

Logger

#01. Log란?

... 이전 수업에서 설명 ...

1) Java의 Log 처리 Framework

Slf4j

- Lombok에서 제공하는 인터페이스

logback

- Slf4j에 대한 구현체
- SpringBoot에서 채택한 로그 처리 모듈)

#02. 프로젝트 설정

1. 패키지 정보

`kr.hossam.logger`

| 항목 | 설정 내용 |
|------------|------------------------|
| GroupId | <code>kr.hossam</code> |
| ArtifactId | <code>logger</code> |

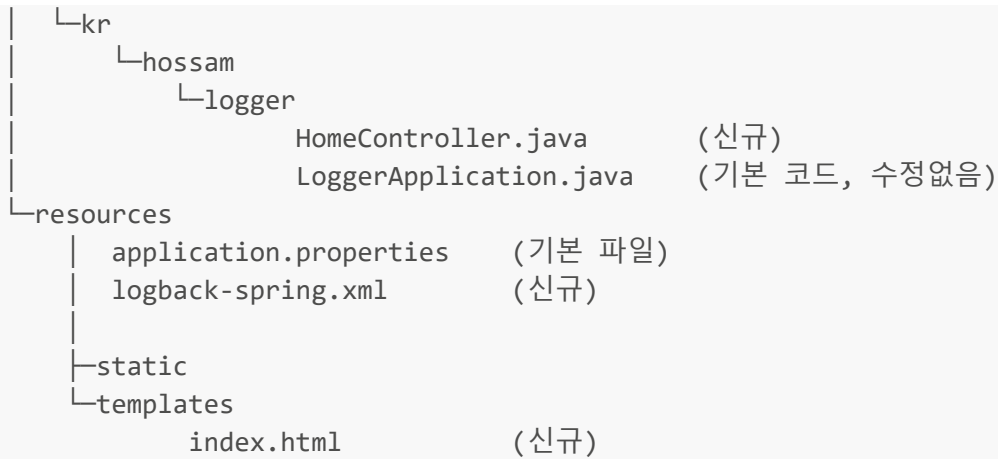
2. 의존성 설정

프로젝트 생성 과정에서 `dependencies` 항목에 대해 아래의 항목을 선택

| 항목 이름 | 구분 |
|----------------------------|-------|
| Spring Boot DevTools | 기존 사용 |
| Spring Boot Actuator Ops | 기존 사용 |
| Spring Web | 기존 사용 |
| Thymeleaf Template Engines | 기존 사용 |
| Lombok | 신규 추가 |

3. 파일 구성

```
C:.\n└─main\n    └─java
```



4. logback 설정 파일 추가

/src/main/resources/logback-spring.xml 파일을 추가

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- 60초마다 설정 파일의 변경을 확인 하여 변경시 갱신 -->
<configuration scan="true" scanPeriod="60 seconds">
    <!-- log file path -->
    <property name="LOG_PATH" value="logs" />
    <!-- log file name -->
    <property name="LOG_FILE_NAME" value="log" />
    <!-- err log file name -->
    <property name="ERR_LOG_FILE_NAME" value="err_log" />
    <!-- pattern -->
    <property name="LOG_PATTERN" value="[%-5level] %d{yy-MM-dd HH:mm:ss}
[%logger{0}:%line] - %msg%n" />

    <!-- Console Appender -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>${LOG_PATTERN}</pattern>
        </encoder>
    </appender>

    <!-- File Appender -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 파일경로 설정 -->
        <file>${LOG_PATH}/${LOG_FILE_NAME}.log</file>

        <!-- 출력패턴 설정-->
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <charset>UTF-8</charset>
            <pattern>${LOG_PATTERN}</pattern>
        </encoder>

        <!-- Rolling 정책 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">

```

```

        <!-- .gz, .zip 등을 넣으면 자동 일자별 로그파일 압축 -->
        <fileNamePattern>${LOG_PATH}/${LOG_FILE_NAME}.%d{yyyy-MM-dd}_%i.log</fileNamePattern>
    <!-- 파일당 최고 용량 kb, mb, gb -->
    <maxFileSize>10MB</maxFileSize>
    <!-- 일자별 로그파일 최대 보관주기(~일), 해당 설정일 이상된 파일은 자동으로 제거-->
    <maxHistory>30</maxHistory>
</rollingPolicy>
</appender>

<!-- 에러의 경우 파일에 로그 처리 -->
<appender name="Error"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <level>error</level>
        <onMatch>ACCEPT</onMatch>
        <onMismatch>DENY</onMismatch>
    </filter>
    <file>${LOG_PATH}/${ERR_LOG_FILE_NAME}.log</file>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <charset>UTF-8</charset>
        <pattern>${LOG_PATTERN}</pattern>
    </encoder>
    <!-- Rolling 정책 -->
    <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
        <!-- .gz, .zip 등을 넣으면 자동 일자별 로그파일 압축 -->
        <fileNamePattern>${LOG_PATH}/${ERR_LOG_FILE_NAME}.%d{yyyy-MM-dd}_%i.log</fileNamePattern>
        <!-- 파일당 최고 용량 kb, mb, gb -->
        <maxFileSize>10MB</maxFileSize>
        <!-- 일자별 로그파일 최대 보관주기(~일), 해당 설정일 이상된 파일은 자동으로 제거-->
        <maxHistory>60</maxHistory>
    </rollingPolicy>
</appender>

<!-- root레벨 설정 -->
<root level="WARN">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</root>

<!-- 특정패키지 로깅레벨 설정 -->
<logger name="org.apache.ibatis" level="DEBUG" additivity="false">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="프로젝트_패키지_이름" level="DEBUG" additivity="false">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />

```

```
<appender-ref ref="Error" />
</logger>
</configuration>
```

#02. Log 사용하기

1. 클래스 어노테이션 추가

@Slf4j 어노테이션을 클래스 선언부에 추가한다.

2. 로그 사용하기

@Slf4j 어노테이션이 명시된 클래스는 log라는 이름의 객체가 자동 생성된다.

```
log.debug("로그내용");
log.info("로그내용");
log.warn("로그내용");
log.error("로그내용");
```

#03. 클라이언트 IP 주소 확인하기

1) VSCode 설정 추가

정확히는 VSCode의 설정이 아니라 VSCode가 자바 컴파일 후 프로그램을 실행할 때 JVM에 전달하는 파라미터임

```
{
  "spring-boot.ls.java.vargs": [
    "-Djava.net.preferIPv4Stack=true"
  ]
}
```

2) IP주소 획득하기

```
/** 접근한 웹 브라우저의 IP */
// 출처: https://velog.io/@chulllll/spring-boot-IPv4-%EC%84%A4%EC%A0%95
String ip = request.getHeader("X-Forwarded-For");
if (ip == null) {
  ip = request.getHeader("Proxy-Client-IP");
  log.info(">>>> Proxy-Client-IP : " + ip);
}
if (ip == null) {
  ip = request.getHeader("WL-Proxy-Client-IP"); // 웹로직
  log.info(">>>> WL-Proxy-Client-IP : " + ip);
}
if (ip == null) {
```

```

        ip = request.getHeader("HTTP_CLIENT_IP");
        log.info(">>>> HTTP_CLIENT_IP : " + ip);
    }
    if (ip == null) {
        ip = request.getHeader("HTTP_X_FORWARDED_FOR");
        log.info(">>>> HTTP_X_FORWARDED_FOR : " + ip);
    }
    if (ip == null) {
        ip = request.getRemoteAddr();
    }
}

```

#04. UserAgent 정보 가져오기

UserAgent란 웹 브라우저가 웹 서버에게 전달하는 자신의 운영체제/웹 브라우저 버전 정보

1) UserAgent 처리 라이브러리 설정

Maven Repository에서 **User Agent Parser For Java**를 검색하여 추가함

Spring 기본 라이브러리가 아니므로 직접 검색이 필요

```

// https://mvnrepository.com/artifact/com.github.ua-parser/uap-java
implementation 'com.github.ua-parser:uap-java:1.6.1'

```

2) 사용 예시

```

/** 접근한 웹 브라우저의 UserAgent값 얻기 */
String ua = request.getHeader("user-agent");
log.debug(ua);

/** 접근한 웹 브라우저의 UserAgent값 얻기 */
String ua = request.getHeader("user-agent");
model.addAttribute("ua", ua);

/** UserAgent값을 파싱하여 브라우저 정보 얻기 */
// --> import ua_parser.Client;
Parser uaParser = new Parser();
// --> import ua_parser.Parser;
Client c = uaParser.parse(ua);

log.debug(c.userAgent.family); // => "Mobile Safari"
log.debug(c.userAgent.major); // => "5"
log.debug(c.userAgent.minor); // => "1"

log.debug(c.os.family); // => "iOS"
log.debug(c.os.major); // => "5"
log.debug(c.os.minor); // => "1"

log.debug(c.device.family); // => "iPhone"

```

그 밖의 클라이언트 접속 정보 획득 방법은 예제코드를 통해 확인하세요.