intra-mart WebPlatform/AppFramework Ver.7.2

Seasar2 連携 プログラミングガイド

2010/04/01 初版

＜＜　変更履歴　＞＞

変更年月日	変更内容
2010/04/01	初版

＜＜　目次　＞＞

1 はじめに

1.1 目的

Seasar2 は DI (Dependency Injection) と AOP (Aspect Oriented Programming) をサポートした軽量コンテナである。ここでは im-JavaEE-Framework と Seasar2 の連携方法について述べる。

1.2 Seasar2 プロダクト

intra-mart7.2にはいくつかの Seasar2 プロダクトが組み込まれている。以下が組み込まれているプロダクトの一覧である。

- S2Container 2.4.34
- S2Struts 1.3.1
- S2Dao 1.0.49

Seasar2 プロダクトについてはThe Seasar FoundationのWebサイトに詳しい情報が記載されている

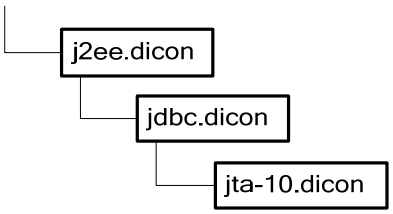
http://www.seasar.org/

2 セットアップ

2.1 トランザクションマネージャとデータソース

S2Container は独自にトランザクションマネージャやコネクションプールを実装しているがイントラマートが動作するアプリケーションサーバ上で利用するには、S2Container はアプリケーションサーバのトランザクションマネージャとコネクションプールを利用してデータベースにアクセスする必要がある。ここではその連携方法を記載する。
イントラマートのデフォルトの dicon ファイルは以下のようにインクルードされている。

＜リスト 2-1 dicon ファイルの構成＞



2.1.1 トランザクションマネージャの設定

S2Container を使用するために、トランザクションマネージャを dicon ファイルに設定する必要がある。
イントラマートでは以下の設定が標準で有効になっている。

＜リスト 2-2 jta-10.dicon＞

```
<component
    class="org.seasar.extension.tx.adapter.JTAUserTransactionAdapter"/>
```

2.1.2 データソース

S2Container が使用するデータソースの設定を行う。
intra-mart はログイングループ毎に異なるデータソースに接続しなければならない。
ユーザがログインするグループ毎に動的にデータソースを取得しなければならないため、通常のデータソースでは動的な取得はできない。AutoDetectedDataSource を利用することで、ログインしたユーザによって、動的にデータソースを利用することが可能である。intra-mart には以下のの設定が標準で有効になっている。

＜リスト 2-3 jdbc.dicon＞

```
<component name="dataSource"
  class="javax.sql.DataSource">
  @org.seasar.extension.j2ee.JndiResourceLocator@lookup("java:comp/env/jdbc/DataSource")
</component>
```

3 アプリケーションの作成

ここでは intra-mart と Seasar2 を連携させたアプリケーションの作成方法を説明する

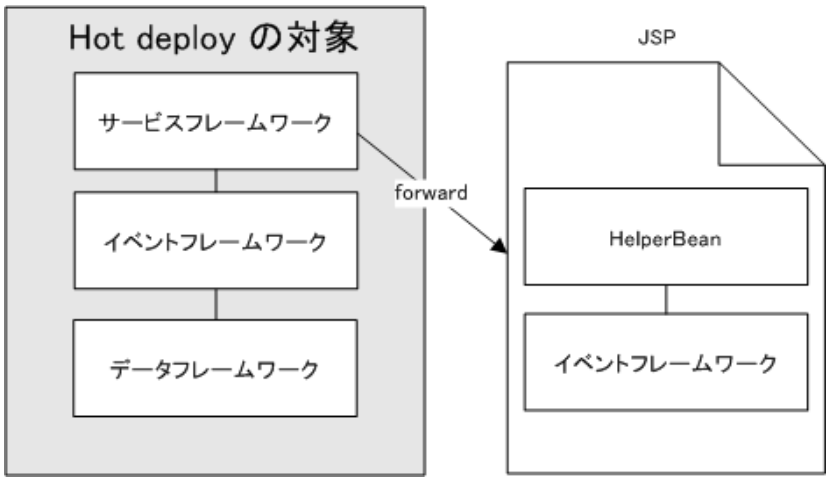
3.1 HOT deploy

HOT deploy を利用することで、アプリケーションサーバを再起動することなくソースコードの修正が即座に反映される。この章では Hot deploy の仕様を記載する。

HOT deploy は、各開発者が個人の PC でテストするときに使うことを想定した機能である。そのため、リクエストを複数同時に処理することはできない。運用時、複数の開発者で利用する場合に、HOT deploy を利用することは推奨されない。

3.1.1 対象となるコンポーネント

以下の図は Hot deploy の有効範囲を表したものである。



Hot deploy はサービスフレームワーク、イベントフレームワーク、データフレームワークで使用可能であるが、サービスフレームワークから forward された JSP では使用できない。そのため、JSP で実行される HelperBean および HelperBean から実行されるイベントフレームワークも Hot deploy の対象外となる。

HOT deploy の対象となるコンポーネントを以下に記す。

サービスフレームワーク	ServiceController
	Transition
	ControllerConverter
	ControllerObject
	Validator
イベントフレームワーク	Event
	EventListener
	EventTrigger
	EventResult
データフレームワーク	DAO

これらのコンポーネント及び、これらのコンポーネントから使用されるクラスは Hot deploy の対象となる。

HOT deploy の対象外となるコンポーネントは以下のものである。

JSP で使用されるコンポーネント	HelperBean クラス
	HelperBean タグから実行されるイベントフレームワーク
	その他 JSP 使用されるクラス

3.1.2 Hot deploy を利用する開発

この章では実際に Hot deploy を利用する開発方法を解説する。

3.1.2.1 Hot deploy の設定

Hot deploy を利用するために必要な設定を以下に記す。

3.1.2.1.1 web.xmlの設定

「doc/imart/WEB-INF/web.xml」に記述されているコメントアウトを削除し、HotdeployFilter を有効にする。

＜リスト 3-1 web.xml＞

```
・・・

<!-- ←この行を削除する
    <filter>
        <filter-name>hotdeployfilter</filter-name>
        <filter-class>org.seasar.framework.container.hotdeploy.HotdeployFilter</filter-class>
    </filter>
--> ←この行を削除する

・・・

<!-- ←この行を削除する
    <filter-mapping>
        <filter-name>hotdeployfilter</filter-name>
        <servlet-name>ServiceServlet</servlet-name>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>
--> ←この行を削除する

・・・
```

3.1.2.1.2 im_hotdeploy.diconの設定

「doc/imart/WEB-INF/classes/im_hotdeploy.dicon」に記述されているコメントアウトを削除し、convention.dicon、customizer.dicon、creator.dicon の include とコンポーネント HotdeployBehavior を有効にする。

＜リスト 3-2 im_hotdeploy.dicon＞

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
  "http://www.seasar.org/dtd/components24.dtd">
<components>
  <!-- ←この行を削除する
  <include path="convention.dicon"/>
  <include path="customizer.dicon"/>
  <include path="creator.dicon"/>
  <component class="org.seasar.framework.container.hotdeploy.HotdeployBehavior"/>
  --> ←この行を削除する
</components>
```

3.1.2.1.3 convention.diconの設定

「doc/imart/WEB-INF/classes/convention.dicon」に Hot deploy の対象とするクラスが格納されている Java パッケージパスを記述する。

＜リスト 3-3 convention.dicon＞

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
  "http://www.seasar.org/dtd/components24.dtd">
<components>
  <component class="org.seasar.framework.convention.impl.NamingConventionImpl">
      <initMethod name="addRootPackageName">
          <arg>"org.seasar.framework.container.warmdeploy"</arg>
      </initMethod>

      <!-- Hot deploy 対象とするパッケージパスを追加 -->
      <initMethod name="addRootPackageName">
          <arg>"test.foo. "</arg>
      </initMethod>

  </component>
  <component class="org.seasar.framework.convention.impl.PersistenceConventionImpl"/>
</components>
```

以上の設定を行うことで、Hot deploy が有効となる。

3.1.2.1.4 property-config.xmlの設定

doc/imart/WEB-INF/classes/property-config.xml の設定を変更することで、サービスフレームワーク、イベントフレームワーク、データフレームワークの各コンフィグファイルをアプリケーションサーバを再起動することなくファイルの変更を反映することが可能である。
各プロパティハンドラのパラメータ「dynamic」を true にすることで、コンフィグファイルの動的読み込みとなる。

動的読み込みは、開発時を想定した機能である。運用時などリクエストを複数同時に処理する場合に使用することは推奨されない。

＜リスト 3-4 property-config.xml＞

```
<service>
    <handler-class>jp.co.intra_mart.framework.base.service.XmlServicePropertyHandler</handler-class>
    <init-param>
        <param-name>dynamic</param-name>
        <param-value>true</param-value>
    </init-param>
</service>

<event>
    <handler-class>jp.co.intra_mart.framework.base.event.XmlEventPropertyHandler</handler-class>
    <init-param>
        <param-name>dynamic</param-name>
        <param-value>true</param-value>
    </init-param>
</event>

<data>
    <handler-class>jp.co.intra_mart.framework.base.data.XmlDataPropertyHandler</handler-class>
    <init-param>
        <param-name>dynamic</param-name>
        <param-value>true</param-value>
    </init-param>
</data>
```

3.1.2.2 Hot deploy を利用するプログラム設計

Hot deploy を有効にした場合、サービスフレームワーク、イベントフレームワーク、データフレームワークで Hot deploy が有効となるが、JSP では Hot deploy は利用できない。そのため、HelperBean を利用するなど JSP に多くのコードを記述することは、開発生産性の面から推奨されない。
そのため、JSP は画面が必要とする値のみを受け取り、HelperBean を使用しないで表示のみを行う方法が推奨される。
以下にそのプログラム例を記載する。

JSP に値を表示するための情報を格納する JavaBean を作成する。

```
public class FooModel {
   private String arg1;
   private int arg2;
   private Date arg3;
   public String getArg1() {
      return arg1;
   }
   public void setArg1(String arg1) {
      this.arg1 = arg1;
   }
   public int getArg2() {
      return arg2;
   }
   public void setArg2(int arg2) {
      this.arg2 = arg2;
   }
   public Date getArg3() {
      return arg3;
   }
   public void setArg3(Date arg3) {
      this.arg3 = arg3;
   }
}
```

Transition#setInformation()を実装する。
ServiceController の処理結果を JavaBean に格納し、request の属性に設定する。

```
public class FooTransition extends DefaultTransition {

   public void setInformation() throws TransitionException {

      FooServiceResult result = (FooServiceResult)getResult();
      FooModel model = new FooModel();
      model.setArg1(result.getID());
      model.setArg2(result.getPrice());
      model.setArg3(result.getDate());
      getRequest().setAttribute("item", model);
   }
}
```

JSP ではカスタムタグを利用して、画面に表示する。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
<html>
<body>

String
${item.arg1}<br>

Number
<imartj2ee:Format value="${item.arg2}" format="#,##,###" /><br>

Date
<imartj2ee:Format value="${item.arg3}" format="yyyy/mm/dd HH:mm:ss" /><br>

</body>
</html>
```

この例では EL を使用して、request の属性を参照し、その値をカスタムタグで整形し画面に表示している。
このようにして、request スコープに値を格納し、カスタムタグや JSTL(JSP Standard TagLibrary)を利用することで、JSP に処理を記述することなくアプリケーションを構築することが可能である。

JavaBeans Component API (http://java.sun.com/j2se/1.5.0/ja/docs/ja/ guide/beans/index.html)
JSP Standard Tag Library （http://java.sun.com/products/jsp/jstl/）
j2ee 1.4 Expression Language (http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html)

Hot deploy を有効にして開発を行う場合、java.lang.LinkageError が発生する場合がある。
これは Hot deploy に対象に設定しているクラスが、実行中のスレッドのコンテキスト ClassLoader に
既にロードされている可能性がある。
これを回避するには Hot deploy に対象と非対称のクラスを明確に分け、
実行中のスレッドのコンテキスト ClassLoader に先に読み込ませないようにすることで回避することができる。

3.2 DIコンテナを使用する

3.2.1 はじめに

この章では im-JavaEE-Framework のコンポーネントに対して自動的に DI およびアスペクトを適用する方法を説明する。

この機能は Hot deploy との併用はできない。
この機能を有効にするには「doc/imart/WEB-INF/classes/imartContainer.properties」を以下のように変更する必要がある。この設定を変更することにより、Hot deploy 機能は無効となる。

```
#serviceContainer=jp.co.intra_mart.framework.base.service.container.ServiceContainerImpl
#eventContainer=jp.co.intra_mart.framework.base.event.container.EventContainerImpl
#dataContainer=jp.co.intra_mart.framework.base.data.container.DataContainerImpl

serviceContainer=jp.co.intra_mart.framework.extension.seasar.service.S2ServiceContainer
eventContainer=jp.co.intra_mart.framework.extension.seasar.event.S2EventContainer
dataContainer=jp.co.intra_mart.framework.extension.seasar.data.S2DataContainer

#objectProvider=jp.co.intra_mart.framework.extension.seasar.system.object.HotdeployObjectProvider
```

3.2.2 DIの適用

im-JavaEE-Framework のコンポーネントは S2Container によって管理されている。サービスフレームワーク、イベントフレームワーク、データフレームワークの設定ファイルに定義されている各オブジェクトは自動的に S2Container に登録されオブジェクト同士のバインド対象となる。具体的に DI 可能なオブジェクトを「＜リスト 3-1 DI 対象のオブジェクト＞」に記す。

＜リスト 3-5 DI 対象のオブジェクト＞

サービスフレームワーク	ServiceController Transition
イベントフレームワーク	Event EventListener
データフレームワーク	DAO

例として ServiceController に DI を適用する例を以下に示す。

まず doc/imart/WEB-INF/classes/examples/dicon/foo.dicon を作成し、ServiceController にバインドするコンポーネント定義を記述する。

＜リスト 3-6 foo.dicon＞

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC
  "-//SEASAR//DTD S2Container 2.4//EN"
  "http://www.seasar.org/dtd/components24.dtd">
<components>
  <component class="examples.service.impl.FooServiceImpl"/>
</components>
```

doc/imart/WEB-INF/classes/app.dicon に foo.dicon をインクルードする。

＜リスト 3-7 app.dicon＞

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC
  "-//SEASAR//DTD S2Container 2.4//EN"
  "http://www.seasar.org/dtd/components24.dtd">
<components>
  <include path=" examples/dicon/foo.dicon" />
</components>
```

バインドするコンポーネントのインターフェースと実装を作成する。

＜リスト 3-8 FooService.java＞

```
package examples.service;
public interface FooService {
  String doSomething();
}
```

＜リスト 3-9 FooServiceImpl.java＞

```
package examples.service.impl;
import examples.service.FooService;
public class FooServiceImpl implements FooService {
  public FooServiceImpl() {
  }
  public String doSomething() {
      return "bar";
  }
}
```

以上で ServiceController にバインドするコンポーネントの準備は完了である。次に ServiceController 本体と設定ファイルを作成する。

＜リスト 3-10 BarServiceController.java＞

```
package examples.controller.service;
import jp.co.intra_mart.framework.base.service.RequestException;
import jp.co.intra_mart.framework.base.service.ServiceControllerAdapter;
import jp.co.intra_mart.framework.base.service.ServiceResult;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;
import examples.service.FooService;

public class BarServiceController extends ServiceControllerAdapter {

  private FooService service_ = null;

    public void check() throws RequestException, SystemException {
    }

    public ServiceResult service()
        throws SystemException, ApplicationException {
      System.out.println(service_.doSomething());
        return null;
    }

  public void setFooService(FooService service) {
      service_ = service;
  }
}
```

＜リスト 3-11 service-config-foo.xml＞

```
<?xml version="1.0" encoding="UTF-8"?>
<service-config>
    <service>
        <service-id>do_something</service-id>
<controller-class>examples.controller.service.BarServiceController</controller-class>
        <next-page>
            <page-path>/index.jsp</page-path>
        </next-page>
    </service>
</service-config>
```

service-config-foo.xml に設定した BarServiceController は自動的に S2Container に登録され、自動バインドの対象となり、実行時には FooService の実装が自動的にインジェクションされる。
同様にイベントオブジェクトも以下のようにイベントフレームワークの設定ファイルに記述するだけで DI が適用される。

＜リスト 3-12 event-config-foo.xml＞

```
<?xml version="1.0" encoding="UTF-8"?>
<event-config>
    <event-group>
        <event-key>do_something</event-key>
        <event-class>examples.model.event.FooEvent</event-class>
        <event-factory>
<factory-class>
jp.co.intra_mart.framework.extension.seasar.event.S2EventListenerFactory
</factory-class>
            <init-param>
                <param-name>listener</param-name>
                <param-value>examples.model.event.FooEventListener</param-value>
            </init-param>
        </event-factory>
    </event-group>
</event-config>
```

＜リスト 3-13 FooEvent.java＞

```
package examples.model.event;
import examples.service.FooService;
import jp.co.intra_mart.framework.base.event.Event;

public class FooEvent extends Event {

  private FooService service_ = null;

  public void setFooService(FooService service) {
      service_ = service;
  }

public FooService getFooService() {
      return service_;
  }
}
```

3.2.3 アスペクトの適用

im-JavaEE-Framework のコンポーネントにアスペクトを適用する方法について記述する。im-JavaEE-Framework のコンポーネントはサービスフレームワーク、イベントフレームワーク、データフレームワークの各設定ファイルに記述するだけで、S2Container に登録されている。このとき各コンポーネントにはコンポーネント名が設定されている。以下がそのコンポーネント以下のようになる。

＜リスト 3-14 コンポーネント名＞

ServiceController	“applicationID”-“serviceID”-controller
Transition	“applicationID”-“serviceID”-transition
Event	“applicationID”-“eventKEY”-event_object
EventListener	“applicationID”-“eventKEY”-event_listener
DAO	“applicationID”-“daoKEY”-dao

このコンポーネント名を指定することで im-JavaEE-Framework のコンポーネントにアスペクトを適用することができる。以下がその例である。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC
  "-//SEASAR//DTD S2Container 2.4//EN"
  "http://www.seasar.org/dtd/components24.dtd">
<components>
  <component class="examples.service.impl.FooServiceImpl"/>
<component name="foo-do_event-event_object"
    class="sample.model.event.SampleEvent">
    <aspect pointcut="getUserInfo">
      <component class="org.seasar.framework.aop.interceptors.TraceInterceptor"/>
    </aspect>
  </component>
</components>
```

3.3 イントラマートのセッション情報を取得する

イントラマートのセッション情報を S2Container で利用するために doc/imart/WEB-INF/classes/imart.dicon に以下のコンポーネントが登録されている。

＜リスト 3-15 AutoDetectedUserInfo＞

```
<component name="userInfo"
        class="jp.co.intra_mart.framework.extension.seasar.util.AutoDetectedUserInfo"
        instance="request"/>
```

AutoDetectedUserInfo は UserInfo インターフェースを実装したコンポーネントである。S2Container が保持するコンポーネントが UserInfo インターフェースのバインド対象である場合。自動的に AutoDetectedUserInfo はインジェクションされる。以下がその例である。

＜リスト 3-16 Client.java＞

```
package examples;
public interface Client {
    String execute();
}
```

＜リスト 3-17 ClientImpl.java＞

```
package examples.impl;
import jp.co.intra_mart.framework.base.util.UserInfo;
import examples.Client;
public class ClientImpl implements Client {

  private UserInfo userInfo_;

    public void setUserInfo(UserInfo userInfo) {
        this.userInfo_ = userInfo;
    }

  public String execute() {
      return userInfo_.getUserID();
  }
}
```

＜リスト 3-18 example.dicon＞

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE components PUBLIC
    "-//SEASAR//DTD S2Container 2.4//EN"
    "http://www.seasar.org/dtd/components24.dtd">
<components>
  <include path="imart.dicon" />
    <component name="client" class="examples.impl.ClientImpl" instance="prototype">
        <property name="userInfo">userInfo</property>
    </component>
</components>
```

このように設定することで Client コンポーネント取得時に UserInfo が自動的にインジェクションされる。

イントラマートのデータベース接続設定を利用する。
イントラマートに設定されているデータベース接続設定を利用してデータソースを取得することが可能である。

＜リスト 3-19 LoginGroupDataSource＞

```
<component class="jp.co.intra_mart.framework.extension.seasar.util.LoginGroupDataSource">
    <arg>initialContext</arg>
    <arg>"default"</arg>
</component>
```

＜リスト 3-20 SystemDataSource＞

```
<component class="jp.co.intra_mart.framework.extension.seasar.util.SystemDataSource">
    <arg>initialContext</arg>
    <arg>"default"</arg>
</component>
```

LoginGroupDataSourceとSystemDataSourceはそれぞれjavax.sql.DataSourceの実装である。コンストラクタの第一引数には初期コンテキストを与え、第二引数にはLoginGroupDataSourceの場合にはログイングループID、SystemDataSourceはシステムには識別子を設定する必要がある。「2.1.1 トランザクションマネージャの設定」で設定したAutoDetectedDataSourceを利用することで接続先を自動的に解決することも可能だが、LoginGroupDataSourceやSystemDataSourceを利用することで明示的に接続先を指定することも可能である。

intra-mart WebPlatform/AppFramework Ver.7.2
Seasar2 連携 プログラミングガイド

2010/04/01 初版

TEL: 03-5549-2821
FAX: 03-5549-2816
E-MAIL: info@intra-mart.jp
URL: http://www.intra-mart.jp/