# Logback을 이용한 SQL 로깅

# Logback

로그백은 SLF4J 구현체 입니다. 스프링 부트의 web 디펜던시를 선택하면 로그백 디펜던시가 자동으로 추가됩니다. 스프링 레가시인 경우는 디펜던시를 직접 추가하세요.

### 디펜던시

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.11</version>
</dependency>
```

logback-classic을 디펜던시에 추가하면 관련 디펜던시도 같이 추가됩니다.

```
Maven: ch.qos.logback:logback-core:1.1.11
Maven: org.slf4j:slf4j-api:1.7.25
```

### 로그를 남기는 방법

logback을 사용할 수 있는지 테스트 합니다.

```
public class Tutorial {
    // @Slf4j 설정으로 대신할 수 있다.
    private static final Logger logger = LoggerFactory.getLogger(Tutorial.class);

    public static void main(String[] args) {
        logger.trace("trace");
        logger.debug("debug");
        logger.info("info");
        logger.warn("warn");
        logger.error("error");
    }
}
```

Logger와 LoggerFactory는 SLF4J에 있는 interface와 implements입니다. 실제 애플리케이션 코드에서 logger를 정의 할때 로그백 관련 코드를 넣지는 않습니다. 이렇게 사용하면 차후 다른 logging framework로 교체하더라도 코드의 수정이 없이 교체가 가능하게 됩니다.

getLogger() 메소드에 String 대신 .class로 클래스 정보를 넘겨주면 .getName() 메소드를 사용하여 클래스 이름을 사용하게 됩니다.

### 로깅 Level

로깅레벨 5가지는 `trace`, `debug`, `info`, `warn`, `error` 입니다.

- trace라고 선언을 하면 모든 Level을 포함합니다.
- info라고 선언을 하면 info 이상인 info, warn, error들이 포함됩니다.

### 설정파일 적용순서

설정파일을 다음 순서대로 찾아서 발견하면 적용됩니다.

1. logback.groovy 파일을 찾는다.
2. logback-test.xml 파일을 찾는다.
3. logback.xml 파일을 찾는다.
4. 모두 없다면 기본 설정 전략을 따른다.

메이븐 프로젝트를 사용하는 경우에는 test/resources에 logback-test.xml을 두고 테스트가 실행될때는 logback-test.xml이 우선적으로 적용되도록 사용할 수 있습니다.

스프링 부트 환경에서는 logback.xml 이름 대신 logback-spring.xml 이라는 이름으로 설정해야 합니다. logback.xml로 설정하면 스프링부트가 설정하기 전에 로그백 관련한 설정을 하기 때문에 제어할 수가 없게 됩니다.

## Appender

Event마다 Log를 기록하는 기능은 Appender가 처리합니다. Appender를 설정하더래도 로그 출력에 해당되지 않으면 작동하지 않습니다. Appender는 출력될 형식을 직접 가지고 있지 않고, 해당 기능은 Layout과 Encoder에 위임을 합니다.

### ConsoleAppender

ConsoleAppender는 OutputStreamAppender를 상속합니다. encoder, pattern으로 PatternLayoutEncoder가 생성되서 Appender에 주입됩니다.

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!--
        By default, encoders are assigned the type
        ch.qos.logback.classic.encoder.PatternLayoutEncoder
     -->
    <encoder>
        <pattern>
            %d{yyyyMMdd HH:mm:ss.SSS} [%thread] %-3level %logger{5} -%msg %n
        </pattern>
    </encoder>
</appender>
```

### RollingFileAppender

로그가 많아지면 파일 하나당 최대 용량 제한도 있고, 로그를 파악하기도 어렵습니다. 이때는 대부분 날짜 기준으로 파일을 남깁니다. 따로 Crontab으로 매일 파일을 rename해서 처리할 수도 있지만, 로그백은 RollingFileAppender로 처리할 수 있습니다. 옵션중 prudent를 true로 선언하면 file에 저장될때 lock을 생성해서 처리합니다. 서로 다른 java vm이 같은 파일을 가리킬때 사용합니다.

```
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>c:/logs/spring/boot/mybatis/project.log</file>
    <prudent>true</prudent>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>mylog-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
        <!-- 30일 지난 파일은 삭제한다. -->
        <maxHistory>30</maxHistory>
        <timeBasedFileNamingAndTriggeringPolicy
      class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>100MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>INFO</level>
    </filter>
    <encoder>
        <pattern>
      %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level --- [%thread] %logger{35} : %msg %n
        </pattern>
    </encoder>
</appender>
```

timeBasedFileNamingAndTriggeringPolicy로 파일마다 트리거를 걸어 파일 최대 용량을 설정하면 새로운 index이름으로 파일을 생성합니다. 파일 확장자에 .zip을 선언하면 새로운 file이 생성될때 이전 파일은 .zip으로 압축을 할 수 있습니다.

## 기타 기능

또한, ch.qos.logback.classic.net.SMTPAppender를 사용해서 에러가 발생하면 관리자에서 메일을 발송하여 리포팅을 하는 기능을 추가할 수 있습니다.

Logback Site

profile 설정에 따라서 설정을 변경해서 적용할 수 있습니다.

```
<!--  -->
<springProfile name="local">
  <root level="DEBUG">
    <appender-ref ref="FILE" />
    <appender-ref ref="STDOUT" />
  </root>
</springProfile>
<springProfile name="dev">
  <root level="INFO">
    <appender-ref ref="FILE" />
    <appender-ref ref="STDOUT" />
  </root>
</springProfile>
```

# SQL 로깅을 위한 설정

개발 시 마이바티스나 JPA 기술을 사용하는 경우 SQL 쿼리문이 실제로 어떻게 디비에 질의되는지 확인할 수

있다면 도움이 된다. SQL관련 로깅처리를 어떻게 하는지 살펴보자.

logback 은 logback-core, logback-classic, logback-access의 3개의 모듈이 있습니다. core는 classic과 access의 공통라이브러리입니다. maven repository를 쓴다면 classic만 추가하면 관련 라이브러리가 추가 됩니다.

디비 드라이버를 프록싱하여 SQL 쿼리문을 로그로 출력하는 DriverSpy를 위한 디펜던시를 추가합니다.

### pom.xml

```
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
    <version>1.16</version>
</dependency>
```

### application.properties

H2 디비를 대상으로 설정하는 예입니다.

```
#spring.datasource.driverClassName=org.h2.Driver
spring.datasource.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
#spring.datasource.url=jdbc:h2:mem:TESTDB;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.url=jdbc:log4jdbc:h2:mem:TESTDB;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=

logging.config=classpath:logback-spring.xml
```

`org.h2.Driver` 대신 `net.sf.log4jdbc.sql.jdbcapi.DriverSpy` 으로 설정합니다. `jdbc:h2` 문자열 사이에 `log4jdbc` 문자열을 집어넣어 `jdbc:log4jdbc:h2` 처럼 설정합니다.

`logging.config=classpath:logback-spring.xml` 설정을 하지 않아도 기본적으로 `logback-spring.xml` 파일을 찾아서 처리합니다.

### logback-spring.xml

resources 폴더 밑에 파일을 배치합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    jdbc.sqlonly    : Logs only SQL
    jdbc.sqltiming  : Logs the SQL, post-execution, including timing execution statistics
    jdbc.audit      : Logs ALL JDBC calls except for ResultSets
    jdbc.resultset  : all calls to ResultSet objects are logged
    jdbc.connection : Logs connection open and close events

    로깅 level 5가지 : trace, debug, info, warn, error
-->
<configuration>
    <!-- 스프링 정해 놓은 기본 환경설정 물려받기 -->
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
```

```
<!-- 변수 지정 -->
<property name="LOG_DIR" value="c:/logs/spring-mybatis-2/" />
<property name="LOG_PATH_NAME" value="${LOG_DIR}/project" />

<!--
    콘솔 출력 어펜더 설정
    어펜더를 설정하더래도 출력에 해당되지 않으면 작동하지 않습니다.
 -->
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!--
        By default, encoders are assigned the type
        ch.qos.logback.classic.encoder.PatternLayoutEncoder
     -->
    <encoder>
        <pattern>
    %d{yyyyMMdd HH:mm:ss.SSS} [%thread] %-3level %logger{5} -%msg %n
        </pattern>
    </encoder>
</appender>

<!-- 파일 출력 롤링어펜더 설정 -->
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${LOG_PATH_NAME}</file>
    <prudent>true</prudent>
    <rollingPolicy
  class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${LOG_PATH_NAME}.%d{yyyyMMdd}.%i.log</fileNamePattern>
        <!-- 30일 지난 파일은 삭제한다. -->
        <maxHistory>30</maxHistory>
        <timeBasedFileNamingAndTriggeringPolicy
    class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>100MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>INFO</level>
    </filter>
    <encoder>
        <pattern>
    %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level --- [%thread] %logger{35} : %msg %n
        </pattern>
    </encoder>
</appender>

<!-- SQL 로깅관련 설정 -->
<logger name="jdbc" level="OFF" />
<!-- additivity를 false로 설정하면 해당 name에만 logger가 적용됩니다. -->
<logger name="jdbc.sqlonly" level="DEBUG" additivity="false">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
</logger>
<logger name="jdbc.resultsettable" level="DEBUG" additivity="false">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
</logger>

 <!-- 패키지 정보를 이용하여 부분적으로 적용하는 로깅 환경설정 -->
<logger name="org.springframework.web" level="INFO"/>
<logger name="org.thymeleaf" level="INFO"/>
<logger name="org.hibernate.SQL" level="INFO"/>
<logger name="org.quartz.core" level="INFO"/>
<logger name="org.h2.server.web" level="INFO"/>

<!-- 프로젝트 전역적으로 적용하는 로깅 환경설정 -->
```

```
    <root level="INFO">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="FILE" />
    </root>

</configuration>
```

스프링 개발자가 미리 정의한 설정정보를 상속받아서 적용할 수 있습니다. `<include resource="org/springframework/boot/logging/logback/base.xml"/>` 형태로 설정합니다.

base.xml

defaults.xml

console-appender.xml

## log4jdbc.log4j2.properties

`log4jdbc.spylogdelegator.name` 정보를 제공해야 합니다. resources 폴더 밑에 파일을 배치합니다.

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
# multi-line query display
log4jdbc.dump.sql.maxlinelength=0
```