스프링

사용할만한 객체들

|  |
| --- |
| HttpStatus.***INTERNAL\_SERVER\_ERROR***  HttpHeaders headers = **new** HttpHeaders();  MediaType.***APPLICATION\_JSON\_UTF8***  **WebAttributes.*AUTHENTICATION\_EXCEPTION; //“***SPRING\_SECURITY\_LAST\_EXCEPTION”  ResponseEntity<String> responseentity = **new** ResponseEntity<String>(GsonUtil.*toJsonExpose*(error), headers, HttpStatus.***INTERNAL\_SERVER\_ERROR***);  컨트롤에서 리턴..  FileSystemResource(file); //파일다운로드 |

viewResolver

<http://isstory83.tistory.com/117>

Message 처리

|  |
| --- |
| <bean id=*"messageSourceAccessor"*  class=*"org.springframework.context.support.MessageSourceAccessor"*>  <constructor-arg ref=*"messageSource"* />  </bean>    <bean id=*"messageSource"* class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>  <property name=*"basenames"*>  <list>  <value>classpath:/message/message-common</value>  <value>classpath:/egovframework/rte/fdl/idgnr/messages/idgnr</value>  <value>classpath:/egovframework/rte/fdl/property/messages/properties</value>  </list>  </property>  <property name=*"cacheSeconds"*>  <value>60</value>  </property>  </bean>  <!-- spring messageSource -->  <!-- <bean id="messageSource\_spring" -->  <!-- class="org.springframework.context.support.ResourceBundleMessageSource"> -->  <!-- <property name="basenames"> -->  <!-- <list> -->  <!-- <value>message/message-common</value> -->  <!-- </list> -->  <!-- </property> -->  <!-- <property name="defaultEncoding" value="UTF-8" /> -->  <!-- </bean> --> |
| MessageSourceAccessor message = ApplicationUtil.*getWebApplicationContext*().getBean(MessageSourceAccessor.**class**); String msg = message.getMessage(atError.getCode()); |
| HashMap<String, String> newBody = **new** HashMap<String, String>();  List<ObjectError> errors = error.getAllErrors();  MessageSourceAccessor message = ApplicationUtil.*getWebApplicationContext*().getBean(MessageSourceAccessor.**class**);  **for** (**int** i = 0; i < errors.size(); i++) {  ObjectError atError = errors.get(i);  String msg = atError.getDefaultMessage();  **try**{  msg = message.getMessage(atError.getCode());  }**catch**(Exception e){}    //(null==atError.getCode()?atError.getDefaultMessage() : message.getMessage(atError.getCode()));  String id = atError.getObjectName();  **if** (atError **instanceof** FieldError){  FieldError fieldError = (FieldError)atError;  id = fieldError.getField();  }**else**{  id = (**null**==atError.getCode()?atError.getObjectName():atError.getCode());  }  newBody.put(id, msg);  }  **this**.body = newBody;  **return** **this**; |
| @Autowired  MessageSourceAccessor msgAccessor;  **if**(**null**!=ex.getErrors()){  HashMap<String, String> newBody = **new** HashMap<String, String>();  List<ObjectError> errors = ex.getErrors().getAllErrors();  **for** (**int** i = 0; i < errors.size(); i++) {  ObjectError atError = errors.get(i);  String msg = atError.getDefaultMessage();  **try**{  msg = msgAccessor.getMessage(atError.getCode());  }**catch**(Exception e){}  String id = atError.getObjectName();  **if** (atError **instanceof** FieldError){  FieldError fieldError = (FieldError)atError;  id = fieldError.getField();  }**else**{  id = (**null**==atError.getCode()?atError.getObjectName():atError.getCode());  }  newBody.put(id, msg);  }  error.put("errors",newBody);  } |

MessageSource  
  
locale값이 정해지면 Spring에서해는 해당 locale의 messagesource의 값으로 변환해서 돌려준다.  
그래서 우선 localechangeinterceptor처럼 xml에 등록먼저 해주자.

<bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">

<property name="basename" value="WEB-INF/messages/Messages"/>

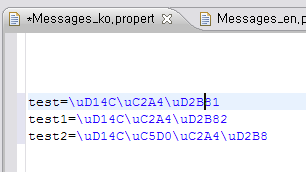
<property name="defaultEncoding" value="UTF-8" />

<property name="cacheSeconds" value="180"/>

</bean>

※ basename : Messages 파일 위치  
  
본인의 설정에서는  
WEB-INF/messages폴더밑에  
**'Messages\_[locale]'** 형태의 파일을 만들었다.

[](http://3.bp.blogspot.com/-pvrnJTNqOj4/UaRrEorDXhI/AAAAAAAAAGw/Qt5i49fzcy4/s1600/message.PNG)

ko와 en두가지 버전이 있다.  
만약 locale값이 ko,en이 아닐 경우  
맨 마지막에 있는 Message.properties를 참조하게 된다.  
[](http://2.bp.blogspot.com/-fzcP9YhAe_M/UaRrqoN-fBI/AAAAAAAAAG4/fM2GO3N6tQE/s1600/message_ko.PNG)  
  
왼쪽은 변수명이고,  
오른쪽은 locale값이 **ko** 일경우에  
돌려주는 값이다.  
Messages 파일에서는 영어가 아닐 경우  
자동으로 유니코드값으로 변환되어 기록된다.  
그러나 출력시에는 저렇게 유니코드값이 아닌 정상적으로 출력된다.  
  
 **3. message 호출**  
위처럼 만들고 써먹으려면 아래 함수를 호출하면 된다.  
code는 2번에서 작성한 좌측의 변수명이고  
locale은 말그대로 locale이다

import org.springframework.context.MessageSource;

......

messageSource.getMessage(code, null, locale)

임의적으로 locale값을 설정하고 싶으면

localeResolver.setLocale(request, response, Locale.KOREA);

처럼 하면 된다.

.

**4. jsp에서 처리 (taglib)**  
localeResolver와 message 프로퍼티 설정이 끝나면  
jsp에서도 바로 다국어 처리가 가능하다.  
  
우선 jsp 상단에

<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>

를 추가하고,  
다국어 처리가 필요한 부분에

<spring:message code="code\_name"/>

를 삽입해주면 된다.

### Spring + locale에 따른 다국어 설정

댓글5

[종벌](http://m.blog.naver.com/PostList.nhn?blogId=mandolle7) [이웃추가](http://m.blog.naver.com/mandolle7/220766980765) | 2016. 7. 20. 14:32

앱으로 보기

이번 글은 Spring 프로젝트에 다국어를 지원할 수 있게 설정하는 방법이다.

회사에서 한국, 영어, 일본, 중국어 페이지 내용이 다 다르다고 4벌을 짜자는 말같지도 않은 소리가 나와서 설정해봄.

웹을 하나만 개발하고 다국어 프로퍼티만 작성해서 프론트에 메세지를 주면 되는건데 말이지...

옛날사람들이 많아서 그런가 쫌 별루임...

암튼 나는 그렇게 하기 싫어 다국어 설정을 해본다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <!-- context-locale.xml -->  <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="<http://www.springframework.org/schema/beans>"      xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd>">        <bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">          <property name="basename" value="classpath:messages/messages"/>          <property name="defaultEncoding" value="UTF-8"/>      </bean>        <bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">          <property name="defaultLocale" value="ko"/>      </bean>    </beans>  [Colored by Color Scripter](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

xml로 빈 설정파일을 작성한다.

messageSource 는 locale에 맞게 해당 국가 언어 메세지 프로퍼티를 가져오기 위해 경로를 지정해준다.

나는 classpath 하위에 messages 폴더에 messages\_en.properties, messages\_ko.properties 등등 국가 메세지 파일들...

기본 인코딩을 utf-8로 지정하였다.

그 다음 localeResolver 는 3가지 방식이 있는데 나는 세션상태에 따른 locale을 가져와서 사용할 것이다.

그 외에 **AcceptHeaderLocaleResolver** 가 있는데 이녀석이 localeResolver의 디폴트 값이다.

localeResolver를 작성하지 않으면 디폴트로 **AcceptHeaderLocaleResolver** 으로 설정된다.

**AcceptHeaderLocaleResolver** 이녀석은 브라우저 설정된 언어값을 사용한다고 하니 사용자의 UI상의 언어변경으로 언어를 바꿔줄 수 없다.

그 다음은 **CookieLocaleResolver** 인데 이녀석은 말그대로 쿠키에 locale을 설정하고 바꿔주는 역할을 한다.

나는 세션을 주로 사용하므로 SessionLocaleResolver를 사용할 것이다.

그리고 디플트 국가언어로 ko 한국어를 지정해놓았다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | <!-- context-servlet.xml -->      <interceptors>          <beans:bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">              <beans:property name="paramName" value="lang"/>          </beans:bean>      </interceptors>  [Colored by Color Scripter](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

이부분은 파라미터로 국가코드를 바꿨을때 인터셉터해서 언어를 바꿔주는 역할을 하는 부분이다.

나는 필요없을 것 같긴한데... 파라미터로 날려서 바꿀 상황이 오지 않을까 해서 남겨놓는다.

예를 들면 이런식으로...

<http://localhost:8000/default/?lang=en>

이제 국가코드에 따른 메세지를 입력해준다.

일단 en과 ko 정도만 하자.

|  |  |  |
| --- | --- | --- |
| 1  2 | # messages\_ko.properties  jb.common.title=스프링프로젝트 | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2 | # messages\_en.properties  jb.common.title=SpringProject | [cs](http://colorscripter.com/info#e) |

이렇게 같은 이름으로 영어와 한글을 작성해주면 된다.

여기서!

아마 이클립스내에 propertiesEditor 가 없는 분들은 한글 메세지파일이 utf-8 모양으로 나올것이다.

|  |  |  |
| --- | --- | --- |
| 1  2 | # messages\_ko.properties  jb.common.title=\uc2a4\ud504\ub9c1\ud504\ub85c\uc81d\ud2b8 | [cs](http://colorscripter.com/info#e) |

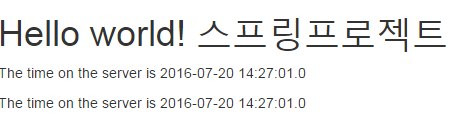
그러면 프로퍼티 에디터를 이클립스에 다운받아서 메세지 프로퍼티파일만 '우클릭 - openWith' 에서 에디터를 바꿔주면 한글을 볼 수 있다.

이제 jsp 페이지에서 자동으로 국가코드에 따라 바뀔 수 있게 셋팅을 해줘야 겠지요~~

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | <%@ taglib prefix="spring" uri="<http://www.springframework.org/tags>" %>  ...      <h1>          Hello world! <spring:message code="jb.common.title"/>      </h1>  ...  [Colored by Color Scripter](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

taglib 라이브러리 설정해주시고

해당 국가별 언어설정할 곳에 메세지를 지정해주시면 됩니다.





다국어 페이지를 여러벌 작성하지 않고도 이렇게 사용할 수 있습니다.

# [[Spring] Locale 처리](http://devbox.tistory.com/entry/Spring-Locale-%EC%B2%98%EB%A6%AC)

http://devbox.tistory.com/entry/Spring-Locale-%EC%B2%98%EB%A6%AC

2014.12.21 02:33

Posted in [Programing/Spring](http://devbox.tistory.com/category/Programing/Spring) by devbox

**Locale 처리**

스프링 제공하는 <spring:message> 커스텀 태그는 웹 요청과 관련된 언어 정보를 이용해서 알맞은 언어의 메시지를 출력한다.

웹 브라우저의 언어 설정을 한국어(ko\_kr)로 했을 때와 영어(en\_us)로 했을 때 <spring:message> 커스텀 태그가 언어에 따라 알맞은 메시지를 출력해 주는 결과 화면을 보여주고 있다.

실제로, 스프링 MVC는 LocaleResolver를 이용해서 웹 요청과 관련된 Locale을 추출하고, 이 Locale 객체를 이용해서 알맞은 언어의 메시지를 선택하게 된다.

본 절에서는 스프링이 제공하는 LocaleResolver를 사용해서 Locale을 변경하는 방법에 대해서 살펴보도록 하겠다.

**1. LocaleResolver 인터페이스**

org,springframework.web.serlvet.LocaleResolver 인터페이스는 다음과 같이 정의 되어 있다.

package org.springframework.web.servlet;

import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public interface LocaleResolver{

Locale resolveLocale(HttpServletRequest request);

void setLocale(HttpServletRequest request, HttpServletResponse response, Locale locale);

}

resolveLocale() 메서드는 요청과 관련된 Locale을 리턴한다. DispatcherServlet은 등록되어 있는 LocaleResolver의 resolveLocale() 메서드를 호출해서 웹 요청을 처리할 때 사용할 Locale을 구한다.

setLocale() 메서드는 Locale을 변경할 때 사용된다. 예를 들어, 쿠키나, 쿠기나 HttpSession에 Locale 정보를 저장할 때에 이 메서드가 사용된다.

**2. LocaleResolver의 종류**

스프링이 기본적으로 제공하는 LocaleResolver 구현 클래스는 다음과 같다.

※ 스프링이 제공하는 LocaleResolver 구현 클래스

|  |  |
| --- | --- |
| 클래스 | 설 명 |
| AcceptHeaderLocaleResolver | 웹 브라우저가 전송한 Accept-Language 헤더로부터 Locale 선택한다. setLocale() 메서드를 지원  하지 않는다. |
| CookieLocaleResolver | 쿠키를 이용해서 Locale 정보를 구한다. setLocale() 메서드는 쿠키에 Locale 정보를 저장한다. |
| SessionLocaleResolver | 세션으로부터 Locale 정보를 구한다. setLocale() 메서드는 세션에 Locale 정보를 저장한다. |
| FixedLocaleResolver | 웹 요청에 상관없이 특정한 Locale로 설정한다. setLocale() 메서드를 지원하지 않는다. |

<bean id="fixedLocaleResolver"

class="org.springframework.web.servlet.i18n.FixedLocaleResolver">

<property name="defaultLocale" value="vi\_VN" />

</bean>

(1) AcceptHeaderLocaleResolver

LocaleResolver를 별도로 설정하지 않을 경우 AcceptHeaderLocaleResolver를 기본 LocalResolver로 사용한다.

AcceptHeaderLocaleResolver는 AcceptLanguage 헤더로부터 Locale 정보를 추출한다.

헤더로부터 Locale 정보를 추출하기 때문에, setLocale() 메서드를 이용해서 Locale 설정을 변경할 수 없다.

(2) CookieLocaleResolver

CookieLocaleResolver는 쿠키를 이용해서 Locale 정보를 저장한다. setLocale() 메서드를 호출하면 Locale 정보를 담은 쿠키를 생성하고, resolveLocale() 메서드는 쿠기로부터 Locale 정보를 가져와 Locale 정보를

담은 쿠키가 존재하지 않을 경우, defaultLocale 프로퍼티의 값을 Locale로 사용한다. defaultLocale 프로퍼티의 값이 null인 경우에는 Accept-Language 헤더로부터 Locale 정보를 추출한다.

CookieLocaleResolver는 쿠티와 관련해서 별도 설정을 필요로 하지 않지만. 생성할 쿠키 이름, 도메인, 경로 등의 설정을 직접하고 싶다면 프로퍼티에 알맞게 설정해주면 된다.

※ CookieLocaleResolver의 쿠키 설정 관련 프로퍼티

|  |  |
| --- | --- |
| 프로퍼티 | 설 명 |
| cookieName | 사용할 쿠키 이름 |
| cookieDomain | 쿠키 도메인 |
| cookiePath | 쿠키 경로, 기본값은 "/"이다. |
| cookieMaxAge | 쿠키 유효 시간 |
| cookieSecure | 보안 쿠키 여부, 기본값은 false 이다. |

(3) SessionLocaleResolver

SessionLocaleResolver HttpSessio에 Locale 정보를 저장한다. setLocale() 메서드를 호출하면 Locale 정보를 세션에 저장하고, resolveLocale() 메서드는 세션으로부터 Locale을 가져와 웹 요청의 Locale을 설정한다.

만약 Locale 정보가 세션에 존재하지 않으면, defaultLocale 프로퍼티의 값을 Locale로 사용한다. defaultLocale 프로퍼티의 값이 null인 경우에는 Accept-Language 헤더로부터 Locale 정보를 추출한다.

(4) FixedLocaleResolver

FixedLocaleResolver는 웹 요청에 상관없이 defaultLocale 프로퍼티로 설정한 값을 웹 요청을 위한 Locale로 사용한다. FixedLocaleResolver는 setLocale() 메서드를 지원하지 않는다.

setLocale() 메서드를 호출할 경우 UnsupportedOperationException 예외를 발생시킨다.

**3. LocaleResolver를 이용한 Locale 변경**

LocaleResolver를 빈으로 등록했다면, 컨트롤러에서 LocaleResolver를 이용해서 Locale을 변경할 수 있게 된다. 예를 들어 다음과 같이 LocaleResolver를 설정했다고 하자.

<bean class="madvirus.spring.chap07.controller.LocaleChangeController">

<property name="localeResolver" ref="localeResolver" />

</bean>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

이 경우, 컨트롤러 클래스는 다음과 같이 LocaleResolver의 setLocale() 메서드를 호출해서 클라이언트의 웹 요청을 위한 Locale을 변경할 수 있다.

import org.springframework.web.servlet.LocaleResolver;

@Controller

public class LocaleChangeController {

private LocaleResolver localeResolver;

@RequestMapping("/changeLanguage")

public String change(@RequestParam("lang") String language, HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

public void setLocaleResolver(LocaleResolver localeResolver) {

this.localeResolver = localeResolver;

}

}

LocaleResolver를 이용해서 Locale을 변경하면, 이후 요청에 대해서는 지정한 Locale을 이용해서 메시지 등을 로딩하게 된다.

ResolverContextUtils 클래스는 웹 요청과 관련된 LocaleResolver를 구할 수 있는 메서드를 제공하고 있으므로, 위 코드를 다음과 같이 변경할 수도 있다.

@Controller

public class LocaleChangeController2 {

@RequestMapping("/changeLanguage2")

public String change(@RequestParam("lang") String language, HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

LocaleResolver localeResolver = RequestContextUtils.getLocaleResolver(request);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

}

**4. LocaleChangeInterceptor를 이용한 Locale 변경**

Locale을 변경하기 위해 별도의 컨트롤러 클래스를 개발한다는 것은 다소 성가신 일이다.

이 경우, 스프링이 제공하는 LocaleChangeInterceptor 클래스를 사용하면 웹 요청 파라미터를 이용해서 손쉽게 Locale을 변경할 수 있다.

LocaleChangeInteceptor 클래스는 handlerInterceptor로서 다음과 같이 HandlerMapping의 intercaptors 프로퍼티에 등록만 하면 설정이 완료된다. 아래 코드는 설정 예이다.

<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"

p:paramName="language" />

<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="interceptors">

<list>

<ref bean="localeChangeInterceptor" />

</list>

</property>

</bean>

paramName 프로퍼티는 Locale을 설정할 때 사용할 파라미터 이름을 명시한다. 예를 들어, 위 코드에서는 paramName 프로퍼티의 값으로 language를 설정했는데,

이 경우 language 요청 파라미터를 사용해서 Locale을 변경할 수 있다.

http://localhost:8080/chap07/jsp/login/login.do?language=en

LocaleChangeInterceptor는 paramName 프로퍼티로 설정한 요청 파라미터가 존재할 경우, 파라미터의 값을 이용해서 Locale을 생성한 뒤 LocaleResolver를 이용해서 Locale을 변경한다.

이후, 요청에서는 변경된 Locale이 적용된다.

Validator

|  |
| --- |
| JSR303 및 여러 개 걸려있는 것.  **public** **class** ValidationUtil {  **public** **static** Errors newErrors(Object object) {  **return** *newErrors*(object,object.getClass().getName());  }  **public** **static** Errors newErrors(Object object, String name) {  BeanPropertyBindingResult errors = **new** BeanPropertyBindingResult(object, name);  **return** errors;  }        /\* 스프링 validator  <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />  @Resource  Validator validator;  \*/  **public** **static** Errors validate(Validator validator,Object object){  **return** *validate*(validator,object,*newErrors*(object, object.getClass().getName()));  }  **public** **static** Errors validate(Validator validator,Object object ,Errors errors){  validator.validate(object, errors);  **return** errors;  }  } |
| Error처리 컨트롤러안쪽에서 Errors로 받아도되고 BindingResult로 받아도된다  Errors는 addError를 못한다.  BindingResult나 Errors는 꼭 ModelAttribute옆에 파라미터로 있어야지 컨트롤단을 우선 탄다  **제가 한참 해매다가 찾았네요. Errors는 모델애트리 바로 옆에 선언되어야 정상 작동한다고 합니다.**  [**http://viralpatel.net/blogs/errorsbindingresult-argument-declared-without-preceding-model-attribute/**](http://viralpatel.net/blogs/errorsbindingresult-argument-declared-without-preceding-model-attribute/)  @RequestMapping(value="/indexTest2", method = {RequestMethod.***GET***, RequestMethod.***POST***},produces = "application/json; charset=utf8")  @ResponseBody  **public** String indexTest2(@Valid @ModelAttribute("FromToVO") FromToVO ft,Errors errors, BindingResult result, HttpServletRequest request, ModelMap model) **throws** Exception { |

Validator

|  |
| --- |
| <bean id=*"messageSource"* class=*"org.springframework.context.support.ReloadableResourceBundleMessageSource"*>  <property name=*"basenames"*>  <list>  <value>classpath:/message/message-common</value>  <value>classpath:/egovframework/rte/fdl/idgnr/messages/idgnr</value>  <value>classpath:/egovframework/rte/fdl/property/messages/properties</value>  </list>  </property>  <property name=*"cacheSeconds"*>  <value>60</value>  </property>  </bean> |
| <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"* %>  <%@ taglib prefix=*"validator"* uri=*"http://www.springmodules.org/tags/commons-validator"* %>  Jsp  <!DOCTYPE html>  <html lang=*"en-us"* class=*"no-js"*>  <head>  <script type=*"text/javascript"*>  <validator:javascript dynamicJavascript="false" staticJavascript="true"/>  </script>  <validator:javascript formName=*"sampleVO"* staticJavascript=*"false"* xhtml=*"true"* cdata=*"false"*/>  <meta charset=*"utf-8"*>  <title>Omnifit</title>  </head> |
| <!-- egov validator -->  <bean id=*"beanValidator"* class=*"org.springmodules.validation.commons.DefaultBeanValidator"*>  <property name=*"validatorFactory"* ref=*"validatorFactory"*/>  </bean>  <bean id=*"validatorFactory"* class=*"org.springmodules.validation.commons.DefaultValidatorFactory"*>  <property name=*"validationConfigLocations"*>  <list>  <value>/WEB-INF/config/validator/validator-rules.xml</value>  <value>/WEB-INF/config/validator/validator.xml</value>  </list>  </property>  </bean> |

message 키값 정책

커스텀 Validator

|  |
| --- |
| <https://docs.spring.io/spring/docs/current/spring-framework-reference/html/validation.html> public class ValidatorDefaultVO implements org.springframework.validation.Validator{  Logger log = LoggerFactory.*getLogger*(this.getClass());  /\*  <!-- jsr303 validator -->  <bean id="validatorJSR303" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"/>  \*/  @Resource(name="validatorJSR303")  Validator validator;  @Override  public boolean supports(Class<?> clazz) {  return DefaultVO.class.isAssignableFrom(clazz);  }   @Override  public void validate(Object target, Errors errors) {  if(DefaultVO.class.isAssignableFrom(target.getClass())){  DefaultVO defaultVO = (DefaultVO)target;  errors = ValidationUtil.*validate*(validator, target);  if(BindingResult.class.isAssignableFrom(errors.getClass())){  defaultVO.validate((BindingResult)errors);  defaultVO.setErrors((BindingResult)errors);  }  }  }   public Validator getValidator() {  return validator;  }   public void setValidator(Validator validator) {  this.validator = validator;  }   } |
| <!-- anotation setting --> <!-- <mvc:annotation-driven /> -->  <mvc:annotation-driven validator="global\_validation"/>  <aop:aspectj-autoproxy/>  <bean id="CMSAOP" class="com.ko.omnicns.omnifit.aop.CMSAOP"></bean>  <bean id="global\_validation" class="com.ko.omnicns.omnifit.validate.ValidatorDefaultVO"></bean> |

1. 에러코드.모델이름.필드이름 : field.required.user.name

2. 에러코드.필드이름 : field.required.name

3. 에러코드.타입이름 : field.required.User

4. 에러코드 :  field.required

에러코드와 모델 오브젝트 이름, 필드 이름 또는 필드 타입의 이름을 결합해서 네 종류의 메시지 키 후보를 만든 것이다. 이 네 가지 메시지 키 후보는 위에서부터 우선적으로 메시지를 찾는데 사용된다. 다음과 같이 우선순위가 높은 field.required.user.name 이라는 메시지 키가 먼저 발견되면 field.required 는 무시된다.

field.required.user.name=이름은 반드시 입력해야 합니다.

field.required=필수 입력 항목입니다.

public class Person {

@NotNull

@Pattern(regexp=".+@.+\\.[a-z]+")

protected String email;

}

Then you could add any of the following messages to your messages.properties file (where "person" and "email" are the name of the class and field in your code):

Code:

Pattern.person.email=Invalid e-mail address

Pattern.email=Invalid e-mail address

Pattern.java.lang.String=Invalid e-mail address

Pattern=Invalid e-mail address

@Pattern(regexp=".+@.+\\.[a-z]+", message="my.message.key") and add in your messages.properties the same key and the translation

public class DomainObject{

...

@Email(message = "{email\_invalid\_error\_message}")

@NotNull(message = "{email\_blank\_error\_essage}")

@Column(unique = true, nullable = false)

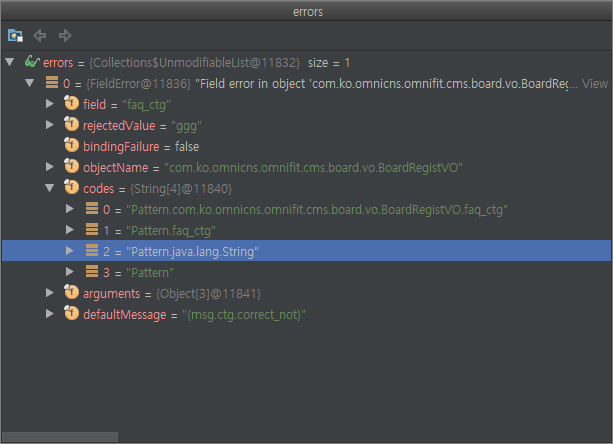
String primaryEmail;

...

}

validation message처리

|  |
| --- |
| <mvc:annotation-driven validator="validator"></mvc:annotation-driven> <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">  <property name="validationMessageSource" ref="messageSource" /> </bean>  <bean id="messageSourceAccessor"  class="org.springframework.context.support.MessageSourceAccessor">  <constructor-arg ref="messageSource" />  </bean>   <bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">  <property name="basenames">  <list>  <value>classpath:/message/message-common</value>  </list>  </property>  <property name="cacheSeconds">  <value>60</value>  </property> </bean> |
| message-common.properties  valid.Pattern.holdFrDt=시작 날짜가 올바르지 않습니다. valid.Pattern.holdToDt=종료 날짜가 올바르지 않습니다. |
| String fltDtlContn; //VARCHAR2(2000) 변동상세내용 + @Column(name="HOLD\_FR\_DT") @Pattern(regexp = "^[1-9][0-9]{3}[0-1][0-2][0-3][0-9]", message = "{valid.Pattern.holdFoDt}") String holdFrDt; //VARCHAR2(8) 보류시작일자 + @Column(name="HOLD\_TO\_DT") @Pattern(regexp = "^[1-9][0-9]{3}[0-1][0-2][0-3][0-9]", message = "{valid.Pattern.holdToDt}") |



Form error

|  |
| --- |
| <form:form commandName=*"FromToVO"*>  <div class=*"container"*>  <div class=*"row"*>  <%-- <form> --%>  <%-- <input type="text" name="from"/>[<form:errors path="from"/>] --%>  <input type=*"text"* name=*"from"*/>[<form:errors path=*"from"*/>]  <input type=*"text"* name=*"to"*/>[<form:errors path=*"to"*/>]  <input type=*"submit"* value=*"submit"*/>  <%-- </form> --%>  dddddddd  </div>  </div> |

Security

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:sec=*"http://www.springframework.org/schema/security"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/security*  *http://www.springframework.org/schema/security/spring-security.xsd"*>  <sec:global-method-security pre-post-annotations=*"enabled"* secured-annotations=*"enabled"*/>  <!-- <sec:global-method-security secured-annotations="enabled"/> -->  <sec:http pattern=*"/css/\*\*"* security=*"none"* />  <sec:http pattern=*"/js/\*\*"* security=*"none"* />  <sec:http pattern=*"/images/\*\*"* security=*"none"* />  <sec:http pattern=*"/resources/\*\*"* security=*"none"* />  <sec:http auto-config=*"true"* use-expressions=*"true"*>  <!-- URL 접근제한을 주기위한 셋팅 DB에서불러와서 처리한다 -->  <sec:custom-filter ref=*"parancoeFilterSecurityInterceptor"* before=*"FILTER\_SECURITY\_INTERCEPTOR"* />  <!--  login-page 로그인 페이지 주소를 지정  username-parameter 로그인 페이지 form에 있는 username을 저장한 변수이름 지정  password-parameter 로그인 페이지 form에 있는 password을 저장한 변수이름 지정  login-processing-url 로그인 페이지 form action에 입력할 주소 지정  default-target-url 로그인 성공인 경우 호출할 주소 지정  authentication-failure-url 로그인 실패인 경우 호출할 주소 지정  always-use-default-target  -->  <sec:form-login login-page=*"/"*  username-parameter=*"username"*  password-parameter=*"password"*  login-processing-url=*"/login"*  default-target-url=*"/"*  authentication-failure-url=*"/"*  />  <sec:logout logout-url=*"/logout"* logout-success-url=*"/"* delete-cookies=*"JSESSIONID"*/>    <!--권한없는 페이지 호출시 web.xml쪽에서 따로 처리해준다-->  <sec:access-denied-handler error-page=*"/common/error/error403.jsp"*/>    <!-- 중복로그인 막아요 session cnt max 1-->  <sec:session-management invalid-session-url=*"/?invalid"*>  <sec:concurrency-control max-sessions=*"1"* expired-url=*"/?expired"* session-registry-ref=*"sessionRegistry"* />  </sec:session-management> </sec:http>      <sec:authentication-manager>  <sec:authentication-provider ref=*"customAuthenticationProvider"*/>  </sec:authentication-manager>      <bean id=*"parancoeFilterSecurityInterceptor"* class=*"org.springframework.security.web.access.intercept.FilterSecurityInterceptor"* >  <property name=*"authenticationManager"* ref=*"org.springframework.security.authenticationManager"*/>  <property name=*"accessDecisionManager"* ref=*"accessDecisionManager"*/>  <property name=*"securityMetadataSource"* ref=*"customFilterInvocationSecurityMetadataSource"*/>  </bean>  <bean id=*"sessionRegistry"* class=*"org.springframework.security.core.session.SessionRegistryImpl"* />  <bean id=*"accessDecisionManager"* class=*"org.springframework.security.access.vote.AffirmativeBased"*>  <constructor-arg>  <list>  <bean class=*"org.springframework.security.access.vote.RoleVoter"*>  <property name=*"rolePrefix"* value=*""* />  </bean>  </list>  </constructor-arg>  <property name=*"allowIfAllAbstainDecisions"* value=*"false"* />  </bean>    <!-- 초기 부트됐을때 메뉴 리스트를 가져온걸로 페이지 호출시 매번 해당페이지가 관리포인트 인지 확인처리 합니다. -->  <bean id=*"customFilterInvocationSecurityMetadataSource"* class=*"com.ko.omnicns.omnifit.boot.security.CustomFilterInvocationSecurityMetadataSource"* />  <!-- 로그인 처리 -->  <bean id=*"customPasswordEncoder"* class=*"com.ko.omnicns.omnifit.login.security.CustomPasswordEncoder"* />  <bean id=*"customAuthenticationProvider"* class=*"com.ko.omnicns.omnifit.login.security.CustomAuthenticationProvider"*/>  </beans> |
| Security web.xml  <!-- 스프링 시큐어리티-->  <filter>  <filter-name>springSecurityFilterChain</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter>  <filter-mapping>  <filter-name>springSecurityFilterChain</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>    <!-- 스프링 세션 리스너 등록 -->  **<listener>**  **<listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>**  **</listener>** |
| Session 처리  <http://antop.tistory.com/151>  @Component  **public** **class** SessionDestoryListener **implements** ApplicationListener<SessionDestroyedEvent> {  Logger log = LoggerFactory.*getLogger*(**this**.getClass());  @Override//session destory될때  **public** **void** onApplicationEvent(SessionDestroyedEvent event) {  List<SecurityContext> lstSecurityContext = event.getSecurityContexts();  //UserDetails ud;  **for** (SecurityContext securityContext : lstSecurityContext){  Object o = securityContext.getAuthentication().getPrincipal();  // ...  }    }  }  @Component  **public** **class** SessionCreateListener **implements** ApplicationListener<SessionCreationEvent> {  Logger log = LoggerFactory.*getLogger*(**this**.getClass());  @Override//Session생성될때  **public** **void** onApplicationEvent(SessionCreationEvent event) {  Object o = event.getSource();    //UserDetails ud;    }  }  @Component  **public** **class** SessionListener **implements** ApplicationListener<ApplicationEvent> {  Logger log = LoggerFactory.*getLogger*(**this**.getClass());  @Autowired  HttpSession httpSession;        @Override  **public** **void** onApplicationEvent(ApplicationEvent applicationEvent) {  **if**(applicationEvent **instanceof** HttpSessionCreatedEvent){ //If event is a session created event  log.debug(" HttpSessionCreatedEvent :" ); //log data  }**else** **if**(applicationEvent **instanceof** HttpSessionDestroyedEvent){ //If event is a session destroy event  // handler.expireCart();  log.debug(""+(Long)httpSession.getAttribute("userId"));  log.debug(" Session is destory :" ); //log data  }**else** **if**(applicationEvent **instanceof** AuthenticationSuccessEvent){ //If event is a session destroy event  log.debug(" athentication is success :" ); //log data  }**else**{  }  }  } |
| <sec:session-management invalid-session-url=*"/?invalid"* session-authentication-error-url=*"/?authentication"* session-fixation-protection=*"newSession"*>  <sec:concurrency-control max-sessions=*"1"* expired-url=*"/?expired"* session-registry-ref=*"sessionRegistry"* />  </sec:session-management>  <bean id=*"sessionRegistry"* class=*"org.springframework.security.core.session.SessionRegistryImpl"* />  **Invalid-session-url : 세션 타임아웃떨어졌을 때**  **expired-url : 동시로그인 됐을대.**  Session Fixation Attack 방지  악의적인 사용자가 사이트에 접근하기 위한 세션을 만들고, 그 세션을 통해 다른 사용자로 로그인 하려고 하는 경우(예를 들어, 세션에 ID를 파라미터로 포함하여 전송하는 경우) Session fixation attack의 잠재적인 위험이 존재하게 된다. 스프링 시큐리티는 이러한 공격을 자동으로 막기 위하여 사용자 로그인 때마다 새로운 세션을 생성한다. 이러한 방지 기능이 필요하지 않거나, 다른 기능들과 충돌이 발생할 경우에는, <session-management>의 session-fixation-protection 속성값으로 동작을 제어할 수 있다. 속성은 다음과 같은 세가지 옵션값들을 가진다.  migrateSession - 새로운 세션을 생성하고 기존의 세션 값들을 새 세션에 복사해준다. 기본값으로 설정되어 있다.  none - 아무것도 수행하지 않는다. 원래의 세션이 유지된다.  newSession - "깨끗한" 새로운 세션을 생성한다. 기존의 세션데이터는 복사하지 않는다.  ex>  <session-management session-fixation-protection="none" />  Sewoo 톰캣 박세우 Park, [18.07.16 16:07]  error-if-maximum-exceeded="true"속성 주면 두번째 인증 아예 못들어가게 하는거고  Sewoo 톰캣 박세우 Park, [18.07.16 16:07]  session-management​의 속성에 session-fixation-protection이란 속성이 있는데  이 부분은 ​Session Fixation( 악성 사용자 가 사이트에 접속하여 세션 ID 를 획득 후 획득 한 세션 ID를 사용하여 다른 사용자가 사이트에 접속하도록 유도하여 다른 사용자가 해당 세션 ID로 사이트 접속시악성 사용자도 같은 세션으로 함께 로그인이 되는 방법​ )을 이용한 악성 사용자를 막기 위해 security에서 매번 인증 요청시마다 session을 재생성 하는데 이를 제어하기 위한 속성이다. |
| <http://docs.spring.io/spring-security/site/docs/current/reference/html/el-access.html>  https://dzone.com/refcardz/expression-based-authorization 26. Expression-Based Access Control Spring Security 3.0 introduced the ability to use Spring EL expressions as an authorization mechanism in addition to the simple use of configuration attributes and access-decision voters which have seen before. Expression-based access control is built on the same architecture but allows complicated boolean logic to be encapsulated in a single expression. 26.1 Overview Spring Security uses Spring EL for expression support and you should look at how that works if you are interested in understanding the topic in more depth. Expressions are evaluated with a "root object" as part of the evaluation context. Spring Security uses specific classes for web and method security as the root object, in order to provide built-in expressions and access to values such as the current principal. 26.1.1 Common Built-In Expressions The base class for expression root objects is SecurityExpressionRoot. This provides some common expressions which are available in both web and method security.  **Table 26.1. Common built-in expressions**   | **Expression** | **Description** | | --- | --- | | hasRole([role]) | Returns true if the current principal has the specified role. By default if the supplied role does not start with 'ROLE\_' it will be added. This can be customized by modifying the defaultRolePrefix on DefaultWebSecurityExpressionHandler. | | hasAnyRole([role1,role2]) | Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings). By default if the supplied role does not start with 'ROLE\_' it will be added. This can be customized by modifying the defaultRolePrefix on DefaultWebSecurityExpressionHandler. | | hasAuthority([authority]) | Returns true if the current principal has the specified authority. | | hasAnyAuthority([authority1,authority2]) | Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings) | | principal | Allows direct access to the principal object representing the current user | | authentication | Allows direct access to the current Authentication object obtained from the SecurityContext | | permitAll | Always evaluates to true | | denyAll | Always evaluates to false | | isAnonymous() | Returns true if the current principal is an anonymous user | | isRememberMe() | Returns true if the current principal is a remember-me user | | isAuthenticated() | Returns true if the user is not anonymous | | isFullyAuthenticated() | Returns true if the user is not an anonymous or a remember-me user | | hasPermission(Object target, Object permission) | Returns true if the user has access to the provided target for the given permission. For example, hasPermission(domainObject, 'read') | | hasPermission(Object targetId, String targetType, Object permission) | Returns true if the user has access to the provided target for the given permission. For example, hasPermission(1, 'com.example.domain.Message', 'read') |  26.2 Web Security Expressions To use expressions to secure individual URLs, you would first need to set the use-expressions attribute in the <http> element to true. Spring Security will then expect the access attributes of the <intercept-url> elements to contain Spring EL expressions. The expressions should evaluate to a boolean, defining whether access should be allowed or not. For example:  사용자 IP 제한  <http>  <intercept-url pattern="/admin\*"  access="hasRole('admin') and hasIpAddress('192.168.1.0/24')"/>  ...  </http>  Here we have defined that the "admin" area of an application (defined by the URL pattern) should only be available to users who have the granted authority "admin" and whose IP address matches a local subnet. We’ve already seen the built-in hasRole expression in the previous section. The expression hasIpAddress is an additional built-in expression which is specific to web security. It is defined by the WebSecurityExpressionRoot class, an instance of which is used as the expression root object when evaluation web-access expressions. This object also directly exposed the HttpServletRequest object under the name request so you can invoke the request directly in an expression. If expressions are being used, a WebExpressionVoter will be added to the AccessDecisionManager which is used by the namespace. So if you aren’t using the namespace and want to use expressions, you will have to add one of these to your configuration. 26.2.1 Referring to Beans in Web Security Expressions If you wish to extend the expressions that are available, you can easily refer to any Spring Bean you expose. For example, assumming you have a Bean with the name of webSecurity that contains the following method signature:  public class WebSecurity {  public boolean check(Authentication authentication, HttpServletRequest request) {  ...  }  }  You could refer to the method using:  <http>  <intercept-url pattern="/user/\*\*"  access="@webSecurity.check(authentication,request)"/>  ...  </http>  or in Java configuration  http  .authorizeRequests()  .antMatchers("/user/\*\*").access("@webSecurity.check(authentication,request)")  ... 26.2.2 Path Variables in Web Security Expressions At times it is nice to be able to refer to path variables within a URL. For example, consider a RESTful application that looks up a user by id from the URL path in the format /user/{userId}.  You can easily refer to the path variable by placing it in the pattern. For example, if you had a Bean with the name of webSecurity that contains the following method signature:  public class WebSecurity {  public boolean checkUserId(Authentication authentication, int id) {  ...  }  }  You could refer to the method using:  <http>  <intercept-url pattern="/user/{userId}/\*\*"  access="@webSecurity.checkUserId(authentication,#userId)"/>  ...  </http>  or in Java configuration  http  .authorizeRequests()  .antMatchers("/user/{userId}/\*\*").access("@webSecurity.checkUserId(authentication,#userId)")  ...  In both configurations URLs that match would pass in the path variable (and convert it) into checkUserId method. For example, if the URL were /user/123/resource, then the id passed in would be 123. 26.3 Method Security Expressions Method security is a bit more complicated than a simple allow or deny rule. Spring Security 3.0 introduced some new annotations in order to allow comprehensive support for the use of expressions. 26.3.1 @Pre and @Post Annotations There are four annotations which support expression attributes to allow pre and post-invocation authorization checks and also to support filtering of submitted collection arguments or return values. They are @PreAuthorize, @PreFilter, @PostAuthorize and @PostFilter. Their use is enabled through the global-method-security namespace element:  <global-method-security pre-post-annotations="enabled"/> Access Control using @PreAuthorize and @PostAuthorize The most obviously useful annotation is @PreAuthorize which decides whether a method can actually be invoked or not. For example (from the"Contacts" sample application)  *@PreAuthorize("hasRole('USER')")*  public void create(Contact contact);  which means that access will only be allowed for users with the role "ROLE\_USER". Obviously the same thing could easily be achieved using a traditional configuration and a simple configuration attribute for the required role. But what about:  *@PreAuthorize("hasPermission(#contact, 'admin')")*  public void deletePermission(Contact contact, Sid recipient, Permission permission);  Here we’re actually using a method argument as part of the expression to decide whether the current user has the "admin"permission for the given contact. The built-in hasPermission() expression is linked into the Spring Security ACL module through the application context, as we’ll[see below](http://docs.spring.io/spring-security/site/docs/current/reference/html/el-access.html#el-permission-evaluator). You can access any of the method arguments by name as expression variables.  There are a number of ways in which Spring Security can resolve the method arguments. Spring Security uses DefaultSecurityParameterNameDiscoverer to discover the parameter names. By default, the following options are tried for a method as a whole.   * If Spring Security’s @P annotation is present on a single argument to the method, the value will be used. This is useful for interfaces compiled with a JDK prior to JDK 8 which do not contain any information about the parameter names. For example: * import org.springframework.security.access.method.P; * ... * *@PreAuthorize("#c.name == authentication.name")*   public void doSomething(*@P("c")* Contact contact);  Behind the scenes this use implemented using AnnotationParameterNameDiscoverer which can be customized to support the value attribute of any specified annotation.   * If Spring Data’s @Param annotation is present on at least one parameter for the method, the value will be used. This is useful for interfaces compiled with a JDK prior to JDK 8 which do not contain any information about the parameter names. For example: * import org.springframework.data.repository.query.Param; * ... * *@PreAuthorize("#n == authentication.name")*   Contact findContactByName(*@Param("n")* String name);  Behind the scenes this use implemented using AnnotationParameterNameDiscoverer which can be customized to support the value attribute of any specified annotation.   * If JDK 8 was used to compile the source with the -parameters argument and Spring 4+ is being used, then the standard JDK reflection API is used to discover the parameter names. This works on both classes and interfaces. * Last, if the code was compiled with the debug symbols, the parameter names will be discovered using the debug symbols. This will not work for interfaces since they do not have debug information about the parameter names. For interfaces, annotations or the JDK 8 approach must be used.   Any Spring-EL functionality is available within the expression, so you can also access properties on the arguments. For example, if you wanted a particular method to only allow access to a user whose username matched that of the contact, you could write  *@PreAuthorize("#contact.name == authentication.name")*  public void doSomething(Contact contact);  Here we are accessing another built-in expression, authentication, which is the Authentication stored in the security context. You can also access its "principal" property directly, using the expression principal. The value will often be a UserDetails instance, so you might use an expression like principal.username or principal.enabled.  Less commonly, you may wish to perform an access-control check after the method has been invoked. This can be achieved using the @PostAuthorize annotation. To access the return value from a method, use the built-in name returnObject in the expression. Filtering using @PreFilter and @PostFilter As you may already be aware, Spring Security supports filtering of collections and arrays and this can now be achieved using expressions. This is most commonly performed on the return value of a method. For example:  *@PreAuthorize("hasRole('USER')")*  *@PostFilter("hasPermission(filterObject, 'read') or hasPermission(filterObject, 'admin')")*  public List<Contact> getAll();  When using the @PostFilter annotation, Spring Security iterates through the returned collection and removes any elements for which the supplied expression is false. The name filterObject refers to the current object in the collection. You can also filter before the method call, using @PreFilter, though this is a less common requirement. The syntax is just the same, but if there is more than one argument which is a collection type then you have to select one by name using the filterTarget property of this annotation.  Note that filtering is obviously not a substitute for tuning your data retrieval queries. If you are filtering large collections and removing many of the entries then this is likely to be inefficient. 26.3.2 Built-In Expressions There are some built-in expressions which are specific to method security, which we have already seen in use above. The filterTarget and returnValue values are simple enough, but the use of the hasPermission() expression warrants a closer look. The PermissionEvaluator interface hasPermission() expressions are delegated to an instance of PermissionEvaluator. It is intended to bridge between the expression system and Spring Security’s ACL system, allowing you to specify authorization constraints on domain objects, based on abstract permissions. It has no explicit dependencies on the ACL module, so you could swap that out for an alternative implementation if required. The interface has two methods:  boolean hasPermission(Authentication authentication, Object targetDomainObject,  Object permission);  boolean hasPermission(Authentication authentication, Serializable targetId,  String targetType, Object permission);  which map directly to the available versions of the expression, with the exception that the first argument (the Authentication object) is not supplied. The first is used in situations where the domain object, to which access is being controlled, is already loaded. Then expression will return true if the current user has the given permission for that object. The second version is used in cases where the object is not loaded, but its identifier is known. An abstract "type" specifier for the domain object is also required, allowing the correct ACL permissions to be loaded. This has traditionally been the Java class of the object, but does not have to be as long as it is consistent with how the permissions are loaded.  To use hasPermission() expressions, you have to explicitly configure a PermissionEvaluator in your application context. This would look something like this:  <security:global-method-security pre-post-annotations="enabled">  <security:expression-handler ref="expressionHandler"/>  </security:global-method-security>  <bean id="expressionHandler" class=  "org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler">  <property name="permissionEvaluator" ref="myPermissionEvaluator"/>  </bean>  Where myPermissionEvaluator is the bean which implements PermissionEvaluator. Usually this will be the implementation from the ACL module which is called AclPermissionEvaluator. See the "Contacts" sample application configuration for more details. Method Security Meta Annotations You can make use of meta annotations for method security to make your code more readable. This is especially convenient if you find that you are repeating the same complex expression throughout your code base. For example, consider the following:  @PreAuthorize("#contact.name == authentication.name")  Instead of repeating this everywhere, we can create a meta annotation that can be used instead.  *@Retention(RetentionPolicy.RUNTIME)*  *@PreAuthorize("#contact.name == authentication.name")*  public *@interface* ContactPermission {}  Meta annotations can be used for any of the Spring Security method security annotations. In order to remain compliant with the specification JSR-250 annotations do not support meta annotations.     <global-method-security secured-annotations="enabled"         jsr250-annotations="enabled" **pre-post-annotations="enabled"** />  시큐리티 설정 파일에 위와 같이 설정하면 @PreAuthorize, @PostAuthorize, @PreFilter, @PostFilter를 사용할 수 있습니다.  이들 애노테이션에서는 스프링 EL을 사용해서 현재 사용자 정보에 접근하거나, (pre 인 경우)메서드의 인자값 또는 (post 인 경우)메서드의 반환값의 정보에 접근할 수 있습니다.      @PreAuthorize("(#study.manager.email == principal.Username) or hasRole('ROLE\_ADMIN')")     public void updateStudy(Study study) {         repository.update(study);     }  위 예제는 다음 주에 오픈 할 봄싹 프로젝트에서 사용하고 있는 코드입니다. Study를 수정하려는 사람이 관리자이거나, 스터디를 만든 사람인지 확인한 뒤에 메서드를 실행합니다. 만약 해당 EL이 false로 판단되면 Access Dinied 에러를 던져줍니다.  애노테이션을 메서드에만 붙이지 않고 클래스에도 붙여서 클래스에 정의한 모든 메서드에 적용할 수도 있습니다. 이런식으로요.  @Service @Transactional @PreAuthorize("hasRole('ROLE\_USER')") public class StudyService {  ...  } |
| [Spring Security에서 로그인 작업 후 부가적인 작업을 설정해보자(로그인 성공시)](http://zgundam.tistory.com/52) http://webcache.googleusercontent.com/search?q=cache:ujW2IphROggJ:zgundam.tistory.com/52+&cd=2&hl=ko&ct=clnk&gl=kr 2014.07.31 18:21  이전 글에서는 로그인 한 사용자의 정보를 화면에 보여주고 로그인 한 사용자의 권한에 따른 동적 메뉴를 구성하는 방법, 그리고 로그아웃에 대해 살펴보았다. 이번에는 이렇게 로그인 기능을 구현한 뒤의 추가 작업을 구성하는 방법에 대해 고민해볼 시간을 갖고록 한다.  로그인 작업이 성공을 하든, 실패를 하든 부가적인 작업이 필요한 상황이 올 수 있다. 예를 들면 로그인을 성공했으면 이를 하루 방문자수에 더한다거나, 로그인 한 사람의 로그인 횟수를 통계 목적을 위해 기록할 수도 있을것이다. 또 로그인에 실패했으면 관련 예외를 다른 방법으로 보여주고 싶을수도 있을 것이다. 즉 로그인 작업이 성공을 하든 실패를 하든 그냥 넘어가는 일은 아마 거의 없을 것이다. 여기서는 그런 작업을 할때 어떤 방법으로 이런 방법을 지정하는지를 알아보도록 하겠다.  로그인 작업이 성공했을때 먼가 부가적인 작업을 하고 싶을 경우 Spring Security에서 제공하는 인터페이스인 **org.springframework.security.web.authentication.AuthenticationSuccessHandler**를 구현한 클래스를 만든뒤에 이 클래스를 Spring Bean으로 등록한다. 그런 후 <form-login> 태그의 authentication-success-handler-ref 속성에 해당 클래스의 id를 넣어주면 된다. 다음은 그런 설정의 예이다.(미리 말하지만 <form-login> 태그의 상위 태그로 <http> 태그가 있다. <http> 태그를 빼고 사용한다는 오해를 불러일으킬수 있을 것 같아 미리 얘기해둔다. <http> 태그 안에 <form-login> 태그를 넣어서 설정한다)  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | <form-login      username-parameter="loginid"      password-parameter="loginpwd"      login-page="/login.do"      default-target-url="/main.do"      authentication-failure-url="/login.do?fail=true"      authentication-success-handler-ref="customAuthenticationSuccessHandler"  />    <beans:bean id="customAuthenticationSuccessHandler" class="com.terry.springsecurity.common.security.handler.CustomAuthenticationSuccessHandler">  </beans:bean> |   form-login 태그의 authentication-success-handler-ref 속성에 CustomAuthenticationSuccessHandler 클래스를 등록한 bean의 id인 customAcuthenticationSuccessHandler를 줌으로써 로그인을 성공했을때 해당 bean을 타도록 지정하는 것이다. 이제 살펴봐야 할 것은 org.springframework.security.web.authentication.AuthenticationSuccessHandler 인터페이스이다. 이 인터페이스에 정의된 메소드는 다음의 메소드 1개 뿐이다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 1 | public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException |   Spring Security는 로그인이 성공한 뒤에 부가적인 작업을 해야 하는 경우 <form-login> 태그의 authentication-success-handler-ref 속성에 설정 된 org.springframework.security.web.authentication.AuthenticationSuccessHandler 인터페이스를 구현한 클래스의 onAuthenticationSuccess 메소드를 실행하는 것으로 로그인 성공 후 작업을 진행하게 되는 것이다. 이 메소드로 넘어오고 있는 것은 **HttpServletRequest 객체, HttpServletResponse 객체, Authentication 객체**가 넘어오고 있는 것을 알 수 있다. 즉 웹으로 넘어온 Request 값을 가져올 수 있고(request.getParameter 메소드), 출력을 정의할 수 있으며(response.getWriter().println 메소드), 인증을 성공했기 때문에 로그인 한 회원의 회원 정보(authentication.getPrincipal())를 가져올 수도 있다. 지금 나열한 것은 일부 예를 들은것이다. HttpServletRequest 객체, HttpServletResponse 객체, Authentication 객체를 이용해서 가져올 수 있는 정보를 이용해 할 수 있는 일은 다 할 수 있다고 보면 된다. 위에서 언급했던 회원별 방문수 증가 작업을 한다면 Authentication 객체를 이용해 회원 로그인 아이디를 가져온뒤 그걸 이용해 DB에 작업하면 되는 것이다. Spring Security 는 org.springframework.security.web.authentication.AuthenticationSuccessHandler 인터페이스를 구현한 클래스 2가지를 제공한 다. org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandler 클래스 와 org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler 클래스이다. 그 중 SavedRequestAwareAuthenticationSuccessHandler 클래스 소스는 한번 봐두길 바란다. 다음부터 설명할 내용도 이 클래스를 기반으로 커스터마이징 한 클래스이기도 하기 때문이다.  그럼 이제부터 우리가 만들 이 CustomAuthenticationSuccessHandler 클래스에서 무엇을 할 것인가? 여기서는 **우리가 로그인 할 때 로그인 후 이동할 URL을 지정하면 해당 URL로 이동하는 기능**을 넣을려고 한다. 이 기능을 하는데 있어 몇가지 우선순위를 잡을 것이다. 다음이 그 우선순위이다.  ● 지정된 Request Parameter(loginRedirect)에 로그인 작업을 마친 뒤 redirect 할 URL을 지정했다면 이 URL로 redirect 하도록 한다.  ● 만약 지정된 Request Parameter에 지정된 URL이 없다면 세션에 저장된 URL로 redirect 하도록 한다.  ● 세션에 저장된 URL도 없다면 Request의 REFERER 헤더값을 읽어서 로그인 페이지를 방문하기 전 페이지의 URL을 읽어서 거기로 이동하도록 한다.(REFERER 기능 사용 여부는 설정 가능하도록 한다. 이 기능 설정을 해야 하는 이유는 밑에서 별도로 설명하도록 하겠다)  ● 위의 3가지 경우 모두 만족하는게 없으면 CustomAuthenticationSuccessHandler 클래스에 있는 defaultUrl 속성에 지정된 URL로  이동하도록 한다.  이런 우선 순위를 생각해두고 이 기능을 구현해보도록 하자. 먼저 첫번째 우선 순위 작업을 위해 작업을 해야 할 부분은 어렵진 않다. onAuthenticationSuccess 메소드에서 HttpServletRequest 객체를 가져오고 있기 때문에 URL이 들어가 있을 파라미터 이름으로 getParameter(파라미터 이름) 메소드를 실행시키면 되기 때문이다.  첫번째 우선 순위 작업을 만족하지 못했을 경우 진행할 두번째 우선 순위, 세션에 있는 URL로의 Redirect를 설명하겠다. Spring Security는 인증을 하지 않은 상태에서 권한이 필요한 화면을 접근할려고 할 경우 로그인 화면을 띄운다고 이전 글에서 언급한 적이 있다. 이때 로그인 화면을 띄우기 전에 필요한 정보를 세션에 저장하게 되는데 이 정보 중에 저장되는 것으로 Spring Security가 띄우는 로그인 화면을 보기 이전의 화면 URL도 저장하고 있다. 바로 이 URL을 꺼낸다는 것이다. 이 부분에 대해 설명할려면 먼저 Spring Security가 제공하는 RequestCache 인터페이스에 대한 이해가 필요하다.    **org.springframework.security.web.savedrequest.RequestCache 인터페이스**는 로그인 화면을 보여주기 전에 사용자 요청을 저장하고 이를 꺼내오는 메카니즘을 정의하는 인터페이스이다. **사용자의 요청은 org.springframework.security.web.savedrequest.SavedRequest 인터페이스를 구현한 클래스 단위로 저장된다.** 즉 사용자가 요청했던 request 파라미터 값들, 그 당시의 헤더값들 등이 SavedRequest 인터페이스를 구현한 클래스에 담겨지게 되는 것이다. Spring Security는 SavedRequest 인터페이스를 구현한 클래스인 **org.springframework.security.web.savedrequest.DefaultSavedRequest 클래스**를 제공하는데  RequestCache 인터페이스를 구현한 클래스는 이 DefaultSavedRequest 클래스 객체를 저장하게 되는것이다. Spring Security는 RequestCache 인터페이스를 구현한 클래스로 **org.springframework.security.web.savedrequest.HttpSessionRequestCache 클래스**를 제공하는데 앞에서 **사용자 요청을 세션에 저장한다 함은 바로 이 HttpSessionRequestCache 클래스를 이용해서 사용자 요청 정보들이 들어있는 DefaultSavedRequest 클래스 객체를 세션에 저장한다는 뜻**이다.  그럼 우리가 만드는 CustomAuthenticationSuccessHandler 클래스에서 세션에 저장되어 있는 사용자가 로그인 화면을 보기 전에 방문했던 URL은 어떻게 가져올까? **HttpSessionRequestCache 클래스 객체를 생성해서 거기서 DefaultSavedRequest 클래스 객체를 가져온뒤 거기서 getRedirectURL 메소드를 호출하면 된다.** 설명은 클래스로 얘기했지만 이 클래스들이 인터페이스를 구현한것이기 때문에 실제 코드는 인터페이스로 코딩할 것이다.  이렇게 두번째 우선 순위에서도 URL을 가져오지 못할 수 있다. 즉 **세션에 DefaultSavedRequest 클래스 객체로 저장되지 않았을 경우**이다. 그러면 그런 경우는 어떤 경우가 있을 수 있을까? **Spring Security가 시스템적으로(자동으로) 로그인 화면을 띄워서 보여주는 것이 아니라 사용자가 직접 로그인 URL로 이동한 경우가 그것이다.** Spring Security가 시스템적으로 로그인 화면을 띄울때는 DefaultSavedRequest 객체를 만들어서 로그인 화면을 보기 전의 화면에 대한 URL과 헤더 정보들을 저장해놓지만, 사용자가 링크를 타고 로그인 화면으로 이동했거나 직접 URL을 입력하여 로그인 화면을 이동했을 경우엔 이런 DefaultSavedRequest 객체를 저장하지를 않기 때문에 세션에 저장되어 있지도 않고 그렇기 땜에 당연 URL도 가져오지 못하는 것이다. 그런 경우를 대비해서 HttpServletRequest 객체의 getHeader 메소드를 이용해 REFERER 헤더 값을 읽어올 수 있다. 위에서 이 기능에 대한 얘기를 했을때 이 기능은 사용 여부를 설정 할 수 있게끔 하도록 한다고 했다. 이 REFERER 헤더를 이용하는 방법은 사실상 비추이기 때문이다.  REFERER 헤더값을 이용하는 방법은 이전 페이지 URL을 얻어오는데 있어 가장 보편적으로 쓰이는 방법이긴 하다. 그러나 로그인의 경우는 이 방법이 비추일수 밖에 없는 상황이 존재한다. 다음의 상황을 생각해 보자. 설명을 위해 다음의 화면 전개를 보자.  메인 화면 -> 로그인 화면 링크 클릭 -> 로그인 화면 -> 아이디와 비밀번호 입력한 뒤 로그인 버튼 클릭 -> 인증 처리 -> 메인 화면  위의 글박스에 언급한 내용은 메인 화면을 보고 있는 상태에서 로그인 화면 보여주는 링크를 클릭해서 로그인 화면을 보는 상황에서 아이디와 비밀번호를 입력한뒤 로그인 과정을 거쳐 메인 화면을 보는 것을 절차로 보여주는 것이다. 위에서 인증 처리라고 얘기했지만 이 부분도 사실 URL이 엄연히 존재한다. 우리가 로그인 화면에서 <form> 태그 작성시 action을 지정하지 않았는가? **j\_spring\_security\_check**라고 엄연히 action을 주었다. 그러면 여기서 로그인 과정을 처리할텐데 이 j\_spring\_security\_check 입장에서 REFERER 헤더값을 읽는다면 어디를 가리키겠는가? 바로 로그인 화면 URL을 가리키게 되는 것이다. 그래서 REFERER로 이동하게끔 지정하면 로그인을 성공해도 다시 로그인 화면을 보여주기 때문에 REFERER 헤더값을 이용하는 방법은 사실상 의미가 없게 된다. 그래서 이 기능에 대해 사용 여부를 설정할 수 있게 한 것이다. 그러나 여기서 굳이 REFERER를 구현한 이유는 Spring Security에서 Redirect URL 선정 기준을 REFERER 헤더값을 읽어서 가도록 하는 방법이 존재하기에 구현은 해보았다.추측엔 웹서비스와 같이 로그인 아이디와 패스워드를 같이 주는 방법에서는 REFERER 헤더값을 해도 지장이 없을것 같아 그렇지 않을까 싶다.  그리고 위에서 언급한 3가지 방법으로도 URL을 구할 수 없으면 지정된 화면으로 이동해야 할 것이다. 그래서 defaultURL 속성을 두어 지정된 화면을 설정하도록 했다. 이런 우선순위를 생각하고 이제 코드를 보도록 하자. 먼저 위에서 언급했던 우선순위 중 어느 것을 사용해야 할지를 결정하는 메소드이다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | /\*\*   \* 인증 성공후 어떤 URL로 redirect 할지를 결정한다   \* 판단 기준은 targetUrlParameter 값을 읽은 URL이 존재할 경우 그것을 1순위   \* 1순위 URL이 없을 경우 Spring Security가 세션에 저장한 URL을 2순위   \* 2순위 URL이 없을 경우 Request의 REFERER를 사용하고 그 REFERER URL이 존재할 경우 그 URL을 3순위   \* 3순위 URL이 없을 경우 Default URL을 4순위로 한다   \* @param request   \* @param response   \* @return   1 : targetUrlParameter 값을 읽은 URL   \*            2 : Session에 저장되어 있는 URL   \*            3 : referer 헤더에 있는 url   \*            0 : default url   \*/  private int decideRedirectStrategy(HttpServletRequest request, HttpServletResponse response){      int result = 0;        SavedRequest savedRequest = requestCache.getRequest(request, response);        if(!"".equals(targetUrlParameter)){          String targetUrl = request.getParameter(targetUrlParameter);          if(StringUtils.hasText(targetUrl)){              result = 1;          }else{              if(savedRequest != null){                  result = 2;              }else{                  String refererUrl = request.getHeader("REFERER");                  if(useReferer && StringUtils.hasText(refererUrl)){                      result = 3;                  }else{                      result = 0;                  }              }          }            return result;      }        if(savedRequest != null){          result = 2;          return result;      }        String refererUrl = request.getHeader("REFERER");      if(useReferer && StringUtils.hasText(refererUrl)){          result = 3;      }else{          result = 0;      }        return result;  } |   코드를 보면 먼저 Spring Security가 세션에 저장하고 있는 SavedRequest 객체를 가져오도록 한다. 그리고 이동해야 할 URL이 지정될 파라미터 이름이 정해져 있는 상태면 그 파라미터 이름으로 request 객체에서 값을 읽어서 URL이 존재하는지 확인하여 그 URL이 존재하면 1을 return 한다(이것이 1순위). 그러나 이 값이 없을 경우 앞에서 가져온 SavedRequest 객체가 null인지 확인한 뒤 null이 아니면 SavedRequest 객체가 있다는 의미이기 때문에 2를 return 한다(이것이 2순위) 그러나 SavedRequest 객체가 null이라면 REFERER 헤더값을 사용하도록 설정했는지 그리고 사용하는 것으로 설정했으면 REFERER 헤더값을 읽어서 URL이 존재하면 3을 return 한다(이것이 3순위) 그리고 지금까지의 모든 조건을 다 만족하지 않으면 0을 return 하도록 한다(이것이 4순위). 4순위일때 0을 return 하는것 빼고는 순위값을 그대로 return 하도록 했다. 그래서 이 return 된 순위값에 따라 적절한 메소드를 호출하여 URL을 얻어오면 되는데 그것이 다음의 코드이다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();    @Override  public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {      // TODO Auto-generated method stub            clearAuthenticationAttributes(request);        int intRedirectStrategy = decideRedirectStrategy(request, response);      switch(intRedirectStrategy){      case 1:          useTargetUrl(request, response);          break;      case 2:          useSessionUrl(request, response);          break;      case 3:          useRefererUrl(request, response);          break;      default:          useDefaultUrl(request, response);      }  }    private void clearAuthenticationAttributes(HttpServletRequest request) {      HttpSession session = request.getSession(false);        if (session == null) {          return;      }        session.removeAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION);  }    private void useTargetUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{      SavedRequest savedRequest = requestCache.getRequest(request, response);      if(savedRequest != null){          requestCache.removeRequest(request, response);      }      String targetUrl = request.getParameter(targetUrlParameter);      redirectStrategy.sendRedirect(request, response, targetUrl);  }    private void useSessionUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{      SavedRequest savedRequest = requestCache.getRequest(request, response);      String targetUrl = savedRequest.getRedirectUrl();      redirectStrategy.sendRedirect(request, response, targetUrl);  }    private void useRefererUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{      String targetUrl = request.getHeader("REFERER");      redirectStrategy.sendRedirect(request, response, targetUrl);  }    private void useDefaultUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{      redirectStrategy.sendRedirect(request, response, defaultUrl);  } |   이 코드에 대한 분석은 그리 어렵지는 않을것이다. AuthenticationHandler 인터페이스에서 정의한 onAuthenticationSuccess 메소드에서 우선순위를 결정짓고 그 우선순위에 따라 이동할 URL을 추출한 뒤 이동하는 것이 전부이다. 상세한 내용은 방금 이 코드를 보여주기 전의 코드에서 우선순위를 결정하는 부분에 대한 설명을 곱씹어보면 해당 순위일때 어떤식으로 URL을 꺼내오는지에 대한 파악이 될 것이다. 여기서는 이전 코드에서 설명하지 않은 부분 위주로 설명을 하겠다.  Spring Security에서 로그인하는 과정에서 로그인이 실패한 경우에 세션에 관련 에러를 저장한다고 예전 블로그에서 언급한 바가 있다. 로그인 화면을 만났을때 로그인 시도 첫번째에 성공했다면 이런 에러가 세션에 저장되어 있지 않았겠지만 어디 그게 순탄하게만 가겠는가? 로그인이 실패한 상황이 한번이라도 발생했으면 에러가 세션에 저장되어 있을것이다. 근데 이런 상태에서 로그인이 성공했다고 생각해보자. 그러면 세션에 있는 에러를 지워야 하지 않겠는가? 그 역할을 하는 것이 clearAuthenticationAttributes 메소드이다. 이 메소드를 보면 세션을 받아와서 WebAttributes.AUTHENTICATION\_EXCEPTION 변수에 정의된 이름으로 된 세션 값을 지우고 있다. 이 변수에 저장된 값은 SPRING\_SECURITY\_LAST\_EXCEPTION 이란 문자열로써 정리하면 Spring Security는 에러 발생시 SPRING\_SECURITY\_LAST\_EXCEPTION이란 key 값으로 저장함을 알 수가 있다.  화면을 이동할때는 org.springframework.security.web.RedirectStrategy 인터페이스를 구현한 객체를 받아서 하고 있다. RedirectStrategy는 Spring Security가 화면 이동에 대한 규칙을 정의하는 부분을 만든 인터페이스로 이 인터페이스를 구현한 객체로 화면 redirect를 하면 된다. 이 인터페이스는 다음의 메소드가 정의되어 있다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 1 | void sendRedirect(HttpServletRequest request, HttpServletResponse response, String url) throws IOException; |   RedirectStrategy 인터페이스를 구현한 클래스를 만들어 거기에서 sendRedirect를 재정의 해주면 되는 것이다. HttpServletRequeet 객체, HttpServletResponse 객체, 이동할 URL을 받아오기 때문에 화면 이동을 구현하는데 아무 지장이 없을 것이다.(redirect가 아니라 forward로 할려고 할 경우도 HttpServletRequest 객체를 받아오기 때문에 가능하다) Spring Security는 이 RedirectStrategy 인터페이스를 구현한 org.springframework.security.web.DefaultRedirectStrategy 클래스를 제공한다. 여기서는 이 클래스를 이용해서 화면을 이동했다. 이 클래스 소스를 보면 내부적으로 HttpServletResponse의 sendRedirect 메소드를 사용해서 화면을 이동하고 있다.  이제 이렇게 설명한 내용이 모두 반영된 CustomAuthenticationSuccessHandler 클래스의 소스를 보면 다음과 같다  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183 | import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;    import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.security.core.Authentication;  import org.springframework.security.web.DefaultRedirectStrategy;  import org.springframework.security.web.RedirectStrategy;  import org.springframework.security.web.WebAttributes;  import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  import org.springframework.security.web.savedrequest.HttpSessionRequestCache;  import org.springframework.security.web.savedrequest.RequestCache;  import org.springframework.security.web.savedrequest.SavedRequest;  import org.springframework.util.StringUtils;    public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {  private Logger logger = LoggerFactory.getLogger(this.getClass());        private RequestCache requestCache = new HttpSessionRequestCache();        private String targetUrlParameter;        private String defaultUrl;        private boolean useReferer;        private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();        public CustomAuthenticationSuccessHandler(){          targetUrlParameter = "";          defaultUrl = "/";          useReferer = false;      }        public String getTargetUrlParameter() {          return targetUrlParameter;      }            public void setTargetUrlParameter(String targetUrlParameter) {          this.targetUrlParameter = targetUrlParameter;      }            public String getDefaultUrl() {          return defaultUrl;      }            public void setDefaultUrl(String defaultUrl) {          this.defaultUrl = defaultUrl;      }            public boolean isUseReferer() {          return useReferer;      }            public void setUseReferer(boolean useReferer) {          this.useReferer = useReferer;      }            @Override      public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {          // TODO Auto-generated method stub            clearAuthenticationAttributes(request);            int intRedirectStrategy = decideRedirectStrategy(request, response);          switch(intRedirectStrategy){          case 1:              useTargetUrl(request, response);              break;          case 2:              useSessionUrl(request, response);              break;          case 3:              useRefererUrl(request, response);              break;          default:              useDefaultUrl(request, response);          }      }        private void clearAuthenticationAttributes(HttpServletRequest request) {          HttpSession session = request.getSession(false);            if (session == null) {              return;          }            session.removeAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION);      }        private void useTargetUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{          SavedRequest savedRequest = requestCache.getRequest(request, response);          if(savedRequest != null){              requestCache.removeRequest(request, response);          }          String targetUrl = request.getParameter(targetUrlParameter);          redirectStrategy.sendRedirect(request, response, targetUrl);      }        private void useSessionUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{          SavedRequest savedRequest = requestCache.getRequest(request, response);          String targetUrl = savedRequest.getRedirectUrl();          redirectStrategy.sendRedirect(request, response, targetUrl);      }        private void useRefererUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{          String targetUrl = request.getHeader("REFERER");          redirectStrategy.sendRedirect(request, response, targetUrl);      }        private void useDefaultUrl(HttpServletRequest request, HttpServletResponse response) throws IOException{          redirectStrategy.sendRedirect(request, response, defaultUrl);      }        /\*\*       \* 인증 성공후 어떤 URL로 redirect 할지를 결정한다       \* 판단 기준은 targetUrlParameter 값을 읽은 URL이 존재할 경우 그것을 1순위       \* 1순위 URL이 없을 경우 Spring Security가 세션에 저장한 URL을 2순위       \* 2순위 URL이 없을 경우 Request의 REFERER를 사용하고 그 REFERER URL이 존재할 경우 그 URL을 3순위       \* 3순위 URL이 없을 경우 Default URL을 4순위로 한다       \* @param request       \* @param response       \* @return   1 : targetUrlParameter 값을 읽은 URL       \*            2 : Session에 저장되어 있는 URL       \*            3 : referer 헤더에 있는 url       \*            0 : default url       \*/      private int decideRedirectStrategy(HttpServletRequest request, HttpServletResponse response){          int result = 0;            SavedRequest savedRequest = requestCache.getRequest(request, response);            if(!"".equals(targetUrlParameter)){              String targetUrl = request.getParameter(targetUrlParameter);              if(StringUtils.hasText(targetUrl)){                  result = 1;              }else{                  if(savedRequest != null){                      result = 2;                  }else{                      String refererUrl = request.getHeader("REFERER");                      if(useReferer && StringUtils.hasText(refererUrl)){                          result = 3;                      }else{                          result = 0;                      }                  }              }                return result;          }            if(savedRequest != null){              result = 2;              return result;          }            String refererUrl = request.getHeader("REFERER");          if(useReferer && StringUtils.hasText(refererUrl)){              result = 3;          }else{              result = 0;          }            return result;      }  } |   그럼 이제는 이렇게 만든 CustomAuthenticationSuccessHandler 클래스를 Spring Security에서 사용하겠다고 적용해야 하지 않겠는가? 그것은 다음과 같이 적용한다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <form-login      username-parameter="loginid"      password-parameter="loginpwd"      login-page="/login.do"      default-target-url="/main.do"      authentication-failure-url="/login.do?fail=true"      authentication-success-handler-ref="customAuthenticationSuccessHandler"  />    <beans:bean id="customAuthenticationSuccessHandler" class="com.terry.springsecurity.common.security.handler.CustomAuthenticationSuccessHandler">      <beans:property name="targetUrlParameter" value="loginRedirect" />      <beans:property name="useReferer" value="false" />      <beans:property name="defaultUrl" value="/main.do" />  </beans:bean> |   CustomAuthenticationSuccessHandler 클래스를 bean으로 등록한 뒤에 <form-login> 태그에 authentication-success-handler-ref 속성에 bean으로 등록한 CustomAuthenticationSuccessHandler 클래스의 id를 넣으면 된다. CustomAuthenticationSuccessHandler 클래스에 보면 targetUrlParameter 속성와 useReferer 속성과 defaultUrl 속성을 지정한 부분이 있다. 로그인 성공한 뒤의 이동할 화면 URL이 들어있는 파라미터 이름을 targetUrlParameter 속성에 넣는다(이 부분은 밑에서 변경된 로그인 페이지와 같이 보면 이해할 수 있을 것이다) useReferer 는 REFERER 헤더 값을 사용할 것인지의 여부를 결정하는 것으로 원래 기본값은 false로 되어 있는 속성이긴 하지만 예시를 위해 지정도 해보았다. true를 할 경우 REFERER 헤더 값을 사용하겠다는 의미이다. 마지막으로 defaultUrl 속성은 위에서 언급했던 우선순위에 모두 만족되지 않을 경우 마지막으로 정해지는 default URL을 설정하는 것으로 여기서는 main 화면 URL을 설정했다.  targetUrlParameter 속성을 넣었기 때문에 우리는 로그인 화면을 약산 수정할 필요가 있다. 왜냐면 로그인 화면에서 아이디와 비밀번호뿐만 아니라 로그인 성공시 이동해야 할 URL도 같이 넘겨줘야 하기 때문이다(물론 안념겨줄수도 있다. 그럴 경우는 정해진 우선순위에 따라 이동할 URL을 결정할 것이다. 다음의 로그인 화면 html 소스를 보자  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | <%@ page contentType="text/html;charset=UTF-8" language="java" %>  <html>  <head>  <title></title>  <jsp:include page="/WEB-INF/views/include/jsInclude.jsp"></jsp:include>  <script type="text/javascript">  $(document).ready(function (){        $("#loginbtn").click(function(){          if($("#loginid").val() == ""){              alert("로그인 아이디를 입력해주세요");              $("#loginid").focus();          }else if($("#loginpwd").val() == ""){              alert("로그인 비밀번호를 입력해주세요");              $("#loginpwd").focus();          }else{              $("#loginfrm").attr("action", "<c:url value='/j\_spring\_security\_check'/>");              $("#loginfrm").submit();          }      });    });  </script>  </head>  <body>  <div style="display:inline-block;">      로그인 화면      <form id="loginfrm" name="loginfrm" action="<c:url value='${ctx}/j\_spring\_security\_check'/>" method="POST">      <table>          <tr>              <td>아이디</td>              <td>                  <input type="text" id="loginid" name="loginid" value="" />              </td>              <td rowspan="2">                  <input type="button" id="loginbtn" value="확인" />              </td>          </tr>          <tr>              <td>비밀번호</td>              <td>                  <input type="text" id="loginpwd" name="loginpwd" value="" />              </td>          </tr>          <c:if test="${not empty param.fail}">          <tr>              <td colspan="2">                  <font color="red">                  <p>Your login attempt was not successful, try again.</p>                  <p>Reason: ${sessionScope["SPRING\_SECURITY\_LAST\_EXCEPTION"].message}</p>                  </font>                  <c:remove scope="session" var="SPRING\_SECURITY\_LAST\_EXCEPTION"/>              </td>          </tr>          </c:if>      </table>      <input type="hidden" name="loginRedirect" value="${loginRedirect}" />      </form>  </div>  </body>  </html> |   로그인 화면 커스터마이징 글에서 로그인 화면 소스를 보여준 적이 있다. 여기서 추가된 부분이 있는데 </table>과 </form> 태그 사이에 있는 다음의 내용이다.  [?](http://zgundam.tistory.com/52)   |  |  | | --- | --- | | 1 | <input type="hidden" name="loginRedirect" value="${loginRedirect}" /> |   hidden type으로 name이 loginRedirect에 Spring MVC Model 객체에서 loginRedirect 로 넣은 값을 value로 셋팅 해주고 있다. 바로 name에 설정한 loginRedirect란 값이 위에서 CustomAuthenticationHandler 클래스를 bean으로 등록할때 targetUrlParameter 속성에 설정한 값인 loginRedirect인 것이다. 즉 name이 loginRedirect란 hidden 태그에 이동해야 할 URL을 설정하면 로그인 성공시 request.getPameter("loginRedirect")를 함으로써 로그인 성공후 이동해야 할 URL을 읽어오게 되는 것이다.  그러면 Spring MVC Model 객체에 loginRedirect 값은 누가 넣는가? 그건 다음 글에서 설명할 로그인 작업이 실패했을때의 별도 작업을 하는 클래스인 CustomAuthenticationFailureHandler 클래스에서 하게 될 것이다. 원래는 이 글에서 다 설명할려고 했는데 두 글로 나눠야겠다. 설명을 너무 생략하면 내용을 이해하기가 어렵고..어쩔수가 없다. 이번에는 로그인 성공했을 때 부가 작업을 수행하는 방법에 대해 설명했다. 다음에서는 로그인 실패했을때 부가 작업을 수행하는 방법에 대해 설명하겠다.  loginRedirect 로 인해 혼선이 있을 수 있을것 같아 설명해 둘 것이 있다. 로그인 하지 않은 상태에서 권한이 필요한 화면으로 가려고 할 경우 로그인 화면을 먼저 보게 되고 이 로그인 화면에서 로그인을 성공하면 권한이 필요한 화면(충분한 권한이 있지 않을 경우 접근 에러 화면이 대신 나온다)으로 이동하게 된다. 근데 이럴 경우를 보면 hidden 태그인 loginRedirect에는 아무런 값도 들어가지 않는데도 권한이 필요한 화면으로 이동을 할 수가 있다. 이것이 가능한 이유는 Spring Security에서 로그인 화면을 보여주기 전에 위에서 설명했던 DefaultSavedRequest 클래스 객체를 만든뒤 거기에 사용자 요청과 이동 URL을 넣고 이를 세션에 저장한뒤 로그인 화면을 보여주기 때문이다. 그래서 loginRedirect에 아무런 값이 없어도 2순위인 세션 검사에서 만족하기 때문에 여기서 해당 URL로 이동하게 되는 것이다.   |  | | --- | | //http://webcache.googleusercontent.com/search?q=cache:ujW2IphROggJ:zgundam.tistory.com/52+&cd=2&hl=ko&ct=clnk&gl=kr  **public** **class** CustomAuthenticationSuccessHandler **implements**  AuthenticationSuccessHandler {  **private** RequestCache requestCache = **new** HttpSessionRequestCache();  **private** String targetUrlParameter;  **private** String defaultUrl;  **private** **boolean** useReferer;  **private** RedirectStrategy redirectStrategy = **new** DefaultRedirectStrategy();  **public** CustomAuthenticationSuccessHandler() {  targetUrlParameter = "";  defaultUrl = "/";  useReferer = **false**;  }  **public** String getTargetUrlParameter() {  **return** targetUrlParameter;  }  **public** **void** setTargetUrlParameter(String targetUrlParameter) {  **this**.targetUrlParameter = targetUrlParameter;  }  **public** String getDefaultUrl() {  **return** defaultUrl;  }  **public** **void** setDefaultUrl(String defaultUrl) {  **this**.defaultUrl = defaultUrl;  }  **public** **boolean** isUseReferer() {  **return** useReferer;  }  **public** **void** setUseReferer(**boolean** useReferer) {  **this**.useReferer = useReferer;  }  @Override  **public** **void** onAuthenticationSuccess(HttpServletRequest request,HttpServletResponse response, Authentication authentication) **throws** IOException, ServletException {    clearAuthenticationAttributes(request);  //무조건 처음화면으로  // redirectStrategy.sendRedirect(request, response, defaultUrl);  **int** intRedirectStrategy = decideRedirectStrategy(request, response);  **switch** (intRedirectStrategy) {  **case** 1:  useTargetUrl(request, response);  **break**;  **case** 2:  useSessionUrl(request, response);  **break**;  **case** 3:  useRefererUrl(request, response);  **break**;  **default**:  useDefaultUrl(request, response);  }  }  **private** **void** clearAuthenticationAttributes(HttpServletRequest request) {  HttpSession session = request.getSession(**false**);  **if** (session == **null**) {  **return**;  }  session.removeAttribute(WebAttributes.***AUTHENTICATION\_EXCEPTION***);  }  **private** **void** useTargetUrl(HttpServletRequest request,  HttpServletResponse response) **throws** IOException {  SavedRequest savedRequest = requestCache.getRequest(request, response);  **if** (savedRequest != **null**) {  requestCache.removeRequest(request, response);  }  String targetUrl = request.getParameter(targetUrlParameter);  redirectStrategy.sendRedirect(request, response, targetUrl);  }  **private** **void** useSessionUrl(HttpServletRequest request,  HttpServletResponse response) **throws** IOException {  SavedRequest savedRequest = requestCache.getRequest(request, response);  String targetUrl = savedRequest.getRedirectUrl();  redirectStrategy.sendRedirect(request, response, targetUrl);  }  **private** **void** useRefererUrl(HttpServletRequest request,  HttpServletResponse response) **throws** IOException {  String targetUrl = request.getHeader("REFERER");  redirectStrategy.sendRedirect(request, response, targetUrl);  }  **private** **void** useDefaultUrl(HttpServletRequest request,  HttpServletResponse response) **throws** IOException {  redirectStrategy.sendRedirect(request, response, defaultUrl);  }  /\*\*  \* 인증 성공후 어떤 URL로 redirect 할지를 결정한다 판단 기준은 targetUrlParameter 값을 읽은 URL이 존재할  \* 경우 그것을 1순위 1순위 URL이 없을 경우 Spring Security가 세션에 저장한 URL을 2순위 2순위 URL이 없을  \* 경우 Request의 REFERER를 사용하고 그 REFERER URL이 존재할 경우 그 URL을 3순위 3순위 URL이 없을 경우  \* Default URL을 4순위로 한다  \*  \* **@param** request  \* **@param** response  \* **@return** 1 : targetUrlParameter 값을 읽은 URL 2 : Session에 저장되어 있는 URL 3 :  \* referer 헤더에 있는 url 0 : default url  \*/  **private** **int** decideRedirectStrategy(HttpServletRequest request, HttpServletResponse response) {  **int** result = 0;  SavedRequest savedRequest = requestCache.getRequest(request, response);  **if** (!"".equals(targetUrlParameter)) {  String targetUrl = request.getParameter(targetUrlParameter);  **if** (StringUtils.*hasText*(targetUrl)) {  result = 1;  } **else** {  **if** (savedRequest != **null**) {  result = 2;  } **else** {  String refererUrl = request.getHeader("REFERER");  **if** (useReferer && StringUtils.*hasText*(refererUrl)) {  result = 3;  } **else** {  result = 0;  }  }  }  **return** result;  }  **if** (savedRequest != **null**) {  result = 2;  **return** result;  }  String refererUrl = request.getHeader("REFERER");  **if** (useReferer && StringUtils.*hasText*(refererUrl)) {  result = 3;  } **else** {  result = 0;  }  **return** result;  }  } | |
| 스프링 시큐리티 커스텀태그에서 expression을 사용하려면  <sec:http auto-config=*"true"* use-expressions=*"true"*> csrf … <sec:csrf disabled="true"/> //🡨csrf사용안함 flag |

Security

|  |
| --- |
| <bean id=*"parancoeFilterSecurityInterceptor"* class=*"org.springframework.security.web.access.intercept.FilterSecurityInterceptor"* >  <property name=*"authenticationManager"* ref=*"org.springframework.security.authenticationManager"*/>  <property name=*"accessDecisionManager"* ref=*"accessDecisionManager"*/>  <property name=*"securityMetadataSource"* ref=*"customFilterInvocationSecurityMetadataSource"*/>  </bean>  <!-- <sec:filter-security-metadata-source id="securityMetadataSource" use-expressions="true"> -->  <!-- <sec:intercept-url pattern="/admin/\*\*" access="hasRole('ROLE\_ADMIN')" /> -->  <!-- </sec:filter-security-metadata-source> -->    <bean id=*"accessDecisionManager"* class=*"org.springframework.security.access.vote.AffirmativeBased"*>  <constructor-arg>  <list>  <bean class=*"org.springframework.security.access.vote.RoleVoter"*>  <property name=*"rolePrefix"* value=*""* />  </bean>  </list>  </constructor-arg>  <property name=*"allowIfAllAbstainDecisions"* value=*"false"* />  </bean>    <!-- 초기 부트됐을때 메뉴 리스트를 가져온걸로 페이지 호출시 매번 해당페이지가 관리포인트 인지 확인처리 합니다. -->  <!-- //여기서 지정한 권한이 있어야지만 페이지 안쪽으로 들어갈수 있다!!!!! -->  <bean id=*"customFilterInvocationSecurityMetadataSource"* class=*"com.omnicns.security.spring.CustomFilterInvocationSecurityMetadataSource"* />  <!-- 로그인 처리 -->  <bean id=*"customPasswordEncoder"* class=*"com.omnicns.security.spring.CustomPasswordEncoder"* />  <bean id=*"customAuthenticationProvider"* class=*"com.omnicns.security.spring.CustomAuthenticationProvider"*/>    <!-- 로그인 완료후 처리할 handler -->  <bean id=*"customAuthenticationSuccessHandler"* class=*"com.omnicns.security.spring.CustomAuthenticationSuccessHandler"*/> |
| **public** **class** CustomFilterInvocationSecurityMetadataSource **implements** FilterInvocationSecurityMetadataSource{  BootManager bootMng = BootManager.*getInstance*();  SecurityManager securityMng = SecurityManager.*getInstance*();  @Autowired  **private** ConfigManager configMng;  Logger log = LoggerFactory.*getLogger*(**this**.getClass());  @Override  **public** Collection<ConfigAttribute> getAttributes(Object object) **throws** IllegalArgumentException {      FilterInvocation fi = (FilterInvocation) object;  // String url = fi.getHttpRequest().getRequestURI();  String uri = RequestUtil.*getURI*(fi.getHttpRequest());  String httpMethod = fi.getRequest().getMethod();  // List<ConfigAttribute> attributes = new ArrayList<com.omnicns.web.spring.security.ConfigAttribute>();  List<ConfigAttribute> attributes = **new** ArrayList<ConfigAttribute>();      com.omnicns.web.spring.security.ConfigAttribute a = **null**;  a = **new** com.omnicns.web.spring.security.ConfigAttribute("ROLE\_SUPER");  attributes.add(a);    **if**(uri.equals(configMng.getParam("root\_uri"))||uri.startsWith(configMng.getParam("anon\_uri"))||uri.startsWith(configMng.getParam("sign\_uri"))){  a = **new** com.omnicns.web.spring.security.ConfigAttribute("ROLE\_ANONYMOUS");  attributes.add(a);  a = **new** com.omnicns.web.spring.security.ConfigAttribute("ROLE\_AUTH");  attributes.add(a);  }          //여기서 지정한 권한이 있어야지만 안쪽페이지로 들어갈수 있다!!!!!  **if**(**null**!=securityMng.getSecurityAuthorities() && SecurityUtil.*isLogin*()){  LoginUserVO user = securityMng.getSecurityUser();  **if**(user.isEnabled()){  securityMng.getSecurityAuthorities().stream().forEach(at->{//.filter(at->user.isEnabled())  LoginRoleAuthVVO auth = at.getAuth();  **if**(**null**!=auth){  String userURI= auth.getRight().getUrl();  **if**(**null**!=uri && **null**!=userURI && uri.indexOf(userURI)==0){  com.omnicns.web.spring.security.ConfigAttribute u = **new** com.omnicns.web.spring.security.ConfigAttribute(at.getAuthority());  attributes.add(u);  }  }  });  }**else**{//사용가능하지 않은사람 즉 카드 인증된지 않은사람은 /authcheck 도허용한다!  **if**(uri.startsWith(configMng.getParam("authcheck\_uri"))){  a = **new** com.omnicns.web.spring.security.ConfigAttribute("ROLE\_AUTH");  attributes.add(a);  }  }      }  // BootMenuVO menu = bootManager.getMenu(url);  // if(null!=menu){  // attributes.add(new CustomConfigAttribute(menu.getMnu\_no()));  // }  // Lookup your database (or other source) using this information and populate the  // list of attributes  // for (int i = 0; null!=bootSelectMenu && i < bootSelectMenu.size(); i++) {  // BootMenuVO at = bootSelectMenu.get(i);  // if(url.equals(at.getMnu\_dir())){  // attributes.add(new CustomConfigAttribute(bootSelectMenu.get(i).getMnu\_no()));  // }  // }  // if("/".equals(url)){  // attributes.add(new com.omnicns.web.spring.security.ConfigAttribute("Anonymous"));  // }else{  // return null;  // }  **return** attributes;  }  @Override  **public** Collection<ConfigAttribute> getAllConfigAttributes() {  **return** **null**;  }  @Override  **public** **boolean** supports(Class<?> clazz) {  **return** FilterInvocation.**class**.isAssignableFrom(clazz);  }  } |
| **public** **class** CustomPasswordEncoder **implements** PasswordEncoder{    String key="";  **public** CustomPasswordEncoder() {  }  **public** CustomPasswordEncoder(String key) {  **this**.key = key;  }    **public** String getKey() {  **return** key;  }  **public** **void** setKey(String key) {  **this**.key = key;  }  @Override  **public** String encode(CharSequence rawPassword) {  String r = "";  **try** {  r = SecurityManager.*getInstance*().encrypt(getKey(), rawPassword.toString());  } **catch** (UnsupportedEncodingException | NoSuchAlgorithmException e) {  e.printStackTrace();  }  **return** r;  }  @Override  **public** **boolean** matches(CharSequence inputPassword, String encodedPasswordByDB) {  inputPassword = encode(inputPassword);  **return** inputPassword.toString().equals(encodedPasswordByDB);  }  } |
| **public** **class** CustomAuthenticationProvider **extends** AuthenticationProvider<LoginUserVO> {  @Autowired  **private** CustomPasswordEncoder customPasswordEncoder;  @Autowired  **private** LoginService loginService;  @Autowired  **private** ConfigManager configMng;    Logger log = LoggerFactory.*getLogger*(**this**.getClass());    @Override  **public** LoginUserVO findUserById(Authentication authentication) **throws** Exception {  WebAuthenticationDetails detail = (WebAuthenticationDetails) authentication.getDetails();  String remoteIP = detail.getRemoteAddress();  String user\_id = (String)authentication.getPrincipal();  String user\_pw = (String)authentication.getCredentials();  LoginUserVO users= loginService.getLoginUsers(user\_id);  log.info("Login try ip : -> "+detail.getRemoteAddress()+" input\_id:"+user\_id+" "+users);  **if**(**null**==users ){  **throw** **new** UsernameNotFoundException("로그인 정보가 올바르지 않습니다.");  }    //////ipcheck  **if**(**null**!=users.getIp() && **null**!=users.getIp().getIp\_number() && users.getIp().getIp\_number().length()>0){  String ip = users.getIp().getIp\_number();  String[] ips = ip.split(",");  **boolean** authIP = **false**;  **for**(String atIP : ips){  atIP = atIP.trim();  atIP = atIP.replaceAll("\\.", "\\\\\\.");  atIP = atIP.replaceAll("\\\*", "\\.\*");  atIP = atIP.replaceAll("\\+", "\\.+");  **if**(StringUtil.*isMatches*(remoteIP, atIP)){  authIP = **true**;  **break**;  }  }  **if**(authIP==**false**){  **throw** **new** UsernameNotFoundException("허용되지 않은 IP입니다. 운영자에게 문의해주세요.");  }  }          //Lock Check  **if**(!users.isAccountNonLocked()){  **throw** **new** UsernameNotFoundException("계정이 잠겼습니다. 운영자에게 문의해주세요.");  }    ////ipcheck    //카드 체크  **if**(**null** == users.getCard\_no()){  users.setEnabled(**true**);  }**else**{  users.setEnabled(**false**); //카드번호가 있으면 카드 체크해야된다 로그인 로직 한단계 더있다.  **if**(**null**!=users.getCheckCard()){  users.getCheckCard().refreshCheckCodeId();  };    }    //pwd expired 체크  **if**(**null**==users.getLast\_pwd\_update()){  users.setPwdNonExpired(**false**);  }**else**{  Date atDate = DateUtil.*getDate*("yyyyMMddHHmmss",users.getLast\_pwd\_update());  Date mDate = DateUtil.*modifyDate*(atDate, Calendar.***DATE***, Integer.*parseInt*(configMng.getParam("login\_pwd\_expired\_day")));  //String atDateStr = DateUtil.dateFormat("yyyyMMddHHmmss", atDate);  //String mDateStr = DateUtil.dateFormat("yyyyMMddHHmmss", mDate);  **if**(mDate.getTime()<=**new** Date().getTime()){  users.setPwdNonExpired(**false**);  }**else**{  users.setPwdNonExpired(**true**);  }  }      **return** users;  }  @Override  **public** List<? **extends** CustomGrantedObjAuthority> getPermission(LoginUserVO userInfo) {  List<LoginRoleAuthVVO> roleAuths = loginService.getRoleAuth(userInfo.getOperator\_id());  List<CustomGrantedObjAuthority> list = **new** ArrayList<CustomGrantedObjAuthority>();  // Set<String> roleSet = new HashSet<String>() ;  HashMap<String, LoginRoleAuthVVO> roleSet = **new** HashMap<String, LoginRoleAuthVVO>() ;  **for** (**int** i = 0; **null**!=roleAuths && i < roleAuths.size(); i++) {  LoginRoleAuthVVO at = roleAuths.get(i);  list.add(**new** CustomGrantedObjAuthority(at.getRight().getCode(), at));//일반 메뉴  // list.add(new CustomGrantedObjAuthority(at.getUri()+"\_"+at.getAuth(), at));  // if("M".equals(at.getRight().getType())) //M인것만 ROLE처리 <--왜그랬을까..?  roleSet.put(at.getRight\_group().getCode(),at);  }    //롤자체를 넣는다. 권한있는자만 허용가능한 페이지 넣는다.  LoginRoleAuthVVO auth = **new** LoginRoleAuthVVO();  LoginRoleAuthRightVO right = **new** LoginRoleAuthRightVO();  right.setUrl(configMng.getParam("auth\_uri"));  auth.setRight(right);  list.add(**new** CustomGrantedObjAuthority("ROLE\_AUTH", auth)); //로그인된사람은 이롤을 무조건 갖는다.    //ROLE은 prefix가 ROLE\_ 이여야한다. 그러면 access="hasRole('/board\_R')" 또는 ROLE\_/board..등으로찾을수 있다  //hasRole이라는 함수로 찾을수 있다.  roleSet.entrySet().stream().forEach(at->{  list.add(**new** CustomGrantedObjAuthority("ROLE\_"+at.getKey(), at.getValue()));  }); //그룹은 ROLE    //user session 에 권한셋팅  userInfo.setAuthorities(list);  **return** list;  }  @Override  **public** **boolean** isValidate(Authentication authentication, LoginUserVO getUserInfo) {  String user\_id = (String)authentication.getPrincipal();  String user\_pw = (String)authentication.getCredentials();  customPasswordEncoder.setKey(getUserInfo.getIndv\_key());  **return** customPasswordEncoder.matches(user\_pw, getUserInfo.getLogin\_pw());  }    @Override // 로그인 성공 했을시..  **public** **void** succse(Authentication authentication, LoginUserVO userInfo) **throws** Exception {  WebAuthenticationDetails detail = (WebAuthenticationDetails) authentication.getDetails();  userInfo.setLast\_login\_ip(detail.getRemoteAddress());  userInfo.setLogin\_fail(0);  userInfo.setLast\_login\_session(RandomUtil.*getRandomString*(100));  userInfo.setLast\_login\_date(DateUtil.*getDate*("yyyyMMddHHmmss"));  log.info("Login succse ip : -> "+detail.getRemoteAddress()+" id:"+userInfo.getLogin\_id()+" name : "+userInfo.getUsername());  /\*  localhost에서 테스트 하는 경우 0:0:0:0:0:0:0:1 값으로 넘어 오는 경우가 있다.  이 값은 IPv6 에서 IPv4의 127.0.0.1 과 같은 값이다.  Tomcat으로 개발시 이게 문제가 되는 경우 vm arguments에 -Djava.net.preferIPv4Stack=true 값을 넣어 주면 된다.  \*/  // log.debug("Login User\_IP -> "+detail.getRemoteAddress());  // log.debug("Login User\_IP -> "+Content.getRequest().getRemoteHost());  // log.debug("Login User\_IP -> "+RequestUtil.getRemoteAddr(Content.getRequest()));  // log.debug("Login User\_IP -> "+Content.getRequest().getRemoteAddr());  loginService.updateLogin(userInfo);  }  @Override  **public** **void** fail(Authentication authentication, LoginUserVO userInfo) **throws** Exception {  WebAuthenticationDetails detail = (WebAuthenticationDetails) authentication.getDetails();  **int** fail=userInfo.getLogin\_fail()+1;  userInfo.setLogin\_fail(fail);  **if**(fail>= Integer.*parseInt*(configMng.getParam("login\_fail\_max"))){  userInfo.setLocked("Y");  }  log.info("Login fail ip : -> "+detail.getRemoteAddress()+" id:"+userInfo.getLogin\_id()+" name : "+userInfo.getUsername()+" failcnt : "+fail);  loginService.updateLogin(userInfo);  }    } |

Security 자동로그인

|  |
| --- |
| String login\_id = configMng.getParam("dits\_login\_id");  String login\_pwd = configMng.getParam("dits\_login\_pwd");  UsernamePasswordAuthenticationToken authentication = **new** UsernamePasswordAuthenticationToken(login\_id, login\_pwd);  WebAuthenticationDetails details = **new** WebAuthenticationDetails(request);  authentication.setDetails(details);  Authentication au = customAuthenticationProvider.authenticate(authentication)  SecurityContextHolder.*getContext*().setAuthentication(au); |
| **public** **class** CustomAuthenticationProvider **extends** AuthenticationProvider<LoginUserVO> {  @Autowired  **private** CustomPasswordEncoder customPasswordEncoder;  @Autowired  **private** LoginService loginService;  @Autowired  **private** ConfigManager configMng;  // @Autowired  // private HttpServletRequest request;    Logger log = LoggerFactory.*getLogger*(**this**.getClass());    @Override  **public** LoginUserVO findUserById(Authentication authentication) **throws** Exception {  String user\_id = (String)authentication.getPrincipal();  String user\_pw = (String)authentication.getCredentials();    LoginUserVO users= loginService.getLoginUsers(user\_id);  **if**(**null**==users ){  **throw** **new** UsernameNotFoundException("로그인 정보가 올바르지 않습니다.");  }    //Lock Check  **if**(!users.isAccountNonLocked()){  **throw** **new** UsernameNotFoundException("계정이 잠겼습니다. 운영자에게 문의해주세요.");  }      //카드 체크  **if**(**null** == users.getCard\_no()){  users.setEnabled(**true**);  }**else**{  users.setEnabled(**false**); //카드번호가 있으면 카드 체크해야된다 로그인 로직 한단계 더있다.  **if**(**null**!=users.getCheckCard()){  users.getCheckCard().refreshCheckCodeId();  };    }    //pwd expired 체크  **if**(**null**==users.getLast\_pwd\_update()){  users.setPwdNonExpired(**false**);  }**else**{  Date atDate = DateUtil.*getDate*("yyyyMMddHHmmss",users.getLast\_pwd\_update());  Date mDate = DateUtil.*modifyDate*(atDate, Calendar.***DATE***, Integer.*parseInt*(configMng.getParam("login\_pwd\_expired\_day")));  //String atDateStr = DateUtil.dateFormat("yyyyMMddHHmmss", atDate);  //String mDateStr = DateUtil.dateFormat("yyyyMMddHHmmss", mDate);  **if**(mDate.getTime()<=**new** Date().getTime()){  users.setPwdNonExpired(**false**);  }**else**{  users.setPwdNonExpired(**true**);  }  }      **return** users;  }  @Override  **public** List<? **extends** CustomGrantedObjAuthority> getPermission(LoginUserVO userInfo) {  List<LoginRoleAuthVVO> roleAuths = loginService.getRoleAuth(userInfo.getOperator\_id());  List<CustomGrantedObjAuthority> list = **new** ArrayList<CustomGrantedObjAuthority>();  // Set<String> roleSet = new HashSet<String>() ;  HashMap<String, LoginRoleAuthVVO> roleSet = **new** HashMap<String, LoginRoleAuthVVO>() ;  **for** (**int** i = 0; **null**!=roleAuths && i < roleAuths.size(); i++) {  LoginRoleAuthVVO at = roleAuths.get(i);  list.add(**new** CustomGrantedObjAuthority(at.getRight().getCode(), at));//일반 메뉴  // list.add(new CustomGrantedObjAuthority(at.getUri()+"\_"+at.getAuth(), at));  // if("M".equals(at.getRight().getType())) //M인것만 ROLE처리 <--왜그랬을까..?  roleSet.put(at.getRight\_group().getCode(),at);  }    //롤자체를 넣는다. 권한있는자만 허용가능한 페이지 넣는다.  LoginRoleAuthVVO auth = **new** LoginRoleAuthVVO();  LoginRoleAuthRightVO right = **new** LoginRoleAuthRightVO();  right.setUrl(configMng.getParam("auth\_uri"));  auth.setRight(right);  list.add(**new** CustomGrantedObjAuthority("ROLE\_AUTH", auth)); //로그인된사람은 이롤을 무조건 갖는다.    //ROLE은 prefix가 ROLE\_ 이여야한다. 그러면 access="hasRole('/board\_R')" 또는 ROLE\_/board..등으로찾을수 있다  //hasRole이라는 함수로 찾을수 있다.  roleSet.entrySet().stream().forEach(at->{  list.add(**new** CustomGrantedObjAuthority("ROLE\_"+at.getKey(), at.getValue()));  }); //그룹은 ROLE    //user session 에 권한셋팅  userInfo.setAuthorities(list);  **return** list;  }  @Override  **public** **boolean** isValidate(Authentication authentication, LoginUserVO getUserInfo) {  String user\_id = (String)authentication.getPrincipal();  String user\_pw = (String)authentication.getCredentials();  customPasswordEncoder.setKey(getUserInfo.getIndv\_key());  **return** customPasswordEncoder.matches(user\_pw, getUserInfo.getLogin\_pw());  }    @Override // 로그인 성공 했을시..  **public** **void** succse(Authentication authentication, LoginUserVO userInfo) **throws** Exception {  WebAuthenticationDetails detail = (WebAuthenticationDetails) authentication.getDetails();  userInfo.setLast\_login\_ip(detail.getRemoteAddress());  userInfo.setLogin\_fail(0);  userInfo.setLast\_login\_session(RandomUtil.*getRandomString*(100));  userInfo.setLast\_login\_date(DateUtil.*getDate*("yyyyMMddHHmmss"));  /\*  localhost에서 테스트 하는 경우 0:0:0:0:0:0:0:1 값으로 넘어 오는 경우가 있다.  이 값은 IPv6 에서 IPv4의 127.0.0.1 과 같은 값이다.  Tomcat으로 개발시 이게 문제가 되는 경우 vm arguments에 -Djava.net.preferIPv4Stack=true 값을 넣어 주면 된다.  \*/  // log.debug("Login User\_IP -> "+detail.getRemoteAddress());  // log.debug("Login User\_IP -> "+Content.getRequest().getRemoteHost());  // log.debug("Login User\_IP -> "+RequestUtil.getRemoteAddr(Content.getRequest()));  // log.debug("Login User\_IP -> "+Content.getRequest().getRemoteAddr());  loginService.updateLogin(userInfo);  }  @Override  **public** **void** fail(Authentication authentication, LoginUserVO userInfo) **throws** Exception {  **int** fail=userInfo.getLogin\_fail()+1;  userInfo.setLogin\_fail(fail);  **if**(fail>= Integer.*parseInt*(configMng.getParam("login\_fail\_max"))){  userInfo.setLocked("Y");  }  loginService.updateLogin(userInfo);  }    } |
| **public** **abstract** **class** AuthenticationProvider<T **extends** UserDetails> **implements** org.springframework.security.authentication.AuthenticationProvider{  /\*    <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:sec="http://www.springframework.org/schema/security" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/security  http://www.springframework.org/schema/security/spring-security.xsd">  <sec:http pattern="/css/\*\*" security="none" />  <sec:http pattern="/js/\*\*" security="none" />  <sec:http pattern="/images/\*\*" security="none" />  <sec:http pattern="/resources/\*\*" security="none" />  <!-- URL 접근제한을 주기위한 셋팅 DB에서불러와서 처리한다 -->  <sec:custom-filter ref="parancoeFilterSecurityInterceptor" before="FILTER\_SECURITY\_INTERCEPTOR" />      <bean id="parancoeFilterSecurityInterceptor" class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor" >  <property name="authenticationManager" ref="org.springframework.security.authenticationManager"/>  <property name="accessDecisionManager" ref="accessDecisionManager"/>  <property name="securityMetadataSource" ref="customFilterInvocationSecurityMetadataSource"/>  </bean>    <bean id="accessDecisionManager" class="org.springframework.security.access.vote.AffirmativeBased">  <constructor-arg>  <list>  <bean class="org.springframework.security.access.vote.RoleVoter">  <property name="rolePrefix" value="" />  </bean>  </list>  </constructor-arg>  <property name="allowIfAllAbstainDecisions" value="false" />  </bean>    <bean id="customFilterInvocationSecurityMetadataSource" class="com.ko.omnicns.omnifit.boot.security.CustomFilterInvocationSecurityMetadataSource" />    \*/  **public** AuthenticationProvider() {  }    @Override  **public** **boolean** supports(Class<?> authentication) {  **return** UsernamePasswordAuthenticationToken.**class**.isAssignableFrom(authentication);//return authentication.equals(UsernamePasswordAuthenticationToken.class);  }    // public boolean preHandle(T userinfo, List<String> permission){  // return true;  // }  **public** **abstract** **void** succse(Authentication authentication, T userInfo) **throws** Exception;  **public** **abstract** **void** fail(Authentication authentication, T userInfo) **throws** Exception;  @Override  **public** Authentication authenticate(Authentication authentication) **throws** AuthenticationException {  // String user\_id = (String)authentication.getPrincipal();  // String user\_pw = (String)authentication.getCredentials();  // log.debug("Welcome authenticate! {}", user\_id + "/" + user\_pw);  // // check whether user's credentials are valid.  // // if false, throw new BadCredentialsException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.badCredentials", "Bad credentials"));  // List<GrantedAuthority> roles = new ArrayList<GrantedAuthority>();  // roles.add(new SimpleGrantedAuthority("ROLE\_USER"));  // UsernamePasswordAuthenticationToken result = new UsernamePasswordAuthenticationToken(user\_id, user\_pw, roles);  // result.setDetails(new CustomUserDetails(user\_id, user\_pw));  // return result;    // UsernamePasswordAuthenticationToken authToken = (UsernamePasswordAuthenticationToken) authentication;  // String user\_id = (String)authentication.getPrincipal();  // String user\_name = (String)authToken.getName();  // String user\_pw = (String)authentication.getCredentials();      //log.debug("Securety: id:"+user\_id+" name:"+user\_name+" pw:"+user\_pw);  T userInfo;  **try** {  userInfo = findUserById(authentication);//db에서가져온다.  } **catch** (Exception e) {  **throw** **new** AuthenticationServiceException(e.getMessage());  }    **if** (userInfo == **null**) {  **throw** **new** UsernameNotFoundException("로그인 정보가 올바르지 않습니다.");  }  **try**{  **if** (!isValidate(authentication, userInfo)) {  fail(authentication, userInfo);  **throw** **new** BadCredentialsException("로그인 정보가 올바르지 않습니다. ");  }  }**catch**(Exception e){  **throw** **new** AuthenticationServiceException(e.getMessage());  }  // List<CustomGrantedAuthority> authorities = Collections.emptyList();  List<? **extends** GrantedObjAuthority> permission = getPermission(userInfo);  // try {  // authorities.addAll(permission);  // for (int i = 0; null!=permission && i < permission.size(); i++) {  // if(i==0){authorities = new ArrayList<GrantedAuthority>();};  // SimpleGrantedAuthority granted = new SimpleGrantedAuthority(permission.get(i));  // authorities.add(granted);  // }  // } catch (Exception e) {  // throw new AuthenticationServiceException(e.getMessage());  // }      UsernamePasswordAuthenticationToken token = **new** UsernamePasswordAuthenticationToken(userInfo,**null**,permission);  **try**{  succse(authentication, userInfo);  }**catch**(Exception e){  **throw** **new** AuthenticationServiceException(e.getMessage());  }    **return** token;    }    //String user\_id = (String)authentication.getPrincipal();  //String user\_pw = (String)authentication.getCredentials();  **public** **abstract** T findUserById(Authentication authentication) **throws** Exception;  **public** **abstract** List<? **extends** GrantedObjAuthority> getPermission(T userInfo);  //String user\_id = (String)authentication.getPrincipal();  //String user\_pw = (String)authentication.getCredentials();  **public** **abstract** **boolean** isValidate(Authentication authentication, T getUserInfo)**throws** Exception;//matchPassword Check    } |

<http://websystique.com/spring-security/spring-security-4-method-security-using-preauthorize-postauthorize-secured-el/>

jsp단에서 security 정보 빼오기

|  |
| --- |
| <**sec:authentication** var="principal" property="principal" scope="page" /> <%-- USER LOGIN 정보 --%> <**sec:authentication** var="authorities" property="principal.authorities" scope="page"/> <%-- USER ROLE authorities 정보--%> |
| <%--권한 리스트 뿌리는것 --%> <%-- <c:forEach items="${authorities}" var="authority" varStatus="vs">  <p>${authority.authority}</p> </c:forEach> --%> |
| 사용  <%-- http://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html https://dzone.com/refcardz/expression-based-authorization --%> <**sec:authorize** access="hasAuthority('GGIJC801')">  <%--<a href='<c:url value="/board/read" />'>게시판 읽기</a>--%>  <button class="btn btn-xs btn-primary content-create-right-btn" ng-click="create()">등록하기</button> </**sec:authorize**> <**sec:authorize** access="hasAuthority('dddddddGGIJC801')">  <button> 권한없는거</button> </**sec:authorize**> |

Jsr303

|  |
| --- |
|  |
| **1. @NotNull  vs  @NotEmpty  vs  @NotBlank**  **String name = null 인 경우...**   |  |  | | --- | --- | | @NotNull | false | | @NotEmpty | false | | @NotBlank | false |   **String name = "" 인 경우...**   |  |  | | --- | --- | | @NotNull | true | | @NotEmpty | false | | @NotBlank | false |   **String name = " " (white space가 있을 경우)인 경우...**   |  |  | | --- | --- | | @NotNull | true | | @NotEmpty | true | | @NotBlank | false |   **String name = "aaaa" (문자열 있을 경우)**   |  |  | | --- | --- | | @NotNull | true | | @NotEmpty | true | | @NotBlank | true | |

다국어

사용하기위해선 *LocaleChangeInterceptor 사용한다*

localResolver는 3가지 방법이 있다.

쿠키이용, 사용자 요청헤더, Session내용 defalut는 AceeptHeaderLocaleResolver이다.

http://changpd.blogspot.kr/2013/05/localeresolver-messagesource.html

|  |
| --- |
| <bean id=*"localeResolver"* class=*"org.springframework.web.servlet.i18n.SessionLocaleResolver"* />  <!-- 쿠키를 이용한 Locale 이용시 <bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/> -->  <bean id=*"localeChangeInterceptor"* class=*"org.springframework.web.servlet.i18n.LocaleChangeInterceptor"*>  <property name=*"paramName"* value=*"language"* />  </bean>  <bean id="localeResolver"  class="org.springframework.web.servlet.i18n.CookieLocaleResolver" >  <property name="cookieName" value="clientlanguage"/>  <property name="cookieMaxAge" value="100000"/>  <property name="cookiePath" value="web/cookie"/>  </bean> |
| <bean id="localeResolver" class="org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver" />  요청헤더. |
| <bean id="localeResolver"  class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />  세션 |
| LocalChangeInterceptor를 적용시켜주기위해선 인텁셉터를 걸어줘야된다  <mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path="/\*\*" />  <ref bean="localeChangeInterceptor" />  </mvc:interceptor> </mvc:interceptors> |

기본 Locale : Locale.getDefault().toString()

Locale.KOREA = {Locale@9717} "ko\_KR"

Locale.JAPAN = {Locale@9723} "ja\_JP"

Locale.US = {Locale@9726} "en\_US"

Locale.CHINA = {Locale@9729} "zh\_CN"

http://cfile7.uf.tistory.com/image/2551BF395449BC6D199850

Locale locale = LocaleContextHolder.getLocale();

*String korMessage = messageSource.getMessage("hello.test",null, "no surch", Locale.KOREA);//한국 기준 메시지 요청String localeMessage = messageSource.getMessage("hello.test",null, "no surch", localeResolver.resolveLocale(request)); //현재 설정된 위치 기준의 메시지 요청*

jsp는 상단에

<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<spring:message code="code\_name"/>

를 삽입해주면 된다.

자바스크립트에 넣을 경우  
<spring:message code="test.input.name" javaScriptEscape="true"/>  
HTML TAG안에 TEXT로 넣을 경우  
<spring:message code="test.input.name"/>

Database source

|  |
| --- |
| <jdbc:embedded-database id=*"dataSource"* type=*"HSQL"*>  <jdbc:script location= *"classpath:/db/sampledb.sql"*/>  </jdbc:embedded-database>    <!-- JNDI -->  <bean id=*"dataSourceSpied"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>  <property name=*"jndiName"* value=*"jdbc/omnifitOracle"*/>  <property name=*"resourceRef"* value=*"true"*></property>  </bean>    <!-- JDBC LOG를 위한 설정 -->  <bean id=*"dataSource"* class=*"net.sf.log4jdbc.Log4jdbcProxyDataSource"*>  <constructor-arg ref=*"dataSourceSpied"* />  <property name=*"logFormatter"*>  <bean class=*"net.sf.log4jdbc.tools.Log4JdbcCustomFormatter"*>  <property name=*"loggingType"* value=*"MULTI\_LINE"* />  <property name=*"sqlPrefix"* value=*"SQL:::"*/>  </bean>  </property>  </bean> |

<http://lahuman.jabsiri.co.kr/87>

## [[Spring-4]국제화 서비스를 위한 MessageSource 설정하기](http://lahuman.jabsiri.co.kr/87)

[설치&설정 관련/Spring Framework](http://lahuman.jabsiri.co.kr/category/%EC%84%A4%EC%B9%98%26%EC%84%A4%EC%A0%95%20%EA%B4%80%EB%A0%A8/Spring%20Framework) 2014.10.24 00:28

**본 포스팅은 UTF-8 Filter 적용 및 ViewResolver 설정 이후, 국제화 서비스 제공을 위한 MessageSource 설정 하는 내용입니다.**

**GIT 주소 :**[**https://github.com/lahuman/SpringBoot.Sample**](https://github.com/lahuman/SpringBoot.Sample)

국제화(Internationalization)란?

사용자의 Locale에 따라 표출되는 언어를 변경하는 서비스 이며, Spring에서는 해당 서비스를 지원하기 위해 MessageSource라는 것을 사용한다.

*준비 사항*

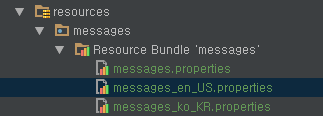
* *JDK 1.6 이상*
* *Gradle 1.11 이상*
* *IntelliJ*

**국제화 서비스를 위한 MessageSource 설정하기**

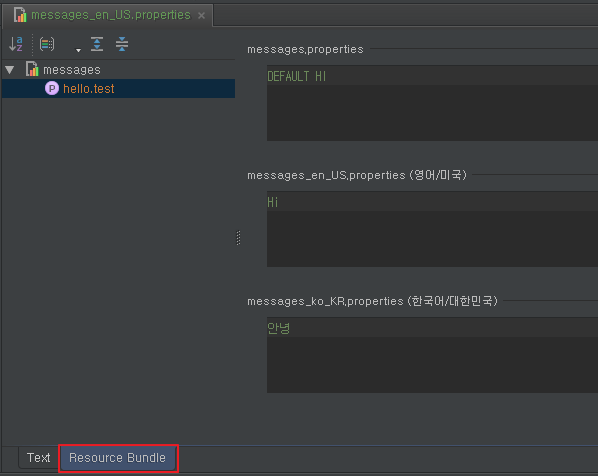
**1. Messages.properties 파일 생성하기**

resources 폴더 밑에 messages.properties 파일을 생성한다.

파일은 다음과 같은 포멧으로 설정 한다. messages\_언어\_나라.properties



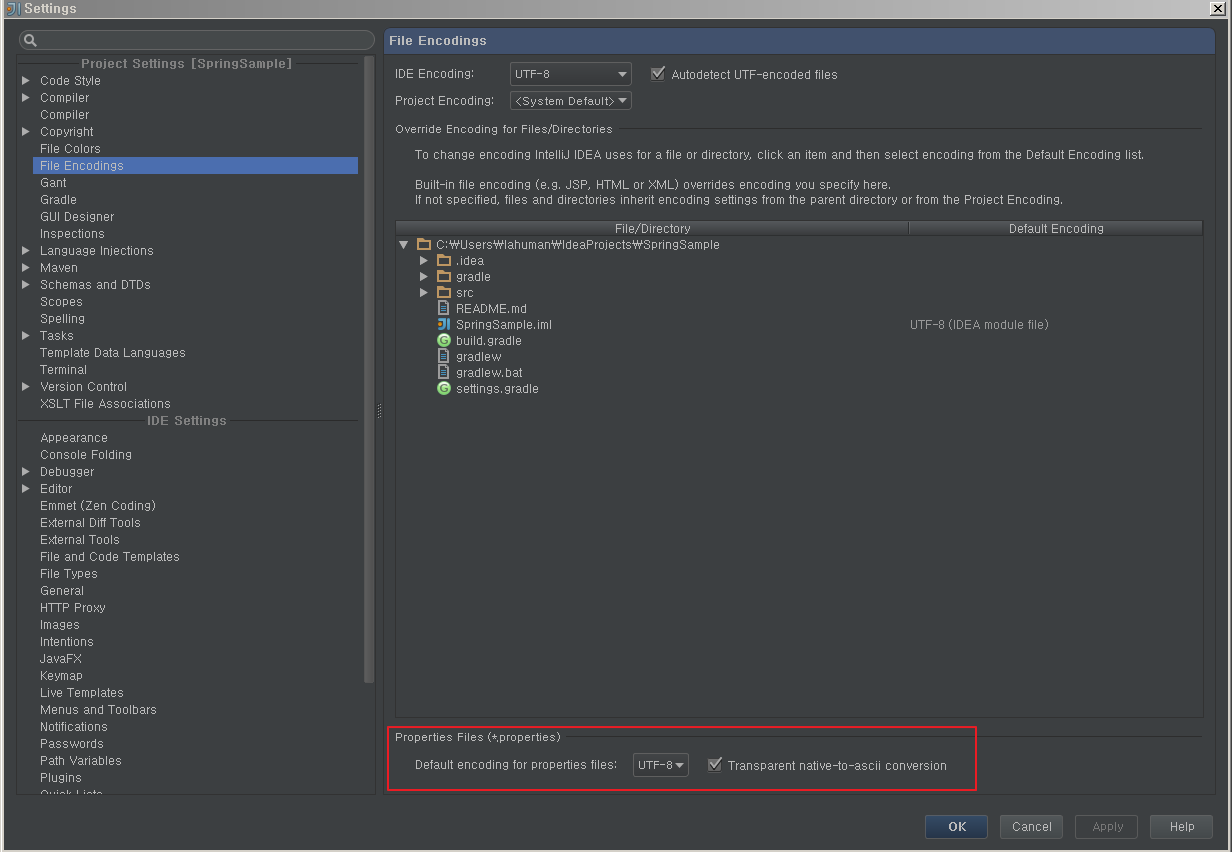
[파일 목록]



[Resource Bundle로 확인시]

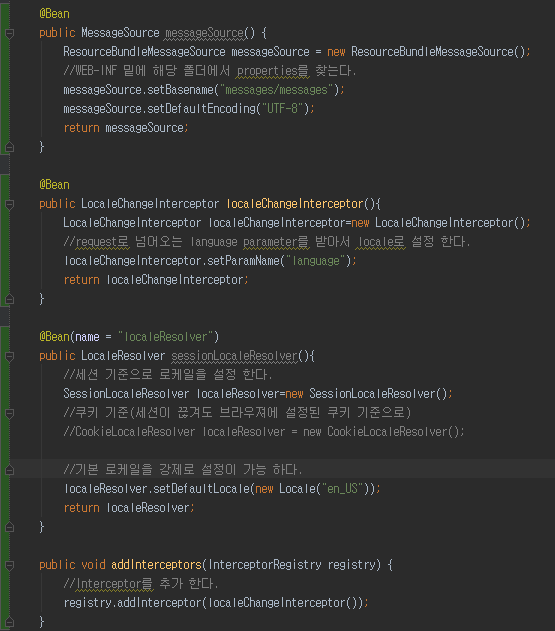
**TIP : Properties 파일에서 한글 사용시, 깨질시 Settings에서 아래와 같이 UTF-8 와 체크박스를 선택한다.**

Eclipse 에서는 플러그인 설치가 필요 하지만 Intellij에서는 기본적으로 제공한다.



**2. Application 에 MessagesSource 설정하기**

WebMvcConfigurerAdapter 상속받은 Application 에서 MessageSource 설정을 진행한다.



|  |
| --- |
| **@Bean**  **public MessageSource messageSource() {**  **ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();**  **//WEB-INF 밑에 해당 폴더에서 properties를 찾는다.**  **messageSource.setBasename("messages/messages");**  **messageSource.setDefaultEncoding("UTF-8");**  **return messageSource;**  **}**    **@Bean**  **public LocaleChangeInterceptor localeChangeInterceptor(){**  **LocaleChangeInterceptor localeChangeInterceptor=new LocaleChangeInterceptor();**  **//request로 넘어오는 language parameter를 받아서 locale로 설정 한다.**  **localeChangeInterceptor.setParamName("language");**  **return localeChangeInterceptor;**  **}**    **@Bean(name = "localeResolver")**  **public LocaleResolver sessionLocaleResolver(){**  **//세션 기준으로 로케일을 설정 한다.**  **SessionLocaleResolver localeResolver=new SessionLocaleResolver();**  **//쿠키 기준(세션이 끊겨도 브라우져에 설정된 쿠키 기준으로)**  **// CookieLocaleResolver localeResolver = new CookieLocaleResolver();**    **//최초 기본 로케일을 강제로 설정이 가능 하다.**  **localeResolver.setDefaultLocale(new Locale("en\_US"));**  **return localeResolver;**  **}**    **public void addInterceptors(InterceptorRegistry registry) {**  **//Interceptor를 추가 한다.**  **registry.addInterceptor(localeChangeInterceptor());**  **}** |

**3. JAVA 또는 JSP에서 사용하기**

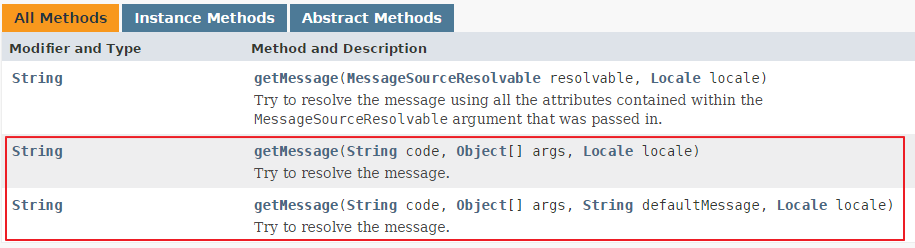
Java에서 MessageSource를 사용하기 위해서는 다음과 같이 MessageSource 를 DI(Dependency Injection) 하여 사용한다.

http://cfile7.uf.tistory.com/image/2551BF395449BC6D199850

[?](http://lahuman.jabsiri.co.kr/87)

|  |  |
| --- | --- |
| 1  2 | @Autowired  private MessageSource messageSource; |

그리고 [MessageSource에서 제공](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/MessageSource.html)되는 다음의 method를 주로 사용한다.



MessageSource Method 인자 설명

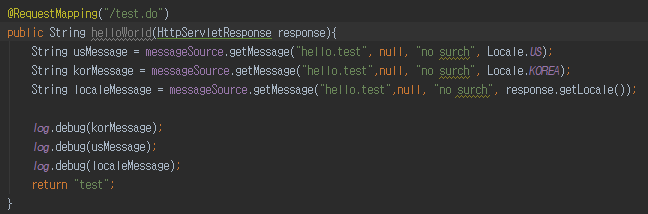
code : properties에서 설정한 Key 값

args : Key에 해당하는 Value값에 {0}, {1} 등으로 매핑되는 값

defaultMessage : Key값을 찾지 못할 경우 표출될 값

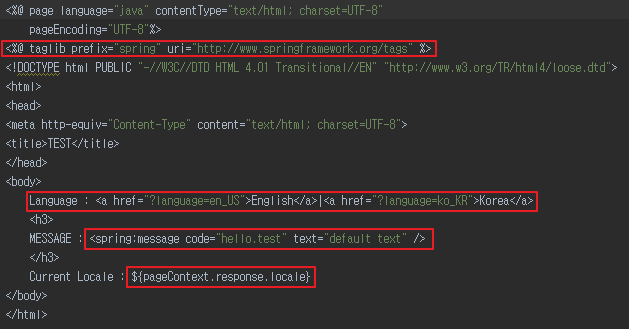
locale : 지역 코드 값

***JAVA 사용 예)***



|  |
| --- |
| ***String usMessage = messageSource.getMessage("hello.test", null, "no surch", Locale.US); //미국 기준 메시지 요청***  ***String korMessage = messageSource.getMessage("hello.test",null, "no surch", Locale.KOREA);//한국 기준 메시지 요청***  ***String localeMessage = messageSource.getMessage("hello.test",null, "no surch", localeResolver.resolveLocale(request)); //현재 설정된 위치 기준의 메시지 요청***  ***//결과 출력***  ***log.debug("KR: " + korMessage);***  ***log.debug("US: " + usMessage);***  ***log.debug("USER SET: " + localeMessage);***  ***log.debug("response SET: " + response.getLocale().toString()); //respons에 설정된 위치 정보 출력(불변)*** |

***JSP 사용 예)***

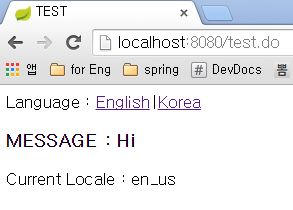


- Spring taglib를 사용하여 MessageSource 를 쉽게 사용할 수 있다

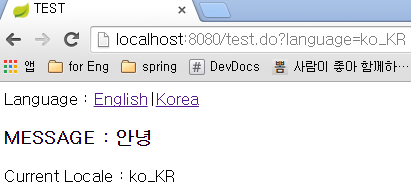
- Parameter에 language=Locale 을 하게 되면, 위치 정보 properties가 Locale에 맞게 출력된다.

**4. 테스트 결과**

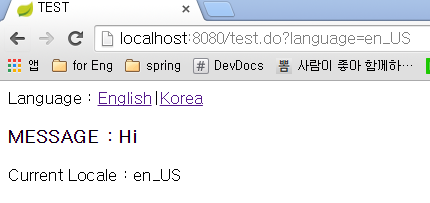
***최초 접속 시)***



***한글 선택 시)***



***영어 선택 시)***



*참고 URL :*

[*https://gist.github.com/marcaurele/3665696*](https://gist.github.com/marcaurele/3665696)

[*http://alex-ilies.blogspot.kr/2013/12/internationalization-spring-mvc-with.html*](http://alex-ilies.blogspot.kr/2013/12/internationalization-spring-mvc-with.html)

먼저 다국어 처리를 위해 메시지를 별도의 파일로 분류해야 한다. 경로를 src/main/resources/locale에 각각 영어, 일본어, 한국어 메시지파일을 만든다.

| **파일명** | **파일내용** |
| --- | --- |
| messages\_en.properties | hello=Hello |
| messages\_ja.properties | hello=こんにちは |
| messages\_ko.properties | hello=안녕하세요 |

그런 다음에 servlet-context.xml에 다국어 처리부분을 추가한다

|  |
| --- |
| <beans:bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  <beans:property name="basenames" value="locale/messages"/>  </beans:bean>  <beans:bean id="localeResolver"  class="org.springframework.web.servlet.i18n.SessionLocaleResolver">  <beans:property name="defaultLocale" value="ko" />  </beans:bean>  <mvc:interceptors>  <beans:bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">  <beans:property name="paramName" value="lang" />  </beans:bean>  </mvc:interceptors> |
| ResourceBundleMessageSource 부분이 바로 위에서 저장된 messages파일들을 읽어오는 부분이다. 해당언어의 파일들이 존재하면 된다. basenames에서 경로/파일명앞부분를 적어준다.  localeResolver에서 다국어처리 방법을 선택한다. localeResolver는 3가지 종류가 있다. 디폴트는 AceeptHeaderLocaleResolver이다. 즉, localeResolver를 별도로 구현하지 않으면 AceeptHeaderLocaleResolver가 브라우저의 설정된 언어값으로 읽어들여 처리한다. 이 값은 변경이 불가능하다. 따라서 사용자가 링크를 통해서 영어, 한국어, 일어등으로 자유롭게 변경하려면 localeResolver를 AceeptHeaderLocaleResolver가 아닌 다른 것으로 선언해야 한다. 반면에 그냥 브라우저에 세팅된 언어로 설정하려면 localeResolver를 없애면 된다.  SessionLocaleResolver request가 가지고 있는 session으로 부터 locale정보를 가져온다. session에 있는 값을 가져오되, 만약 값이 없다면 defaultLocale로 설정을 지정할 수 있다. 아마 맨처음 접근시에는 session값이 없으므로 defaultLocale를 읽게 된다. 위에서는 SessionLocaleResolver를 정의하고 defaultLocale를 ko로 했으므로 브라우저에 세팅된 언어보다 위에 설정된 언어로 표시하게 된다.  CookieLocaleResolver는 이름에서 알수있듯이 언어를 변경하고 쿠키에 값을 저장한다. |

쿠키이용 다국어

|  |
| --- |
| <beans:bean id="localeResolver"  class="org.springframework.web.servlet.i18n.CookieLocaleResolver" >  <beans:property name="cookieName" value="clientlanguage"/>  <beans:property name="cookieMaxAge" value="100000"/>  <beans:property name="cookiePath" value="web/cookie"/>  </beans:bean> |

| **속성** | **기본값** | **설명** |
| --- | --- | --- |
| cookieName | classname+locale | 저장할 쿠키명 |
| cookieMaxAge | -1 | -1은 브라우저를 닫으면 지움 |
| cookiePath | / | 경로를 지정할 수 있음 |

만약 저장된 쿠키값이 없어서 특정값으로 세팅하고자 한다면 defaultLocale를 추가하면 된다.

|  |
| --- |
| <beans:property name="defaultLocale" value="ko" /> |

SessionLocaleResolver나 CookieLocaleResolver에 defaultLocale이 없다면 브라우저의 언어설정을 따르게 된다.

LocaleChangeInterceptor는 사용자가 url를 통해서 언어를 변경할 경우 사용한다. 만약 홈경로가 http://localhost/home이라고 하면 ?lang=언어값을 붙여서 사용한다.

* http://localhost/home/?lang=ko
* http://localhost/home/?lang=en
* http://localhost/home/?lang=ja

그렇게 되면 링크를 통해 언어변환처리를 할 수 있다. <mvc:interceptors></mvc:interceptors>로 감싸면 Spring MVC에서 controller를 통해서 접근시 가로채서 LocaleChangeInterceptor를 적용하게 된다.

servlet-context.xml의 최종소스이다.

|  |
| --- |
| <?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"  xmlns:mvc="http://www.springframework.org/schema/mvc"  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 />  <!-- 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>  <context:component-scan base-package="com.andwise.home" />  <beans:bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">  <beans:property name="basenames" value="locale/messages"/>  </beans:bean>  <beans:bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">  <beans:property name="defaultLocale" value="ko" />  </beans:bean>  <!--  <beans:bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver" >  <beans:property name="defaultLocale" value="ko" />  <beans:property name="cookieName" value="home-lang"/>  <beans:property name="cookieMaxAge" value="-1"/>  <beans:property name="cookiePath" value="/"/>  </beans:bean>  -->  <mvc:interceptors>  <beans:bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">  <beans:property name="paramName" value="lang" />  </beans:bean>  </mvc:interceptors>  </beans:beans> |

Embedded-database

|  |
| --- |
| <jdbc:embedded-database id=*"dataSourceSpied"* type=*"HSQL"*>  <jdbc:script location=*"classpath:/db/script\_hsql.sql"* />  <jdbc:script location=*"classpath:/db/data\_hsql.sql"* />  </jdbc:embedded-database> |
| Database connect (스프링 처음 부팅되면 client 뛰운다 )  <https://www.mkyong.com/spring/spring-view-content-of-hsqldb-embedded-database/> **Spring – View content of HSQLDB embedded database** By [mkyong](https://www.mkyong.com/author/mkyong/) | June 22, 2015 | Viewed : 7,150 times +121 pv/w  A Spring @Configuration example to start an HSQLDB embedded database or in-memory database.  DataSourceConfig.java  package com.mkyong.config.db;  import javax.sql.DataSource;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.jdbc.datasource.embedded.EmbeddedDatabase;  import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;  import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;  @Configuration  public class DataSourceConfig {  @Bean  public DataSource dataSource(){  //jdbc:hsqldb:mem:testdb  EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();  EmbeddedDatabase db = builder.setType(EmbeddedDatabaseType.HSQL)  .addScript("db/hsqldb/db.sql")  .build();  return db;  }  }  Review the Spring log, a “testdb’ embedded database will be created, and you can access via jdbcjdbc:hsqldb:mem:testdb  INFO o.s.j.d.e.EmbeddedDatabaseFactory - Creating embedded database 'testdb'  DEBUG o.s.jdbc.datasource.DataSourceUtils - Fetching JDBC Connection from DataSource  DEBUG o.s.j.d.SimpleDriverDataSource - Creating new JDBC Driver Connection to [jdbc:hsqldb:mem:testdb]  INFO o.s.jdbc.datasource.init.ScriptUtils - Executing SQL script from class path resource [db/hsqldb/db.sql] View content of HSQLDB embedded database Start the DatabaseManagerSwing in the same Spring container.  SpringRootConfig.java  package com.mkyong.config;  import javax.annotation.PostConstruct;  import javax.sql.DataSource;  import org.hsqldb.util.DatabaseManagerSwing;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Import;  import org.springframework.jdbc.core.JdbcTemplate;  import com.mkyong.config.db.DataSourceConfig;  @Configuration  @Import({DataSourceConfig.class})  @ComponentScan({ "com.mkyong" })  public class SpringRootConfig {  @Autowired  DataSource dataSource;  @Bean  public JdbcTemplate getJdbcTemplate(){  return new JdbcTemplate(dataSource);  }  //default username : sa, password : ''  **@PostConstruct**  **public void getDbManager(){**  **DatabaseManagerSwing.main(**  **new String[] { "--url", "jdbc:hsqldb:mem:testdb", "--user", "sa", "--password", ""});**  **}**  }  When the Spring container is started, a Swing HSQL database manager will be prompted.  hsql-database-manager  If you start the DatabaseManagerSwing via a command prompt or terminal, the Swing HSQL database manager will still be prompted, but it is UNABLE to connect to the embedded database that started by Spring, because both are different JVM.  java -cp hsqldb-2.3.2.jar org.hsqldb.util.DatabaseManagerSwing  (adsbygoogle=window.adsbygoogle||[]).push({}); |

viewResolver

|  |
| --- |
| <!-- view Resolver Setting tiles3  <bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">  <property name="definitions">  <value>/WEB-INF/config/tiles/tiles-layouts.xml</value>  </property>  </bean>  <bean class="org.springframework.web.servlet.view.UrlBasedViewResolver" p:order="1"  p:viewClass="org.springframework.web.servlet.view.tiles3.TilesView"/>  -->      <!-- view Resolver Setting UrlBasedViewResolver-->  <bean class=*"org.springframework.web.servlet.view.UrlBasedViewResolver"* p:order=*"2"*  p:viewClass=*"org.springframework.web.servlet.view.JstlView"*  p:prefix=*"/WEB-INF/jsp/"* p:suffix=*".jsp"*/> |

Aop

<http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte:fdl:aop:aspectj>

# @AspectJ 어노테이션을 이용한 AOP 지원

## 개요

@AspectJ는 Java 5 어노테이션을 사용한 일반 Java 클래스로 관점(Aspect)를 정의하는 방식이다. @AspectJ 방식은 AspectJ 5 버전에서 소개되었으며, Spring은 2.0 버전부터 AspectJ 5 어노테이션을 지원한다. Spring AOP 실행환경은 AspectJ 컴파일러나 직조기(Weaver)에 대한 의존성이 없이 @AspectJ 어노테이션을 지원한다.

## 설명

### @AspectJ 설정하기

@AspectJ를 사용하기 위해서 다음 코드를 Spring 설정에 추가한다.

<aop:aspectj-autoproxy/>

### 관점(Aspect) 정의하기

클래스에 @Aspect 어노테이션을 추가하여 Aspect를 생성한다. @Aspect 설정이 되어 있는 경우 Spring은 자동적으로 @Aspect 어노테이션을 포함한 클래스를 검색하여 Spring AOP 설정에 반영한다.

import *org.aspectj.lang.annotation.Aspect*;

@Aspect

public class AspectUsingAnnotation {

..

}

### 포인트컷(Pointcut) 정의하기

포인트컷은 결합점(Join points)을 지정하여 충고(Advice)가 언제 실행될지를 지정하는데 사용된다. Spring AOP는 Spring 빈에 대한 메소드 실행 결합점만을 지원하므로, Spring에서 포인트컷은 빈의 메소드 실행점을 지정하는 것으로 생각할 수 있다.

다음 예제는 egovframework.rte.fdl.aop.sample 패키지 하위의 Sample 명으로 끝나는 클래스의 모든 메소드 수행과 일치할 'targetMethod' 라는 이름의 pointcut을 정의한다.

@Aspect

public class AspectUsingAnnotation {

...

@Pointcut("execution(public \* egovframework.rte.fdl.aop.sample.\*Sample.\*(..))")

public void targetMethod() {

*// pointcut annotation 값을 참조하기 위한 dummy method*

}

...

}

#### 포인트컷 지정자(Designators)

Spring에서 포인트컷 표현식에 사용될 수 있는 지정자는 다음과 같다. 포인트컷은 모두 public 메소드를 대상으로 한다.

* execution: 메소드 실행 결합점(join points)과 일치시키는데 사용된다.
* within: 특정 타입에 속하는 결합점을 정의한다.
* this: 빈 참조가 주어진 타입의 인스턴스를 갖는 결합점을 정의한다.
* target: 대상 객체가 주어진 타입을 갖는 결합점을 정의한다.
* args: 인자가 주어진 타입의 인스턴스인 결합점을 정의한다.
* @target: 수행중인 객체의 클래스가 주어진 타입의 어노테이션을 갖는 결합점을 정의한다.
* @args: 전달된 인자의 런타입 타입이 주어진 타입의 어노테이션을 갖는 결합점을 정의한다.
* @within: 주어진 어노테이션을 갖는 타입 내 결합점을 정의한다.
* @annotation: 결합점의 대상 객체가 주어진 어노테이션을 갖는 결합점을 정의한다.

#### 포인트컷 표현식 조합하기

포인트컷 표현식은 '&&', '||' 그리고 '!' 를 사용하여 조합할 수 있다.

@Pointcut("execution(public \* \*(..))")

private void anyPublicOperation() {}

@Pointcut("within(com.xyz.someapp.trading..\*)")

private void inTrading() {}

@Pointcut("anyPublicOperation() && inTrading()")

private void tradingOperation() {}

#### 포인트컷 정의 예제

Spring AOP에서 자주 사용되는 포인트컷 표현식의 예를 살펴본다.

|  |  |
| --- | --- |
| **Pointcut** | **선택된 Joinpoints** |
| execution(public \* \*(..)) | public 메소드 실행 |
| execution(\* set\*(..)) | 이름이 set으로 시작하는 모든 메소드명 실행 |
| execution(\* set\*(..)) | 이름이 set으로 시작하는 모든 메소드명 실행 |
| execution(\* com.xyz.service.AccountService.\*(..)) | AccountService 인터페이스의 모든 메소드 실행 |
| execution(\* com.xyz.service.\*.\*(..)) | service 패키지의 모든 메소드 실행 |
| execution(\* com.xyz.service..\*.\*(..)) | service 패키지와 하위 패키지의 모든 메소드 실행 |
| within(com.xyz.service.\*) | service 패키지 내의 모든 결합점 |
| within(com.xyz.service..\*) | service 패키지 및 하위 패키지의 모든 결합점 |
| this(com.xyz.service.AccountService) | AccountService 인터페이스를 구현하는 프록시 개체의 모든 결합점 |
| target(com.xyz.service.AccountService) | AccountService 인터페이스를 구현하는 대상 객체의 모든 결합점 |
| args(java.io.Serializable) | 하나의 파라미터를 갖고 전달된 인자가 Serializable인 모든 결합점 |
| @target(org.springframework.transaction.annotation.Transactional) | 대상 객체가 @Transactional 어노테이션을 갖는 모든 결합점 |
| @within(org.springframework.transaction.annotation.Transactional) | 대상 객체의 선언 타입이 @Transactional 어노테이션을 갖는 모든 결합점 |
| @annotation(org.springframework.transaction.annotation.Transactional) | 실행 메소드가 @Transactional 어노테이션을 갖는 모든 결합점 |
| @args(com.xyz.security.Classified) | 단일 파라미터를 받고, 전달된 인자 타입이 @Classified 어노테이션을 갖는 모든 결합점 |
| bean(accountRepository) | “accountRepository” 빈 |
| !bean(accountRepository) | “accountRepository” 빈을 제외한 모든 빈 |
| bean(\*) | 모든 빈 |
| bean(account\*) | 이름이 'account'로 시작되는 모든 빈 |
| bean(\*Repository) | 이름이 “Repository”로 끝나는 모든 빈 |
| bean(accounting/\*) | 이름이 “accounting/“로 시작하는 모든 빈 |
| bean(\*dataSource) || bean(\*DataSource) | 이름이 “dataSource” 나 “DataSource” 으로 끝나는 모든 빈 |

### 충고(Advice) 정의하기

충고(Advice)는 관점(Aspect)의 실제 구현체로 포인트컷 표현식과 일치하는 결합점에 삽입되어 동작할 수 있는 코드이다. 충고는 결합점과 결합하여 동작하는 시점에 따라 before advice, after advice, around advice 타입으로 구분된다.

#### Before advice

Before advice는 @Before 어노테이션을 사용한다.

다음은 Before 충고를 사용하는 예제이다. Before 충고인 beforeTargetMethod() 메소드는 targetMethod()로 정의된 포인트컷 전에 수행된다.

@Aspect

public class AspectUsingAnnotation {

..

@Before("targetMethod()")

public void beforeTargetMethod(JoinPoint thisJoinPoint) {

Class clazz = thisJoinPoint.getTarget().getClass();

[String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) className = thisJoinPoint.getTarget().getClass().getSimpleName();

[String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) methodName = thisJoinPoint.getSignature().getName();

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.beforeTargetMethod executed.");

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println(className + "." + methodName + " executed.");

}

}

#### After returning advice

After returing 충고는 정상적으로 메소드가 실행될 때 수행된다. After returning 충고는 @AfterReturing 어노테이션을 사용한다.

다음은 After returning 충고를 사용하는 예제이다. afterReturningTargetMethod() 충고는 targetMethod()로 정의된 포인트컷 후에 수행된다. targetMethod() 포인트컷의 실행 결과는 retVal 변수에 저장되어 전달된다.

@Aspect

public class AspectUsingAnnotation {

..

@AfterReturning(pointcut = "targetMethod()", returning = "retVal")

public void afterReturningTargetMethod(JoinPoint thisJoinPoint,

[Object](http://www.google.com/search?hl=en&q=allinurl%3Aobject+java.sun.com&btnI=I%27m%20Feeling%20Lucky) retVal) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.afterReturningTargetMethod executed." +

" return value is [" + retVal + "]");

}

}

#### After throwing advice

After throwing 충고는 메소드가 수행 중 예외사항을 반환하고 종료하는 경우 수행된다. After throwing 충고는 @AfterThrowing 어노테이션을 사용한다.

다음은 After throwing 충고를 사용하는 예제이다. afterThrowingTargetMethod() 충고는 targetMethod()로 정의된 포인트컷에서 예외가 발생한 후에 수행된다. targetMethod() 포인트컷에서 발생된 예외는 exception 변수에 저장되어 전달된다. 예제에서는 전달 받은 예외를 한번 더 감싸서 사용자가 쉽게 알아 볼 수 있도록 메시지를 설정하여 반환한다.

@Aspect

public class AspectUsingAnnotation {

..

@AfterThrowing(pointcut = "targetMethod()", throwing = "exception")

public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,

[Exception](http://www.google.com/search?hl=en&q=allinurl%3Aexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) exception) throws [Exception](http://www.google.com/search?hl=en&q=allinurl%3Aexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.afterThrowingTargetMethod executed.");

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("에러가 발생했습니다.", exception);

throw new BizException("에러가 발생했습니다.", exception);

}

}

#### After (finally) advice

After (finally) 충고는 메소드 수행 후 무조건 수행된다. After (finally) 충고는 @After 어노테이션을 사용한다. After 충고는 정상 종료와 예외 발생 경우를 모두 처리해야 하는 경우에 사용된다. 리소스 해제와 같은 작업이 해당된다.

다음은 After (finally) 충고를 사용하는 예제이다. afterTargetMethod() 충고는 targetMethod()로 정의된 포인트컷 이후에 수행된다.

@Aspect

public class AspectUsingAnnotation {

..

@After("targetMethod()")

public void afterTargetMethod(JoinPoint thisJoinPoint) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.afterTargetMethod executed.");

}

}

#### Around advice

Around 충고는 메소드 수행 전후에 수행된다. Around 충고는 @Around 어노테이션을 사용한다.

다음은 Around 충고를 사용하는 예제이다. aroundTargetMethod() 충고는 파라미터로 ProceedingJoinPoint을 전달하며 proceed() 메소드 호출을 통해 대상 포인트컷을 실행한다. 포인트컷 수행 결과값인 retVal을 Around 충고 내에서 변환하여 반환할 수 있음을 보여준다.

@Aspect

public class AspectUsingAnnotation {

..

@Around("targetMethod()")

public [Object](http://www.google.com/search?hl=en&q=allinurl%3Aobject+java.sun.com&btnI=I%27m%20Feeling%20Lucky) aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)

throws [Throwable](http://www.google.com/search?hl=en&q=allinurl%3Athrowable+java.sun.com&btnI=I%27m%20Feeling%20Lucky) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.aroundTargetMethod start.");

long time1 = [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).currentTimeMillis();

[Object](http://www.google.com/search?hl=en&q=allinurl%3Aobject+java.sun.com&btnI=I%27m%20Feeling%20Lucky) retVal = thisJoinPoint.proceed();

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("ProceedingJoinPoint executed. return value is [" + retVal + "]");

retVal = retVal + "(modified)";

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("return value modified to [" + retVal + "]");

long time2 = [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).currentTimeMillis();

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("AspectUsingAnnotation.aroundTargetMethod end. Time(" + (time2 - time1) + ")");

return retVal;

}

}

### 관점(Aspect) 실행하기

앞서 정의한 관점(Aspect)가 정상적으로 동작하는지 확인하기 위해 테스트 코드를 이용해 확인해 본다. AnnotationAspectTest 클래스는 대상 메소드 수행시 예외없이 정상 실행하는 경우와 예외 발생의 경우를 구분해서 테스트 한다.

#### 정상 실행의 경우

testAnnotationAspect() 함수는 대상 메소드가 정상 수행되는 사례를 보여준다. egovframework.rte.fdl.aop.sample 패키지에 속하는 AnnotationAdviceSample 클래스의 someMethod() 메소드는 before, after returning, after finally, around 충고(Advice)가 적용된다.

public class AnnotationAspectTest {

@Resource(name = "annotationAdviceSample")

AnnotationAdviceSample annotationAdviceSample;

@Test

public void testAnnotationAspect() throws [Exception](http://www.google.com/search?hl=en&q=allinurl%3Aexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) {

SampleVO vo = new SampleVO();

..

[String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) resultStr = annotationAdviceSample.someMethod(vo);

assertEquals("someMethod executed.(modified)", resultStr);

}

}

테스트 코드를 수행한 결과 로그는 다음과 같다.

AspectUsingAnnotation.beforeTargetMethod executed.

AspectUsingAnnotation.aroundTargetMethod start.

ProceedingJoinPoint executed. return value is [someMethod executed.]

return value modified to [someMethod executed.(modified)]

AspectUsingAnnotation.aroundTargetMethod end. Time(78)

AspectUsingAnnotation.afterTargetMethod executed.

AspectUsingAnnotation.afterReturningTargetMethod executed. return value is [someMethod executed.(modified)]

콘솔 로그 출력을 보면 충고(Advice)가 적용되는 순서는 다음과 같다.

* @Before
* @Around (대상 메소드 수행 전)
* 대상 메소드
* @Around (대상 메소드 수행 후)
* @After(finally)
* @AfterReturning

주의할 점은 @Around 충고는 대상 메소드의 반환 값(return value)를 변경 가능하지만, After returning 충고는 반환 값을 참조 가능하지만 변경할 수 없다.

#### 예외 발생의 경우

testAnnotationAspectWithException() 함수는 대상 메소드에 오류가 발생한 사례를 보여준다. egovframework.rte.fdl.aop.sample 패키지에 속하는 AnnotationAdviceSample 클래스의 someMethod() 메소드는 before, after throwing, after finally, around 충고(Advice)가 적용된다.

public class AnnotationAspectTest {

@Resource(name = "annotationAdviceSample")

AnnotationAdviceSample annotationAdviceSample;

@Test

public void testAnnotationAspectWithException() throws [Exception](http://www.google.com/search?hl=en&q=allinurl%3Aexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) {

SampleVO vo = new SampleVO();

*// exception 을 발생시키도록 플래그 설정*

vo.setForceException(**true**);

..

try {

*// vo 의 forceException 플래그가 true 이면 - / by zero 상황을 강제로 처리함*

resultStr = annotationAdviceSample.someMethod(vo);

fail("exception 을 강제로 발생시켜 이 라인이 수행될 수 없습니다.");

} catch ([Exception](http://www.google.com/search?hl=en&q=allinurl%3Aexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) e) {

..

}

}

}

테스트 코드를 수행한 결과 로그는 다음과 같다.

AspectUsingAnnotation.beforeTargetMethod executed.

AspectUsingAnnotation.aroundTargetMethod start.

AspectUsingAnnotation.afterTargetMethod executed.

AspectUsingAnnotation.afterThrowingTargetMethod executed.

에러가 발생했습니다.

java.lang.ArithmeticException: / by zero

...

콘솔 로그 출력을 보면 충고(Advice)가 적용되는 순서는 다음과 같다.

* @Before
* @Around (대상 메소드 수행 전)
* 대상 메소드 (ArithmeticException 예외가 발생한다)
* @After(finally)
* @AfterThrowing

예외가 발생하더라도 after 로 정의한 충고(Advice)는 수행되는 것을 확인할 수 있다. After Throwing 충고(Advice)는 에러 메시지를 재설정하고 새로운 예외를 생성하여 전달할 수 있다.

## 참고자료

* [Spring 2.5 Reference Documentation](http://static.springframework.org/spring/docs/2.5.x/reference/aop.html)

|  |
| --- |
| @AspectJ를 사용하기 위해서 다음 코드를 Spring 설정에 추가한다.  <aop:aspectj-autoproxy/>  <!-- aop 사용하지않음  <bean id="PrivacyAOP" class="com.ko.omnicns.omnifit.security.encrypt.PrivacyAOP"/>  <bean id="UserService" class="com.ko.omnicns.omnifit.cms.user.service.UserServiceImpl"/>  <aop:aspectj-autoproxy />  --> |
| @Aspect  **public** **class** ~~PrivacyAOP~~ {  // @Pointcut("execution(public \* com.ko.omnicns.omnifit.cms.user.service..\*Impl.\*(..))")  **public** PrivacyAOP() {  System.***out***.println("--PrivacyAOP-");  }  //com.ko.omnicns.omnifit.cms.sample  // com.ko.omnicns.omnifit.cms.sample.service.CMSSampleServiceImpl  // @Before("execution(\* com.ko.omnicns.omnifit.cms.sample.service..\*Impl..\*(..))")  // @Before("execution(public \* com.ko.omnicns.omnifit.cms.sample.service..\*Impl..\*(..))")  // @Before("execution(public \* com.ko.omnicns.omnifit.cms.user.service.UserMapper..\*(..))")  // com.ko.omnicns.omnifit.cms.user.service.selectUserPermission(..)  // @Before("execution(public \* com.ko.omnicns.omnifit.cms..\*.service..\*..\*(..))")  // @Before("execution(public \* com.ko.omnicns.omnifit.cms..\*.service..\*Impl..\*(..))")      // @Before("execution(\* com.ko.omnicns.omnifit.cms.user.service.UserServiceImpl..\*(..))")    // @Before("execution(\* com.ko.omnicns.omnifit.cms.sample.CMSSampleController..\*(..))")  // @Before("execution(\* com.ko.omnicns.omnifit.cms.user.UserController..\*(..))")  //execution(\* com.ko.omnicns.omnifit.\*\*.\*Impl.\*(..))  //com.ko.omnicns.omnifit.cms.\*\*.service..\*Impl..\*(..)  // @Before("execution(com.ko.omnicns.omnifit.cms.\*\*.service..\*Impl.\*(..)")  // @Before("execution(public \* com.ko.omnicns.omnifit.cms.\*\*.service..\*Impl.\*(..))")  **public** **void** trace(JoinPoint joinPoint) **throws** Throwable {  System.***out***.println("--PrivacyAOP-"+joinPoint);  }  } |

Global Exception Handler

|  |
| --- |
| <!-- global Exception -->  <bean class=*"c*  @ControllerAdvice("com.ko.omnicns.omnifit.cms")  **public** **class** CustomException {  @ResponseStatus(value=HttpStatus.***NOT\_FOUND***, reason="IOException occured")  @ExceptionHandler(RuntimeException.**class**)  **public** String handlerRuntimeException(){  **return** "cmmn/exception";  }  }  *om.ko.omnicns.omnifit.comm.exception.CustomException"* /> |
| http://www.journaldev.com/2651/spring-mvc-exception-handling-exceptionhandler-controlleradvice-handlerexceptionresolver-json-response-example |
| @ControllerAdvice("com.ko.omnicns.omnifit.cms")  **public** **class** CustomException {      @ExceptionHandler(ErrosException.**class**)  @ResponseBody  **public** String errosexception(ErrosException errors){  **return** **new** JsonVO(errors.getErrors()).toJson();  }      @ExceptionHandler(Exception.**class**)  @ResponseBody  **public** ResponseEntity<?> handleAnyException(Exception e) {  **return** errorResponse(e, HttpStatus.***INTERNAL\_SERVER\_ERROR***);  }  **protected** ResponseEntity<String> errorResponse(  Throwable throwable, HttpStatus status) {  **if** (**null** != throwable) {  **return** response(**new** JsonVO(JsonHead.***CODE\_ERROR***,throwable.getMessage()).toJson(), status);  } **else** {  **return** response(**null**, status);  }  }  **protected** <T> ResponseEntity<T> response(T body, HttpStatus status) {  **return** **new** ResponseEntity<T>(body, **new** HttpHeaders(), status);  }  } |
| Controller  @RequestMapping(value="/indexTest2", method = {RequestMethod.***GET***, RequestMethod.***POST***},produces = "application/json; charset=utf8")  @ResponseBody  **public** String indexTest2(@Valid @ModelAttribute("FromToVO") FromToVO ft,Errors erros, HttpServletRequest request, ModelMap model) **throws** Exception {  log.debug(ft.getFrom());  log.debug(ft.getTo());  **if**(erros.hasErrors()){  **throw** **new** ErrosException(erros);  }  log.debug("1");  **int** x = 0;  log.debug("2"+x);  **int** y = 1;  log.debug("3"+y);  **int** z = y / x;  log.debug("4"+z);    **return** "{}";  } |
| 통합  @ControllerAdvice("com.ko.omnicns.omnifit")  **public** **class** OmniCNSExceptionHandler **extends** ResponseEntityExceptionHandler {    Logger log = LoggerFactory.*getLogger*(**this**.getClass());    @ExceptionHandler(InvalidRequestRuntimeException.**class**)  @ResponseBody  **protected** String handleInvalidRequest(RuntimeException e, WebRequest request) {  InvalidRequestRuntimeException ire = (InvalidRequestRuntimeException) e;  List<FieldErrorResource> fieldErrorResources = **new** ArrayList<😠);    List<FieldError> fieldErrors = ire.getErrors().getFieldErrors();  **for** (FieldError fieldError : fieldErrors) {  FieldErrorResource fieldErrorResource = **new** FieldErrorResource();  fieldErrorResource.setResource(fieldError.getObjectName());  fieldErrorResource.setField(fieldError.getField());  fieldErrorResource.setCode(fieldError.getCode());  fieldErrorResource.setMessage(fieldError.getDefaultMessage());  fieldErrorResources.add(fieldErrorResource);  }  ErrorResource error = **new** ErrorResource("InvalidRequest", ire.getMessage());    error.setFieldErrors(fieldErrorResources);  HttpHeaders headers = **new** HttpHeaders();  headers.setContentType(MediaType.APPLICATION\_JSON);  log.debug("===========================================");  log.debug("접근 RuntimeException");  log.debug("===========================================");  //handleExceptionInternal(e, error, headers, HttpStatus.UNPROCESSABLE\_ENTITY, request); //e.getErrCode()  //"name : " +e.getClass().getSimpleName()+" , code : "+e.hashCode()+" , message : "+e.getMessage();  **return** "\"aa\":\"ok\"";  }    @ExceptionHandler(InvalidRequestException.**class**)  @ResponseBody  **protected** String handleInvalidRequest(Exception e, WebRequest request) {  InvalidRequestException ire = (InvalidRequestException) e;  List<FieldErrorResource> fieldErrorResources = **new** ArrayList<😠);    List<FieldError> fieldErrors = ire.getErrors().getFieldErrors();  **for** (FieldError fieldError : fieldErrors) {  FieldErrorResource fieldErrorResource = **new** FieldErrorResource();  fieldErrorResource.setResource(fieldError.getObjectName());  fieldErrorResource.setField(fieldError.getField());  fieldErrorResource.setCode(fieldError.getCode());  fieldErrorResource.setMessage(fieldError.getDefaultMessage());  fieldErrorResources.add(fieldErrorResource);  }  ErrorResource error = **new** ErrorResource("InvalidRequest", ire.getMessage());    error.setFieldErrors(fieldErrorResources);  HttpHeaders headers = **new** HttpHeaders();  headers.setContentType(MediaType.APPLICATION\_JSON);  log.debug("===========================================");  log.debug("접근 Exception ");  log.debug("===========================================");    //handleExceptionInternal(e, error, headers, HttpStatus.UNPROCESSABLE\_ENTITY, request);  //"name : " +e.getClass().getSimpleName()+" , code : "+e.hashCode()+" , message : "+e.getMessage();  **return** "\"aa\":\"ok2\"";  }  /\*protected Logger logger;  public OmniCNSExceptionHandler() {  logger = LoggerFactory.getLogger(getClass());  }  @ResponseBody  public ResponseEntity<?> handleUnauthenticationException(Exception e) {  return errorResponse(e, HttpStatus.BAD\_REQUEST);  }  @ExceptionHandler({ DataIntegrityViolationException.class,  SQLIntegrityConstraintViolationException.class })  @ResponseBody  public ResponseEntity<?> handleConflictException(Exception e) {  return errorResponse(e, HttpStatus.CONFLICT);  }  @ExceptionHandler({ SQLException.class, DataAccessException.class,  RuntimeException.class })  @ResponseBody  public ResponseEntity<?> handleSQLException(Exception e) {  return errorResponse(e, HttpStatus.INTERNAL\_SERVER\_ERROR);  }  @ExceptionHandler({ InvalidKeyException.class,  NoSuchAlgorithmException.class })  @ResponseBody  public ResponseEntity<?> handleHashException(Exception e) {  return errorResponse(new Exception("Encrypt/Decrypt key is requested"),  HttpStatus.LOCKED);  }  @ExceptionHandler({ Exception.class })  @ResponseBody  public ResponseEntity<?> handleAnyException(Exception e) {  return errorResponse(e, HttpStatus.INTERNAL\_SERVER\_ERROR);  }  protected ResponseEntity<String> errorResponse(  Throwable throwable, HttpStatus status) {  if (null != throwable) {  return response(new ExceptionMessage(throwable), status);  } else {  return response(null, status);  }  }  protected <T> ResponseEntity<T> response(T body, HttpStatus status) {  return new ResponseEntity<T😠body, new HttpHeaders(), status);  }\*/ |

DI

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:c=*"http://www.springframework.org/schema/c"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd"*>  <bean id=*"user1"* class=*"net.madvirus.spring4.chap02.User"*  c:id=*"bkchoi"* c:password=*"1234"* />    <bean id=*"user2"* class=*"net.madvirus.spring4.chap02.User"*  c:\_0=*"madvirus"* c:\_1=*"qwer"* />  <bean id=*"pwChangeSvc"*  class=*"net.madvirus.spring4.chap02.PasswordChangeService"*  c:userRepository-ref=*"userRepository"* />    <bean id=*"authFailLogger"* class=*"net.madvirus.spring4.chap02.AuthFailLogger"*  p:threshold=*"2"* />  <bean id=*"authenticationService"*  class=*"net.madvirus.spring4.chap02.AuthenticationService"*  p:failLogger-ref=*"authFailLogger"*  p:userRepository-ref=*"userRepository"* />    <bean id=*"userRepository"* class=*"net.madvirus.spring4.chap02.UserRepository"*>  <property name=*"users"*>  <list>  <ref bean=*"user1"* />  <ref bean=*"user2"* />  </list>  </property>  </bean>  </beans> |
| AOP 완전체  **public** **class** CMSAOP {  Logger log = LoggerFactory.*getLogger*(**this**.getClass());    // @Pointcut("execution(public \* com.ko.omnicns.omnifit.cms.user.service..\*Impl.\*(..))")  **public** CMSAOP() {  }  //execution(\* com.ko.omnicns.omnifit.\*\*.\*Impl.\*(..))  //com.ko.omnicns.omnifit.cms.\*\*.service..\*Impl..\*(..)  // @Before("execution(com.ko.omnicns.omnifit.cms.\*\*.service..\*Impl.\*(..)")  // @Around("execution(public \* com.ko.omnicns.omnifit.cms..\*Controller.\*(..))")  // public Object before(ProceedingJoinPoint joinPoint) throws Throwable {  // System.out.println("AspectUsingAnnotation.aroundTargetMethod start.");  // long time1 = System.currentTimeMillis();  // Object retVal = joinPoint.proceed();  //  // System.out.println("ProceedingJoinPoint executed. return value is [" + retVal + "]");  //  // retVal = retVal + "(modified)";  // System.out.println("return value modified to [" + retVal + "]");  //  // long time2 = System.currentTimeMillis();  // System.out.println("AspectUsingAnnotation.aroundTargetMethod end. Time(" + (time2 - time1) + ")");  // return retVal;  // }      @Before("execution(public \* com.ko.omnicns.omnifit.cms..\*Controller.\*(..))")  **public** **void** before(JoinPoint joinPoint) **throws** Throwable {  Object[] args = AOPUtil.*getParamter*(joinPoint);  Class returnType = AOPUtil.*getMethodReturnType*(joinPoint);  **boolean** isJson = **false**;  **for** (**int** i = 0; i < args.length; i++) {  **if**(**null**!=args[i] && Json.**class**.isAssignableFrom(args[i].getClass()) && Void.***TYPE***==returnType){  isJson=**true**;  }  }  // ResponseBody body = AOPUtil.getMethodAnnotation(joinPoint,ResponseBody.class);  **for** (Object signatureArg: args) {  **if**(**null**!=signatureArg && DefaultVO.**class**.isAssignableFrom(signatureArg.getClass())){  DefaultVO dv = (DefaultVO)signatureArg;  BindingResult bind = dv.getErrors();  **if**(**null**!=bind && bind.hasErrors()){  **if**(isJson){ //json  **throw** **new** JsonErrorsException(bind);  }**else**{//JSP  **throw** **new** JspErrorsException(bind);  }  }  }  }  }                @AfterReturning(pointcut="execution(public \* com.ko.omnicns.omnifit.cms..\*Controller.\*(..))",returning = "retVal")  **public** **void** after(JoinPoint joinPoint, Object retVal) **throws** Throwable {        Object[] args = AOPUtil.*getParamter*(joinPoint);  **boolean** isJson = **false**;  **for** (**int** i = 0; i < args.length; i++) {  **if**(**null**!=args[i] && Json.**class**.isAssignableFrom(args[i].getClass())){  isJson=**true**;  }  }        MethodSignature methodSignature = (MethodSignature) joinPoint.getStaticPart().getSignature();  Method method = methodSignature.getMethod();  Annotation[] methodAnnotations = method.getAnnotations();  ResponseBody body = method.getAnnotation(ResponseBody.**class**);  Annotation[][] parameterAnnotations = method.getParameterAnnotations();        **for** (Object signatureArg: args) {  **if**(**null**!=signatureArg && ModelMap.**class**.isAssignableFrom(signatureArg.getClass())){  **final** ModelMap modelMap = (ModelMap)signatureArg;  ConvertUtil.*convertValueSet*(modelMap, **new** Convertor() {  @Override  **public** Object converting(Object injection) **throws** Exception {  **if**(injection!=**null** && !modelMap.equals(injection)&&  (  DefaultVO.**class**.isAssignableFrom(injection.getClass())||  Collection.**class**.isAssignableFrom(injection.getClass())||  Map.**class**.isAssignableFrom(injection.getClass())||  injection.getClass().isArray()  )){  SecurityManager.*getInstance*().privacyAnnoConvertByDefaultVO(injection);  }  **return** injection;  }  });  }    **if**(**null**!=signatureArg && **null**==retVal && Json.**class**.isAssignableFrom(signatureArg.getClass())){  Json json = (Json)signatureArg;  **if**(**null**!=json.getBody()){  SecurityManager.*getInstance*().privacyAnnoConvertByDefaultVO(json.getBody());  }  // HttpServletRequest request = RequestUtil.getRequest();  // HttpServletResponse response = ResponseUtil.getResponse();  HttpServletResponse response = Content.*getResponse*();  response.setContentType("application/json; charset=utf8");  // response.getOutputStream().print(json.toJson());  response.getOutputStream().write(json.toJson().getBytes());  response.getOutputStream().close();  }  }    } |

트랜잭션

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd"*>  <bean id=*"txManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  </bean>  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"*>  <tx:attributes>  <tx:method name=*"\*"* rollback-for=*"Exception"*/>  </tx:attributes>  </tx:advice>  <aop:config>  <aop:pointcut id=*"requiredTx"* expression=*"execution(\* com.ko.omnicns.omnifit.api.\*\*.Service.\*(..))"*/>  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"requiredTx"* />  </aop:config>  </beans> |

트랜젝션

|  |
| --- |
| @Autowired  **private** DataSourceTransactionManager transactionManager;  public Object updateDownwardCnt(ExerMgmtVO exerMgmtVO) throws Exception{                  DefaultTransactionDefinition def = new DefaultTransactionDefinition();        def.setName("transaction");        def.setPropagationBehavior(TransactionDefinition.PROPAGATION\_REQUIRED);                TransactionStatus status = transactionManager.getTransaction(def);                  boolean tf = false;                  //하향조정횟수 업데이트         if(exerMgmtDao.updateDownwardCnt(exerMgmtVO) > 0) {                              tf = true;                  }                  //목표심박수 재계산 update         try {                          String mbrId = exerMgmtVO.getMbrId();                          ExerMgmtVO vo = new ExerMgmtVO();             vo.setMbrId(mbrId);                          //회원 나이, 성별, 운동단계 조회             ExerMgmtVO baseList = selectCalcCondition(vo);             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.baseList.cardioYn:==" + baseList.getCardioYn());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.baseList.age:==" + baseList.getAge());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.baseList.sex:==" + baseList.getSex());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.baseList.exerStep:==" + baseList.getExerStep());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.baseList.dnwdTmscnt:==" + baseList.getDnwdTmscnt());                          //목표심박수 계산을 위해 계산 계수, 단계하향01, 02 조회             vo.setCardioYn(baseList.getCardioYn());             vo.setAge(baseList.getAge());             vo.setSex(baseList.getSex());             vo.setExerStep(baseList.getExerStep());                          ExerMgmtVO calcList = selectCalcList(vo);             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.calcList.calcCfcnt:==" + calcList.getCalcCfcnt());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.calcList.stepDnwd01:==" + calcList.getStepDnwd01());             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.calcList.stepDnwd02:==" + calcList.getStepDnwd02());                          //최대심박수(HR max) & 목표심박수 계산             double THR = 0;                                                //목표심박수             //double HRrest = 0;                                        //안정시 심박수             double HRrest = baseList.getStbizHr();                        //안정시 심박수                          if("Y".equals(baseList.getCardioYn())) { //심질환                                  double Hrmax = 206 - (0.88 \* baseList.getAge());        //심질환 최대심박수                 [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.Hrmax(심질환최대심박):==" + Hrmax);                if(baseList.getDnwdTmscnt() == 0) {                                          THR = (Hrmax - HRrest) \* calcList.getCalcCfcnt() + HRrest;                                       } else if(baseList.getDnwdTmscnt() == 1) {                                          THR = (Hrmax - HRrest) \* calcList.getStepDnwd01() + HRrest;                                      } else {                                          if("P1".equals(baseList.getExerStep()) || "P2".equals(baseList.getExerStep())) {                                                  THR = 0;                                              } else {                                                  THR = (Hrmax - HRrest) \* calcList.getStepDnwd02() + HRrest;                                              }                                      }                                  [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.THR:==" + THR);                 int goalHr = Integer.parseInt(String.valueOf(Math.round(THR)));                                  vo.setGoalHr(goalHr);                              } else { //기타(당뇨, 고혈압, 간질환)질환                                  double Hrmax = 220 - baseList.getAge();                 [LOGGER.info](http://logger.info/)("ExerMgmtController.Hrmax:==" + Hrmax);                                  if(baseList.getDnwdTmscnt() == 0) {                                          THR = (Hrmax - HRrest) \* calcList.getCalcCfcnt() + HRrest;                                       } else if(baseList.getDnwdTmscnt() == 1) {                                          THR = (Hrmax - HRrest) \* calcList.getStepDnwd01() + HRrest;                                      } else {                                          THR = (Hrmax - HRrest) \* calcList.getStepDnwd02() + HRrest;                                      }                [LOGGER.info](http://logger.info/)("ExerMgmtController.THR:==" + THR);                 int goalHr = Integer.parseInt(String.valueOf(Math.round(THR)));                                  vo.setGoalHr(goalHr);                              }                          exerMgmtDao.updateGoalHr(vo);             transactionManager.commit(status);             [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.하향조정-목표심박수 업데이트 Sucess!!!");                      } catch (Exception e) {                          [LOGGER.info](http://logger.info/)("ExerMgmtServiceImpl.하향조정-목표심박수 업데이트 Fail!!!");             transactionManager.rollback(status);             throw e;                      }  [*[*9:41*]*](https://omnicns.slack.com/archives/D0YL22ZFS/p1461544874000003)  위에 선언  [*[*9:41*]*](https://omnicns.slack.com/archives/D0YL22ZFS/p1461544875000004)  @Autowired     private DataSourceTransactionManager transactionManager;  [*[*9:41*]*](https://omnicns.slack.com/archives/D0YL22ZFS/p1461544895000005)  AOP 안잡히는 특이케이스 부분 요렇게 처리  //커밋  transactionManager.commit(status);  //롤백  transactionManager.rollback(status); |

Scope Annotation

|  |
| --- |
| @Service("UserService")  @Scope("singleton")  **public** **class** UserServiceImpl **implements** UserService {      Logger log = LoggerFactory.*getLogger*(**this**.getClass());  **public** **static** **int** *i*=0; |

FileSystemResource (파일다운로드)

|  |
| --- |
| @RequestMapping(value = "/files/{file\_name}", method = RequestMethod.GET)  @ResponseBody  public FileSystemResource getFile(@PathVariable("file\_name") String fileName) {  return new FileSystemResource(myService.getFileFor(fileName));  }  // @RequestMapping(value="/app/apk/{fileName}", method = {RequestMethod.GET, RequestMethod.POST},produces = "application/vnd.android.package-archive")    @RequestMapping(value="/app/apk/{fileName}", method = {RequestMethod.***GET***, RequestMethod.***POST***} ,produces = MediaType.***APPLICATION\_OCTET\_STREAM\_VALUE***)  @ResponseBody  **public** FileSystemResource appAPK(@PathVariable("fileName") String fileName, HttpServletRequest request, HttpServletResponse response) **throws** Exception {  String nfs = PropertyUtil.*getSystemEnv*("OMNIFIT\_NFS");  fileName = fileName.replaceAll("\\.\\.", "");  File file = **new** File(nfs+"/app/apk/"+fileName+".apk");  response.setHeader("Content-Disposition", "attachment; filename="+file.getName());  **return** **new** FileSystemResource(file);  } |
| @RequestMapping(value="/snapshot/pdf/{name}", method = {RequestMethod.***GET***, RequestMethod.***POST***})  @ResponseBody  **public** FileSystemResource pdf(@PathVariable("name") String name, HttpServletRequest request, HttpServletResponse response) **throws** IOException {  // String cmd = "wkhtmltopdf --javascript-delay 10000 --page-size A4 --viewport-size 1090x1024 http://localhost/snapshot/view/1461857700068\_rqlwGXnmHJFRJqcgvYSofszsLzfGWIUQ d:\omnifit\_report.pdf";  String filePath = RequestUtil.*getRealPath*(request,cManager.getSnapshotPath()+PropertyUtil.*getFileSeparator*()+name+".pdf");  String cmd = "wkhtmltopdf --page-size A4 --viewport-size 1090x1024 http://localhost/snapshot/view/"+name+" "+filePath;  log.debug("cmd --> : "+cmd);  log.debug(RunTimeUtil.*execute*(cmd));  response.setContentType("application/pdf");  response.setHeader("Content-Disposition", "attachment; filename=report.pdf");  **return** **new** FileSystemResource(**new** File(filePath));  } |
| @RequestMapping(value = "/stuff/{stuffId}", method = RequestMethod.GET)  public ResponseEntity<InputStreamResource> downloadStuff(@PathVariable int stuffId)  throws IOException {  String fullPath = stuffService.figureOutFileNameFor(stuffId);  File file = new File(fullPath);  HttpHeaders respHeaders = new HttpHeaders();  respHeaders.setContentType("application/pdf");  respHeaders.setContentLength(12345678);  respHeaders.setContentDispositionFormData("attachment", "fileNameIwant.pdf");  InputStreamResource isr = new InputStreamResource(new FileInputStream(file));  return new ResponseEntity<InputStreamResource>(isr, respHeaders, HttpStatus.OK);  } |
| @RequestMapping(value="/downloadLogFile")  public void getLogFile(HttpSession session,HttpServletResponse response) throws Exception {  try {  String filePathToBeServed = //complete file name with path;  File fileToDownload = new File(filePathToBeServed);  InputStream inputStream = new FileInputStream(fileToDownload);  response.setContentType("application/force-download");  response.setHeader("Content-Disposition", "attachment; filename="+fileName+".txt");  IOUtils.copy(inputStream, response.getOutputStream());  response.flushBuffer();  inputStream.close();  } catch (Exception e){  LOGGER.debug("Request could not be completed at this moment. Please try again.");  e.printStackTrace();  }  } |
| @RequestMapping(value = "/download", method = RequestMethod.GET)  public ResponseEntity<byte[]> getDownloadData() throws Exception {  String regData = "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.";  byte[] output = regData.getBytes();  HttpHeaders responseHeaders = new HttpHeaders();  responseHeaders.set("charset", "utf-8");  responseHeaders.setContentType(MediaType.valueOf("text/html"));  responseHeaders.setContentLength(output.length);  responseHeaders.set("Content-disposition", "attachment; filename=filename.txt");  return new ResponseEntity<byte[]>(output, responseHeaders, HttpStatus.OK);  } |
| @RequestMapping(value="/snapshot/pdf", method = {RequestMethod.***GET***, RequestMethod.***POST***}, produces = "application/pdf")  **public** ResponseEntity<**byte**[]> savePDF(@RequestParam("html") String html, HttpServletRequest request, HttpServletResponse response) **throws** Exception {  File fileJsp = saveSnapshot(html,request,response);  File filePDF = saveSnapshotPDF(fileJsp.getName(), request);  HttpHeaders headers = **new** HttpHeaders();  headers.add("content-disposition", "inline; filename=" + filePDF.getName());  ResponseEntity<**byte**[]> responseb = **new** ResponseEntity<**byte**[]>(FileUtil.*readFileToByte*(filePDF), headers, HttpStatus.***OK***);  **return** responseb;  } |

@PathVariable 확장자 까지 포함해서 받기

|  |
| --- |
| @RequestMapping(value="/wait/download/{wait\_mode\_id}/{filename:.+}", method = {RequestMethod.*GET*, RequestMethod.*POST*}) |

파일업로드

Multipart

|  |
| --- |
| <!-- MultipartResolver 설정 -->  <bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <property name=*"maxUploadSize"* value=*"100000000"* />  <property name=*"maxInMemorySize"* value=*"100000000"* />  </bean>  @RequestMapping(value="/save", params="ac=ajax\_update")  **public** Code update(@RequestParam(value="file", required=**false**) MultipartFile file, @Valid @ModelAttribute("Notice") NoticeVO notice, HttpServletRequest request,HttpServletResponse reponse, ModelMap model) **throws** Exception {  service.updateNotice(notice);  service.deleteInsertFile(**new** NoticeFileVO(notice.getNotice\_id(),file) );  **return** Code.***SUCCESS***;  } |
| $scope.save = **function**(id) {  **var** data = $.fn.serializeObject("#form");  **var** formData = **new** FormData($("#form").get(0));  $oc.request(  {  url : $scope.actionUrl+"/save?"+$scope.CSRF\_NAME+"="+$scope.CSRF\_TOKEN+"&ac="+$("#ac").val(),  // url : $scope.actionUrl+"/"+$("#ac").val()+"?"+$scope.CSRF\_NAME+"="+$scope.CSRF\_TOKEN,  data : formData,  mimeType : "multipart/form-data",  processData : **false**, contentType : **false**, cache:**false**  },  **function**(data, textStatus, jqXHR){  $("#myModal").hide();  $location.path($scope.actionUrl);  }  );  } |
| http://winmargo.tistory.com/102  <bean id="multipartResolver"  class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>  참고로 프로퍼티들은  maxUploadSize(최대업로드 가능한 바이트크기) maxInMemorySize(디스크에 임시 파일을 생성하기 전에 메모리에 보관할수있는 최대 바이트 크기) defaultEncoding(요청을 파싱할 때 사용할 캐릭터 인코딩. 기본값 ISO-8859-1)  한글깨짐 처리  **new String(name.getBytes("8859\_1"),"utf-8");**  **또는**  <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

화면단에서csrf token을 header로 넘겨줘도된다.

|  |
| --- |
| sendParam.headers["X-CSRF-TOKEN"] = **this**.csrf\_value; |

파일업로드 Multipart

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <http://devbox.tistory.com/entry/Spring-%ED%8C%8C%EC%9D%BC%EC%97%85%EB%A1%9C%EB%93%9C-%EC%B2%98%EB%A6%AC> [[Spring] 파일업로드 처리](http://devbox.tistory.com/entry/Spring-%ED%8C%8C%EC%9D%BC%EC%97%85%EB%A1%9C%EB%93%9C-%EC%B2%98%EB%A6%AC) 2014.12.18 23:07  Posted in [Programing/Spring](http://devbox.tistory.com/category/Programing/Spring) by devbox  **파일 업로드 처리**  인코딩 타입이 Multipart인 경우 파라미터나 업로드한 파일을 구하려면 전송 데이터를 알맞게 처리해 주어야 한다.  스프링은 Multipart 지원 기능을 제공하고 있기 때문에, 이 기능을 이용하면 추가적인 처리없이 Multipart 형식으로 전송된 파라미터와 파일 정보를 쉽게 구할 수 있다.  **1. MultipartResolver 설정**  Multipart 지원 기능을 이용하려면 먼저 MultipartResolver를 스프링 설정 파일에 등록해 주어야 한다. MultipartResolver는 Multipart 형식으로 데이터가 전송된 경우, 해당 데이터를 스프링 MVC에서 사용할 수 있도록  변환해준다. 예를 들어, @PathVariable 어노테이션을 이용해서 Multipart로 전송된 파라미터와 파일을 사용할 수 있도록 해준다.  스프링이 기본으로 제공하는 MultipartResolver는 CommmnosMultipartResolver이다. CommmnsMultipartResolver는 Commons FileUpload API를 이용해서 Multipart를 처리해준다.  CommmonsMultipartResolver를 MultipartResolver로 사용하려면 다음과 같이 빈 이름으로 "multipartResolver"를 사용해서 등록하면 된다.  <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"></bean>  cf.) DispatcherServlet은 이름이 "multipartResolver"인 빈을 사용하기 때문에 다른 이름을 지정할 경우 MultipartResolver로 사용되지 않는다.  ※ CommonsMultipartResolver 클래스의 프로퍼티   |  |  |  | | --- | --- | --- | | 프로퍼티 | 타입 | 설명 | | maxUploadSize | long | 최대 업로드 가능한 바이트 크기, -1은 제한이 없음을 의미한다. 기본 값은 -1이다. | | maxInMemorysize | int | 디스크에 임시 파일을 생성하기 전에 메모리에 보관할 수 있는 최대 바이트 크기,   기본 값은 10240 바이트이다. | | defaultEncording | String | 요청을 파싱할 때 사용할 캐릭터 인코딩, 지정하지 않은 경  우 HttpServletRequest.setEncording() 메서드로 지정한 캐릭터 셋이 사용된다.   아무 값도 없을 경우 ISO-8859-1을 사용한다. |     **2. @RequestParam 어노테이션을 이용한 업로드 파일 접근**  업로드한 파일을 전달받는 첫 번째 방법은 @RequestParam 어노테이션이 적용된 MultipartFile 타입의 파라미터를 사용하는 것이다. 예를 들어, HTML 입력폼이 다음과 같이 작성되어 있다고 해보자  <form action="submitReport1.do" method="post" enctype="multipart/form-data">  학번: <input type="text" name="studentNumber" />  <br/>  리포트파일: <input type="file" name="report" />  <br/>  <input type="submit" />  </form>  위 HTML 코드에서 파일은 report 파라미터를 통해서 전달된다.  이 경우 다음 코드와 같이 @RequestParam 어노테이션과 MultipartFile 타입의 파라미터를 이용해서 업로드 파일 데이터를 전달받을 수 있다.  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.multipart.MultipartFile;  @Controller  public class ReportSubmissionController {  @RequestMapping(value = "/report/submitReport1.do", method = RequestMethod.POST)  public String submitReport1(  @RequestParam("studentNumber") String studentNumber,  // MultipartFile이 제공하는 메서드를 이용해서 업로드 데이터 접근  @RequestParam("report") MultipartFile report) {  printInfo(studentNumber, report);  return "report/submissionComplete";  }  ...  }  MultipartFile 인터페이스는 스프링에서 업로드 한 파일을 표현할 때 사용되는 인터페이스로서, MultipartFile 인터페이스를 이용해서 업로드 한 파일의 이름, 실제 데이터, 파일 크기 등을 구할 수 있다.  **3. MultipartHttpServletRequest를 이용한 업로드 파일 접근**  업로드한 파일을 전달받는 두 번째 방법은 MultipartHttpServletRequest 인터페이스를 이용하는 것이다.  import org.springframework.web.multipart.MultipartFile;  import org.springframework.web.multipart.MultipartHttpServletRequest;  @Controller  public class ReportSubmissionController {  @RequestMapping(value = "/report/submitReport2.do", method = RequestMethod.POST)  public String submitReport2(MultipartHttpServletRequest request) {  String studentNumber = request.getParameter("studentNumber");  MultipartFile report = request.getFile("report");  printInfo(studentNumber, report);  return "report/submissionComplete";  }  }  MultopartHttpServletRequest 인터페이스는 스프링이 제공하는 인터페이스로서, Multipart 요청이 들어올 때 내부적으로 원본 HttpServletRequest 대신 사용되는 Multipart 요청이 들어올 때  내부적으로 원본 HttpServletRequest 대신 사용되는 인터페이스이다. MultipartHttpServletRequest 인터페이스는 실제로는 어떤 메서드도 선언하고 있지 않으며,  HttpServletRequest 인터페이스와 MultipartRequest 인터페이스를 상속받고 있다. MultipartHttpServletRequest 인터페이스는 javax.servlet.HttpServletRequest 인터페이스를 상속받기 때문에 웹 요청 정보를 구하기 위한 getParameter()나 getHeader()와 같은 메서드를 사용할 수 있으며, 추가로 MultipartRequest 인터페이스가 제공하는 Multipart 관련 메서드를 사용할 수 있다.  ※ MultipartRequest 인터페이스의 파일 관련 주요 메서드   |  |  | | --- | --- | | 메서드 | 설 명 | | Iterator<String> getFileNames() | 업로드 된 파일들의 이름 목록을 제공하는 Iterator를 구한다. | | MultipartFile getfile(String name) | 파라미터 이름이 name이 업로드 파일 정보를 구한다. | | List<MultipartFile> getFiles(String name) | 파라미터 이름이 name인 업로드 파일 정보 목록을 구한다. | | Map<String, MultipartFile> getFileMap() | 파라미터 이름을 키로 파라미터에 해당하는 파일 정보를 값으로 하는 Map을 구한다. |   **4. 커맨드 객체를 통한 업로드 파일 접근**  커맨드 객체를 이용해도 업로드 한 파일을 전달받을 수 있다. 단지 커맨드 클래스에 파라미터와 동일한 이름의 MultipartFile 타입 프로퍼티를 추가해주기만 하면 된다.  예를 들어, 업로드 파일의 파라미터 이름이 "report"인 경우, 다음과 같이 "report" 프로퍼티를 커맨드 클래스에 추가해 주면 된다.  import org.springframework.web.multipart.MultipartFile;  public class ReportCommand {  private String studentNumber;  private MultipartFile report;  public String getStudentNumber() {  return studentNumber;  }  public void setStudentNumber(String studentNumber) {  this.studentNumber = studentNumber;  }  public MultipartFile getReport() {  return report;  }  public void setReport(MultipartFile report) {  this.report = report;  }  }  위 코드와 같이 MultipartFile 타입의 프로퍼티를 커맨드 클래스에 추가해주었다면, @RequestMapping 메서드의 커맨드 객체로 사용함으로써 업로드 파일 정보를 커맨드 객체를 통해서 전달받을 수 있게 된다.  @Controller  public class ReportSubmissionController{  ...  @RequestMapping(value="/report/submitReport3.do",medhod = RequestMethod.POST)  public String submitReport3(ReportCommand reportCommand){  ...  }  }  **5. MultupartFile 인터페이스 사용**  org.springframework.web.MutipartFile 인터페이스는 업로드 한 파일 및 파일 데이터를 표현하기 위한 용도로 사용된다.  MultipartFile 인터페이스가 제공하는 주요 메서드는 다음과 같다.  ※ MulripartFile 인터페이스의 주요 메서드   |  |  | | --- | --- | | 메서드 | 설 명 | | String getName() | 파라미터 이름을 구한다. | | String getOriginalFilename() | 업로드 한 파일의 이름을 구한다. | | String isEmpty() | 업로드 한 파일이 존재하지 않는 경우 true를 리턴한다. | | long getSize() | 업로드 한 파일의 크기를 구한다. | | byte[ ] getBytes() throws IOExcetion | 업로드 한 파일 데이터를 구한다. | | InputStream getInputStream() throws  IOException | 업로드한 파일 데이터를 읽어노는 InputStream을 구한다. InputStream의 사용  이 끝나면 알맞게 종료해  주어야 한다. | | void transferTo(File dest) throws IOException | 업로드 한 파일 데이터를 지정한 파일에 저장한다. |   업로드 한 파일 데이터를 구하는 가장 단순한 방법은 MultipartFile.getByte() 메서드를 이용하는 것이다. 바이트 배열을 구한 뒤에 파일이나 DB등에 저장하면된다.  if(mutipartFile.isEmpty()){  byte[ ] fileData = multipartFile.getBytes();  //byte 배열을 파일/DB/네트워크 등으로 전송  ...  }  업로드 한 파일 데이터를 특정 파일로 저장하고 싶다면 MultipartFile.transferTo() 메서드를 사용하는 것이 편리하다.  if(mutipartFile.isEmpty()){  File file = new File(fileName);  multipartFile.transferTo(file);  ...  } |

Security

|  |
| --- |
| 시큐리티 쪽 컨트롤 접근  http://syaku.tistory.com/278  http://goldenraccoon.tistory.com/entry/SPRING-SECURITY-Secured-%EC%96%B4%EB%85%B8%ED%85%8C%EC%9D%B4%EC%85%98-%EC%82%AC%EC%9A%A9  ExpressionUtils.evaluateAsBoolean(accessExpression, createExpressionEvaluationContext(handler))  org.springframework.security.core.context.SecurityContextImpl@fbe4be2f: Authentication: org.springframework.security.authentication.UsernamePasswordAuthenticationToken@fbe4be2f: Principal: com.ko.omnicns.omnifit.cms.user.vo.UserVO@3a81f90b[admin\_id=AD1000000000,counsel\_id=<null>,corp\_id=CP100000,admin\_lgin\_id=omnifit,admin\_lgin\_pwd=93cf03922d751a19ceee3e2522f04c435c1eb5af0deb53c3f4cdddf53f79ff136c686c6b06cb1b644cd1676dbec71486cb02f2b90932dfe1d33128439812da38,distb\_id=<null>,admin\_gp\_no=AG100001,use\_type=UT001,use\_yn=Y,login\_fail\_cnt=0,login\_waittime=2016-03-23 18:25:26,access\_token=eSSNMNGVtzDNNWqIubHsIKdEYWGCfxAVuogbLrIuHlnJwTYmXXsqHbyAwIKsRPTddrBCaBZOOkEijDEIJWqXWGdypStWWnsDDqSSjBHinNhxesQDbfuLiiXpMjSvopEQ,refresh\_token=refresh\_token,expire\_dt=Thu Apr 07 19:35:39 KST 2016,indv\_key=7ce748b0387711c674a0ab4f52b2903b,allow\_ip=<null>,last\_login\_dt=Mon Mar 28 19:35:39 KST 2016,last\_login\_ip=0:0:0:0:0:0:0:1,svc\_id=SC1020,id=1]; Credentials: [PROTECTED]; Authenticated: true; Details: org.springframework.security.web.authentication.WebAuthenticationDetails@0: RemoteIpAddress: 0:0:0:0:0:0:0:1; SessionId: 7F9B2E758131F20685C3477356F99688; Granted Authorities: 10000196, 10000196\_R, 10000196\_C, 10000196\_U, 10000196\_D, /stics/meas/pulse/avg/vitality, /stics/meas/pulse/avg/vitality\_R, /stics/meas/pulse/avg/vitality\_C, /stics/meas/pulse/avg/vitality\_U, /stics/meas/pulse/avg/vitality\_D, 10000197, 10000197\_R, 10000197\_C, 10000197\_U, 10000197\_D, /stics/meas/pulse/avg/sns\_active, /stics/meas/pulse/avg/sns\_active\_R, /stics/meas/pulse/avg/sns\_active\_C, /stics/meas/pulse/avg/sns\_active\_U, /stics/meas/pulse/avg/sns\_active\_D, 10000198, 10000198\_R, 10000198\_C, 10000198\_U, 10000198\_D, /stics/meas/pulse/avg/psns\_active, /stics/meas/pulse/avg/psns\_active\_R, /stics/meas/pulse/avg/psns\_active\_C, /stics/meas/pulse/avg/psns\_active\_U, /stics/meas/pulse/avg/psns\_active\_D, 10000199, 10000199\_R, 10000199\_C, 10000199\_U, 10000199\_D, /stics/meas/pulse/avg/ans\_active, /stics/meas/pulse/avg/ans\_active\_R, /stics/meas/pulse/avg/ans\_active\_C, /stics/meas/pulse/avg/ans\_active\_U, /stics/meas/pulse/avg/ans\_active\_D, 10000200, 10000200\_R, 10000200\_C, 10000200\_U, 10000200\_D, /stics/meas/pulse/avg/ans\_balance, /stics/meas/pulse/avg/ans\_balance\_R, /stics/meas/pulse/avg/ans\_balance\_C, /stics/meas/pulse/avg/ans\_balance\_U, /stics/meas/pulse/avg/ans\_balance\_D, 10000201, 10000201\_R, 10000201\_C, 10000201\_U, 10000201\_D, /stics/meas/pulse/avg/blood, /stics/meas/pulse/avg/blood\_R, /stics/meas/pulse/avg/blood\_C, /stics/meas/pulse/avg/blood\_U, /stics/meas/pulse/avg/blood\_D, 10000202, 10000202\_R, 10000202\_C, 10000202\_U, 10000202\_D, /stics/mental, /stics/mental\_R, /stics/mental\_C, /stics/mental\_U, /stics/mental\_D, 10000203, 10000203\_R, 10000203\_C, 10000203\_U, 10000203\_D, /stics/mental/active\_type/cnt, /stics/mental/active\_type/cnt\_R, /stics/mental/active\_type/cnt\_C, /stics/mental/active\_type/cnt\_U, /stics/mental/active\_type/cnt\_D, 10000204, 10000204\_R, 10000204\_C, 10000204\_U, 10000204\_D, /stics/mental/active\_type/ratio, /stics/mental/active\_type/ratio\_R, /stics/mental/active\_type/ratio\_C, /stics/mental/active\_type/ratio\_U, /stics/mental/active\_type/ratio\_D, 10000205, 10000205\_R, 10000205\_C, 10000205\_U, 10000205\_D, /stics/mental/pstrs/cnt, /stics/mental/pstrs/cnt\_R, /stics/mental/pstrs/cnt\_C, /stics/mental/pstrs/cnt\_U, /stics/mental/pstrs/cnt\_D, 10000206, 10000206\_R, 10000206\_C, 10000206\_U, 10000206\_D, /stics/mental/pstrs/ratio, /stics/mental/pstrs/ratio\_R, /stics/mental/pstrs/ratio\_C, /stics/mental/pstrs/ratio\_U, /stics/mental/pstrs/ratio\_D, 10000207, 10000207\_R, 10000207\_C, 10000207\_U, 10000207\_D, /stics/mental/adhd\_adult/cnt, /stics/mental/adhd\_adult/cnt\_R, /stics/mental/adhd\_adult/cnt\_C, /stics/mental/adhd\_adult/cnt\_U, /stics/mental/adhd\_adult/cnt\_D, 10000208, 10000208\_R, 10000208\_C, 10000208\_U, 10000208\_D, /stics/mental/adhd\_adult/ratio, /stics/mental/adhd\_adult/ratio\_R, /stics/mental/adhd\_adult/ratio\_C, /stics/mental/adhd\_adult/ratio\_U, /stics/mental/adhd\_adult/ratio\_D, 10000209, 10000209\_R, 10000209\_C, 10000209\_U, 10000209\_D, /stics/mental/adhd\_student/cnt, /stics/mental/adhd\_student/cnt\_R, /stics/mental/adhd\_student/cnt\_C, /stics/mental/adhd\_student/cnt\_U, /stics/mental/adhd\_student/cnt\_D, 10000210, 10000210\_R, 10000210\_C, 10000210\_U, 10000210\_D, /stics/mental/adhd\_student/ratio, /stics/mental/adhd\_student/ratio\_R, /stics/mental/adhd\_student/ratio\_C, /stics/mental/adhd\_student/ratio\_U, /stics/mental/adhd\_student/ratio\_D, 10000011, 10000011\_R, 10000011\_C, 10000011\_U, 10000011\_D, /service, /service\_R, /service\_C, /service\_U, /service\_D, 10000115, 10000115\_R, 10000115\_C, 10000115\_U, 10000115\_D, /counsel/meas\_h/pulse/sns\_active, /counsel/meas\_h/pulse/sns\_active\_R, /counsel/meas\_h/pulse/sns\_active\_C, /counsel/meas\_h/pulse/sns\_active\_U, /counsel/meas\_h/pulse/sns\_active\_D, 10000220, 10000220\_R, 10000220\_C, 10000220\_D, /main, /main\_R, /main\_C, /main\_D, 10000007, 10000007\_R, 10000007\_C, 10000007\_U, 10000007\_D, /user, /user\_R, /user\_C, /user\_U, /user\_D, 10000008, 10000008\_R, 10000008\_C, 10000008\_U, 10000008\_D, /user/list, /user/list\_R, /user/list\_C, /user/list\_U, /user/list\_D, 10000009, 10000009\_R, 10000009\_C, 10000009\_U, 10000009\_D, /user/detail, /user/detail\_R, /user/detail\_C, /user/detail\_U, /user/detail\_D, 10000010, 10000010\_R, 10000010\_C, 10000010\_U, 10000010\_D, /user/regist, /user/regist\_R, /user/regist\_C, /user/regist\_U, /user/regist\_D, 10000012, 10000012\_R, 10000012\_C, 10000012\_U, 10000012\_D, /service/svc, /service/svc\_R, /service/svc\_C, /service/svc\_U, /service/svc\_D, 10000013, 10000013\_R, 10000013\_C, 10000013\_U, 10000013\_D, /service/svc/list, /service/svc/list\_R, /service/svc/list\_C, /service/svc/list\_U, /service/svc/list\_D, 10000014, 10000014\_R, 10000014\_C, 10000014\_U, 10000014\_D, /service/svc/detail, /service/svc/detail\_R, /service/svc/detail\_C, /service/svc/detail\_U, /service/svc/detail\_D, 10000015, 10000015\_R, 10000015\_C, 10000015\_U, 10000015\_D, /service/svc/regist, /service/svc/regist\_R, /service/svc/regist\_C, /service/svc/regist\_U, /service/svc/regist\_D, 10000016, 10000016\_R, 10000016\_C, 10000016\_U, 10000016\_D, /service/dvc, /service/dvc\_R, /service/dvc\_C, /service/dvc\_U, /service/dvc\_D, 10000017, 10000017\_R, 10000017\_C, 10000017\_U, 10000017\_D, /service/dvc/list, /service/dvc/list\_R, /service/dvc/list\_C, /service/dvc/list\_U, /service/dvc/list\_D, 10000018, 10000018\_R, 10000018\_C, 10000018\_U, 10000018\_D, /service/dvc/detail, /service/dvc/detail\_R, /service/dvc/detail\_C, /service/dvc/detail\_U, /service/dvc/detail\_D, 10000019, 10000019\_R, 10000019\_C, 10000019\_U, 10000019\_D, /service/dvc/regist, /service/dvc/regist\_R, /service/dvc/regist\_C, /service/dvc/regist\_U, /service/dvc/regist\_D, 10000020, 10000020\_R, 10000020\_C, 10000020\_U, 10000020\_D, /device, /device\_R, /device\_C, /device\_U, /device\_D, 10000021, 10000021\_R, 10000021\_C, 10000021\_U, 10000021\_D, /device/list, /device/list\_R, /device/list\_C, /device/list\_U, /device/list\_D, 10000022, 10000022\_R, 10000022\_C, 10000022\_U, 10000022\_D, /device/detail, /device/detail\_R, /device/detail\_C, /device/detail\_U, /device/detail\_D, 10000023, 10000023\_R, 10000023\_C, 10000023\_U, 10000023\_D, /device/regist, /device/regist\_R, /device/regist\_C, /device/regist\_U, /device/regist\_D, 10000024, 10000024\_R, 10000024\_C, 10000024\_U, 10000024\_D, /mnt, /mnt\_R, /mnt\_C, /mnt\_U, /mnt\_D, 10000025, 10000025\_R, 10000025\_C, 10000025\_U, 10000025\_D, /mnt/corp, /mnt/corp\_R, /mnt/corp\_C, /mnt/corp\_U, /mnt/corp\_D, 10000026, 10000026\_R, 10000026\_C, 10000026\_U, 10000026\_D, /mnt/corp/list, /mnt/corp/list\_R, /mnt/corp/list\_C, /mnt/corp/list\_U, /mnt/corp/list\_D, 10000027, 10000027\_R, 10000027\_C, 10000027\_U, 10000027\_D, /mnt/corp/info, /mnt/corp/info\_R, /mnt/corp/info\_C, /mnt/corp/info\_U, /mnt/corp/info\_D, 10000028, 10000028\_R, 10000028\_C, 10000028\_U, 10000028\_D, /mnt/corp/service, /mnt/corp/service\_R, /mnt/corp/service\_C, /mnt/corp/service\_U, /mnt/corp/service\_D, 10000029, 10000029\_R, 10000029\_C, 10000029\_U, 10000029\_D, /mnt/corp/img, /mnt/corp/img\_R, /mnt/corp/img\_C, /mnt/corp/img\_U, /mnt/corp/img\_D, 10000030, 10000030\_R, 10000030\_C, 10000030\_U, 10000030\_D, /mnt/corp/regist, /mnt/corp/regist\_R, /mnt/corp/regist\_C, /mnt/corp/regist\_U, /mnt/corp/regist\_D, 10000031, 10000031\_R, 10000031\_C, 10000031\_U, 10000031\_D, /mnt/admin, /mnt/admin\_R, /mnt/admin\_C, /mnt/admin\_U, /mnt/admin\_D, 10000032, 10000032\_R, 10000032\_C, 10000032\_U, 10000032\_D, /mnt/admin/list, /mnt/admin/list\_R, /mnt/admin/list\_C, /mnt/admin/list\_U, /mnt/admin/list\_D, 10000033, 10000033\_R, 10000033\_C, 10000033\_U, 10000033\_D, /mnt/admin/info, /mnt/admin/info\_R, /mnt/admin/info\_C, /mnt/admin/info\_U, /mnt/admin/info\_D, 10000034, 10000034\_R, 10000034\_C, 10000034\_U, 10000034\_D, /mnt/admin/menu, /mnt/admin/menu\_R, /mnt/admin/menu\_C, /mnt/admin/menu\_U, /mnt/admin/menu\_D, 10000035, 10000035\_R, 10000035\_C, 10000035\_U, 10000035\_D, /mnt/admin/corp, /mnt/admin/corp\_R, /mnt/admin/corp\_C, /mnt/admin/corp\_U, /mnt/admin/corp\_D, 10000036, 10000036\_R, 10000036\_C, 10000036\_U, 10000036\_D, /mnt/admin/regist, /mnt/admin/regist\_R, /mnt/admin/regist\_C, /mnt/admin/regist\_U, /mnt/admin/regist\_D, 10000037, 10000037\_R, 10000037\_C, 10000037\_U, 10000037\_D, /mnt/mng, /mnt/mng\_R, /mnt/mng\_C, /mnt/mng\_U, /mnt/mng\_D, 10000038, 10000038\_R, 10000038\_C, 10000038\_U, 10000038\_D, /mnt/mng/list, /mnt/mng/list\_R, /mnt/mng/list\_C, /mnt/mng/list\_U, /mnt/mng/list\_D, 10000039, 10000039\_R, 10000039\_C, 10000039\_U, 10000039\_D, /mnt/mng/user, /mnt/mng/user\_R, /mnt/mng/user\_C, /mnt/mng/user\_U, /mnt/mng/user\_D, 10000040, 10000040\_R, 10000040\_C, 10000040\_U, 10000040\_D, /mnt/mng/counsel, /mnt/mng/counsel\_R, /mnt/mng/counsel\_C, /mnt/mng/counsel\_U, /mnt/mng/counsel\_D, 10000041, 10000041\_R, 10000041\_C, 10000041\_U, 10000041\_D, /mnt/mng/sensing, /mnt/mng/sensing\_R, /mnt/mng/sensing\_C, /mnt/mng/sensing\_U, /mnt/mng/sensing\_D, 10000042, 10000042\_R, 10000042\_C, 10000042\_U, 10000042\_D, /mnt/mng/regist, /mnt/mng/regist\_R, /mnt/mng/regist\_C, /mnt/mng/regist\_U, /mnt/mng/regist\_D, 10000043, 10000043\_R, 10000043\_C, 10000043\_U, 10000043\_D, /bd\_old, /bd\_old\_R, /bd\_ol...  /WEB-INF/jsp/main/index2.jsp  <c:if test="${not empty error}">  <div class="errorblock">  Your login attempt was not successful, try again.<br /> Caused : ${sessionScope["SPRING\_SECURITY\_LAST\_EXCEPTION"].message}  </div>  </c:if>  <%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>  @RequestMapping(value="/main",params="act=menulist", method = {RequestMethod.GET, RequestMethod.POST},produces = "application/json; charset=utf8")  @PreAuthorize("hasAuthority('/main\_U')")  // @PreAuthorize("hasRole('\/main\_C')")  // @AuthorizeRequest("hasRole('ROLE\_USER')")  // @Secured("/main\_C")  @ResponseBody  public String main\_menuList(@RequestBody String body,HttpServletRequest request, HttpServletResponse response) throws Exception {  log.debug("menulist\*\*");  Collection<? extends GrantedAuthority> list = SecurityUtil.getGrantedAuthorityList();  for (GrantedAuthority grantedAuthority : list) {  log.debug("grantedAuthority.getAuthority()"+grantedAuthority.getAuthority());  }    return "main/index";  }  로그인 로그아웃  <c:url value="j\_spring\_security\_logout" /> 로그아웃버튼  http://zgundam.tistory.com/51  <sec:authorize access="isAnonymous()">  <http://goldenraccoon.tistory.com/entry/%EB%A1%9C%EA%B7%B8%EC%9D%B8-%EC%97%AC%EB%B6%80%EC%97%90-%EB%94%B0%EB%9D%BC-%EB%A9%94%EB%89%B4-%EC%88%A8%EA%B8%B0%EA%B3%A0-%EB%82%98%ED%83%80%EB%82%B4%EA%B8%B0-SECURITY>  @InitBinder  http://kunner.tistory.com/m/post/998 |

**관련 문서http://noritersand.tistory.com/156#@Service**

* <http://docs.spring.io/spring/docs>

**목차**

* [@Component](http://noritersand.tistory.com/156#@Component)
* [@Required](http://noritersand.tistory.com/156#@Required)
* [@Autowired](http://noritersand.tistory.com/156#@Autowired)
* [@Qualifier](http://noritersand.tistory.com/156#@Qualifier)
* [@Resource](http://noritersand.tistory.com/156#@Resource)
* [@Scope](http://noritersand.tistory.com/156#@Scope)
* [@PostConstruct](http://noritersand.tistory.com/156#@PostConstruct)
* [@PreDestroy](http://noritersand.tistory.com/156#@PreDestroy)
* [@Inject](http://noritersand.tistory.com/156#@Inject)
* [@Service](http://noritersand.tistory.com/156#@Service)
* [@Repository](http://noritersand.tistory.com/156#@Repository)
* [@Controller](http://noritersand.tistory.com/156#@Controller)
* [@RequestMapping](http://noritersand.tistory.com/156#@RequestMapping)
* [@RequestParam](http://noritersand.tistory.com/156#@RequestParam)
* [@SessionAttributes](http://noritersand.tistory.com/156#@SessionAttributes)
* [@InitBinder](http://noritersand.tistory.com/156#@InitBinder)
* [@ModelAttribute](http://noritersand.tistory.com/156#@ModelAttribute)
* [@RequestBody](http://noritersand.tistory.com/156#@RequestBody)
* [@ResponseBody](http://noritersand.tistory.com/156#@ResponseBody)
* [@PathVariable](http://noritersand.tistory.com/156#@PathVariable)

개발단계에서 사용빈도가 높은 어노테이션 위주로 정리.

목차에 없는 항목은 API 문서를 참고할 것. ~~구글링하는게속편한건함정~~

## @Component

패키지: org.springframework.stereotype

버전: spring 2.5

설정 위치: 클래스 선언부 앞

<context:component-scan> 태그를 설정파일에 추가하면 해당 어노테이션이 적용된 클래스를 빈으로 등록하게 된다. 범위는 디폴트로 singleton이며 @Scope를 사용하여 지정할 수 있다.

사용하려면 XML 설정파일에 <context:component-scan>을 정의하고 적용할 기본  패키지를 base-package 속성으로 등록한다.

context:annotation-config 태그는 어노테이션과 관련해서 다음의 BeanPostProcessor를 함께 등록 한다.

* @Required(RequiedAnnotationBeanPostProcessor)
* @Autowired(AutowiredAnnotationBeanPostProcessor)
* @Resource, @PostConstruct, @PreDestory(CommonAnnotationBeanPostProcessor)
* @Configuration(ConfigurationClassPostProcessor)
* 그 외 Repository, Service, Controller 포함

예를 들어 다음처럼 설정하면:

|  |  |
| --- | --- |
| <context:component-scan base-package="xxx"/> | [cs](http://colorscripter.com/info#e) |

xxx 패키지 하위에 @Component로 선언된 클래스를 bean으로 자동 등록한다. bean의 이름은 해당 클래스명(첫글자는 소문자)이 사용된다.

<context:component-scan /> 요소에는 scoped-proxy 속성이 존재 한다. scoped-proxy는 <aop:scoped-poxy/>처럼 WebApplicationContext 에서만 유효하며 "session", "globalSession", "request" 이외의 scope는 무시 되며 아래의 3가지 값을 설정 할 수 있다.

* no: proxy를 생성하지 않는다.(기본값)
* interfaces: JDK Dynamic Proxy를 이용한 Proxy 생성
* targetClass: 클래스에 대해 프록시를 생성(CGLIB를 이용한 Proxy 생성)

|  |  |
| --- | --- |
| @Component  @Scope("prototype")   // 생략하면 싱글톤  public class Test {         .....  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**CGLIB**

기존의 자바 클래스파일로부터 자바의 소스코드를 동적으로 생성하는 라이브러리(자바 소스 변경)

<http://sourceforge.net/projects/cglib/>

**스캔 대상 클래스 범위 지정하기**

<context:include-filter> 태그와 <context:exclude-filter> 태그를 사용하면 자동 스캔 대상에 포함시킬 클래스와 포함시키지 않을 클래스를 구체적으로 명시할 수 있다.

|  |  |
| --- | --- |
| <context:component-scan base-package="spring.demo" scoped-proxy="no">     <context:include-filter type="regex" expression="\*HibernateRepository"/>     <context:exclude-filter type="aspectj" expression="..\*IBatisRepository"/>  </context:component-scan>  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

위와 같이 <context:include-filter> 태그와 <context:exclude-filter> 태그는 각각 type 속성과 expresseion 속성을 갖는데, type 속성에 따라 expression 속성에 올 수 있는 값이 달라진다. type 속성에 입력가능한 값은 다음과 같다:

* annotation: 클랙스에 지정한 어노테이션이 적용됐는지의 여부. expression 속성에서는 "org.example.SomeAnnotation"와 같은 어노테이션 이름을 입력한다.
* assignable: 클래스가 지정한 타입으로 할당 가능한지의 여부.  expression 속성에는 "org.exampleSomeClass" 와 같은 타입 이름을 입력한다.
* regex: 클래스 이름이 정규 표현식에 매칭되는 지의 여부.  expression 속성에는 "org\.example\.Default.\*" 와 같이 정규표현식을 입력한다.
* aspectj: 클래스 이름이 AspectJ 의 표현식에 매칭되는 지의 여부.  expression 속성에는 "org.example..\*Service+" 와 같이 AspectJ 의 표현식을 입력한다.

## @Required

패키지: org.springframework.beans.factory.annotation

버전: spring 2.0

설정 위치: setter 메서드 앞

Required 어노테이션은 필수 프로퍼티임을 명시하는 것으로 필수 프로퍼티를 설정하지 않을 경우 빈 생성시 예외를 발생시킨다.

|  |  |
| --- | --- |
| import org.springframework.beans.factory.annotation.Required    public class TestBean {      @Required      private TestDao testDao;        public void setTestDao(TestDao testDao) {          this.testDao = testDao;      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |
| <bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanpostProcessor"/>  <bean name="testBean"  class="han.test.TestBean">      <property name="testDao" ref="testDao"/>      <!-- @Required 어노테이션을 적용하였으므로 설정하지 않으면 예외를 발생시킨다. -->  </bean>  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

RequiredAnnotationBeanPostProcessor 클래스는 스프링 컨테이너에 등록된 bean 객체를 조사하여 @Required 어노테이션으로 설정되어 있는 프로퍼티의 값이 설정되어 있는지 검사한다.

사용하려면 <bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" /> 클래스를 빈으로 등록시켜줘야 하지만 이를 대신하여 <context:annotation-config> 태그를 사용해도 된다:

|  |  |
| --- | --- |
| <beans xmlns="http://www.springframework.org/schema/beans"            xmlns:context="http://www.springframework.org/schema/context"            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:schemaLocation="http://www.springframework.org/schema/beans               http://www.springframework.org/schema/beans/spring-beans-3.1.xsd               http://www.springframework.org/schema/context               http://www.springframework.org/schema/context/spring-context-3.1.xsd">      <context:annotation-config/>  </beans>  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @Autowired

패키지: org.springframework.beans.factory.annotation

버전: spring 2.5

설정 위치: 생성자, 필드, 메서드(setter메서드가 아니여도 된다) 앞

의존관계를 자동설정할 때 사용하며 타입을 이용하여 의존하는 객체를 삽입해 준다. 그러므로 해당 타입의 빈객체가 존재하지 않거나 또는 2개 이상 존재할 경우 스프링은 예외를 발생시키게 된다.

**options:**

* required: Autowired 어노테이션을 적용한 프로퍼티 중 반드시 설정할 필요가 없는 경우에 false값을 주어 프로퍼티가 존재하지 않더라도 스프링이 예외를 발생하지 않도록 한다. 기본값은 TRUE. ex) @Autowired(required=false)

사용하려면 <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor" /> 클래스를 빈으로 등록시켜줘야 한다. 해당 설정 대신에 <context:annotation-config> 태그를 사용해도 된다.

@Autowired를 적용할 때 같은 타입의 빈이 2개 이상 존재하게 되면 예외가 발생하는데, Autowired도 이러한 문제가 발생한다. 이럴 때 @Qualifier를 사용하면 동일한 타입의 빈 중 특정 빈을 사용하도록 하여 문제를 해결할 수 있다.

|  |  |
| --- | --- |
| @Autowired  @Qualifier("test")  private Test test; | [cs](http://colorscripter.com/info#e) |

## @Qualifier

패키지: org.springframework.beans.factory.annotation

버전: spring 2.5

설정 위치: @Autowired 어노테이션과 함께 사용된다.

qualifier 어노테이션은 @Autowired의 목적에서 동일 타입의 빈객체가 존재시 특정빈을 삽입할 수 있게 설정한다. @Qualifier("mainBean")의 형태로 @Autowired와 같이 사용하며 해당 <bean>태그에 <qualifire value="mainBean" /> 태그를 선언해주어야 한다. 메서드에서 두개이상의 파라미터를 사용할 경우는 파라미터 앞에 선언해야한다.

**options**:

* name: alias명

사용하려면 동일타입의 빈객체 설정에서 <qualifier value="[alias명]" />를 추가해 준다.

|  |  |
| --- | --- |
| <bean id="user2" class="com.sp4.UserImpl">      <property name="name" value="스프링"/>      <property name="age" value="20"/>      <property name="tel" value="000-0000-0000"/>  </bean>    <bean id="userService1" class="com.sp4.UserService"/> | [cs](http://colorscripter.com/info#e) |
| public class UserService {      @Autowired      @Qualifier("user2")      private User user;        public String result() {          return user.getData();      }  } | [cs](http://colorscripter.com/info#e) |

## @Resource

자바 6 및 JEE5에 추가된 것으로 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용 한다. 스프링 2.5 부터 지원하는 어노테이션으로 스프링에서는 의존하는 빈 객체를 전달할 때 사용한다.

@Autowired와 흡사하지만 @Autowired는 타입으로(by type), @Resource는 이름으로(by name)으로 연결한다는 점이 다르다.

**options**:

* name: 자동으로 연결될 빈객체의 이름을 입력한다. ex) @Resource(name="testDao")

사용하려면 <bean class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/> 클래스를 빈으로 등록시켜줘야 한다. 해당 설정 대신에 <context:annotation-config> 태그를 사용해도 된다.

|  |  |
| --- | --- |
| <beans>      <!-- 기타 설정 생략 -->      <context:annotation-config/>        <bean id="user2" class="com.test.UserImpl" p:data="65536"/>  </beans>  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |
| public class UserService {      @Resource(name="user2")      private User user;      //UserImpl user2 = new UserImpl();      //User user = user2;        public void setUser(User user) {          this.user = user;      }      public String result() {          return user.getData();      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @Scope

패키지: org.springframework.beans.factory.annotation

설정: prototype, singleton, request, session, globalSession

스프링은 기본적으로 빈의 범위를 "singleton" 으로 설정한다. "singleton" 이 아닌 다른범위를 지정하고 싶다면 @Scope 어노테이션을 이용하여 범위를 지정한다.

|  |  |
| --- | --- |
| @Component  @Scope(value="prototype")  public class Worker { } | [cs](http://colorscripter.com/info#e) |
| @Component  @Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET\_CLASS)  public class Worker { }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @PostConstruct

패키지: javax.annotation

버전: jdk1.6, spring 2.5

설정 위치: 초기화 작업 수행 메서드 앞

의존하는 객체를 설정한 이후에 초기화 작업을 수행하기 위해 사용한다.

사용하려면 CommonAnnotationBeanPostProcessor 클래스를 빈으로 등록시켜줘야 한다. <context:annotation-config> 태그로 대신할 수 있다.

|  |  |
| --- | --- |
| @PostConstruct  public void init() {      System.out.println("객체 생성 후 내가 먼저 실행된다.");  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @PreDestroy

패키지: javax.annotation

버전: jdk1.6, spring 2.5

설정 위치: 해당 작업 메서드 앞

컨테이너에서 객체를 제거하기 전에 해야할 작업을 수행하기 위해 사용한다.

사용하려면 CommonAnnotationBeanPostProcessor 클래스를 빈으로 등록시켜줘야 한다. <context:annotation-config> 태그로 대신할 수 있다.

## @Inject

SR-330 표준 Annotation으로 Spring 3 부터 지원하는 Annotation이다. 특정 Framework에 종속되지 않은 어플리케이션을 구성하기 위해서는 @Inject를 사용할 것을 권장한다. @Inject를 사용하기 위해서는 클래스 패스 내에 JSR-330 라이브러리인 javax.inject-x.x.x.jar 파일이 추가되어야 함에 유의해야 한다.

## @Service

@Service를 적용한 Class는 비지니스 로직이 들어가는 Service로 등록이 된다. Controller에 있는 @Autowired는 @Service("xxxService")에 등록된 xxxService와 변수명이 같아야 하며 Service에 있는 @Autowired는 @Repository("xxxDao")에 등록된 xxDao와 변수명이 같아야 한다.

|  |  |
| --- | --- |
| @Service("helloService")  public class HelloServiceImpl implements HelloService {      @Autowired      private HelloDao helloDao;        public void hello() {          System.out.println("HelloServiceImpl :: hello()");          helloDao.selectHello();      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

helloDao.selectHello(); 와 같이 @Autowired를 이용한 객체를 이용하여 Dao 객체를 호출한다:

|  |  |
| --- | --- |
| @Service("test2.testService")  //괄호 속 문자열은 식별자를 의미한다.  //괄호를 생략할 경우 클래스명 그대로 사용한다.  //따라서 ,같은 클래스명이 존재 할 시 같은 식별자가 생성되기때문에 에러가 발생한다.  public class TestService {      public String result(int num1, int num2, String oper) {          String str = null;            if (oper.equals("+")) {              //...              return str;          }      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

@Resouce로 연결

|  |  |
| --- | --- |
| @Resource(name="test2.testService")  //name에 필요한 것은 @Service("test2.testService") <- 여기서 괄호 속 문자열, 즉 식별자    private TestService service;  //TestService service = new TestService(); 라고 하는것과 같은 식    @RequestMapping(value="/test2/oper.action", method={RequestMethod.GET})  public String form() throws Exception {      return "test2/write";  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @Repository

패키지: org.springframework.stereotype

버전: spring 2.0

@Repository는 일반적으로 DAO에 사용되며 DB Exception을 DataAccessException으로 변환한다.

|  |  |
| --- | --- |
| @Repository("bbs.boardDAO")  public class BoardDAO {      private SqlSession sqlSession;        public int insertBoard(Board dto) throws Exception {          ...      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |
| public class BoardServiceImpl implements BoardService {      @Resource(name="bbs.boardDAO")      private BoardDAO dao;        public int insertBoard(Board dto){}  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @Controller

<http://noritersand.tistory.com/474>

## @RequestMapping

<http://noritersand.tistory.com/475>

## @RequestParam

<http://noritersand.tistory.com/357>

## @SessionAttributes

SessionAttribute annotation은 세션상에서 model의 정보를 유지하고 싶을 경우 사용한다.

|  |  |
| --- | --- |
| @Controller  @SessionAttributes("blog")  public class BlogController {      // 중간생략        @RequestMapping("/createBlog")      public ModelMap createBlogHandler() {          blog = new Blog();          blog.setRegDate(new Date());          return new ModelMap(blog);      }        // 중간생략  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @InitBinder

WebDataBinder를 초기화하는 method를 지정 할 수 있는 설정을 제공한다.

일반적으로 WebDataBinder는 annotation handler 메서드의 command 와 form 객체 인자를 조작하는데 사용된다.

InitBinder 메서드가 필수적으로 반환값을 가질 필요는 없으며, 일반적으로 이런 경우에 void를 선언한다. 특별한 인자는 WebdataBinder와 WebRequest또는 Locale의 조합으로 이루어지며, 이러한 조건이 만족되면 context-specific editors를 등록하는것이 허용된다.

WebdataBinder: WebDataBinder는 web request parameter를 javaBean 객체에 바인딩하는 특정한 DataBinder이다. WebDataBinder는 웹 환경이 필요하지만, Servlet API에 의존적이지 않다. servlet API에 의존적인 ServletRequestDataBinder와 같이 특정한 DaraBinder를 위한 더많은 base class를 제공한다.

RequestMapping: RequestMapping annotation은 web request를 특정한 handler class와 handler method에 mapping하는 역활을 수행한다. 대응하는 handlerMapping(for type level annotation)과 HandlerAdapter(for method level annotation)가 dispatch에 존재한다면, @RequestMapping이 처리된다.

WebRequest: WebRequest는 웹 요청에 대한 Generic interface이다. 주로 일반 request metadata에 generic web request interceptors의 접근을 허용하여 metadata에 대한 처리를 하기 위한 것이지 request 자체를 처리하기 위한 것은 아니다.

Annotation 기반 Controller 에서 ServletContext 구하기:

|  |  |
| --- | --- |
| @Controller  @RequestMapping("/common/download")  public class DownloadController {      @Autowired      private ServletContext sc;        @RequestMapping      public ModelAndView download(@RequestParam("filePath") String filePath) throws Exception {          String path = sc.getRealPath(filePath);          return new ModelAndView("common.download", "downloadFile", new File(path));      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @ModelAttribute

<http://noritersand.tistory.com/365>

## @RequestBody

@RequestBody 어노테이션이 적용된 파라미터는 HTTP Request body의 내용이 전달된다.

참고: <http://java.ihoney.pe.kr/283>

|  |  |
| --- | --- |
| @RequestMapping(value="/test")  public void penaltyInfoDtlUpdate(@RequestBody String body,          HttpServletRequest req, HttpServletResponse res,          Model model, HttpSession session) throws Exception  {        System.out.println(body);  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @ResponseBody

참고: <http://ismydream.tistory.com/140>

클라이언트에 JSON 형식의 값을 응답할 때 유용하다. 메서드에 @ResponseBody를 적용한 후 문자열을 리턴하면 그 값은 HTTP response header가 아니라 HTTP response body에 쓰여진다. 객체를 넘길경우 스프링에 내장된 JACKSON에 의해 문자열로 변환될 것이다.

또한 @ResponseBody가 적용된 컨트롤러는 context에 설정된 resolver를 무시한다.

|  |  |
| --- | --- |
| @RequestMapping("/getVocTypeList")  @ResponseBody  public ArrayList<Object> getVocTypeList() throws Exception {      HashMap<String, Object> vocData = gvocInf.searchVocTypeList();      return (ArrayList<Object>) vocData.get("data");  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

## @PathVariable

URL의 일부를 파라미터 혹은 변수로 사용한다.

|  |  |
| --- | --- |
| package com.sp.ex;    @Controller("ex.exController")  public class ExController{      @RequestMapping(value="/blog/{userId}/main.action", method=RequestMethod.GET)      public String main(HttpServletRequest req                         , @PathVariable String userId) throws Exception    {            req.setAttribute("userId", userId);          return "restful/result";      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

우째 설정할때마다 까묵까묵해서......정리하여 본다.  
  
서블릿컨텍스트에는

<context:component-scan base-package="com.kimddochi" use-default-filters="false">

  <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

기본 패키지에서 @Conponent, @Service, @Repository와 같은 스트레오애노테이션을 제외하고(use-default-filters="false") @Controller로 표현된 클래스만 빈으로 등록하겠다.는 의미  
  
  
루트컨텍스트에서는

<context:component-scan base-package="com.kimddochi">

  <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

서블릿컨텍스트와 반대의 의미.  
기본 패키지내에 모든 스트레오애노테이션으로 표현된 클래스를 빈으로 등록하되 @Controller로 표현된 클래스는 제외(context:exclude-filter)하겠다.는 의미

**어노테이션 Component-scan 분리하기**

어노테이션이 적용된 class(@Controller, @Service, @Repository 를 포함한 class)를

로딩 base-package로 부터 스캔을 할 때 Controller 타입은 제외 시킨다.

제외 시키는 이유는 spring mvc에 관련된 설정 파일은 dispatcher-servlet.xml 에서 스캔 하기 때문이다.

**application-context.xml**

<context:component-scan base-package="com.web.myproject" scoped-proxy="targetClass">

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

**scoped-proxy 는 아래의 3가지를 지정할 수 있다.**

**no**: default, proxy를 생성하지 않는다.

**interface**: JDK Dynamic Proxy를 이용한 Proxy 생성

**targetClass**: CGLIB(code generator library)를 이용한 Proxy 생성

**Dispatcher-servlet.xml**

<context:component-scan base-package="com.web.myproject" use-default-filters="false">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

base-package로 부터 스캔을 할 때, spring mvc에 관련된 Controller 타입만 스캔 한다.

기본 필터의 사용을 false로 지정하면 @Component, @Repository, @Service, @Controller

어노테이션을 사용하는 클래스를 **bean으로 인식하지 않는다.**

### [[Spring] servlet context와 root context의 component-scan의 미묘한 차이점 (잘못하면 404 에러뜸)](http://ddoong2.com/946)

[Java/SpringFramework](http://ddoong2.com/category/Java/SpringFramework) 2014.03.26 12:09

설명보다 제목적기가 더 힘들다.

Bean을 일일이 설정하기 힘들기 때문에 SpringMVC에서는 MVC 구분에 맞춰서

@Controller, @Service, @Repository가 있다. 또한 이와 관계없이 @Component  어노테이션이 존재한다.

나는 보통 @Controller는 servlet context에 설정하고

@Service, @Repository, @Component는 root context에 설정한다.

그래서 servlet-context에 아래와 같이 설정했다.

<context:component-scan base-package="com.ddoong2">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository" />

</context:component-scan>

그리고 root-context에는 다음과 같이 설정했다.

<context:component-scan base-package="com.ddoong2">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Service" />

<context:include-filter type="annotation" expression="org.springframework.stereotype.Repository" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller" />

</context:component-scan>

그런데 이번에 @Component 어노테이션을 사용할 일이 있어서 사용했더니 이개 두번 생성된다.

servlet context와 root context에서 같이 생성되는 현상이 발생했다.

그래서 servlet context를 아래와 같이 설정했다.

<context:component-scan base-package="com.ddoong2">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository" />

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Component" />

</context:component-scan>

그랬더니 페이지를 찾을 수 없다는 404에러가 발생했다.

원인은 컨트롤러 빈이 로딩이 되지 않는것이였다.

원인을 찾아보니 @Controller 어노테이션의 소스에서 찾았다.

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Component

public @interface Controller {

/\*\*

\* The value may indicate a suggestion for a logical component name,

\* to be turned into a Spring bean in case of an autodetected component.

\* @return the suggested component name, if any

\*/

String value() default "";

}

Controller 어노테이션이 Component 어노테이션을 사용하고 있는 것

그리고 최종으로 아래와 같이 수정해서 해결했다.

servlet context

<context:component-scan base-package="com.ddoong2" use-default-filters="false">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller" />

</context:component-scan>

root context

<context:component-scan base-package="com.ddoong2" use-default-filters="false">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Service" />

<context:include-filter type="annotation" expression="org.springframework.stereotype.Repository" />

<context:include-filter type="annotation" expression="org.springframework.stereotype.Component" />

</context:component-scan>

component-scan의 프로퍼티중에 use-default-filters의 기본값은 true 이다.

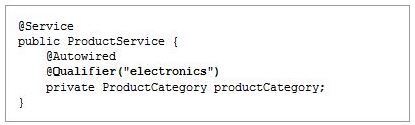
이 부분을 false로 하고 설정을 하면 다른 필터는 로딩되지 않고 순수하게 설정된 부분만 필터링이된다.

@Autowired, @Services 두가지 어노테이션 모두 특정 프로퍼티를 자동으로 연결해주는 작업을 한다.

특정 프로퍼티를 자동으로 연결해 주는 이유는 특정 기능을 수행하기 위해 해당 빈을 사용하기 위함이다.

예를들어, 컨트롤러 클래스에서 BO클래스에 있는 기능을 수행하기 위해서는 위에 있는 두 가지 중 하나를 사용하여 특정 프로퍼티를 자동으로 연결해 주는 것이다.

* @Autowired의 경우

1. @Autowired는 Spring Framework에서 지원하는 의존주입 용도의 어노테이션이다.  (기본적으로 Type-driven Injection 으로 사용된다.)
2. 말 그대로 **타입으로 참조할 빈을 찾았을 때 Injection**이 일어나는데, IoC 컨테이너 내에 같은 타입의 빈이 여러 개 검색되었을 경우, @Qualifier 어노테이션을 사용하여 구분할 수 있다.
3. 기본적으로 @Autowired가 지정된 프로퍼티는 모두 설정되어야 한다. 만약 스프링이 연결할 빈을 찾지 못하면 예외처리가 발생되고, 이 경우를 대비하여 Required 애트리뷰트를 false로 설정하면 스프링이 호환되는 빈을 찾지 못하여도, 예외처리 없이 프로퍼티를 설정하지 않은 채로 남겨둘 것이다.



* @Resource 어노테이션의 경우.

1. @Resource는**bean name으로 의존주입을 하고자 하는 경우 사용된다.** 기본적으로 스프링은 해당 프로퍼티와 일치하는 빈을 찾을 것이고, 혹은 name 속성을 이용해 명시적으로 빈의 이름을 설정할 수 있다.
2. 이름으로 빈 프로퍼티를 자동 연결하려면 JSR-250의 @Resorce 어노테이션을 세터메서드, 생성자, 필드에 적용한다. (JSR-250어노테이션을 사용하려면 JSR 250 의존 라이브러리들을 포함시켜야 한다.)

<참고>

1. <http://dev.anyframejava.org/anyframe/doc/core/3.2.0/corefw/guide/annotation-dependencies.html>

2. 스프링3 레시피

메일

## [Send Mail using Spring JavaConfig (Spring 메일 발송)](http://noviceany.tistory.com/61)

[〃framework/spring](http://noviceany.tistory.com/category/%E3%80%83framework/spring) 2014.06.26 17:23

|  |  |
| --- | --- |
|  |  |

Spring을 이용한 메일 보내기를 하기 위해서 대부분 **JavaMailSender**를 <bean />으로 설정하고

요리조리 코드를 잘 복사 붙여넣기 하여 구현하였다

오늘의 핵심은***xml없는* Spring 프레임워크** 만들기

Spring3.0 부터 Java 코드를 이용한 DI 설정이 가능해 지면서

복잡하다고만 느꼈던 xml 설정을 Java 코드로 모두 대체 할 수 있게 됐다

|  |  |
| --- | --- |
|  | <bean id="mailSender" class ="org.springframework.mail.javamail.JavaMailSenderImpl" > |
|  | <property name="host" value="smtp.gmail.com" /> |
|  | <property name="port" value="587" /> |
|  | <property name="username" value="id@gmail.com" /> |
|  | <property name="password" value="pw" /> |
|  | <property name="javaMailProperties"> |
|  | <props> |
|  | <prop key="mail.smtp.starttls.enable">true</prop> |
|  | </props> |
|  | </property> |
|  | </bean> |

[**view raw**](https://gist.github.com/noviceInAny/7d86480d2ba6a7e6c0e2/raw/320e8cdbcf9d877e058c06dd61997ea2f6291549/sendmail.xml)[**sendmail.xml**](https://gist.github.com/noviceInAny/7d86480d2ba6a7e6c0e2#file-sendmail-xml) hosted with ❤ by [**GitHub**](https://github.com/)

위와 같이 작성됐던 xml 코드를 Java Config 를 이용해서 Java 코드로 사용 가능하다

|  |  |
| --- | --- |
|  | import org.springframework.context.annotation.Bean; |
|  | import org.springframework.context.annotation.Configuration; |
|  | import org.springframework.mail.javamail.JavaMailSender; |
|  | import org.springframework.mail.javamail.JavaMailSenderImpl; |
|  |  |
|  | @Configuration |
|  | public class MailConfig { |
|  |  |
|  | @Bean |
|  | public static JavaMailSender mailSender(){ |
|  |  |
|  | JavaMailSenderImpl mailSender = new JavaMailSenderImpl(); |
|  | mailSender.setHost("host"); |
|  | mailSender.setUsername("username"); |
|  | mailSender.setPassword("password"); |
|  |  |
|  | return mailSender; |
|  |  |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/noviceInAny/2af0c707b559620743b1/raw/9ae68887ef6211fb19918d59a283c85ae86692ee/MailConfig.java)[**MailConfig.java**](https://gist.github.com/noviceInAny/2af0c707b559620743b1#file-mailconfig-java) hosted with ❤ by [**GitHub**](https://github.com/)

Bean을 가져다가 사용하는 컨트롤러는 기존코드와 변화가 없다

단순하게 Bean 설정을 xml에서 Java 코드로 변환한것 뿐이다

|  |  |
| --- | --- |
|  | import javax.mail.internet.MimeMessage; |
|  |  |
|  | import org.springframework.stereotype.Controller; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.mail.javamail.JavaMailSender; |
|  | import org.springframework.mail.javamail.MimeMessageHelper; |
|  | import org.springframework.web.bind.annotation.RequestMapping; |
|  |  |
|  | @Controller |
|  | public class MailController { |
|  |  |
|  | @Autowired |
|  | private JavaMailSender mailSender; |
|  |  |
|  | private String from = "보내는사람"; |
|  | private String subject = "메일제목 (생략가능)"; |
|  |  |
|  | @RequestMapping(value = "/mail") |
|  | public String sendMail() { |
|  |  |
|  | try { |
|  | MimeMessage message = mailSender.createMimeMessage(); |
|  | MimeMessageHelper messageHelper = new MimeMessageHelper(message, true, "UTF-8"); |
|  | messageHelper.setTo("받는사람"); |
|  | messageHelper.setText("메일본문"); |
|  | messageHelper.setFrom(from); |
|  | messageHelper.setSubject(subject); // 메일제목은 생략이 가능하다 |
|  |  |
|  | mailSender.send(message); |
|  | } catch(Exception e){ |
|  | System.out.println(e); |
|  | } |
|  |  |
|  | return "Sucess"; |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/noviceInAny/1d243b6b648a3755cd30/raw/4800fb610da76bdc6654dce9d80143f5ca9af63c/MailController.java)[**MailController.java**](https://gist.github.com/noviceInAny/1d243b6b648a3755cd30#file-mailcontroller-java) hosted with ❤ by [**GitHub**](https://github.com/)

**== 참고 ==**

메일보내기를 구현하는 도중 메일이 죽어라 발송이 되지 않았다

굉장히 단순한 문제였는데 아차 하는 순간 놓치게 되는것 같다

기본중에 기본!!

**1. javax.mail-1.x.x.jar 파일이 필요하다**

- jar 파일이 없으면 빈을 생성하지 못한다. 톰캣 실행도 못해보고 계속 삽질만

- maven을 사용한다면 아래의 dependency를 추가하면 된다

<!-- Java Mail API -->

<dependency>

    <groupId>javax.mail</groupId>

    <artifactId>mail</artifactId>

    <version>1.4.3</version>

</dependency>

**2. 보내는 사람이 없으면 발송이 되지 않는다**

 단순 테스트를 위해 보내는 사람은 가볍게 무시하고 계속 발송을 시도했으나 메일 발송이 되지 않았다

 기존에 있는 코드와 비교해보니 보내는 사람이 빠져 있었고 보내는 사람을 추가하니 그제서야 메일이 정상적으로 발송되었다

- 제목은 생략이 가능하다

MimeMessage message = mailSender.createMimeMessage();

MimeMessageHelper messageHelper = new MimeMessageHelper(message, true, "UTF-8");

messageHelper.setTo("받는사람");

messageHelper.setText("Test!! blabla...");

**messageHelper.setFrom("보내는사람");    // 보내는 사람이 필요하다**

톰캣

## [Tomcat 로컬환경에서 세션이 끊길때(JSessionID 충돌)](http://fruitdev.tistory.com/124)

로컬환경으로 Tomcat을 이용하여 개발하다 보면 종종 세션이 끊길때가 있다.  
물론 Tomcat 1개에 1개의 서비스만 올려서 사용중이라면 그럴일은 없겠지만, 여러 프로젝트를 여러개의 서비스에 올려서 개발할경우 종종 로그인이 끊어지는 경우가 있다.  
  
여러개의 서비스 일지라도 1개의 서비스만 집중적으로 개발할 경우 그럴일은 없는데, 하루에 여러 프로젝트를 손대다보면 세션 끊김이 발생한다.  
  
예를들어 로컬에서 개발시 A라는 프로젝트를 로그인을 한 후 B라는 프로젝트를 로그인 하면 A 프로젝트에서 로그인 했던  
세션이 끊어진다.  
  
해당 문제는 바로 JSessionID가 충돌하여 발생하는 문제라고 볼수 있다.  
JSessionID는 브라우저에서 로그인등을 통해 세션이나 쿠키등이 생성될경우 사용자의 고유정보를 갖고있는데, 이를 통해 서버는 클라이언트가 어떤 녀석인지 구분을 하게 된다.  
  
이 JSessionID는 사용자 IP를 가지고 구분하게 되는데 우리가 보통 로컬환경에서 개발을 할때는 서비스에 따라 포트만 달리하여 개발을 하기 때문에 기존 세션이 없어지는 문제가 생기는 것이다.  
  
다행히 톰캣은 서비스에 따라 JSessionID의 이름을 사용자가 지정할수 있다.  
  
server.xml의 context 부분에 sessionCookieName을 지정해 주면 된다.

|  |
| --- |
| <Context path="" reloadable="false" privileged="true" docBase="D:/Web/WebContent"                sessionCookieName="test\_JSESSIONID" workDir="" crossContext="true" /> |

\* 사실 도메인단위로 운영되는 실 서비스 환경에서는 sessionCookieName을 지정하지 않아도 된다.

Junit 이용 sring config

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = "classpath\*:config/xml/spring-application.xml")  @TransactionConfiguration(transactionManager = "txManager", defaultRollback = **true**)  @Transactional  **public** **class** BaseTestCase {  **protected** Logger logger = Logger.*getLogger*(getClass());    } |
| **public** **class** I18nDtlMapperTester **extends** BaseTestCase {  @Autowired  **private** I18nDtlMapper i18nDtlMapper;  @Test  **public** **void** selectById() {  String code = "button.reset";  String locale = "ko";  I18nDtl i18nDtl = i18nDtlMapper.selectById(code, locale, **false**);  System.***out***.println("--");  System.***out***.println(i18nDtl);  System.***out***.println("--");  }  @Test  **public** **void** update() {  I18nDtl i18nDtl = **new** I18nDtl();  i18nDtl.setCode("antop.jjang");  i18nDtl.setLanguage("ko");  i18nDtl.setMessage("안탑짱");  i18nDtlMapper.update(i18nDtl);  }  @Test  **public** **void** delete() {  String code = "antop.jjang";  String locale = "Ko";  i18nDtlMapper.delete(code, locale);  }  } |
| **public** **class** I18nMapperTester **extends** BaseTestCase {  @Autowired  **private** I18nMapper mapper;  @Test  **public** **void** selectById() {  I18n i18n = mapper.selectById("location.title.street.address");  System.***out***.println("--");  System.***out***.println(i18n);  System.***out***.println("--");  }  @Test  **public** **void** selectByMap() {  MapVo map = **new** MapVo();  map.put("code", "location");  map.put("desc", "아이디");  List<I18n> list = mapper.selectByMap(map);  System.***out***.println("--");  **for** (I18n i18n : list) {  System.***out***.println(i18n);  }  System.***out***.println("--");  }  } |

스케쥴러

quartz

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd"*>  <!-- 1. Cron 대상이 되는 클래스 정의 -->  <bean id=*"baseDataJob"* class=*"org.springframework.scheduling.quartz.JobDetailBean"*>  <property name=*"jobClass"* value=*"com.omnicns.scheduler.BaseDataJob"* />  </bean>  <bean id=*"configDataJob"* class=*"org.springframework.scheduling.quartz.JobDetailBean"*>  <property name=*"jobClass"* value=*"com.omnicns.scheduler.ConfigDataJob"* />  </bean>  <!--  Cron Expression    총 7개의 필드 있고 마지막 필드(년도)는 생략 가능하다    필드이름 허용 값  초(Seconds) 0 ~ 59  분(Minutes) 0 ~ 59  시간(Hours) 0 ~ 23  달의 날짜(Day-of-month) 1 ~ 31  달(Month) 1 ~ 12 or JAN ~ DEC  주의 날짜(Day-of-week) 1 ~ 7 or SUN-SAT  년도(Year) (선택가능) 빈값, 1970 ~ 2099    Cron Expression의 특수문자  Expression 설명 예시  \* 모든 수를 나타냄  - 값의 사이를 의미 \* 10-13 \* \* \* \* 10,11,12,13분에 동작함  , 특정값 지칭 \* 10,11,13 \* \* \* \* 10,11,13분에 동작함  / 값의 증가를 표현 \* 0/5 \* \* \* \* 0분부터 시작해서 5분마다 동작  ? 특별한 값이 없음을 나타냄(day-of-month, day-of-week 필드만 사용)  L 마지막 날을 나타냄(day-of-month, day-of-week 필드만 사용)      참고로 아래의 표는 CronTrigger 에서 작업주기를 설정하는 식의 예와 설명이다.  순서  1. Seconds (0-59) - \* /  2. Minutes (0-59) - \* /  3. Hours (0-23) - \* /  4. Day-of-month (1-31) - \* ? / L W C  5. Month (1-12 or JAN-DEC) - \* /  6. Day-of-week (1-7 or SUN-SAT) - \* ? / L C #  7. Year (optional, empty, 1970-2099) - \* /  식  의미  "0 0 12 \* \* ?"  매일 12시(정오)에실행    "0 15 10 ? \* \*"  매일 오전 10시15분에 실행    "0 15 10 \* \* ?"  매일 오전 10시15분에 실행    "0 15 10 \* \* ? \*"  매일 오전 10시15분에 실행    "0 15 10 \* \* ? 2005"  2005년의 매일 오전 10시15분에 실행    "0 \* 14 \* \* ?"  매일 오후 2시부터 오후 2시59분까지 매분마다 실행    "0 0/5 14 \* \* ?"  매일 오후 2시부터 오후 2시55분까지 매5분마다 실행    "0 0/5 14,18 \* \* ?"  매일 오후 2시부터 오후 2시55분까지 매5분마다 실행 그리고 매일 오후 6시부터 오후 6시55분까지 매5분마다 실행    "0 0-5 14 \* \* ?"  매일 오후 2시부터 오후 2시05분까지 매분마다 실행    "0 10,44 14 ? 3 WED"  3월의 매주 수요일 오후 2시10분과 오후 2시44분에 실행    "0 15 10 ? \* MON-FRI"  매주 월, 화, 수, 목, 금요일 오전 10시15분에 실행    "0 15 10 15 \* ?"  매월 15일 오전 10시15분에 실행    "0 15 10 L \* ?"  매월 마지막날 오전 10시15분에 실행    "0 15 10 ? \* 6L"  매월 마지막 금요일 오전 10시15분에 실행    "0 15 10 ? \* 6L 2002-2005"  2002년부터 2005년까지의 매월 마지막 금요일 오전 10시15분에 실행    "0 15 10 ? \* 6#3"  매월 세번째 금요일 오전 10시15분에 실행  [출처] [Java] Quartz (쿼츠)를 사용하여 자바 스케줄링(scheduling) 하기|작성자 마루아라    \*/  /\*  \* : 모든 값을 의미.  ? : 특정 값을 정하지 않음.  - : 범위를 의미. 예) 0-10이면 0부터 10까지.  , : 값을 추가. 0-10,20-30은 0부터 10까지, 그리고 20부터 30까지.  / : 증분을 의미. 예) 초에 0/15를 사용하면 15초마다(0, 15, 30, 45) .  L : 마지막을 의미. 날짜에 사용하면 월의 마지막 날을 의미.  31, 30 또는 28(윤달에는 29).  W : 주중(weekday)를 의미. 날짜와 같이 쓰면 그 날짜가 주중인 날을 의미.  \*/    -->  <!-- 2. Cron 시간 설정 -->  <bean id=*"baseDataJobTrigger"* class=*"org.springframework.scheduling.quartz.CronTriggerBean"*>  <property name=*"jobDetail"* ref=*"baseDataJob"* />  <property name=*"cronExpression"* value=*"0 0/5 \* \* \* ?"* /> <!-- 5분마다 -->  </bean>  <bean id=*"configDataTrigger"* class=*"org.springframework.scheduling.quartz.CronTriggerBean"*>  <property name=*"jobDetail"* ref=*"configDataJob"* />  <property name=*"cronExpression"* value=*"0/10 \* \* \* \* ?"* /> <!-- 10초마다 -->  </bean>  <!-- 3. Cron 실행 -->  <bean class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>  <property name=*"triggers"*>  <list>  <ref bean=*"baseDataJobTrigger"* />  <ref bean=*"configDataTrigger"* />  </list>  </property>  <property name=*"quartzProperties"*>  <props>  <prop key=*"org.quartz.threadPool.class"*>org.quartz.simpl.SimpleThreadPool</prop>  <prop key=*"org.quartz.threadPool.threadCount"*>3</prop>  <prop key=*"org.quartz.threadPool.threadPriority"*>4</prop>  <prop key=*"org.quartz.jobStore.class"*>org.quartz.simpl.RAMJobStore</prop>  <prop key=*"org.quartz.jobStore.misfireThreshold"*>60000</prop>  </props>  </property>  </bean>  </beans> |
| //@Component  **public** **class** ConfigDataJob **extends** QuartzJobBean{  Logger log = LoggerFactory.*getLogger*(**this**.getClass());  // @Autowired 매번 생성되기떄문에 AutoWired가 되지 않는다 quartzJob  ConfigManager configMng;  //잡실행시 매번 생성된다  **public** ConfigDataJob() {  log.info("ConfigDataJob constructor");  configMng = ApplicationUtil.*getWebApplicationContext*().getBean(ConfigManager.**class**);  }  @Override  **protected** **void** executeInternal(JobExecutionContext context) **throws** JobExecutionException {  **long** time = System.*currentTimeMillis*();  log.info("ConfigDataJob start" + time);  **if**(**null**!=configMng){  configMng.loadSetConfigFile();  }  log.info("ConfigDataJob end" + time);  log.info("ConfigDataJob data" + configMng.getParam("cbis\_title"));  }  } |

object mapping 객체 맵핑 카피등

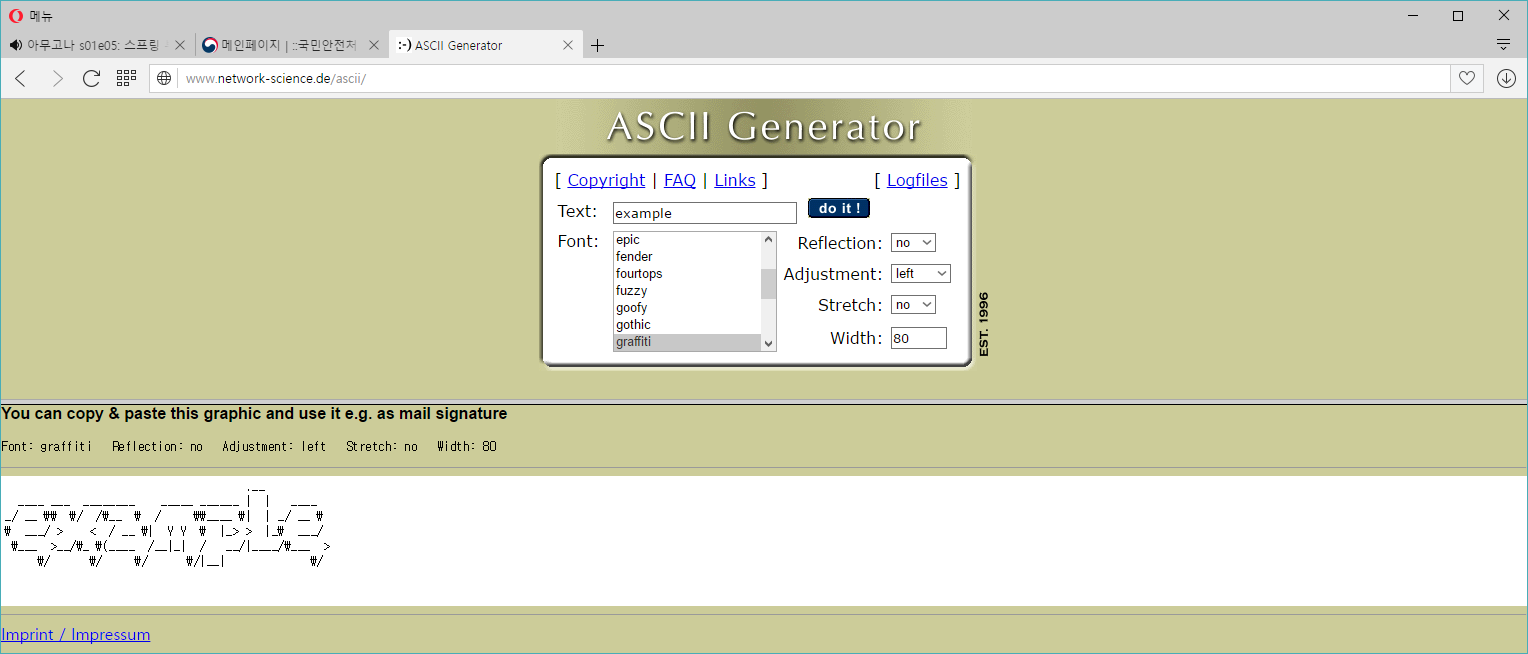
BeanUtils.copyProperties()...

또는 ModelMapper.map(dto,AA.class)

Json처리

|  |
| --- |
| jackson/gson 을 이용한 JSON->Object 변환 간혹 JSON 에 대한 질문글이 올라와서, 언젠가 시간 나면 한 번 정리해서 블로그 같은데 올려야겠다 싶었는데...오늘 짬이 살짝 나서 일부분만 적어봅니다.  간혹 JSON 구문을 Java 의 Bean 형태로 저장하고 싶은 분들이 계실 겁니다. 보통 많은 예제가 있는 것은 gson 인데, 개인적으론 Spring 3.x 을 쓰면서 @ResponseBody 가 처음 제공됐을 때 유일하게 제공된 라이브러리인 Jackson 을 사용하는 코드들로 개발을 계속 해오고 있어서 jackson 을 더 많이 쓰고 있습니다.  서론이 길었는데, 간단한 예제로 사용법을 올립니다. 물론 JSON 구문이라고 되어 있는 json 의 내용은 실제 JSON 구문이 들어간다는 것입니다. 오해 마시길...  import org.codehaus.jackson.map.ObjectMapper; import com.google.gson.Gson;  String json = "JSON 구문";  // jackson ObjectMapper mapper = new ObjectMapper(); UserBean bean = mapper.readValue(json, UserBean.class);  // gson Gson gson = new Gson(); bean = gson.fromJson(json, UserBean.class);  금방 이해되실 겁니다. jackson 은 ObjectMapper, gson 은 Gson 을 이용해서 변환을 한다는 것만 아시면 됩니다. Bean 을 JSON String 으로 변환할 때에도 ObjectMapper 와 Gson 을 이용하고 별도의 데이터 저장이 없기 때문에 static 영역에 선언해도...될까요?  주의하실 것은 저장하려는 Bean(여기서는 UserBean 이라고 선언된)에 JSON 의 키값과 일치하는 Object 가 없을 경우 오류가 발생합니다. 그러므로 문제가 없도록 모든 키를 명시해주는 것이 좋습니다. 물론 내부적으로 배열이 있거나 한다면 List 등을 Bean 안에 선언해주어야 합니다.  그런데, 보통은 이렇게 Bean 형태로 저장을 많이 합니다만(속도 면에서도 가장 빠릅니다) Bean 을 만들기 귀찮거나 굳이 만들어야 하나...싶을 때도 있다는 겁니다. 그래서 Map 형태로 저장하는 것도 알려드립니다.  // jackson Map<String, Object> map = mapper.readValue(json, new TypeReference<Map<String, Object>>() {});  // gson map = gson.fromJson(json, Map.class);  TypeReference 을 이용해서 정의를 하거나 Map 의 형태 그대로 저장도 할 수 있습니다. 그런데 후자의 경우는...사실 groovy 에서만 테스트한 코드라...Java 에서는 오류가 날 수 있네요. 사실 전 위의 TypeReference 을 이용해서 처리하고 있습니다.  이렇게 Bean 으로 변환하는 방법 말고 jackson 에서는 Tree 형태로도 이용할 수 있습니다. gson 은 아직 안써봐서...-\_-;;;  // jackson readTree JsonNode node = mapper.readTree(json); String value1 = node.path("key1").getTextValue();  이런 식으로 XML 의 DOM 와 같이 Tree 형태로 접근이 가능합니다. 속도는 Object 에 mapping 하는것보단 느리지만 별도의 Bean 을 만들지 않아도 되고 Map 형태같이 1차원적이지 않은 데이터를 가볍게 접근하고자 할 때 유용합니다.   Object 을 String 으로 만드는건 더 쉬운데 jackson 은 .writeValueAsString() 로, gson 은 toJson() 으로 String 으로 변환 가능합니다. 단, 한 줄로 죽 나오기 때문에 예쁘게 나오게 하기 위해서 mapper.writerWithDefaultPrettyPrinter().writeValueAsString() 을 이용하시면 됩니다.     누락된 점이나 틀린 내용은 저에게 알려주세요~ |

<http://www.network-science.de/ascii/>



/banner.txt

스프링 스케쥴러 Spring Scheduler

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:tx="http://www.springframework.org/schema/tx"  
 **xmlns:task="http://www.springframework.org/schema/task"** xmlns:util="http://www.springframework.org/schema/util"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  
 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd  
 **http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task-3.0.xsd**  
 http://www.springframework.org/schema/util  
 http://www.springframework.org/schema/util/spring-util.xsd"  
 >

...

<context:component-scan base-package="com.omnicns" >  
 <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  
 <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service"/>  
 <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository"/>  
 </context:component-scan>  
<!-- Spring schduler -->  
<task:scheduler id="gsScheduler" pool-size="10" />  
<task:executor id="gsTaskExecutor" pool-size="10" />  
<task:annotation-driven executor="gsTaskExecutor" scheduler="gsScheduler" />

...</beans>

package com.omnicns.scheduler;  
  
import com.omnicns.boot.BootManager;  
import com.omnicns.boot.service.BootService;  
import com.omnicns.config.ConfigManager;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Component;  
  
@Component  
@Slf4j  
public class Scheduler {  
// @Autowired  
 BootManager bootMng = BootManager.*getInstance*();  
 @Autowired  
 BootService bootService;  
// @Autowired  
 ConfigManager configMng = ConfigManager.*getInstance*();  
 public Scheduler() {  
 }  
  
 //5분마다 (55-5 분 사이는 피한다)  
 @Scheduled(cron = "0 5,10,15,20,25,30,35,40,45,50,55 \* \* \* \*")  
 public void baseDataJob() {  
 long time = System.*currentTimeMillis*();  
 *log*.info("BaseDataJob start " + time);  
 if(null!=bootMng && null!=bootService){  
 bootMng.setRightGroup(bootService.getRightGroup());  
 bootMng.setMsgClass(bootService.getMsgClass());  
 bootMng.setArea(bootService.getArea());  
 bootMng.setMsgTemplate(bootService.getMsgTemplate());  
 }  
 }  
  
 //10초마다  
 @Scheduled(cron = "10,20,30,40,50 \* \* \* \* \*")  
 public void configDataJob() {  
 long time = System.*currentTimeMillis*();  
 *log*.info("ConfigDataJob start " + time);  
 if(null!=configMng){  
 configMng.loadSetConfigFile();  
 }  
 }  
  
}

[http://www.gravatar.com/avatar/1b89ce290b3d05938500e7a2fb7a5ff0?d=identicon&s=40](http://okky.kr/user/info/17612)

[pumkis](http://okky.kr/user/info/17612)

569

3년 전

7

7082

#234292 [Q&A](http://okky.kr/articles/questions) [spring](http://okky.kr/articles/tagged/spring) [스프링](http://okky.kr/articles/tagged/%20%EC%8A%A4%ED%94%84%EB%A7%81)

## jquery ajax+spring에서 @RequestBody 질문입니다.

pom.xml파일의 디펜던시 추가한 사항  
<dependency>  
<groupId>org.codehaus.jackson</groupId>  
<artifactId>jackson-mapper-asl</artifactId>  
<version>1.9.0</version>  
</dependency>  
<dependency>  
<groupId>org.codehaus.jackson</groupId>  
<artifactId>jackson-core-asl</artifactId>  
<version>1.9.0</version>  
</dependency>  
ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ  
현재 json 포맷 xml 설정  
  
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">  
<property name="messageConverters">  
<list>  
<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />  
</list>  
</property>  
</bean>   
  
ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ  
json 포맷   
  
$.ajax  
(  
{  
url: "CMD996.do",  
type: "get",  
contentType: "application/json",  
data: {"ent\_code": "sabun"},  
dataType: "json",   
success: function(data) {  
},  
error: function(data) {  
alert("error : " + data);  
//console.log(data);  
}  
});  
ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ  
  
@RequestMapping(value = "/CMD996.do", method = RequestMethod.GET, consumes="application/json")  
@ResponseBody  
public Command001VO queryListResult(@RequestBody String ent\_code) throws Exception{  
System.out.println("=====");  
log.debug("TestForm :", ent\_code);  
Command001VO vo = new Command001VO();  
vo.setEnt\_code("1135778");  
vo.setSabun("1111");  
System.out.println("=====");  
return vo;  
}  
  
이러한 흐름으로 view 와 controll이 데이터를 주고 받는데   
@ResponseBody에 대한 테스트는 잘되는데  
@RequestBody를 테스트하려니 데이터를 못받아 옵니다..ㅠㅠ  
  
뭐가 잘못된건지 모르겠네요   
  
log 찍어봤는데 ent\_code 값이 프린트가 안되네요  
  
혹시 제가 잘못 설정한게 있나요?  
  
이거로 하루다 보내네요.ㅠㅠ

0

0

### 답변 7

양식의 맨 위

[http://www.gravatar.com/avatar/eb3de8873500c4dd9014d9b36e37aee3?d=identicon&s=40](http://okky.kr/user/info/20523)

[dgkim](http://okky.kr/user/info/20523)

1k

3년 전

ent\_code를 속성으로 가지는 클래스는 없나요?  
  
--  
<http://www.dgkim.net/>

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/88919e72f440dc7ae80aa7c41759d677?d=identicon&s=40](http://okky.kr/user/info/10613)

[커비](http://okky.kr/user/info/10613)

2k

3년 전

@RequestBody 를 @RequestParam("ent\_code")으로 바꾸시면 될듯?  
  
만약에 json데이타를 받고 싶으시면   
  
data: JSON.stringify({"ent\_code": "sabun"}),   
  
이런형태로 하시는 방법도 있습니다.  
  
그럼 데이타 그대로 가져오던가 하듯

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/1b89ce290b3d05938500e7a2fb7a5ff0?d=identicon&s=40](http://okky.kr/user/info/17612)

[pumkis](http://okky.kr/user/info/17612)

569

3년 전

//dgkim  
ent\_code를 속성으로 가지는 클래스는 Command001VO 클래스에 겟터셋터 정의했습니다.  
  
//커비  
커비님께서 말씀하신것처럼 해도 이상하게 json 데이터를 못가져오네요.ㅠㅠ  
json 형태의 데이터를 view에서 받게끔 인터페이스가 정의되어있기 때문에 @RequestParam으로는 받을 수가 없네요.ㅠㅠ

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/eb3de8873500c4dd9014d9b36e37aee3?d=identicon&s=40](http://okky.kr/user/info/20523)

[dgkim](http://okky.kr/user/info/20523)

1k

3년 전

@RequestBody Command001VO ent\_code 형태는 안되나요?  
  
--  
<http://www.dgkim.net/>

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/1b89ce290b3d05938500e7a2fb7a5ff0?d=identicon&s=40](http://okky.kr/user/info/17612)

[pumkis](http://okky.kr/user/info/17612)

569

3년 전

//dgkim  
  
해당 형태도 안되네요, 아무래도 버전 문제일듯한데  
  
현재 spring 3.2.3, json 2.3.x 버전인데 버전을 낮추고 테스트 해야될듯하네요.ㅠㅠ

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/21dd6054118bc54bda694ab21c53f53e?d=identicon&s=40](http://okky.kr/user/info/23917)

[speed1139](http://okky.kr/user/info/23917)

172

3년 전

잭슨 라이브러리로 한번 써보심이.. Jackson Json라이브러리요.

0

양식의 맨 아래

양식의 맨 위

양식의 맨 아래

양식의 맨 위

[http://www.gravatar.com/avatar/1b89ce290b3d05938500e7a2fb7a5ff0?d=identicon&s=40](http://okky.kr/user/info/17612)

[pumkis](http://okky.kr/user/info/17612)

569

3년 전

해결 햇네요.ㅠㅠ   
spring 3.2.3에서 xml 설정부분이 좀 바꼈네요  
  
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">   
<property name="messageConverters">   
<list>   
<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />   
</list>   
</property>   
</bean>   
  
에서 다음과 같이 바꿨어요  
  
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">  
<property name="messageConverters">  
<list>  
<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />  
</list>  
</property>  
</bean>

양식의 맨 아래

--------- filter

|  |
| --- |
| */\*\*  \*   \** ***@프로그램*** *설명 : HTTP 헤더 보안 강화  \*  \** ***@FileName*** *: ClickjackFilter.java  \** ***@Project*** *: ggij  \** ***@Date*** *: 2017. 1. 12.   \** ***@Author*** *: user  \*  \** ***@변경이력*** *:  \*/* public class ClickjackFilter implements Filter {  private static final Logger *LOGGER* = LoggerFactory.*getLogger*(ClickjackFilter.class);   @Override  public void init(FilterConfig filterConfig) throws ServletException  {  *LOGGER*.debug("### Init");  }   @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException  {  HttpServletResponse res = (HttpServletResponse) response;   // 부적절한 서버 설정으로 인한 정보노출 취약점 개선  res.addHeader("X-Frame-Options" , "SAMEORIGIN" );  res.addHeader("X-XSS-Protection" , "1; mode=block");  res.addHeader("X-Content-Type-Options" , "nosniff" );   // 자바스크립트 오작동으로 인한 주석 처리 (향후 재검토 필요)  // res.addHeader("Content-Security-Policy", "default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self';");   // 다른 사이트 허용하려면 아래와 같이 하면 된다.  // res.setHeader("X-Frame-Options", "ALLOW-FROM https://apps.facebook.com");   chain.doFilter(request, response);  }   @Override  public void destroy()  {  *LOGGER*.debug("### Destroy");  } } |

UriComponentsBuilder URI 처리

|  |
| --- |
| UriComponentsBuilder ub = UriComponentsBuilder.*newInstance*();  ub.scheme("http");  ub.host("asd");  ub.path("/vvv");  ub.replaceQueryParam("wgg", "asd");  ub.replaceQueryParam("visualkhh[ggg]", "asd");  ub.queryParam("wgvvg", "asd");   Map<String, String> uriVariables = new HashMap<>();  uriVariables.put("where[asd]", "search");  Object[] uriParam = new Object[]{uriVariables};  http://192.168.11.102:8180/GGIJB602/material/BA0202004/AB0101?draw=1&start=0&length=10&order%5BfoodStfCd%5D=asc&where%5BcntrCd%5D=AB0101&foodCd=BA0202004&\_=1489993479724  *log*.info(ub.toUriString());  String url1 = ub.toUriString();  URI url = new URI(url1);  RestTemplate restTemplate = new RestTemplate();  HttpEntity n = new HttpEntity(new HttpHeaders()); // ResponseEntity result = restTemplate.exchange(url, HttpMethod.GET, n, String.class, uriParam);  ResponseEntity result = restTemplate.exchange(url, HttpMethod.*GET*, n, String.class); |
| public <T> ResponseEntity<T> exchange(Class<T> responseClass) {  UriComponentsBuilder uriBuilder = UriComponentsBuilder.newInstance();  uriBuilder.scheme(this.scheme);  uriBuilder.host(this.host);  uriBuilder.port(this.port);  uriBuilder.path(this.path);  HttpHeaders header = new HttpHeaders();  Optional.ofNullable(this.header).ifPresent((it) -> {  header.putAll(it);  });  RestTemplate restTemplate = null;  if(null != this.clientHttpRequestFactory) {  restTemplate = new RestTemplate(this.clientHttpRequestFactory);  } else {  this.clientHttpRequestFactory = new SimpleClientHttpRequestFactory();  ((SimpleClientHttpRequestFactory)this.clientHttpRequestFactory).setReadTimeout(this.getReadTimeout());  ((SimpleClientHttpRequestFactory)this.clientHttpRequestFactory).setConnectTimeout(this.getConnectTimeout());  restTemplate = new RestTemplate(this.clientHttpRequestFactory);  }   HttpEntity n = null;  if(HttpMethod.GET == this.method && null != this.request && null != this.request.getParameterMap()) {  this.request.getParameterMap().entrySet().stream().forEach((it) -> {  uriBuilder.replaceQueryParam((String)it.getKey(), (Object[])it.getValue());  });  }   if(HttpMethod.GET != this.method && null != this.request && null != this.request.getQueryString() && this.request.getQueryString().length() > 0) {  uriBuilder.replaceQuery(this.request.getQueryString());  }   Optional.ofNullable(this.queryParam).ifPresent((it) -> {  it.entrySet().stream().forEach((sit) -> {  uriBuilder.replaceQueryParam((String)sit.getKey(), new Object[]{sit.getValue()});  });  });  if(HttpMethod.GET == this.method && !TypeUtil.isDataType(this.body) && !TypeUtil.isContainer(this.body)) {  Finder url = (new Finder()).setIncludeSuperClass(true);  url.setAuto(this.body);  url.getFullChildFindItems().entrySet().stream().filter((it) -> {  return null != ((Finder)it.getValue()).getObject() && TypeUtil.isDataType(((Finder)it.getValue()).getObject());  }).forEach((it) -> {  uriBuilder.replaceQueryParam((String)it.getKey(), new Object[]{((Finder)it.getValue()).getObject()});  });  n = new HttpEntity(header);  } else {  header.put("Content-Type", Arrays.asList(new String[]{"application/json;charset=UTF-8"}));  n = new HttpEntity(this.body, header);  }   String url1 = "";  if(null != this.uriBuild) {  url1 = uriBuilder.buildAndExpand(this.uriBuild).toUriString();  } else {  url1 = uriBuilder.toUriString();  }   ResponseEntity result = null;   try {  result = restTemplate.exchange(new URI(url1), this.method, n, responseClass);  return result;  } catch (Exception var9) {  throw new RestClientException(var9.getMessage(), var9);  } } |

### JAX-RS @Produces와 @Consumes

버리야 2009.06.02 10:10

\* Content negotiation

Representation을 서버에서 클라이언트로 받을 때 정보를 넘겨줄때 두가지 전략이 있다. (뷰를 판단하는 방법)

1. 하나는 **다른 URI를 주는것.**

http://www.flyburi.com/user/buri.pdf

http://www. flyburi.com/user/buri.xml

이렇게 다른 파일 확장자를 주는 방법

2. 다른 하나는 같은  URI를 주고 **미디어타입의 리스트를 Accept HTTP request header에 주는 방법.**

http://www.flyburi.com/user/buri 라고 주고 Accept header에 application/pdf or text/xml이라고 주는 방법 이 방법을 content negotiation이라고 한다.

Accept HTTP 헤더로 뷰를 판단하고 서버 응답은 Content-Type를 통해 뷰를 전달한다.

더 자세한 건[위키피디아 참고](http://en.wikipedia.org/wiki/Content_negotiation)

Content negotiation을 JAX-RS에서 쓸려면..

@Consumes, @Produces(요청/응답 미디어 타입 선언) annotation을 사용

리소스 메소드나 리소스 클래스 또는 엔티티 프로바이더에 적용, 생략은 어느 타입(\*/\*)이나 지원을 의미

Accept 헤더 : 응답 결과 받을 수 있는 가능한 미디어 타입 명시

Content-Type 헤더 : 요청 보낸 entity-body에 대한 미디어 타입 -> 즉, request의 body에 대한 타입

**@Produces**

@Path("/hi")

**@Produces("text/plain")**

public class HiResource {

     @GET

     public String doGetAsPlainText() { ... }

     @GET

**@Produces("text/html")**

     public String doGetAsHtml() { ...  }

}

클라이언트에서 헤더에 Accept:text/plain을 보냈다면, 서버에서 @Produces("text/plain")을 명시한 Root Resource인 HiResource에서 doGetAsPlainText() 메소드를 실행하고 "text/plain" 타입을 리턴한다.

클라이언트에서 Accept:text/html 를 보냈다면 doGetAsHtml() 메소드를 실행하고 "text/html" 타입을 리턴한다.

**@Consumes**

@Path("/add")

@POST

**@Consumes(MediaType.APPLICATION\_JSON)**

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

public User add(User input){

try{

User user = User.create(input.getId(), input.getName(), input.getPassword());

userList.add(user);

return user;

}catch(Exception e){

throw new WebApplicationException(e, Response.Status.INTERNAL\_SERVER\_ERROR);

}

}

클라이언트에서 POST를 수행시에 Data를 전달할때 "JSON" 타입으로 전달하고 싶을때

"Content-Type:application/json"  이렇게 주었을때 @Consumes가 JSON 타입을 받아들일 수 있으므로

add()메소드를 실행할 수 있다.

그리고 @Produces가 JSON, XML 형식으로 명시되어 있으므로 두가지 타입으로 리턴할 수 있다.

\* 참고

위의 예제에서 @Produces와 @Consumes의 타입을 명시할때

첫번째에선, **@Produces("text/plain")**단순 텍스트로 넣었고,  두번째에선 **@Consumes(MediaType.APPLICATION\_JSON)** 이렇게 넣었는데 둘다 결과는 동일하다.

다만 변수로 접근한 두번째 방법이 변수이기 때문에 오타를 줄일 수 있는 이점이 있다.

고로 절때 @Produces에선 텍스트로 쓰고 @Consumes에선 변수로 써야하는 법은 없다는거

출처: <http://flyburi.com/461> [버리야 날자]

**6.3.2.4 소비가능한 미디어 타입(Consumable Media Types)**  
소비가능한 미디어 타입의 목록을 지정해서 주요한 매핑을 제한할 수 있다. Content-Type 요청 헤더가 지정한 미디어타입과 일치할 때만 요청이 매칭할 것이다. 예를 들면 다음과 같다.

Java

@Controller

@RequestMapping(value = "/pets", method = RequestMethod.POST, consumes="application/json")

public void addPet(@RequestBody Pet pet, Model model) {

// 구현부는 생략한다

}

소비가능한 미디어 타입 표현식에서 text/plain의 Content-Type을 제외한 모든 요청과 일치하도록 !text/plain처럼 부정문을 사용할 수도 있다.

**Tip**  
consumes 조건은 타입수준과 메서드 수준에서 지원한다. 대부분의 다른 조건과는 달리 타입수준에서 사용했을 때 메서드 수준의 소비가능한 타입이 타입 수준의 소비가능한 타입을 확장하는 게 아니라 오버라이드한다.

**16.3.2.5 생산가능한 미디어 타입(Producible Media Types)**  
생산가능한 미디어 타입의 목록을 지정해서 주요 매핑을 제한할 수 있다. Accept 요청헤더가 이러한 값 중 하나와 일치할 때만 요청이 매칭될 것이다. 게다가 produces 상태를 사용하면 produces 조건에 지정한 미디어 타입과 관련된 응답을 생성하는데 사용한 실제 컨텐트 타입을 보장한다. 예를 들면 다음과 같다.

Java

@Controller

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, produces="application/json")

@ResponseBody

public Pet getPet(@PathVariable String petId, Model model) {

// 구현부는 생략한다

}

consumes에서처럼 생산가능한 미디어타입 표현식에는 text/plain의 Accept 헤더값을 가진 요청을 제외한 모든 요청에 매칭되도록 !text/plain처럼 부정문을 사용할 수 있다.

**Tip**  
produces 조건은 타입수준과 메서드 수준에서 지원한다. 다른 대부분의 조건과는 달리 타입수준에서 사용했을 때 메서드수준의 producible 타입은 타입수준의 producible 타입을 확장하는 게 아니라 오버라이드한다.

**16.3.2.6 요청 파라미터와 헤더값**  
"myParam", "!myParam", "myParam=myValue"같은 요청 파라미터 조건으로 요청에 대한 매칭을 제한할 수 있다. 앞에 두가지는 요청파라미터의 존재/부재 여부를 확인하고 세번째는 특정 파라미터 값을 확인한다. 다음은 요청 파라미터 값 조건에 대한 예제이다.

Java

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET, params="myParam=myValue")

public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

// 구현부는 생략한다

}

}

요청 헤더의 존재/부재 여부에 대한 검사나 지정한 요청 헤더값에 기반한 매칭도 동일하게 이뤄질 수 있다.

Java

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets", method = RequestMethod.GET, headers="myHeader=myValue")

public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {

// 구현부는 생략한다

}

}

**Tip**  
미디어타입 와일드카드를 사용해서 Content-Type와 Accept 헤더값을 일치하도록 할 수 있기는 하지만 대신에 consumes와 produces 조건을 각각 사용하는 것을 권장한다. 이 둘은 해당 목적에 맞게 만들어진 것이다.

하이버네이트 트랜젝션걸기

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"> <!--  http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html\_single/#persistent-classes-dynamicmodels  https://docs.jboss.org/hibernate/orm/3.3/reference/ko-KR/html\_single/ 위링크 예제파일 잘나와있음   --> <!-- <bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" lazy-init="false"> --> <bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean" lazy-init="false">  <property name="dataSource" ref="dataSource" />   <!-- <property name="configLocation"> --> <!-- <value>classpath:/egovframework/hibernate/hibernate.cfg.xml</value> --> <!-- </property> -->  <property name="mappingLocations">  <value>classpath:/hibernate/\*.hbm.xml</value>  </property>     <property name="hibernateProperties">  <props>  <!--hibernate dialect-START-->  <prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop> <!-- <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop> -->  <!--hibernate dialect-END-->  <prop key="hibernate.format\_sql">false</prop>  <prop key="hibernate.show\_sql">false</prop>  <prop key="hibernate.hbm2ddl.auto">none</prop>  <prop key="hibernate.use\_sql\_comments">false</prop>  <prop key="hibernate.jdbc.batch\_size">50</prop>    <prop key="spring.jpa.properties.javax.persistence.validation.mode">false</prop>  <prop key="hibernate.query.startup\_check">false</prop>  <prop key="hibernate.validator.apply\_to\_ddl">false</prop>  <prop key="hibernate.validator.autoregister\_listeners">false</prop>  <prop key="javax.persistence.validation.mode">none</prop>    </props>  </property>   <!-- <property name="annotatedClasses">  <list>  <value>egovframework.rte.rex.cgr.service.CategoryVO</value>   </list>  </property>  --> <property name="packagesToScan">  <list>  <value>com.nhis.\*</value>  </list>  </property>  </bean>      <!--<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">-->  <bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">  <property name="sessionFactory" ref="sessionFactory" />  </bean>    <!-- Proxy FactoryBean -->   <bean id="txProxyTemplate" abstract="true" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">  <property name="transactionManager" ref="transactionManager"/>  <property name="transactionAttributes">  <props>  <prop key="insert\*">PROPAGATION\_REQUIRED</prop>  <prop key="update\*">PROPAGATION\_REQUIRED</prop>  <prop key="delete\*">PROPAGATION\_REQUIRED</prop>  <prop key="\*">PROPAGATION\_REQUIRED,readOnly</prop>  </props>  </property>  </bean>    </beans>  http://mainia.tistory.com/194 |

## [스프링(Spring) 개발 - (11) HandlerMethodArgumentResolver 적용](http://addio3305.tistory.com/75)

[Spring](http://addio3305.tistory.com/category/Spring) 2015.05.13 21:08

이번글에서는 지난글에서 이야기한 HandlerMethodArgumentResolver 라는것을 이야기하려고 합니다.

게시판 목록을 작성한 후, 게시판 글 등록 및 수정, 삭제에 대한 내용이 나와야하지 않나? 라고 생각하는 분들도 많이 있으실겁니다.

사실 HandlerMethodArgumentResolver 가 없어도 개발은 할 수 있지만, 개발을 더욱 편하게 할수 있는 역할을 하기때문에, 먼저 이야기를 합니다.

------------------------------------------------------------------------------------

**1. HandlerMethodArgumentResolver 란?**

HandlerMethodArgumentResolver 는 스프링 3.1에서 추가된 인터페이스다. 스프링 3.1 이전에는 WebArgumentResolver 라는 인터페이스였는데,

스프링 3.1 이후부터 HandlerMethodArgumentResolver 라는 이름으로 바뀌었다.

이것이 하는 역할은 다음과 같다.

스프링 사용 시, 컨트롤러(Controller)에 들어오는 파라미터(Parameter)를 수정하거나 공통적으로 추가를 해주어야 하는 경우가 있다.

예를 들어, 로그인을 한 사용자의 사용자 아이디나 닉네임등을 추가하는것을 생각해보자.

보통 그런 정보는 세션(Session)에 담아놓고 사용하는데, DB에 그러한 정보를 입력할 때에는

결국 세션에서 값을 꺼내와서 파라미터로 추가를 해야한다.

그런 경우가 뭐 하나나 두번 정도 있다면 몰라도, 여러번 사용되는 값을 그렇게 일일히 세션에서 가져오는건 상당히 번거로운 일이다.

HandlerMethodArgumentResolver 는 사용자 요청이 Controller에 도달하기 전에 그 요청의 파라미터들을 수정할 수 있도록 해준다.

자세한건 이제 소스를 보면서 하나씩 살펴보자.

**1. CommandMap 클래스 생성**

request에 담겨있는 파라미터를 Map에 담아주는 역할을 하는 클래스다. 지난 글에서 컨트롤러를 다시한번 살펴보자.

public ModelAndView openSampleBoardList(Map<String,Object> commandMap) throws Exception{ 라고 선언을 했었다.

여기서 Map<String,Object> commandMap에 사용자가 넘겨준 파라미터가 저장되어 있다. (이는 앞으로 그렇게 하겠다는 의미이고, 현재는 저장되지 않는다.)

**그런데 여기서 문제는 HandlerMethodArgumentResolver는 컨트롤러의 파라미터가 Map 형식이면 동작하지 않는다.**

**엄밀히 말을하면, 스프링 3.1에서 HandlerMethodArgumentResolver를 이용하여 그러한 기능을 만들더라도, 컨트롤러의 파라미터가 Map 형식이면 우리가 설정한 클래스가 아닌, 스프링에서 기본적으로 설정된 ArgumentResolver를 거치게 된다.**

항상 그렇게 동작하는것은 아니고, 스프링의 <mvc:annotation-driven/>을 선언하게 되면 위에서 이야기한것처럼 동작하게 된다. (본인은 처음에 이것을 몰라서 진짜 몇날 몇일을 삽질했다.)

따라서 <mvc:annotation-driven/>을 선언하려면 Map을 그대로 사용할 수 없고, 선언하지 않으면 문제는 없다. 그렇지만 앞으로 포스팅할 내용중에는 <mvc:annotation-driven/>을 선언해야 하는 경우가 있기때문에, 여기서는 Map을 대신할 CommandMap을 작성한다.

first 프로젝트의 common 패키지 밑에 common 패키지를 만들고, 다음을 작성하자.

CommandMap.java

[?](http://addio3305.tistory.com/75)

|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | package first.common.common;    import java.util.HashMap;  import java.util.Map;  import java.util.Map.Entry;  import java.util.Set;    public class CommandMap {      Map<String,Object> map = new HashMap<String,Object>();        public Object get(String key){          return map.get(key);      }        public void put(String key, Object value){          map.put(key, value);      }        public Object remove(String key){          return map.remove(key);      }        public boolean containsKey(String key){          return map.containsKey(key);      }        public boolean containsValue(Object value){          return map.containsValue(value);      }        public void clear(){          map.clear();      }        public Set<Entry<String, Object>> entrySet(){          return map.entrySet();      }        public Set<String> keySet(){          return map.keySet();      }        public boolean isEmpty(){          return map.isEmpty();      }        public void putAll(Map<? extends String, ?extends Object> m){          map.putAll(m);      }        public Map<String,Object> getMap(){          return map;      }  } |

클래스는 별다른 부분은 없다. 내부적으로 Map을 하나 생성하고, 그 맵에 모든 데이터를 담는 역할을 한다.

**여기서 중요한점은 절대로 Map을 상속받으면 안된다.**

Map을 상속받게 되면, 우리가 작성할 ArgumentResolver를 거치지 않게 되니 주의하자.

여러가지 메서드들이 보이는데, 거의 대부분은 map의 기본기능을 다시 호출하는것에 지나지 않는다. 보통 가장 많이 사용하는 get, put 메서드만 있더도 큰 문제는 없겠지만, 여기서는 필자가 생각할때 필요한 맵의 기능들을 몇가지 골라서 추가했다.

그리고 다른곳에서는 이 CommandMap을 map과 똑같이 사용할 수 있도록 getMap 메서드를 추가했다.

**2. HandlerMethodArgumentResolver 작성**

first > common 패키지 밑에 resolver 패키지를 작성 후 다음을 작성하자.

CustomMapArgumentResolver.java

[?](http://addio3305.tistory.com/75)

|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39 | package first.common.resolver;    import java.util.Enumeration;    import javax.servlet.http.HttpServletRequest;    import org.springframework.core.MethodParameter;  import org.springframework.web.bind.support.WebDataBinderFactory;  import org.springframework.web.context.request.NativeWebRequest;  import org.springframework.web.method.support.HandlerMethodArgumentResolver;  import org.springframework.web.method.support.ModelAndViewContainer;    import first.common.common.CommandMap;    public class CustomMapArgumentResolver implements HandlerMethodArgumentResolver{      @Override      public boolean supportsParameter(MethodParameter parameter) {          return CommandMap.class.isAssignableFrom(parameter.getParameterType());      }        @Override      public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception {          CommandMap commandMap = new CommandMap();            HttpServletRequest request = (HttpServletRequest) webRequest.getNativeRequest();          Enumeration<?> enumeration = request.getParameterNames();            String key = null;          String[] values = null;          while(enumeration.hasMoreElements()){              key = (String) enumeration.nextElement();              values = request.getParameterValues(key);              if(values != null){                  commandMap.put(key, (values.length > 1) ? values:values[0] );              }          }          return commandMap;      }  } |

이제 하나씩 살펴보자.

HandlerMethodArgumentResolver 인터페이스를 상속(인터페이스도 상속이라고 해야하는지 햇갈리긴 하지만...)하면 두가지 메서드를 반드시 구현해야 하는데, supportsParameter 메서드와 resolveArgument 메서드가 그것이다.

이름에서 알수 있듯이 supportsParameter 메서드는 Resolver가 적용 가능한지 검사하는 역할을 하고, resolverArgument 메서드는 파라미터와 기타 정보를 받아서 실제 객체를 반환한다.

supportsparameter 메서드는 컨트롤러의 파라미터가 CommandMap 클래스인지 검사하도록 하였다.

이를 위해서 추후 Controller의 Map<String,Object> 형식을 CommandMap이라고 변경할 것이다. (잠시후에 다시 좀 더 자세하게 볼 것이다.)

그 다음 중요한것이 resolverArgument 메서드다.

중요한 부분만 살펴보자.

먼저, 아까 정의했던 CommandMap 객체를 생성하였다. (23번 줄)

그 다음으로, request에 담겨있는 모든 키(key)와 값(value)을 commandMap에 저장하였다. (34번 줄)

30번 줄부터 32번 줄은 request에 있는 값을 iterator를 이용하여 하나씩 가져오는 로직이다.

마지막으로 모든 파라미터가 담겨있는 commandMap을 반환하였다. (37번 줄)

**3. CustomMapArgumentResolver 등록**

이제 CustomMapArgumentResolver를 등록하자.

CustomMapArgumentResolver는 root context 영역에 등록이 되어야 한다. 따라서 action-servlet.xml에 등록해야 한다. (root context에 대한 내용은 추후 다시 하겠다.)

action-servlet.xml에 다음과 같이 등록하자.

[?](http://addio3305.tistory.com/75)

|  |  |
| --- | --- |
| 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  32  33  34  35  36 | <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns:context="<http://www.springframework.org/schema/context>"      xmlns:p="<http://www.springframework.org/schema/p>"      xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xmlns="<http://www.springframework.org/schema/beans>"      xmlns:mvc="<http://www.springframework.org/schema/mvc>"      xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>  <http://www.springframework.org/schema/context> <http://www.springframework.org/schema/context/spring-context-3.0.xsd>  <http://www.springframework.org/schema/mvc> <http://www.springframework.org/schema/mvc/spring-mvc.xsd>">        <context:component-scan base-package="first"></context:component-scan>        <mvc:annotation-driven>          <mvc:argument-resolvers>              <bean class="first.common.resolver.CustomMapArgumentResolver"></bean>          </mvc:argument-resolvers>      </mvc:annotation-driven>        <mvc:interceptors>          <mvc:interceptor>              <mvc:mapping path="/\*\*"/>              <bean id="loggerInterceptor" class="first.common.logger.LoggerInterceptor"></bean>          </mvc:interceptor>      </mvc:interceptors>        <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>        <bean class="org.springframework.web.servlet.view.BeanNameViewResolver" p:order="0" />      <bean id="jsonView" class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />        <bean          class="org.springframework.web.servlet.view.UrlBasedViewResolver" p:order="1"          p:viewClass="org.springframework.web.servlet.view.JstlView"          p:prefix="/WEB-INF/jsp/" p:suffix=".jsp">      </bean>  </beans> |

위의 action-servlet.xml은 기존 글에서 작성된것에 CustomMapArgumentResolver를 등록한 내용이다. 실제로 등록을 한 부분은 13~17번째 줄이다.

<mvc:"argument-resolvers> 태그를 이용하여 우리가 만든 CustomMapArgumentResolver의 빈(bean)을 수동으로 등록했다.

4. Controller의 수정 및 테스트

이제 위에서 작성한 것들이 정확히 동작하는지 확인해볼 시간이다.

Controller에 다음을 추가하자.

[?](http://addio3305.tistory.com/75)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @RequestMapping(value="/sample/testMapArgumentResolver.do")  public ModelAndView testMapArgumentResolver(CommandMap commandMap) throws Exception{      ModelAndView mv = new ModelAndView("");        if(commandMap.isEmpty() == false){          Iterator<Entry<String,Object>> iterator = commandMap.getMap().entrySet().iterator();          Entry<String,Object> entry = null;          while(iterator.hasNext()){              entry = iterator.next();              log.debug("key : "+entry.getKey()+", value : "+entry.getValue());          }      }      return mv;  } |

좀 복잡해보일수도 있는데, 간단한 내용이다.

먼저 확인해야 할것은 public ModelAndView testMapArgumentResolver(CommandMap commandMap) throws Exception{ 부분이다.

지난 글에서 우리는 Controller를 public ModelAndView openSampleBoardList(Map<String,Object> commandMap) throws Exception{ 와 같이 작성했었다. (<http://addio3305.tistory.com/72>)

여기서 Map<String,Object>가 방금 만든 CommandMap으로 바뀌었다.

그 후, commandMap에 있는 모든 파라미터를 iterator를 이용하여 출력하였다.

이제 서버를 실행시키고 테스트를 해보자.

주소창에 localhost:8080/first/sample/testMapArgumentResolver.do?aaa=temp 를 입력해보자. 방금 위에서 만든 컨트롤러에 get방식을 이용하여 aaa라는 키로 temp라는 값을 추가하였다. (주소 뒤에 ?를 붙이고 key=value 형식으로 파라미터를 추가할 수 있다.)

이클립스의 콘솔창을 확인하면 다음과 같은 결과를 볼 수 있다.



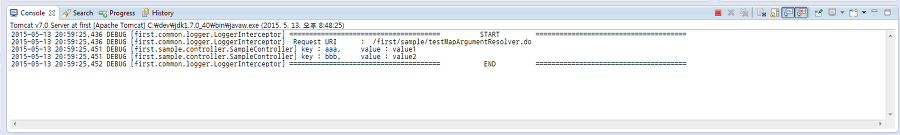
key : aaa, value : temp라는 로그를 보자.

아까 위에서 우리가 get으로 전송한 aaa라는 키와 temp라는 값이다.

위와 같은 결과가 나오면 정상적으로 CustomMapArgumentResolver가 등록된 것이다.

이번에는 두개의 키와 값을 전송해보자.

localhost:8080/first/sample/testMapArgumentResolver.do?aaa=value1&bbb=value2 라고 입력해보자.



정상적으로 aaa와 bbb에 해당하는 값 value1과 value2가 commandMap에 담겨져서 출력됨을 알 수 있다.

이것으로 HandlerMethodArgumentResolver에 대한 내용을 마무리한다.

------------------------------------------------------------------------------------

이번글은 조금 짧았지만, 중요한 내용에 대해서 이야기를 했습니다.

다음글에서는 이제 게시글 작성 및 수정, 삭제에 대해서 이야기를 할 예정입니다.

사용자가 게시글을 작성하면, 그 데이터는 request에 담겨서 서버로 전송이 되고, 서버에서는 그 데이터를 DB에 입력 및 수정을 하게 됩니다.

오늘 내용은 그 부분을 편하게 하기 위해서 먼저 설명을 했습니다.

만약 MapArguemtResolver를 등록하지 않았다면, 컨트롤러에서 request.getParameter 메서드 등을 이용하여 하나하나씩

값을 가져와야 하기 때문에 상당히 번거롭고 코드가 길어지게 됩니다.

(사실 Annotation을 이용하여 그러한 과정을 생략할 수 있지만, 위에서 이야기한것처럼 값을 수정하거나 추가하기 위해서

MapArguementResolver를 등록했습니다.)

그럼 다음글에서 뵙겠습니다.

출처: <http://addio3305.tistory.com/75> [흔한 개발자의 개발 노트]

### [블로그개발\_14] 스프링 WebMvc 설정 (Interceptor, ArgumentResolver)

<http://millky.com/@origoni/post/1177>

블로그개발 시리즈 - 다른글 : <http://millky.com/@origoni/folder/30/post/list>

* [[블로그개발\_01] STS로 Spring Boot 웹 프로젝트 시작하기](http://millky.com/@origoni/post/1100)
* [[블로그개발\_02] Spring Boot 와 Spring Data (JPA) 를 이용한 간단한 DB 입출력](http://millky.com/@origoni/post/1101)
* [[블로그개발\_03] STS에서 GitHub에 있는 메이븐 프로젝트 받아서 시작해보기. (+ gradle to maven)](http://millky.com/@origoni/post/1140)
* [[블로그개발\_04] 스프링 부트(Spring Boot)의 세가지 뷰 (JSP, Velocity, Thymeleaf)](http://millky.com/@origoni/post/1144)
* [[블로그개발\_05] STS에서 GitHub에 있는 메이븐 프로젝트 받아서 시작해보기. (for Windows User)](http://millky.com/@origoni/post/1145)
* [[블로그개발\_06] webjars를 이용한 bootstrap, jquery 적용해보기 (Spring Boot)](http://millky.com/@origoni/post/1147)
* [[블로그개발\_07] Blog theme 적용해보기 (+bower, +webjars)](http://millky.com/@origoni/post/1152)
* [[블로그개발\_08] 블로그 글 저장 및 표시 해보기 (Spring Boot + Spring Data JPA + H2)](http://millky.com/@origoni/post/1155)
* [[블로그개발\_09] WYSIWYG 에디터 붙이기](http://millky.com/@origoni/post/1162)
* [[블로그개발\_10] 글에 부제목 추가 (Post Entity 모델링 +lombok, @Valid)](http://millky.com/@origoni/post/1164)
* [[블로그개발\_11] 글 수정, 삭제... (+Date 부분 변경)](http://millky.com/@origoni/post/1167)
* [[블로그개발\_12] 소셜 로그인 붙이기 (spring-social-facebook)](http://millky.com/@origoni/post/1170)
* [[블로그개발\_13] 글 목록 페이징 및 정렬 (SpringData Pageable, Page)](http://millky.com/@origoni/post/1171)
* [[블로그개발\_14] 스프링 WebMvc 설정 (Interceptor, ArgumentResolver)](http://millky.com/@origoni/post/1177)
* [[블로그개발\_15] 카테고리 추가 (@ManyToOne, stream())](http://millky.com/@origoni/post/1181)

라이브 데모 : <http://blog.millky.com/post/list>

자바 웹 개발 시작하기 : <http://www.slideshare.net/origoni/presentations>

프로젝트가 끝나고 안정화가 되었다. 일정을 맞추느라 마지막 열을쯤 안하던 야근을 하니 몸이 ㅠㅠ

... 오래간만에 글을 적어본다.

오늘은 컨트롤러 도우미 둘을 사용해 보려고 한다.

지난시간 컨트롤러를 보면.. 사용자 정보를 가지고 오기위해 매우 복잡해 졌다.

<https://github.com/origoni/Spring-Blog/blob/v0.0.13/src/main/java/com/millky/blog/presentation/controller/PostController.java>

컨트롤러에 사용자 정보를 가지고 오는 메서드가 추가되고..

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | private User getConnect() {      Connection<Facebook> connection = connectionRepository.findPrimaryConnection(Facebook.class);      if (connection == null) {          return null;      }      ConnectionData data = connection.createData();      return new User(data.getProviderUserId(), data.getDisplayName());  } |

사용한는 곳마다 실행을 시켰다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @RequestMapping(value = "/write", method = RequestMethod.POST)  public String write(@Valid Post post, BindingResult bindingResult) {      User user = getConnect();        ...  } |

그리고 뷰단에 내려주기 위해.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @RequestMapping("/{id}")  public String view(Model model, @PathVariable int id) {      ...        User user = getConnect();      model.addAttribute("user", user);        return "post/post";  } |

이런식의 작업도 하였다.

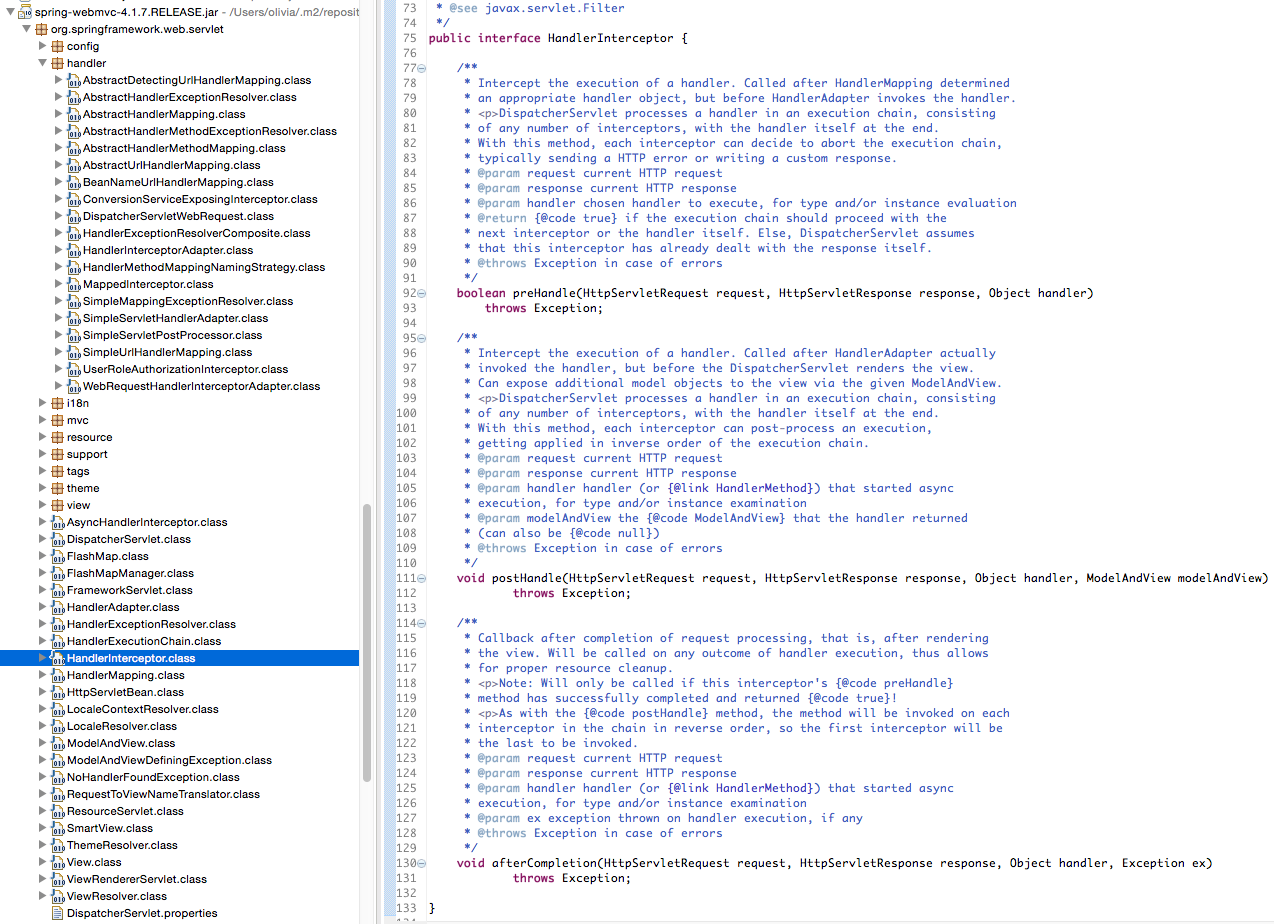
뭐 한둘 두줄 늘어나는것이긴 하지만. 이런것들이 모이면 점점 복잡도가 증가한다.

스프링에서는 아주 깔끔하게 뺄 수 있는 방법이 있다.

#### 우선은 HandlerInterceptor 확인해보자.

HandlerInterceptor 는 컨트롤러 실행 전후에 공통적으로 해야 하는 작업을 처리 해준다.

인터페이스는 요렇게 생겼다.



우리는 preHandle 만 사용할 예정이어서 HandlerInterceptorAdapter 를 사용하였다.

<https://github.com/origoni/Spring-Blog/blob/v0.0.14/src/main/java/com/millky/blog/application/aop/UserSessionInterceptor.java>

|  |  |
| --- | --- |
| 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  32  33  34  35 | package com.millky.blog.application.aop;    import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.springframework.social.connect.Connection;  import org.springframework.social.connect.ConnectionData;  import org.springframework.social.connect.ConnectionRepository;  import org.springframework.social.facebook.api.Facebook;  import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;    import com.millky.blog.domain.model.UserSession;    public class UserSessionInterceptor extends HandlerInterceptorAdapter {        private ConnectionRepository connectionRepository;        public UserSessionInterceptor(ConnectionRepository connectionRepository) {          this.connectionRepository = connectionRepository;      }        public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {            Connection<Facebook> connection = connectionRepository.findPrimaryConnection(Facebook.class);            if (connection != null) {              ConnectionData data = connection.createData();                request.setAttribute("user",                      new UserSession(data.getProviderUserId(), data.getImageUrl(), data.getDisplayName()));          }            return true;      }  } |

간단하게 살펴보면.

리퀘스트 요청이 있을때마다. 사용자 정보를 가지고 와서 있으면 user에 넣어준다. (음 우선 호환성을 위해 user 를 그냥 사용하였고. 다음번에 가독성을 높이기 위해 변경해보자.)

이전의

|  |  |
| --- | --- |
| 1  2 | User user = getConnect();  model.addAttribute("user", user); |

를 대신하는 부분이다.

아직 컨트롤러가 많이 않아서 느끼기 힘들수도 있지만. 잘 활용하면 많은 부분은 줄여준다.

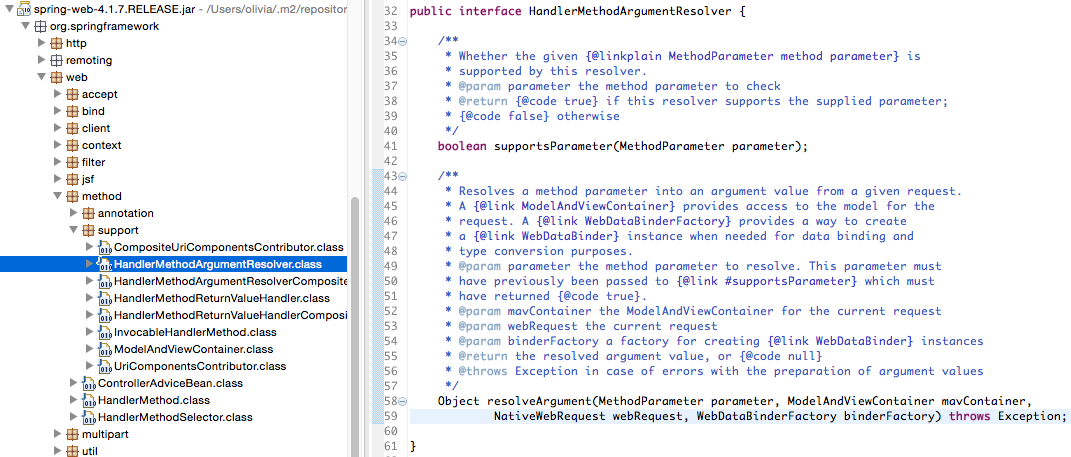
#### 다음으로 ArgumentResolver를 확인해 보자.

뷰에서도 사용자 정보가 필요하지만 컨트롤러에서도 사용할 일이 많다.

가끔... HandlerInterceptor 에서 뷰에 내려준 정보를 이용해서... 사용자 정보등을 넣으시는 분이 계신데. 절대로.. 절대로 뷰에서 넘어오는 값은 믿어서는 안된다.

get 요청이야. 원래 조작하라고 있는거소. post 도 정말 간단한 툴만 가지고 조작이 가능하다.

(그래서... 글 쓸때 entity 바로 담아서 오지 말고... command 같은것을... 나중에 만들어 보겠다.)



HandlerMethodArgumentResolver 를 구현하여 만들었다.

<https://github.com/origoni/Spring-Blog/blob/v0.0.14/src/main/java/com/millky/blog/application/aop/UserSessionArgumentResolver.java>

|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38 | package com.millky.blog.application.aop;    import org.springframework.core.MethodParameter;  import org.springframework.social.connect.Connection;  import org.springframework.social.connect.ConnectionData;  import org.springframework.social.connect.ConnectionRepository;  import org.springframework.social.facebook.api.Facebook;  import org.springframework.web.bind.support.WebDataBinderFactory;  import org.springframework.web.context.request.NativeWebRequest;  import org.springframework.web.method.support.HandlerMethodArgumentResolver;  import org.springframework.web.method.support.ModelAndViewContainer;    import com.millky.blog.domain.model.UserSession;    public class UserSessionArgumentResolver implements HandlerMethodArgumentResolver {        private ConnectionRepository connectionRepository;        public UserSessionArgumentResolver(ConnectionRepository connectionRepository) {          this.connectionRepository = connectionRepository;      }        @Override      public boolean supportsParameter(MethodParameter parameter) {          return UserSession.class.isAssignableFrom(parameter.getParameterType());      }        @Override      public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer,              NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception {          Connection<Facebook> connection = connectionRepository.findPrimaryConnection(Facebook.class);          if (connection == null) {              return null;          }          ConnectionData data = connection.createData();          return new UserSession(data.getProviderUserId(), data.getImageUrl(), data.getDisplayName());      }  } |

두개의 매서드가 있는데.

supportsParameter 는 파라미터를 확인해서 Resolver 적용 가능한지 확인을 한다. (한번 확인된 것은 캐시 한다. [HandlerMethodArgumentResolverComposite](https://github.com/spring-projects/spring-framework/blob/master/spring-web/src/main/java/org/springframework/web/method/support/HandlerMethodArgumentResolverComposite.java) 요기 참고)

resolveArgument 는 지원되는 파라미터가 들어오면 작업을 하여 생성된 객체를 리턴 한다.

뭐 말이 써놓고 보니 어려운데.

간단하게 말하면. 파라미터로 들어온것중 UserSession 가 있으면.. connection 정보를 가지고 와서 new UserSession(...) 이렇게 생성해서 리턴한다.

글쓰기 부분을 보자.

<https://github.com/origoni/Spring-Blog/blob/v0.0.14/src/main/java/com/millky/blog/presentation/controller/PostController.java>

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @RequestMapping(value = "/write", method = RequestMethod.POST)  public String write(@Valid Post post, BindingResult bindingResult, UserSession user) {        if (bindingResult.hasErrors()) {          return "post/form";      }        post.setRegDate(LocalDateTime.now());      post.setUserId(user.getProviderUserId());      post.setName(user.getDisplayName());      return "redirect:/post/" + postDao.save(post).getId();  } |

이전에

|  |  |
| --- | --- |
| 1 | User user = getConnect(); |

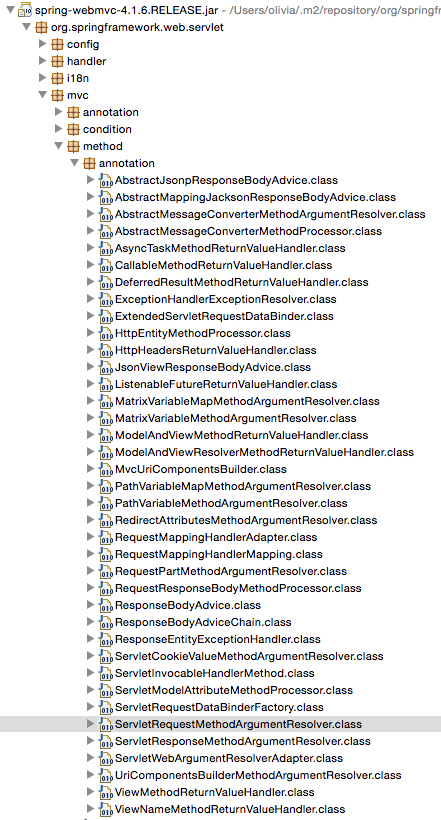
이런식으로 사용자 정보를 가지고 오던 부분이

|  |  |
| --- | --- |
| 1 | write(..., UserSession user) |

이렇게 가지고 올 수 있게 되었다.

UserSession은 UserSessionArgumentResolver에 정의 하였기에 바로 사용할 수 있다.

우리는 사용자 정보 넣은것 하나 만들었지만 이미 만들어진 많은 ArgumentResolver 들이 있다.



컨트롤러가 많이 깔끔해 짐을 느낄 수 있다.

(아.. 아직 뭔가 너무 많다. 점점 이동할 예정이니 기다려 보자 ^^;)

그리고. 이 두 클래스의 패키지 위치가 ...aop 인것도... 무시하자.

필터라고 해야 하나, 인터셉터라고 해야하나.. 뷰(presentation/support)에 넣어야 하나 아직도 고민이 많다.

(사실 a로 시작해서 가장 위로 올라가서 했다... 는 아니다. ^^; 고민 중이다.)

#### 마지막으로 설정 부분을 보자.

WebMvcConfigurerAdapter 를 상속받아 만들었다.

<https://github.com/origoni/Spring-Blog/blob/v0.0.14/src/main/java/com/millky/blog/application/configuration/WebMvcConfig.java>

|  |  |
| --- | --- |
| 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 | package com.millky.blog.application.configuration;    import java.util.List;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Configuration;  import org.springframework.social.connect.ConnectionRepository;  import org.springframework.web.method.support.HandlerMethodArgumentResolver;  import org.springframework.web.servlet.config.annotation.InterceptorRegistry;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;    import com.millky.blog.application.aop.UserSessionArgumentResolver;  import com.millky.blog.application.aop.UserSessionInterceptor;    @Configuration  public class WebMvcConfig extends WebMvcConfigurerAdapter {        @Autowired      private ConnectionRepository connectionRepository;        @Override      public void addInterceptors(InterceptorRegistry registry) {          registry.addInterceptor(new UserSessionInterceptor(connectionRepository));      }        @Override      public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {          argumentResolvers.add(new UserSessionArgumentResolver(connectionRepository));      }  } |

역시 Adapter 라서 다른 많은 설정들이 있지만 일단 addInterceptors, addArgumentResolvers 만 사용한다.

connectionRepository를 여기서 주입해 주고 있다는 점을 확인 하자.

오늘은 여기까지..

오래간만에 글을 쓰려니... 뭔가 좀 감을 잃은 느낌이다 ㅠ

개인적인 일이 조금 있어 피곤하기도 하도...

아까 커밋할때 깃헙도 꼬여서;;; 지금 뭔가가 이상하다. 내일 다시 봐야지 ㅠㅠ

다음시간에는 카테고리 기능을 넣어 볼까 생각 중이다.

감사합니다.

오늘 작업한 코드는 <https://github.com/origoni/Spring-Blog/tree/v0.0.14> 여기를 참고 부탁드린다.

라이브 데모 : <http://blog.millky.com/post/list>

변경사항은 : <https://github.com/origoni/Spring-Blog/commit/>

깃헙 : <https://github.com/origoni/Spring-Blog>

프로젝트 다운받아 시작하는 방법은(for Windows User)? <http://millky.com/@origoni/post/1145>

블로그개발 시리즈 - 다른글 : <http://millky.com/@origoni/folder/30/post/list>

역순 정렬도 있어요.. : <http://millky.com/@origoni/folder/30/post/list?sort=regDate,asc>

자바 웹 개발 시작하기 : <http://www.slideshare.net/origoni/presentations>