AOP(Aspect Oriented Programming)について

平成19年度　OISA 「技術研究会」
Java第1部会　報告書

2008年　2月

部会員およびアドバイザー・技術委員

【部会委員】　（順不同）

野中　健太郎	株式会社富士通大分ソフトウェアラボラトリ
阿部　高晴	株式会社シーエイシー
安部　智仁	株式会社ワイズ・システムズ
今宮　和則	株式会社オーイーシー
兒玉　清幸	大分大学工学部
柿添　亮平	新日鉄ソリューションズ株式会社
谷村　聡	大分交通株式会社
岩佐　俊一	KCS大分情報専門学校

【アドバイザー・技術委員】　（順不同）

三宮　由裕	三井造船システム技研株式会社
築城　久敏	株式会社システムトレンド
清水　太	株式会社ワイズ・システムズ

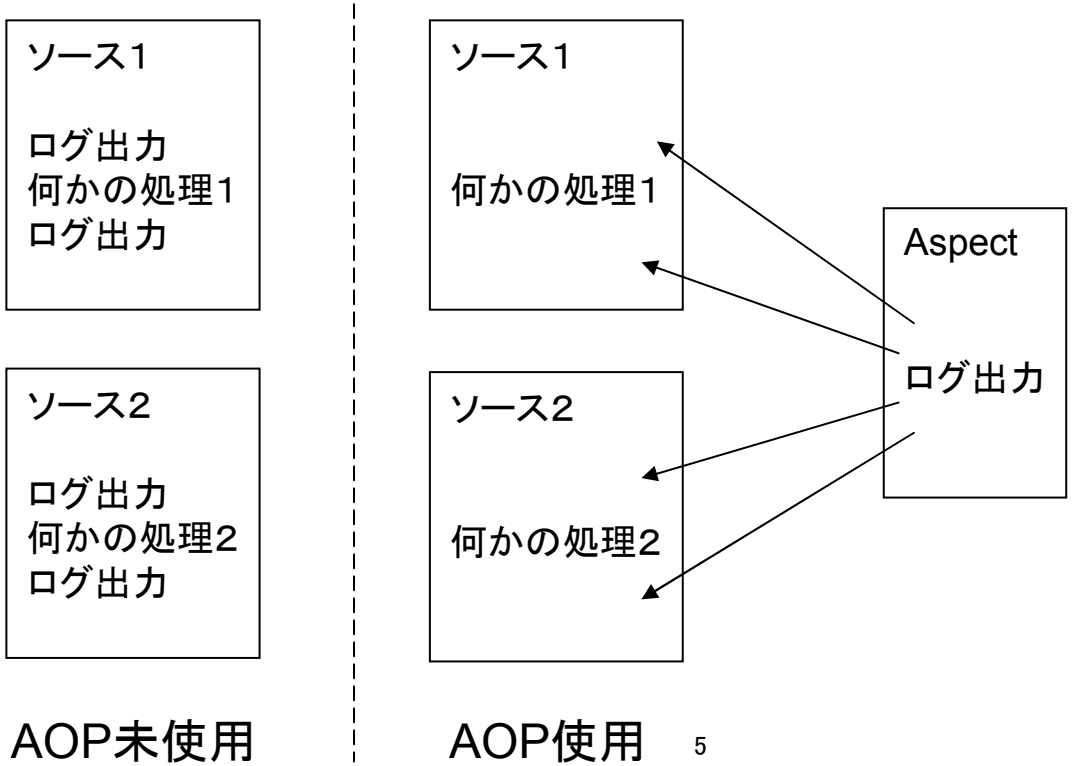
目次

1．はじめに

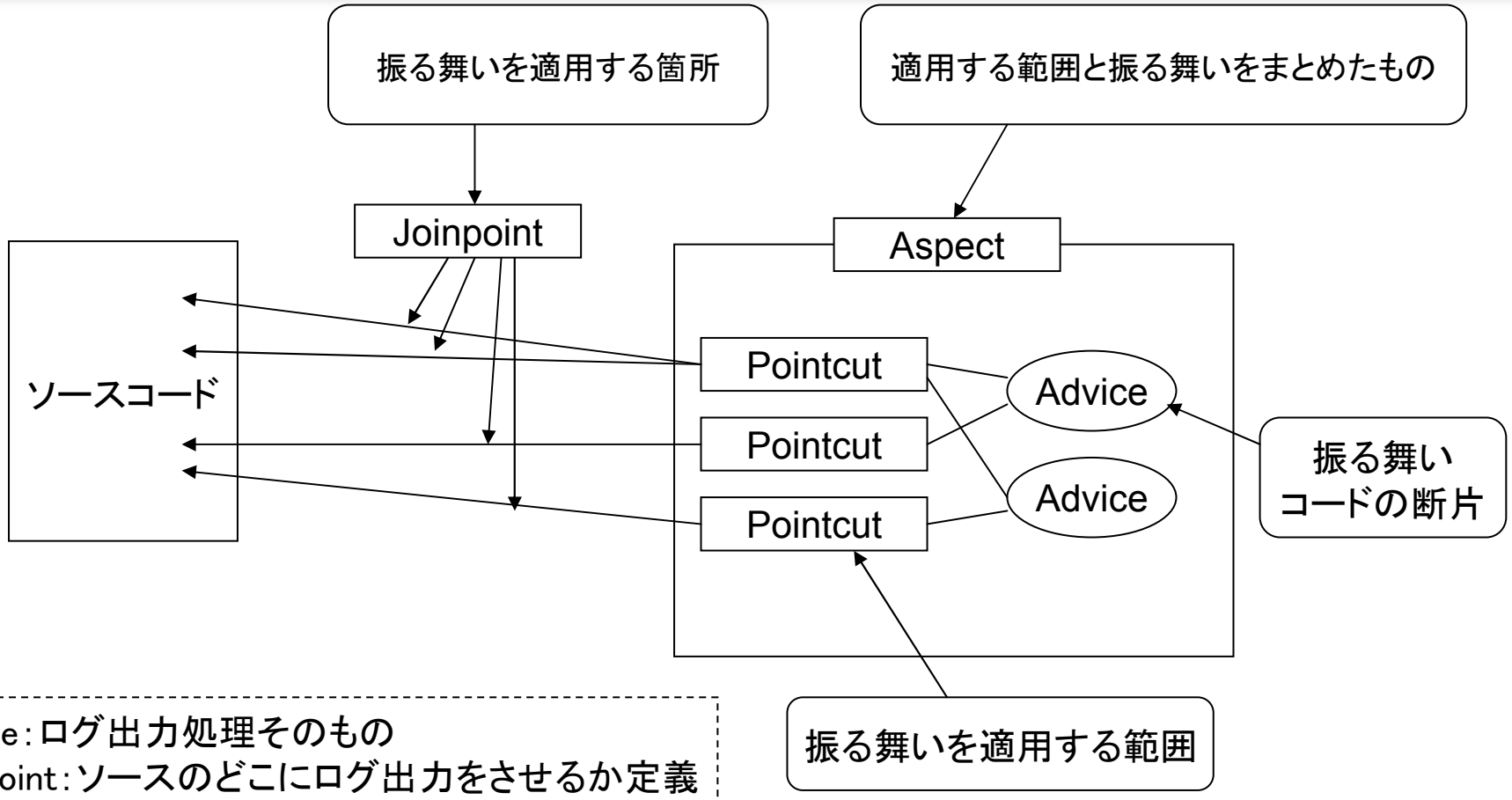
1．1アスペクト指向プログラミング（AOP）の概要

・システム全体に於いて、様々な箇所で何度も行われる処理を
「横断的な関心事」と呼び、システム本来の関心事から分離し、まとめることが目的。
・作りあげたオブジェクト郡に対して「機能を挿入する」ことができ、基本的にどのような処理
にも挿入することができる。

・例　：複数のソースにまたがるログ出力処理を1箇所にまとめる事が出来る。
同じような機能（コード）の重複を避ける事ができる。



AOPの要素の関係



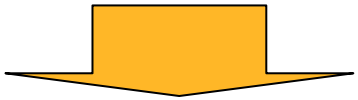
Advice：ログ出力処理そのもの
Joinpoint：ソースのどこにログ出力をさせるか定義
Pointcut：Joinpointをまとめたもの
Acpect：AdviceとPointcutをまとめたもの

Weaving：分離したものを織り込む（くっつける）

AOPを備えたフレームワーク

AspectJ

Java言語を拡張した独自言語によって静的にWeavingを行う。
アスペクトを適用、追加、削除する度にコンパイル処理が必要となるので、かなり不便。



動的なWeavingを行うフレームワーク（Dynamic AOPフレームワーク）が登場
・ランタイムにアスペクトを追加、削除することができる。
・アスペクトに関する情報は、XMLファイルなどで定義する。
・アスペクトはjavaプログラムとして定義できる。

＜2種類のDynamic AOPフレームワーク＞

1．クラスのロード時や実行時にバイトコードの修正によりアスペクトをWeavingする。
　・・・・JBossAOP、AspectWerkz、JAC、 Seasar2等。

2．proxyを利用したインターセプタによって、プログラム実行時にメソッドの呼び出しを傍受し
　アスペクをWeavingする。
　・・・・Spring、Nanning、dynaop等。

上記フレームワークの中でAspectJ、Spring、Seasar2の評価を行う。

2. 各フレームワーク評価

AspectJ

・アスペクトの静的なWeavingを行うフレームワーク

・パロアルト研究所で開発が行われ、Eclipseプロジェクトへ移管。

・現在のバージョンはAspectJ5(1.5)

・「aspect」と呼ばれるモジュール単位が追加されている。

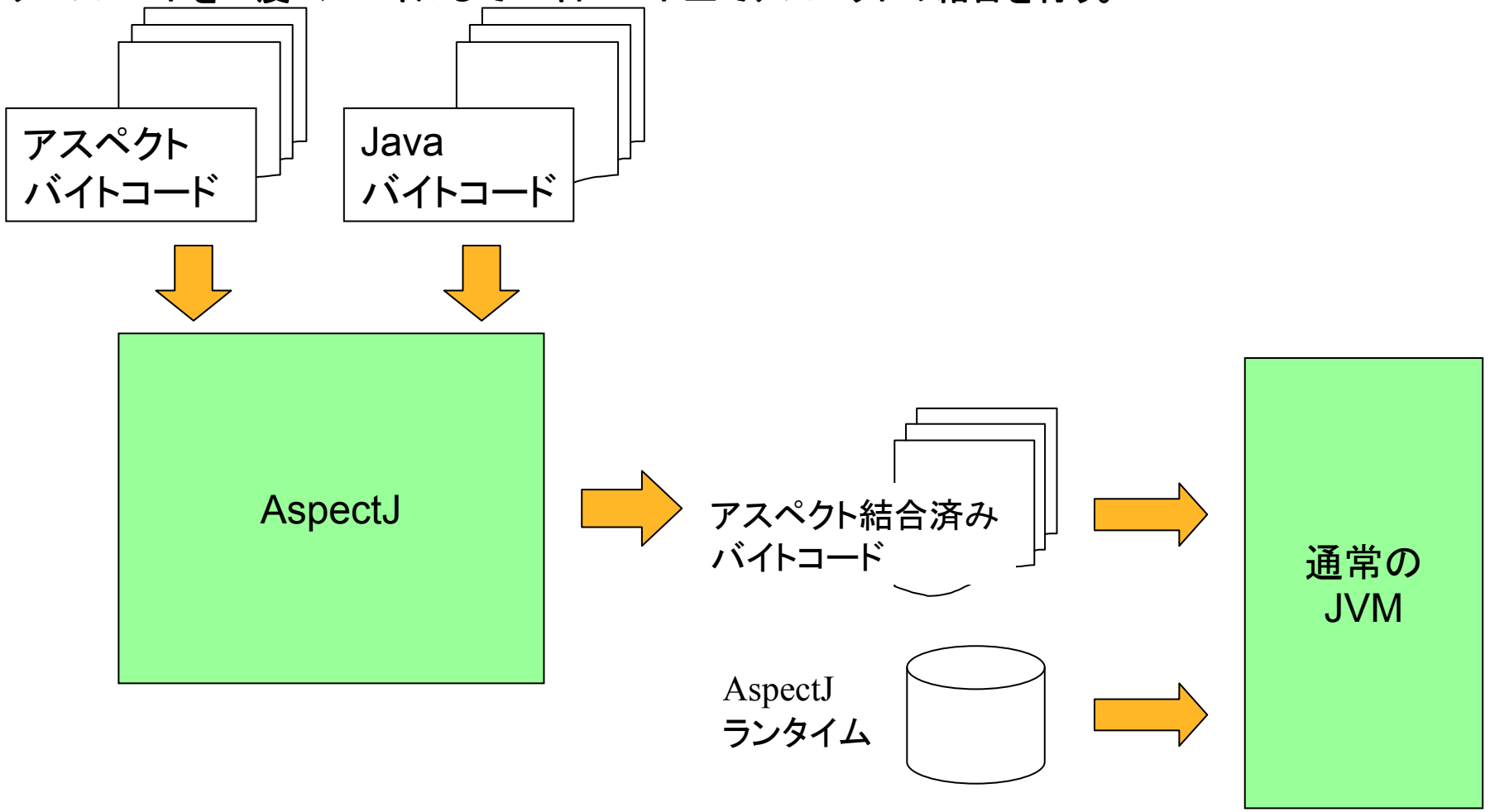
・classと似た定義形式

```
aspect AspectName(){
    アスペクトメンバーの定義
}
```

・アスペクトを適用、追加/削除にはコンパイルが必要。(HotDeployは行えない)

AspectJ 言語処理

・ソースコードを一度コンパイルしてバイトコード上でアスペクトの結合を行う。



・アスペクトの結合をクラスロード時まで遅らせることも可能
最終的なバイトコードはJava標準のもの

AspectJ サンプルプログラム

func1を呼び出すクラス

```
public Class Sample1{
    public static void main(String[] args){
        new Sample1.func1();
    }

    private void func1(){
        SyStem.out.plintln("実行");
    }
}
```

○このクラスの処理のみ
実行結果

実行

・Point Cutの定義

・Join Pointの設定

・adviceの定義

```
aspect AspectSample1 {

    pointcut atFunc1() : call(void Sample1.func1());

    before() : atFunc1() {
        System.out.println("実行開始");
    }
    after() : atFunc1() {
        System.out.println("実行終了");
    }
}
```

○実行結果

実行開始
実行
実行終了

Spring AOP

・ Springはロッドジョンソン氏の著書“Expert OneonOneJ2EE Design and Development”の中で使用されたコードをベースとした Java/J2EE アプリケーションフレームワーク

・現在のバージョンは、JDK1.4以上　2.5.1
JDK1.3　2.0.8

・AOPによってメソッド処理をインターセプト
（メソッドの呼び出しタイミングで振る舞いを挿入する)の定義ができる。
→「処理に何らかの付加を行う」目的でAOPが使われる。

・ソースのメタデータを使用して処理内容を組み込むアスペクト指向プログラミングができる。

・Spring AOPはPure Javaで実装されていて、特別なコンパイルを必要としない。

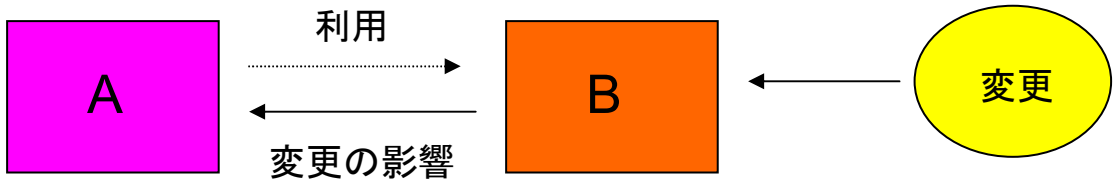
・ポイントカットや異なるadvice型を表すためのクラスが提供されている。

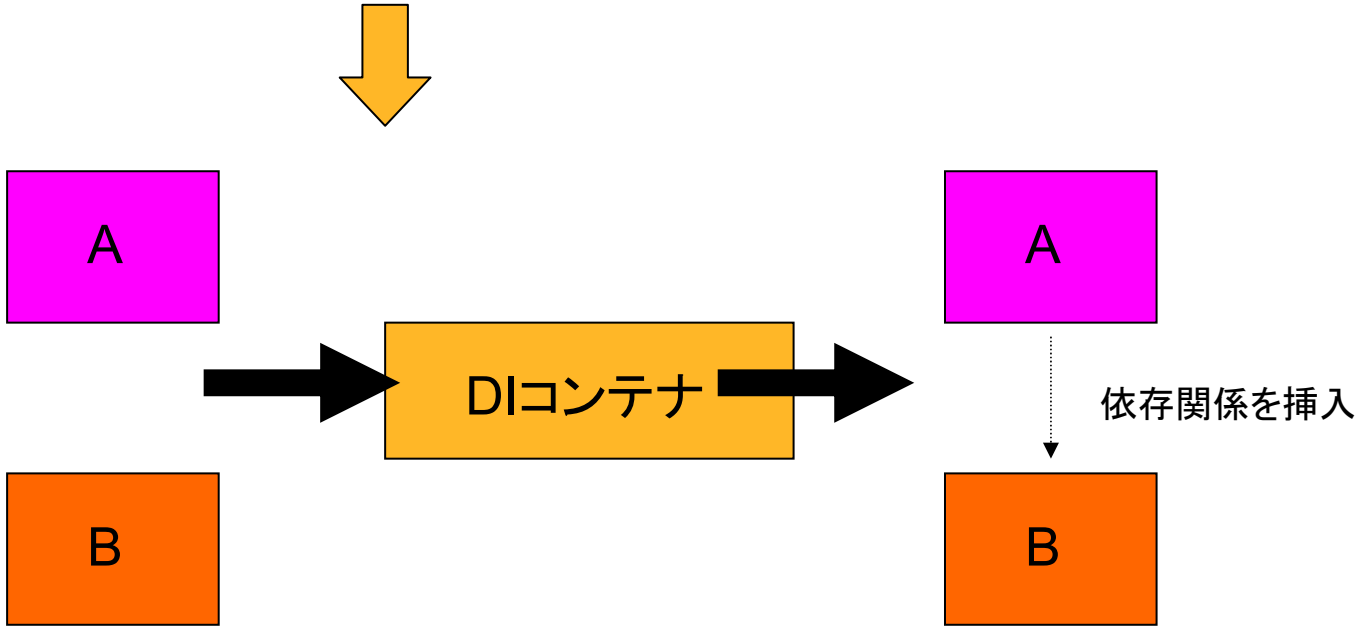
・SpringのAOPは普通SpringIocコンテナと一緒に使用され、adviceやポイントカットはそれ自身SpringのIoc(DI)に管理されている。

・Adviceは通常のBean定義の文法を用いて指定する。

DIコンテナの役割

・コンポーネント間の依存関係を小さくする
「コンポーネントはある機能を提供するクラス（群）」





Spring AOP サンプルプログラム

Printメソッド定義クラス

```
package sample;

public class Sample1 {

    public void print() {
        System.out.println("実行");
    }
}
```

○このクラスを呼び出す処理のみ

実行結果

→ 実行

```
package sample;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class MyIntercepter implements MethodInterceptor{

  // メソッド呼び出し前後に文字列を出力する
  public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("実行開始");
        Object result = invocation.proceed();
        System.out.println("実行終了");
        return result;
    }
}
```

SpringAOPの
インターフェース

Around Advice
の定義

Spring AOP サンプルプログラム2

Bean定義ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

  <!-- 実行処理を定義 -->
  <bean id="sample" class="sample.Sample1" />

  <!-- AOPに関する情報を定義    -->
  <aop:config>
    <aop:advisor pointcut="execution(* sample.*.print())"
      advice-ref="intercepter" />
  </aop:config>

  <!-- 追加処理定義 -->
  <bean id="intercepter" class="sample.MyIntercepter"/>
</beans>
```

基本処理クラスの指定

・Point Cutの定義

advice-ref属性の定義

adviceの参照(ref)を指定

adviceの定義

追加処理を行うクラスの指定

Spring AOP サンプルプログラム3

実行クラス

```
package sample;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
  public static void main(String[] args) {
    ApplicationContext context =
      new ClassPathXmlApplicationContext(
        "sample/applicationContext.xml");

    // 実行
    Sample1 bean = (Sample1)context.getBean("sample");
    bean.print();
  }
}
```

実行結果

```
実行開始
実行
実行終了
```

Bean定義ファイルの指定

Seasar2 AOP

・Seasar2は、「**システム構築の現場にもっと“易しさ”と“優しさ”を**」を目的にSeasarファウンデーションによって開発が進められているAOP機能を備えたDIコンテナ。
・ひがやすお氏がチーフコミッタ

・現在のバージョンは2.4.17

・AOPによってメソッド処理をインターセプト(メソッドの呼び出しタイミングで振る舞いを挿入する)の定義ができる。
→「処理に何らかの付加を行う」目的でAOPが使われる。

・ポイントカットや異なるadvice型を表すためのクラスが提供されている。

・diconファイルで結びつきを定義する(<component>タグ)

・diconファイルで「クラスをまたいで存在する関心事(Interceptor)」の挿入が出来る
(挿入対象のプロパティを指定する<property>タグ、Interceptorを織り込む<aspect>タグ)

Seasar2 AOP サンプルプログラム

Printメソッド定義クラス

```
//文字列出力クラス
package sample;

public class Sample1 {

    public void print() {
        System.out.println("実行");
    }
}
```

○このクラスを呼び出す処理のみ
実行結果

→ 実行

```
package aop;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class MyIntercepter implements MethodInterceptor{
    public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("実行開始");
        Object result = invocation.proceed();
        System.out.println("実行終了");
        return result;
    }
}
```

Seasar2AOPのインターフェース

Around Adviceの定義

Seasar2 AOP サンプルプログラム2

diconファイル

```
//XMLの設定ファイル(diconファイル)
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.3//EN"
"http://www.seasar.org/dtd/components23.dtd">

<components>
    <component name="myInterceptor"
        class="aop.MyIntercepter"/>

    <component name="sample" class="sample.Sample1">
        <aspect pointcut="print">myInterceptor</aspect>
    </component>

</components>
```

adviceの定義

・Point Cutの定義

Seasar2 AOP サンプルプログラム3

実行クラス

```
//実行クラス
package aop;

import org.seasar.framework.container.S2Container;
import org.seasar.framework.container.factory.S2ContainerFactory;

import sample.Sample1;

public class Main {

    public static void main(String[] args) {
        S2Container container = S2ContainerFactory.create("aop/test.dicon");
        Sample1 sample = (Sample1)container.getComponent("sample");
        sample.print();
    }

}
```

diconファイルの指定

実行結果

```
実行開始
実行
実行終了
```

3．AOPの利点・欠点

3．1 AOPの利点

・横断的関心事を1箇所にまとめることで複雑さを軽減
　・・・システム全体で共通の処理を分離するため，プログラムがシンプルになり
　　管理しやすくなる。

・既存のシステムを変更せず、新しい機能を追加
　・・・例えば、トランザクションのような処理を，「作成済みのクラスには一切手を入れず
　　に」後から追加できる。

・再利用性の向上
　・・・コンポーネントをまとめて管理できるため再利用性、変更への耐性が高くなる。

3．1 AOPの欠点

・可読性の低下

・・・オブジェクト（ソース）を見ただけでは動作は分からない（メインプログラムから adviceの処理は見えない）ため可読性が低下する。

・アスペクトを多用することで、アスペクトの衝突が発生

・・・機能を簡単に追加することができる為、何でもアスペクトで挿入してしまうと アスペクト同士の衝突が発生してしまう。

3．2 各フレームワークの比較（1／2）

●静的AOP（Aspect J）と動的AOP（Spring,Seasar）の比較

静的AOP	動的AOP
・アスペクトは特別な言語を習得する必要がある。	・アスペクトはJavaプログラムとして定義できる
・アスペクトを適用するためにはコンパイル処理が必要	・アスペクトに関する情報は設定ファイルで定義
・プログラムの追加、削除にアプリケーションの停止を行う必要がある	・プログラムの追加、削除にアプリケーションの停止が不要（Hot Deployに対応）
・広範囲なjoin point	・静的AOPほどjoin pointは多くない

3．2 各フレームワークの比較（2／2）

●Spring、Seasar2の比較

Spring	Seasar2
・開発実績多数	・Springに比べると開発実績は多くない
・ドキュメントは英語＋日本語で多数	・日本発のFWで日本語ドキュメント多数 MLも日に10～20通で活発 2～3日で障害修正が行われる事も…
・HotDeployはclassファイルのみ	・Seasar2.4から完全HotDeploy(設定ファイルも対象)

4. まとめ

4．1 結論

- AOPはオブジェクト指向プログラミングの不得意とする点を補完する。
- 横断的関心事は色々あるが、トレースログやトランザクション管理での利用が最適。
- 例えばAOPフレームワーク用のデバッグツールなど、各種ツールの充実化によって、さらにAOPが使いやすいものになる。

4．2 各メンバーの所感

- この研究を通じて、SpringやSeasarなどのフレームワークについて学べたこと。

- 業務以外で他社の人や大学生と一緒にひとつのことに取り組めたこと。