

arcclamp 

アプリケーション・フレームワーク
"枠組み"の"ワケミ"を見抜こう

鈴木 雄介 アークランプ
<http://www.arcclamp.jp/>

フレームワーク
とは何か？

物事を構造化する
ための枠組み

まず
物事の構造化とは

物事がより小さな
単位に分解され、その
組み合わせで成り立っ
ている状態

例：
家は、
柱＋壁＋床

例：
戦略は、
戦術＋実行計画

例：
肉野菜炒めは、
肉＋野菜

再び
フレームワーク
とは

構造の
パターンや
テンプレート
を集めた
ガイドライン

例：
家は、
構造設計

例：
戦略は、
メソッド

例：
肉野菜炒めは、
レシピ

とりあえず、
アプリケーション
フレームワーク
とは

アプリケーションは、
コード+
コンポーネント+
ライブラリ+
コンフィグ...
で構造化されている

仕様を実装する時、
どんな構造にすべきか
を決めた枠組み

では、
アプリ*開発*の
フレームワーク
とは？

アプリ開発の
ポイントは
2つ

1. 属人性の排除

2. 情報劣化の防止

これらに向けて
策定された
具体的なルール
= 広義のフレームワーク

例：属人性排除
クエリの
記述・配置に
ルールを

例：情報劣化防止
バリデーション
仕様書の書き方
にルールを

では、
良いルールを
策定すれば
良いのか？

もちろん
ルール重要

でも、
最も重要なのは

“**実行性**”

守れない
ルールは、
ただのゴミ

100ページに及ぶ
コード規約書

守らないほうが
悪い？

守れないルールを
作るほうが悪い！

<<重要>>
実行性のある
ルールを！

本題

再び、
アプリケーション
フレームワーク
とは

ルールに実行性を
あたえるもの

例：属人性排除
特定の
記述・配置で
書かれたクエリで
O/Rマップ

例：情報劣化防止
バリデータXML
を仕様書から
自動生成

ルールを
守らないと
アプリが動かない
というか、

ルールに
従った方が
幸せ

例
ルール：
箱に卵を6個
つめたい

ありがち
ルールのみ



実行性のある
枠組み*だけ*用意



ルールを
気にしなくても
間違わない

フレームワークに
ルールが織り込ま
れているから

つまり、
アプリケーション
フレームワーク
とは

仕様をアプリ化す
る枠組みをソフト
ウェア化したもの

では、
どうやって作る？

車輪の
再発明は悪

既存の
プロダクトを
利用しよう

言語：
Java、Ruby、C#

Webアプリ・フレ：
Struts、Ruby on
Rails

どれを
選ぶべきか？

なぜ、
1つのカテゴリに
たくさんの
プロダクトが？

例：
気温の高い
地域の家の
構造設計

“高気密”
冷房効率アップ

“低気密”
風通し最高

どちらも正解
施主の考え方
=思想

プロダクトには
“何らかの思想”
がソフト化されて
いる

フレームワークと
プロジェクトの
思想がずれていると？

枠組みに逆らうと
ミスが増える



だからこそ、
枠組みのワケグミ
を見抜く

体感：
O/R マッパ

Hibernate
Torque
iBATIS

設定ファイルを
見れば一目瞭然

Hibernate

http://www.hibernate.org/hib_doc/v3/reference/en/html/mapping.html#mapping-declaration

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
1.2/EN" "http://hibernate.sourceforge.net/hibernate-mapping-1.0.dtd">
<hibernate-mapping package="org">
<class name="org.hibernate.tutorial.ch10.C"
id name="id">
<generator class="native"/>
</id>
<discriminator column="subclass" type="character"/>
<property name="weight"/>
<property name="birthdate" type="date" not-null="true" update="false"/>
<property name="color" type="eg_type.ColorUserType"
not-null="true" update="false"/>
<property name="age" not-null="true" update="false"/>
<property name="litterid" column="litterid" update="false"/>
</class>
</hibernate-mapping>

```

**<class>
<property>
オブジェクト指向**

Torque
<http://db.apache.org/torque/relnotes/torque-3.2/tutorial/step2.html>

```

<!DOCTYPE database SYSTEM "http://db.apache.org/torque/dtd/database_3.2.dtd">
<database name="bookstore" defaultMethod="iBrokeIt">
<table name="publisher" description="Publisher table">
<column name="publisher_id" required="true"
primary="true" type="INTEGER" description="Publisher id"/>
<column name="name" required="true" type="VARCHAR" size="128"
description="Publisher name"/>
</table>
</database>

```

**<database>
<table>
<column>
データベース指向**

iBATIS
<http://cvs.apache.org/dist/ibatis/ibatis.java/docs/IBatis-SqlMaps-2.pdf>

```

<sqlMap id="Product">
<select id="getProduct" parameterClass="com.ibatis.example.Product"
resultClass="com.ibatis.example.Product">
select
PID_ID as id,
PID_DESCRIPTION as description
from PRODUCT
where PID_ID = #id
</select>
</sqlMap>

```

**<sqlmap>
<select>
SQL指向**

かなり違います
hibernate : OOA
Torque : DOA
iBATIS : SqIOA(?)

体感：
DIコンテナ
Seasar2
SpringFramework

AOPに対する
設定ファイル
(ログを出力する)

Seasar2

```
<?xml version="1.0" encoding="Windows-31J"?>
<!DOCTYPE beans PUBLIC "-//SEASAR2.1//DTD S2Container//EN"
"http://www.seasar.org/dtd/components21.dtd">
<component name="traceInterceptor">
  <class>org.seasar.framework.aop.interceptors.TraceInterceptor</class>
  <component name="Hello" class="hoge.Hello">
    <property name="helloMsg">Hello</property>
    <aspect pointcut="*"Hello">seasarInterceptor</aspect>
  </component>
</component>
```

<aspect>あり
だって、
分かりやすいから

SpringFramework

```
<?xml version="1.0" encoding="Windows-31J"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<bean id="debugInterceptor">
  <class>org.springframework.aop.interceptor.DebugInterceptor</class>
</bean>
<bean id="debugAdvisor">
  <class>org.springframework.aop.support.AspectMethodPointcutAdvisor</class>
  <property name="advise">ref bean="debugInterceptor"/</property>
  <property name="patterns">value="Hello"/</property>
</bean>
<bean id="helloTarget" class="hoge.Hello">
  <property name="helloMsg" value="Hello"/>
</bean>
<bean id="hello">
  <class>org.springframework.aop.framework.ProxyFactoryBean</class>
  <property name="target">ref local="helloTarget"/>
  <property name="interceptorNames">
    <list>
      <value>debugAdvisor</value>
    </list>
  </property>
  <property name="proxyInterfaceName">
    <value>hoge.Hello</value>
  </property>
</bean>
</beans>
```

<aspect>なし
だって、
DIコンテナの機能
ではないから

補足

Seasar2は、開発者を楽にするという明確な思想があります。ですから、積極的にSeasarファミリーとして同じ思想のコンテナネットを提供しています。SeasarはDIコンテナというより、DIコンテナを中心としたWebアプリケーションフレームワークと捉えるべきでしょう。

SpringFrameworkは、無思想という思想と呼べるものです。経営されるソフトウェアの思想を守るために、Spring自身はなるべく汎用的な機能しか提供しません。ですから、DIについては、DIコンテナそのものを交換することが可能です。SpringはまさにDIコンテナであらうとしています。

よく見ていくと、Seasar2とSpringFrameworkは、そもそも違うカテゴリのプロダクトなのかもしれません。

では、
フレームワークの
選択基準は？

スキル、ツール、コ
スト、契約形態、拠
点、ハード、テーブ
ル設計、好み、政治、
人員リソース、運用
体制、占い...

では、
フレームワークは
誰が決める？

顧客、プロマネ、営
業、プログラマ、ア
ーキテクト、エンド
ユーザー、コンサル
タント、株主、社長、
細〇数字...

永遠のテーマ
解はない

大事なものは
フィット&ギャップ
を考えること

最後に心得

アプリケーション
フレームワーク
とは

エンジニアに
ルールを守らせる
ために存在する
わけではなく、

ルールによって
エンジニアを
助けるために
存在する

2006年4月発売
「開発の現場 vol.4」
同内容の記事あり