Logging

로깅이란?

- 비 기능적 요구사항 (Not Functional Requirement)
- 하지만, 프로그램 개발 중 디버깅 및 개발 완료 후 문제점 추적 및 분석을 위해 필수적으로 갖추어야 할 요구 조건
- 로그(Log)는 프로그램 개발이나 운영시 발생하는 문제점을 추적하거나 운영상태를 모니터링하는 정보
- 로깅(Logging)이란 로그(Log)를 생성하도록 시스템을 작성하는 활동
- 얻을 수 있는 것
  1. 재현하기 힘든 (테스트 환경이 아닌 개발 완료된 환경에서 발생하는) 버그에 대한 유용한 정보
  2. 성능에 관한 통계와 정보를 제공할 수 있음

자바에서의 로깅

- 자바 로깅의 종류
  - java.util.logging
  - Apache의 Jakarta Commons Logging
  - Log4J
  - Log4J의 성능과 기능상의 이유로 대체 로거들이 많아짐
  - 자바 로거의 표준과 일관된 인터페이스가 필요

- SLF4J (Simple Logging Façade)
- Logback(SLF4J의 구현체 중 하나)
  1. Log4J 보다 속도와 메모리 사용량이 개선
  2. XML & Groovy 설정 지원 (자동 리로딩 기능)
  3. 다양한 Appender (console, file, socket, JDBC 등)
  4. Rolling & Archiving (자동 압축)
  5. HTTP 디버깅 (logback-access 컴포넌트)

Logback 라이브러리 추가

- 버전 정보 (property in pom.xml)

```
<properties>
  <org.springframework-version>6.1.9</org.springframework-version>
  <jcloverslf4j.version>2.0.13</jcloverslf4j.version>
  <logback.version>1.5.6</logback.version>
</properties>
```

- SLF4J 기반의 Logback 라이브러리를 로거로 사용할 것이기 때문에 JCL은 제외

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${org.springframework-version}</version>
  <exclusions>
    <exclusion>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Logback 라이브러리 추가

- JCL-over-SLF4J 라이브러리 추가

  JCL을 제외시켰으므로 기존 Commons Logging을 통해 로그를 남기는 코드들은 에러를 발생시킨다.
  이를 이어주어야 하는 브릿지 혹은 일종의 어댑터가 필요

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>jcl-over-slf4j</artifactId>
   <version>${jcloverslf4j.version}</version>
</dependency>
```

- SLF4J 구현체인 Logback 라이브러리 추가

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
</dependency>
```

Logback 설정

logback.xml은 Logback의 설정파일로 /src/main/resources 아래 위치시킨다

- Console Appender

```
<appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
  <encoder>
    <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg %n</Pattern>
  </encoder>
  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <!- 특정 임계값을 넘은 로그만 기록 -->
    <level>TRACE</level>
  </filter>
</appender>
```

- Logger 등록 (in logback.xml)

```
<root level="debug">
    <appender-ref ref="consoleAppender" />
</root>
```

- Log Level
  - TRACE - DEBUG - INFO - WARN - ERROR - FATAL

Logback 설정

- Rolling File Appender

```
<appender name="fileAppender"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>\log-mysite\mysite.log</file>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <Pattern>
        %d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n
        </Pattern>
    </encoder>
    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
        <FileNamePattern>\log-mysite\exec-time.%i.log.zip</FileNamePattern>
        <MinIndex>1</MinIndex>
        <MaxIndex>10</MaxIndex>
    </rollingPolicy>
    <triggeringPolicy
        class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
        <MaxFileSize>1MB</MaxFileSize>
    </triggeringPolicy>
</appender>
```

Logback 설정

- Layout Pattern
  - %d{format}
    - 로그 날짜와 시간 포맷
  - %logger{length}
    - Logger name의 이름을 축약할 수 있음. {length}는 최대 자릿수
  - %thread
    - 현재 Thread name
  - %-5level
    - Log Level. -5는 출력 고정폭 값
  - %msg
    - Log message. %message는 alias
  - %n
    - New Line

Logback 설정
: rollingPolicy

- Logback에서 rollingPolicy는 롤오버된 로그 파일의 관리 방식을 정의함
  - 로그 파일을 어떻게 롤오버할지, 롤오버된 파일을 얼마나 오래 보관할지, 파일 이름을 어떻게 지정할지 등을 설정

- FixedWindowRollingPolicy
  - 롤오버 파일을 순차적으로 인덱싱하며, 오래된 파일부터 삭제

```
<rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
  <fileNamePattern>\log-myportal\logFile.%i.log</fileNamePattern>
  <minIndex>1</minIndex>
  <maxIndex>10</maxIndex>
</rollingPolicy>
```

  - myportal.1.log ~ myportal.10.log를 유지

Logback 설정
: rollingPolicy

- TimeBasedRollingPolicy
  - 시간과 파일 크기를 기준으로 롤오버를 수행
  - 파일이 특정 크기에 도달하거나 지정된 시간 간격이 지나면 롤오버를 실행
  - 로그 파일의 크기를 제어하면서 시간에 따른 정리도 가능하게 함

```
<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <fileNamePattern>logFile.%d{yyyy-MM-dd}.log</fileNamePattern>
    <maxHistory>30</maxHistory>
</rollingPolicy>
```

Logback 설정
: rollingPolicy

- SizeAndTimeBasedRollingPolicy
  - 시간 기준으로 롤오버를 수행
  - 일반적으로 파일 이름 패턴에 날짜와 시간 포맷을 포함하여 특정 시간 간격으로 새 파일을 생성
    - 시간별, 일별, 월별 등 다양한 간격으로 롤오버 설정 가능

```
<rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
    <fileNamePattern>logFile.%d{yyyy-MM-dd}.%i.log</fileNamePattern>
    <maxFileSize>10MB</maxFileSize>
    <maxHistory>30</maxHistory>
</rollingPolicy>
```

Logback 설정
: triggeringPolicy

- SizeBasedTriggeringPolicy
  - 로그 파일의 크기가 설정된 최대값에 도달하면 롤오버를 트리거
  - 특히 로그 파일이 너무 커져서 시스템 성능에 영향을 미칠 수 있는 상황을 방지하는 데 유용

```
<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
    <MaxFileSize>100MB</MaxFileSize>
</triggeringPolicy>
```

Logback 설정

- Logback Debug 모드
  - configuration 노드에 debug 어트리뷰트를 true로 주면 구동시 logback 상태 확인 가능

```
<configuration debug="true">
```

- additivity
  - false로 설정하면 지정한 이름에만 logger가 적용

```
<logger name="com.example.mysite.exception" level="error" additivity="false">
    <appender-ref ref="consoleAppender" />
    <appender-ref ref="fileAppenderException" />
</logger>
```

- root
  - logger 들은 name으로 등록이 되며 기본적으로는 java package 구조와 동일하게 적용.
  - tree 구조이므로 최상단 root를 적용하면 모든 tree 이하에 로거를 적용할 수 있음

Logger의 사용

- 각 로그 레벨에 맞는 메서드들이 존재

```
private static final Logger logger = LoggerFactory.getLogger(GuestbookService.class);

//...

logger.trace("TRACE");
logger.debug("DEBUG");
logger.info("INFO");
logger.warn("WARN");
logger.error("ERROR");
```