이안_ian_
SMU개발자

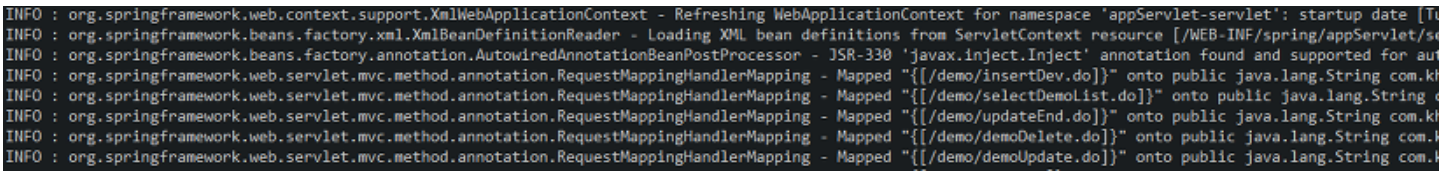
CATEGORY

Back-end/Spring

Spring log4j 이용하여 기록남기기

사용자 이안_ian_ 2019. 2. 19. 17:23

프로그램을 개발하다보면 에러와 값이 제대로 들어오는지 확인하기 위해 System.out.println을 사용하는 개발자들이 많을 것이다. 하지만 이것도 하나의 입출력에 해당하기에 개발이 완료된 시점에서는 남기지 않는게 좋다. 그런데 수많은 서블릿에 사용된 sysout을 일일히 찾아서 지우는 것은 쉽지않은 일이다. 물론 사용이 끝나면 바로 지우는 습관을 들이는 것도 좋은 방법이지만, 고쳤던 문제가 다시 발생해 전에 썼던 sysout구문을 그대로 사용해야한다면 다시 찾아서 써야하지 않은가? 그래서 프로그램의 log 기록을 남기는 방법이 있는데 이것이 바로 log4j를 사용하는 것이다.



사실 스프링의 서버를 켰을 때 console창에 찍히는 INFO도 log4j였다!!

1.콘솔에 로그 찍기

log4j.xml 파일 설정하기

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

    <!-- Appenders -->
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="Target" value="System.out" />
        <layout class="org.apache.log4j.PatternLayout">
            <!--<param name="ConversionPattern" value="%-5p: %c - %m%n" /> -->
                <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p: %l - %m%n"

        </layout>
    </appender>

<!-- Application Loggers -->
    <logger name="com.kh.spring2">
        <level value="info" />
    </logger>

```

이안_ian_
SMU개발자

CATEGORY

```
    </root>

</log4j:configuration>
```

src/main/resources에 가면 있는 log4j.xml의 일부를 발췌한 부분이다. 새로운 로그를 찍고 싶을 경우 5번째 줄에 있는 appender를 설정해줘서 어떤 것의 로그 기록을 찍을 것이고 레이아웃과 패턴들을 지정해주는 것이다. 그리고 root 아래에 appender-ref로 저렇게 설정한 걸 이제 사용하겠다는 의미이며, 16번째 줄은 com.kh.spring2 밑으로 info레벨 이상의 log 기록들을 찍겠다는 거다. info를 기준으로 오른쪽의 레벨들은 출력하겠다는 의미.

* logger level : debug < info < warn < error < fatal

test 해보기

```
1  package com.kh.spring2;
2
3  import org.apache.log4j.Logger;
4
5  import com.kh.spring2.demo.model.vo.Dev;
6
7  public class Log4jTest {
8
9      private Logger log = Logger.getLogger(Log4jTest.class);
10
11     public static void main(String[] args) {
12         new Log4jTest().test();
13     }
14     public void test() {
15         Dev d = new Dev();
16         d.setDevName("홍길동");
17         d.setDevAge(24);
18
19         log.debug("debug임!!"+d);
20         log.info("Info임!!");
21         log.warn("warn임!!");
22         log.error("error임");
23         log.fatal("fatal임!!");
24
25
26     }
27 }
28
```

```
Console  Progress  Problems
<terminated> Log4jTest [Java Application] C:₩Program Files₩Java₩jre1.8.0_181₩bin₩javaw.exe (2019. 2. 19. 오
2019-02-19 16:07:01 INFO : com.kh.spring2.Log4jTest.test(Log4jTest.java:20) - Info임!!
2019-02-19 16:07:01 WARN : com.kh.spring2.Log4jTest.test(Log4jTest.java:21) - warn임!!
2019-02-19 16:07:01 ERROR: com.kh.spring2.Log4jTest.test(Log4jTest.java:22) - error임
2019-02-19 16:07:01 FATAL: com.kh.spring2.Log4jTest.test(Log4jTest.java:23) - fatal임!!
```

먼저 객체를 가져와야 하는데 getLogger의 인자값으로 로그를 찍을 클래스명과 맞춰주고 .class를 해주면된다. 그럼 해당하는 페이지의 값 찍어 볼 수 있다. 그리고 위에 보면 레벨을 info로 설정한 상태로 위와 같은 코드를 실행 했다. debug를 제외하고 출력되는 것을 볼 수가 있다. 또한 패턴값을 앞에 날짜가 출력되게 하여 실행한 날짜와 시간이 나오며 옆옆칸에는 어디서 실행되었는지 알려준다. 패턴값 %l 을 줄 경우 링크를 제공해서 클릭하면 해당부분으로 손쉽게 이동이 가능하다. 이제 level을 debug로 낮추고 결과값을 보자.

level: debug

이안_ian_
SMU개발자

CATEGORY

```
22        <level value="info" />
23    </logger>
24
25    <logger name="org.springframework.beans">
26        <level value="info" />
27    </logger>
28
29    <logger name="org.springframework.context">
30        <level value="info" />
31    </logger>
```

Design Source

Console Progress Problems

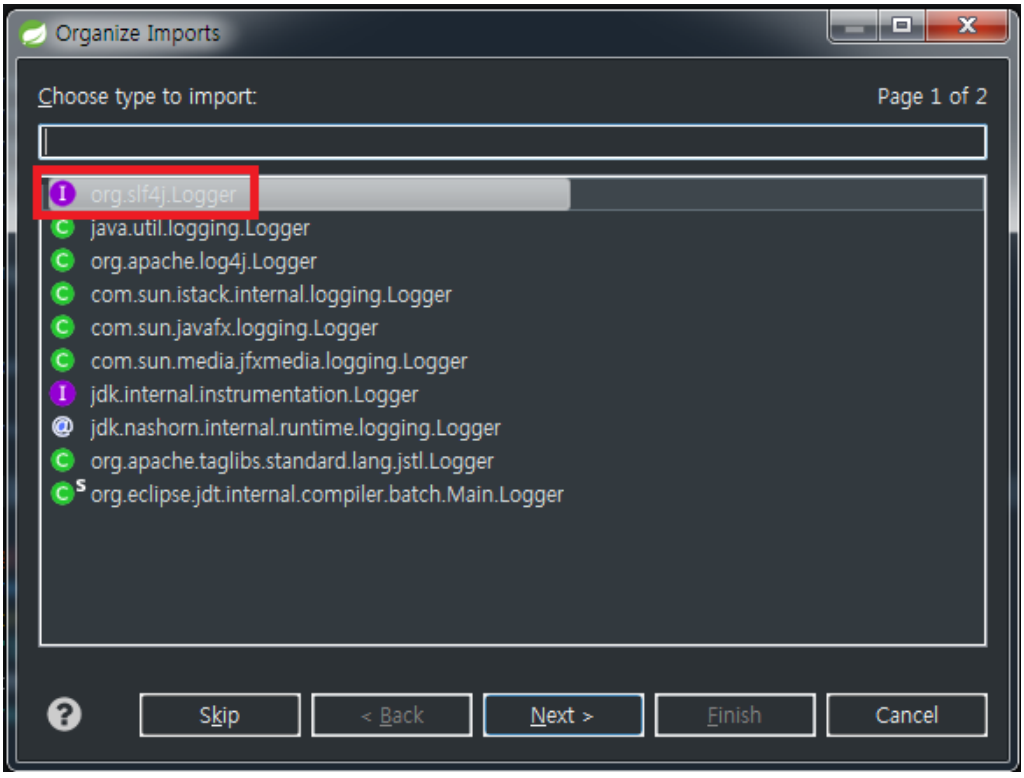
<terminated> Log4jTest [Java Application] C:₩Program Files₩Java₩jre1.8.0_181₩bin₩javaw.exe (2019. 2. 19. 오후 4:32:32)

```
2019-02-19 16:32:32 DEBUG: com.kh.spring2.Log4jTest.test(Log4jTest.java:19) - debug임!!Dev [devNo=0, devName=홍길동, devAge=24, devEmail=null, devGender=null, devLang=null]
2019-02-19 16:32:32 INFO : com.kh.spring2.Log4jTest.test(Log4jTest.java:20) - Info임!!
2019-02-19 16:32:32 WARN : com.kh.spring2.Log4jTest.test(Log4jTest.java:21) - warn임!!
2019-02-19 16:32:32 ERROR: com.kh.spring2.Log4jTest.test(Log4jTest.java:22) - error임
2019-02-19 16:32:32 FATAL: com.kh.spring2.Log4jTest.test(Log4jTest.java:23) - fatal임!!
```

위에 보이는 것 처럼 debug로 찍은 문구와 데이터값이 보인다. 이렇게 레벨을 debug로 해놓고 개발이 끝난 시점에 레벨을 info이상으로 올리면 더이상 콘솔에 찍히지 않기 때문에 일괄처리가 가능하다. 이러다가 다시 문제가 생겨 로그값을 봐야할 때 레벨만 낮추면 썼던 값들을 볼수 있으니 매우 편리한 시스템이다.

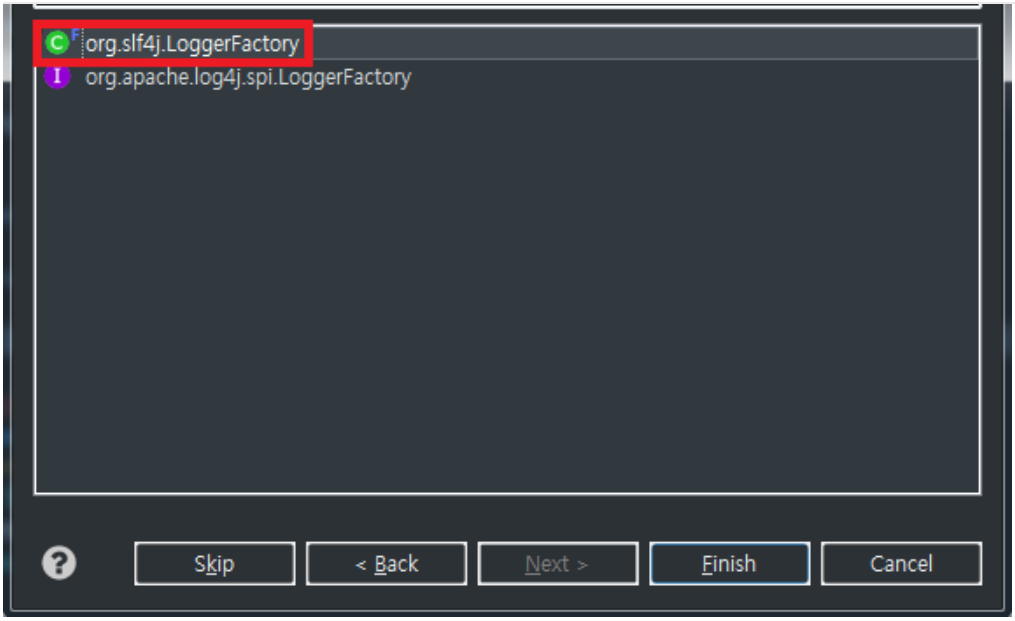
*Import순서

```
@Controller
public class MemoController {
    @Autowired
    MemoService service;

    private Logger log = LoggerFactory.getLogger(MemoController.class);
```



이안_ian_
SMU개발자

CATEGORY



다소 헷갈리는 임포트때문에 첫 화면에서 임포트를 할 때의 순서를 올려 놓겠다.

2.log파일 따로 만들기

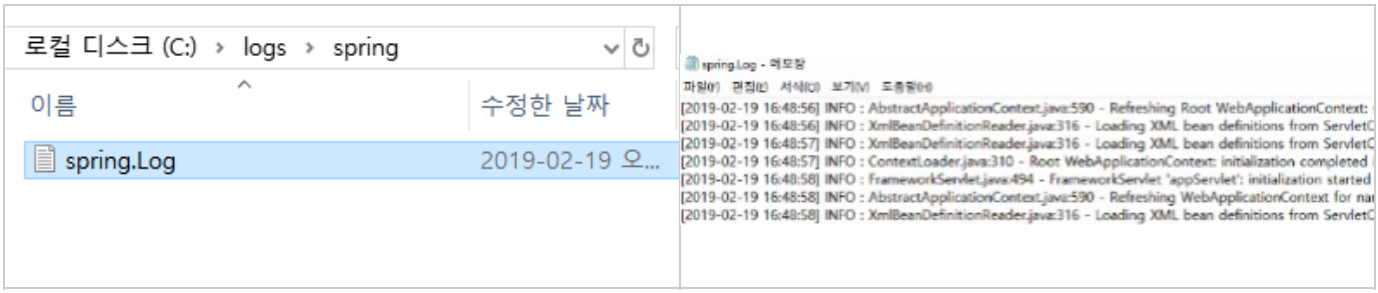
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

    <!-- Appenders -->
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="Target" value="System.out" />
        <layout class="org.apache.log4j.PatternLayout">
            <!--<param name="ConversionPattern" value="%-5p: %c - %m%n" /> -->
                <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p: %l - %m%n" />

        </layout>
    </appender>

    <!-- 파일에 로그 찍기! -->
    <appender name="fileLogger" class="org.apache.log4j.DailyRollingFileAppender">
        <param name="file" value="c://logs//spring//spring.Log"/>
        <param name="Append" value="true"/>
        <param name="dataPattern" value=".yyyy-MM-dd"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="[%d{yyyy-MM-dd HH:mm:ss}] %-5p: %F:%L - %m%n" /
        </layout>
    </appender>

    <!-- Root Logger -->
    <root>
        <priority value="warn" />
        <appender-ref ref="console"/>
        <appender-ref ref="fileLogger"/>
    </root>

</log4j:configuration>
```

Colored by Color Script

로그와 관련된 문건은 따로 파일을 만들어서 관리하는게 보통이다. 그래서 아까 설정했던 appender 밑에 경로를 C드라이브 아래 log/spring 폴더 속에 spring.Log라는 파일에 저장되도록 하는 구문이다. 레이아웃 패턴은 이전과 비슷하게 설정했으

이안_ian_
SMU개발자

CATEGORY



3.SQL의 쿼리문에 대한 구문과 값 보기위한 log

서블릿에서 데이터를 넣거나 가져올 때 Statement계열의 변수를 사용했었다. 근데 이러한 객체를 사용하면 나의 sql구문을 보거나 해당값이 잘 들어 가는지 보기가 어려웠다. 이 부분을 스프링에서 해결이 가능하다. 설정을 해보도록 하자

pom.xml 라이브러리 추가하기

```
        <!-- 디비에 저장할 log4jdbc -remix -->
        <!-- https://mvnrepository.com/artifact/org.lazyluke/log4jdbc-remix -->
        <dependency>
            <groupId>org.lazyluke</groupId>
            <artifactId>log4jdbc-remix</artifactId>
            <version>0.2.7</version>
        </dependency>
```

Colored by Color Scripter cs

다른 log4j와는 달리 jdbc에 관련된 부분은 라이브러리가 필요하니 log4jdbc -remix로 설정을 해주자

root-context.xml 설정하기

```
    <!-- Root Context: defines shared resources visible to all other web components -->
    <bean id="realDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="clos
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="spring"/>
        <property name="password" value="spring"/>
    </bean>
    <!-- 데이터베이스에도 log를 찍기위한 컨피그 -->
    <bean id="dataSource" class="net.sf.log4jdbc.Log4jdbcProxyDataSource">
        <constructor-arg ref="realDataSource"/>
        <property name="logFormatter">
            <bean class="net.sf.log4jdbc.tools.Log4JdbcCustomFormatter">
                <property name="loggingType" value="MULTI_LINE"/>
                <property name="sqlPrefix" value="[SQL]"/>
            </bean>
        </property>
    </bean>
```

Colored by Color Sc

원래 id값을 앞부분만 real로 바꿔주고 그 아래 새로운 bean 등록을 해서 중간에 가로채서 로그값을 찍을 수 있도록 하고 sqlSessionFactory에게 주는 구문이다.

log4j.xml 설정하기

```
    <!-- 디비에 로그 찍기 -->
    <appender name="sqlLogger" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="[%d{yyyy-MM-dd HH:mm:ss}] %-5p:%m%n"/>
        </layout>
```

이안_ian_
SMU개발자

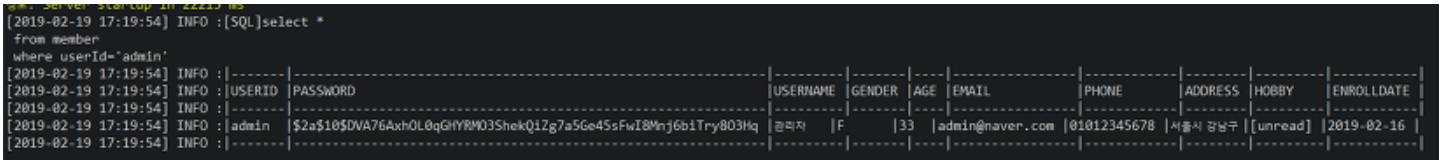
CATEGORY

```
        <appender-ref ref="sqlLogger"/>
    </logger>
    <logger name="jdbc.resultsettable" additivity="false">
        <level value="info"/>
        <appender-ref ref="sqlLogger"/>
    </logger>
```

Colored by Color Scripter cs

똑같이 appender 설정을 해주고 아래에 찍힐 level 설정을 하면서 아래다 바로 appender-ref도 연결해 줬다. 처음 부분은 내가 적은 sql문이 어떤지 보여주는 구문이고 두번째는 그 sql로 리턴되는 값을 찍어주는 것이다. additivity=false 는 이 로그가 다른곳에 영향을 주지 않도록 하겠다(false)는 옵션이다.

결과 창



위에 select * from member where userId='admin' 으로 내가 입력한 데이터까지 완벽하게 보여주고 아래는 해당하는 유저의 데이터를 가져온 것이다. 하지만 이러한 DB의 기록은 매우매우 위험할 수 있으니 개발 땐 유용하게 잘 사용하고 개발 완료 시점에 잊지말고 출력이 안되도록 처리 하길 바랍니다.

'**Back-end** > **Spring**' 카테고리의 다른 글

Spring AOP (0)
Spring log4j 이용하여 기록남기기 (1)
Spring Bcrypt로 패스워드 암호화 하기 (0)
Spring Interceptor (중간에 가로채기) (0)
Spring Session으로 로그인관리, ModelAndVie... (0)
Spring STS TypeHandler, String[] ->String으... (0)

댓글 **1**

이안_ian_
SMU개발자

CATEGORY

여러분의 소중한 댓글을 입력해주세요

이름 비밀번호 비밀글 입력

1 … 95 96 97 98 99 100 101 102 103 … 187

링크

성주
충현이
수빈이형
홍범이

최근에 달린 댓글

도움되서 기쁘네요 ㅎㅎ
정말 감사해요!! 찾고있었는…
그거 아마 자정지나면 되돌려…
감사합니다!!

글 보관함

2020/08 (3)
2020/05 (3)
2020/04 (1)
2020/03 (2)

Total

100,499

Today 8
Yesterday 451

최근에 올라온 글

EC2 인바운드 규칙 생성하기
스프링부트에서 JPA
SpringBoot 기본구조와 단…
flutter API 호출하기

« 2020/08 »

일	월	화	수	목	금	토
						1
2	3	4	5	6	7	**8**
9	10	**11**	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					