Spring Boot

소프트웨어융합공학과
웹서버 프로그래밍
허우행

스프링의 다양한 기능

1. Logback 사용하기
2. Log4JDBC 사용하기
3. 인터셉터 사용하기
4. AOP 사용하기
   - 4.1 로그 출력
   - 4.2 트랜잭션 적용
5. 예외처리하기
6. 한글 인코딩

1. Logback 사용하기

- http://logback.qos.ch/
- log4j보다 약 10배 정도 빠르게 수행되도록 내부가 변경되었으며, 메모리 효율성 향상
- 설정 파일을 변경하였을 경우, 서버 재기동 없이 변경 내용이 자동으로 갱신
- RollingFileAppender를 사용할 경우 자동적으로 오래된 로그를 지워주며 Rolling 백업 처리
- 계층적인 5가지의 로그 메시지 레벨을 사용하며 레벨 별 제어 가능
  - ① ERROR : 일반 에러가 일어 났을 때 사용
  - ② WARN : 에러는 아니지만 주의할 필요가 있을 때 사용
  - ③ INFO : 일반 정보를 나타낼 때 사용
  - ④ DEBUG : 일반 정보를 상세히 나타낼 때 사용
  - ⑤ TRACE : 경로추적을 위해 사용

Logback 설정하기

/board/src/main/resources/logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>
<configuration debug="true">
	<appender name="console" class="ch.qos.logback.core.ConsoleAppender">
		<encoder>
			<Pattern>%d %5p [%c] %m%n</Pattern>
		</encoder>
	</appender>
	<appender name="console-infolog" class="ch.qos.logback.core.ConsoleAppender">
		<encoder>
			<Pattern>%d %5p %m%n</Pattern>
		</encoder>
	</appender>

	<logger name="kr.ac.inha.board" level="DEBUG" appender-ref="console" />

	<!-- 프레임웍 로거 -->
	<logger name="org.springframework" level="ERROR"/>
	<logger name="org.springframework.jdbc" level="ERROR"/>

	<root level="off">
		<appender-ref ref="console" />
	</root>
</configuration>
```

Console 창 출력 포맷
http://logback.qos.ch/manual/layouts.html 참고

프로젝트 레벨 설정

프레임워크 레벨 설정

Logback 적용(1)

/board/src/main/java/kr/ac/inha/board/board/controller/BoardController.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Controller
public class BoardController {
	@Autowired
	private BoardService boardService;

	Logger log = LoggerFactory.getLogger(this.getClass());

	@RequestMapping("/board/openBoardList.do")
	public ModelAndView openBoardList() throws Exception {
		ModelAndView mv = new ModelAndView("/board/boardList");

		List<BoardDto> list = boardService.selectBoardList();
		mv.addObject("list", list);

		log.trace("trace level log");
		log.debug("debug level log");
		log.info("info level log");
		log.warn("warn level log");
		log.error("error level log");

		return mv;
	}
}
```

Logback 적용(2) - 롬복사용

/board/src/main/java/kr/ac/inha/board/board/controller/BoardController.java

```
~~import org.slf4j.Logger;~~
~~import org.slf4j.LoggerFactory;~~
import lombok.extern.slf4j.Slf4j;

@Slf4j
@Controller
public class BoardController {
	@Autowired
	private BoardService boardService;

	~~Logger log = LoggerFactory.getLogger(this.getClass());~~

	@RequestMapping("/board/openBoardList.do")
	public ModelAndView openBoardList() throws Exception {
		ModelAndView mv = new ModelAndView("/board/boardList");

		List<BoardDto> list = boardService.selectBoardList();
		mv.addObject("list", list);

		log.trace("trace level log");
		log.debug("debug level log");
		log.info("info level log");
		log.warn("warn level log");
		log.error("error level log");

		return mv;
	}
}
```

Slf4j 추가

@Slf4j 로 대체

삭제

2. Log4JDBC 사용하기

/board/build.gradle 에 dependencies 내용 추가

implementation group: ' org.bgee.log4jdbc-log4j2 ' , name: ' log4jdbc-log4j2-jdbc4.1 ' , version: ' 1.16 '

/board/src/main/resources/log4jdbc.log4j2.properties 추가

```
log4jdbc.spylogdelegator.name = net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.dump.sql.maxlinelength = 0
```

/board/src/main/resources/application.properties 설정 변경

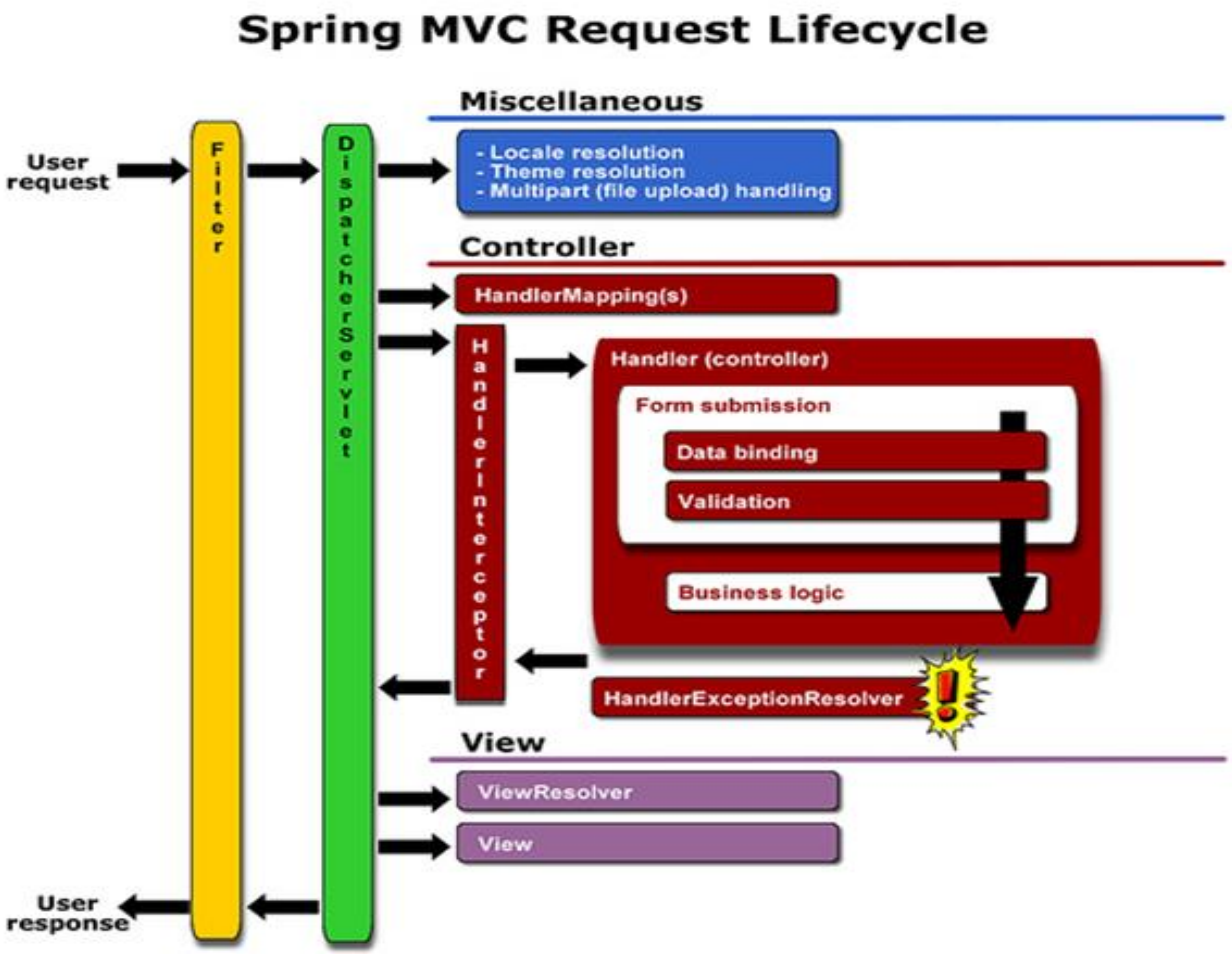
```
#spring.datasource.hikari.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.datasource.hikari.jdbc-url: jdbc:mysql://localhost:3306/board?useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC
➔
spring.datasource.hikari.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
spring.datasource.hikari.jdbc-url: jdbc:log4jdbc:mysql://localhost:3306/board?useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC
```

/board/src/main/resources/logback-spring.xml 내용 추가

```
<logger name="jdbc" level="ERROR" appender-ref="console"/>
<logger name="jdbc.sqlonly" level="ERROR" appender-ref="console"/>
<logger name="jdbc.sqltiming" level="DEBUG" appender-ref="console"/>
<logger name="jdbc.audit" level="ERROR" appender-ref="console"/>
<logger name="jdbc.resultset" level="ERROR" appender-ref="console"/>
<logger name="jdbc.resultsettable" level="DEBUG" appender-ref="console"/>
<logger name="jdbc.connection" level="ERROR" appender-ref="console"/>
```

타입	설명
sqlonly	SQL을 로그로 남기며, Prepared Statement와 관련된 파라미터는 자동으로 변경되어 SQL을 출력합니다.
sqltiming	SQL과 SQL 실행 시간(milliseconds 단위)을 출력합니다.
audit	ResultSet을 제외한 모든 JDBC 호출 정보를 출력합니다. JDBC 관련 문제를 추적하는 경우를 제외하고는 사용이 권장되지 않습니다.
resultset	ResultSet을 포함한 모든 JDBC 호출 정보를 출력합니다.
resultsettable	SQL 조회 결과를 테이블 형태로 출력합니다.
connection	Connection의 연결과 종료에 관련된 로그를 출력합니다. 커넥션 누수 문제 해결에 도움이 됩니다.

3. 인터셉터 사용하기



- 필터는 디스패처 서블릿 앞 단에서 동작
- 인터셉터는 디스패처 서블릿에서 핸들러 컨트롤러로 가기 전에 동작
- 인터셉터에서는 스프링 빈을 사용할 수 있음

- 인터셉터 메서드
  - preHandle : 컨트롤러 실행전에 수행
  - postHandle : 컨트롤러 수행 후 결과를 뷰로 보내기 전에 수행
  - afterCompletion : 뷰의 작업까지 완료 후 수행

인터셉터 적용하기

/board/src/main/java/kr/ac/inha/board/interceptor/LoggerInterceptor.java

```
package kr.ac.inha.board.intercepter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import lombok.extern.slf4j.Slf4j;

@Slf4j
public class LoggerInterceptor extends HandlerInterceptorAdapter {
	@Override
	public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
		log.debug("====================== START ======================");
		log.debug(" Request URI : " + request.getRequestURI());
		return super.preHandle(request, response, handler);
	}

	@Override
	public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
		log.debug("====================== END ======================");
	}
}
```

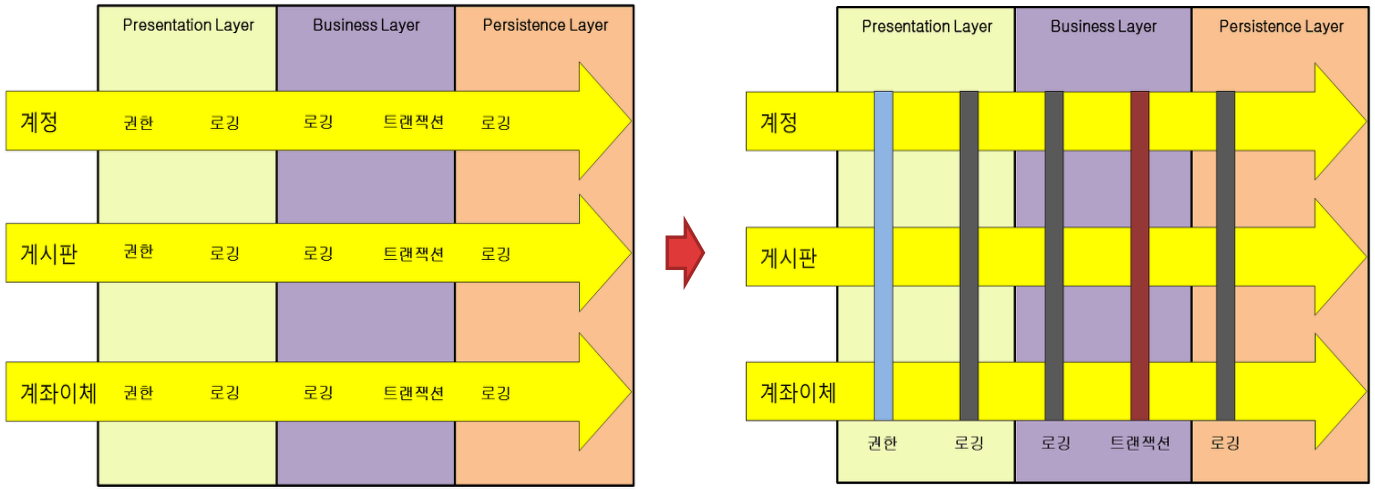
인터셉터 적용하기

/board/src/main/java/kr/ac/inha/board/configuration/WebMvcConfiguration.java

```
package kr.ac.inha.board.configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import kr.ac.inha.board.intercepter.LoggerInterceptor;

@Configuration
public class WebMvcConfiguration implements WebMvcConfigurer{
	@Override
	public void addInterceptors(InterceptorRegistry registry) {
		registry.addInterceptor(new LoggerInterceptor())
		.excludePathPatterns("/**/css/**/")
		.excludePathPatterns("/**/js/**/")
		.excludePathPatterns("/**/scss/**/")
		.excludePathPatterns("/**/vendor/**/");
	}
}
```

4. AOP 사용하기

- 관점 지향 프로그래밍(Aspect Oriented Programming)
- 실행 시점을 지정할 수 있는 애노테이션
  - @Before (이전) : 어드바이스 타겟 메소드가 호출되기 전에 어드바이스 기능을 수행
  - @After (이후) : 타겟 메소드의 결과에 관계없이(즉 성공, 예외 관계없이) 타겟 메소드가 완료 되면 어드바이스 기능을 수행
  - @AfterReturning (정상적 반환 이후)타겟 메소드가 성공적으로 결과값을 반환 후에 어드바이스 기능을 수행
  - @AfterThrowing (예외 발생 이후) : 타겟 메소드가 수행 중 예외를 던지게 되면 어드바이스 기능을 수행
  - @Around (메소드 실행 전후) : 어드바이스가 타겟 메소드를 감싸서 타겟 메소드 호출전과 후에 어드바이스 기능을 수행




```
@Component
@Aspect
public class LoggerAspect {
   @Pointcut("execution(* com.qhedge.online..controller.*Controller.*(..)) || execution(*
com.qhedge.online..service.*Impl.*(..)) || execution(*
com.qhedge.online..mapper.*Mapper.*(..))")
   public void getOnline(){}
   @Around("getOnline")
      public Object logPrint(ProceedingJoinPoint joinPoint) throws Throwable {
         String type = "";
         String name = joinPoint.getSignature().getDeclaringTypeName();
         if (name.indexOf("Controller") > -1) { type = ">> Controller : ";
         } else if (name.indexOf("Service") > -1) { type = ">> ServiceImpl : ";
         } else if (name.indexOf("Mapper") > -1) { type = ">> Mapper : ";
         }

         log.debug(type + name + "." + joinPoint.getSignature().getName() + "()");
         return joinPoint.proceed();
   }
}
```

4.1 로그 출력

/board/src/main/java/kr/ac/inha/board/aop/LoggerAspect.java

```
@Component
@Aspect
@Slf4j
public class LoggerAspect {
	@Around("execution(* kr.ac.inha.board.board..controller.*Controller.*(..)) or "
		+ "execution(* kr.ac.inha.board.board..service.*Service.*(..)) or "
		+ "execution(* kr.ac.inha.board.board..mapper.*Mapper.*(..))")
	public Object logPrint(ProceedingJoinPoint joinPoint) throws Throwable {
		String type = "";
		String name = joinPoint.getSignature().getDeclaringTypeName();

		if (name.indexOf("Controller") > -1) {
		  type = "Controller : ";
		}
		else if (name.indexOf("Service") > -1) {
		  type = "Service : ";
		}
		else if (name.indexOf("Mapper") > -1) {
		  type = "Mapper : ";
		}

		log.debug("Start => " + type + name + "." + joinPoint.getSignature().getName() + "()");

		Object obj = joinPoint.proceed();

		log.debug("End => " + type + name + "." + joinPoint.getSignature().getName() + "()");

		return obj;
	}
}
```

4.2 트랜잭션 적용

/board/src/main/java/kr/ac/inha/board/aop/TransactionAspect.java

```
@Configuration
public class TransactionAspect {
private static final String AOP_TRANSACTION_METHOD_NAME = "*";
private static final String AOP_TRANSACTION_EXPRESSION = "execution(* kr.ac.inha.board.board..service.*Impl.*(..))";

@Autowired
private PlatformTransactionManager transactionManager;

@Bean
public TransactionInterceptor transactionAdvice() {
      MatchAlwaysTransactionAttributeSource source = new MatchAlwaysTransactionAttributeSource();
      RuleBasedTransactionAttribute transactionAttribute = new RuleBasedTransactionAttribute();
      transactionAttribute.setName(AOP_TRANSACTION_METHOD_NAME);
      transactionAttribute.setRollbackRules(Collections.singletonList(new RollbackRuleAttribute(Exception.class)));
      source.setTransactionAttribute(transactionAttribute);

      return new TransactionInterceptor(transactionManager, source);
}

@Bean
public Advisor transactionAdviceAdvisor() {
            AspectJExpressionPointcut pointcut = new AspectJExpressionPointcut();
            pointcut.setExpression(AOP_TRANSACTION_EXPRESSION);
            return new DefaultPointcutAdvisor(pointcut, transactionAdvice());
      }
}
```

5. 예외처리하기

```
@Override
public Map<String, Object> batchSlideSlaveOptList(String id) throws Exception {

try {
 SshUtil sshUtil = new SshUtil(serverInfoDto.getHostIp(), serverInfoDto.getPortNo(), serverInfoDto.getUserId(), serverInfoDto.getUserPass());
 sshUtil.download(serverInfoDto.getRootPath() + "/" + batchSlideSlaveDto.getZipFileName(), tmpPath.toAbsolutePath().toString());
 } catch (IOException ex) {
  log.error("IOError writing file to output stream");
  throw ex;
 } catch (Exception e) {
  log.error(e.getMessage());
  e.printStackTrace();
  throw e;
 }
}
```

try/catch 를 이용한 예외처리

```
@RestController
public class MyRestController {
 ...
 ...
 @ExceptionHandler(NullPointerException.class)
 public Object nullex(Exception e) {
   System.err.println(e.getClass());
   return "myService";
 }
}
```

@ExceptionHandler를 이용한 예외처리

예외처리하기

/board/src/main/java/kr/ac/inha/board/common/ExceptionHandler.java

```
package kr.ac.inha.board.common;

import javax.servlet.http.HttpServletRequest;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.servlet.ModelAndView;

import lombok.extern.slf4j.Slf4j;

@ControllerAdvice
@Slf4j
public class ExceptionHandler {
	@org.springframework.web.bind.annotation.ExceptionHandler(Exception.class)
	public ModelAndView defaultExceptionHandler(HttpServletRequest request, Exception exception) {
		ModelAndView mv = new ModelAndView("/error/error_default");

		mv.addObject("exception", exception);

		log.error("exception", exception);

		return mv;
	}
}
```

@ControllerAdvice를 이용한 전역 예외처리

에러화면(html)

/board/src/main/resources/templates/error/error_default.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>공통 에러 페이지</title>
<link rel="stylesheet" th:href="@{/css/style.css}"></link>
</head>
<body>
<h3>공통 에러 페이지</h3>
<p th:text="${exception}"></p>
<ul th:each="list : ${exception.getStackTrace()}" th:text="${list.toString()}"></ul>
</body>
</html>
```

6. 한글 인코딩

/board/src/main/java/kr/ac/inha/board/configuration/WebMvcConfiguration.java

```
import java.nio.charset.Charset;
import javax.servlet.Filter;

@Bean
public Filter CharacterEncodingFilter() {
	CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();
	characterEncodingFilter.setEncoding("UTF-8");
	characterEncodingFilter.setForceEncoding(true);

	return characterEncodingFilter;
}

@Bean
public HttpMessageConverter<String> responseBodyConverter() {
	return new StringHttpMessageConverter(Charset.forName("UTF-8"));
}
```