ＳＴＥＰ０８

Ｓｐｒｉｎｇフレームワーク入門

コードで理解するＳｐｒｉｎｇの世界

AOPによる走行ログ出力

目次

[1 AOPの準備 1](#_Toc428289231)

[1.1 Spring AOPライブラリ登録 （Maven） 1](#_Toc428289232)

[1.2 Spring AOPを有効にする設定 （Spring定義ファイル） 2](#_Toc428289233)

[2 AOPによるログ出力処理を実装 3](#_Toc428289234)

[2.1 AOPにより実行する処理を実装 3](#_Toc428289236)

[2.2 テスト実行 4](#_Toc428289237)

# AOPの準備

## Spring AOPライブラリ登録 （Maven）

１）Maven設定ファイルにSpring AOPライブラリを追記

|  |
| --- |
| /pom.xml |
| ● 35行目付近 |
| <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${org.springframework-version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${org.springframework-version}</version>  </dependency>  46  <!-- aop -->  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.8.9</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${org.springframework-version}</version>  </dependency>  <!-- PostgreSQL + JDBC -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${org.springframework-version}</version>  </dependency> |

## Maven設定ファイルを更新するとプロジェクトがリビルドする。

## ビルドの進捗は右下に地味に表示されているだけなので見落とさないこと

## ビルドに数秒～数分がかかる場合があるが、終わるまで待っているほうが吉。

## Spring AOPを有効にする設定 （Spring定義ファイル）

１）Spring MVC の Bean 定義ファイル（application-context-biz.xml）に設定追加

Spring AOP を有効にする設定を追記する。

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-biz.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"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="  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  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd">    <!-- Defines shared resources visible to all other web components -->  <context:component-scan base-package="jp.sample.bookmgr.biz" />    <!-- Enables the Spring AOP -->  <aop:aspectj-autoproxy />    <!-- データソース用のリソース -->  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  <property name="location" value="classpath:/jdbc.properties" />  </bean>  ・・・  ・・・ 以下変更なし |

＃ Spring AOPを有効にする設定内容の詳細は、「Spring3入門」の90ページ（第3章）を参照のこと。

＃ Spring AOPに関する基本的な説明は、「Spring3入門」の第3章を参照のこと。

# AOPによるログ出力処理を実装



## AOPにより実行する処理を実装

１）AOPを用いてログ出力を行うクラス（BookAspect.java）の新規作成

|  |
| --- |
| /src/main/java で右クリック → New → Class    ● Java Class 画面  ---------------  Package: 「jp.sample.bookmgr.biz.aspect」  Name: 「BookAspect」 と入力して Finish  ---------------  クラスのひな形が自動作成される |

２）AOPを用いてログ出力を行うクラス（BookAspect.java）コーディング

|  |
| --- |
| /src/main/java/jp.sample.bookmgr.biz.aspect.BookAspect.java （新規作成） |
| package jp.sample.bookmgr.biz.aspect;  import org.aspectj.lang.JoinPoint;  import org.aspectj.lang.annotation.After;  import org.aspectj.lang.annotation.AfterThrowing;  import org.aspectj.lang.annotation.Aspect;  import org.aspectj.lang.annotation.Before;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.stereotype.Component;  /\*\*  \* 書籍管理アプリケーションのアスペクト処理実装クラス  \*  \* @author 長住@NTT-AT  \* @version 1.0  \*/  @Aspect  @Component  public class BookAspect {  /\*\*  \* ロガー  \*/  private static final Logger logger = LoggerFactory.getLogger(BookAspect.class);  /\*\*  \* jp.sample.bookmgr.biz パッケージの全てのクラス・メソッドの開始前に呼ばれる処理  \* APO対象となるクラスとメソッドの名前を取得し、メソッド開始ログを出力する  \*  \* @param jp JoinPointオブジェクト  \*/  @Before("execution(\* jp.sample.bookmgr.biz..\*(..))")  public void beforeMethod(JoinPoint jp) {  // AOP対象のクラス名を取得  String className = jp.getTarget().getClass().getName();  // AOP対象のメソッド名を取得  String methodName = jp.getSignature().getName();  // ログ出力  logger.info("Before: " + className + "#" + methodName + " START!!");  }  /\*\*  \* jp.sample.bookmgr.biz パッケージの全てのクラス・メソッドが終了する直前に呼ばれる処理  \* APO対象となるメソッドの名前を取得し、メソッド終了ログを出力する  \*  \* @param jp JoinPointオブジェクト  \*/  @After("execution(\* jp.sample.bookmgr.biz..\*(..))")  public void afterMethod(JoinPoint jp) {  // AOP対象のクラス名を取得  String className = jp.getTarget().getClass().getName();  // AOP対象のメソッド名を取得  String methodName = jp.getSignature().getName();  // ログ出力  logger.info("After: " + className + "#" + methodName + " END!!");  }  /\*\*  \* jp.sample.bookmgr.biz パッケージの全てのクラス・メソッドが例外をスローした時に呼ばれる処理  \* APO対象となるクラスとメソッドの名前を取得し、例外メッセージを出力する  \*  \* @param jp JoinPointオブジェクト  \*/  @AfterThrowing(value="execution(\* jp.sample.bookmgr.biz..\*(..))", throwing="ex")  public void afterThrowingMethod(JoinPoint jp, Exception ex) {  // AOP対象のクラス名を取得  String className = jp.getTarget().getClass().getName();  // AOP対象のメソッド名を取得  String methodName = jp.getSignature().getName();  // Exceptionメッセージを取得  String mes = ex.toString();  // ログ出力  logger.info("AfterThrowing: " + className + "#" + methodName + " Throw(\"" + mes + "\")");  }  } |

## テスト実行

書籍管理プログラムを起動し、コンソール画面に出力されるログメッセージを確認する。

