스프링(Spring) 개발

(8) 로그 (Log4j) 및 인터셉터 (Interceptor) 설정

1. Log4j 설정

Log4j는 자바기반의 로깅 유틸리티로, Apache에서 만든 오픈소스 라이브러리다.

간단하게 살펴보면

1) 운영시에 불필요한 로그가 계속 출력된다. - 시스템을 개발하고 운영할 때, System.out.println() 을 모두 찾아서 지워준다면 몰라도, 거의 대부분은 그냥 한다.

2) 모든 로그를 지워버리면, 에러가 났을 경우, 그 에러 원인을 찾기가 어려울 수도 있다. - 예를 들어, 시스템에 중대한 에러가 날 경우, 로그를 출력해 놓도록 해 놨는데, 위에서 System.out.println을 모두 지워버렸다면, 로그가 안 남을 수도 있다.

3) 성능에 큰 영향을 미친다. - 사실 가장 중요한 문제다. 우리가 프로그램을 실행하다가 System.out.println() 을 굉장히 많이 호출하면 프로그램의 전체적인 성능이 떨어지는 것을 확인할 수 있다. 예를 들어 1부터 100까지를 모두 더하는 프로그램을 만들었을 때, 로그를 하나도 안 찍으면 정말 0.01초도 안 걸려서 끝나지만, 그 계산과정을 모두 System.out.println()으로 화면에 찍어보면 한참 걸린다. 특히 다중 사용자를 처리해야 하는 웹에서 System.out.println()은 정말 큰 문제를 만들어 버린다.

Log4j는 위의 문제점들을 손쉽게 해결할 수 있다.

Logj4는 시스템의 성능에 큰 영향을 미치지 않으면서도, 옵션 설정을 통해서 다양한 로깅 방법을 제공한다.

환경설정을 통해서 선택적인 로그를 남긴다거나, 특정 파일 등에 로그를 생성하는 등 다양한 이점을 가지고 있다.

Log4j의 구조는 다음과 같다.

요소	설명
Logger	출력할 메시지를 Appender에 전달한다.
Appender	전달된 로그를 어디에 출력할지 결정한다. (콘솔 출력, 파일 기록, DB 저장 등)
Layout	로그를 어떤 형식으로 출력할지 결정한다.

Log4j는 다음과 같은 로그 레벨을 가진다.

로그 레벨	설명
FATAL	아주 심각한 에러가 발생한 상태를 나타낸다.
ERROR	어떠한 요청을 처리하는 중 문제가 발생한 상태를 나타낸다.
WARN	프로그램의 실행에는 문제가 없지만, 향후 시스템 에러의 원인이 될수 있는 경고성 메시지를 나타낸다.
INFO	어떠한 상태변경과 같은 정보성 메시지를 나타낸다.
DEBUG	개발시 디버그 용도로 사용하는 메시지를 나타낸다.
TRACE	디버그 레벨이 너무 광범위한것을 해결하기 위해서 좀 더 상세한 이벤트를 나타낸다.

1. pom.xml에 Log4j를 추가한다.

우리가 생성한 프로젝트에는 기본적으로 Log4j 라이브러리가 추가되어 있고 기본설정도 되어있기 때문에, 넘어 가도록 한다.

2. Log4j 설정

인터넷에서 Log4j에 대해서 찾아보면 여러가지 글이 나오는데, 상당히 많은 글들이 .properties 파일을 이용한 설정방법이다.

Log4j에 .properties를 사용하는 건 최악이다. 절대로 하지말자.

이제 .properties는 굉장히 옛날 방식이고 절대로 사용해서는 안되는 방식이다

최신 Log4j에서는 xml과 json을 이용한 설정만 지원하고 있다.

src/main/resources 폴더 밑에 있는 log4j.xml 파일을 열자.

log4j.xml이 Log4j의 설정파일로, 여기서 로그 출력 형식과 레벨 등을 모두 지정할 수 있다.

콘솔에서 이것저것 출력되었던 것도 여기서 설정된 것을 이용해서 출력이 된것이다.

당시 사용했던 이미지를 다시 가져왔는데, 빨간색 점선으로 표시된 부분이 Log4j를 통해서 출력된 로그들이다. " INFO : " 라고 되어있는 게 위에서 설명한 INFO 레벨의 로그이다.

우리는 앞으로 여러가지 로그를 찍고, 각 로그 종류에 따라서 화면에 보이는게 다르게 하기 위해서 다음과 같이 log4j.xml을 변경한다.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
3  <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
4
5      <!-- Appenders -->
6      <appender name="console" class="org.apache.log4j.ConsoleAppender">
7          <layout class="org.apache.log4j.PatternLayout">
8              <param name="ConversionPattern" value="%d %5p [%c] %m%n" />
9          </layout>
10     </appender>
11
12     <appender name="console-infolog" class="org.apache.log4j.ConsoleAppender">
13         <layout class="org.apache.log4j.PatternLayout">
14             <param name="ConversionPattern" value="%d %5p %m%n" />
15         </layout>
16     </appender>
17
18     <!-- Application Loggers -->
19     <logger name="first" additivity="false">
20         <level value="debug" />
21         <appender-ref ref="console"/>
22     </logger>
23
24     <!-- Query Loggers -->
25     <logger name="jdbc.sqlonly" additivity="false">
26         <level value="INFO"/>
```

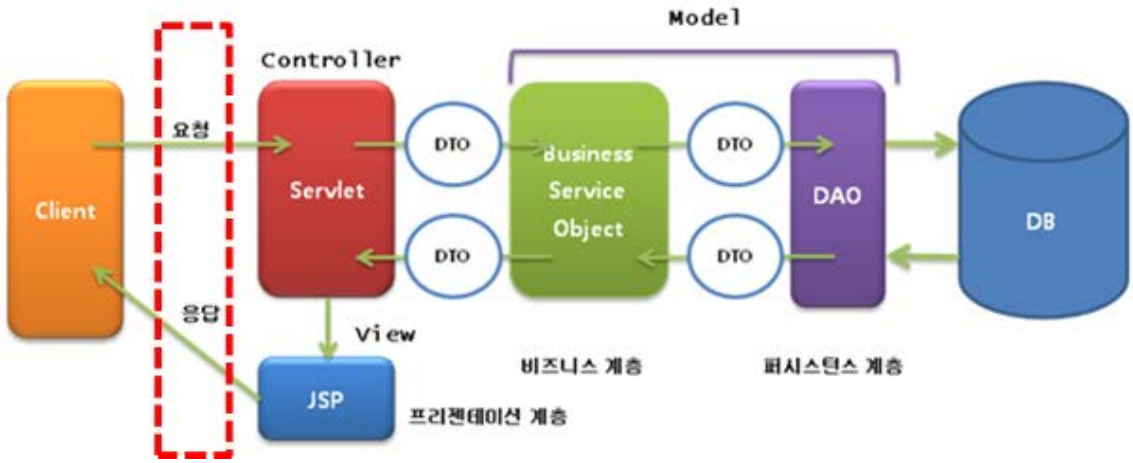
```
27        <appender-ref ref="console-infolog"/>
28    </logger>
29
30    <logger name="jdbc.resultsettable" additivity="false">
31        <level value="INFO"/>
32        <appender-ref ref="console"/>
33    </logger>
34
35    <!-- Root Logger -->
36    <root>
37        <priority value="off"/>
38        <appender-ref ref="console" />
39    </root>
40
41</log4j:configuration>
```

여기서 2개의 appender와 3개의 logger를 볼 수 있다.

일반적인 debug 레벨은 기존의 appender를 이용하고, 그 외 필요한 여러가지 정보는 info 레벨로 약간 다른 형식으로 출력하려고 한다.

2. 인터셉터 (Interceptor) 설정

인터셉터는 중간에 무엇인가를 가로챈다는 의미이다. 스프링에서도 말 그대로 중간에 요청을 가로채서 어떠한 일을 하는것을 의미한다. 서블릿(Servlet)을 사용해 본 사람이라면 필터(Filter)를 들어봤을텐데, 비슷한 의미로 사용된다. 그럼 어느 중간에서 요청을 가로채서 무엇을 하는지를 간단히 살펴보자.



인터셉터는 위 이미지의 빨간색 박스 부분에서 동작한다. 인터셉터의 정확한 명칭은 핸들러 인터셉터 (Handler Interceptor)이다. 인터셉터는 DispatcherServlet이 컨트롤러를 호출하기 전,후에 요청과 응답을 가로채서 가공할 수 있도록 해준다.

예를 들어, 로그인 기능을 구현한다고 했을 때, 어떠한 페이지를 접속하려고 할 때, 로그인된 사용자만 보여주고, 로그인이 되어있지 않다면 메인화면으로 이동시키려고 해 보자. 기존에는 로그인 체크 로직을 만들어서 각 화면마다 일일이 Ctrl + C,V로 만들기도 했다.

스프링에서는 인터셉터를 사용하여 위의 기능을 간단히 만들 수 있다.

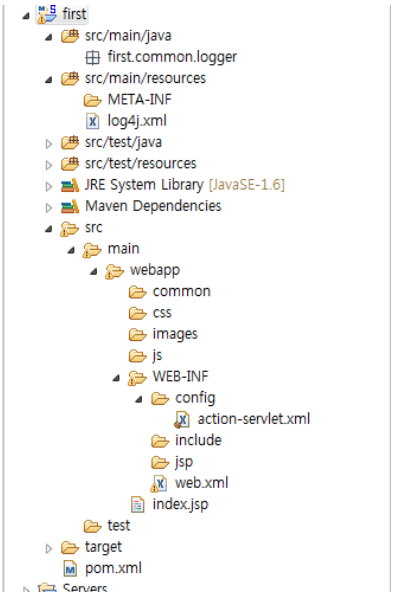
인터셉터에서 어떠한 요청이 들어올 때, 그 사람의 로그인 여부를 판단해서 로그인이 되어 있으면 요청한 페이지로 이동시키고, 로그인이 되어 있지 않을 경우 바로 메인 페이지로 이동시키면 끝이다.

즉, 단 하나의 인터셉터로 프로젝트 내의 모든 요청에서 로그인 여부를 관리할 수 있는 것 이다.

1. 먼저 src/main/java/first 패키지 내에 common 패키지를 생성하고 그 밑에 logger 패키지를 생성한다.

기존에 작성되어 있던 com.company.first 패키지를 삭제하고, first.common.logger 패키지를 생성한다.

마찬가지로 HomeController.java도 삭제한다.



2. logger 패키지 밑에 LoggerInterceptor.java를 생성한다.

이제 위에서 설명한 인터셉터를 만들 차례다. 인터셉터는 HandlerInterceptorAdapter 클래스를 상속받아서 만든다.

HandlerInterceptorAdapter에서는 사용할 수 있는 몇가지 메서드들이 있는데 우리는 일단 두가지만 구현하려고 한다. 전처리기와 후처리기가 바로 그것인데, 위에서 client -> controller 로 요청할 때, 그 요청을 처리할 메서드 하나(전처리기)와 controller -> client 로 응답할 때, 그 요청을 처리할 메서드 하나(후처리기)를 만들 예정이다.

```
   public class LoggerInterceptor extends HandlerInterceptorAdapter {
       protected Log log = LogFactory.getLog(LoggerInterceptor.class);
1
2      @Override
3      public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
4  Object handler) throws Exception {
5          if (log.isDebugEnabled()) {
6              log.debug("======================================           ST
7  ART          ======================================");
8              log.debug(" Request URI ₩t:   " + request.getRequestURI());
9          }
10         return super.preHandle(request, response, handler);
11     }
12
13     @Override
14     public void postHandle(HttpServletRequest request, HttpServletResponse response,
15 Object handler, ModelAndView modelAndView) throws Exception {
16         if (log.isDebugEnabled()) {
17             log.debug("======================================           E
18 ND           ======================================₩n");
19         }
       }
   }
```

이 소스에서 우리가 살펴봐야 할 게 몇 가지 있다.

1) 위에서 우리는 Log4j 를 사용해서 로그를 출력하기로 했었다. 이제 화면에 무엇인가를 출력할 때는 모두 Log4j를 사용하는 데 이는 다음과 같이 사용한다.

protected Log log = LogFactory.getLog(LoggerInterceptor.class); 는 Log4j의 Log 객체를 log라는 이름으로 생성한다.

Log 객체를 생성할 때는 몇가지 방법이 있는데 여기서는 생성자에 현재 클래스를 입력 하였다.

2) 전처리기와 후처리기의 메서드를 등록한다.

preHandler()과 postHandle() 메서드가 전처리기와 후처리기에 해당된다. preHandler()은 컨트롤러가 호출되기 전에 실행되고, postHandle()은 컨트롤러가 실행되고 난 후에 호출된다.

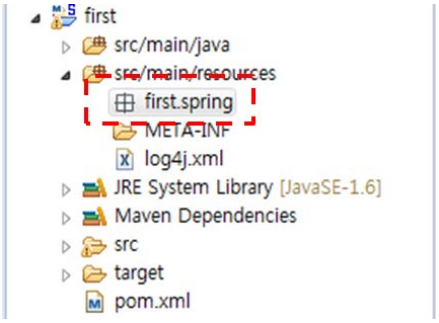
여기서는 단순히 START와 END의 로그를 출력함으로써, 하나의 요청을 쉽게 볼 수 있도록 경계선을 그어주는 역할을 한다.

3) preHandle()에서 현재 호출된 URI가 무엇인지 보여주도록 한다.

3. 방금 만든 인터셉터를 등록한다.

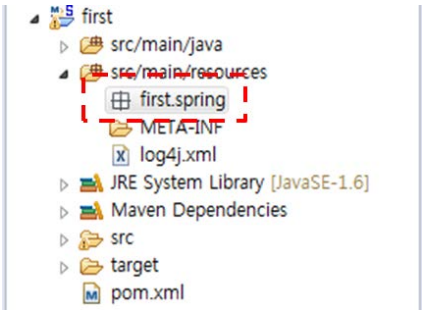
1) src/main/resources 폴더 밑에 first/spring 폴더를 생성한다.

폴더를 생성하면 다음과 같이 패키지로 보이는 경우가 있을수 있다.



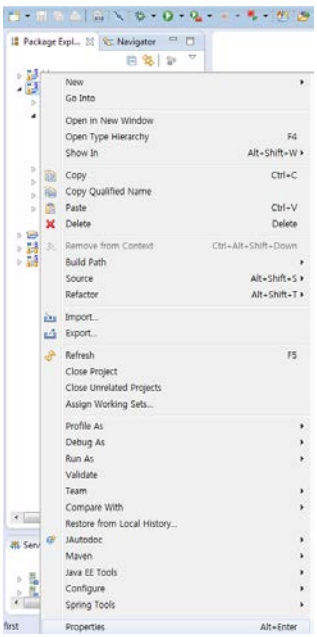
STS로 프로젝트 생성 후, resources 폴더에 새로운 폴더를 생성할 때, 패키지로 보이는 문제 해결 방법

이클립스에서 STS를 설치하고, Spring 프로젝트를 생성하고 나서 src/main/resources 폴더에 새로운 폴더를 추가하면, 당연히 폴더 아이콘으로 보여야 하는데, 패키지로 보이는 경우가 있습니다.

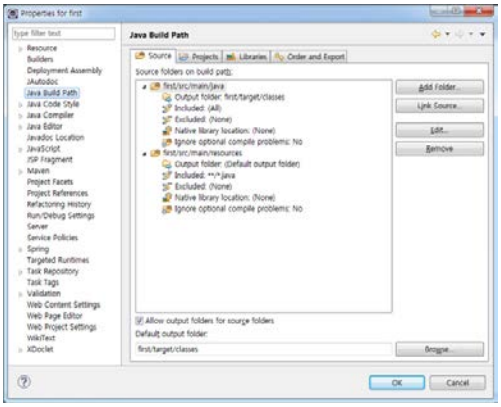


사실 이건 정상적인 상황으로 실제 프로젝트에는 문제가 없지만, 좀 찜찜하고 눈에 거슬리기도 합니다. 이럴때는 다음과 같이 설정하시면 됩니다.

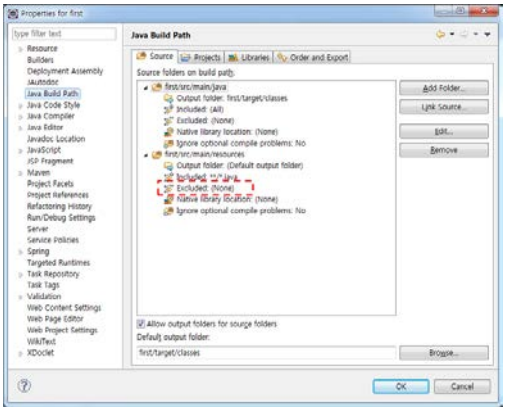
1. 프로젝트 우클릭 -> Properties

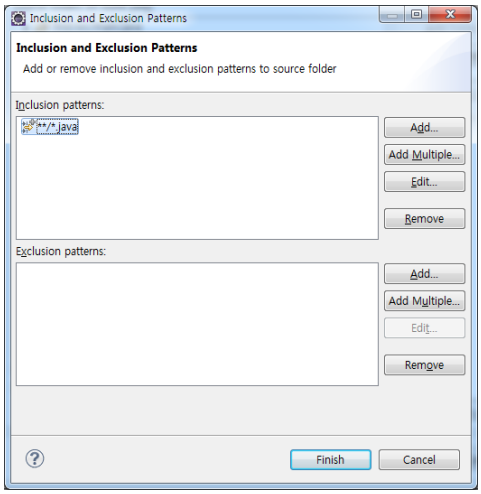


2. Java Build Path -> Source 선택

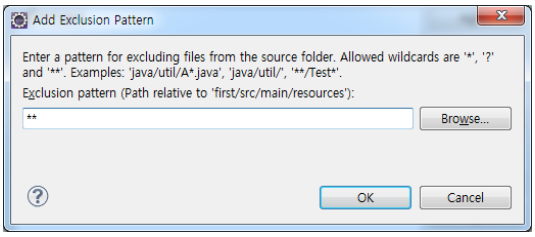


3. 하단에 있는 /src/main/resources의 Excluded를 선택하고 Edit



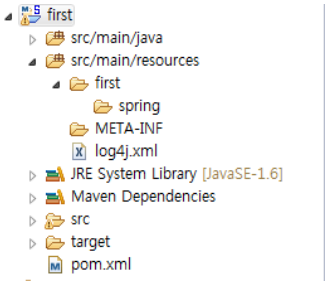


4. Exclusion patterns의 Add를 선택하고 ** 를 입력



5. Finish

그럼 다음과 같이 폴더 형식으로 보입니다.



2) action-servlet.xml을 다음과 같이 수정한다.

화면 하단의 mvc 탭 또는 Overview 탭에서 마우스 오른쪽 클릭으로 요소를 추가하고 값 입력 처리한다.

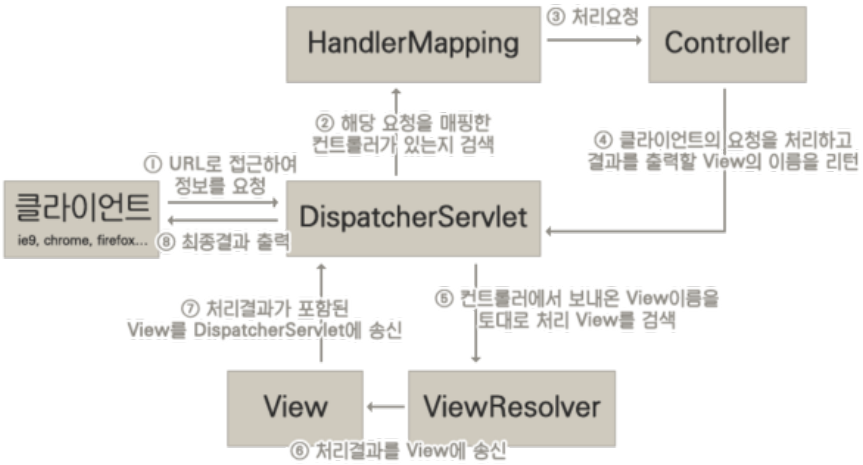
```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xmlns:beans="http://www.springframework.org/schema/beans"
		xmlns:context="http://www.springframework.org/schema/context"
		xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
			http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
			http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

		<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

		<!-- Enables the Spring MVC @Controller programming model -->
		<annotation-driven />

		<context:component-scan base-package="first" />

		<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources
in the ${webappRoot}/resources directory -->
		<resources mapping="/resources/**" location="/resources/" />

		<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-
INF/views directory -->
		<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
			<beans:property name="prefix" value="/WEB-INF/views/" />
			<beans:property name="suffix" value=".jsp" />
		</beans:bean>

		<interceptors>
			<interceptor>
				<mapping path="/**" />
				<beans:bean id="loggerInterceptor"
					class="first.common.logger.LoggerInterceptor">
				</beans:bean>
			</interceptor>
		</interceptors>
</beans:beans>
```

이 소스를 간단히 살펴보자. 스프링 3.2 이상에서는 mvc를 설정하는게 많이 바뀌었다.

<mapping path/>를 통해서 인터셉터가 동작할 URL을 지정할 수 있다. 지금 작성하는 로거는 모든 요청에서 동작을 하기 때문에 전체 패스를 의미하는 "/**" 로 설정하였다.

그 후, bean을 수동으로 등록한다.

중요!!! Interceptor는 Controller가 요청되기 전에 수행된다. 즉, Interceptor는 DispatcherServlet과 같은 위치에 등록이 되어있어야 정상적으로 수행이 된다.



DispatcherServlet은 사용자(클라이언트)의 요청을 받아서 해당 요청에 매핑되는 컨트롤러와 연결한 후, 컨트롤러에서 정의된 view를 사용자의 브라우저에 출력하는 역할을 수행한다.

web.xml을 다시 한번 살펴보자.

```
1       <servlet>
2           <servlet-name>action</servlet-name>
3           <servlet-class>
4               org.springframework.web.servlet.DispatcherServlet
5           </servlet-class>
6           <init-param>
7               <param-name>contextConfigLocation</param-name>
8               <param-value>
9                   /WEB-INF/config/*-servlet.xml
10              </param-value>
11          </init-param>
12          <load-on-startup>1</load-on-startup>
13      </servlet>
14      <servlet-mapping>
15          <servlet-name>action</servlet-name>
16          <url-pattern>*.do</url-pattern>
17      </servlet-mapping>
```

우리는 여기서 이미 DispatcherServlet을 정의하였다. 그리고 그 DispatcherServlet의 설정파일의 위치는 /WEB-INF/config/ 폴더 밑의 -servlet.xml 로 끝나는 모든 xml 파일이라고 명시한 것이다.

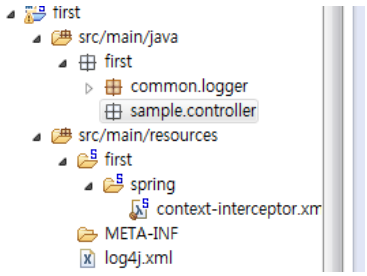
즉, action-servlet.xml에 interceptor를 설정함으로써, 우리는 DispatcherServlet과 Interceptor를 같은 위치에 등록을 한 것이다.

만약, action-servlet에서 Interceptor의 설정을 분리하여 다른 파일로 만들고 싶으면, action-servlet.xml이 있는 폴더에 *-servlet.xml의 이름 형식 (예를 들어 interceptor-servlet.xml)으로 만들면 된다.

4. 이제 인터셉터가 제대로 동작하는지 확인하자.

인터셉터는 컨트롤러의 요청과 응답 시에 호출된다. 현재 우리는 컨트롤러가 아무것도 없으니, 당연히 아무것도 보이지 않는다. 따라서 간단한 컨트롤러를 하나 만들고 테스트를 해보자.

1) src/main/java/first 밑에 sample.controller 패키지를 만든다.



2) controller 패키지에 SampleController 클래스를 생성한다.

여기서 대소문자에 꼭 주의하도록 한다.

3) 다음의 소스를 붙여넣자.

```
1    @Controller
2    public class SampleController {
3            Logger log = Logger.getLogger(this.getClass());
4
5            @RequestMapping(value="/sample/openSampleList.do")
6            public ModelAndView openSampleList(Map<String, Object> commamdMap)
7                    throws Exception{
8                ModelAndView mv = new ModelAndView("sampleView");
9                log.debug("인터셉터 테스트");
10
11               return mv;
12           }
13   }
```

위 소스를 간단히 설명하면,

1번째 줄의 @Controller 는 어노테이션(Annotation)으로, 스프링 프레임워크에 현재 클래스가 컨트롤러 라는것을 명시해준다.

3번째 줄에서 우리가 사용할 Log4j 로그를 선언해놨다.

5번째 줄에서 @Requestmapping 역시 어노테이션으로, DisptacherServlet은 이 어노테이션을 기준으로 어떤 컨트롤러의 메서드가 호출되어야 할지를 결정한다.

그 뒤의 (value="/sample/openSampleList.do")는 프로젝트가 실행될 주소를 의미한다.

7번째 줄에서 ModelAndView 클래스형식의 mv 인스턴스를 생성하였다.

여기서 new ModelAndView("")를 할때 생성자 부분에는 이 컨트롤러가 실행되고 나서 보여줄 view (사용자에게 보여줄 화면)를 설정할 수 있다.

여기서는 인터셉터가 동작하는지 확인하기 테스트이기 때문에 따로 view를 설정하지 않겠다.

8번째 줄의 log.debug("인터셉터 테스트"); 를 통해서 컨트롤러가 실행되고 log4j의 로거도 동작하는것을 보려고 한다.

4) 위에서 선언한 컨트롤러를 호출하자.

우리는 그동안 프로젝트가 실행되면 자동으로 index.jsp가 실행되도록 했었다. index.jsp를 이용해서 프로젝트가 실행되면 바로 방금 만든 /sample/openSampleList.do를 호출하도록 변경하자.

index.jsp를 열고, 다음의 코드를 추가해 넣는다.

```
<a href="/first/sample/openSampleList.do">인터셉터</a>
```

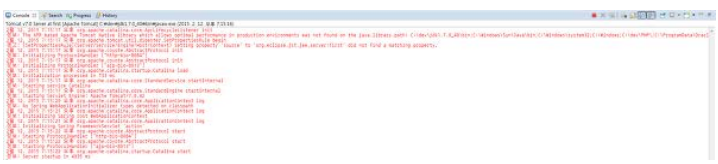
index.jsp가 호출되면 자동으로 /sample/openSampleList.do 의 URL로 연결되도록 하였다.

5) 응답할 뷰 파일을 추가 생성한다.

src/main/sebapp/WEB-INF/views/sampleView.jsp 파일을 새로 만든다. 간단히 내용을 추가한다.

```
<body>
        <h2>샘플 화면</h2>
</body>
```

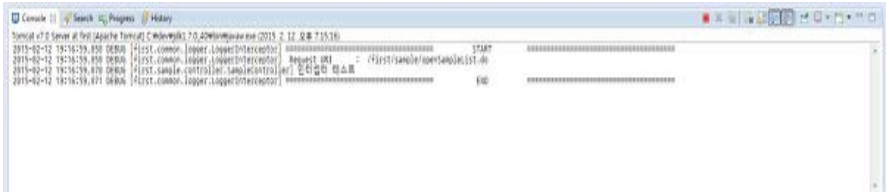
그럼 실행시켜 보자. 이클립스에서 first 서버를 실행시킨다.



그럼 위와 같이 에러가 없이 서버가 정상적으로 실행된다.

그 다음에는 브라우저에서 주소창에 localhost:8080/first를 입력하자.

우리가 정말로 확인해야 하는것은 이클립스의 콘솔창이다.



위와 같은 화면이 보이면 정상적으로 인터셉터와 Log4j가 설정된 것이다.

로그를 살펴보자. 우리는 아까 LoggerInterceptor를 생성해서 전처리기와 후처리기인 preHandle()과 postHandle()을 만들고, 거기에 로그를 작성했다.

하나의 요청의 시작과 끝을 구분짓는다고 했는데 START는 요청의 시작, END는 모든 로직이 완료된 것을 의미한다.

START 밑에는 Request URI : /first/sample/openSampleList.do 라는 로그가 찍혀있다.

이것은 현재 어떤 주소를 호출했는지를 보여준다.

index.jsp에서 /sample/openSampleList.do 라는 주소를 호출하였고, 그 주소가 실행되었음을 알 수 있다.

마지막으로 log.debug("인터셉터 테스트"); 라는 것을 통해서 사용자의 로그를 출력하도록 했는데, 그 로그 역시 정상적으로 출력된 것을 확인할 수 있다.

여기서, 로그를 자세히 살펴보자. 로그를 자세히 살펴보면 약간 다른것을 발견할 수 있다.

DEBUG [first.common.logger.LoggerInterceptor] 와 DEBUG [first.sample.controller.SampleController]의 두종류의 로그가 있는것을 확인해 보자.

우리는 Logger 객체를 생성할 때, 우리는 해당 클래스를 Logger 클래스의 생성자에 변수로 넣어주었다.

이를 이용하여 Log4j가 알아서 어떤 클래스에서 로그가 출력된 것인지를 보여주는 것이다.

만약 우리가 System.out.println()을 사용했으면 어떤 클래스에서 출력된 로그인지를 알기가 쉽지 않은데, Log4j를 사용해서 어디서 출력된 로그인지를 알 수 있도록 하였다.