Load-time weavingで広がるAOPの可能性

自己紹介

- 名前：木村　聡（きむら　さとし）
- Seasarプロジェクトコミッタ：
  - S2Struts
  - S2Mai
  - 舞姫
- 仕事
  - （株）フルネス
  - フレームワーク
  - 自動生成ツール

これまで書いたものとか

- 書籍:
  - Eclipseで学ぶはじめてのJava
  - Seasar入門　～はじめてのDI&AOP～
- 雑誌、Web記事
  - CodeZine
  - DB Magazine
  - WEB+DB
    - 「Seasar2徹底攻略」(Vol.31)
  - JavaWorld
    - 「開発者にとって“易しく、優しい”軽量コンテナSeasar2の実力を探る」(2005/05)

はじめに

- AOPの乱用禁止

AOPとは

- Aspect Oriented Programの略
- 日本語では「アスペクト指向プログラム」
- ソフトウェアの複雑さの低減や再利用性を向上させる
- よく使われる機能
  - ログ
  - 例外処理
  - トランザクション
  - 認証処理
  - RPC
  - 障害対応
- S2プロダクトだと
  - S2Dao

簡単に言うと

- AOPの仕組みを使うと、
  後から機能や処理を挿入することができるようになる

例

```
public String hello(String arg) {
      Strimg message = "Hello " + arg;
      System.out.println(message);
      return message;
}
```


```
Hello World
```

例

```
public String hello(String arg) {
      Strimg message = "Hello " + arg;
      System.out.println(message);
      return message;
}
```

AOP使って実行

＋設定ファイル
など

```
2009-03-14 17:00.00 BEGIN hello("World")
Hello World
2009-03-14 17:00.01 END   hello("World") : "Hello World"
```

AOPのメリット

- 生産性の向上
  - 共通処理を記述しなくても良い
- 品質の向上
  - 共通処理の埋め込み忘れが減る
- アノテーションの処理を組み込みやすい

よく見かける文章

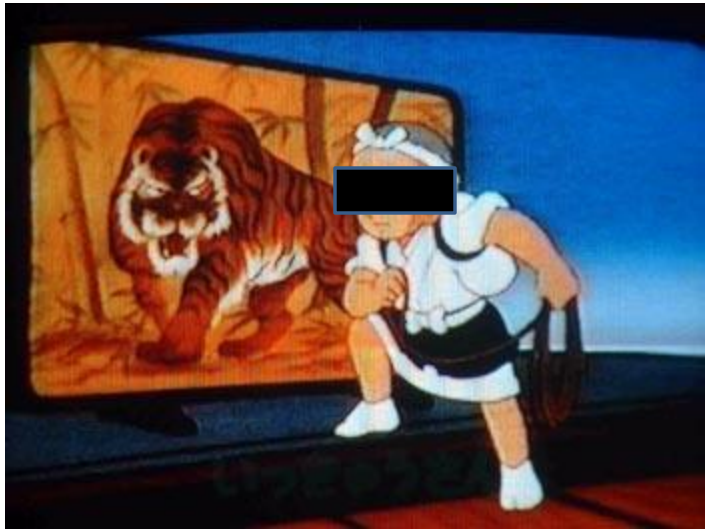
- AOPを使って、
  アプリケーションのソースコードを一切変更することなく、〇〇します。

Struts Only

このStrutsで作られたシステムに
AOPでTraceを出してくれ

Struts Only

将軍様
Seasarで作り直してください
そうしたら出来ます



AOPは適用が難しい

- クラスの書き換えが必要
  - コンパイル時に書き換える
- ツール／文法が難しい
  - Java以外の言語、ツールを覚える必要がある
    - AspectJとか

これまでの条件

- SeasarなどAOPの仕組みが提供されているフレームワークを使っていれば出来る
- AspectJなどを使う場合、コンパイルし直せば出来る

- AOPとDIコンテナは相性が良い
  - DIコンテナのメリット

深いところには手が届かない

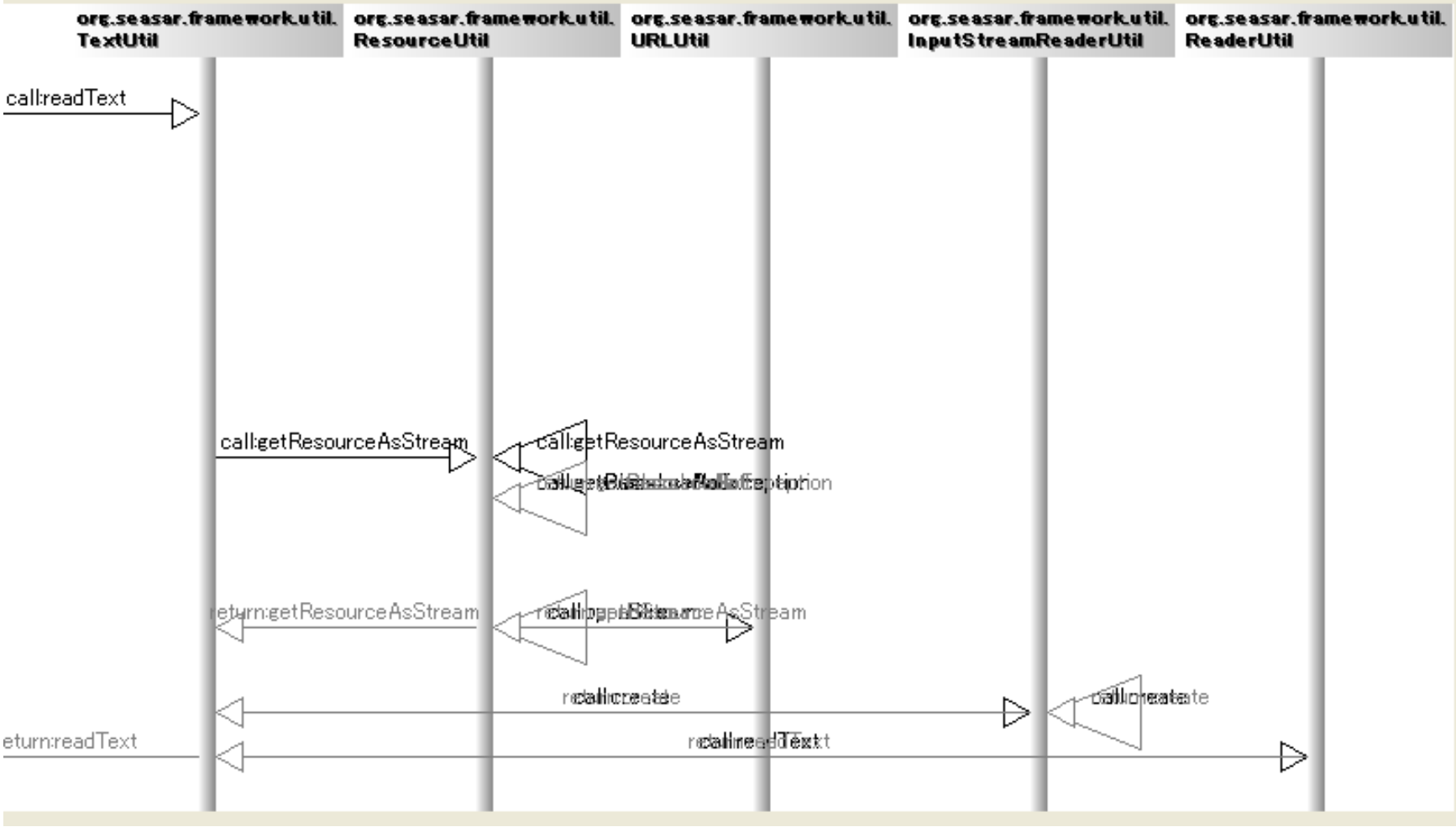
- Strutsとかライブラリの中のクラス
  - ライブラリもコンパイルしなおせばOK

現在は

- できます

Demo

Demo



- S2のUtilクラスのシーケンス図をS2Javelinで出力

内部的には

- バイトコードエンジニアリング
- Java5から動的にバイトコードを操作するための枠組みが提供されるようになった
  - - javaagent
    - クラスロード時に操作可能
      - 実行時、コンパイル時ではなく
    - JavaRebel
    - Pleiades

できること

- メソッドに仕掛ける
  - privateも
  - final,staticも
- コンストラクタに仕掛ける

基本スペック

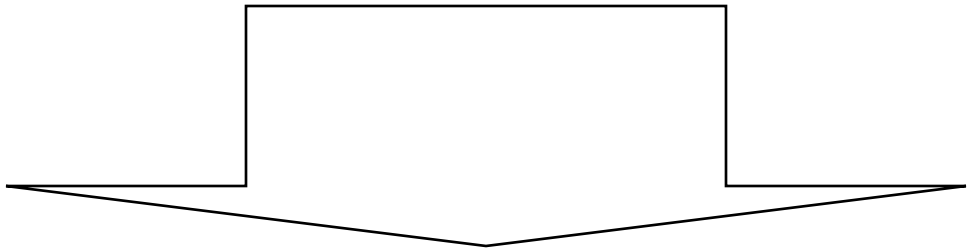
- AOP Alliance
  - ○ MethodInterceptor
  - ○ ConstructorInterceptor
  - × FieldInterceptor
- Java標準
  - javax.interceptor.
    - ○ AroundInvoke(AOP AllianceのMethodInterceptor)
    - ○ InvocationContext(AOP AllianceのInvocation)
    - ○ ExcludeClassInterceptors
    - × Interceptors

こんな時に使える

- バグFix
- 深い場所にあるクラスのログなど
- デバッグ時のtoString
  - 個人の生産性を高める
- Unitテスト
  - 戻り値や引数を強引に変更
- キャッシュ

こんな時に使える

- バグFix
  - ライブラリのバージョンを上げたい
    - でも、アプリのコードがコンパイルエラーになるので見送り



- 部分的なパッチを作りAOPで対応
    - アプリのコードが影響の出ないようにできる

使用方法

1. jarを配置
2. java のオプション指定

   java **-javaagent:kimu-aop-core.jar** foo.bar.Main

3. 設定ファイルを用意

設定ファイル解説

- 基本
  - ファイル名:aspect.xml
  - Seasar2のdiconファイルを意識

対象となるクラスを指定

```
<aspect-config>
  <def target="org.apache.struts.Action">
    <aspect>
      new jp.dododododo.aop.interceptors.TraceInterceptor()
    </aspect>
  </def>
</aspect-config>
```

機能(TraceInterceptorは、ログ)

設定ファイル解説

- 正規表現で指定

正規表現も可能

```
<aspect-config>
  <def target="jp.co.foo.bar.*ServiceImpl">
    <aspect>
      new jp.dododododo.aop.interceptors.TraceInterceptor()
    </aspect>
  </def>
</aspect-config>
```

設定ファイル解説

- メソッドの指定

pointcutで指定

```
<aspect-config>
<def target="jp.co.foo.bar.*ServiceImpl">
  <aspect pointcut="execute,toString">
    new jp.dododododo.aop.interceptors.TraceInterceptor()
  </aspect>
</def>
</aspect-config>
```

・指定しない場合は、implメソッド
・Genericsも対応

設定ファイル解説

- メソッドの指定

正規表現も可能

```
<aspect-config>
<def target="jp.co.foo.bar.*ServiceImpl">
  <aspect pointcut=".*">
    new jp.dododododo.aop.interceptors.TraceInterceptor()
  </aspect>
</def>
</aspect-config>
```

設定ファイル解説

- メソッドの指定

notで除外

```
<aspect-config>
<def target="jp.co.foo.bar.*ServiceImpl">
  <aspect pointcut=".*" not="toString,hashCode">
    new jp.dododododo.aop.interceptors.TraceInterceptor()
  </aspect>
</def>
</aspect-config>
```

設定ファイル解説

- アクセス修飾子の指定

・modifierで指定
・指定した修飾子以上

```
<aspect-config>
<def target="jp.co.foo.bar.*ServiceImpl">
  <aspect modifier="private">
    new jp.dododododo.aop.interceptors.TraceInterceptor()
  </aspect>
</def>
</aspect-config>
```

・public（デフォルト）
・protected
・package-private
・private

設定ファイル解説

- Interceptorの指定

```
<def target="jp.co.foo.bar.*ActionImpl">
 <aspect>
  new jp.dododododo.aop.interceptors.TraceInterceptor()
 </aspect>
</def>

<def target="jp.co.foo.bar.*ServiceImpl">
 <aspect>
  jp.dododododo.aop.interceptors.TraceInterceptor.getInstance()
 </aspect>
</def>
```

Javaのコードを
1ステートメントで記述

設定ファイル解説

- 優先度

・上に書いてある方が優先

```
<aspect-config>
  <def target="jp.co.foo.bar.BazServiceImpl">
    <aspect>
      new jp.dododododo.aop.interceptors.SimpleTraceInterceptor()
    </aspect>
  </def>
  <def target="jp.co.foo.bar.*ServiceImpl">
    <aspect>
      new jp.dododododo.aop.interceptors.TraceInterceptor()
    </aspect>
  </def>
</aspect-config>
```

設定ファイル解説

- 優先度

・AOP対象外

```
<aspect-config>
  <def target="org.apache.log4j.*" />

  <def target="jp.co.foo.bar.*ServiceImpl">
    <aspect>
      new jp.dododododo.aop.interceptors.TraceInterceptor()
    </aspect>
  </def>
</aspect-config>
```

特定のメソッドを対象外にする

- Seasar2だと
  - finalメソッドにする
  - Interfaceからメソッドを除く
  - pointcutで頑張る

```
public final String hello(String arg) {
        Strimg message = "Hello " + arg;
        System.out.println(message);
        return message;
}
```

特定のメソッドを対象外にする

- kimu-aop
  - Enhanceアノテーションで指定
  - ExcludeClassInterceptorsアノテーションで指定
  - (notで指定)

無限ループしないように
自作のインターセプタとかに付ける

```
@Enhance(false)
public String hello(String arg) {
        Strimg message = "Hello " + arg;
        System.out.println(message);
        return message;
}
```

特定のメソッドを対象外にする

- kimu-aop
  - Enhanceアノテーションで指定
  - ExcludeClassInterceptorsアノテーションで指定
  - (notで指定)

```
@ExcludeClassInterceptors
public String hello(String arg) {
        Strimg message = "Hello " + arg;
        System.out.println(message);
        return message;
}
```

用意しているInterceptor

- お約束（S2AOPと同じ）
  - ログ系
    - TraceInterceptor
    - SimpleTraceInterceptor
    - ThrowsInterceptor
      - TraceThrowsInterceptor
  - 同期
    - SyncInterceptor

用意しているInterceptor

- 独自
  - TraceWithCustomLoggerInterceptor
  - TakeOverInterceptor
    - CommonsBuilderInterceptor
    - ToStringInterceptor
    - CloneInterceptor

TraceWithCustomLoggerInterceptor

- クラス単位でTraceログの設定が可能
  - log4j.properties

TakeOverInterceptor

- Interceptorに定義しているメソッドにシグネチャが同じメソッドがあれば、そのメソッドを実行

CommonsBuilderInterceptor

- Objectクラスメソッドの実装

```
public class CommonsBuilderInterceptor
                    extends TakeOverInterceptor {
  public String toString() {
    Object target = getThis();
    return ToStringBuilder.reflectionToString(target);
  }
  public int hashCode() {
    Object target = getThis();
    return HashCodeBuilder.reflectionHashCode(target);
  }
  public boolean equals(Object obj) {
    Object target = getThis();
    return EqualsBuilder.reflectionEquals(target, obj);
  }
}
```

ToStringInterceptor

- 配列などに対応

CloneInterceptor

- シリアライズ、デシリアライズ

S2AOPとの違い

- エンハンスしたクラス
  - S2AOP:対象のクラスを継承したクラス
  - KimuAop:対象のクラスそのもの
    - ターゲットのクラス取得の作法が異なる（独自のInterceptorを作る場合に注意かも）
- InterceptorはSingletonでない
  - 定義時にgetInstance()とか書けばOK
- インターフェースを
  そのままインスタンス化できない
  - ex:S2Dao

今後

- Unitテスト用の仕組みを提供
  - (今もあるけど、変更する予定)

プロダクト情報

- 名前
  - Kimu-aop
- URL
  - http://code.google.com/p/kimu-aop/

まとめ

- AOPの乱用禁止

- kimu-aop
  - ほぼ何でも出来る
    - いままでリーチできなかったクラスにリーチ
  - S2AOPとの違いに注意

終わり

- ご清聴ありがとうございました