

サーバサイド AOP

千葉 滋
東京工業大学

丸山先生レクチャーシリーズ 2005年1月28日

このごろ世間ではやるもの

- Component Framework なら
- Dependency Injection + AOP



だから素晴らしい... というが本当?

Jan 28, 2005

Dependency Injection

- コンポーネントの独立性を高める
- フィールドの初期化コードを書かずにすむ

取得方法が変わってもそのまま再利用可能

```
class BookShop {
    StockDAO stock;
    void buy(String title) {
        在庫から1冊減らす
        stock.sell(title);
    }
}
```

フレームワークが提供。取得方法はフレームワークごとに異なる。

```
class StockDAO {
    データベース操作
    用のメソッド
}
```

フレームワークが stock の値を inject する

AOP

- Aspect-Oriented Programming
- おまけ?
- いや、どうも便利らしい...

業務ロジックから系統的機能を「Crosscutting Concern」に排出した「Core Concern」は、シンプルなソースになります。本来のやりたかったことだけが記述されます。(Seasar のドキュメントより)

Jan 28, 2005

Logging aspect の例

- 実行時間を計る (非AOP)

チューニングが済んだら、簡単に除去できるか?

```
class BookShop {
    StockDAO stock;
    void buy(String title) {
        long t = System.currentTimeMillis();
        在庫から1冊減らす
        stock.ship(title);
        t = System.currentTimeMillis() - t;
        System.out.println(t);
    }
}
```

Jan 28, 2005

On/Off を簡単に

- (古い)JBoss AOP を利用

もとのコードはそのまま

```
class BookShop {
    StockDAO stock;
    void buy(String title) {
        在庫から1冊減らす
        stock.ship(title);
    }
}
```

アスペクトの定義 (買って来たコード)

```
class Logger implements Interceptor {
    void invoke(Invocation i) {
        long t = System.current...
        i.invokeNext();
        t = System.current...
        System.out.println(t);
    }
}
```

intercept

Jan 28, 2005

JBoss AOP の場合 (続き)

- 結合は XML ファイルで指定

```
<bind pointcut= "public void BookShop->buy(String title)">
  <interceptor class="LoggingAspect"/>
</bind >
```

実行時間を計るコードを
別コンポーネントに分離できた

Jan 28, 2005

7

Non-functional concern

- 各業務ロジックに共通する汎用の機能
 - OS やミドルウェアが提供すべきシステムの機能
 - Logging、分散処理、トランザクション、セキュリティ...
- これをコンポーネント化するのが AOP
 - 一般ユーザは、XML で設定するだけで利用できる。便利。

Jan 28, 2005

8

あれ、横断的関心事は？

- Crosscutting Concern
 - これをコンポーネント化するのが AOP

あ、それですか？
Non-functional Concern と Crosscutting Concern
は同じものですよ。

(ちなみに Seasar のドキュメントの話ではありません)




Jan 28, 2005

9

Non-functional concern を扱うだけなら AOP は不要

Metaobject Protocol (MOP)

- Non-functional concern を
コンポーネント化する技術 
- 15年以上前から存在。AOP の源流。
- 実は interceptor model (古い JBoss AOP) は MOP とほとんど同じもの。
違いは名前だけ。
 - Interceptor を metaobject と読み替える。
 - 車輪の再発見？

Jan 28, 2005

10

では、AOP とはいったい
何なのでしょう？



Jan 28, 2005

11

Logging 再び

- 非 AOP プログラムを少し改良してみる

Logging 処理を別コンポーネントに
(売ってイソウ。Log4J?)

```
class BookShop {
  StockDAO stock;
  Logger logger;
  void buy(String title) {
    logger.start();
    在庫から1冊減らす
    stock.ship(title);
    logger.finish();
  }
}
```

```
class Logger {
  long t;
  void start() {
    t = System.current...
  }
  void finish() {
    t = System.current...
    System.out.println(t);
  }
}
```

12

これなら on/off が容易？

- No! これでもだめ。
 - Logger のメソッド呼び出しが含まれるから。

Logging 処理の一部がまぎれこんでいる。

```

class BookShop {
    StockDAO stock;
    Logger logger;
    void buy(String title) {
        logger.start();
        在庫から1冊減らす
        stock.ship(title);
        logger.finish();
    }
}

```

Jan 28, 2005 13

横断的関心事

- Crosscutting concern
 - 実装の一部が他のコンポーネントの中へ散らばってしまう処理

このメソッド呼び出しは Logging 処理の一部。BookShop 処理とは無関係。

```

class BookShop {
    StockDAO stock;
    Logger logger;
    void buy(String title) {
        logger.start();
        在庫から1冊減らす
        stock.ship(title);
        logger.finish();
    }
}

```

Jan 28, 2005

理念としてよくない

- ... のは理解できます。しかし、実用上、本当によくないのですか？

メソッド呼び出し2つぐらい気にならない？

```

class BookShop {
    StockDAO stock;
    Logger logger;
    void buy(String title) {
        logger.start();
        在庫から1冊減らす
        stock.ship(title);
        logger.finish();
    }
}

```

Jan 28, 2005

Dependency がある

- BookShop 単独で再利用できない
 - 常に Logger と一緒に再利用するか、メソッド呼び出しを手で削除するしかない

```

class BookShop {
    StockDAO stock;
    Logger logger;
    void buy(String title) {
        logger.start();
        在庫から1冊減らす
        stock.ship(title);
        logger.finish();
    }
}

```

依存

```

class Logger {
    long t;
    void start() {
        t = System.current...
    }
    void finish() {
        t = System.current...
        System.out.println(t);
    }
}

```

6

コンポーネントの上下関係

- 上位コンポーネント **BookShop**
 - 下位コンポーネント抜きで再利用できない
 - 下位コンポーネントに依存
- 下位コンポーネント **Logger**
 - 再利用が容易

Jan 28, 2005 17

依存を減らし再利用を促進

- AOP 版
 - Logger 抜きの BookShop 単独で再利用可能

```

class BookShop {
    StockDAO stock;
    void buy(String title) {
        在庫から1冊減らす
        stock.ship(title);
    }
}

```

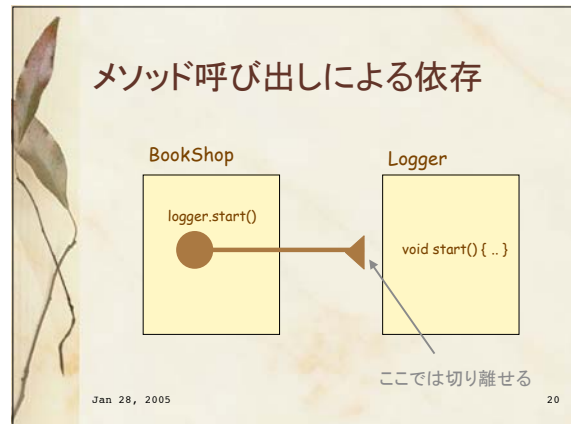
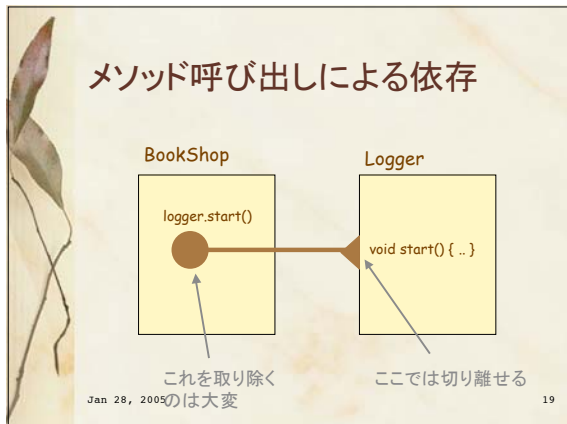
メソッド呼び出しを含まない

```

class Logger
    implements Interceptor {
    void invoke(Invocation i) {
        long t = System.current...
        i.invokeNext();
        t = System.current...
        System.out.println(t);
    }
}

```

Jan 28, 2005



- ### AOP の目的
- メソッド呼び出しにかかわるコンポーネント間の dependency を低減
 - 再利用を促進
 - 応用例
 - Non-functional concern
 - 一時的な機能追加
 - ショッピングサイトに、送料無料キャンペーンの処理をアスペクトとして追加
- Jan 28, 2005 22

新しい JBoss AOP

- アスペクトはより普通のオブジェクトに

```
class Logger {
    Object printTime(Invocation i) {
        long t = System.current...
        Object r = i.invokeNext();
        t = System.current...
        System.out.println(t);
        return r;
    }
}
```

好きな名前だよ

Logger 内に複数のメソッドを定義してもよい。

条件によって異なるメソッドが呼ばれるように XML で設定できる。

戻り値を返す

Jan 28, 2005 23

新しい JBoss AOP

- Dependency Injection に対応

```
class BookShop {
    @Inject StockDAO stock;
    void buy(String title) {
        在庫から1冊減らす
        stock.ship(title);
    }
}
```

```
class InjectDAO {
    StockDAO dao = ...
    Object access(FieldReadInvocation i) {
        return dao;
    }
    Object access(FieldWriteInvocation i) {
        throw new RuntimeException();
    }
}
```

実は古い Jboss AOP でも可能だが...

Jan 28, 2005

まとめ

- AOP と Dependency Injection は同根
- 新世代の container framework
= Dependency Injection × AOP

↑
たし算ではなく、かけ算