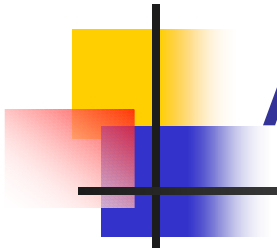
オープンソースによる開発率向上
～アンチJ2EEなEoD～

2005/12/09
(株)NTTデータ
基盤システム事業本部
岡本隆史

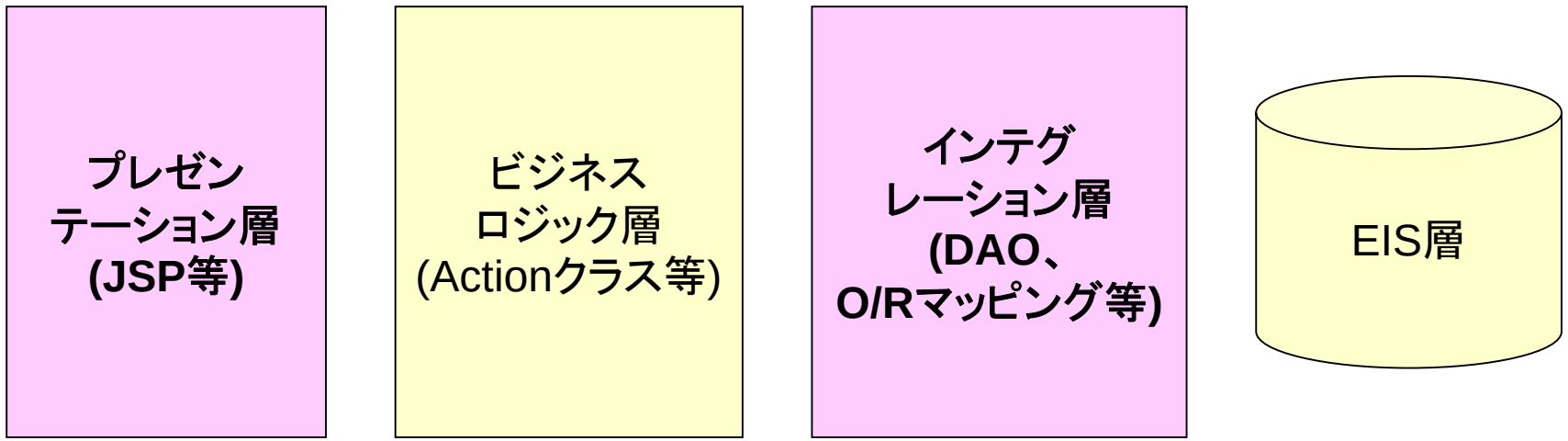


Agenda

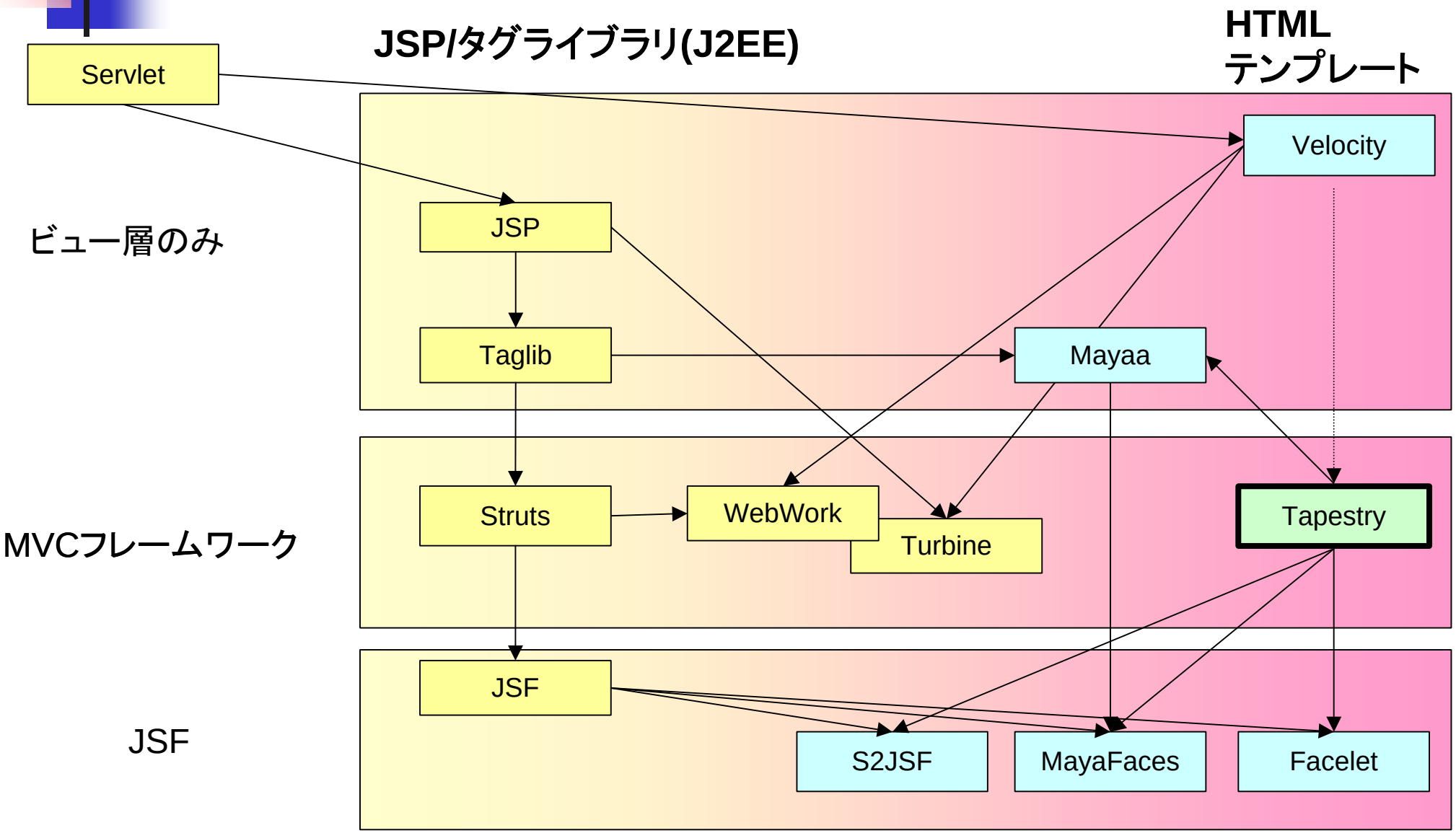
- サーバサイト技術の進化と
  Tapestry/Hibernateの位置づけ
- TapestryによるプレゼンテーションEoD
- HibernateによるインテグレーションEoD
- まとめ

サーバサイドJava技術の進化からみたTapestryとHibernateの位置づけ

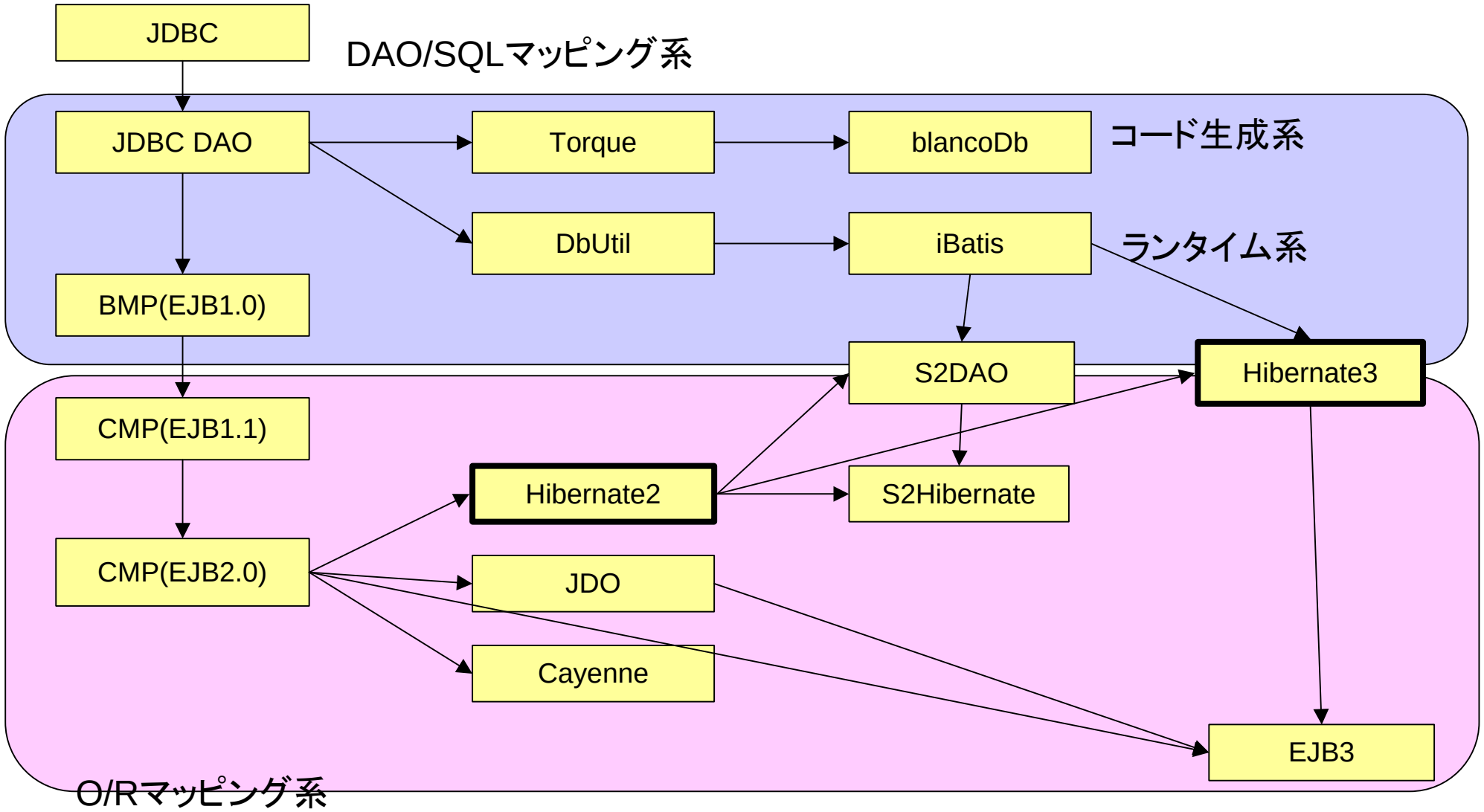
- プレゼンテーション層のフレームワークの進化
- インテグレーション層のフレームワークの進化



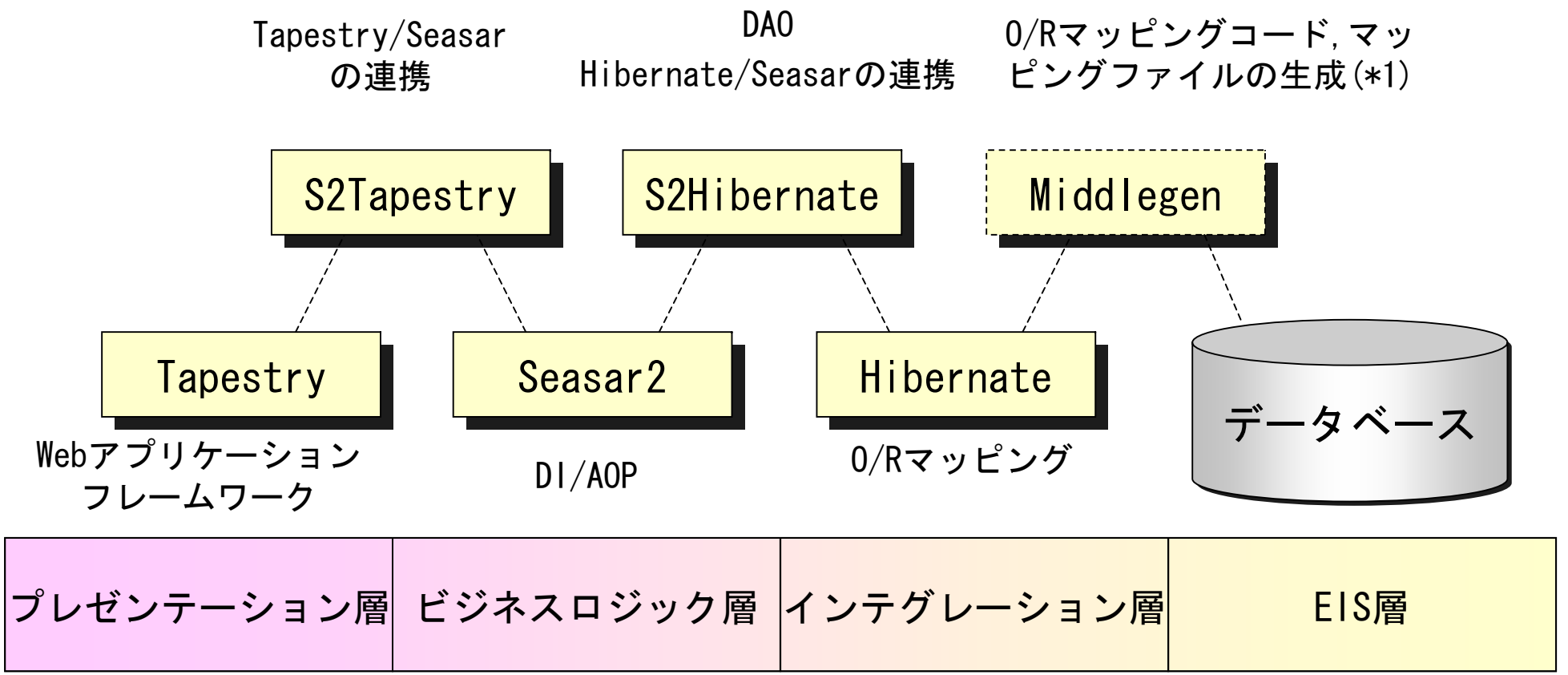
プレゼンテーション層のフレームワークの進化



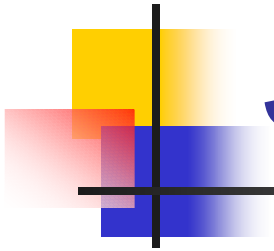
インテグレーション層のフレームワークの進化



オープンソースによるEoD

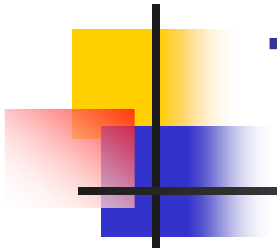


*1:アプリケーション実行時は不要



J2EE技術との比較

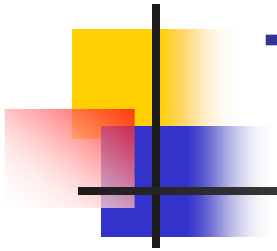
- J2EE(JSF/EJB3)におけるEoD
  - 標準化
  - 長いプロダクトサイクル
  - ツール依存
  - 未成熟
- Tapesty/Hibernate/SeasarによるEoD
  - 非標準
  - 短いプロダクトサイクル
  - ツール非依存
  - 将来の技術の基礎
  - 導入実績多数



Tapestryによる プレゼンテーション層のEoD

JSP(Struts/JSF)の抱える問題

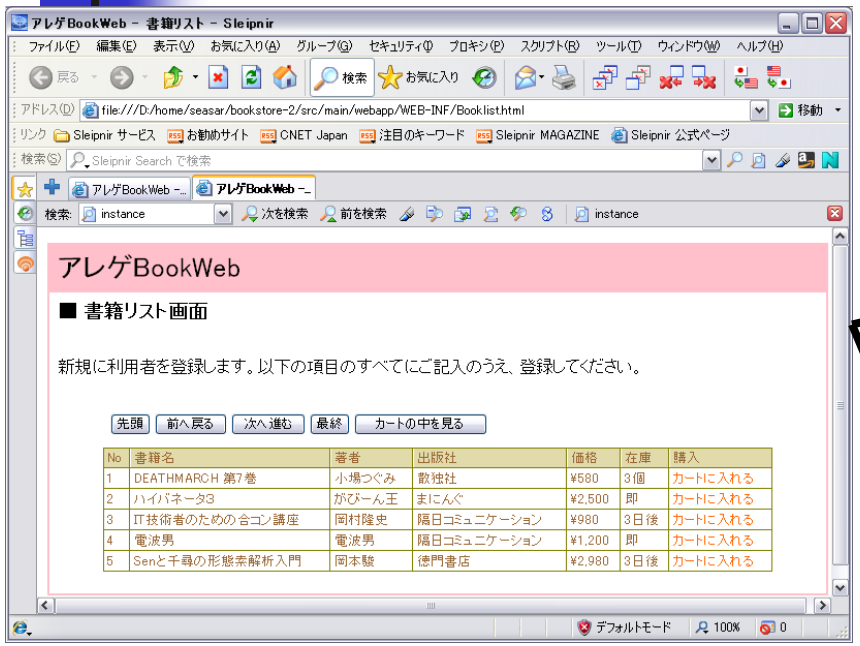
- 専用のタグを利用
  - Webのオーサリングツール/ブラウザ表示不可能
  - デザイナからプログラマに渡ると二度とデザイナは編集できない
  - 画面イメージが確認できない
  - デザインの変更/デバッグコストが高い
- WYSIWIGな編集に専用ツールが必要
  - デザイナは利用できない
  - ツールの修得にコストが掛かる
  - 使い勝手が使い慣れたオーサリングツール程良くない

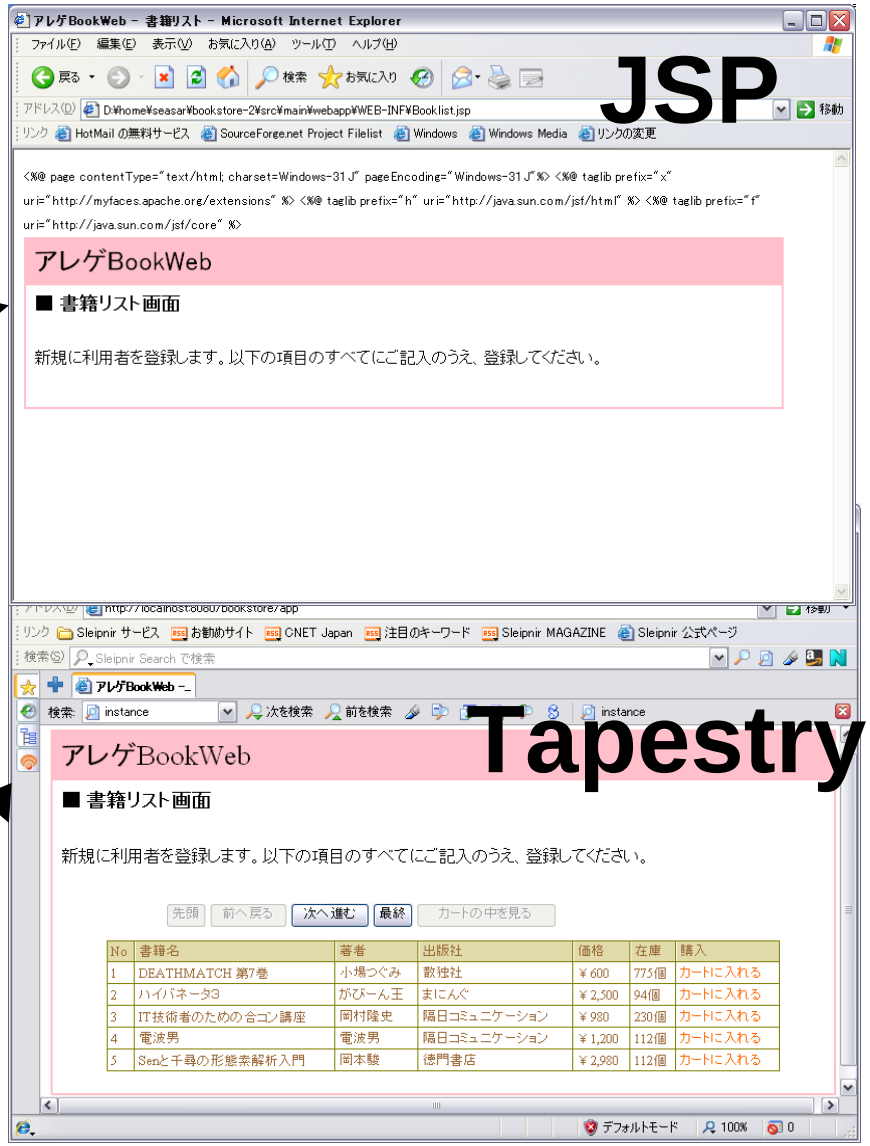


Tapestryの特徴(1)

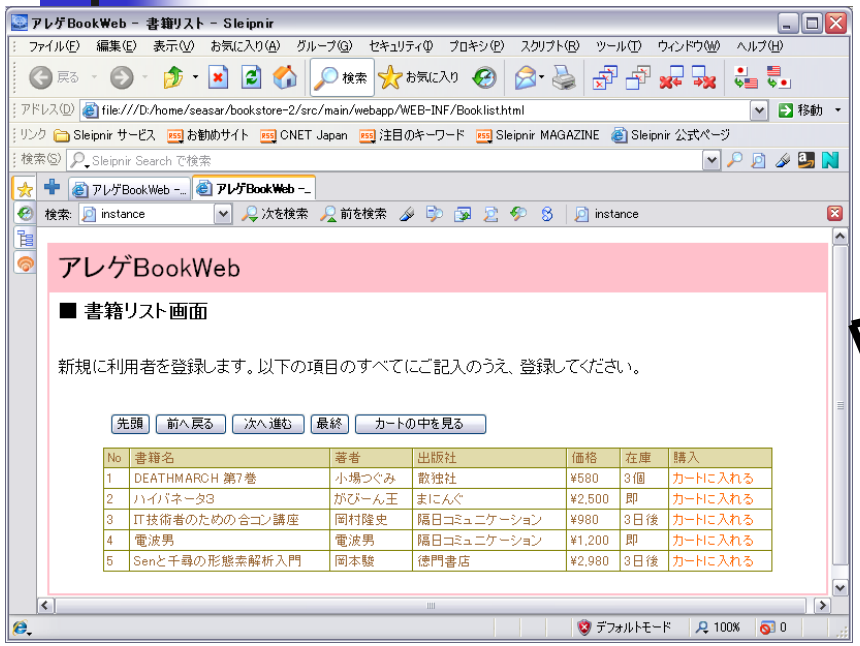
- **シンプルなHTMLテンプレート(不可視なJSPタグは不要)**
- 当初SourceForgeで開発し、後にJakartaプロジェクトへ移動
- ポータブル(Servletのみ利用)
- Servlet APIの隠蔽化
- 再利用可能なコンポーネント
- インスペクタ
- バリデーション機能
- JavaScriptサポート

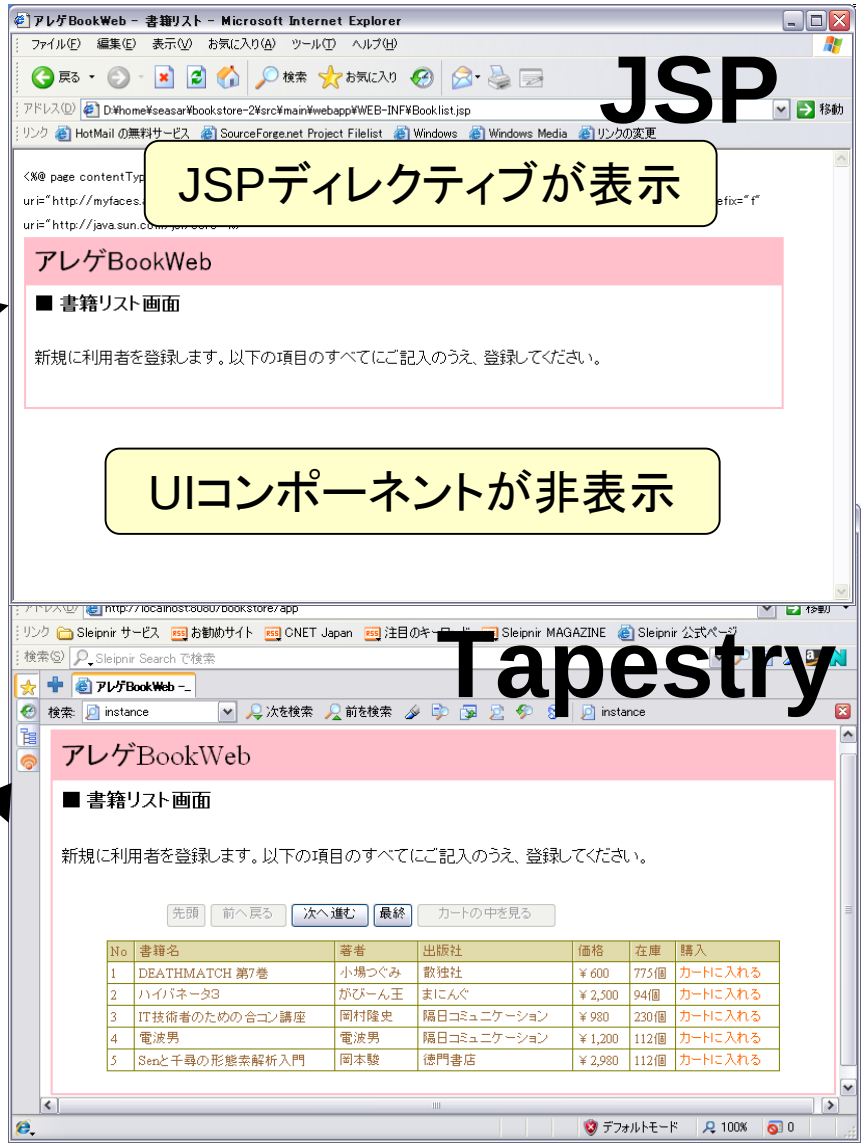
JSPとTapestryのビューの比較





JSPとTapestryのビューの比較

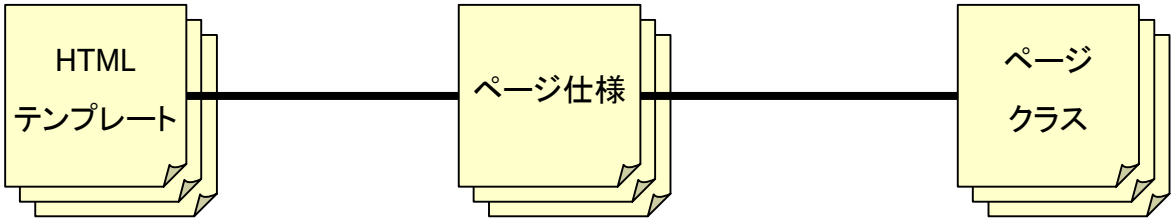




JSPはJSPタグを利用しているため、Webのオーサリングツール/ブラウザでデザインが破壊されるが、Tapestryはデザインを保ったまま、ビューを作成可能。

Tapestryの概要

- HTMLテンプレート
  - HTMLをベースとしたブラウザ/オーサリングツールで閲覧可能なページ
  - HTML内へのパラメータ埋め込みとページ仕様への外出しを選択可能
- ページクラス
  - 画面からの入出力
  - リスナの実装
- ページ仕様
  - ページクラスとHTMLテンプレートのバインド
  - バリデーションの設定



Tapestryの仕組み(1)

ページ仕様でコンポーネントを定義

・HTMLテンプレートとページクラスとをページ仕様で関連付ける

・HTMLタグをjwcid属性でコンポーネントにバインド

ユーザID	
パスワード	

HTMLテンプレート(Login.html)

```
ユーザID:   <input type="text" jwcid="name" /><br />
パスワード: <input type="password" jwcid="password" />
```

HTML内のコンポーネントID
とコンポーネントをバインド

ページ仕様(Login.page)

```
<page-specification class="LoginPage">
<component id="name" type="TextField">
  <binding name="value" expression="name" />
 </component>
 <component id="password" type="TextField">
  <binding name="value" expression="password" />
 </component>
```

valueパラメータにname
プロパティを指定

ページクラス(LoginPage.java)

```
public class LoginPage extends BasePage {
  private String name;
  private String password;
  public String getName(){
   return name;
  }
  public void setName(String name){
   this.name = name;
  }
 ...
```

Tapestryの仕組み(2)

HTMLテンプレートの中にコンポーネントを埋め込み

- JSFライクにHTMLにパラメータを埋め込むことも可能
- プラグラマはページ仕様を作成する必要がないので楽
- デザインとロジック(パラメータ)が分離できない

ユーザID	ognl:userid
パスワード	●●●●●●●●●●●●

HTMLテンプレート(Login.html)

```
ユーザID:   <input type="text" jwcid="name@TextField" value="ognl:name" /><br />
パスワード: <input type="password" jwcid="password@TextField" value="ognl:password" />
```

ページクラス(LoginPage.java)

```
public class LoginPage extends BasePage {
  private String name;
  private String password;
  public String getName(){
   return name;
  }
  public void setName(String name){
    this.name = name;
  }
  …
```

ページ仕様(Login.page)

HTMLにパラメータを記述したため、パラメータは不要

ダミーのデザイン($remove$)

- ダミーのデザインを定義することにより、ブラウザ/オーサリングツールによるプレビュー時にレンダリングイメージを示すことが可能。

```
<tr jwcid="list" element="tr">
  <td><span jwcid="name">山田太郎</span></td>
  <td><span jwcid="age">19</span></td>
  <td><span jwcid="address">千葉県市川市…</span></td>
</tr>

<tr jwcid="$remove">
  <td>山田次郎</td>
  <td>17</td>
  <td>千葉県市川市…</td>
</tr>

<tr jwcid="$remove">
  <td>山田三郎</td>
  <td>16</td>
  <td>千葉県市川市…</td>
</tr>
```

デザインはブラウザで確認可能

名前	年齢	住所
山田太郎	19	千葉県市川市…
山田次郎	17	千葉県市川市…
山田三郎	16	千葉県市川市…

サーバサイトのページとして評価されない、ダミーのデザインを記述

リストの表示(Foreach)

UserList.html

```
<tr jwcid="list" element="tr">
  <td><span jwcid="name" >山田太郎</span></td>
  <td><span jwcid="age" >19</span></td>
  <td><span jwcid="address" >千葉県市川市…</span></td>
</tr>
```

UserList.page

```
<component id="list" type="Foreach" >
    <binding name="source" expression="getUsers()"/>
</component>
<component id="name" type="Insert">
    <binding name="value" expression="components.list.value.name"/>
</component>
<component id="age" type="Insert">
    <binding name="value" expression="components.list.value.age"/>
</component>
<component id="address" type="Insert">
    <binding name="value"
             expression="components.list.value.address"/>
</component>
```

UserListPage.java

```
public class UserListPage {
  public ArrayList<User> getUsers(){
    return userlist;
  }
}
```

User.java

```
public class User {
  public String getName(){…}
  public String getAge(){…}
  public String getAddress(){…}
}
```

イベントリスナ (ロジックの実行/画面遷移)

フォームがサブミットされたときに呼ばれる
リスナを利用してロジックを実行

Login.page

```
<form jwcid="@Form" listener="ognl:listeners.submit" />
  <input type="text" jwcid="@TextField" value="ognl:name" />
 <input type="text" jwcid="@TextField" value="ognl:password" />
```

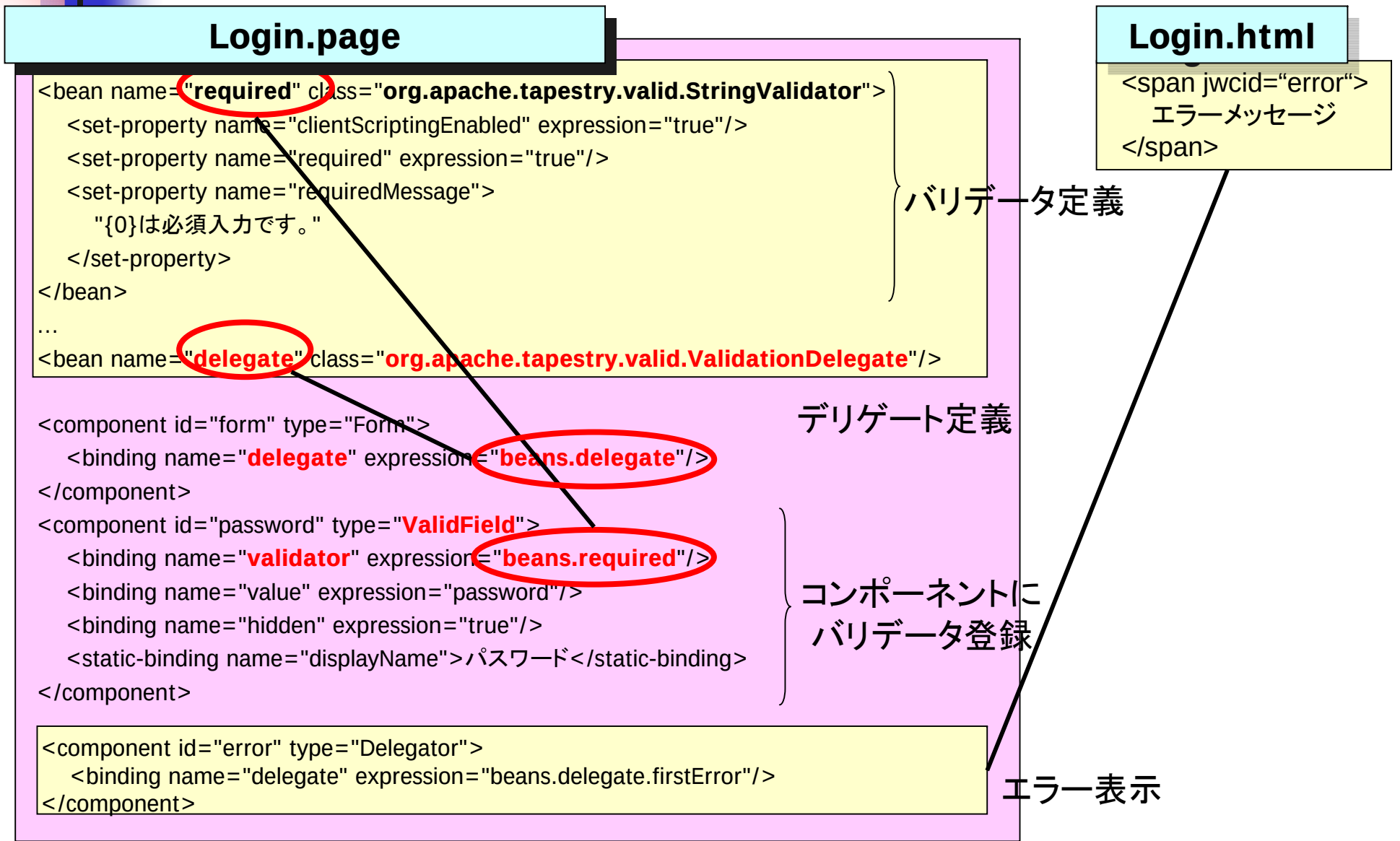
LoginPage.java

```
public void submit(IRequestCycle req){
        checkLogin(user,password);
        if(loginSuccess){
                req.activate("Booklist");
        } else {
                delegate.record(
                        new ValidatorException("ユーザ名/パスワードが正しくありません。"));
        }
}
```

リスナを定義フォームが送信されたときにリスナが呼ばれる。

遷移先のページを設定

バリデーション(コンポーネントを利用)



バリデーション(ロジック内で実行)

```
public void login(IRequestCycle req){
    ValidationDelegate delegate = (ValidationDelegate)getBeans().getBean("delegate");
    if(delegate.getHasErrors()){
        return;
    }

    if(!confirmPassword(password)){
        delegate.record(
            new ValidatorException("ユーザ名/パスワードが正しくありません。")
        );
        return;
    }
```

エラーが発生済みなら、何もしない

ValidationDelegate取得

パスワードが不正なら、エラーメッセージを表示

セッション変数(Visit)

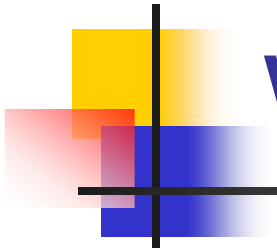
- TapestryはServletAPIをラッピング
- HttpSessionの代わりにVisitと呼ばれるセッション変数を設定

Visit.java

```
public class Visit implements Serializable {
    private ArrayList<Item> cart = new ArrayList<Item>();
    private long loginTime = 0;
    public void addItem(Item item){
                cart.add(item);
    }
}
```

xxx.application

```
<application name="bookstore">
    <property name="org.apache.tapestry.visit-class">app.Visit</property>
</application>
```



web.xmlの設定

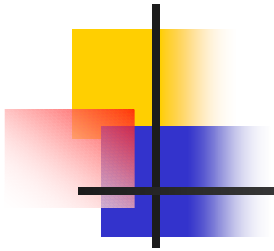
- TapestryのApplicationServlet をWebアプリケーションのルートとして設定

```
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
    <init-param>
        <param-name>TreeRootDir</param-name>
        <param-value>/</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
</servlet>
```

Tapestry/Struts/JSFの比較

	Tapestry	JSF	Struts
バリデーション	ページ仕様に記述(ビューとバリデーションを分離可能)	Taglib(バリデーションをビューに埋め込み)	バリデーション用ファイルに記述(ビューとバリデーションを分離可能)
画面遷移	ページへ直接ジャンプ	ナビゲーション	ナビゲーション
コンポーネント記述	jwcid属性に記述 (HTMLタグを利用可能)	Taglib(独自タグ)	Taglib(独自タグ)
アクション	ページクラス(リスナ)	Managed Bean (リスナ)	Actionクラス
フォームの値	ページクラス	Managed Bean	ActionForm
セッション変数	Visitクラス	Managed Bean (session)	ActionForm(sessionスコープ) HttpSession
カスタム コンポーネント	Javaクラス+jwcファイル	JSP Taglib	JSP Taglib

Tapestryの開発を支援するツール/フレームワーク



- Spindle
- S2Tapestry

Spindle

・Spindleを利用するとビューの開発は簡単になるが、XMLの設定ファイル等の記述は必要。
・XHTML形式でビューを開発する必要があるため、タグの閉じ忘れなどがあるとエラーになる。

Spindle

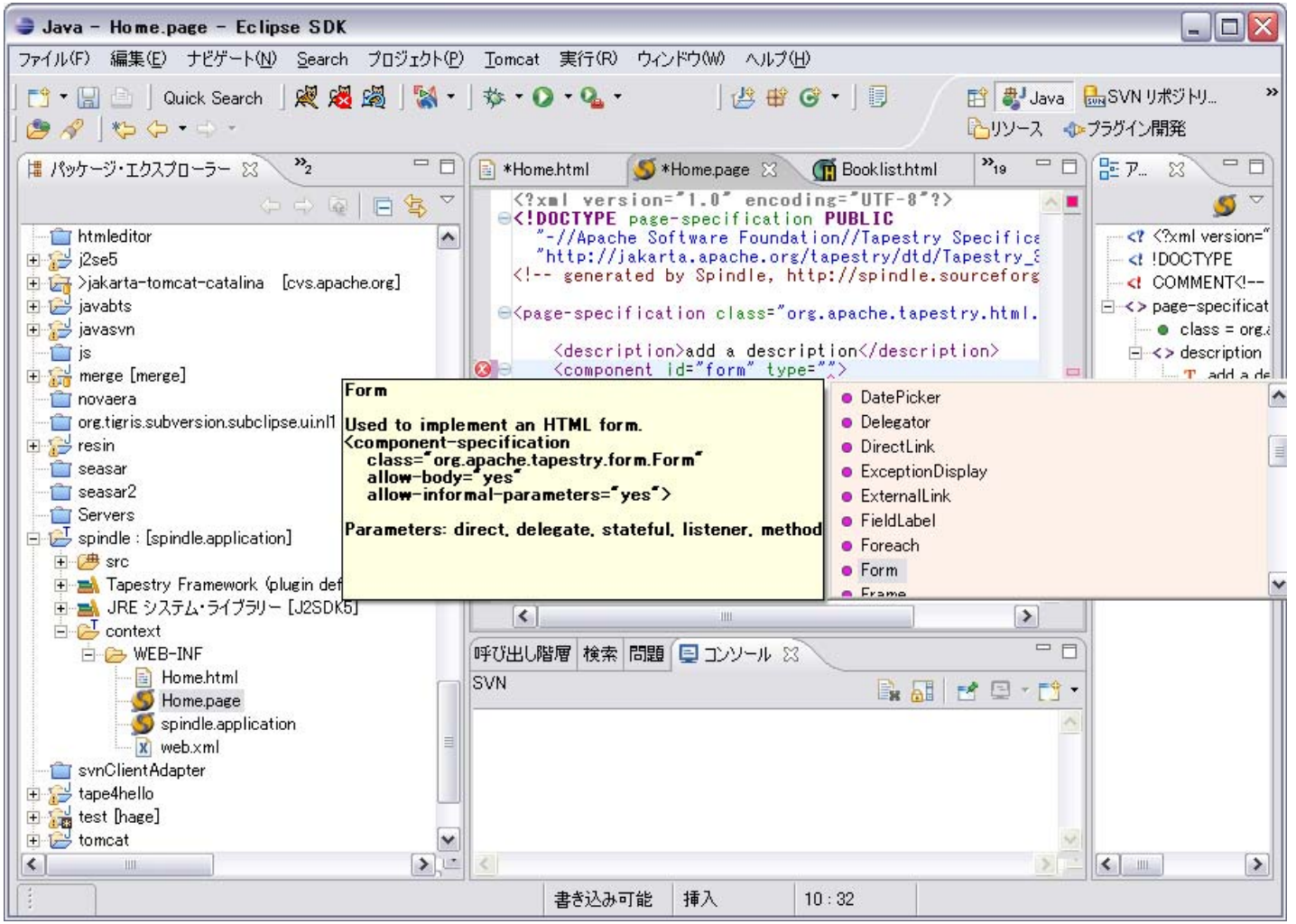
Eclipse上でTapestryアプリケーションの開発をサポートするプラグイン

各種ウィザード
・プロジェクト、ページ、コンポーネントを作成するウィザード

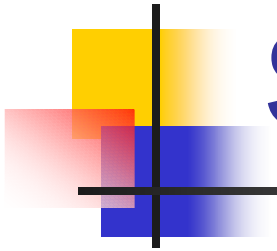
テンプレートHTMLエディタ
・jwcidの入力補完機能
・バリデーション機能

設定ファイル用XMLエディタ
・ページファイル、jwcファイル、アプリケーションファイルの編集
・入力の補完、バリデーション

Spindleの画面



ページ仕様を編集している様子



S2Tapestry

特徴

- TapestryのページクラスがSeasar2で管理
- ページクラスで、DI/AOP/トランザクション制御等、Seasar2の機能が利用可能に
- サービスクラスをDIでインジェクト
- Tapestry3.0.xに対応

注意

- Concreteなクラスである必要あり。
  (Dynamicプロキシは利用できない)

S2Tapestryの利用 (S2TapestryServetの設定)

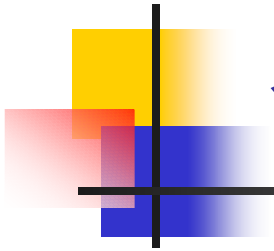
- web.xml
  - TapestryのApplicationServletの代わりにS2TapestryServletを利用

```
<servlet>
    <servlet-name>bookstore</servlet-name>
    <servlet-class>org.seasar.tapestry.S2TapestryServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>bookstore</servlet-name>
    <url-pattern>/app</url-pattern>
</servlet-mapping>
```

S2Tapestryの利用 (コンポーネントの登録)

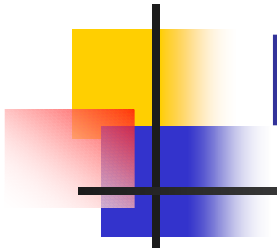
- Diconファイル
  - S2Tapestry付属のs2tapestry.diconを用意し、アプリケーションのdiconファイルでinclude
  - Tapestryのページクラスをコンポーネントとして登録

```
…
<include path="s2tapestry.dicon"/>
…
<component name=“ListPage" class=“examle.ListPage" autoBinding="none" instance="prototype" />
<component name=“LoginPage" class=“examle.LoginPage" autoBinding="none" instance="prototype" />
…
```



Hibernateによる インテグレーション層のEoD

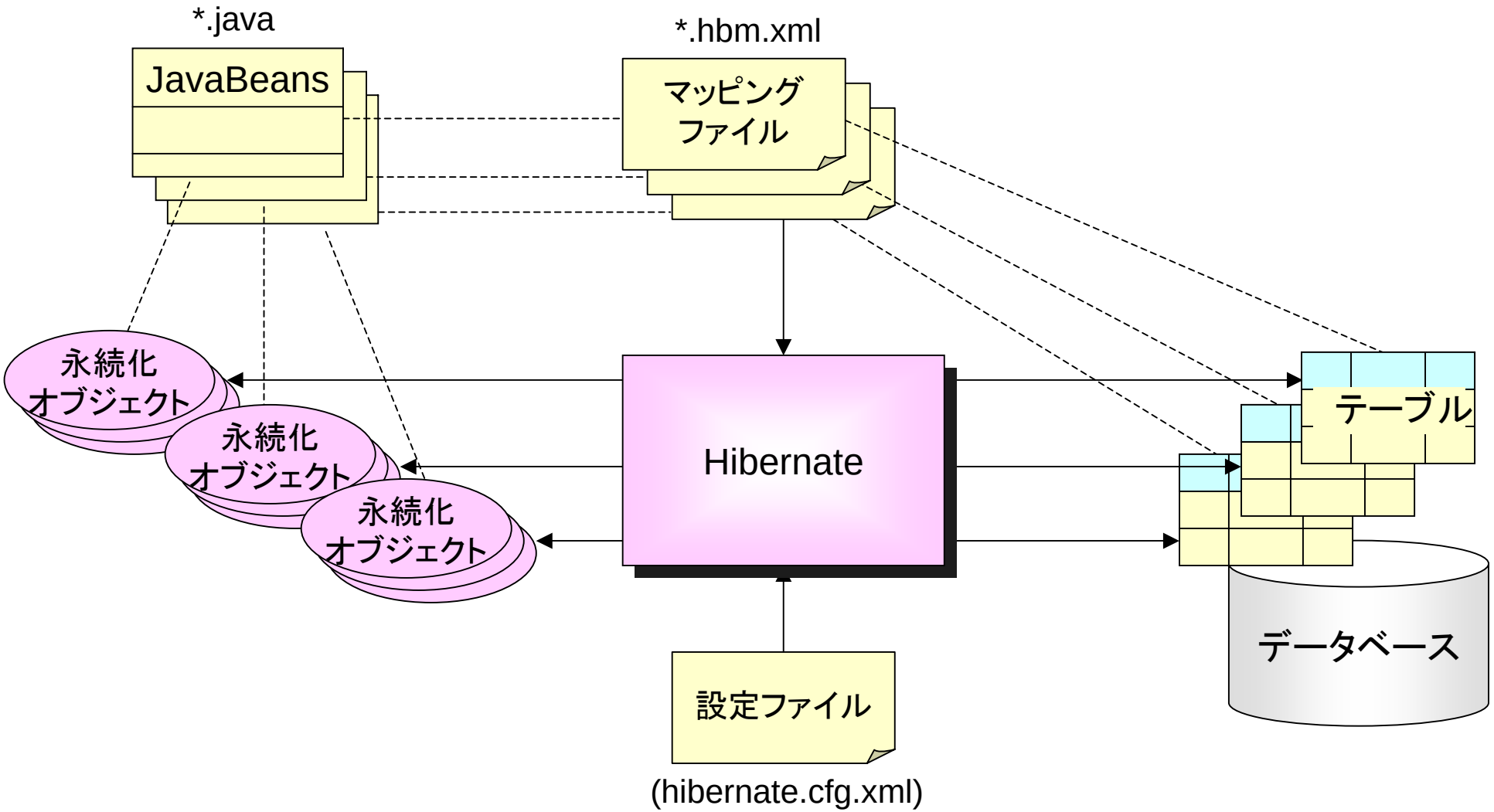
- MiddlegenとS2Hibernateを用いたインテグレーション層のEoD



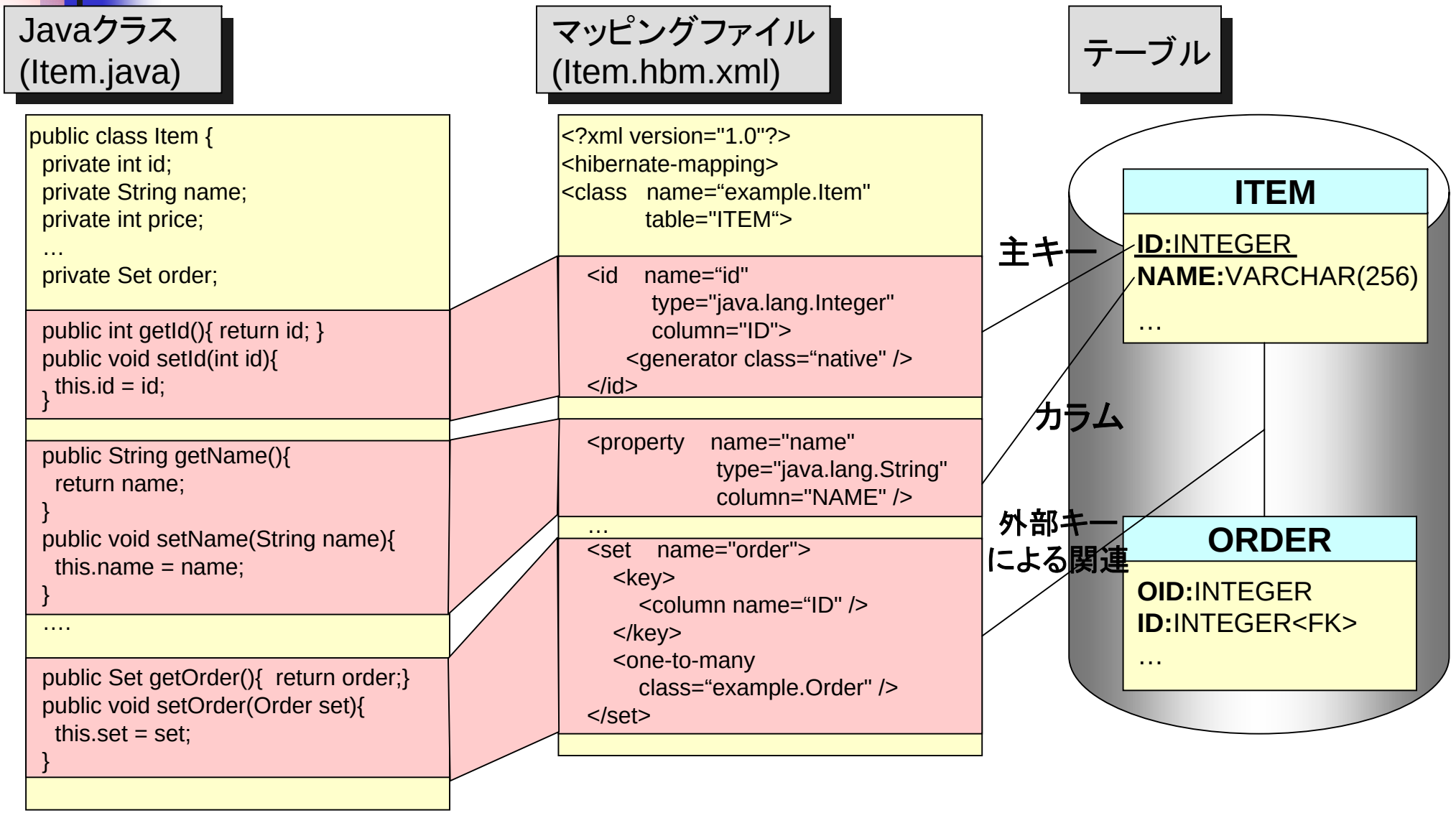
Hibernateとは?

- XMLのマッピングファイルを利用して JavaBeansとRDBMSをマッピング
- EJB3のアイデアのもと
  - Hibernateの開発者であるGavin KingはEJB仕様の策定に参加

Hibernateの概要(1)



Hibernateの概要(2)



Hibernateの概要(3)

- Hibernateを利用したコーディング

```
Configuration config = new Configuration();
config = config.configure();


SessionFactory sessionfactory = config.buildSessionFactory();

Session session = sessionfactory.openSession();
Item item = session.load(Item.class, new Integer(1);
```

他に下記のメソッドを利用

- Session#save();
- Session#update();
- Session#delete();
- Session#find();

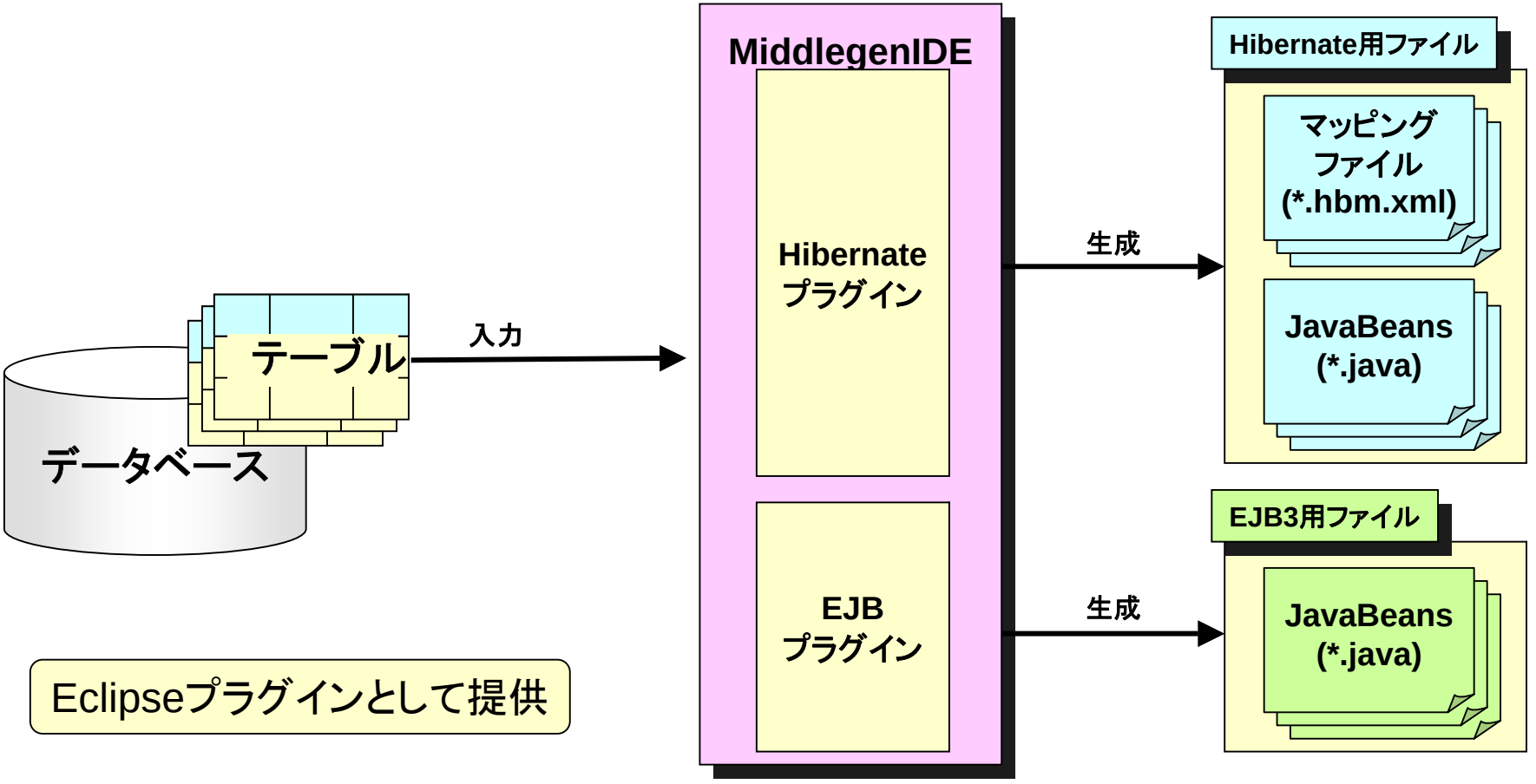
S2HibernateとMiddlegenIDEによるHibernateを利用したEoD

- Hibernateの欠点
  - マッピングファイルの作成
    - マッピングファイルの学習と作成コストが必要
  - Hibernate専用APIを利用
    - APIの学習コストがかかる

・データモデルからマッピングファイルとJavaBeansを自動生成
・S2HibernateによるDAOの実装の省略とHibernateAPIの隠蔽化

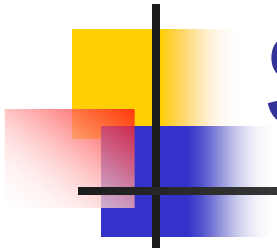
MiddlegenIDE(1)

- データベースに定義されたテーブルから、Hibernate/EJB3に対応したJavaBeans/マッピングファイルを生成



MiddlegenIDE(2)

- 開発の流れ
  - ER図作成(ERツール)
  - DDLのエキスポート(ERツール)
  - テーブル作成(ERツール/SQL)
  - 永続化オブジェクト生成(MiddlegenIDE)
- Hibernateの設定はGUIで可能
- ただし、細かい設定(リレーション等)については、生成されたファイルを修正する必要があり。



S2Hibernate

- Seasarと組み合わせて利用
- DAOの実装を簡略化
  - 決められたメソッドのインタフェースを追加するだけでCRUDに対する基本操作を提供
  - 定数アノテーションによるコンフィギュレーション
  - HQLを定数アノテーションで記述
- POJO/POJI
  - Hibernate固有のAPIを利用することなくDAOを実装可能
- Open Session in Viewに対応

S2HibernateのDAOの実装

命名規則に従ってインタフェースを定義するだけでDAOの実装を取得可能

ItemDAO.java

```
public interface ItemDAO {
    public Class BEAN= Item.class;

    public Item load(Integer id);
    public List getAllItems();
    public void save(Item item);
    public void delete(Item item);
    public void update(Item item)

    public String getItemCount_HQL = "select count(*) from Item";
    public int getItemCount();
}
```

エンティティクラスを指定

定数アノテーション

CRUDの基本操作

HQLの実行と結果取得

HQLを指定(HQLの結果を取得するメソッド名＋_HQL)

S2HibernateのDAOの利用

TestServiceImpl.java

```
ItemDAO itemDAO;
…
public void setItemDAO(ItemDAO itemDAO){
   this.itemDAO = itemDAO;
}
…
public void action(){
  …
   Item item = itemDAO.load(new Integer(id));
}
```

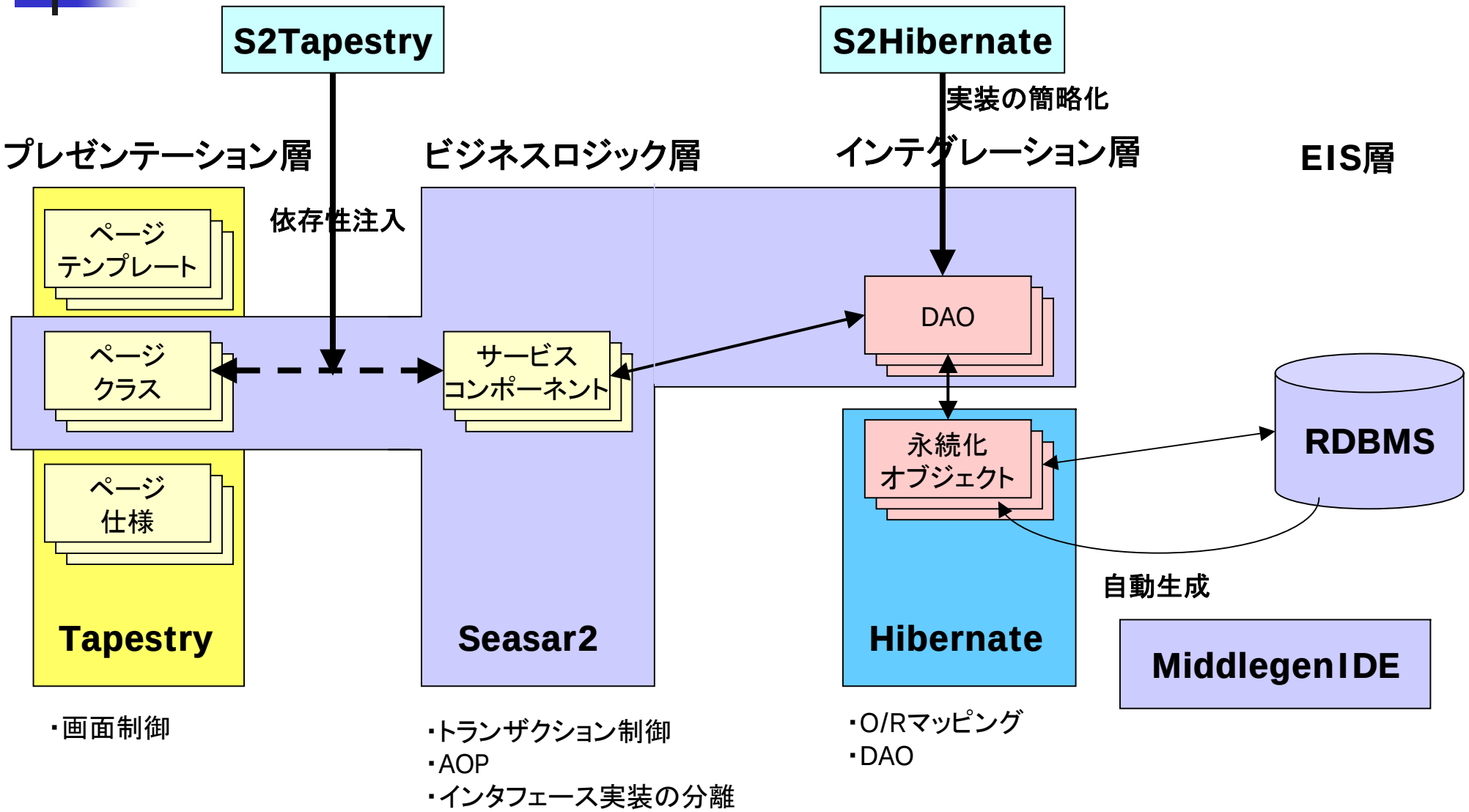
app.dicon

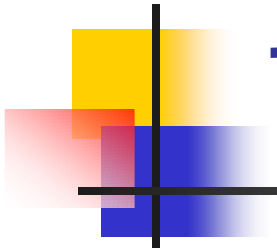
```
<include path="s2Hibernate3.dicon"/>
  <include path="aop.dicon"/>

  <component name="itemDAO" class="example.ItemDAO">
     <aspect>aop.traceInterceptor</aspect>
     <aspect>s2hibernate.interceptor</aspect>
  </component>
   <component name="bookstoreService" class="exmaple.TestServiceImpl">
     <property name="itemDAO">itemDAO</property>
  </component>
```

ItemDAOクラスのインジェクト

aopとs2hibernateのアスペクトを指定

Tapesty/Hibernate/Seasarをベースとした EoDの全体像





まとめ

- J2EE技術を巡る進化において、TapestryとHibernateは重要なポジションを占めている
- TapestryによるビューのEoDを示した
- HibernateとS2Hibernate/MiddlegenIDEによるインテグレーション層のEoDを示した

本公演の内容の概要は、技術評論社 Java Press Vol.44にて紹介されています。合わせてご覧ください。また、Tapestry+Seasar+Hibernateを利用したブックストアのサンプルが下記のURLからダウンロード可能です。ご利用下さい。

http://www.gihyo.co.jp/magazines/javapress/archive/Vol44