

Practice Makes Perfect
The Ashikunep Kotan 伍

WEBフレームワーク

Ikushipe

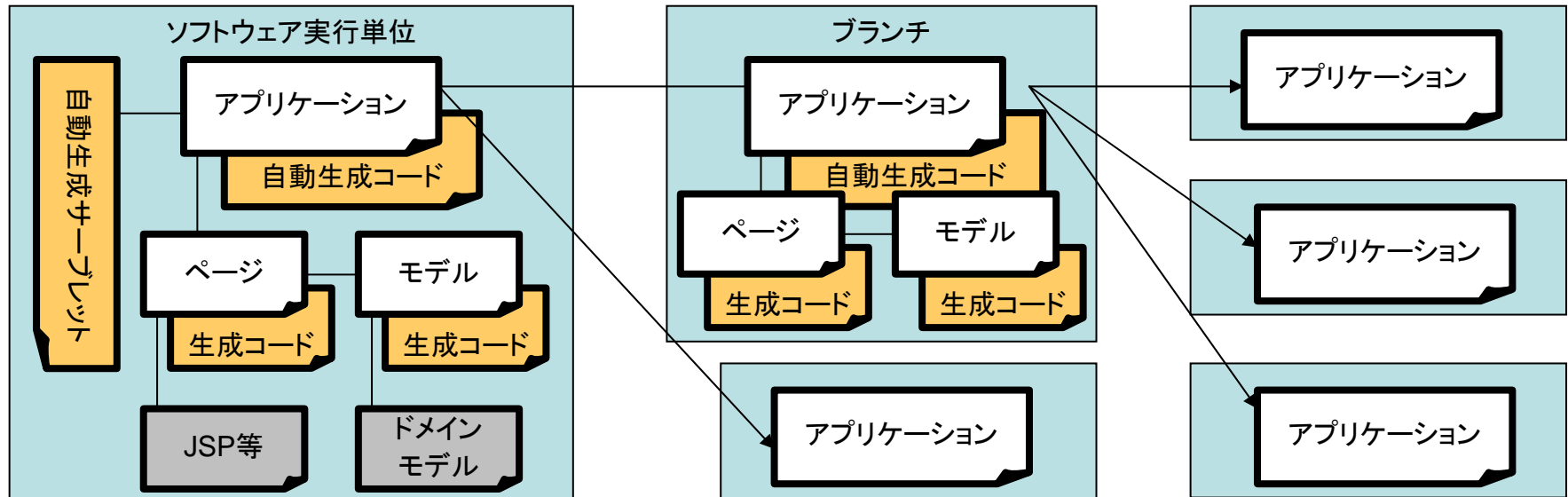
アーキテクチャ

2006年8月24日

- Ikushipeとは、Java製のWEBフレームワークで、Servlet APIの上で構築されている
- Ikushipeの機能
 - 画面遷移の管理
 - リンククリック時のイベント発生、遷移先決定を行う
 - DTOと画面のバインディング
 - 画面の描画自体はJSPやMayaaなどを用いる
 - ビジネスロジックやデータアクセスは、既存のSeasar2等のDIコンテナやEJB3、O/Rマッピングツールなどを用いる
 - フォーム入力の値処理
 - 型変換
 - バリデーション

- 型付指向
 - Java5.0のアノテーションによってPOJO/POJIに設定
 - ユーザーコードを、型付が厳しいスタイルに誘導
- ページ指向
 - ページ(=画面)単位で開発する
 - フォームの送信は表示ページにて受け取る
 - ページモデル(=DTO)はページ毎にひとつ
- コンパイル時ソリューション
 - ユーザーコードをコンパイルするタイミングで処理
 - コードを自動生成する
 - リフレクションを極力廃し、実行時オーバーヘッド軽減
 - POJO/POJIの規約とアノテーションを手がかりに制御コードを仕込んだアダプタクラスを生成

- ソフトウェア実行単位
 - 同一のパッケージに属するPOJO/POJIによって作られるページ・モデル・アプリケーションをまとめる
 - コンパイル時に、コード自動生成によって、ページ・モデル・アプリケーションのそれぞれサポートクラスを生成し、さらに起動用のサーブレットを生成する
- ブランチ機能
 - 実行単位(ブランチ)をまとめ、大きなソフトウェアを作る



- アプリケーションの実装
 - @WebApplicationアノテーションによってPOJO/POJIを修飾
 - アプリケーションレベルの初期化処理を記述
 - @PageResolverでページの名前解決を行うハンドラ設定。@PageResolverアノテーションはデフォルトの属性値でよい場合、省略可能。
- フレームワークがサーブレットを生成
 - アプリケーションと同じパッケージに、@WebApplicationアノテーションのservletName属性に指定した名前で作成
 - 設定省略時は、デフォルトの”ApplicationServlet”で生成。
- ウェルカムページの指定
 - “/”にてアクセスされた際には、@WebApplicationアノテーションのwelcomePage属性に指定したページレゾルバで解決する

```
@WebApplication(servletName="SimpleSampleServlet",  
    welcomePage="simple")  
public interface SimpleApplication {  
    @PageHandler // 省略可能  
    Class<SimplePage> simple();  
}
```

- 返値型での指定
 - 返値ではClass型が求められる。名前に対してページ固定の場合は、型引数に解決結果を指定する。この場合、抽象メソッドでよい
 - `@PageResolver`の`possibleTo`属性に、返す可能性のあるページクラス型の配列を指定する。この場合、メソッドの返値は`Class<?>`とする。
- レゾルバメソッドの引数でサービス要求
 - 名前解決の際、DI等で供給されるバッキングサービスを利用できる。その際は、必要コンポーネントの型をメソッド引数に指定する。
- 重複の問題
 - ページの名前がレゾルバメソッド名。そのため`possibleTo`属性を用いる場合、アプリケーション内のページレゾルバ全体で、クラス型から名前へ一意の関係でなければいけない。

@WebApplication

```
public abstract class SwitchApplication {  
    public abstract Class<SimplePage> index();  
    @PageHandler(possibleTo={LoginPage.class, MemberPage.class})  
    public Class<?> switch(MyPageManager mpm, Locale locale) {  
        return mpm.getPageClass(locale);  
    }  
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
  <servlet>
    <servlet-name>calc</servlet-name>
    <servlet-class>
      org.ashikunep.ikushipe.sample.calc.ApplicationServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>calc</servlet-name>
    <url-pattern>/calc/*</url-pattern>
  </servlet-mapping>
</web-app>
```

↑
指定名でフレームワーク
が生成するサーブレット

↑
フォルダおよび拡張子
のどちらでもマップ可能

- アプリケーションのツリー化
 - 子ブランチのクラス型を返値で示すメソッドを `@BranchResolver` アノテーションで修飾することによって指定。
- アプリケーション間でのページ遷移
 - 名前で公開するページは `@PageResolver` アノテーションの `export` 属性の値を `true` にする。
 - アップストリームは公開ページ名で遷移可能。
 - コンパイル時に必ずしも参照の解決ができるとは限らないため
 - ダウンストリームは公開ページ名もしくはページクラス型とも遷移可能。
 - コンパイル時に必ず参照の解決ができるため。

```
@WebApplication
public interface MainApplication {
    @BranchResolver
    Class<SubApplication> sub();
    @PageResolver(export=true)
    Class<MainPage> index();
}
```

- サーブレット初期化時に、アプリケーションの初期化ハンドラの実行を行う
 - @InitHandlerアノテーションで修飾したメソッドを順に実行していく
 - アプリケーションスコープのモデルやコンポーネントの場合は、ハンドラで用いることができる
 - ハンドラの引数に宣言しておくフレームワークが自動で要求モデル・コンポーネントを用意し、引き渡す

@WebApplication

```
public abstract class TaskApplication {  
    public abstract Class<TaskPage> index();  
    @InitHandler  
    public void prepare(TaskModel model) {  
        List<Task> taskList = new ArrayList<Task>();  
        model.setTask(taskList);  
    }  
}
```

フレームワーク
が用意して引き渡し

- 例外処理をアプリケーション統一に行う
 - @ThrowableHandlerアノテーションでハンドラメソッドを修飾する
 - ハンドラは画面遷移先を返す
 - アプリケーションスコープのモデルやコンポーネントの場合は、ハンドラで用いることができる
 - 例外もメソッドの引数に宣言すれば受け取れる

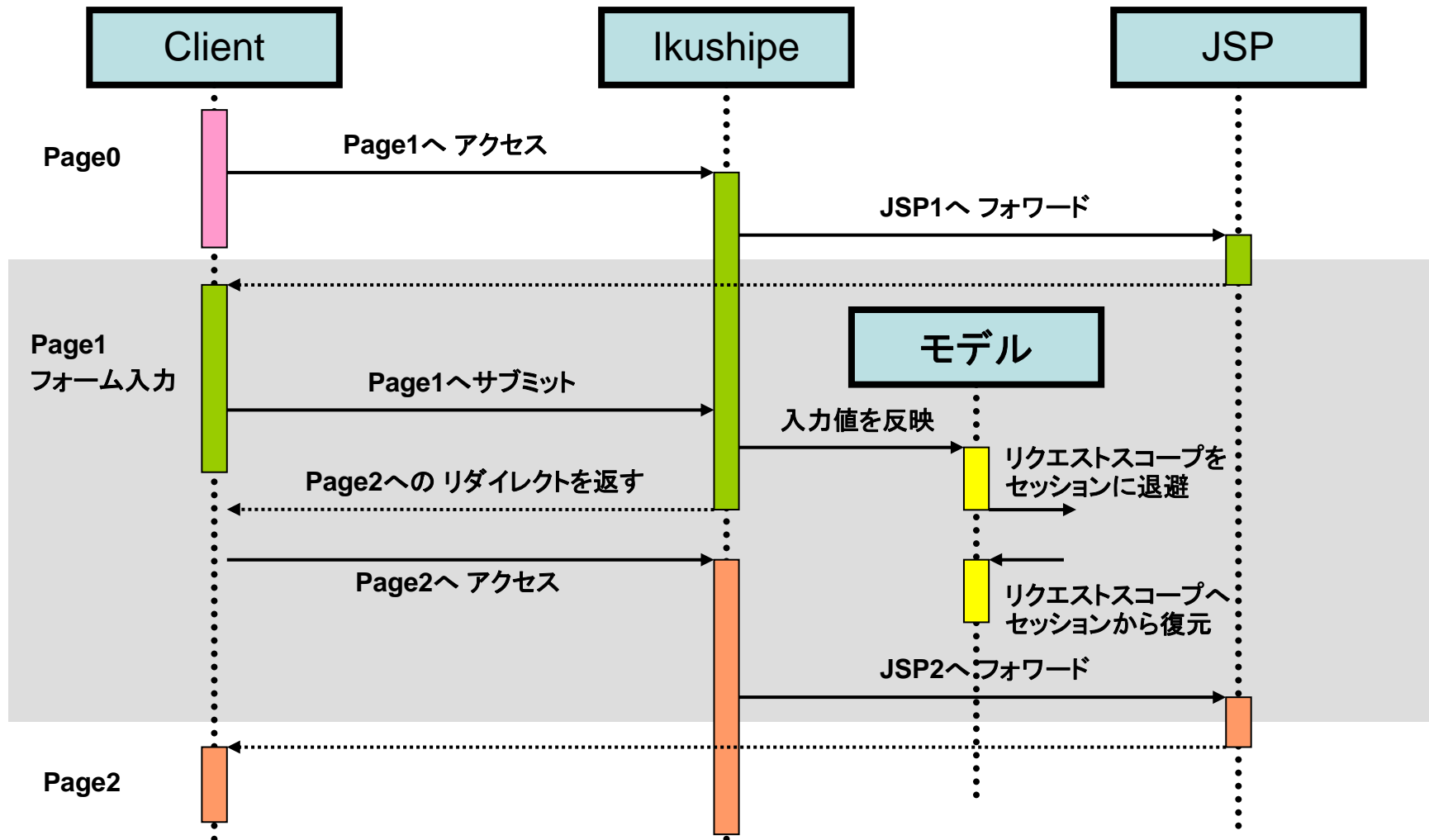
```
@WebApplication
```

```
public abstract class CalcApplication {  
    public abstract Class<CalcPage> index();  
    @ThrowableHandler  
    public Class<ErrorPage> handleAllException(  
        Throwable t, LogManager logManager) {  
        logManager.error(t);  
        return ErrorPage.class;  
    }  
}
```

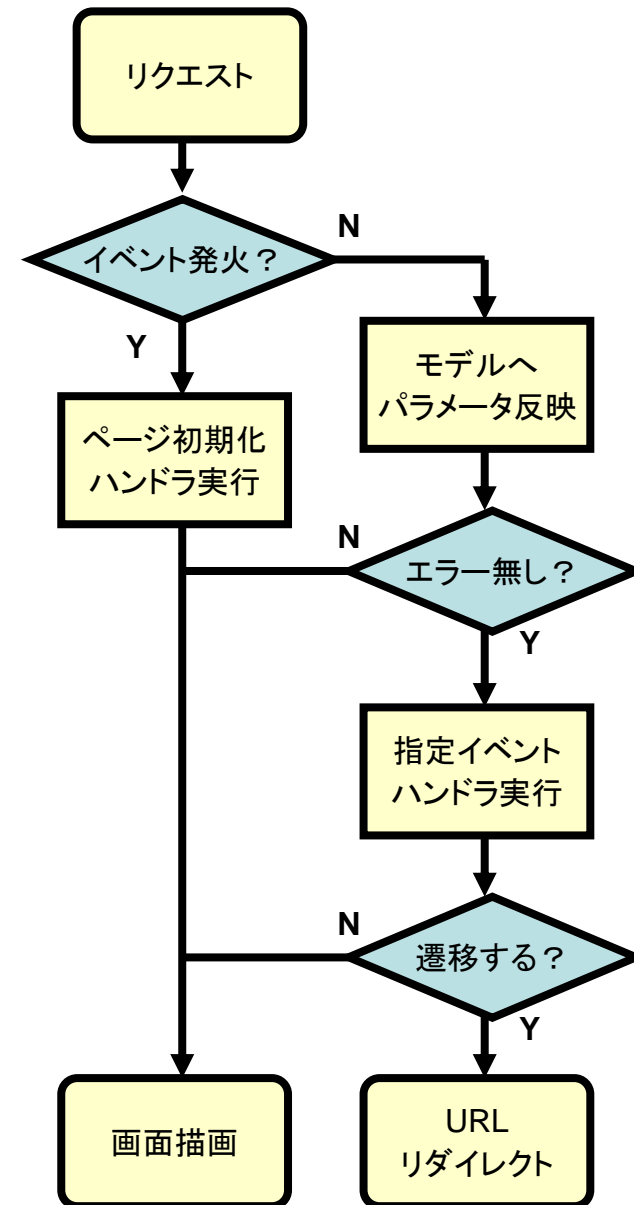
処理後の遷移先

- /Context[/Folder][Branch[/Branch]...]/Page[.Extention]
 - URLはアプリケーションの構成を示す
- Context
 - WEBアプリケーションの名前。
- Folder
 - web.xmlのservlet-mappingに「/Folder/*」と設定した場合。
 - Extentionと排他。
- Branch
 - @BranchResolverで解決されたブランチの階層。
- Page
 - @PageResolverで解決されたページ名。
- Extention
 - web.xmlのservlet-mappingに「*.Extention」と設定した場合。
 - Folderと排他。

- フォワードとリダイレクトを併用



- リクエストを受けると、リクエストURIに対応したページオブジェクトを取得
 - リクエストパラメータを調べ、イベント発火を行うかどうかを判断して処理分岐
- ページへの通常描画シーケンス
 - ページ内の初期化ハンドラを実行
 - 画面描画を行う
- フォームSubmitおよびイベントへのリンクの場合のシーケンス
 - モデルへのパラメータ反映
 - 検証エラーが生じると、イベント発火を行わず、画面描画(もしくは例外ハンドラへ)
 - パラメータの指定するページ内のイベントハンドラを実行
 - 遷移する場合はリダイレクト、遷移しない場合は画面描画



- 画面の実装
 - @WebPageアノテーションによってPOJO/POJIを修飾
 - イベントハンドラメソッドを記述
 - イベントハンドラが引数を取る場合、フレームワークのほうで判断して必要なものを渡してくれる
 - ハンドラでは遷移先画面クラスを返すように実装

```
@WebPage(model=CalcModel.class)
public class CalcPage {
    public Class<ResultPage> add(CalcModel model) {
        long num1 = model.getNum1();
        long num2 = model.getNum2();
        model.setResult(num1 + num2);
        return ResultPage.class;
    }
}
```

- ロジックを持たないページはPOJIで作る
 - イベントハンドラメソッドは、単純な遷移先指定
 - メソッドの返値のClass型の型引数が、遷移先ページの型となる。
 - 遷移先ページを文字列で指定する場合には、`@EventHandler` アノテーションの`visitTo`属性で指定
 - 実装が無いため、テストが省略できる

```
@WebPage(model=CalcModel.class)
public interface ResultPage {
    Class<CalcPage> back();
    @EventHandler(visitTo="exportedPage")
    String gotoAnother();
}
```

- ページの一度目表示の際に初期化
 - イベント発火シーケンスではない場合
 - イベントURL描画/発火は自ページ内にて完結して行われるために、発火は必ず二度目表示以降
 - `@InitHandler`アノテーションで修飾、ハンドラの引数に宣言された要求モデル・コンポーネントが自動で渡されるのは、アプリケーションの初期化と同じ
- `@ThrowableHandler`で例外ハンドラを修飾
 - アプリケーションと同様

```
@WebPage(model=CalcModel.class)
public class CalcPage {
    @InitHandler
    public void prepare(CalcModel model) {
        model.setResult(0);
    }
}
```

- フレームワークでJSPカスタムタグを提供
 - フォーム機能を提供するタグ
 - リンク機能を提供するタグ
 - ループを便利に行うタグ

- /WEB-INF フォルダの下に配置する
 - 直接アクセスされないようにするため
 - /WEB-INF/contextName[/viewFolder]/viewId.jsp
 - @WebApplicationのviewFolder属性
 - @WebPageのviewId属性

- モデルの実装
 - @PageModelアノテーションによってPOJO/POJIを修飾
 - @PageModelアノテーションのscope属性に、ModelScope列挙値にてスコープ指定
 - デフォルトのスコープ設定は、ModelScope.**RENDERING**
 - ほか、**APPLICATION** / **COOKIE** / **SESSION** / **NOT_MANAGED**

```
@PageModel(scope=ModelScope.SESSION)
public class ServiceModel implements Serializable {
    String _userName;
    public String getUsername() {
        return _userName;
    }
    public void setUsername(String userName) {
        _userName = userName;
    }
}
```

- POJIでもモデルを実装することができる
 - プロパティを定義
 - Getter/Setter両方宣言して、どちらか一方でもよい
 - イベントハンドラなどで用いる必要のあるものを用意
 - フレームワークが補完的なコード生成
 - Getter/Setterセットの実装
 - 配列型もしくはjava.util.List型のプロパティは、インデックスアクセスできる補助的なアクセッサも実装 → ループ対応機能

```
@PageModel
public interface CalcModel {
    long getNum1();
    long getNum2();
    void setResult(long result);
}
```

- Formにて、チェックボックスを用いる
 - HTMLのinputタグにて、type=“checkbox”
 - チェックされて無いとパラメータが送信されないため、フレームワークで対応していないと状況更新ができない → チェックボックス問題
 - モデルでチェックボックスと関係づけるプロパティに `@ModelProperty` アノテーションの `alwaysDecode` 属性を **true** 設定する
 - パラメータが送信されない時、**null**あるいは0、**false**に更新する

```
@PageModel
```

```
public interface CheckBoxModel {  
    @ ModelProperty(alwaysDecode=true)  
    boolean getDMSendOK();  
}
```

- モデルへのプロパティ値反映の際に実行
 - モデルプロパティにアノテーションで設定
 - `@ModelProperty`アノテーションの`errorTo`に指定された名前があれば、`HttpServletRequest#getAttribute()`を経由してその名前で検証例外メッセージを取得できる。
 - イベントが発火されないで、同一画面が表示される

`@PageModel`

```
public interface LoginModel {  
    @ LengthValidator(min=1, max=16)  
    @ ModelProperty(errorTo="userNameError")  
    String getUsername();  
}
```

- 検証処理を委譲するPOJOを作成
 - 設定項目は、プロパティとして実装
 - 検証メソッドは、返値なし・引数に検証対象オブジェクトおよび検証処理に必要なコンポーネントを持つ
 - 検証エラーは例外メッセージとして投げる

```
public class SuffixValidatorDelegate {  
    private String _suffix;  
    public void setSuffix(String suffix) { _suffix = suffix; }  
    public void validate(String value) throws Throwable {  
        if(value == null || value.endsWith(_suffix) == false) {  
            throw new Exception(_suffix + “を末尾につけてください”);  
        }  
    }  
}
```

- 独自に設定用アノテーションを作成
 - @PropertyValidatorのdelegateTo属性に検証委譲先のPOJOを指定
 - 設定項目を属性として定義する
 - POJOの検証メソッドが”validate”ではないとき、@PropertyValidatorアノテーションのhandler属性にメソッド名を指定する

```
@PropertyValidator(delegateTo=SuffixValidatorDelegate.class)
public @interface SuffixValidator {
    String suffix();
}
```

@PageModel

```
public interface MailModel {
```

```
    @ SuffixValidator(suffix="@ashikunep.org")
```

```
    String getMailAddress();
```

```
}
```

カスタムバリデータ 利用例

- モデルへのプロパティ値反映の際に実行
 - バリデータと同様に、プロパティにアノテーションで設定
 - バリデータと同様に、変換時のエラーメッセージをモデルに格納することができる
 - バリデータと同様に、カスタムコンバータを、
`@PropertyConverter`アノテーションで作ることができる

`@PageModel`

```
public interface DateModel {  
    @ DateConverter("yyyy年MM月dd日")  
    @ ModelProperty(errorTo="birthdayError")  
    String getBirthday();  
    String getBirthdayError();  
}
```

- 開発サイト
 - <http://ikushipe.ashikunep.org/>
- 開発ソースコードリポジトリ
 - <https://www.ashikunep.org/svn/ikushipe/trunk/ikushipe>
- 動作環境
 - JavaVM
 - Java5.0
 - WEBコンテナ
 - Servlet2.4 & JSP2.0
- 名前の由来
 - Ikushipeはイクシペ、と読む
 - イクシペは、アイヌ語で「柱」の意味