実践的なサンプルアプリを その場でコーディングします！

株式会社 ティーアンドエフカンパニー
出羽 健一

講師紹介

- 名前： 出羽 健一
  - dewa@tafc.co.jp / http://d.hatena.ne.jp/
- 所属： 株式会社 ティーアンドエフカンパニー
  - http://www.tafc.co.jp/
- 所属： 金沢工業大学 大学院 客員准教授
  - 担当： サーバーサイドJava特論

本セッションについて

- 本セッションは以下の執筆記事の
  ライブコーディング版です
- WEB + DB PRESS vol. 41 (技術評論社)
  - 特集2：つらいJavaから楽しいJavaへ
    Seasar2 サクサク開発　実践カリキュラム

サンプルコードのダウンロード

http://www.gihyo.co.jp/magazines/webdbpress/support/Vol41/

このセッションの目的

- 開発現場でアーキテクトが作成するお手本的なサンプルアプリケーションをより良いものに仕上げるためのヒントを提供する

- 活字では伝わりにくいライブならではの開発リズム、イメージを体感してもらう

★ マスターメンテナンス用のコードジェネレータでできることをわざわざ手間暇かけて伝えようとしているのではありません！

サンプルアプリの要件

- アーキテクチャや共通コンポーネント、お作法

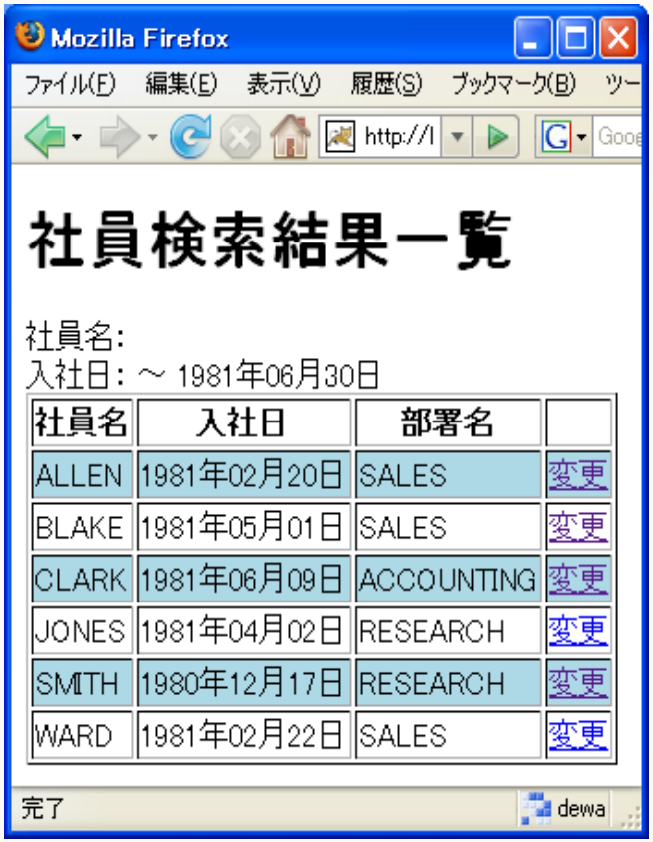
- 一部の機能だが最終品質レベルを目指す
  - 正常系だけでなく、異常系も考慮
  - 最低でも「参照系」と「更新系」を1機能づつ作る
  - 生産性・保守性・開発容易性・セキュリティ・運用・パフォーマンス等を考慮する

サンプルアプリ開発は難しい？

- そもそもWebアプリ特有の非機能要件の対応は易しくない
- フレームワークやライブラリが提供している個別の機能をどのように組み合わせて開発すれば良いかが分からない
- 正常系以外の要件をどこまで盛り込むかに迷う
  - 技術的な問題ではなく、プロジェクト管理の問題
  - 『できない』と『やらない』は違う

サンプルアプリの仕様

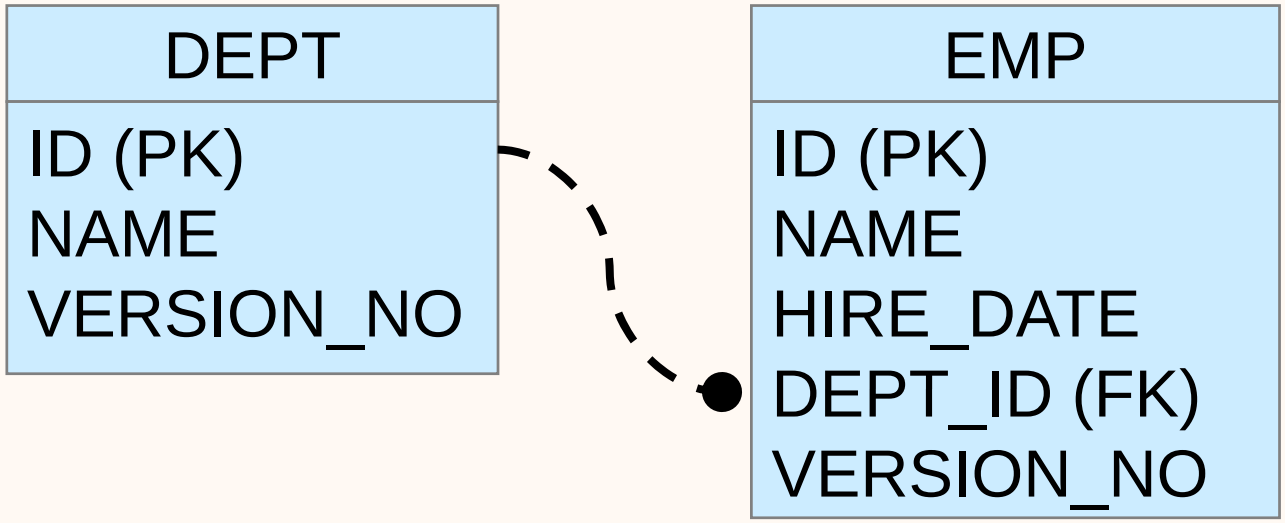
画面遷移図(参照系)



画面遷移図(更新系)

簡単な仕様ですが、応用が効きます！

ER図



- EMPはEmployeeの略で「社員」テーブル
- DEPTはDepartmentの略で「部署」テーブル

DDL

```
CREATE TABLE EMP (
    ID INTEGER NOT NULL PRIMARY KEY,
    NAME VARCHAR(20),
    HIRE_DATE DATE,
    DEPT_ID INTEGER,
    VERSION_NO INTEGER );

CREATE TABLE DEPT (
    ID INTEGER NOT NULL PRIMARY KEY,
    NAME VARCHAR(20),
    VERSION_NO INTEGER );

ALTER TABLE EMP ADD CONSTRAINT DEPT_ID_FK
    FOREIGN KEY (DEPT_ID) REFERENCES DEPT (ID);
```

ライブコーディング開始！

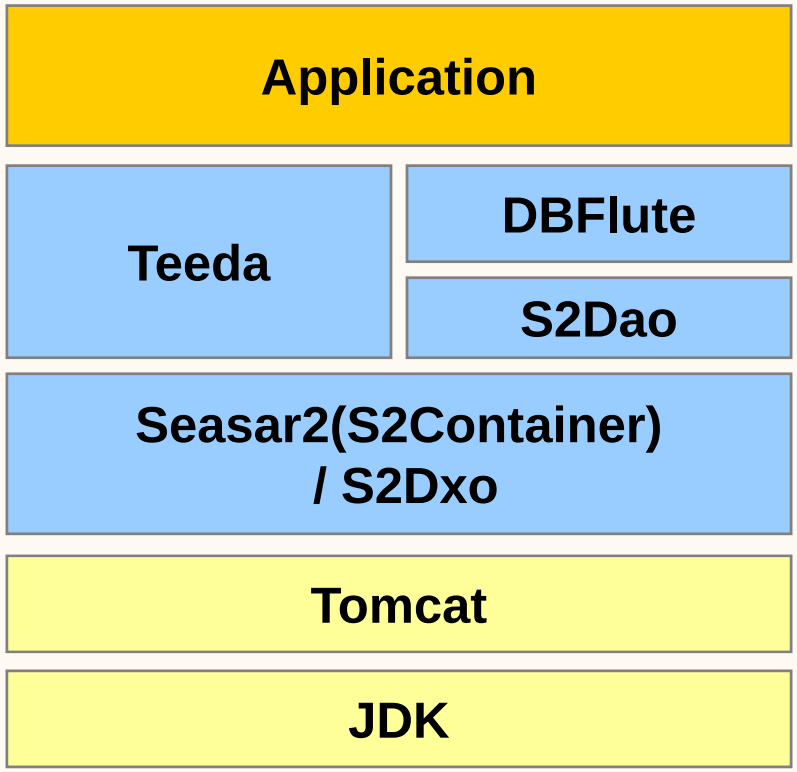
手順 ～ 準備編 ～

- Step 1: 開発環境の構築 ※省略
- Step 2: DB設計 ※省略
- Step 3: Eclipse起動 ※省略
- Step 4: プロジェクトの作成
- Step 5: DB起動
- Step 6: DBFluteのセットアップ
- Step 7: DBからコード生成
- Step 8: Tomcat起動

使用する主なS2関連プロダクト

ベースフレームワーク	Seasar 2.4
プレゼンテーション層のフレームワーク	Teeda
データアクセス層のフレームワーク	DBFlute
Eclipseプラグイン	Dolteng

ミドルウェア構成



Seasar2.4.18-RC1
Teeda 1.0.11-SP1
S2Dao 1.0.47-RC1
DBFlute 0.5.7

開発環境

IDE	Eclipse 3.2.2 (Language Pack)
IDE用プラグイン	Dolteng 0.24.0 EMecha 0.1.0 S2JSF プラグイン 1.1.1 DbLancher 0.1.0
Web	Tomcat 5.5.23
DB	H2

あまり聞いたことないなぁ～。

EMecha とは？

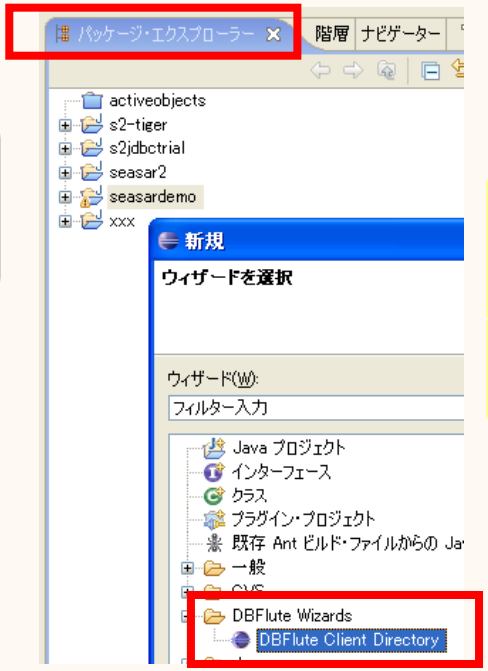
- DBFluteの開発環境構築を簡単にするEclipseプラグイン

- Eclipseプラグイン更新サイトから入手可能
  - http://eclipse.seasar.org/updates/3.2/

EMecha の使い方

1. パッケージ・エクスプローラーにて Dolteng で作成したプロジェクトを選択
2. メニューから[新規] - [その他] - [DB Flute Wizards] - [DBFlute Client Directory] を選択

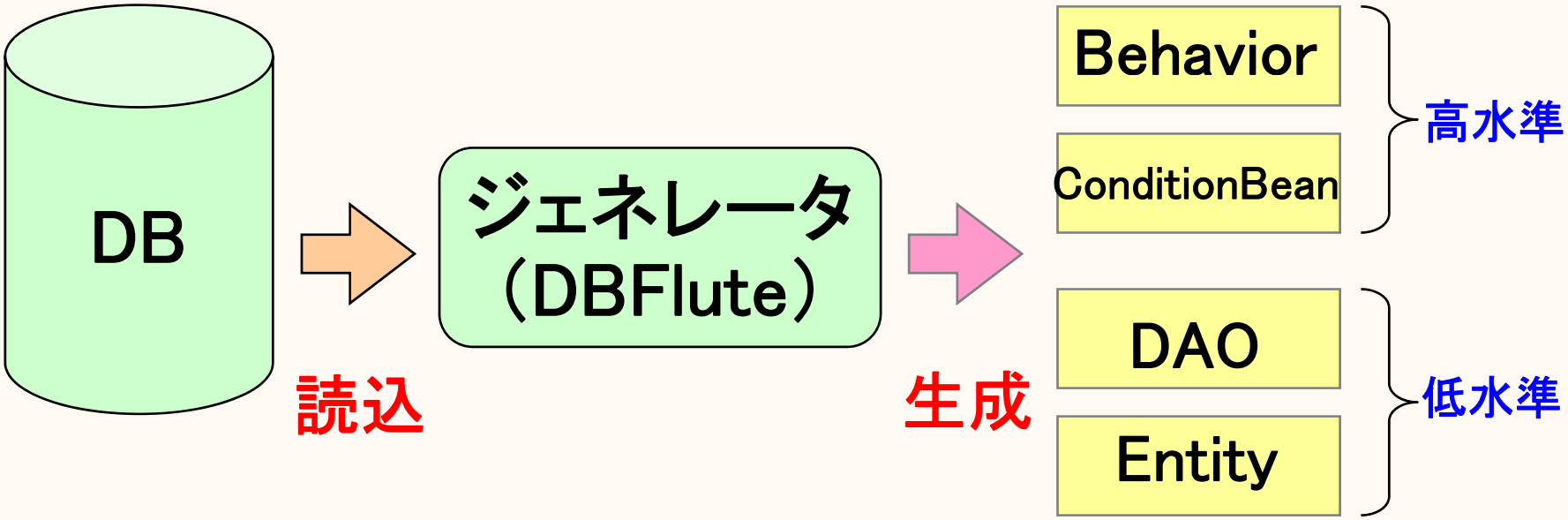
ナビゲーターとかでは動かないので注意！！



後の手順は省略（何とかなると思う）

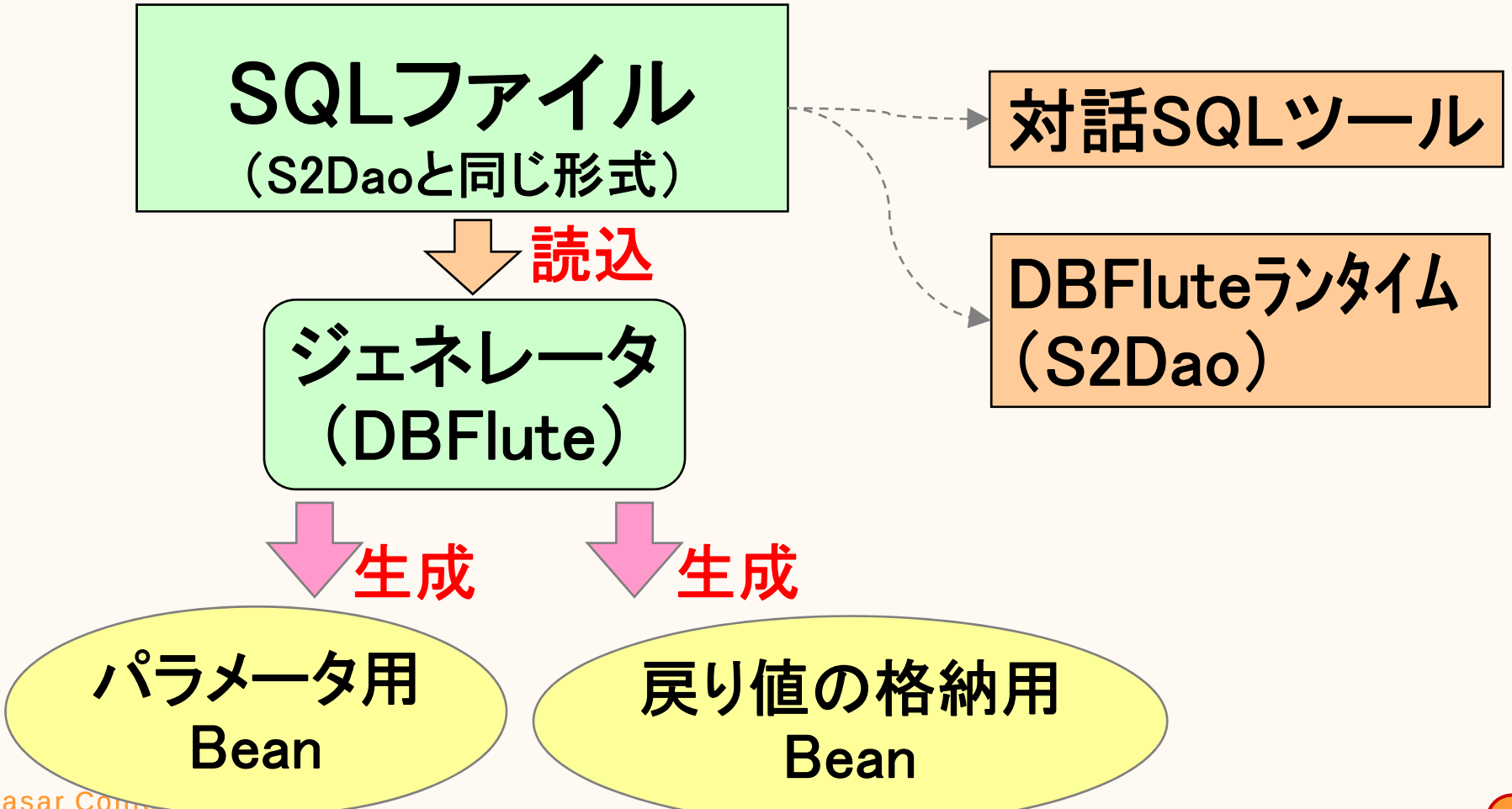
DBFluteの概要①

- DBからDBアクセス用のコードを自動生成
  - アプリケーションの約9割はこれでまかなう



DBFluteの概要②

- 複雑な処理は『外だしSQL』で対応する



手順 ～ 参照系アプリ ～

- Step 1: HTMLファイルからPageクラス生成
- Step 2: 検索条件入力ページ
  - 画面遷移
- Step 3: 検索結果一覧ページ
  - 一覧処理(データアクセス、モデル変換)
  - 絞りこみ・並べ替え
  - 日付書式

インジェクション

- @Bindingアノテーションによるインジェクション
  - Setterインジェクションの代替

ソースコードの
可読性が向上する

public スコープで
なくても良い

インジェクションの失敗時に
null をセットするではなく、
例外発生させる為の指定

```
@Binding(bindingType = BindingType.MUST)
ValueLabelService valueLabelService;
```

複数のクラスから使用する
場合は、共通の親クラスに定義する

フィールド名は
コンポーネント名にする

サクサク開発のテクニック①

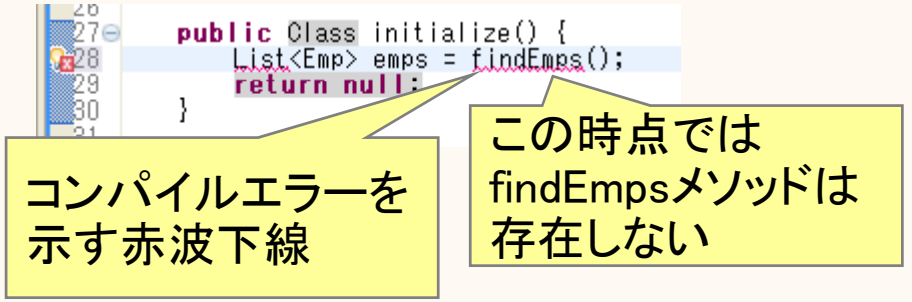
- わざとコンパイルエラーを作りこみ、Eclipseに修正候補を提示させるテクニック
  - お手軽で強力、かつ、応用が効く
    ぜひマスターしよう！
  - 次の特性によって生み出されたサクサク感
    - コードジェネレータ
    - タイプセーフ言語
    - IDEサポート
    - ホットデプロイ

サクサク開発のテクニック②

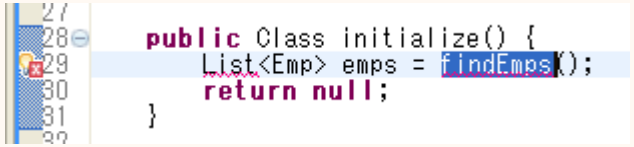
- 基本手順
  1. わざとコンパイルエラーを作りこむ
     - 例:
       - ・存在しないメソッドをでっちあげて呼び出そうとする
         ⇒ メソッドの雛形を自動生成
       - ・宣言してない変数を使ってメソッド呼び出し
         ⇒ 変数宣言のコード生成
  2. エラー箇所に素早く移動する
  3. Eclipseに修正候補を提示させて選択する

サクサク開発のテクニック③

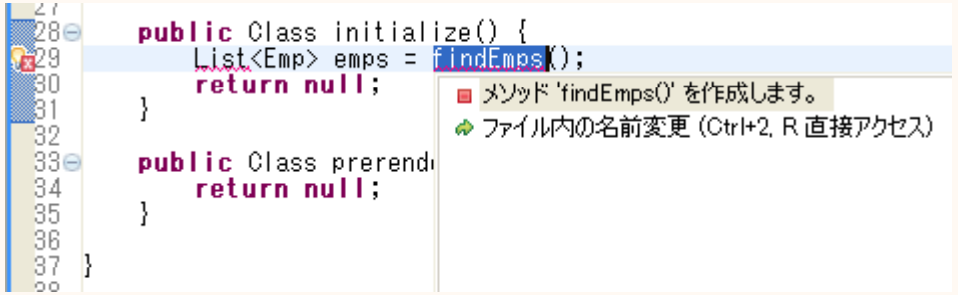
① 未定義メソッド呼出等でコンパイルエラーをでっち上げる



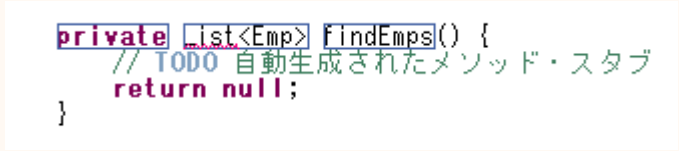
② Ctrl + < でエラー箇所へ移動する
（Ctrl + > でもOK）



③ Ctrl + 1 でエラー修正候補を表示して選択



④ メソッドの雛形が生成される



S2Dxoのポイント①

- 役割
  - マッピング処理を簡素化する
- 特徴
  - インターフェースにメソッドを宣言するだけ
    - 実装クラスは書かなくて良い
  - 同名プロパティの自動マッピング
  - マッピング漏れはアノテーションで指定する

S2Dxoのポイント②

- Before ー S2Dxoを使わない通常のマッピング

```
EmpUpdatePage convert(Emp entity) {
    EmpUpdatePage page = new EmpUpdatePage();
    page.setEmpId(entity.getId());
    page.setEmpName(entity.getName());
    page.setHireDate(entity.getHireDate());
    page.setDeptId(entity.getDeptId());
    Stirng deptName = null;
    if (entity.getDept() != null) {
        deptName = entity.getDept().getDeptName();
    }
    page.setDeptName(deptName);
    return page;
}
```

ネストに伴う
NullPointerException対策

S2Dxoのポイント③

- After ー S2Dxoを使ったマッピング

```
@ConversionRule("empId : id, empName : name")
EmpUpdatePage convert(Emp emp);
```

メソッド宣言と
アノテーションのみ！！

S2Dxoのポイント④

- Dxoインターフェースに定義するメソッド規約
  - 【その1】

```
@ConversionRule("変換先プロパティ:変換元プロパティ")
 変換先 convert(変換元);
```

  - 【その2】

```
@ConversionRule("変換先プロパティ:変換元プロパティ")
 void convert(変換元, 変換先);
```

※ メソッド名のconvertは慣習的なもので、実際には任意のメソッド名でOKです。

S2Dxoのポイント⑤

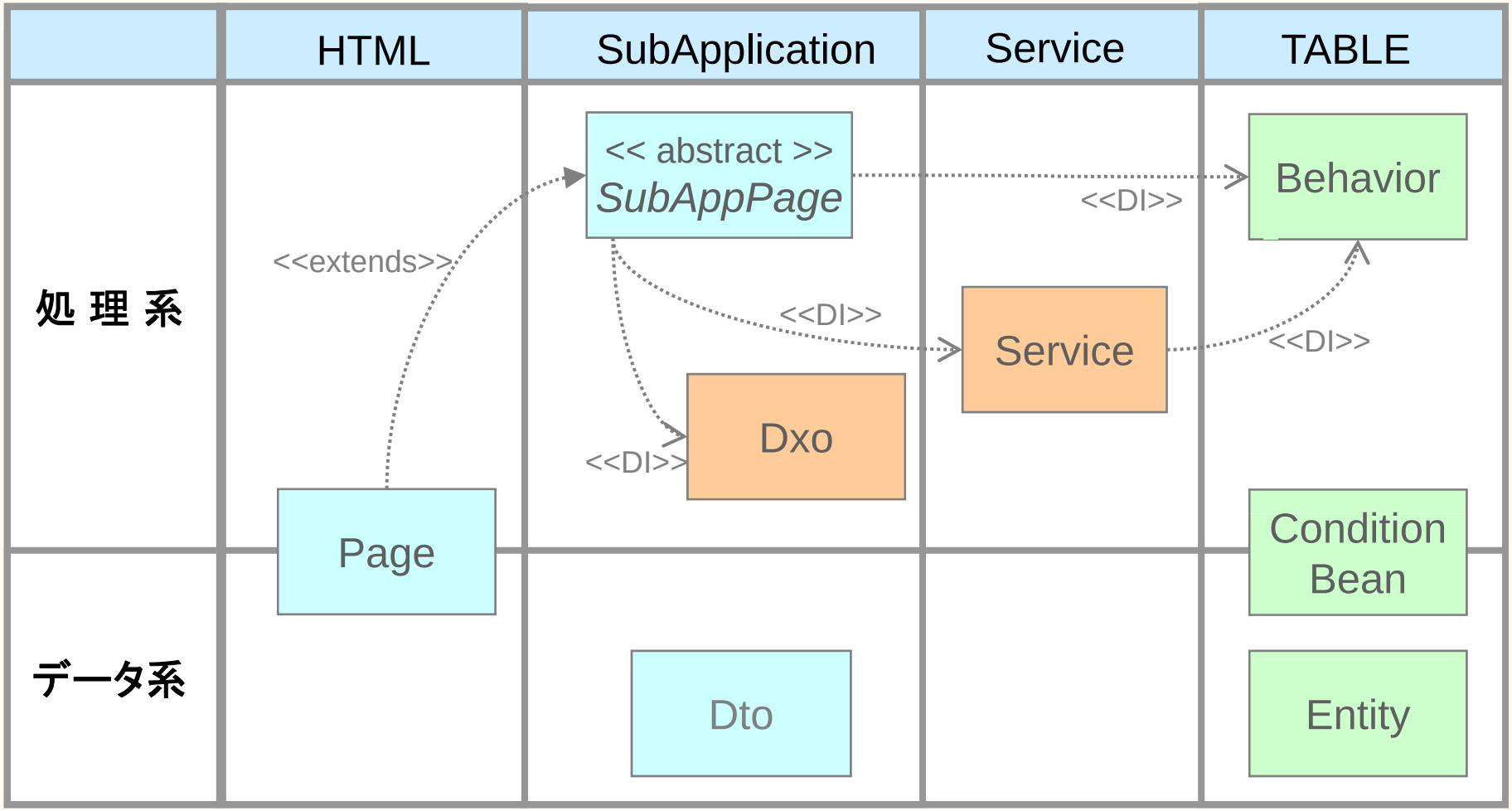
- 例：自動マッピングしない社員名と部署名を明示的に指定してマッピング
  - 【変換先】 EmpListPageクラス側
    - 社員名： empNameプロパティ
    - 部署名： deptName プロパティ
  - 【変換元】 Empクラス側
    - 社員名： namaプロパティ
    - 部署名： dept.name プロパティ

```
@ConversionRule(“empName : name, deptName : dept.name”)
EmpDto[] convert(List<Emp> emps);
```

手順　- 更新系アプリの実装 -

- Step 1: HTMLファイルからPageクラス生成
- Step 2: 一覧ページからの遷移
- Step 3: 社員変更ページ
  - プルダウンリスト
  - 1件分の照会処理、モデル変換
  - バリデーション
- Step 4: 社員変更確認ページ
  - ラベル表示
  - モデル変換、テーブル更新処理

モジュール相関図



Doltengが生成　手作り　DBFluteが生成　※ SubApplicatonの粒度は「ディレクトリ」

New Window①

- 新規ウィンドウオープンはバグの温床
  - 複数のウィンドウ間でのメモリ空間（セッション）の共有を意識していないと、思わぬバグを引き起こす
- TeedaのNew Window機能
  - 複数の新規ウィンドウを開いても、各ウィンドウごとに別々のメモリを確保するため互いに影響を及ぼさない！

New Window②

- やり方はたったこれだけ
  - Aタグに target="_blank" を指定する

```
<a href="hoge.html" target="_blank"/>hoge</a>
```

※ リンク先のクエリーストリングに newwindow=true を含めるやり方もある
※ 参考URL：
http://teeda.seasar.org/ja/extension_features.html#newwindow

確認ページのコツ①

- 「表示用項目」と「サブミット用項目」を用意
  - 【課題】
    - Page側で表示用とサブミット用の
      プロパティを二重に用意することは避けたい
    - HTML内で同一のidが存在することはNG
  - 【解決策】
    - HTML側でハイフン付きのidを使う
      - idに-（ハイフン）を含めておくと、ハイフンから後ろは無視される
      - HTML内で同一のidを複数回使用する時に有効

確認ページのコツ②

HTML側

表示用

サブミット用

```
社員名：<span id="empName" >SMITH</span>
        <input id="empName-hidden" type="hidden" />
入社日：<span id="hireDate" >2007年09月10日</span>
        <input id="hireDate-hidden" type="hidden" />
```

-(ハイフン)以降は
無視される

サブミット用の項目は
HIDDENにして、
見えないようにする

Page側

```
public String empName;
public Date hireDate;
```

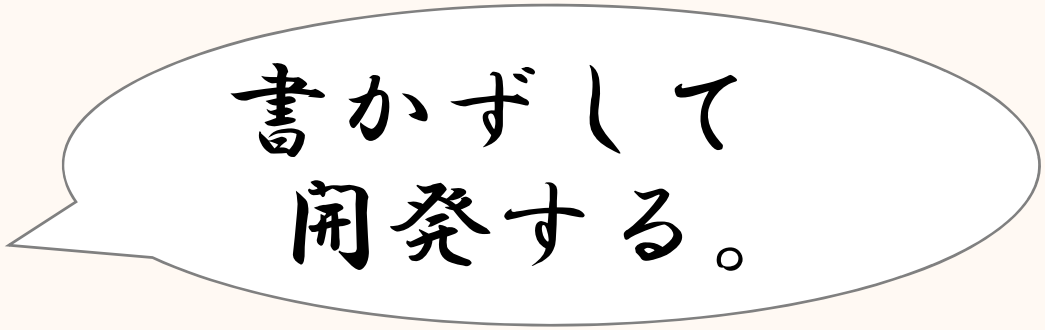
Page側の empNameプロパティは
HTML側の empName と
empName-hidden の両方に対応

手順 – 共通系処理 -

- Step 1: レイアウト機能
- Step 2: 共通エラーページ
- Step 3: ダブルサブミット対策

書かない技術①

- 異常系のソースコードを
  書かないが品質を確保できるのが理想
  - 正常系APIを呼ぶだけで、透過的に異常系処理が考慮される仕組み



言いたいだけ？

書かない技術②

- 例えば、
  「社員一覧ページ」から「社員変更ページ」へ遷移する時
  - パタメーターとして社員IDを受け取り、
    DBアクセスして、該当社員データを取得する

```
Emp emp = empDao.find(empId);
```

一見、問題無いように思われるが、、、

書かない技術③

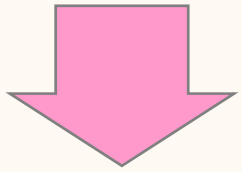
- empId に該当するレコードが見つからない時、DAOメソッドは null を返す
  - nullを想定した存在チェックのコードを書かないとNullPointerExceptionが発生してしまう

  - 初級者は異常ケースの処理を書かない人が多く、品質低下をまねく

書かない技術④

- そこで「実行時例外」と「共通エラーページ」を組み合わせたテクニックを使う

```
Emp emp =
    empBhv.selectByPKValueWithDeletedCheck(empId);
```



該当レコードが無い場合、
実行時例外である
EnityAlreadyDeletedException が発生する

該当データは既に
削除されました。

共通エラーページ
(error.html & ErrorPage.java)
を使って、メッセージを表示する

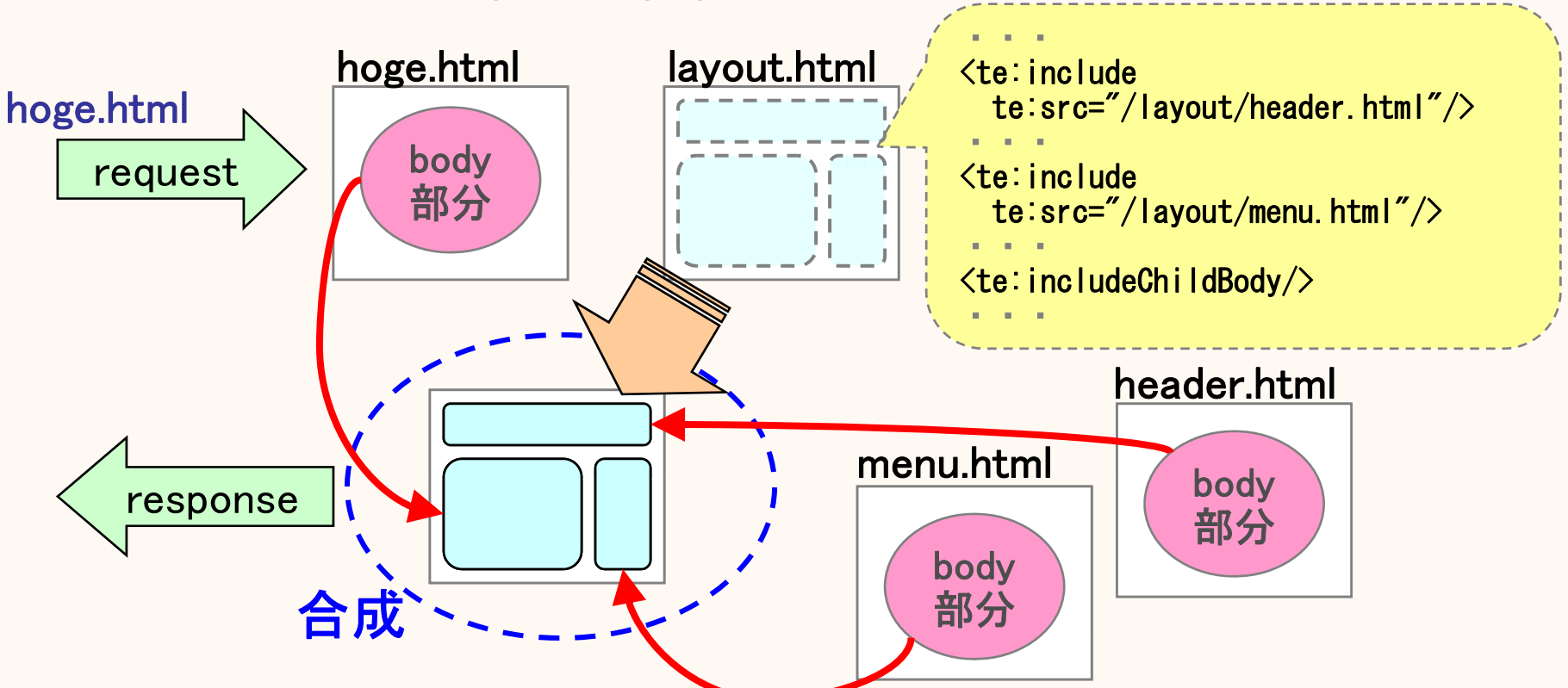
書かない技術⑤

- 今回のように、DBFluteのBehaviorを使えば、実行時例外を逆手にとった手法により「省力化（書かない）」と「品質向上（バラつき無し）」の両方を入手可能

- Behavior(Bhv)はDBFluteが生成するクラス
  - 開発現場から生まれたDAOとEntityのベストプラクティス的な定番処理を扱う
  - 責務はSQL実行より抽象度が高いDBアクセスモジュール

レイアウト機能（Teeda）

- 共通のヘッダ、フッタ、メニュー等を簡単に扱える
  - view/layout/layout.html を用意するだけで自動適用
    - と言いつつ LayoutPage.java は必ず用意すること！



http://teeda.seasar.org/ja/extension_features.html#layout

ダブルサブミット対策

- KumuのJavaScriptライブラリを使う
  - Kumu.Html.Disabled で二重送信を防止
  - 機能
    - formの二重送信防止
    - アンカーの二重クリック防止
  - 導入
    1. jsファイルをダウンロードして適当な箇所に配置
    2. HTMLファイルのheadタグ内でjsファイルを読み込む

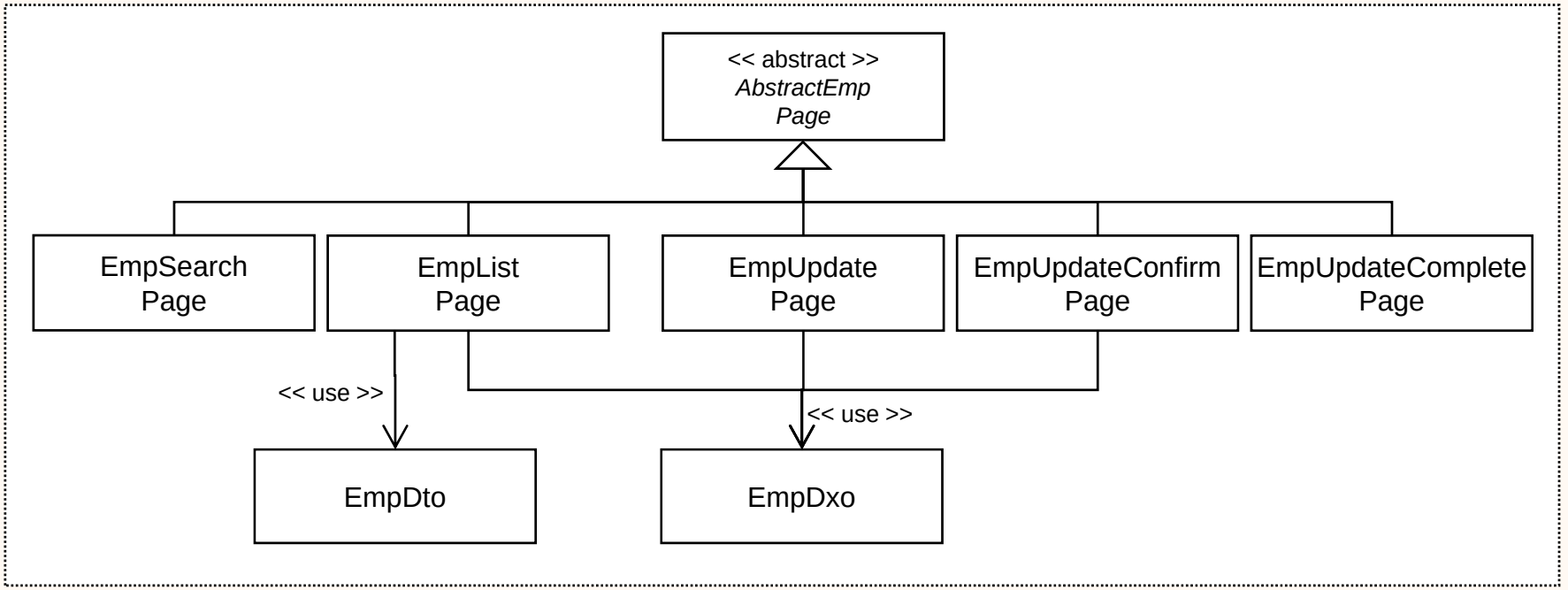
```
<script type="text/javascript" src="./js/kumu.js"></script>
<script type="text/javascript" src="./js/event.js"></script>
<script type="text/javascript" src="./js/disabled.js"></script>
```

＊ http://teeda.seasar.org/ja/kumu_disabled.html

Appendix

クラス図

- sample.web.empパッケージ



ソースコードのポイント

AbstractEmpPageクラス

```
public class AbstractEmpPage {

    @Binding(bindingType = BindingType.MUST)
    Emp empBhv;

    @Binding(bindingType = BindingType.MUST)
    EmpDxo empDxo;

    @Binding(bindingType = BindingType.MUST)
    ValueLabelService valueLabelService;

}
```

ソースコードのポイント

EmpListPage#prerenderメソッド

```
public EmpDto[] empItems;

public Class prerender() {
    List<Emp> emps = findEmps();
    empItems = empDxo.convert(emps);
    return null;
}
```

ソースコードのポイント

EmpDtoクラス

```
public class EmpDto implements Serializable {

    /* 社員ID */
    public Integer id;

    public String empName;

    public Date hireDate;

    public String deptName;
}
```

ソースコードのポイント

EmpListPage#findEmpsメソッド

```
protected List<Emp> findEmps() {
    EmpCB cb = new EmpCB();

    /* テーブル結合:DEPT */
    cb.setupSelect_Dept();

    /* 範囲検索:入社日 */
    cb.query().setHireDate_DateFromTo(condFromHireDate, condToHireDate);

    /* 先頭一致:社員名 */
    cb.query().setName_PrefixSearch(condEmpName);

    /* 並べ替え:社員名(昇順) */
    cb.query().addOrderBy_Name_Asc();

    /* 検索の実施 */
    return empBhv.selectList(cb);
}
```

ソースコードのポイント

EmpDxoインターフェース

```
public interface EmpDxo {
    /* 社員一覧ページで使う変換処理 */
    @ConversionRule("empName : name, deptName : dept.name")
    EmpDto[] convert(List<Emp> emps);

    /* 社員更新ページで使う変換処理 */
    @ConversionRule("empName : name, deptName : dept.name")
    void convert(Emp src, EmpUpdatePage dest);

    /* 社員更新確認ページで使う変換処理 */
    @ConversionRule("name : empName")
    Emp convert(EmpUpdateConfirmPage page);
}
```

ソースコードのポイント

EmpUpdatePage#prerenderメソッド

```
public List<ValueLabelDto> deptIdItems;

public Class prerender() {

    /* プルダウンリスト（部署）のデータ準備 */
    deptIdItems = valueLabelService.findDepts();


    /* DBから1件分のEntityを取得(削除されていれば例外をスロー)*/
    Emp emp = empBhv.selectByPKValueWithDeletedCheck(id);

    /* EntityのプロパティをPageクラスのプロパティへマッピング */
    empDxo.convert(emp, this);

    return null;
}
```

ソースコードのポイント

ValueLabelDtoクラス

```
public class ValueLabelDto implements Serializable {

    private static final long serialVersionUID = 1L;

    public Object value;

    public String label;

}
```

これがないと
例外が出る

ソースコードのポイント

EmpUpdateConfirmPage#doOnceUpdateメソッド

```
/* doOnceメソッドによるダブルサブミット防止 */
public Class doOnceUpdate() {

    /* モデル変換後にテーブル更新 */
    Emp emp = empDxo.convert(this);
    empBhv.update(emp);

    /* 社員更新完了ページへ遷移 */
    return EmpUpdateCompletePage.class;
}
```

最後に

- 復習重要。復習はこちらで！

WEB + DB PRESS vol. 41（技術評論社）
特集2：つらいJavaから楽しいJavaへ
Seasar2 サクサク開発　実践カリキュラム

サンプルコードのダウンロード

http://www.gihyo.co.jp/magazines/webdbpress/support/Vol41/

ありがとう
ございました！