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()); |

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  @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> |

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>  <sec:concurrency-control max-sessions=*"1"* expired-url=*"/"*/>  </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=*"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> |

다국어

|  |
| --- |
| <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> |

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> |

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

|  |
| --- |
| @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="/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);  } |

**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.

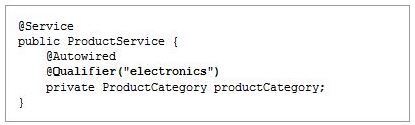
이 부분을 false로 하고 설정을 하면 다른 필터는 로딩되지 않고 순수하게 설정된 부분만 필터링이된다.

@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