초격차 패키지 Online.

IT 서비스 회사에서 사용하는 진짜 프로젝트 가성비있게 맛보기

Clip 1 | 비동기 프로그래밍 구현해보기

Clip 2 | 실수하기 딱 좋은 비동기 프로그래밍

Clip 3 | 실무 스타일로 Feign Client 사용해보기

Clip 4 | Log를 관리하기 위한 Logback 설정

Chapter 2에서 다룰 내용들

- Spring에서 비동기 프로그래밍
- 외부 요청을 위한 Feign Client 학습
- Log를 관리하기 위한 Logback 설정

초보 개발자티 벗어나기

1 비동기 프로그래밍 구현해보기

비동기 프로그래밍이란?

- Async 한 통신
- **실시간성 응답**을 필요로 하지 않는 상황에서 사용
- ex) Notification, Email 전송, Push 알림

비동기 프로그래밍이란?

- 개발자답게 표현해보자면
- Main Thread가 Task를 처리하는 게 아니라
- Sub Thread에게 Task를 위임하는 행위라고 말할 수 있다.

비동기 프로그래밍이란?

- 개발자답게 표현해보자면
- Main Thread가 Task를 처리하는 게 아니라
- Sub Thread에게 Task를 위임하는 행위라고 말할 수 있다.

- **그러면 여기서 Sub Thread는 어떻게 생성하고**
- **어떻게 관리를 해야 할까?**

Spring에서 비동기 프로그래밍

- 우리는 Spring Framework를 사용한다.
- Spring에서 비동기 프로그래밍을 하기 위해선
- ThreadPool을 정의할 필요가 있다.

ThreadPool 생성이 필요한 이유

- 비동기는 Main Thread가 아닌 Sub Thread에서 작업이 진행
- Java에서는 ThreadPool을 생성하여 Async 작업을 처리

ThreadPool 생성 옵션

1. CorePoolSize
2. MaxPoolSize
3. WorkQueue
4. KeepAliveTime

Constructor Detail

ThreadPoolExecutor

```
public ThreadPoolExecutor(int corePoolSize,
                  int maximumPoolSize,
                  long keepAliveTime,
                  TimeUnit unit,
                  BlockingQueue<Runnable> workQueue)
```

Creates a new `ThreadPoolExecutor` with the given initial parameters and default thread factory and rejected execution handler. It may be more convenient to use one of the `Executors` factory methods instead of this general purpose constructor.

Parameters:

`corePoolSize` - the number of threads to keep in the pool, even if they are idle, unless `allowCoreThreadTimeOut` is set

`maximumPoolSize` - the maximum number of threads to allow in the pool

`keepAliveTime` - when the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating.

`unit` - the time unit for the `keepAliveTime` argument

`workQueue` - the queue to use for holding tasks before they are executed. This queue will hold only the `Runnable` tasks submitted by the `execute` method.

Throws:

`IllegalArgumentException` - if one of the following holds:
`corePoolSize < 0`
`keepAliveTime < 0`
`maximumPoolSize <= 0`
`maximumPoolSize < corePoolSize`

`NullPointerException` - if `workQueue` is null

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html

ThreadPoolExecutor 생성자

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue) {
    this(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue,
         Executors.defaultThreadFactory(), defaultHandler);
}
```

Q. 다음 코드를 설명해보시오

```
ThreadPoolExecutor executorPool =
    new ThreadPoolExecutor(5, 10, 3, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(50));
```

ThreadPool 생성시 주의해야할 부분 - (1)

- CorePoolSize 값을 너무 크게 설정할 경우 Side Effect 고려해보기

ThreadPool 생성시 주의해야할 부분 - (2)

Throws:

`IllegalArgumentException` - if one of the following holds:
`corePoolSize < 0`
`keepAliveTime < 0`
`maximumPoolSize <= 0`
`maximumPoolSize < corePoolSize`

`NullPointerException` - if `workQueue` is null

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html

ThreadPool 정리 - CorePoolSize


```
if ( Thread 수 < CorePoolSize )
    New Thread 생성

if ( Thread 수 > CorePoolSize )
    Queue에 요청 추가
```

ThreadPool 정리 - MaxPoolSize


```
if ( Queue Full && Thread 수 < MaxPoolSize )
    New Thread 생성

if ( Queue Full && Thread 수 > MaxPoolSize )
    요청 거절
```

초보 개발자티 벗어나기

2 실수하기 딱 좋은 비동기 프로그래밍

Spring Async 환경 세팅 – Config


```java
@Configuration
public class AppConfig {

    @Bean(name = "defaultTaskExecutor", destroyMethod = "shutdown")
    public ThreadPoolTaskExecutor defaultTaskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(200);
        executor.setMaxPoolSize(200);
        return executor;
    }

    @Bean(name = "messagingTaskExecutor", destroyMethod = "shutdown")
    public ThreadPoolTaskExecutor messagingTaskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(200);
        executor.setMaxPoolSize(200);
        return executor;
    }
}
```

```java
@Configuration
@EnableAsync
public class AsyncConfig {
}
```

Spring Async 환경 세팅 - Controller, Service 1


```java
@RestController
@RequiredArgsConstructor
public class AsyncController {

    private final AsyncService asyncService;

    @GetMapping("/1")
    public String asyncCall_1() {
        asyncService.asyncCall_1();
        return "success";
    }

    @GetMapping("/2")
    public String asyncCall_2() {
        asyncService.asyncCall_2();
        return "success";
    }

    @GetMapping("/3")
    public String asyncCall_3() {
        asyncService.asyncCall_3();
        return "success";
    }
}
```

```java
@Service
@RequiredArgsConstructor
public class AsyncService {

    private final EmailService emailService;

    public void asyncCall_1() {
        System.out.println("[asyncCall_1] :: " + Thread.currentThread().getName());
        emailService.sendMail();
        emailService.sendMailWithCustomThreadPool();
    }

    public void asyncCall_2() {
        System.out.println("[asyncCall_2] :: " + Thread.currentThread().getName());
        EmailService emailService = new EmailService();
        emailService.sendMail();
        emailService.sendMailWithCustomThreadPool();
    }

    public void asyncCall_3() {
        System.out.println("[asyncCall_3] :: " + Thread.currentThread().getName());
        sendMail();
    }

    @Async
    public void sendMail() {
        System.out.println("[sendMail] :: " + Thread.currentThread().getName());
    }
}
```

Spring Async 환경 세팅 - Controller, Service 2


```java
@Service
@RequiredArgsConstructor
public class EmailService {

    @Async("defaultTaskExecutor")
    public void sendMail() {
        System.out.println("[sendMail] :: " + Thread.currentThread().getName());
    }

    @Async("messagingTaskExecutor")
    public void sendMailWithCustomThreadPool() {
        System.out.println("[sendMailWithCustomThreadPool] :: " + Thread.currentThread().getName());
    }
}
```

Spring에서 Async 사용

```
public void asyncCall_1() {
    System.out.println("[asyncCall_1] :: "
                       + Thread.currentThread().getName());
    emailService.sendMail();
    emailService.sendMailWithCustomThreadPool();
}
```

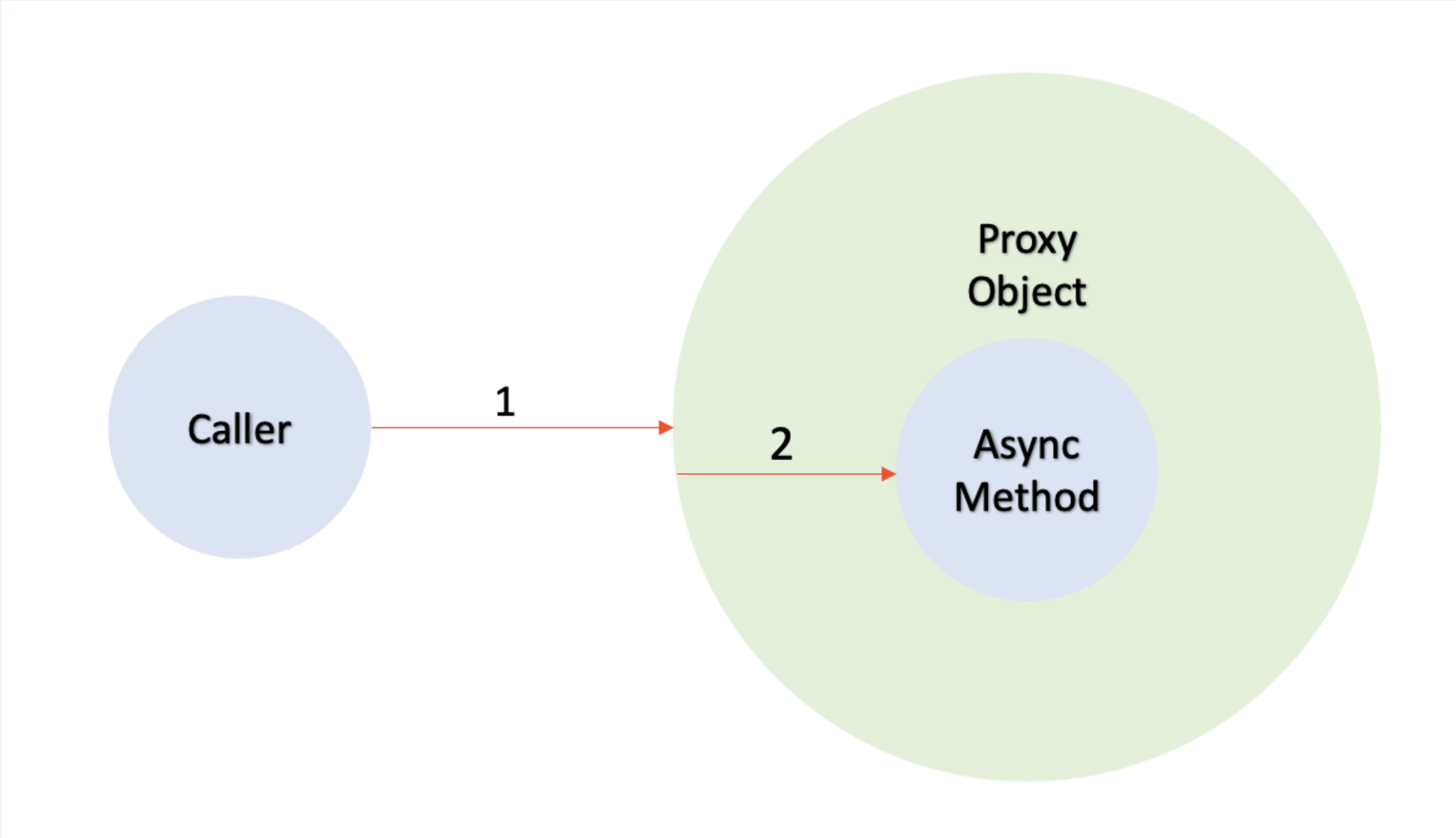
```
@Async("defaultTaskExecutor")
public void sendMail() {
    System.out.println("[sendMail] :: "
                       + Thread.currentThread().getName());
}

@Async("messagingTaskExecutor")
public void sendMailWithCustomThreadPool() {
    System.out.println("[sendMailWithCustomThreadPool] :: "
                       + Thread.currentThread().getName());
}
```

Output

```
[asyncCall_1] :: http-nio-8080-exec-1
[sendMail] :: defaultTaskExecutor-1
[sendMailWithCustomThreadPool] :: messagingTaskExecutor-1
```

Spring에서 Async 동작 원리



Spring에서 Async 사용 시 주의해야할 부분 - (1)

```java
public void asyncCall_2() {
    System.out.println("[asyncCall_2] :: "
                       + Thread.currentThread().getName());
    EmailService emailService = new EmailService();
    emailService.sendMail();
    emailService.sendMailWithCustomThreadPool();
}
```

```java
@Async("defaultTaskExecutor")
public void sendMail() {
    System.out.println("[sendMail] :: "
                       + Thread.currentThread().getName());
}

@Async("messagingTaskExecutor")
public void sendMailWithCustomThreadPool() {
    System.out.println("[sendMailWithCustomThreadPool] :: "
                       + Thread.currentThread().getName());
}
```

Output

```
[asyncCall_2] :: http-nio-8080-exec-2
[sendMail] :: http-nio-8080-exec-2
[sendMailWithCustomThreadPool] :: http-nio-8080-exec-2
```

Spring에서 Async 사용 시 주의해야할 부분 - (2)

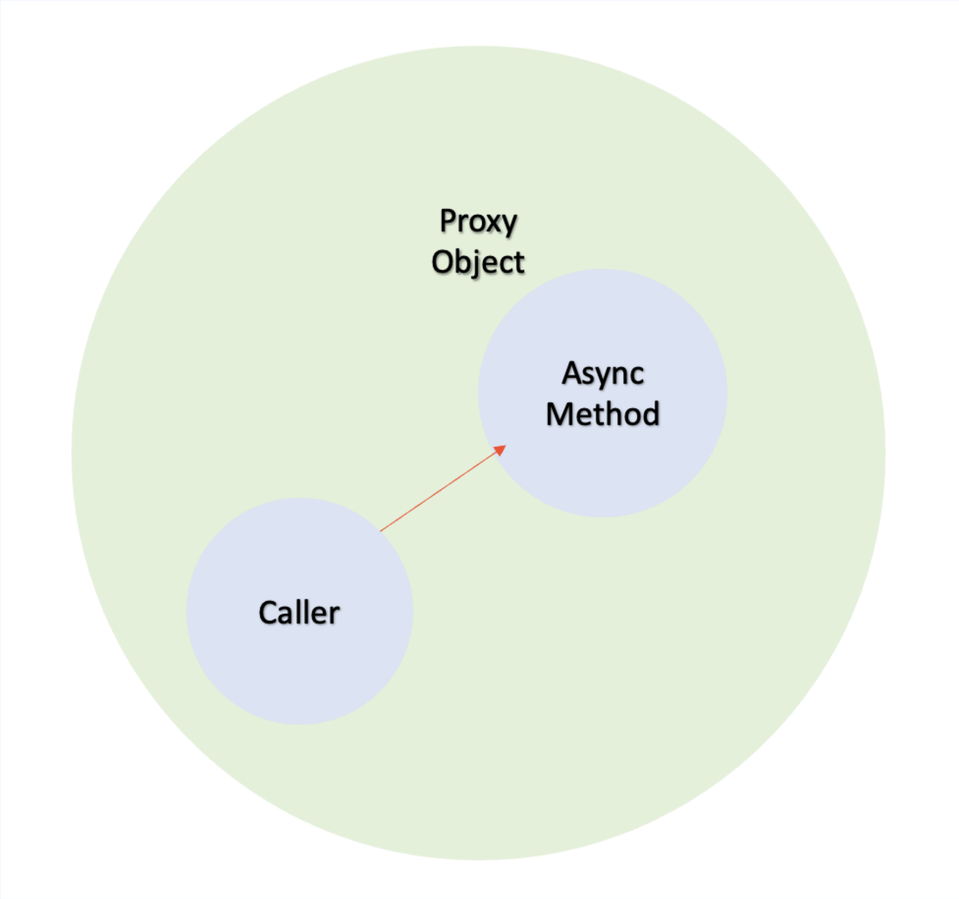
```java
public void asyncCall_3() {
    System.out.println("[asyncCall_3] :: " + Thread.currentThread().getName());
    sendMail();
}

@Async
public void sendMail() {
    System.out.println("[sendMail] :: " + Thread.currentThread().getName());
}
```

Output

```
[asyncCall_3] :: http-nio-8080-exec-3
[sendMail] :: http-nio-8080-exec-3
```

Spring에서 Async 사용 시 주의해야할 부분 – 문제 원인



초보 개발자티 벗어나기

3 실무 스타일로 Feign Client 사용해보기

Feign Client

- Feign 이란 선언적으로 사용할 수 있는 Client이다.
- (= Feign is a declarative web service client.)
- ref : https://docs.spring.io/spring-cloud-openfeign/docs/current/reference/html

Feign Feature

- Connection/Read Timeout
- Feign Interceptor
- Feign Logger
- Feign ErrorDecoder

Connection/Read Timeout

- 외부 서버와 통신 시
- Connection / Read Timeout
- 설정이 가능하다.

```yaml
feign:
  url:
    prefix: http://localhost:8080/target_server # DemoFeignClient에서 사용할 url prefix 값
  client:
    config:
      default:
        connectTimeout: 1000
        readTimeout: 3000
        loggerLevel: NONE
      demo-client: # DemoFeignClient에서 사용할 Client 설정 값
        connectTimeout: 1000
        readTimeout: 10000
        loggerLevel: HEADERS # 여기서 설정한 값은 FeignCustomLogger -> Logger.Level logLevel 변수에 할당됨

#  [loggerLevel 옵션]
#      ref : feign.Logger.Level
#  ```
#  NONE, // No logging.
#  BASIC, // Log only the request method and URL and the response status code and execution time.
#  HEADERS, // Log the basic information along with request and response headers.
#  FULL // Log the headers, body, and metadata for both requests and responses.
# ```
```

Feign Interceptor

- 외부로 요청이 나가기 전에
- 만약 공통적으로
- 처리해야하는 부분이 있다면
- Interceptor를 재정의하여
- 처리가 가능하다.

```java
@RequiredArgsConstructor(staticName = "of")
public final class DemoFeignInterceptor implements RequestInterceptor {

    @Override
    public void apply(RequestTemplate template) { // 필요에 따라 template 필드 값을 활용하자!

        // get 요청일 경우
        if (template.method() == HttpMethod.GET.name()) {
            System.out.println("[GET] [DemoFeignInterceptor] queries : " + template.queries());
            // ex) [GET] [DemoFeignInterceptor] queries : {name=[CustomName], age=[1]}
            return;
        }

        // post 요청일 경우
        String encodedRequestBody = StringUtils.toEncodedString(template.body(), UTF_8);
        System.out.println("[POST] [DemoFeignInterceptor] requestBody : " + encodedRequestBody);
        // ex) [POST] [DemoFeignInterceptor] requestBody : {"name":"customName","age":1}

        // Do Something
        // ex) requestBody 값 수정 등등

        // 새로운 requestBody 값으로 설정
        template.body(encodedRequestBody);
    }
}
```

Feign CustomLogger

- Request / Response 등
- 운영을 하기 위한
- 적절한 Log를 남길 수 있다.

```java
@Slf4j
@RequiredArgsConstructor
public class FeignCustomLogger extends Logger {
    private static final int DEFAULT_SLOW_API_TIME = 3_000;
    private static final String SLOW_API_NOTICE = "Slow API";

    @Override
    protected void log(String configKey, String format, Object... args) {
        // log를 어떤 형식으로 남길지 정해준다.
        System.out.println(String.format(methodTag(configKey) + format, args));
    }

    @Override
    protected void logRequest(String configKey, Logger.Level logLevel, Request request) {
        /**
         * [값]
         * configKey = DemoFeignClient#callGet(String,String,Long)
         * logLevel = BASIC # "feign.client.config.demo-client.loggerLevel" 참고
         *
         * [동작 순서]
         * `logRequest` 메소드 진입 -> 외부 요청 -> `logAndRebufferResponse` 메소드 진입
         *
         * [참고]
         * request에 대한 정보는
         * `logAndRebufferResponse` 메소드 파라미터인 response에도 있다.
         * 그러므로 request에 대한 정보를 [logRequest, logAndRebufferResponse] 중 어디에서 남길지 정하면 된다.
         * 만약 `logAndRebufferResponse`에서 남긴다면 `logRequest`는 삭제해버리자.
         */
        System.out.println(request);
    }

```

Feign CustomLogger

3.
실무 스타일로 Feign Client 사용해보기

```
47      @Override
48      protected Response logAndRebufferResponse(String configKey, Logger.Level logLevel,
49                                                Response response, long elapsedTime) throws IOException {
50          /**
51           * [참고]
52           * - `logAndRebufferResponse` 메소드내에선 Request, Response에 대한 정보를 log로 남길 수 있다.
53           * - 매소드내 코드는 "feign.Logger#logAndRebufferResponse(java.lang.String, feign.Logger.Level, feign.Response, long)"에서 가져왔다.
54           *
55           * [사용 예]
56           * 예상 요청 처리 시간보다 오래 걸렸다면 "Slow API"라는 log를 출력시킬 수 있다.
57           * ex) [DemoFeignClient#callGet] <--- HTTP/1.1 200 (115ms)
58           *     [DemoFeignClient#callGet] connection: keep-alive
59           *     [DemoFeignClient#callGet] content-type: application/json
60           *     [DemoFeignClient#callGet] date: Sun, 24 Jul 2022 01:26:05 GMT
61           *     [DemoFeignClient#callGet] keep-alive: timeout=60
62           *     [DemoFeignClient#callGet] transfer-encoding: chunked
63           *     [DemoFeignClient#callGet] {"name":"customName","age":1,"header":"CustomHeader"}
64           *     [DemoFeignClient#callGet] [Slow API] elapsedTime : 3001
65           *     [DemoFeignClient#callGet] <--- END HTTP (53-byte body)
66           */
67
68          String protocolVersion = resolveProtocolVersion(response.protocolVersion());
69          String reason = response.reason() != null
70                          && logLevel.compareTo(Level.NONE) > 0 ? " " + response.reason() : "";
71          int status = response.status();
72          log(configKey, "<--- %s %s%s (%sms)", protocolVersion, status, reason, elapsedTime);
73          if (logLevel.ordinal() >= Level.HEADERS.ordinal()) {
74
75              for (String field : response.headers().keySet()) {
76                  if (shouldLogResponseHeader(field)) {
77                      for (String value : valuesOrEmpty(response.headers(), field)) {
78                          log(configKey, "%s: %s", field, value);
79                      }
80                  }
81              }
82
83              int bodyLength = 0;
84              if (response.body() != null && !(status == 204 || status == 205)) {
85                  // HTTP 204 No Content "...response MUST NOT include a message-body"
86                  // HTTP 205 Reset Content "...response MUST NOT include an entity"
87                  if (logLevel.ordinal() >= Level.FULL.ordinal()) {
88                      log(configKey, ""); // CRLF
89                  }
90                  byte[] bodyData = Util.toByteArray(response.body().asInputStream());
91                  bodyLength = bodyData.length;
92                  if (logLevel.ordinal() >= Level.HEADERS.ordinal() && bodyLength > 0) {
93                      log(configKey, "%s", decodeOrDefault(bodyData, UTF_8, "Binary data"));
94                  }
95                  if (elapsedTime > DEFAULT_SLOW_API_TIME) {
96                      log(configKey, "[%s] elapsedTime : %s", SLOW_API_NOTICE, elapsedTime);
97                  }
98                  log(configKey, "<--- END HTTP (%s-byte body)", bodyLength);
99                  return response.toBuilder().body(bodyData).build();
100             } else {
101                 log(configKey, "<--- END HTTP (%s-byte body)", bodyLength);
102             }
103         }
104         return response;
105     }
```

Feign ErrorDecoder

- 요청에 대해
- 정상 응답이 아닌 경우
- 핸들링이 가능하다.

```java
public final class DemoFeignErrorDecoder implements ErrorDecoder {
    private final ErrorDecoder errorDecoder = new Default();

    @Override
    public Exception decode(String methodKey, Response response) {
        final HttpStatus httpStatus = HttpStatus.resolve(response.status());

        /**
         * [참고]
         * 외부 컴포넌트와 통신 시
         * 정의해놓은 예외 코드 일 경우엔 적절하게 핸들링하여 처리한다.
         */
        if (httpStatus == HttpStatus.NOT_FOUND) {
            System.out.println("[DemoFeignErrorDecoder] Http Status = " + httpStatus);
            throw new RuntimeException(String.format("[RuntimeException] Http Status is %s", httpStatus));
        }

        return errorDecoder.decode(methodKey, response);
    }

}
```

Feign 프로젝트 실습

3.

실무 스타일로 Feign Client 사용해보기

- 실제로 코딩을 해봅시다 !

초보 개발자티 벗어나기

4 Log를 관리하기 위한 Logback 설정

Logback 개념 (1)

- SLF4J(Simple Logging Facade for Java)라는 인터페이스를 구현하는 구현체이다.

(Simple Logging Facade for Java provides a Java logging API by means of a simple facade pattern. The underlying logging backend is determined at runtime by adding the desired binding to the classpath and may be the standard Sun Java logging package java.util.logging, log4j, reload4j, logback or tinylog.)

- 즉 Logging Framework라고 생각하면 된다.

Logback 개념 (2)

Appender 종류

- ConsoleAppender : 콘솔에 log를 출력
- FileAppender : 파일 단위로 log 를 저장
- RollingFileAppender : (설정 옵션에 따라) log 를 여러 파일로 나누어 저장
- SMTPAppender : log 를 메일로 전송 하여 기록
- DBAppender: log 를 DB에 저장

Logback 사용 이유

운영을 위해서는 Log는 반드시 필요하다.

Logback 설정을 하여 운영을 하기 위한 Log를 관리한다.

뿐만 아니라 데이터는 돈이므로 Log는 곧 값 비싼 자산이다.

Logback 설정 및 실습

4.

Log를 관리하기 위한
Logback 설정

- 실제로 코딩을 해봅시다 !