[4주차] 로깅 (SLF4J, Logback)

로깅의 중요성과 장점

- 로깅이란?

  : 정보를 제공하는 일련의 기록인 로그(log)를 생성하도록 시스템을 작성하는 활동이다. 버그에 대한 유용한 정보를 제공할 수 있다.

  ex) System.out.println() 과 로깅 라이브러리 사용

- 중요성과 장점

  1. 상황에 따라 level 별 메시지 기능
  2. 프로그램의 실행에 대한 흐름과 에러 확인 가능
  3. 자유로운 출력 형식과 위치 지정 가능
  4. 프레임워크를 이용하여 쉽게 설정 가능

로깅(Loggin) 라이브러리

- **Log4j** - 가장 오래된 로깅 프레임워크. 2015년 기준으로 개발이 중단되었다.
- **Logback** - log4j 이후 출시된 로깅 프레임워크이다. logback은 Spring boot의 spring-boot-web 안에 spring-boot-starter-logging에 logback이 포함되어 있다.
- **Log4j2** - 가장 최근에 등장, Logback과 같은 기능을 가지고 있다.

---

slf4j 소개

- Simple Logging Facade For Javs로 logger 추상체이다.
- logback이나 log4j와 같은 프레임워크의 인터페이스 역할을 해준다.
- log4j를 사용하다가 log4j2로 교체하면 많은 코드 수정이 발생하는데 이러한 점을 고려해서 slf4j를 사용하고 log4j를 연결하여 사용하면 이후에 수정이 쉽다.

Logback 설정 방법 (logback-spring.xml)

- logback-spring.xml 은 크게 appender와 logger, root로 구성된다.

- `logger` : <logger> 태그로 로그의 name 속성을 통해 class 별로 지역설정을 할 수 있으며 additivity 속성을 통해 log level 을 상속 유무 설정이 가능하다.
- `root` : <root> 태그로 구성되며 logger 중에 전역 logger를 root로 따로 분리한 것이다. 전역 설정이기 때문에 name 속성이 없으며 level 속성을 통해 log level만 지정해 주면 된다.
- `appender` :<appender> 태그로 구성되며 log 메시지가 출력될 대상을 결정

  가장 많이 사용되는 속성은 `ConsoleAppender` 과 `RollingFileAppender`

  - `ConsoleAppender` : 콘솔에 로그를 찍음. 로그를 OutputStream에 작성하여 콘솔에 출력되도록 한다.
  - `RollingFileAppender` : 여러 개의 파일을 롤링, 순회하면서 로그를 찍는다. (FileAppender를 상속 받는다.) 지정 용량이 넘어간 Log File을 넘버링하여 나누어 저장할 수 있다.

- logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
        <!-- 1. 로그 출력 패턴 설정 : 날짜, 시간, 스레드 이름, 로그 레벨,
    <property name="CONSOLE_PATTERN" value="%d{yyyy-MM-dd HH:mm:
        <property name="FILE_PATTERN" value="%d{yyyy-MM-dd HH:mm

    <!-- 2. Console appender 설정 : 로그를 콘솔에 출력하도록 설정 -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAp
        <encoder>
                <!-- 1번에 지정한 CONSOLE_PATTERN 패턴을 참조-->
            <Pattern>${CONSOLE_PATTERN}</Pattern>
        </encoder>
    </appender>

      <!-- 3. RollingFileAppender 설정 : 로그를 파일로 저장하고 일정 조
      <appender name="FILE" class="ch.qos.logback.core.rolling.R
          <!-- 로그 파일 위치 설정 -->
          <file>logs/app.log</file>

          <!-- 롤링 정책 설정 -->
          <rollingPolicy class="ch.qos.logback.core.rolling.Time
              <!-- 롤링된 파일 이름 패턴 (날짜 기반으로 새로운 파일 생성)
```

```
            <fileNamePattern>logs/app.%d{yyyy-MM-dd}.log</file

            <!-- 보관할 로그 파일 최대 개수 -->
            <maxHistory>30</maxHistory> <!-- 최근 30일치 로그 보관
        </rollingPolicy>

        <encoder>
                <!-- 1번에 지정한 FILE_PATTERN 패턴을 참조-->
            <Pattern>${FILE_PATTERN}</Pattern>
        </encoder>
    </appender>

    <!-- 4. name에 있는 패키지의 로그를 DEBUG 레벨로 출력 -->
    <logger name="com.example.todolist.service" level="DEBUG" ad
        <appender-ref ref="STDOUT"/>
    </logger>

        <!-- root Logger 설정 -->
        <!-- 모든 로그 레벨 중 INFO 이상의 로그를 출력 -->
    <root level="INFO">
        <appender-ref ref="STDOUT"/>
    </root>
</configuration>
```

로그 레벨 (TRACE, DEBUG, INFO, WARN, ERROR)

- 로그 레벨 순서 : TRACE > DEBUG > INFO > WARN > ERROR
- TRACE : DEBUG 보다 상세한 정보를 보여준다. 모든 레벨에 대한 로깅이 추적
- DEBUG : 개발 단계에서 사용하며, 일반 정보를 나타낼 때 사용
- INFO : 상태 변경과 운영에 참고할만한 정보성 로그를 표시한다.
- WARN : 처리 가능한 문제, 향후 에러의 원인이 될 수 있는 경고성 메시지
- ERROR : 오류가 발생

로그 패턴 설정

패턴	설명	예시 출력
%p	로그 메시지의 우선순위 (priority)	`INFO`, `DEBUG`, `WARN`
%m	로그 메시지 내용	This is a log message
%d	로그 발생 시간 (포맷 가능)	2024-11-20 14:32:15
%d{HH:mm:ss,SSS}	시간:분:초, 밀리초	14:32:15,123
%d{yyyy MMM dd HH:mm:ss,SSS}	연도 월 일 시간:분:초, 밀리초	2024 Nov 20 14:32:15,123
%t	로그 발생 쓰레드 이름	`main`, `worker-thread`
%%	% 문자 출력	%
%n	개행 문자	(줄바꿈)
%c	카테고리 이름	com.example.MyClass
%l	로그 호출자의 정보 (클래스, 메서드, 파일, 라인 번호)	com.example.MyClass.myMethod(MyClass.java:42)
%M	로그 호출 메서드 이름	myMethod
%F	로그 발생 소스 파일명	MyClass.java
%L	로그 호출 코드의 라인 번호	42

파일 로깅과 로그 로테이션

- 파일 로깅 (File Logging)

  : 애플리케이션이 생성한 로그 메시지를 파일에 기록하는 방식이다.

  - 특징

    1. 실행 상태나 오류 정보를 기록
    2. .log 혹은 .txt 파일로 로그 파일의 이름과 경로를 설정할 수 있다.
    3. 애플리케이션 종료 후에도 로그 데이터를 보존할 수 있어 에러를 추적하는데 유용하다.

- 로그 로테이션 (Log Rotation)

: 일정 크기 이상이 되거나 특정 기간이 지나면 기존 로그 파일을 백업하고 새 로그 파일을 생성하는 방식이다.

- 특징
  1. 로그 파일이 너무 커져서 저장 공간을 차지하는 것을 방지한다.
  2. 일별, 주별, 월별로 파일을 분리하여 관리할 수 있다.
  3. 이름 규칙을 지정하여 로그 파일을 보관할 수 있다.

MDC (Mapped Diagnostic Context)

- MDC란?

  : Map 형식을 이용하여 클라이언트 특징적인 데이터를 저장하기 위한 메커니즘이다. 스레드 단위로 데이터를 읽고 쓸 수 있는 기능이다.

  - slf4j, logback, log4j2 등 Logger에서 MDC를 제공한다.
  - MDC는 key/value 저장소를 지원하며 이 저장소는 ThreadContext에 의존한다.
  - 멀티 스레드 환경에서 각 스레드의 실행 컨텍스트를 구분할 수 있어서 모니터링시 추적에 용이합니다.