Seasar Conference
2007 Spring

現場で役立つ実践Teeda

2007.5.27

エスエムジー株式会社
テクニカルコンサルタント
鈴木 貴典

はじめまして

- 名前：鈴木 貴典
- Blog：http://d.hatena.ne.jp/szk-takanori/
- 所属：エスエムジー株式会社（http://www.smg.co.jp）
- コミッタ：S2Axis／S2Axis2
- メール ：takanorig@gmail.com
- 主な仕事：
  - フレームワーク開発
  - Web＋DBアプリ開発
  - プロセス改善（SEPG）

- このセッションのゴール

Teedaをより便利に使いこなせるようになってもらいたい！

- このセッションの位置づけ
  - Teedaプロジェクトリーダの大谷さんによる「今から役立つTeeda入門」に続くものです

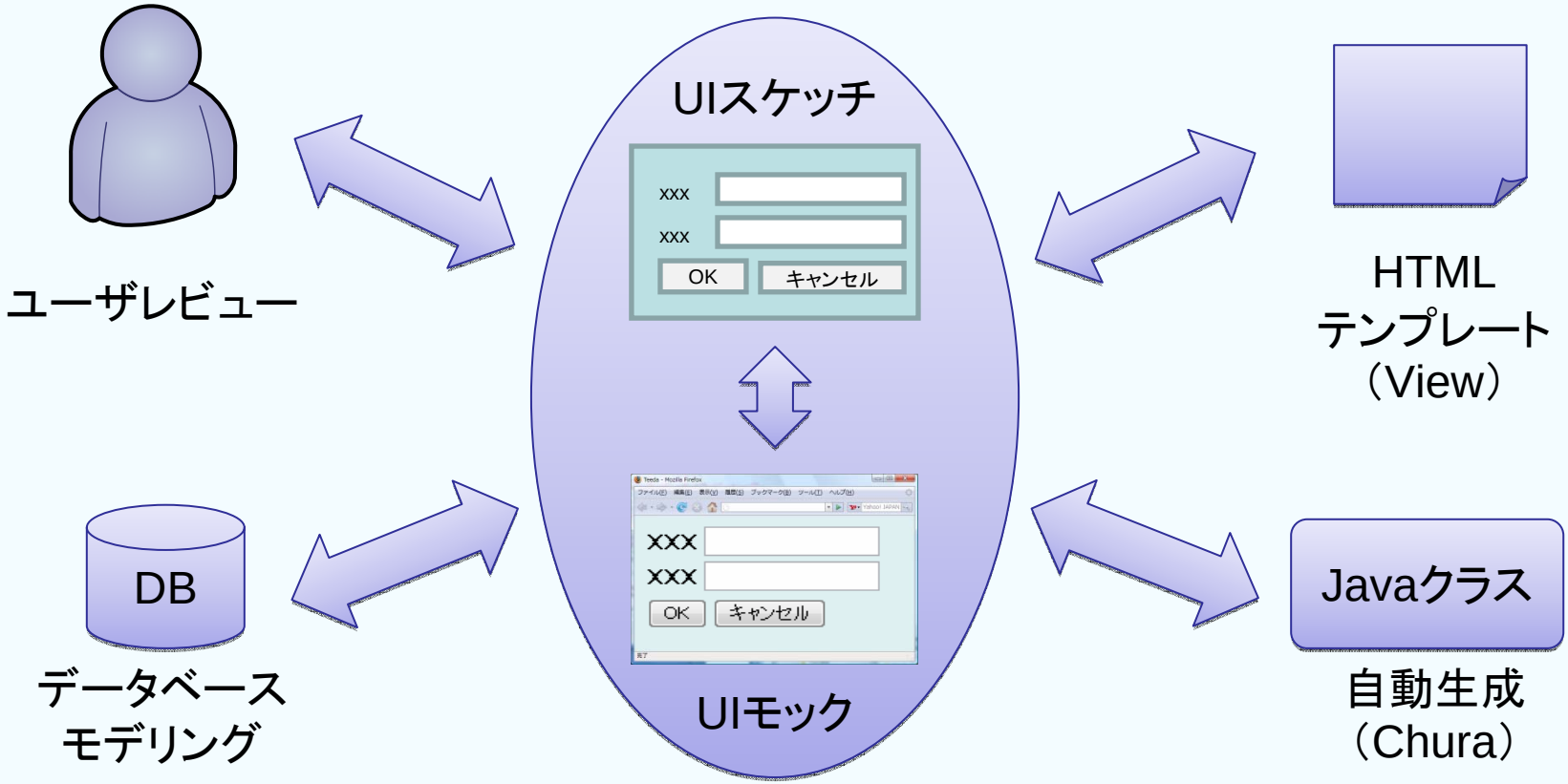
Agenda

1. Teedaの特徴
2. アプリケーション・アーキテクチャを決める
3. HTMLモックアップ作成のコツ
4. スコープ管理を使いこなす
5. かゆいところに手が届く便利機能
6. Teedaにおけるテスト実践

Teedaの特徴

ページ開発駆動

- UI(＝ページ)を中心に開発するスタイル
  - ユーザにとって、最も分かりやすいものを中心に据える
  - Teedaは、この開発スタイルによって最適化されている



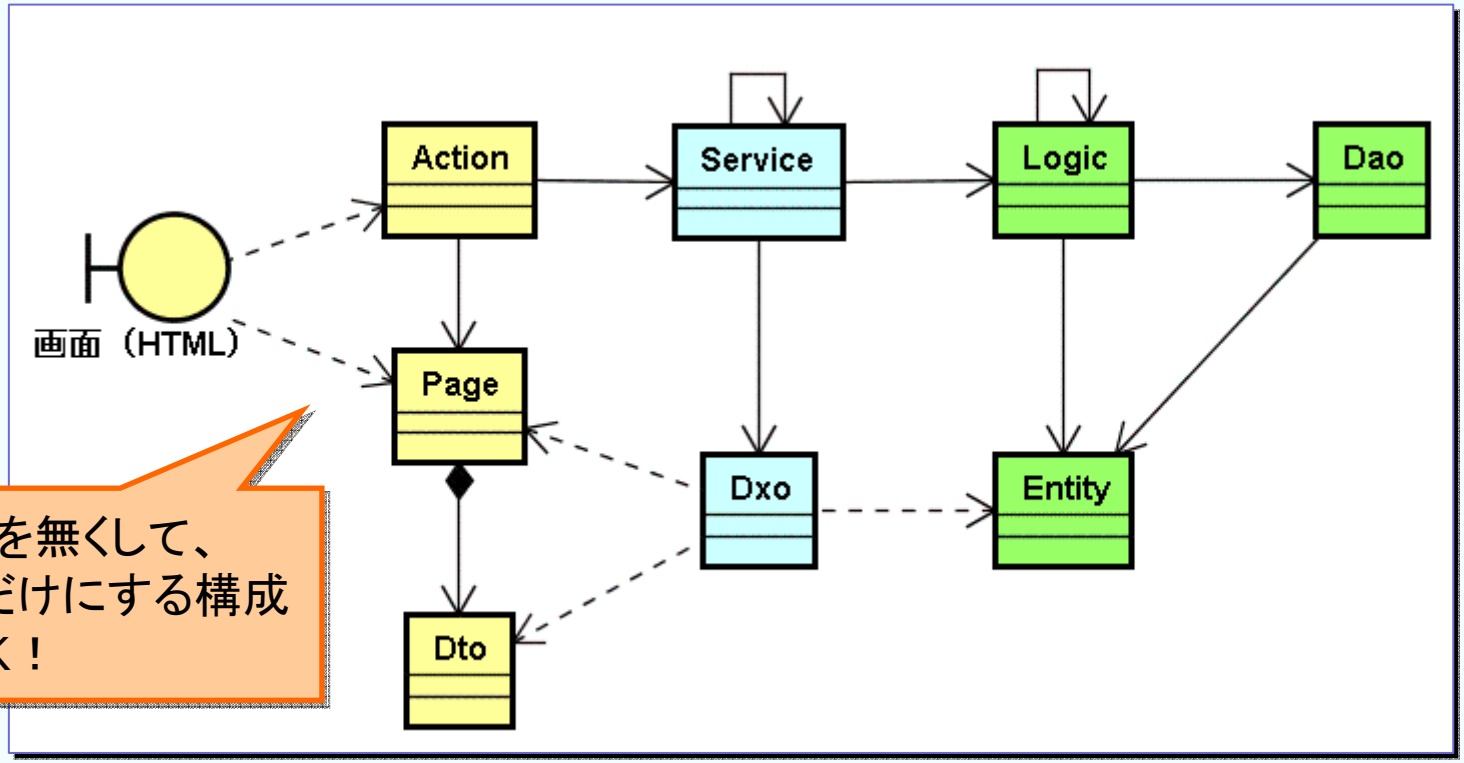
アプリケーション・アーキテクチャを決める

アプリケーション・アーキテクチャを決める

- レイヤーとコンポーネントの構成を考える
  - Goyaベース
  - DxoやDoltengとの親和性の向上を目指す
- Teedaでは、画面と1:1になるようにPageを作成する
  - employeeList.html → EmployeeListPage.java
- ただし、Page（画面の項目と関連）とAction（画面のボタンと関連）を分割することも可能
  - employeeList.html → EmployeeListPage.java
    → EmployeeListAction.java

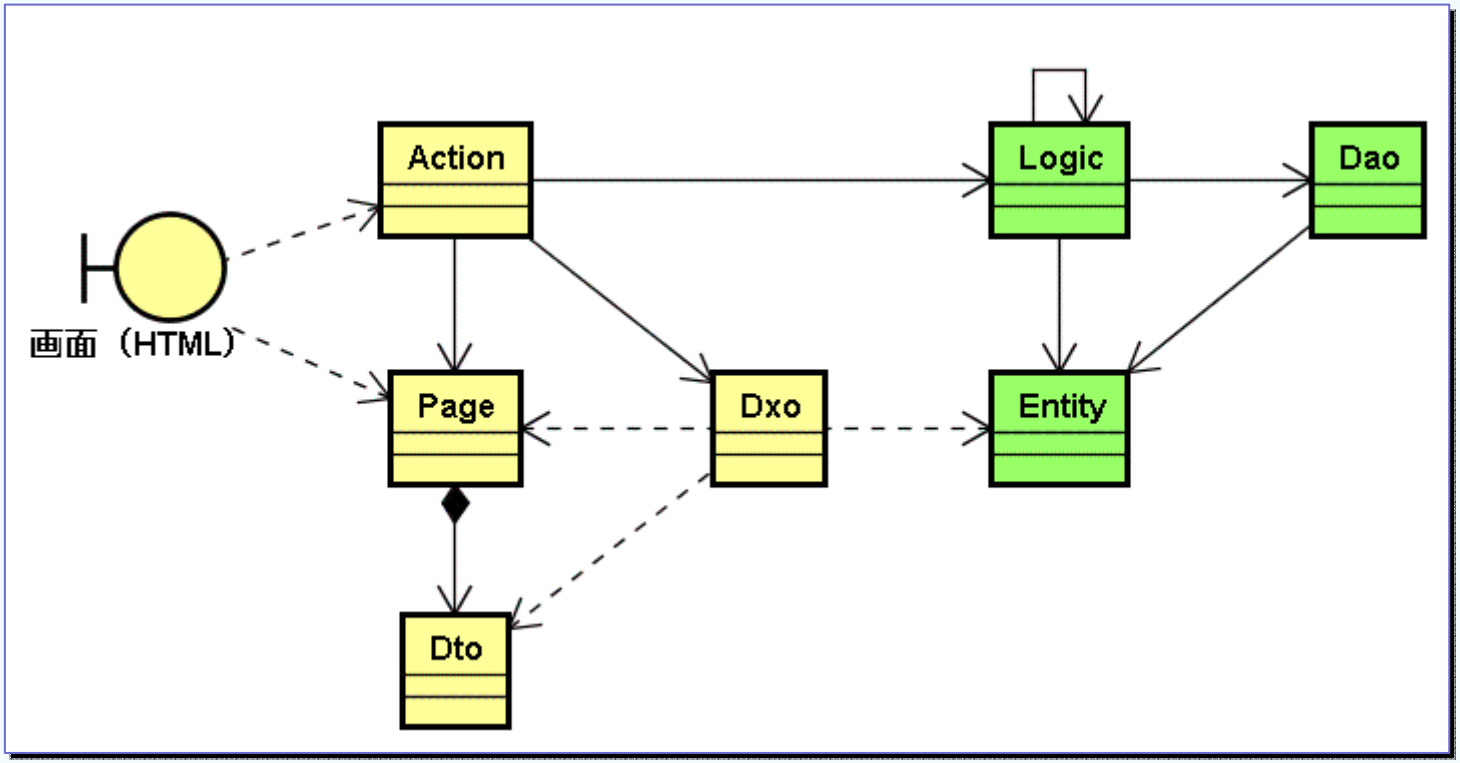
Full Pattern

- 大規模アプリケーション向け
- 画面とロジックを分担して共同開発したり、フロー制御や他システム連携が多かったりするシステムに向く



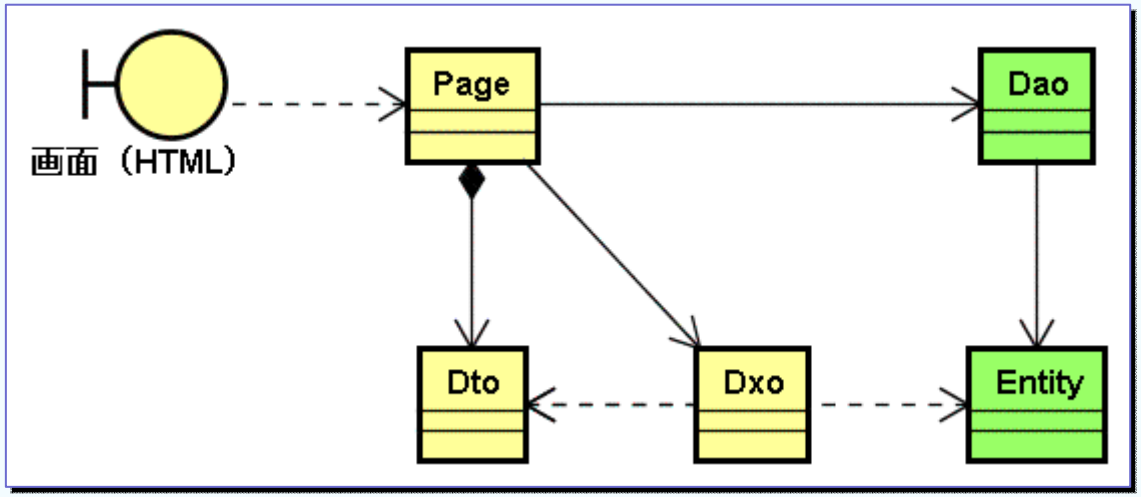
Middle Pattern

- 中規模アプリケーション向け
- 画面ロジックとドメインロジックを2つのレイヤーに集約させたパターンであり、大抵のシステムに対応できる



Lightweight Pattern

- 小規模アプリケーション向け
- 画面とDBテーブルが1対1で、ロジックはほぼない、というシステムに向く



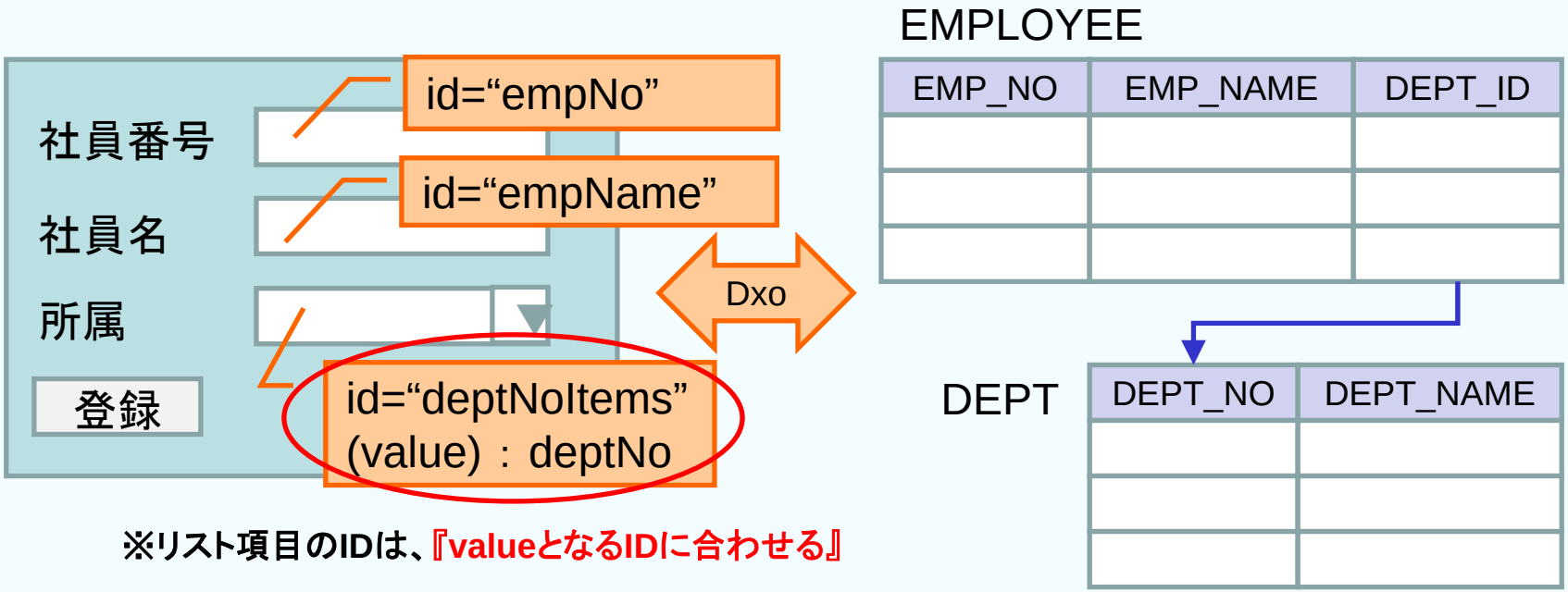
HTMLモックアップ作成のコツ

IDの決め方

- 全フィールドに対して、ユニークになるように付ける
  - ×：name　○：userName
  - HTMLのIDと、DBのフィールドを一致させる

（DBのカラム名もユニークにする（※ID／VersionNo以外））

Dxoでの変換時に、変換ルールを指定しなくても、自動的にマッピングされる。



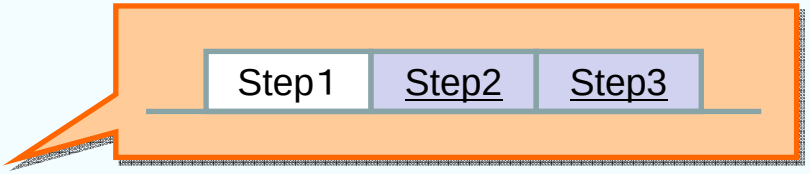
※リスト項目のIDは、『valueとなるIDに合わせる』

ボタンとリンクの使い分け

- 『ボタン』と『リンク』では、値の引き継がれ方が異なる
  - ボタン（doXxx, goXxx, jumpXxx）
    - 全プロパティが次画面に引き継がれる
  - リンク（goXxx）
    - クエリストリングで指定したプロパティのみ、次画面に引き継がれる

画面設計の段階で、どちらを利用するのかを考慮！

- でも、リンクで遷移して、かつ、値も引き継ぎたい場合もある・・・
  - 例
    - タブ（に見せかけたリンク）を利用した自画面遷移
  - 回避策
    - 非表示のボタンを用意しておき、JavaScriptを利用して、リンクが押下された際にボタンが押下されるようにする
    - <a id="xxx" href="#?id=1" onclick="navigate(this)">



モックタグの活用

- モックだけで表示させ、動作時には表示されないタグを記述することが可能
  - （※ idを『mock』から始める）
- こんなときに便利
  - HTML自体に仕様やメモを書いておく

```
<div id="mockComment">
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
</div>
```

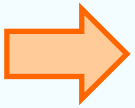
  - JavaScriptをモックのときだけ動作させたい

```
<script id="mockScript" type="text/javascript">
<!—
    alert("Hello"); //
-->
</script>
```

スコープ管理を使いこなす

HTTP Request／Sessionの限界

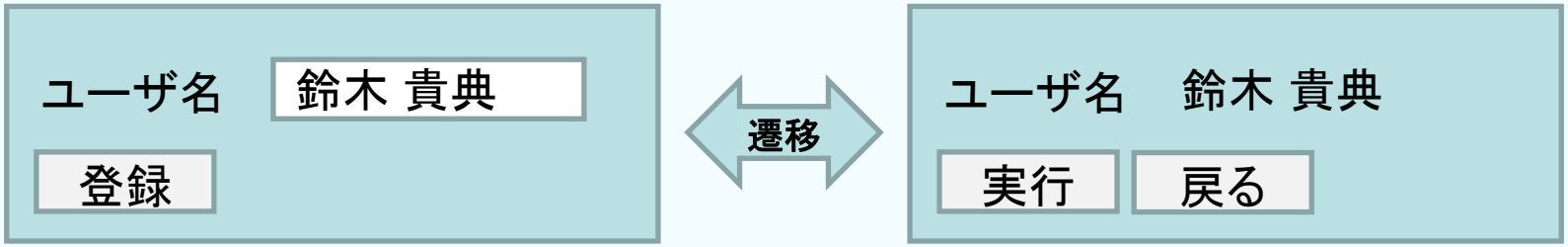
- HTTP Request／Sessionの限界
  - Request
    - ウィザードなどの複数リクエストにまたがる処理で利用するには、ライフサイクルが短すぎる
      - 帯に短し
    - 値を引き継ぐために、hiddenを多用する必要がある
      - パラメータ改ざんの危険性も出てくる
  - Session
    - ウィザードなどの複数リクエストにまたがる処理で利用するには、ライフサイクルが長すぎる
      - タスキに長し
    - Sessionへの格納（setAttribute）、および、Sessionからの削除（removeAttribute）を、開発者が管理する必要がある
      - バグが増える
    - マルチウィンドウ



Request／Sessionだけでは不便！

Teedaにおけるスコープ管理(1/4)

- Teedaに画面遷移時の値引き継ぎ原則



/view/employee/Input.html

```
<input type="text" id="userName" />
```

/web/employee/InputPage.java

```
public void InputPage {
    private String userName;
}
```

/view/employee/confirm.html

```
<span id="userName" />
```

/web/employee/ConfirmPage.java

```
public void ConfirmPage {
    private String userName;
}
```

遷移の前後で、同じプロパティがあれば、
次の画面で、値がインジェクションされる
(※同一サブアプリケーション内でのボタンでの遷移の場合)

Teedaにおけるスコープ管理(2/4)

- Teeda独自のスコープ

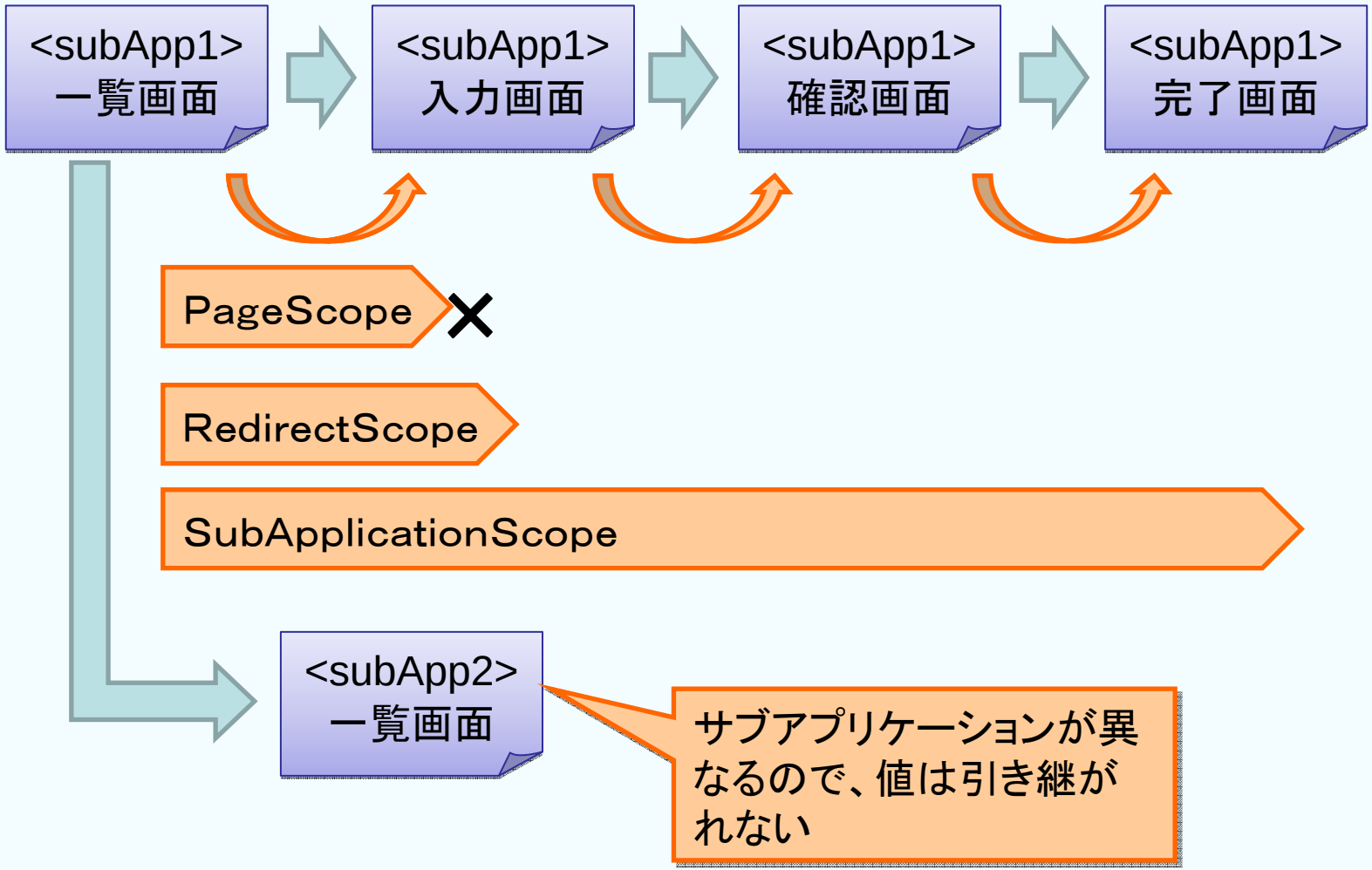
スコープ名	ライフサイクル
SubApplicationScope（デフォルト）	同一のサブアプリケーション内ならば、プロパティを引き継ぐ。異なるサブアプリケーションに遷移した場合は、プロパティ値は削除される。
RedirectScope	次の画面にリダイレクト後、プロパティ値は削除される。つまり、次の画面までしか値を引き継がない。
PageScope	次の画面に値を引き継がない。

```
public void EmployeePage {

    @PageScope
    private String userName;

}
```

プロパティに
アノテーションを指定する

Teedaにおけるスコープ管理(3/4)

- Teeda独自のスコープ



Teedaにおけるスコープ管理(4/4)

- Session／Requestも使えます
  - Pageクラスにインジェクションする

```
public void EmployeePage {
    private HtttpSession session;

    public void setSession(HttpSession session) {
        this.session = session;
    }
}
```

  - Dtoのクラスなどに、アノテーションを指定する

```
@Component(instance = InstanceType.SESSION)
public class UserDto implements Serializable {
    ....
}
```

TakeOver

- TakeOverアノテーションを利用して、引き継ぐプロパティを個別に指定することもできる

タイプ	説明	アノテーション
NEVER	次画面に何も引き継がない	@TakeOver(type=TakeOverType.NEVER)
INCLUDE	次画面に「name」「value」のみを引き継ぐ	@TakeOver(type=TakeOverType.INCLUDE, properties="name, value")
EXCLUDE	次画面に「name」「value」以外のプロパティを引き継ぐ	@TakeOver(type=TakeOverType.EXCLUDE, properties="name, value")

メソッドにアノテーションを指定する

```
@TakeOver(type = TakeOverType.NEVER)
public String doExecute() {
    return
}
```

かゆいところに手が届く便利機能

こんな機能があります

- Teedaでは、以下の機能が便利
  - 拡張されたバリデータ
    - ボタン指定
    - エラー発生時のスタイルシート自動設定
  - 簡単なメッセージ表示
    - 個別プロパティ／画面全体に対するメッセージ表示
    - Utilの活用
  - ラベルの引き継ぎ
    - プルダウンやラジオボタンで、valueだけでなく、labelの引き継ぎができる
  - List<Map>の利用
    - 繰り返しや一覧の項目で、わざわざDtoを作成しなくても済む

拡張されたバリデータ

- JSFのバリデータを超えています
  - ボタン指定
    - 1画面内に複数のボタンがある場合にバリデーションが動作するボタンを指定できる

```
public void EmployeePage {
    @Required(target="doExecute")
    private String empName;
}
```

target属性に、ボタンのidを指定する
指定したボタンが押下されたときだけ、バリデーションが実行される

  - エラー発生時のスタイルシート自動設定
    - 「onTeedaError」というスタイルが出力される

```
<input type="text"
    id="empName"
    name="form:empName"
    value=""
    class="onTeedaError" />
```

バリデーションエラーとなった項目に、色を付ける等、分かりやすい表示が可能になる

簡単なメッセージ表示

- これさえ知っていれば問題なし
  - HTMLの指定

id	説明	HTMLでの指定
allMessages	全てのメッセージ	
messages	項目にヒモ付かないメッセージ	
id+Message	指定されたidにヒモ付くメッセージ	

  - レベルに合わせたスタイルの切り替え

```
<span id="allMessages"
    te:fatalClass=“fatalMessage“  te:errorClass="errorMessage“
    te:warnClass="warnMessage”  te:infoClass="infoMessage"
</span>
```

  - サーバ側の指定

```
FacesMessageUtil.addErrorMessage(String messageId)
FacesMessageUtil.addErrorMessage(String messageId, Object[] args)
```

メッセージ表示は、これだけで可能になる

ラベルの引き継ぎ

- N:1の関連でよく使います
  - プルダウンやラジオボタンなどで、値（value）だけでなく、ラベル（label）を引き継ぐことができる
  - このようなパターンの画面で効いてくる
    - 入力画面ではプルダウンで選択
    - 確認画面では選択されたlabelを表示
    - DBには、valueで登録

```
<input type="hidden" id="fooItemsSave" />

<select id="fooItems">
    <option value="0">ラベル0</option>
</select>
```

※注意
ItemsSave／ItemsSessionSaveを利用する必要がある

```
public void SelectPage {
    private List    fooItems;
    private Integer foo;
    private String  fooLabel;
}
```

xxxLabelというプロパティがあれば、自動的にラベル値がインジェクションされる

List<Map>の利用

- 実装量が減らせます
  - 繰り返しや一覧（Grid／ForEach／SelectOneRadioなど）のプロパティ（xxxItems）では、Dtoのリスト／配列だけでなく、Mapのリストも利用できる
  - 更新処理がなければ、Mapのリストで十分

```
<select id="fooItems">
    <option value="0">選択</option>
</select>
```

```
public void SelectPage {
    private List<Map<String, String>> fooItems;
}
```

  - Dxoとの連携で、簡単に実現可能

```
@ConversionRule("value : id, label : empName")
List<Map<String, String>> convert(Entity[] entities);
```

ConversionRuleで、プロパティの関連付けを行う

Teedaにおけるテスト実践

テストに関する機能

- Teedaでは、テストのことも考えられています
  - RequestDumpFilter
    - HTTPリクエストの情報をログに出力するServletFilter
  - TeedaTestCase
    - JUnitの拡張テストケース
    - HttpServletResponseやFacesContextなどのモックが、標準で用意されている
  - TeedaWebTester
    - TomcatなどのAPサーバを起動した状態でテストする

RequestDumpFilter

- RequestDumpFilter
  - デバッグやテスト実施時のエビデンスとして役に立つ

```
<filter>
   <filter-name>requestDumpFilter</filter-name>
   <filter-class>org.seasar.teeda.core.filter.RequestDumpFilter</filter-class>
   <init-param>
      <param-name>beforeRequestParameter</param-name>
      <param-value>true</param-value>
   </init-param>
    ....
</filter>
```

```
** before *************: /view/add/add.html
  Request class=org.seasar.extension.httpsession.
  RequestedSessionId=c0a8811c1aaf64da9e22e349fb91
  REQUEST_URI=/teeda-html-example/view/add/add.html,
  SERVLET_PATH=/view/add/add.html
       ....
  [param]_id24:addForm/view/add/add.html=_id24:addForm
  [param]_id24:addForm:arg1=1
  [param]_id24:addForm:arg2=2
  [param]_id24:addForm:doCalculate=calculate
```

入力値や押下したボタンのIDなどが分かる

TeedaTestCase

- TeedaTestCase
  - HttpServletResponse／FacesContextなどのモックが利用できるので、ServletやJSFのAPIが絡むテストも可能

```
public class DownloadActionTest extends TeedaTestCase {

    public void testResponseComplete() throws Exception {

        DownloadAction action = new DownloadAction();
        action.setResponse(getResponse());

        assertEquals(false, getFacesContext().getResponseComplete());
        action.download();

        assertEquals(true, getFacesContext().getResponseComplete());
    }

}
```

TeedaWebTester

- TeedaWebTester
  - 画面からの操作を自動化するイメージで使う
  - 実行する前に、対象のアプリケーションを起動しておく

```
public class AddWebTest extends TestCase {

    public void testAdd() throws Exception {

        final TeedaWebTester tester = new TeedaWebTester();
        tester.beginAt(“http://localhost:8080/teeda-html-example”, "view/add/add.html");

        tester.setTextById("arg1", “5");
        tester.setTextById("arg2", “27");
        tester.submitById("doCalculate");

        tester.assertTextEqualsById(“result”, “32");
    }

}
```

ふりかえり

1. Teedaの特徴
   - ページ駆動開発
2. アプリケーション・アーキテクチャを決める
   - Goyaベースのアーキテクチャ
   - Full Pattern
   - Middle Pattern
   - Lightweight Pattern
3. HTMLモックアップ作成のコツ
   - IDの付け方
   - ボタンとリンクの使い分け
   - モックタグの活用
4. スコープ管理を使いこなす
   - SubApplicationScope
   - RedirectScope
   - PageScope
   - TakeOverアノテーション
5. かゆいところに手が届く便利機能
   - 拡張されたバリデータ
   - 簡単なメッセージ表示
   - ラベルの引き継ぎ
   - List<Map>の利用
6. Teedaにおけるテスト実践
   - RequestDumpFilter
   - TeedaTestCase
   - TeedaWebTester

ご静聴
ありがとうございました