitBreaker.sleepWindowInMilliseconds` ( 기본값은 5초, 5000)

=> 서킷 브레이커가 열렸을 때 얼마나 지속될지를 설정한다.

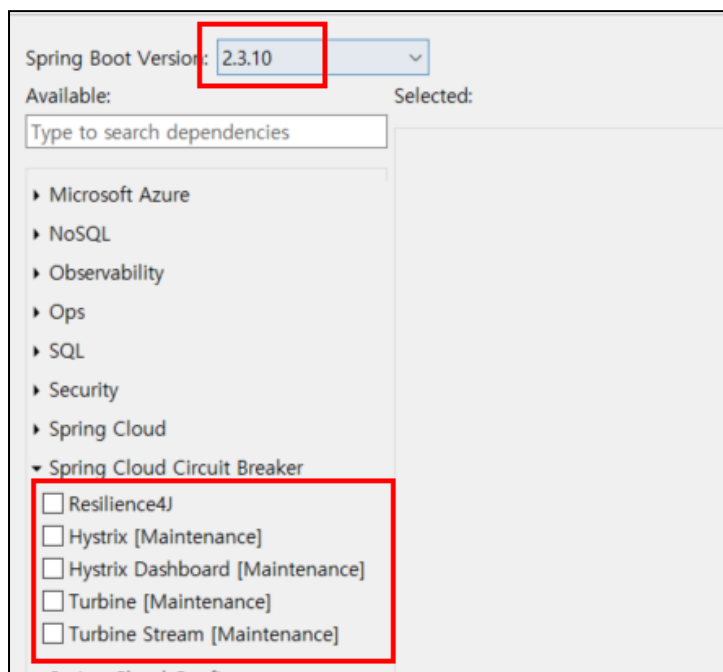
## 6. Circuit Breaker – Hystrix 회로차단 결정과정



## 7. SpringBoot 버전에 따른 의존성 지원 여부

### 버전에 따른 의존성 차이

SpringBoot 2.3.X



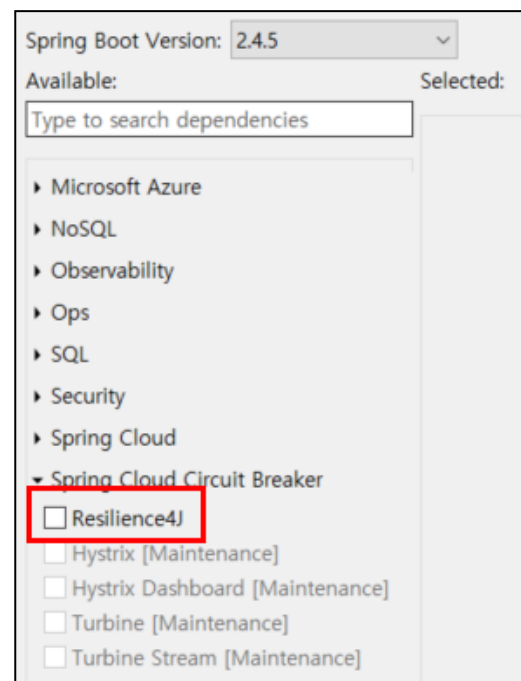
Spring Boot Version: 2.3.10

Available: Selected:

Type to search dependencies

- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▼ Spring Cloud Circuit Breaker
  - ☐ Resilience4J
  - ☐ Hystrix [Maintenance]
  - ☐ Hystrix Dashboard [Maintenance]
  - ☐ Turbine [Maintenance]
  - ☐ Turbine Stream [Maintenance]

SpringBoot 2.4.X



Spring Boot Version: 2.4.5

Available: Selected:

Type to search dependencies

- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▼ Spring Cloud Circuit Breaker
  - ☐ Resilience4J
  - ☐ Hystrix [Maintenance]
  - ☐ Hystrix Dashboard [Maintenance]
  - ☐ Turbine [Maintenance]
  - ☐ Turbine Stream [Maintenance]