
Spring 後端檢核

Spring 4 & Hibernate 4 使用 Annotation 建立後端檢核

Jean Hu - 20 October 2015

前言	2
目的	2
開始前準備	2
Hibernate配置	2
一. pom.xml 加入相關 library	2
二. Hibernate Configuration	3
三. Properties File	4
使用Spring AOP機制插入後端檢核	4
一. pom.xml 加入 Bean Validation API 和 Hibernate Validator	4
二. KungFu Model 中設定屬性驗證	4
三. 將錯誤訊息(message) 寫入多國語言檔中	5
四. 自定義 Validation	6
五. KungFu Controller加入後端驗證	8
六. 前端利用 Spring 的 tag library 做 Error binding	9
七. 加入 Spring AOP 機制	9
參考來源	12

前言

- 使用 Annotation 提早在編譯期間處理錯誤及減少配置文件。
- Spring AOP 機制管理重複使用的功能。
- 開發工具使用 spring-tool-suite 3.6.2.RELEASE，以下簡稱 STS。
- 本文件適用於 Spring MVC 開發。

目的

- 管理 Model 建立關聯性。
- 後端檢核 Model。
- Spring 與 Hibernate 整合將 transaction 交由 Spring 控管。

開始前準備

本架構建立於以下版本的環境：

- JDK7
- STS 3.6.2.RELEASE
- Tomcat 7
- MySQL 6.3
- Maven3.1 (STS 3.6.2中內建)

Hibernate配置

一. pom.xml 加入相關 library

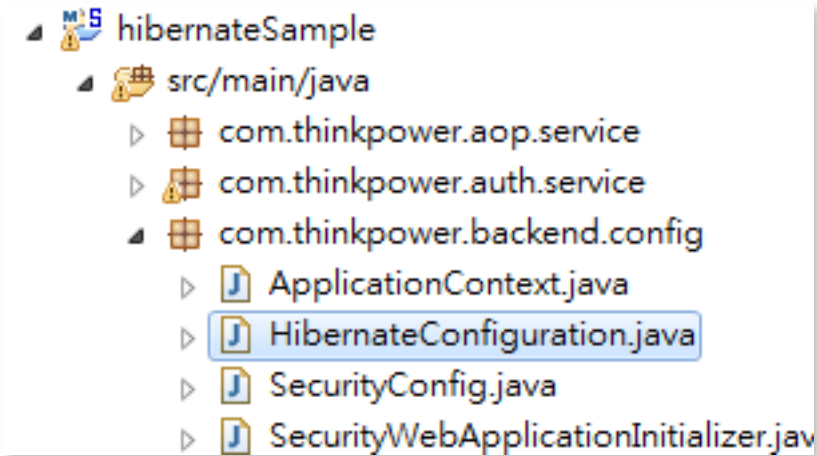
```
<!-- Hibernate -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate.version}</version>
</dependency>

<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
</dependency>

<!-- Hibernate / JPA -->
<hibernate.version>4.3.6.Final</hibernate.version>

<!-- MySQL -->
<mysql.version>5.1.31</mysql.version>
```

二. Hibernate Configuration



```
package com.thinkpower.backend.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;
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.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate4.HibernateTransactionManager;
import org.springframework.orm.hibernate4.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement
@ComponentScan({ "com.thinkpower.dao.hibernate" })
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfiguration {

    @Autowired
    private Environment environment;

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan(new String[] { "com.thinkpower.model" });
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));
        dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));
        dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));
        dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));
        return dataSource;
    }

    private Properties hibernateProperties() {
        Properties properties = new Properties();
        properties.put("hibernate.dialect", environment.getRequiredProperty("hibernate.dialect"));
        properties.put("hibernate.show_sql", environment.getRequiredProperty("hibernate.show_sql"));
        properties.put("hibernate.format_sql", environment.getRequiredProperty("hibernate.format_sql"));
        properties.put("hibernate.generate_statistics", environment.getRequiredProperty("hibernate.generate_statistics"));
        return properties;
    }

    @Bean
    @Autowired
    public HibernateTransactionManager transactionManager(SessionFactory s) {
        HibernateTransactionManager txManager = new HibernateTransactionManager();
        txManager.setSessionFactory(s);
        return txManager;
    }
}
```

hibernate.show_sql: 顯示查詢操作資料庫的SQL
hibernate.format_sql: SQL排版
hibernate.generate_statistics: 匯出操作統計資料

三. Properties File

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/tku
jdbc.username = tku
jdbc.password = tku123
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true
hibernate.generate_statistics = true
```

使用Spring AOP機制插入後端檢核

一. pom.xml 加入 Bean Validation API 和 Hibernate Validator

```
<!-- Validation -->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>${javax.validation.version}</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>${hibernate.validation.version}</version>
</dependency>

<!-- Hibernate / Javax Validation -->
<hibernate.validation.version>5.2.2.Final</hibernate.validation.version>
<javax.validation.version>1.1.0.Final</javax.validation.version>
```

二. KungFu Model 中設定屬性驗證

```

@Entity
@Table(name="kung_fu")
public class KungFu {
    @Id
    @Column(name="kung_fu_id")
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int kung fu id;

    @NotBlank
    @Size(min = 2, max = 20)
    private String title;

    @NotBlank
    @Size(max = 5)
    private String type;

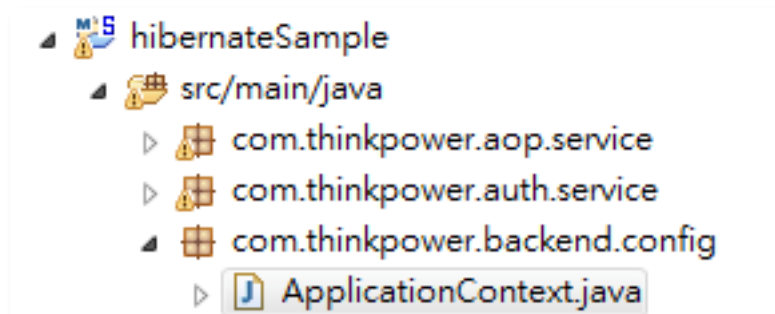
    @NotNull
    @Min(value = 0, message = "最小攻擊力應大於 [0]")
    @Max(value = 99999, message = "最大攻擊力應小於 [99999]")
    private Integer hurt;

    @NotNull
    @DecimalMax(value = "99999.99", message = "最大耗氣真氣應小於 [99999.99] ")
    @DecimalMin(value = "10.00", message = "最小耗氣真氣應大於 [10.00] ")
    private Integer energy;
}

```

- @NotNull：檢查是否不為 null。
- @NotEmpty：檢查 Collection 是否不為空。
- @NotBlank：檢查字串 trim 後長度大於 0。
- @DecimalMin：檢查被註解的元素是否為數字，且大於等於註解內的值。
- message：自訂義的錯誤訊息。

三. 將錯誤訊息(message)寫入多國語言檔中
ApplicationContext.java增加多國語言設定。



```

@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    messageSource.setBasename("i18n/messages");
    messageSource.setUseCodeAsDefaultMessage(true);
    return messageSource;
}

```

messages_zh_TW.properties 新增錯誤訊息。錯誤訊息的 key pattern 為 {ValidationClass}.{modelName}.{field}

Size	[0] 需介於 [2] 與 [1] 字元之間
KungFuTitle	失傳已久...無法再次取得
NotBlank	[0] 不可為空!
NotNull.kung_fu.title	[功夫名號] 為必填!

四. 自定義 Validation

定義 Annotation (@KungFuTitle) :

```

@Documented
@Constraint(validatedBy = KungFuTitleValidator.class)
@Target( { ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface KungFuTitle {
    String message() default "{KungFuTitle}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
    @interface List {
        KungFuTitle[] value();
    }
}

```

- @Documented：是否再使用此 Annotation 的物件文件上顯示此 Annotation，若使用 Eclipse 產生文件似乎沒有影響，不論是否使用皆會顯示。
- @Constraint：用來作為驗證的 class，此 class 必須實作 javax.validation.ConstraintValidator。
- @Target：此 Annotation 可以用在何處，如欄位、方法等等，以此例來說，若將此 Annotation 加在 class 上會顯示錯誤。
 - TYPE：適用 class, interface, enum。
 - FIELD：適用 field。

- METHOD：適用 method。
- PARAMETER：適用 method 上之 parameter。
- CONSTRUCTOR：適用 constructor。
- LOCAL_VARIABLE：適用區域變數。
- ANNOTATION_TYPE：適用 annotation 型態。
- PACKAGE：適用 package。
- @Retention：告知編譯器如何處理 Annotation。
- SOURCE：編譯器處理完 Annotation 資訊後就沒事了。
- CLASS：編譯器將 Annotation 儲存於 class 檔中。
- RUNTIME：編譯器將 Annotation 儲存於 class 檔中，可由 VM 讀入。
- message()：驗證錯誤後的提示信息。
- groups()：分組驗證，同一個待驗證對象在不同驗證方法下支援不同種驗證規則。
- payload()：驗證負載，Client端可以客製化不同的payload物件限制，此屬性不會被API本身用到。

定義驗證用的Validator (KungFuTitleValidator.java)：如下圖所示，在 isValid 方法內撰寫驗證方法，若驗證錯誤則回傳false 驗證正確則回傳true，驗證錯誤時將會自動套用加入Annotation時傳入的message參數。

```
public class KungFuTitleValidator implements ConstraintValidator<KungFuTitle, String> {  
    private String[] lostKungFu = {"野蠻", "野蠻"};  
  
    @Override  
    public void initialize(KungFuTitle constraint/Annotation) {  
    }  
  
    @Override  
    public boolean isValid(String value, ConstraintValidatorContext context) {  
        for(String word : lostKungFu) {  
            if(value.contains(word)) {  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

KungFu Model 中設定自定義的驗證Annotation：


```

@Entity
@Table(name="kung_fu")
public class KungFu {
    @Id
    @Column(name="kung fu id")
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int kung_fu_id;

    @NotBlank
    @Size(min = 2, max = 20)
    @KungFuTitle
    private String title;
}

```

五. KungFu Controller加入後端驗證

```

@Controller
@RequestMapping("/kungfu")
public class KungFuController {

    Logger logger = LoggerFactory.getLogger(KungFuController.class);

    @Autowired
    KungFuService kungFuService;

    /**
     * 新增
     * @return
     */
    @RequestMapping("/add")
    public String addKungfu(HttpSession session, Model model){
        Kungfu kungfu = new Kungfu();
        model.addAttribute("kungfu", kungfu);
        return "kungfu.add";
    }

    /**
     * 新增成功
     * @return
     */
    @RequestMapping("/addSuccess")
    public String addKungfuSuccess(
        @Valid @ModelAttribute("kungfu") KungFu kungfu,
        BindingResult result,
        Model model){

        if (result.hasErrors()) {
            logger.debug("addKungFu() Validate Error!");
            return "kungfu.add";
        }
        kungFuService.insertKungfu(kungfu);
        model.addAttribute("kungfu", kungfu);
        return "kungfu.addSuccess";
    }
}

```


- @Valid：驗證KungFu物件。
- BindingResult：帶有“前一個”命令或表單對象（前一方法參數）的驗證結果，所以“必須”緊接在被驗證的參數後方，若任何屬性為背了約束，則會丟出error，可以用hasErrors來判斷是否存在error。

六. 前端利用 Spring 的 tag library 做 Error binding

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<div align="center">
    <form:form action="addSuccess" method="post" commandName="kungfu">
        <table border="1" class="form">
            <tr>
                <td colspan="2" align="center"><spring:message code="kung_fu.title" /></td></tr>
            <tr>
                <td><spring:message code="kung_fu.title" /></td>
                <td><form:input path="title" /><form:errors path="title" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.type" /></td>
                <td><form:input path="type" /><form:errors path="type" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.hurt" /></td>
                <td><form:input path="hurt" /><form:errors path="hurt" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.energy" /></td>
                <td><form:input path="energy" /><form:errors path="energy" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.recover" /></td>
                <td><form:input path="recover" /><form:errors path="recover" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.teacher" /></td>
                <td><form:input path="teacher" /><form:errors path="teacher" cssClass="error" /></td>
            </tr>
            <tr>
                <td><spring:message code="kung_fu.remark" /></td>
                <td><form:input path="remark" /><form:errors path="remark" cssClass="error" /></td>
            </tr>
            <tr>
                <td colspan="2" align="center">
                    <input type="submit" value="SAVE" />
                    <input type="button" value="cancel" onClick="location.href = '/dujinyang/kungfu/list'"/>
                </td>
            </tr>
        </table>
    </form:form>
</div>
```

七. 加入 Spring AOP 機制

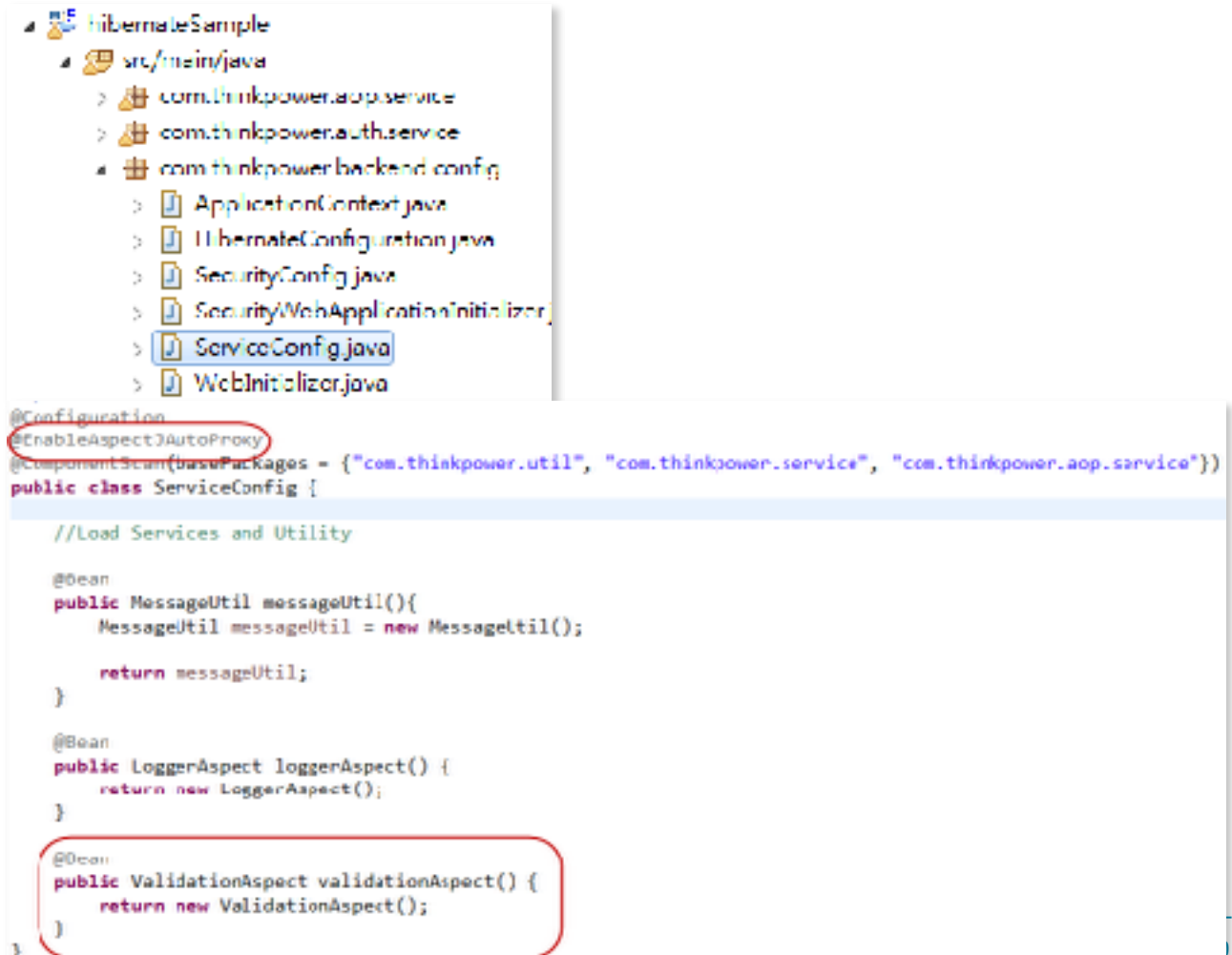
因為每個需要檢核的地方都要插入判斷是否有錯誤的程式片段，會有很多重複的程式碼，於是加入 Spring AOP 機制，根據定義的 Point Cut 抓出需要用到檢核的 JoinPoint 加入 Advice (檢核判斷是否有誤邏輯)。

pom.xml 加入 AspectJ :

```
<!-- AspectJ -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${aspectj.version}</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectj.version}</version>
</dependency>
<!-- AspectJ -->
<aspectj.version>1.7.4</aspectj.version>
```

Service Config 加入 EnableAspectJAutoProxy 及 Aspect Bean :



```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages = {"com.thinkpower.util", "com.thinkpower.service", "com.thinkpower.aop.service"})
public class ServiceConfig {

    //Load Services and Utility

    @Bean
    public MessageUtil messageUtil(){
        MessageUtil messageUtil = new MessageUtil();

        return messageUtil;
    }

    @Bean
    public LoggerAspect loggerAspect() {
        return new LoggerAspect();
    }

    @Bean
    public ValidationAspect validationAspect() {
        return new ValidationAspect();
    }
}
```

定義 PointCut :

```
package com.thinkpower.bsp.service;

import org.aspectj.lang.annotation.Aspect;

@Aspect
public class PointcutDefinition {

    @Pointcut("within(com.thinkpower.controller..*)")
    public void webLayer() {
    }

    @Pointcut("within(com.thinkpower.service..*)")
    public void serviceLayer() {
    }

    @Pointcut("within(com.thinkpower.dao..*)")
    public void dataAccessLayer() {
    }

    @Pointcut("execution(* com.thinkpower.controller..*.*(@javax.validation.Valid (*)...))")
    public void validationLayer() {
    }
}
```

加入 ValidationAspect.java :

```
package com.thinkpower.bsp.service;

import java.util.List;

@Aspect
public class ValidationAspect {

    private Logger logger = LoggerFactory.getLogger(getClass());

    @Before(value = "com.thinkpower.bsp.service.PointcutDefinition.validationLayer()")
    public Object validateAround(ProceedingJoinPoint joinPoint) throws Throwable {
        logger.debug("**** validateAround Start ****");

        BindingResult bindingResult = null;
        // 取出JoinPoint的參數BindingResult
        for (Object arg : joinPoint.getArgs()) {
            if (arg instanceof BindingResult) {
                bindingResult = (BindingResult) arg;
            }
        }
        if (bindingResult != null) {
            List<ObjectError> errors = bindingResult.getAllErrors();
            if (errors.size() > 0) {
                // 返回驗證結果
                logger.debug(joinPoint.getSignature().getName() + " not pass!");

                // Get referer url
                HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.currentRequestAttributes()).getRequest();
                String referer = request.getHeader("referer");

                // 取得Referer的Domain
                String[] refererDomain = referer.split("/");
                String lastDomain = refererDomain[refererDomain.length - 1];
                String view = (lastDomain.endsWith(".htm") || lastDomain.endsWith(".html")) ? "add" : "edit";
                String refererView = refererDomain[refererDomain.length - 2] + "." + view;
                return refererView;
            }
        }
        logger.debug("**** validateAround End ****");
        return joinPoint.proceed();
    }
}
```

- @Before : 前置通知，在方法執行之前執行。

-
- @After：後置通知，在方法執行之後執行。
 - @AfterReturning：返回通知，在方法返回結果之後執行。
 - @AfterThrowing：異常通知，在方法拋出異常之後。
 - @Around：環繞通知，圍繞着方法執行。

參考來源

- Hibernate Validator - <http://hibernate.org/validator/documentation/getting-started/>
- 後端驗核範例 - <http://codetutr.com/2013/05/28/spring-mvc-form-validation/>
- 自訂validator - <https://docs.jboss.org/hibernate/validator/4.1/reference/en-US/html/validator-customconstraints.html>
- Spring AOP - <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html>
- AspectJ - <https://eclipse.org/aspectj/>
- Spring AOP 範例 - <http://www.byteslounge.com/tutorials/spring-aop-pointcut-advice-example>
- 使用Spring AOP管理所有Valid的bindingResult - <http://www.cnblogs.com/zhoujingjie/articles/4066597.html>
- 使用Spring AOP機制插入日誌 - <http://samchu.logdown.com/posts/243374-using-the-spring-aop-mechanism-to-insert-records>