



# Next Generation Dependency Injection

Yasuo Higa  
Seasar Foundation/Chief Committer  
<http://www.seasar.org>

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

1



私は世界中の開発者にこう聞いてみたい

設定ファイルを書きたいのか  
書きたくないのか

Seasar2は、あなたをXML地獄から救い出します。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

2



## Agenda

- DIが生まれたきっかけ
- 現在のDI(Spring)の問題点
- 次世代DI(Seasar2)の進む道
- Seasar2 VS EJB3

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

3



## DIがうまれるさらに前

- コンポーネント化とは
  - ブラックボックス化されたコンポーネントを組み合わせてアプリケーションを組み立てること。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

4



## コンポーネント化のメリット・デメリット

- メリット
  - コンポーネントが再利用できる。
  - コンポーネント同士を簡単に組み合わせられる。
- デメリット
  - 特定の規約に従う(APIを実装する)必要がある。
    - 特定のAPIの実装にコストがかかる。
    - 異なる規約のコンポーネントは組み合わせることができない。



## コンポーネント化の光と影

- 光
  - GUIの世界ではActiveXに代表されるように成功。
- 影
  - 業務ロジックの世界ではことごとく失敗。
    - 再利用性が低いため特定のAPIを実装するコストをかけても元が取れない。



## 典型的失敗例が

# EJB

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

7



## EJB(SessionBean)の問題点

- 1つのコンポーネントをつくるのに必要なファイルが多くコストがかかる。
  - 2つのインターフェース
  - 1つの実装クラス
  - 設定ファイル
- 必要なものを1つにまとめてアプリケーションサーバにデプロイするのが面倒。
- さらに修正のたびにデプロイするのが面倒。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

8



## EJB(SessionBean)の問題点

- アプリケーションサーバの中でしか稼動できないので、テストするのが面倒。
- APIが複雑で仕様を覚えるのが面倒。



## EJBの悲惨な実体

- 標準ということで多くの開発者がチャレンジ。
- でも苦労することが多く結局普及せず。



## DIがうまれたきっかけ

- EJBの代替として登場。
- EJBの問題点を解決
  - 特定のAPIを実装する必要が無い。
  - いちいちデプロイする必要が無い。
  - アプリケーションサーバ外で動作可能。



## DIのコンセプト

- POJO(Plain Old Java Object)
  - 特定のAPIに依存しない。
    - 再利用性が向上。
    - 特定のAPIを覚えるコストがかからない。
    - アプリケーションサーバ外でも稼動。
    - テストが容易。



- オブジェクト間の依存関係をDIContainerが解決
  - 各オブジェクトはプロパティの型をインターフェースで定義し実装クラスには依存しない。
    - オブジェクト間の結合度が低くなるため保守性・再利用性が向上。
    - 実装をモックに簡単に交換できるのでテストが容易。
  - 実行時にDIContainerが実装クラスのオブジェクトを生成し依存関係を解決する。
    - 依存関係はXMLファイルに定義されることが多い。



- あいさつクラス
  - あいさつ用の文字列を返す。
- あいさつクライアントクラス
  - あいさつクラスから取得したあいさつ(文字列)をコンソールに出力する。
- あいさつメインクラス
  - 起動用のクラス。あいさつクラスとあいさつクライアントクラスの組み立ても行う。



**Seasar**  
DI Container with AOP

## Greeting.java

```
package examples.di;  
  
public interface Greeting {  
    String greet();  
}
```

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

15



**Seasar**  
DI Container with AOP

## GreetingImpl.java

```
package examples.di.impl;  
  
import examples.di.Greeting;  
  
public class GreetingImpl implements Greeting {  
    public String greet() {  
        return "Hello World!";  
    }  
}
```

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

16





## GreetingClient.java

```
package examples.di;  
  
public interface GreetingClient {  
    void execute();  
}
```

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

17



## GreetingClientImpl.java

```
package examples.di.impl;  
  
import examples.di.Greeting;  
import examples.di.GreetingClient;  
  
public class GreetingClientImpl implements GreetingClient {  
    private Greeting greeting;  
  
    public void setGreeting(Greeting greeting) {  
        this.greeting = greeting;  
    }  
  
    public void execute() {  
        System.out.println(greeting.greet());  
    }  
}
```

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

18



## beans.xml(Spring版)

```
<beans>
  <bean id="greeting"
        class="examples.di.impl.GreetingImpl"/>
  <bean id="greetingClient"
        class="examples.di.impl.GreetingClientImpl">
    <property name="greeting">
      <ref bean="greeting"/>
    </property>
  </bean>
</beans>
```



## GreetingMain.java(Spring版)

```
package examples.di.main;

import ...;

public class GreetingMain {

    public static void main(String[] args) {
        ClassPathResource res =
            new ClassPathResource("beans.xml");
        XmlBeanFactory factory = new XmlBeanFactory(res);
        GreetingClient greetingClient = (GreetingClient)
            factory.getBean("greetingClient");
        greetingClient.execute();
    }
}
```



## サンプルの確認ポイント

- 機能を利用するクラス(GreetingClientImpl)
  - 機能を提供するクラスのインターフェース(Greeting)でプロパティの型を宣言している。
  - 機能を提供する実装クラス(GreetingImpl)に依存していない。
- DIの設定ファイル(beans.xml)
  - コンポーネントの宣言とDIの情報が記述されている。



## よくあるDIへの疑問Q1

- 必ずインターフェースを使わなければいけませんか?
  - 必須ではありませんが使うことを推奨します。



## よくあるDIへの疑問Q2

- なぜインターフェースを使った方がいいのですか？
  - 仕様(インターフェース)が決まっていれば実装を意識する必要が無いからです。
    - 実装をモックなどに変えてテストがやりやすくなります。
    - 実装が完成するのを待たずにモックなどを使って並行に作業を進めることができます。



## よくあるDIへの疑問Q3

- 最初にインターフェースを考えるのが面倒なんです？
  - 仕様をきちんと決めずに実装をはじめていませんか。
  - 目的地を決めずにとりあえず歩いているようなものでいろいろ問題が起きやすくなるでしょう。
  - 最初に仕様をきちんと決めていればインターフェースを考えるのは簡単なことです。



## よくあるDIへの疑問Q4

- 最初に仕様をきちんと決めるのはXPのYAGNIに違反していませんか?
  - YAGNI(どうせ必要にはならないよ)は起こるかどうか分からないニーズに備えて過剰な設計をしてはいけないことを戒めるものです。
  - 仕様は起こるかどうか分からないニーズではなく**今必要なもの**です。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

25



## よくあるDIへの疑問Q5

- 最初に仕様をきちんと決めていればいきなり実装クラスを作っても良いですか?
  - Q2の回答にあるようにインターフェースを使うことはメリットがあります。
  - 仕様が決まっていれば作る手間はたいしたことないのでからできる限り作るようにしてください。
  - 1人で開発し1人でメンテナンスする以外はインターフェースを使うメリットのほうが大きいはずです。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

26



## よくあるDIへの疑問Q6

- ソースから相手のクラスの実装が直接追えないので分かりにくくなりませんか？
  - 相手の仕様が重要であり、実装を気にしてはいけません。
  - 何をしてくれるのかが重要で、どう実装しているかを気にしてはいけません。
  - 相手の実装に依存せず疎結合であることが再利用性・保守性を向上させるのです。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

27



## よくあるDIへの疑問Q7

- DIの設定をXMLに書くことが面倒なんですか？
  - その通りです。
  - それが現在のDI(Spring)の問題点です。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

28



## 現在のDI(Spring)の問題点

- XML地獄
  - 扱うコンポーネントの数が増えるとXMLファイルが膨れ上がりXML地獄に陥ってしまう。



## 次世代DI(Seasar2)の進む道

- Less Configuration
  - できる限り設定の記述量を減らす。
  - でもどうやって？



**Seasar**  
DI Container with AOP

## Less Configurationのポイントその1

- Convention over Configuration
  - 適切な規約を守っていれば、設定を特にしなくてもフレームワークが適切な設定をしてくれるという考え方。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

31



**Seasar**  
DI Container with AOP

## Convention over Configurationの例1

- Convention
  - プロパティの型をインターフェースで定義する。
- Auto Configuration
  - プロパティの型がインターフェースで、DIコンテナの中にインターフェースを実装したオブジェクトが1つあれば自動的に設定する。
  - あらゆる型を自動で設定すると危険だけど、対象をインターフェースに限れば**たいていの場合うまくいく**。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

32





**Seasar**  
DI Container with OOP

## Convention over Configurationの例2

- Convention
  - インターフェースの名前がXxxなら実装クラスはXxxImplにする。
- Auto Configuration
  - あるパッケージ配下を再帰的にクラス名がImplで終わっているクラスがあればDIコンテナに自動登録する。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

33



**Seasar**  
DI Container with OOP

## Convention over Configurationの結果

- コンポーネントの登録が不要。
- DIの設定が不要。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

34



```
<components>
  <component
    class="...FileSystemComponentAutoRegister">
    <initMethod name="addClassPattern">
      <arg>"examples.di.impl"</arg>
      <arg>"*.Impl"</arg>
    </initMethod>
    <initMethod name="registAll"/>
  </component>
</components>
```



- Configuration by Exception
  - 適切なデフォルト値を決め、指定がなければデフォルト値を適用する。
  - デフォルト値が適用できない場合に、明示的に設定するようにする。
  - Convention over Configurationによってできる限り明示的な設定は避けることが重要。



## Less Configurationのポイントその2

- Configuration by Exception
  - 設定はアノテーションを利用する。
  - アノテーションとXMLを比較すると対象のソースの直ぐ近くにある分アノテーションのほうが分かりやすい。



## Configuration by Exceptionの例

```
//明示的にhoge2を指定する場合
@Binding("hoge2")
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//自動的にバインディングしないことを指定する場合
@Binding(bindingType=BindingType.NONE)
public void setHoge(Hoge hoge) {
    ...;
}
```



## 3つのアノテーション

```
//Tigerアノテーション
@Binding("hoge2")
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//backport175アノテーション(JDK1.4でもOK)
/**
 * @...backport175.Binding("hoge2")
 */
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//定数アノテーション
public static final String hoge_BINDING = "hoge2";
```

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

39



## Less Configurationの対象はどれ?

- コンポーネントの宣言やDIに関する設定
  - 設定ファイルに記述すると面倒で間違いが起きやすくなるので極力減らすべき。
  - つまり対象。
- 環境に依存するパラメータ
  - データベースへの接続情報など環境に依存するパラメータは設定ファイルに外だしすべき。
  - つまり非対象。

© Copyright The Seasar Project and others 2004-2005. all rights reserved.

40



## EJB3の良くなった点

- POJOベースになった。
- 設定にアノテーションを使えるようになった。
  - Configuration by Exceptionの適用。



## EJB3がSeasar2に劣っている点

- Convention over Configurationの考え方が無いのでアノテーションをある程度書かなければいけない。
- デプロイが面倒。
- モックを使わないテストは、アプリケーションサーバにデプロイしてアプリケーションサーバ上で行う必要がある。
- AOPの機能が弱い。



## Summary

- EJBはコンポーネント化に失敗した。
- DIはPOJOをコンセプトにしてある程度受け入れられた。
- 現在のDI(Spring)は規模が大きくなるとXML地獄に陥る。
- 次世代のDI(Seasar2)はLess ConfigurationによりXML地獄問題を解決した。