

コンテナ層

概略

S2 と Spring は Spring がかなり簡便化されたため、使い勝手もかなり近くなった。

DI コンテナはインスタンスを注入するものなのでインスタンス定義もかけなくてはならない。S2 と Spring は XML でプロパティ定義やコンストラクタ定義を書くことでインスタンスを設定する。Guice は Java コードで書けるので、インスタンスを Java コード内で生成して、オブジェクトを指定できる。

Guice はアノテーションと Java クラスとインターフェースのみで動くので、タイプセーフである。

また、Module クラスという単位で設定をクラスとして分離できる。

定義の簡便さ

Seasar2 と Spring はほぼ同じ。Seasar2 のほうがアスペクトの指定は比較的簡単かも。

Guice はメソッドでシンプルに定義できる。

Seasar 2

```
<component name="trace" class="jp.co.dwango.interceptor.LoggingInterceptor" />
<component name="me" class="jp.co.dwango.bean.UserImpl">
  <property name="userName">"自分"</property>
  <property name="friend">friend</property>
  <property name="address">shinjuku</property>
  <aspect pointcut="getAddresses">trace</aspect>
```

```
</component>

<component name="friend" class="jp.co.dwango.bean.UserImpl">
  <property name="userName">"友人"</property>
  <property name="address">shibuya</property>
</component>

<component name="shinjuku" class="jp.co.dwango.bean.AddressImpl">
  <property name="address">"新宿"</property>
</component>

<component name="shibuya" class="jp.co.dwango.bean.AddressImpl">
  <property name="address">"渋谷"</property>
</component>
```

Spring

コンポーネント定義

```
<bean id="shinjuku" class="jp.co.dwango.bean.AddressImpl">
  <property name="address" value="新宿" />
</bean>

<bean id="shibuya" class="jp.co.dwango.bean.AddressImpl">
  <property name="address" value="渋谷" />
```

```
</bean>

<bean id="friend" class="jp.co.dwango.bean.UserImpl">
    <property name="userName" value="友人" />
    <property name="address" ref="shibuya" />
</bean>

<bean id="user" class="jp.co.dwango.bean.UserImpl">
    <property name="userName" value="テスト" />
    <property name="address" ref="shinjuku" />
    <property name="friend" ref="friend" />
</bean>
```

アスペクト定義

```
<aop:config>
    <aop:pointcut id="pc" expression="execution(* getAddress())" />
    <aop:aspect id="exAspect" ref="afterAdvice">
        <aop:after pointcut-ref="pc" method="after" />
    </aop:aspect>
</aop:config>

<bean id="afterAdvice" class="jp.co.dwango.spring.advice.AfterAdvice" />
```

Guice

コンポーネント定義

```
protected void configure() {  
    Injector f = Guice.createInjector(new InnerModule());  
    User u = f.getInstance(User.class);  
    u.setUserName("本人");  
  
    User friend = f.getInstance(User.class);  
    friend.setAddress(new AddressImpl("渋谷"));  
    friend.setUserName("相手");  
  
    u.setFriend(friend);  
  
    Address address = f.getInstance(Address.class);  
    address.setAddress("新宿");  
    u.setAddress(address);  
  
    bind(User.class).annotatedWith(Me.class).toInstance(u);  
}
```

アスペクト指定

```
protected void configure() {  
    bind(User.class).to(UserImpl.class);  
    bind(Address.class).to(AddressImpl.class);  
    bindInterceptor(subclassesOf(User.class), begin("get"), new LoggingInterceptor());  
}
```

他のフレームワークとの親和性

どのコンテナも、フレームワークとの統合を考えずに API 経由で呼び出す分には使うことができる。

フレームワークとの統合を考えると、Seasar 系フレームワークは Seasar2 に強く結びついている。なしでも使えるようだが、Seasar2 を使ってやっている部分を手動でやる必要があるため、現実的ではないし、利点が失われるので、S2 と組み合わせるのがベストな選択と思われる。

Wicket は三つのコンテナすべてに対して統合機能をもっていたが、Seasar との統合を行う S2Wicket だけが開発が止まっており、最新版の Wicket 1.4 で動かない（どうも 1.2 までしかサポートしてないように見える）。Spring と Guice については Apache Wicket プロジェクトで拡張機能が開発されていて、Wicket のバージョンアップにあわせて拡張機能もバージョンアップされている。

下図は、サポート関係にあるものを同色で表している。

SAStruts	Wicket	
Seasar 2	Spring	Google Guice
JPA(Hibernate)	S2JDBC	Cayenne