

# Struts 1 日入門

株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

Version 0.9.004

# 本ドキュメントについて



- この作品は、クリエイティブ・コモンズの表示-改変禁止 2.1 日本ライセンスの下でライセンスされています。この使用許諾条件を見るには、<http://creativecommons.org/licenses/by-nd/2.1/jp/> をチェックするか、クリエイティブ・コモンズに郵便にてお問い合わせください。住所は: 559 Nathan Abbott Way, Stanford, California 94305, USA です。
- 本ドキュメントの最新版は、<http://www.knowledge-ex.jp/opendoc/struts.html> より入手することができます。

あなたは以下の条件に従う場合に限り、自由に



本作品を複製、頒布、展示、実演することができます。

あなたの従うべき条件は以下の通りです。



**表示.** あなたは原作者のクレジットを表示しなければなりません。



**改変禁止.** あなたはこの作品を改変、変形または加工してはなりません。

- 再利用や頒布にあたっては、この作品の使用許諾条件を他の人々に明らかにしなければなりません。
- 著作(権)者から許可を得ると、これらの条件は適用されません。
- Nothing in this license impairs or restricts the author's moral rights.

# Agenda

- Strutsとは
  - 概要
  - 2つのStruts
  - フレームワークの長所と短所
- Strutsの基本コンポーネント
  - ActionServlet
  - ActionForm
  - Action
  - JSP
  - アクションコンフィグレーションファイル

# Strutsとは

株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# Strutsとは

- Strutsとは
  - JavaでWebアプリケーションを開発する際に用いられるフレームワークのひとつ
  - オープンソースである
  - すでに多くの利用実績があり、信頼性が高い
  - 書籍やインターネットを通じて多くの情報を入手できる

# 2つのStruts

- Strutsには2つの系統がある
  - Apache Struts Action Framework
    - 通常のStruts (Struts 1.x系)
    - Struts 1.3として開発が継続されている
  - Apache Shale Framework
    - Struts 2.0として提案されたもの
    - Struts 1.x系とは違いが大きかったため、別プロジェクトになった

(※)本コースでは Struts Action Frameworkを扱います

# フレームワークの長所と短所

## 長所

- 共通したアプローチを開発者全員に強制できる
- どんなアプリケーションでも同様のアプローチで開発することになるので、高い保守性が期待できる
- 一定の品質が保障される

## 短所

- フレームワーク特有の開発パターン・技術を学習しなければならないので、習熟に一定の期間が必要

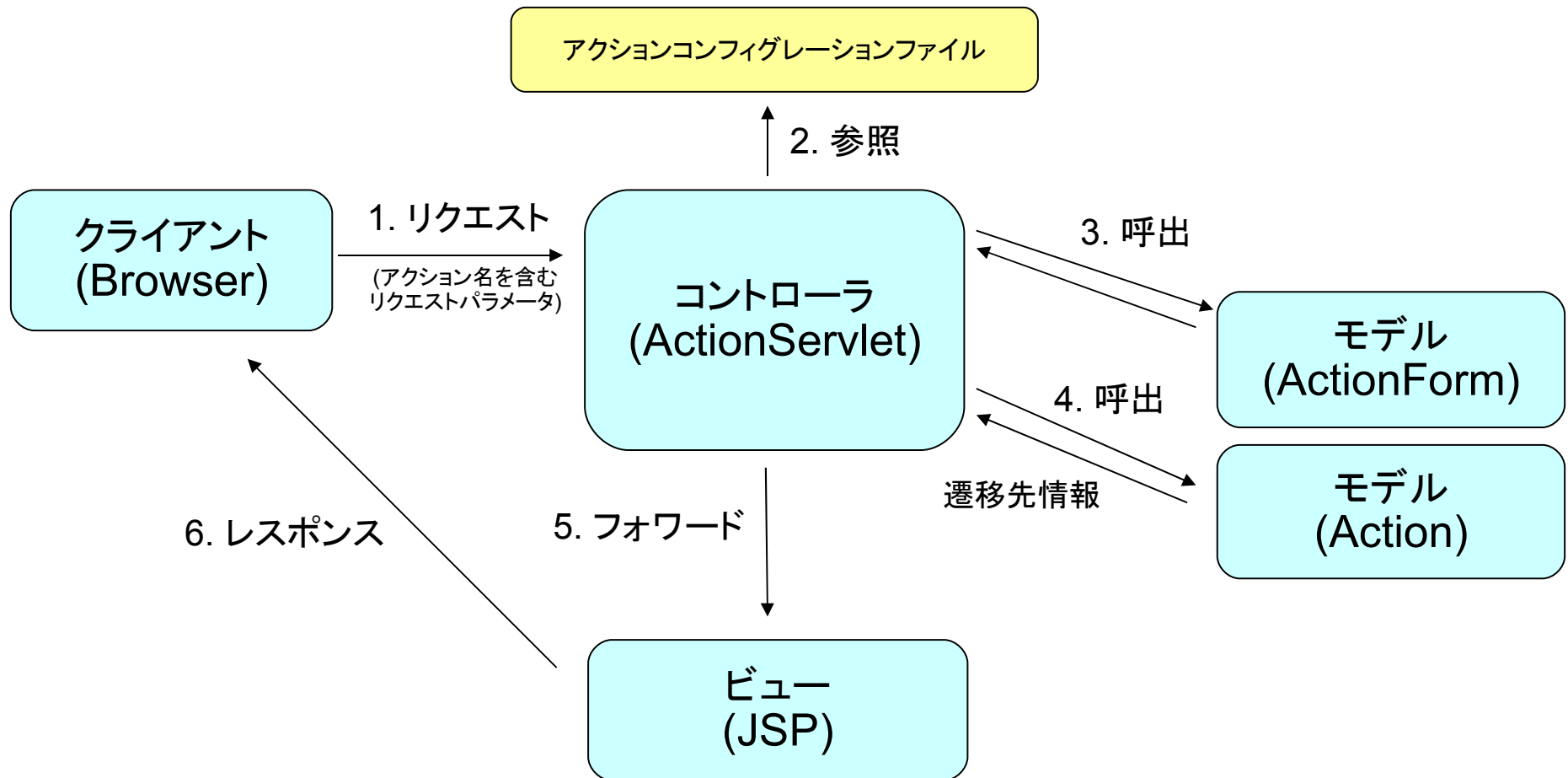
# Strutsの基本コンポーネント

- ActionServlet
- ActionForm
- Action
- JSP
- アクションコンフィギュレーションファイル
- web.xml



# Strutsのアーキテクチャ

- Strutsは「MVC モデル2」アーキテクチャに基づいている



# Strutsの処理フロー

1. ActionServletがアクション名を含むリクエストを受け取る
2. ActionServletはアクション名をアクションコンフィギュレーションファイルに照合し必要なActionFormやActionを特定する
3. ActionServletは特定されたActionFormのメソッドを実行し、必要に応じて処理結果を受け取る
4. ActionServletは特定されたActionのメソッドを実行し、遷移先ページについての情報を受け取る
5. ActionServletは Actionの処理結果をもとに適切なJSPへ画面を遷移(フォワード)する
6. JSPが実行されレスポンスを返す

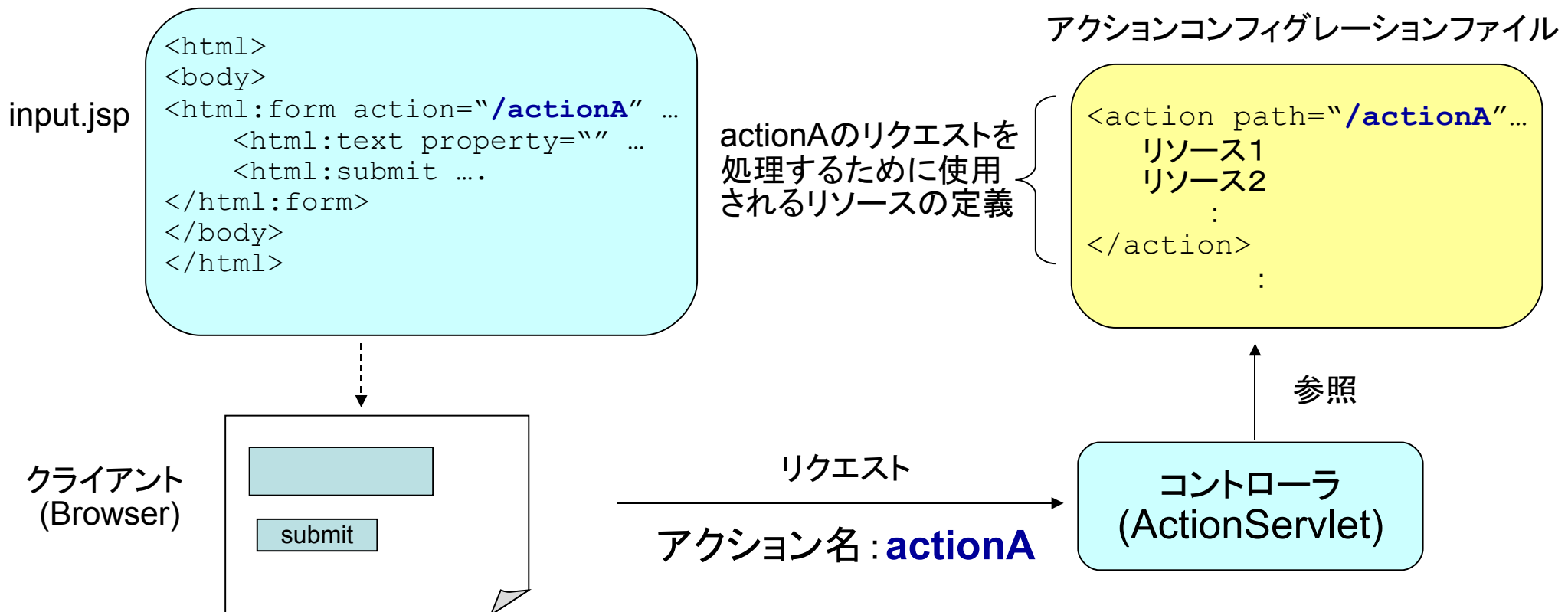
# アクション名

株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# アクション名

- アクション名は、あるリクエスト処理に必要なリソースを特定するための名前である

※ アクション名は任意に設定できるが、アクションコンフィグレーションファイルの定義とマッチしている必要がある



# アクション名の使われ方

1. アクション名はリクエストとともに送られる
2. ActionServletはアクション名を受け取る
3. ActionServletはアクションコンフィグレーションファイルを参照し、そのアクション名が定義されているセクションを特定する
4. ActionServletはリクエストを処理するためにそのセクションで定義されたリソースを使用する

# ActionServlet

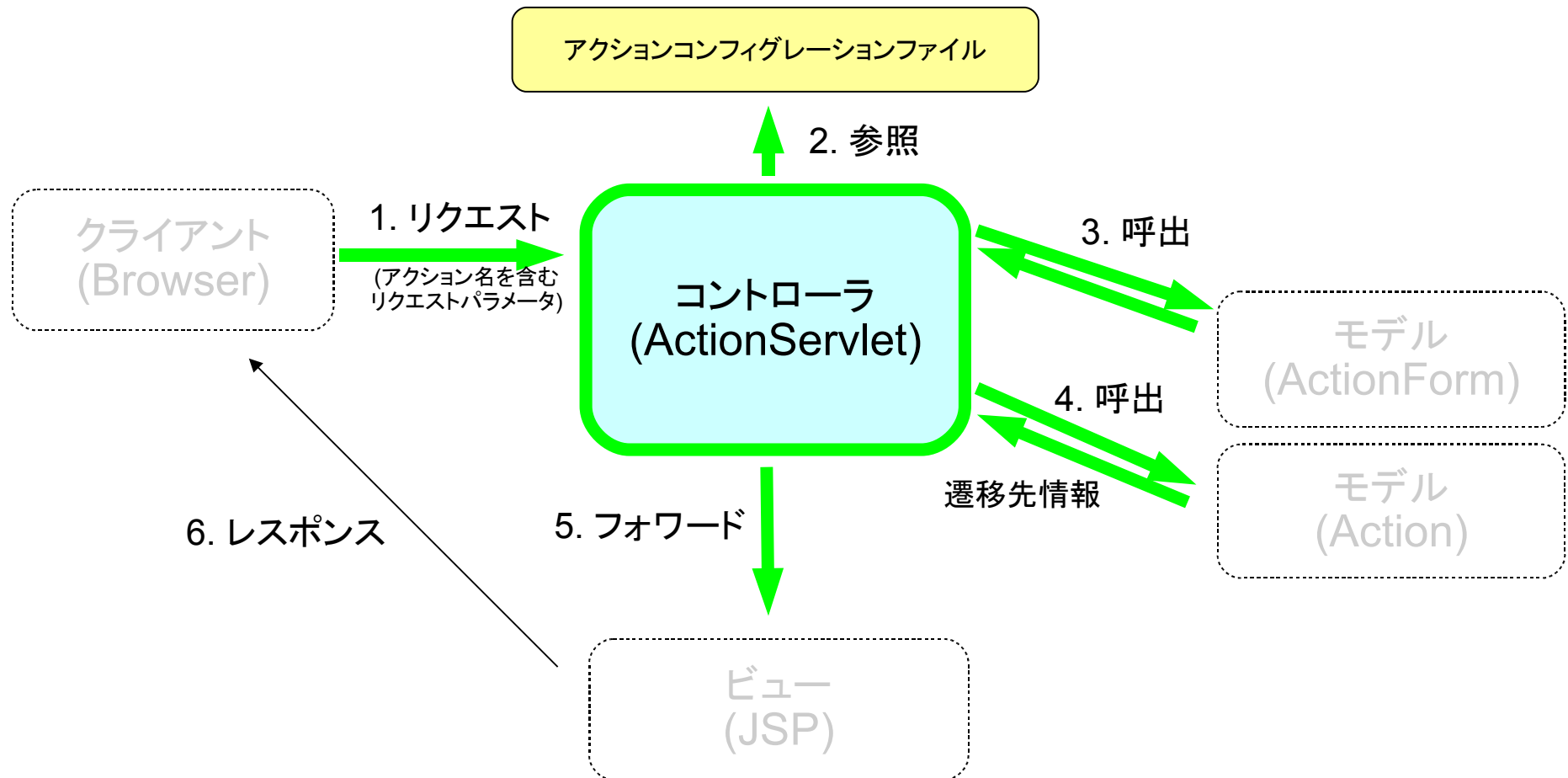
株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# ActionServlet

- ActionServletはMVCのコントローラに相当する
- アクションコンフィグレーションファイルの定義に基づいてリクエストを処理する
- Strutsフレームワークの中核となるコンポーネント
- 基底クラスは、  
`org.apache.struts.action.ActionServlet`
- 通常このクラスをそのままコントローラとして利用する

# ActionServletの役割

- リクエスト処理の流れをコントロールする





# ActionServletの動き

1. ActionServletがアクション名を含むリクエストを受け取る
2. ActionServletはアクション名をアクションコンフィギュレーションファイルに照合し必要なActionFormやActionを特定する
3. ActionServletは特定されたActionFormのメソッドを実行し、必要に応じて処理結果を受け取る
4. ActionServletは特定されたActionのメソッドを実行し、遷移先ページについての情報を受け取る
5. ActionServletは Actionの処理結果をもとに適切なJSPへ画面を遷移(フォワード)する

# ActionForm

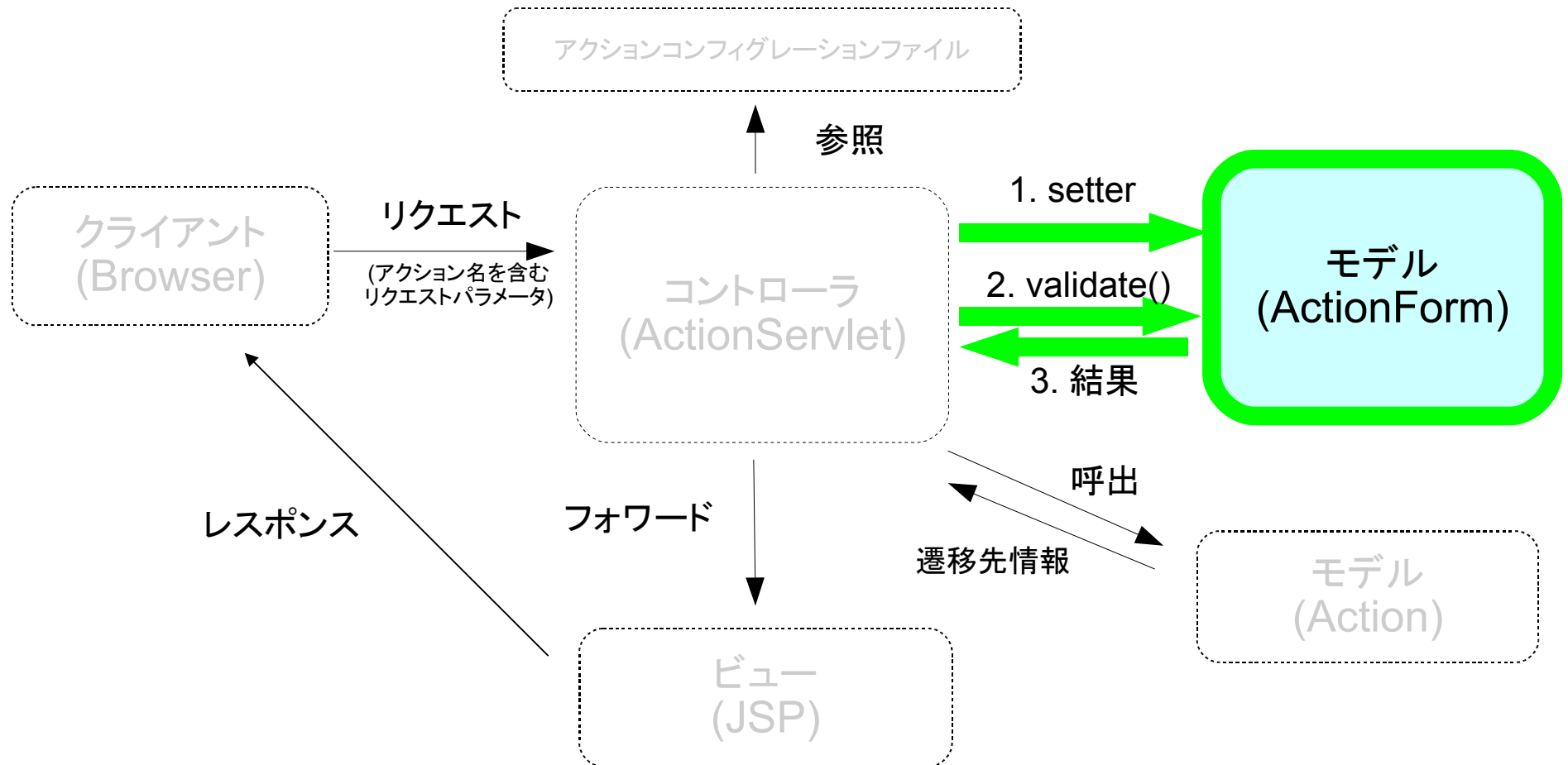
株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# ActionForm

- ActionFormはクライアントから送られるリクエストパラメータの値を自動的に保管したり、その値の妥当性をチェックしたりするコンポーネントである
- 基底クラスは、org.apache.struts.action.ActionForm
- 開発者は、この基底クラスを継承してアプリケーションに応じたサブクラスを作成して利用する
  - setter/getterを実装したり、validate()メソッドなどをオーバーライドする

# ActionFormの役割

- リクエストパラメータの格納と妥当性チェックを行う

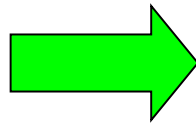
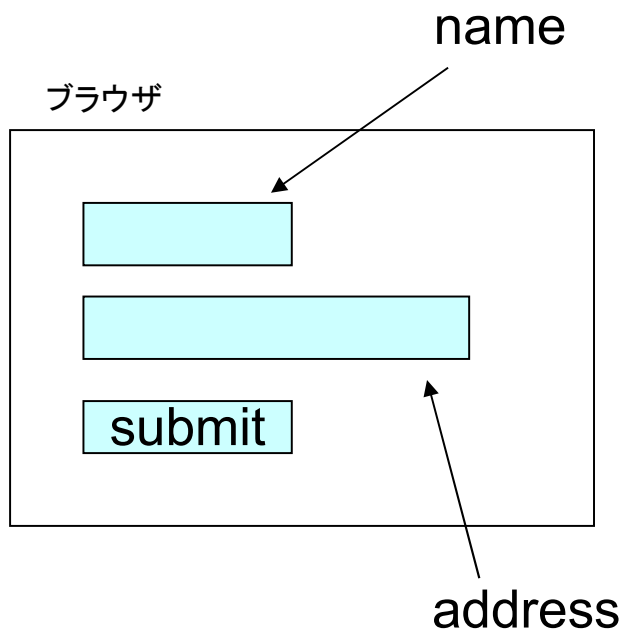


# ActionFormの動き

1. ActionServletが各リクエストパラメータに対応したsetterメソッドを実行し、値をActionFormに格納
2. validate()によって、リクエストパラメータの妥当性がチェックされ、結果がActionServletに返される

# ActionFormの例

- リクエストパラメータに対応したプロパティとsetter/getterおよびvalidate()メソッドを持つクラス



```
public class SampleForm
    extends ActionForm{

    private String name = null;
    private String address = null;

    public String getName() {...}
    public String getAddress() {...}

    public void setName(String n) {...}
    public void setAddress(String a) {...}
        :
    public ActionErrors validate(...) { ... }
}
```

# validate()メソッドの例

```
public ActionErrors validate (ActionMapping mapping,
                             HttpServletRequest request) {

    ActionErrors errors = new ActionErrors();

    if (name == null || name.equals("")) {
        errors.add("name", new ActionMessage("name.required"));
    }

    :
    :

    return errors;
}
```

name フィールドの  
必須入力チェック

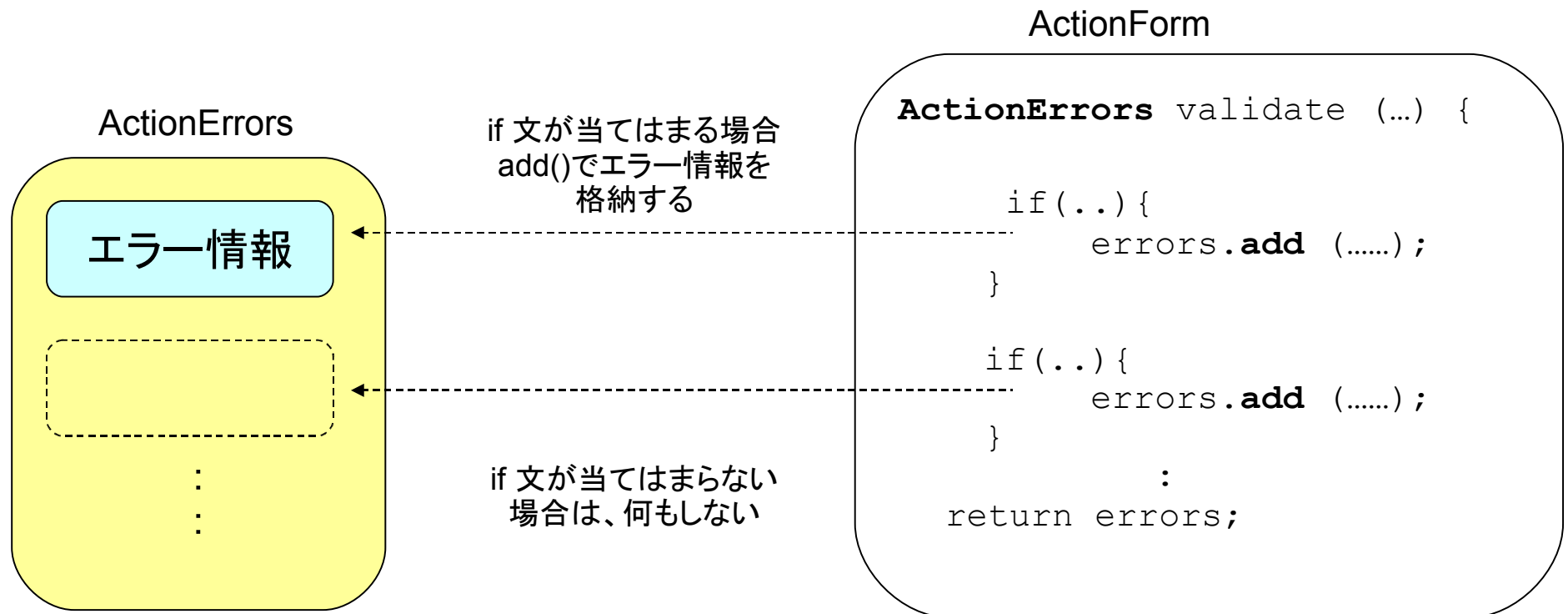
# validate()メソッドの例

- 戻り値はActionErrors型
- 好ましくないケースをチェックし、当てはまればエラーメッセージを生成
  - new ActionMessage()でメッセージを生成し、errors (ActionErrors型)にadd()メソッドで格納する
- 戻り値としてerrorsを返却
- ActionServletがerrorsを受け取る
  - もしerrorsが「空」なら、エラーなしと判断
  - ひとつ以上のActionMessageオブジェクトが格納されていれば、エラーありと判断する



# ActionErrors とは

- ActionErrors は、各エラーメッセージを保持するコンテナオブジェクト



# ActionErrors#add()メソッド

- 構文

```
add("プロパティ名", メッセージ)
```

- プロパティ名 : メッセージの識別名
- メッセージ : ActionMessageオブジェクトで指定

# ActionMessage とは

- ActionMessage はメッセージをメッセージリソースファイルから取得し保持するためのオブジェクト
- validate()では、不正なパラメータが発見された時、それに対応するエラーメッセージを取得するために使用される

## 構文

```
new ActionMessage ("キー文字列")
```

コンストラクタにキー文字列を与えることで、そのキー文字列に定義付けられているメッセージをメッセージリソースファイルから取得する

# メッセージリソースファイルとは

- アプリケーションで使用されるメッセージを設定したファイル
- Javaプロパティファイルと呼ばれ、拡張子は .properties
- [キー文字列=表示用メッセージ]という形式で作成されており、キー文字列を指定することで、表示用メッセージを取得できる

メッセージリソースファイル sample.properties

```
input.required=Please input a data  
input.invalid=Input data is invalid
```

↑  
キー文字列

↑  
表示用メッセージ

# メッセージリソースファイルと日本語

- メッセージリソースファイルに日本語を含める場合は、Unicodeエスケープされていなければならない
- JDKに付属しているnative2ascii コマンドで、日本語の文字をUnicodeエスケープされたものに変換することができる

```
native2ascii [変換前ファイル名] [変換後ファイル名]
```

コマンドプロンプト

# メッセージリソースファイルと日本語

sample.txt

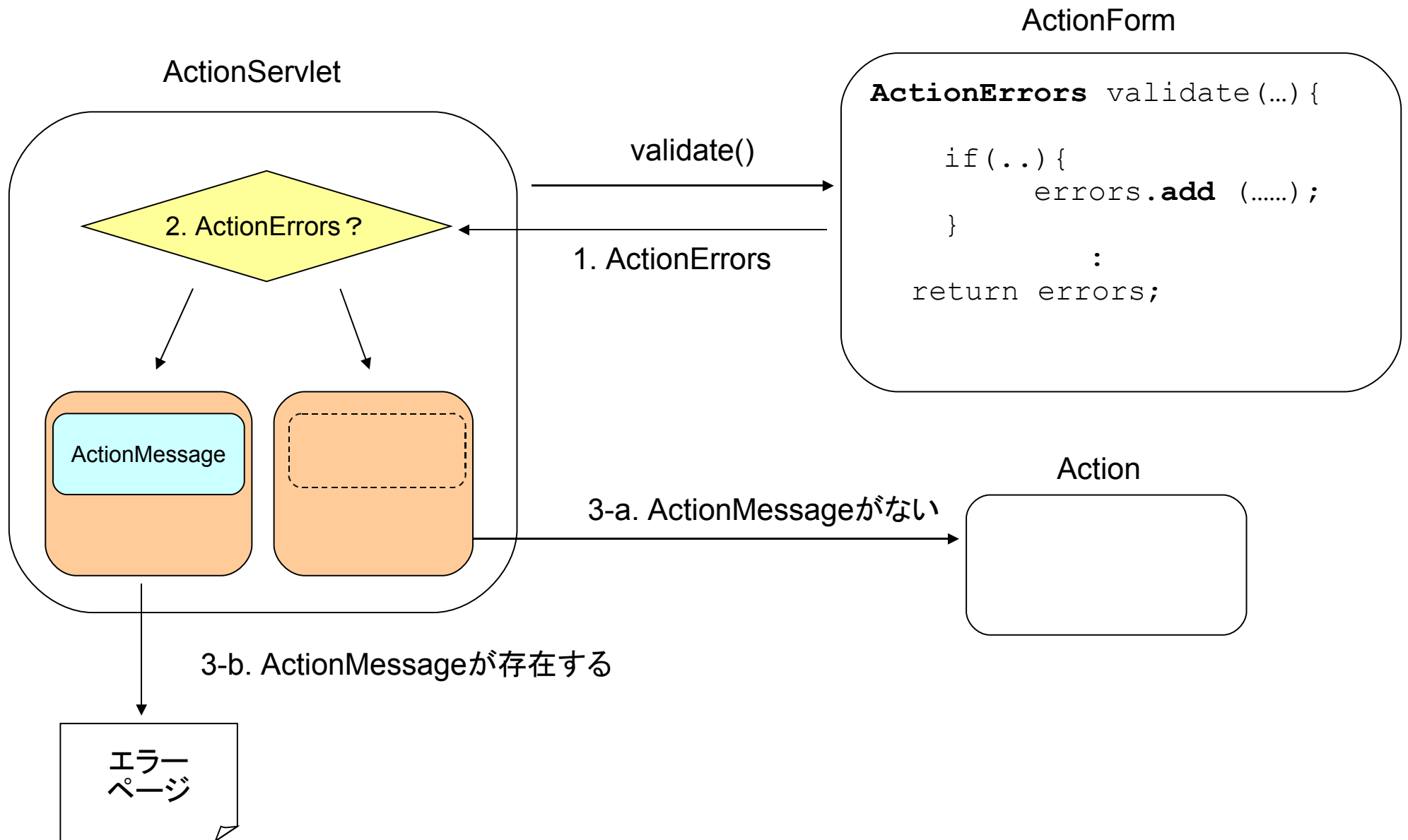
```
input.required=値を入力してください  
input.invalid=入力値が不正です  
:
```

```
native2ascii sample.txt sample.properties
```

sample.properties

```
input.required=\u383a\u388a\u245b...  
input.invalid=\u345a\u445b\u222c...  
:
```

# validate()結果による処理フロー



# validate()結果による処理フロー

1. validate()の結果としてActionErrorsオブジェクトがActionServletに返される
2. ActionErrorsの中身がチェックされる
- 3-a. ActionErrorsがActionMessageを含まない場合、パラメータは妥当であったと判断され、処理はActionへ進む
- 3-b. もしひとつ以上のActionMessageオブジェクトを含む場合は、エラー用のページに遷移しエラーメッセージを表示する



# Action

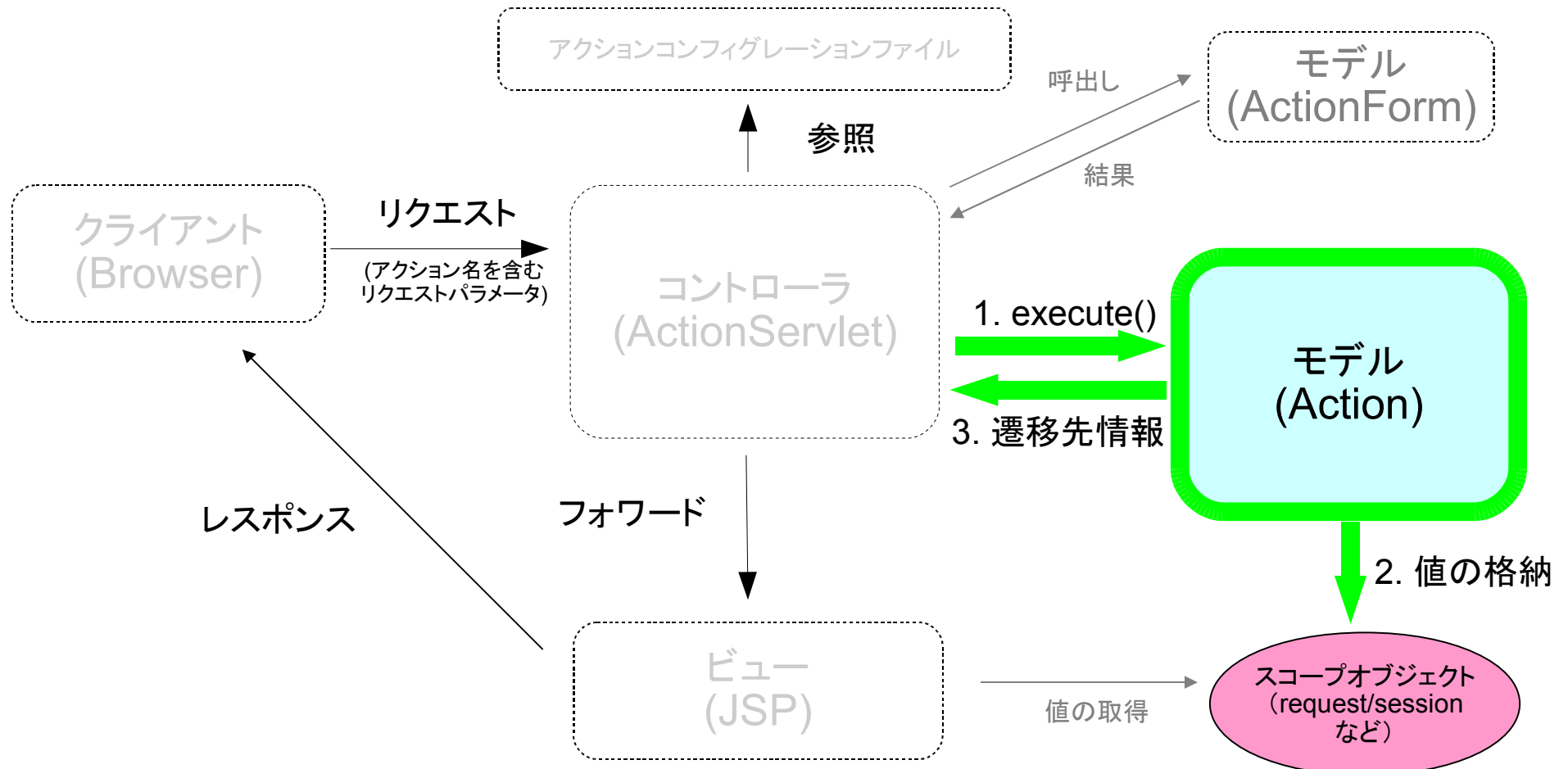
株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# Action

- Actionは、主にビジネスロジックを実行して、次の遷移先を設定するコンポーネントである
- 基底クラスは、`org.apache.struts.action.Action`
- 開発者は、この基底クラスを継承してアプリケーションに応じたサブクラスを作成して利用する
  - `execute()`メソッドをオーバーライドする

# Actionの役割

- ビジネスロジックを実行し次の遷移先を設定する



# Actionの動き

1. ActionServletがexecute()を呼び出す
2. execute()内でビジネスロジックなどを実行
3. 必要に応じて、処理結果をスコープに格納(JSPからも参照できるようにするため)
4. 遷移先情報(次に遷移すべき画面を指定)をActionServletに返却

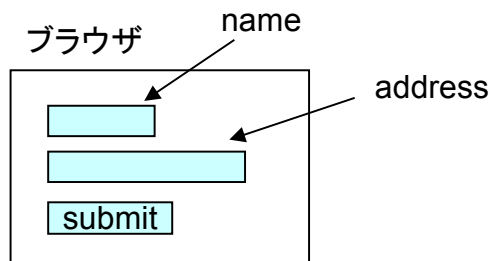
# スコープとは

- 「スコープ」とはデータを保持することのできる期間
- 保持できる期間ごとに4つのスコープがある
- データは各スコープに対応するオブジェクトに格納することができる

スコープ	オブジェクト	説明
page	PageContext	あるページ内でのみアクセス可能
request	HttpServletRequest	リクエストを受けてからレスポンスを返すまでの間、アクセス可能
session	HttpSession	セッションが継続している間、アクセス可能
application	ServletContext	アプリケーションが継続している間、アクセス可能

# Actionの例

- execute()メソッドの第2引数にはActionFormオブジェクトが渡される



```
class SampleForm
    extends ActionForm{
private String name = null;
private String address = null;
    :
```

```
public class SampleAction
    extends Action{

    public ActionForward execute (
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception {
        :
        return mapping.findForward("success");
    }
}
```

# execute()メソッドの例

```
public ActionForward execute (ActionMapping mapping,  
                             ActionForm form,  
                             HttpServletRequest request,  
                             HttpServletResponse response)  
                             throws Exception {  
  
    SampleForm sampleObj = (SampleForm) form;  
                           :  
    request.setAttribute("sample", sampleObj);  
  
    return mapping.findForward("success");  
}
```

SampleFormへのキャスト

requestスコープへの格納

ActionForwardオブジェクトを返す

findForward()メソッドの  
戻り値はActionForward

# execute()メソッドの例

- 戻り値はActionForward型
- 第2引数のActionFormに対してsetter/getterを利用したい場合は、そのクラスの実際の型(ここではSampleForm)にキャストする必要がある
- Actionでの実行結果をJSPから利用したい場合は、適切なスコープオブジェクト(requestやsession)に格納しておく
- 遷移先をActionMappingのfindForward()メソッドで設定して、ActionForwardオブジェクトを返却
- ActionServletが返却された遷移先情報からアクションコンフィギュレーションファイルを参照し、フォワードすべきJSPを特定する



# ActionMapping#findForward()メソッド

- 構文

```
mapping.findForward("キー文字列")
```

キー文字列をもとにアクションコンフィグレーションファイルから取得した遷移先情報をActionForwardオブジェクトに設定する

# setAttribute() メソッド

- 構文

```
スコープオブジェクト.setAttribute(登録名,オブジェクト);
```

- 登録名 : java.lang.String型
- オブジェクト : java.lang.Object型
  - スコープオブジェクト内に、登録名がバインドされたオブジェクトが格納される

例) `request.setAttribute("n1", name);`  
nameというオブジェクトを”n1”という名前でrequestスコープに登録する

# getAttribute() メソッド

- 構文

```
スコープオブジェクト.getAttribute(登録名);
```

- 登録名 : java.lang.String型
  - 登録名にバインドされたオブジェクトを取得する

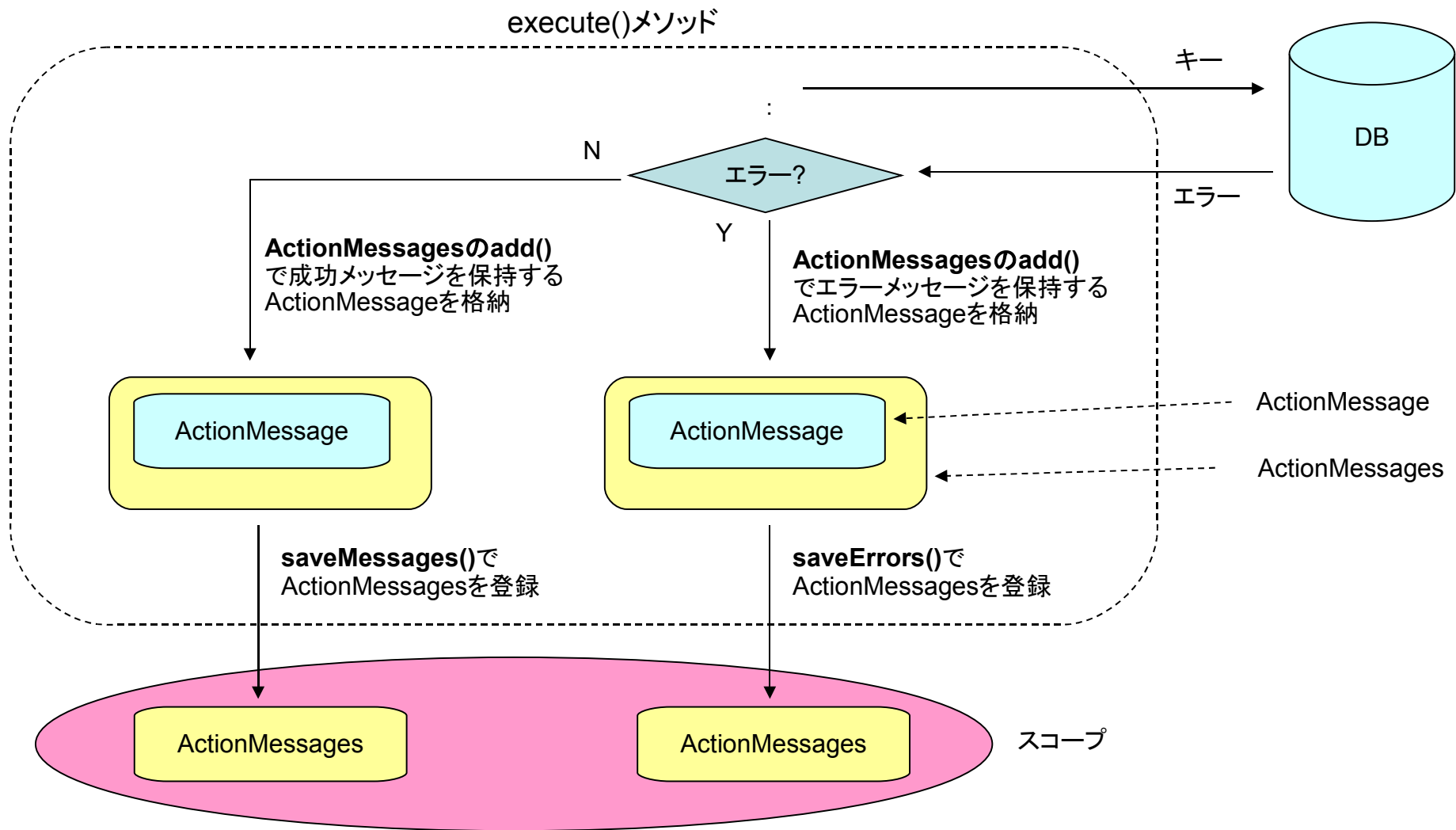
例) `request.getAttribute("n1");`

"n1"を指定することで、requestスコープに"n1"という名前で登録されているオブジェクトを取得できる

# Actionでのメッセージ設定

- ActionFormのvalidate()メソッドでは、ActionErrorsのadd()メソッドを使ってエラーメッセージを設定できた
- Actionのexecute()ではActionMessagesのadd()メソッドを使って適切なメッセージを設定した後に
  - エラーでないメッセージはsaveMessages()メソッドでスコープに登録する
  - エラーメッセージはsaveErrors()メソッドでスコープに登録する

# Actionでのメッセージ設定



# Actionでのメッセージ設定

```
public ActionForward execute (ActionMapping mapping, ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {
    ActionForward forward = new ActionForward();
    SampleForm sForm = (SampleForm)form;
    String id = sForm.getId();
    DBSearch search = new DBSearch();
    String id_found = search.findById(id);

    ActionMessages msgs = new ActionMessages();

    if (id_found != null) {
        msgs.add("id", new ActionMessage("id.find.success"));
        saveMessages(request, msgs);
        forward = mapping.findForward("success");
    }else{
        msgs.add("id", new ActionMessage("id.find.error"));
        saveErrors(request, msgs);
        forward = mapping.findForward("error");
    }
    return forward;
}
```

SampleFormから取得した id をキーにして、DBから id がマッチするデータを取得 (見つからなければnullを返す)

ActionMessagesの生成

データを取得できたかチェック

add()メソッドで成功メッセージを格納

saveMessages()でスコープに登録

add()メソッドでエラーメッセージを格納

saveErrors()でスコープに登録

# ActionMessages とは

- メッセージを保持するコンテナオブジェクト
- – add()メソッドを使ってメッセージを追加する
  - add(“プロパティ名”, メッセージ)
  - メッセージにはActionMessageオブジェクトを指定
- ActionのsaveMessages()メソッドやsaveErrors()メソッドの第2引数に指定し格納する

# Action#saveMessages() メソッド

## Action#saveErrors()メソッド

- 構文

```
saveMessages(スコープ, メッセージリスト);  
saveErrors(スコープ, メッセージリスト);
```

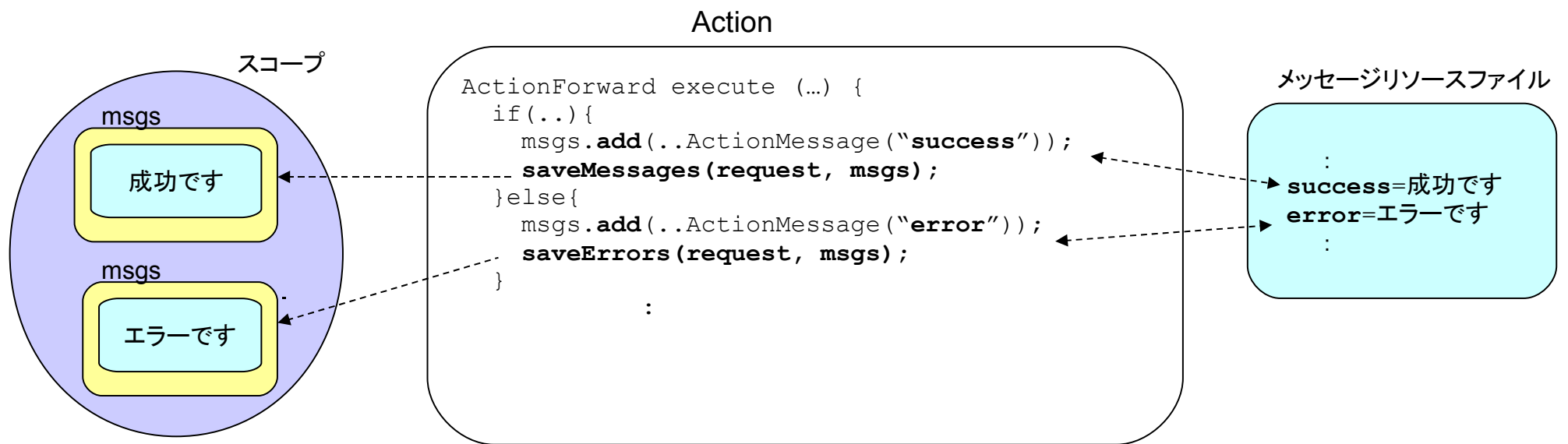
- スコープ : request か session
- メッセージリスト : ActionMessagesオブジェクト
  - add()メソッドによって追加されたActionMessageオブジェクトを保持している

※ saveMessages()メソッド : エラー以外のメッセージをスコープに登録する

※ saveErrors()メソッド : エラーメッセージをスコープに登録する



# メッセージの流れ



# JSP

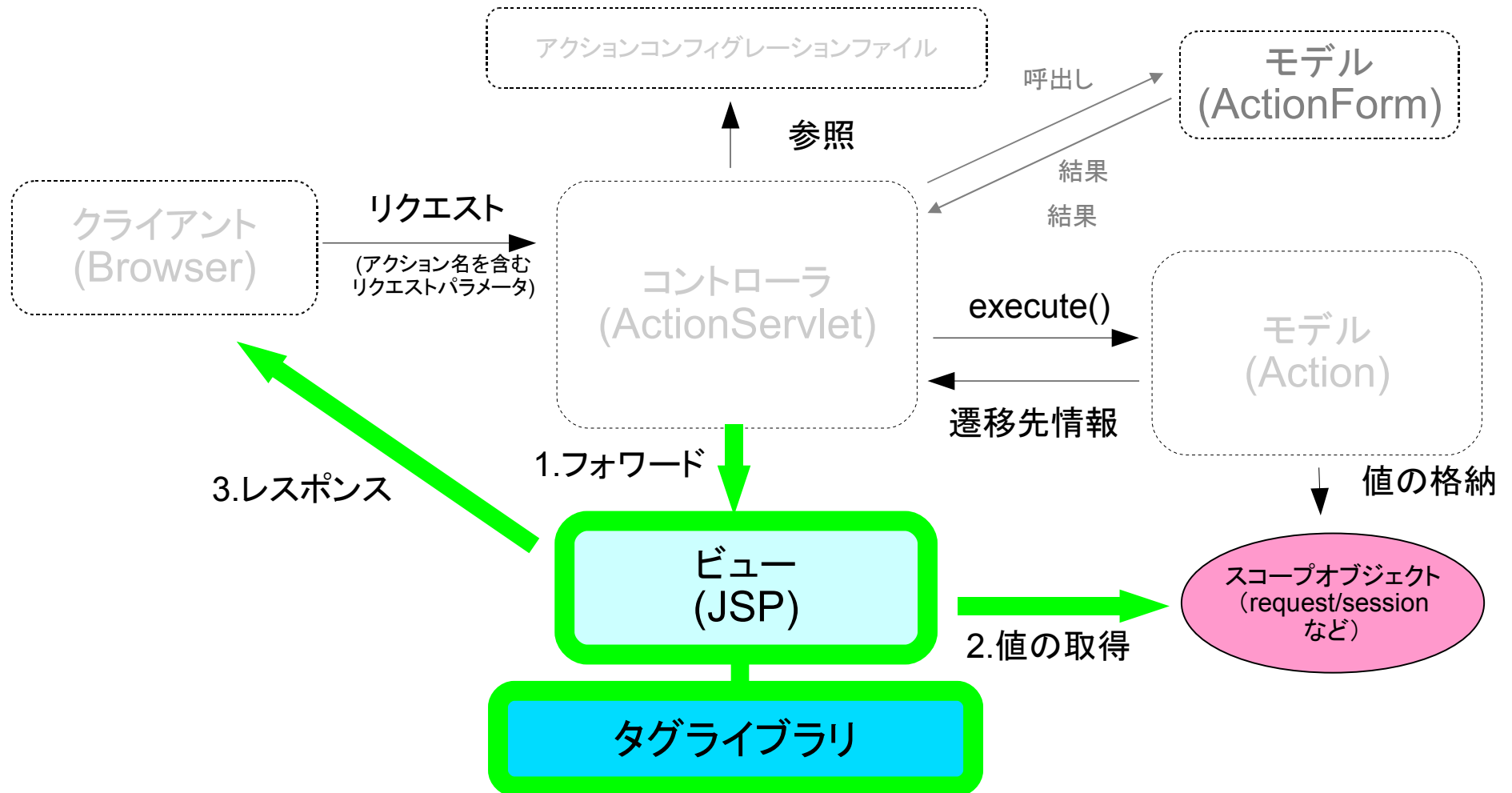
株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# JSP

- ページ生成に用いられるコンポーネント
- Strutsでは専用タグライブラリ等を利用してページを効率的に作成することができる
  - ScriptletやExpressionによるJavaコードの使用を抑え、HTMLタグと類似のタグのみでコードを記述することで可読性を高めることができる

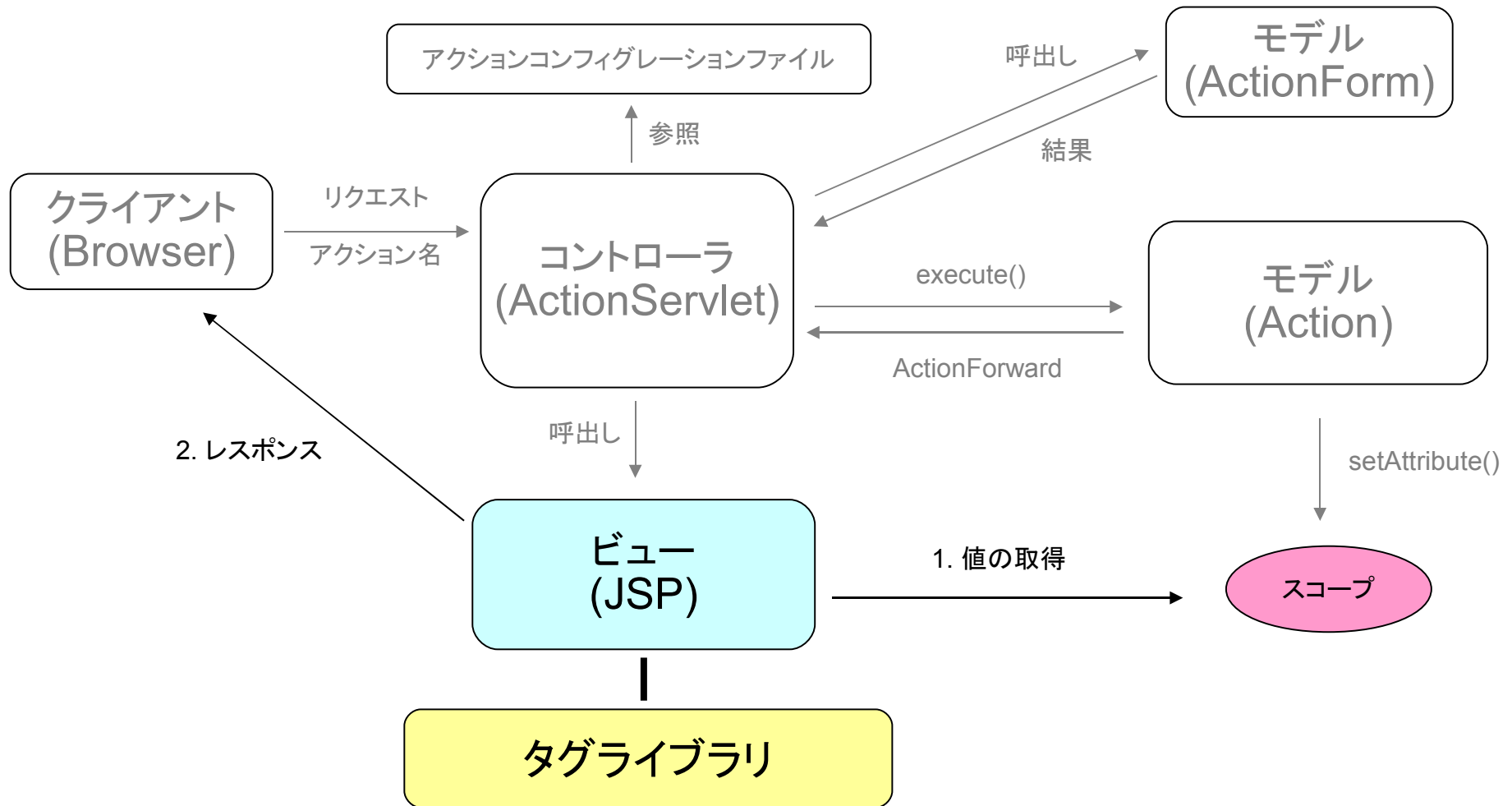
# JSPの役割

- レスポンスを生成する



# JSPの役割

- レスポンスページを生成する



# JSPの動き

1. ActionServletからリクエスト・レスポンスがJSPにフォワードされる
2. JSPは必要に応じてタグライブラリを利用しながらスコープに登録されているオブジェクトにアクセスし表示すべきデータを取得する
3. レスポンスページを生成しブラウザに表示する

# Strutsの主なタグライブラリ

- Bean タグライブラリ
  - JavaBeanのプロパティのデータを操作したりメッセージを表示するためのライブラリ
- HTML タグライブラリ
  - フォームを作成するタグなど、HTMLタグの機能をStruts専用タグで代替したライブラリ
- Logic Tags ライブラリ
  - 条件分岐、繰り返し、比較などプログラムを制御するためのライブラリ

# taglibディレクティブ

- タグライブラリを使用するにはJSPファイルの先頭に taglibディレクティブの宣言が必要

## JSPサンプル

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>  
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>  
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
```

属性	説明
uri	tldファイルの配置場所を指定する (※ tldファイルはタグとタグ実装クラスの対応を定義したファイル)
prefix	プレフィックスとして使用する名称を指定する



# タグの共通構文

## 構文

```
<プレフィックス:タグ名 属性=" " [属性=" "….] />
```

プレフィックス:タグライブラリ毎に決められたものを使う

例) <bean:write name="emp" .... / >



プレフィックス

タグライブラリ名	プレフィックス
Bean タグライブラリ	bean
HTML タグライブラリ	html
Logic タグライブラリ	logic

# タグの閉じ方

パターン1

<プレフィックス:タグ 属性=" " ... />

タグがそれ自身で完結している場合の閉じ方

パターン2

<プレフィックス:タグ1 属性=" " ... >  
<プレフィックス:タグ2 属性=" " ... />  
</プレフィックス:タグ1>

※ 閉じていないことに注意

タグが内部にボディ(子要素)を持つ場合の閉じ方

- ※ タグ内部に文字列がある場合もボディとなるのでパターン2で閉じる
- 例) <プレフィックス:タグ1 属性=" " ... >  
Hello Struts !  
</プレフィックス:タグ1>

# Bean タグライブラリ

## 主なタグ

- `<bean:define>`
  - 識別名の宣言と登録
- `<bean:write>`
  - 値の取得と表示
- `<bean:message>`
  - メッセージの取得と表示

# <bean:define> タグ

## <bean:define> タグの主な属性

属性	説明
id	• 識別名を宣言する
name	• スコープ内の識別名を指定する
property	• スコープ内の識別名にバインドされたオブジェクトのプロパティを指定する
scope	• name属性で指定された識別名を検索するスコープを指定する。省略するとpage-request-session-applicationの順で検索される
toScope	• 識別名をどのスコープに登録するかを指定する。規定値はpageスコープ
value	• 登録する値 (String型)を指定する

# <bean:define> タグ

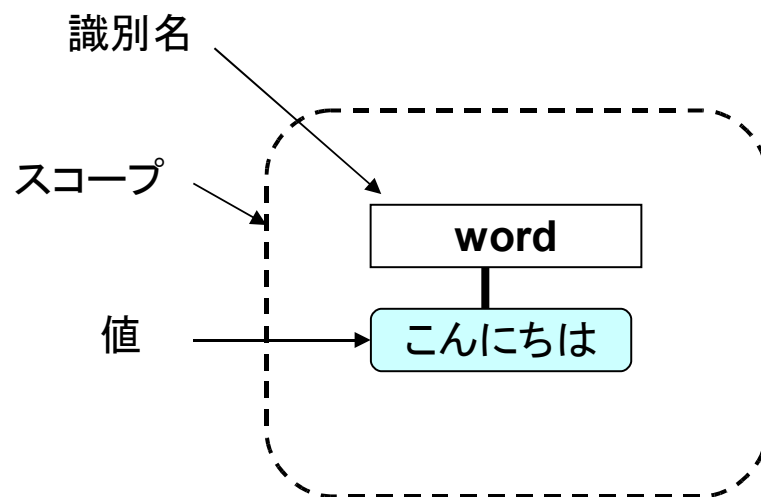
- 識別名を宣言し、value属性でString型の値をバインドして適当なスコープに登録する（スコープの規定値はpageスコープ）

JSPサンプル

識別名

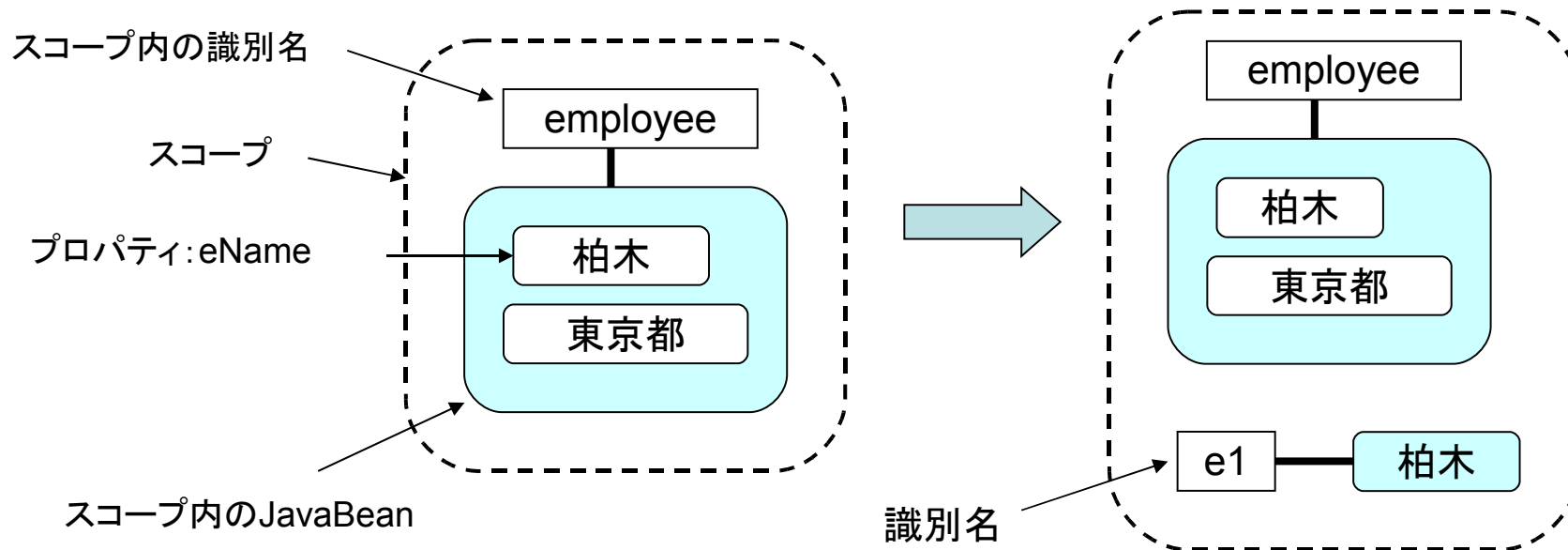
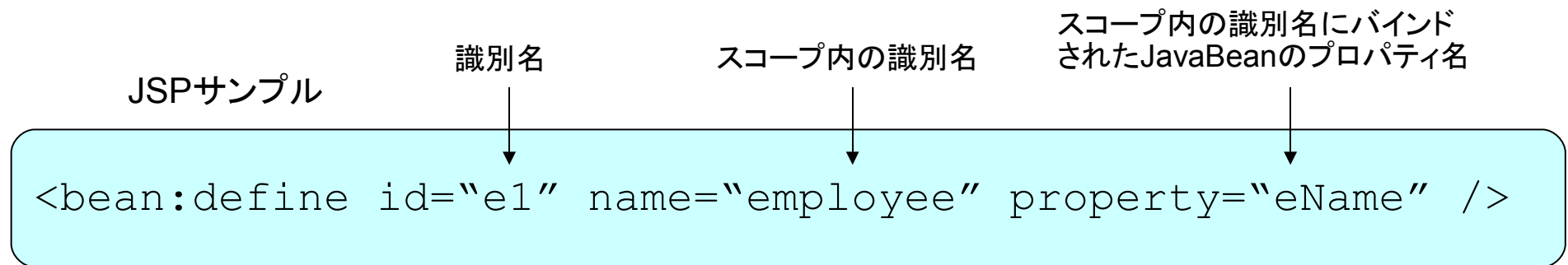
値

```
<bean:define id="word" value="こんにちは" />
```



# <bean:define> タグ

- スコープ内のJavaBeanにアクセスし、プロパティの値を識別名にバインドして、新たに登録する



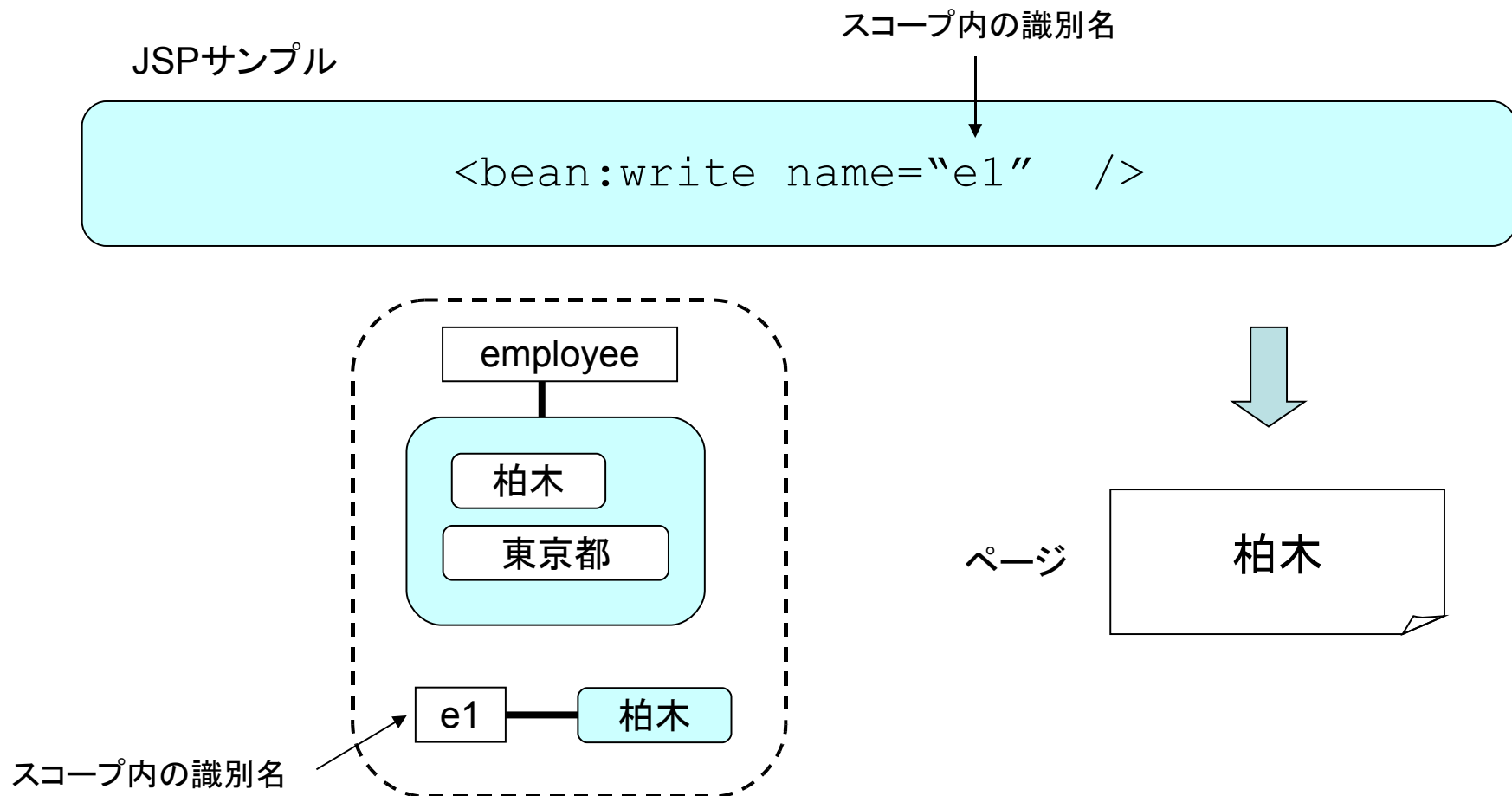
# <bean:write> タグ

## <bean:write> タグの主な属性

属性	説明
name	• スコープ内の識別名を指定する
property	• スコープ内の識別名にバインドされたJavaBeanのプロパティを指定する

# <bean:write> タグ

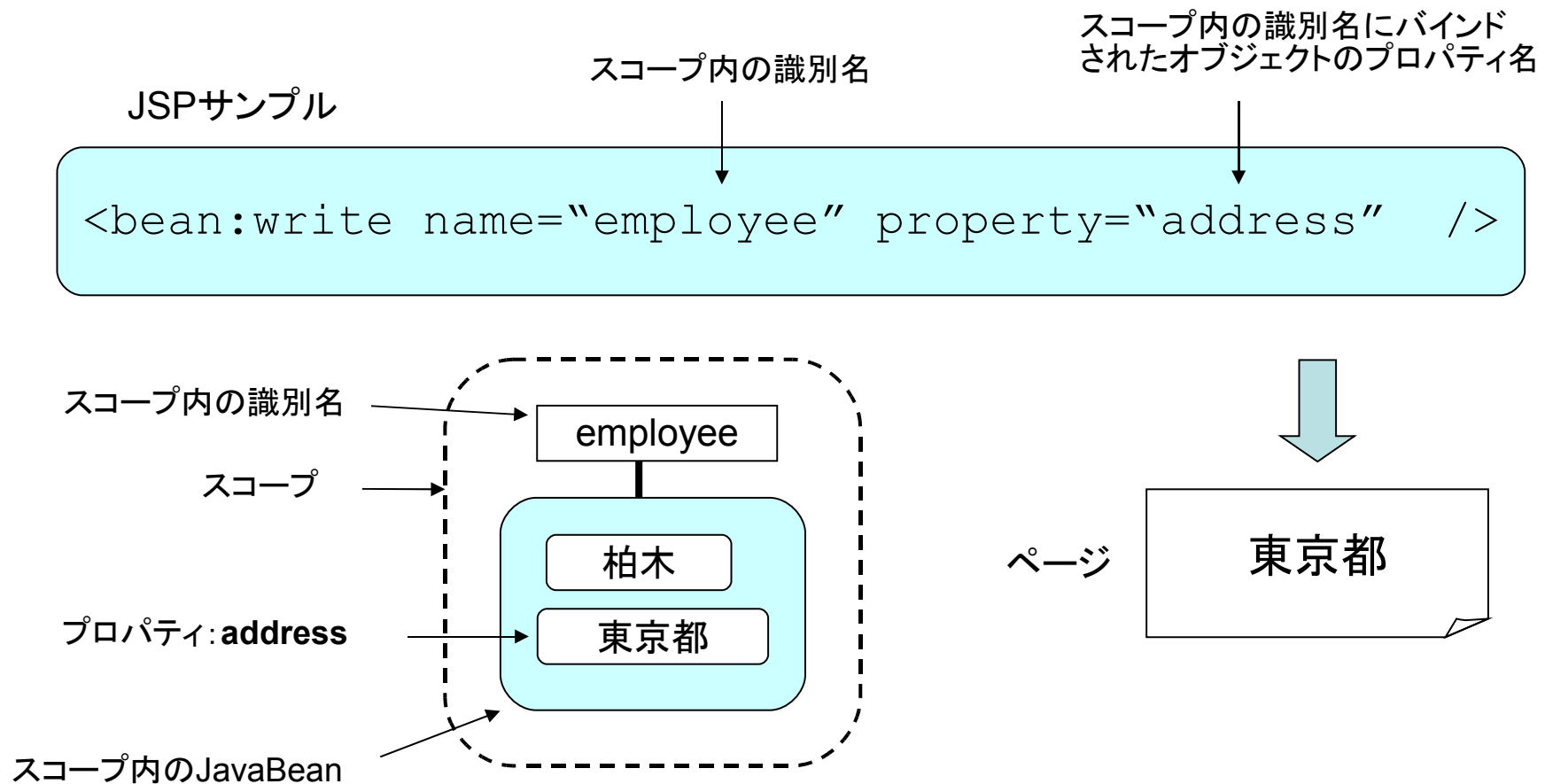
- すでにスコープに登録されている識別名を指定してバインドされている値を表示する





# <bean:write> タグ

- すでにスコープに登録されている識別名を指定し、バインドされているオブジェクトのプロパティ名を指定してその値を表示する



# <bean:message> タグ

## <bean:message> タグの主な属性

属性	説明
key	<ul style="list-style-type: none"><li>キー文字列を指定する (キー文字列はメッセージリソースファイルに定義してあること)</li></ul>

# <bean:message> タグ

- キー文字列を指定し、対応するメッセージをメッセージリソースファイルから取得して表示する

JSPサンプル

キー文字列

```
<bean:message key="input.required" />
```

メッセージリソースファイル

```
:  
input.required=Please input a data  
input.invalid=Input data is invalid  
:
```

ページ

Please input a data

# HTML タグライブラリ

## 主なタグ

- `<html:form>`
  - フォームの定義
- `<html:text>`
  - テキストボックスの定義
- `<html:submit>`
  - submitボタンの定義
- `<html:link>`
  - リンクの定義
- `<html:messages>`
  - メッセージの表示
- `<html:errors>`
  - エラーメッセージの表示

# <html:form>タグ

## <html:form>タグの主な属性

属性	説明
action	• このフォームをsubmitする時に送信されるアクション名を指定する
method	• フォームをsubmitする時に使用するHTTPメソッドのタイプ、GETかPOSTを指定する(規定値はPOST)

# <html:form>タグ

- フォームを定義し、内部にテキスト要素やsubmit ボタンなどを配置することができる
- submit される時、action属性で指定されたアクション名が送信され、それを元に適切なActionFormやActionが特定される

JSPサンプル

```
<html:form action="/input.do" >  
    :  
</html:form>
```

# <html:text>タグ

## <html:text>タグの主な属性

属性	説明
name	• スコープ内のJavaBean識別名を指定する
property	• フォームがsubmitされる時、送信されるテキスト要素の識別名を指定する

# <html:text>タグ

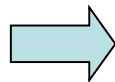
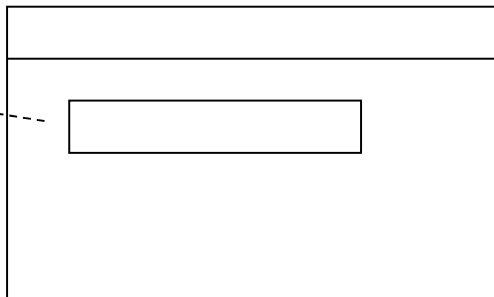
- フォームの内部にテキスト要素を定義する(フォームの内部に記述しなければならない)
- property属性で指定した識別名とActionFormのsetterが関連づけられる

JSPサンプル

```
<html:form action="/input.do" >
  <html:text property="id" />
  :
</html:form>
```

ブラウザ

id



ActionForm

```
SampleForm extends ActionForm{
  private String id = null;
  :
  public void setId(String id){
    this.id = id,
  }
  :
```

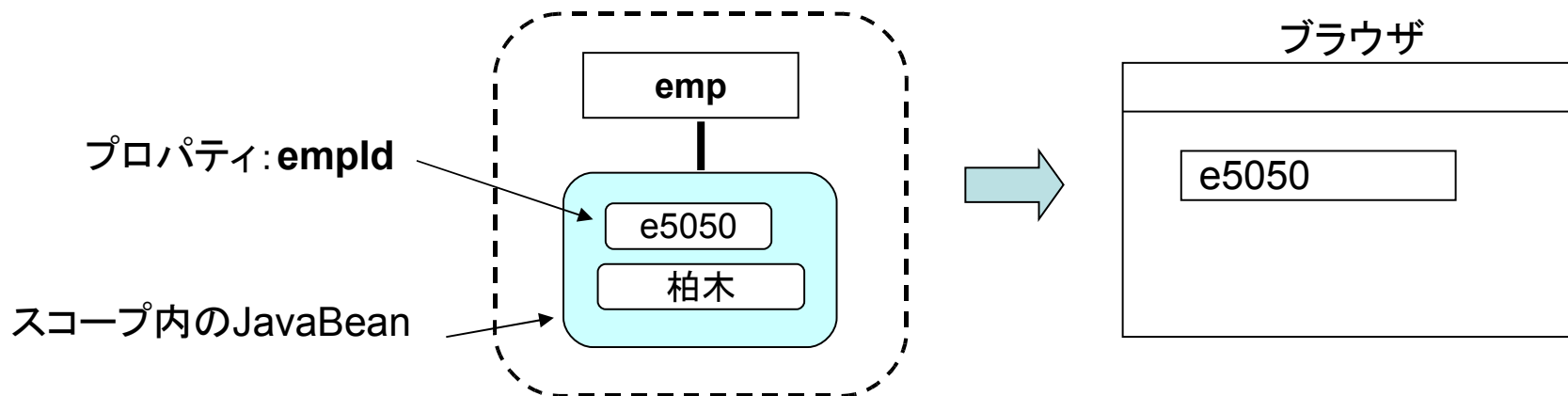


# <html:text>タグ

- name属性でスコープ内のJavaBean識別名を指定する(例ではemp)
  - この場合のproperty属性は、name属性で指定されたJavaBeanのプロパティ名を指している(例ではempId)
  - テキストには、初期値として、empIdの値が表示される

JSPサンプル

```
<html:form action="/input.do" >
  <html:text name="emp" property="empId" />
  :
</html:form>
```



# <html:submit>タグ

## <html:submit>タグの主な属性

属性	説明
property	• フォームがsubmit される時、送信されるsubmit ボタン情報の識別名を指定する
value	• submit ボタンのラベル名を指定する

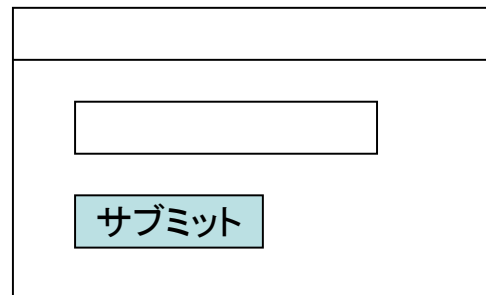
# <html:submit>タグ

- フォームの内部にsubmit ボタンを定義する(フォームの内部に記述しなければならない)

JSPサンプル

```
<html:form action="/input.do" >  
    :  
    <html:submit value="サブミット" />  
</html:form>
```

ブラウザ



The diagram shows a browser window with a form. The form contains a text input field and a button labeled "サブミット" (Submit).

# <html:link>タグ

## <html:link>タグの主な属性

属性	説明
action / page / forward	• アクション名かJSPファイル名か<global-forwards>のforward名を指定してリンク先を指定する
paramId	• リンク先へ送る情報の識別名を指定する
paramName	• スコープ内の識別名を指定する
paramProperty	• paramName属性で指定された識別名にバインドされた値がJavaBeanの場合、そのプロパティ名を指定して、paramIdにバインドする値を取得する

# <html:link>タグ

- action属性: リンクをクリックすると、アクション名nextで定義されているアクションが実行され、遷移先JSPへ移行し、実行結果が表示される

JSPサンプル

```
<html:link action="/next.do">リンク</html:link>
```

```
<action path="/next"  
  :  
  <forward... path="nextA.jsp">  
</action>
```

nextA.jsp

```
<html>  
<body>  
  こんにちは!  
  :
```

こんにちは!

# <html:link>タグ

- page属性: リンクをクリックすると、page属性で指定されているJSPへ移行し、実行結果を画面に表示する

JSPサンプル

```
<html:link page="/nextB.jsp" >リンク</html:link>
```

nextB.jsp

```
<html>  
<body>  
さよなら!  
:
```

さよなら!

# <html:link>タグ

- forward属性: リンクをクリックすると、アクションコンフィグレーションの<global-forwards>を参照し、一致した名前前で定義されているJSPへ移行し、実行結果を画面に表示する

JSPサンプル

```
<html:link forward="menu">リンク</html:link>
```

```
<global-forwards>  
  <forward name="menu"  
    path="nextC.jsp" />  
  :  
</global-forwards>  
  :
```

nextC.jsp

```
<html>  
<body>  
  またね!  
  :
```

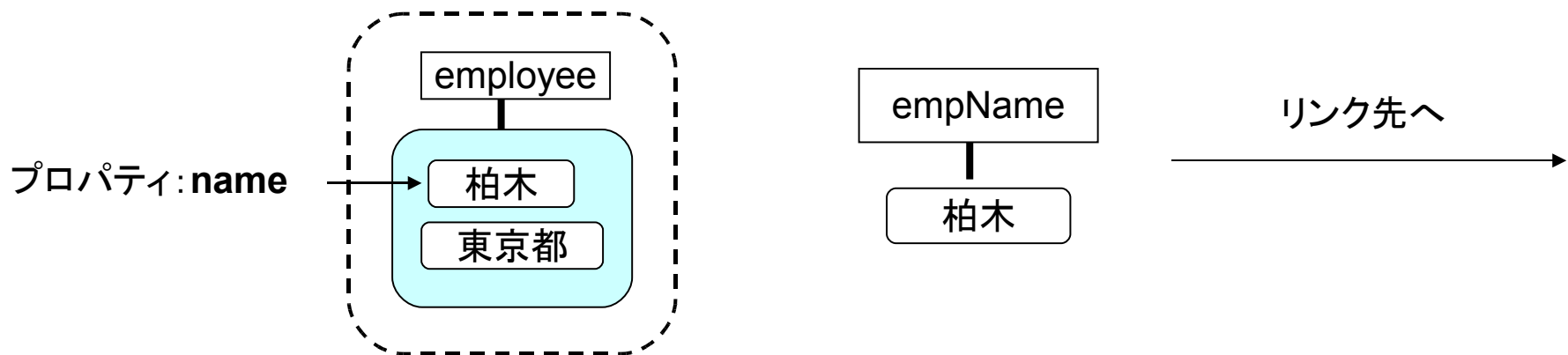
またね!

# <html:link>タグ

- プロパティnameに格納されている「柏木」という値が empName という識別名にバインドされて付属情報としてリンク先へ送られる

JSPサンプル

```
<html:link action="/next.do"
           paramId="empName"
           paramName="employee"
           paramProperty="name" >リンク</html:link>
```





# <html:messages>タグ

## <html:messages>タグの主な属性

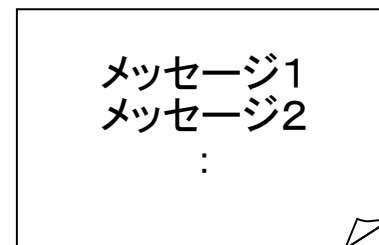
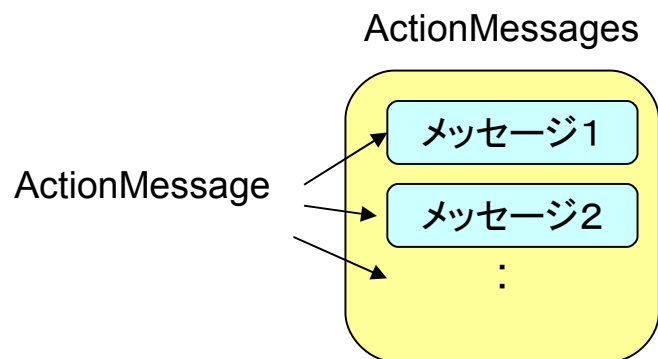
属性	説明
id	<ul style="list-style-type: none"><li>• &lt;html:messages&gt;タグ内で使用する識別名(任意)を指定する</li><li>• ActionMessagesオブジェクトに保持されているすべてのメッセージは繰り返して取り出され、そのつど、この識別名に代入される</li></ul>
message	<ul style="list-style-type: none"><li>• 表示対象となるメッセージの種類をtrue/falseで指定する true: saveMessages()メソッドで格納された通常メッセージ false: saveErrors()メソッドで格納されたエラーメッセージ</li></ul>
property	<ul style="list-style-type: none"><li>• ActionMessagesのadd()メソッドの第1引数で指定したプロパティ名を指定することで指定したメッセージだけを表示できる</li></ul>

# <html:messages>タグ

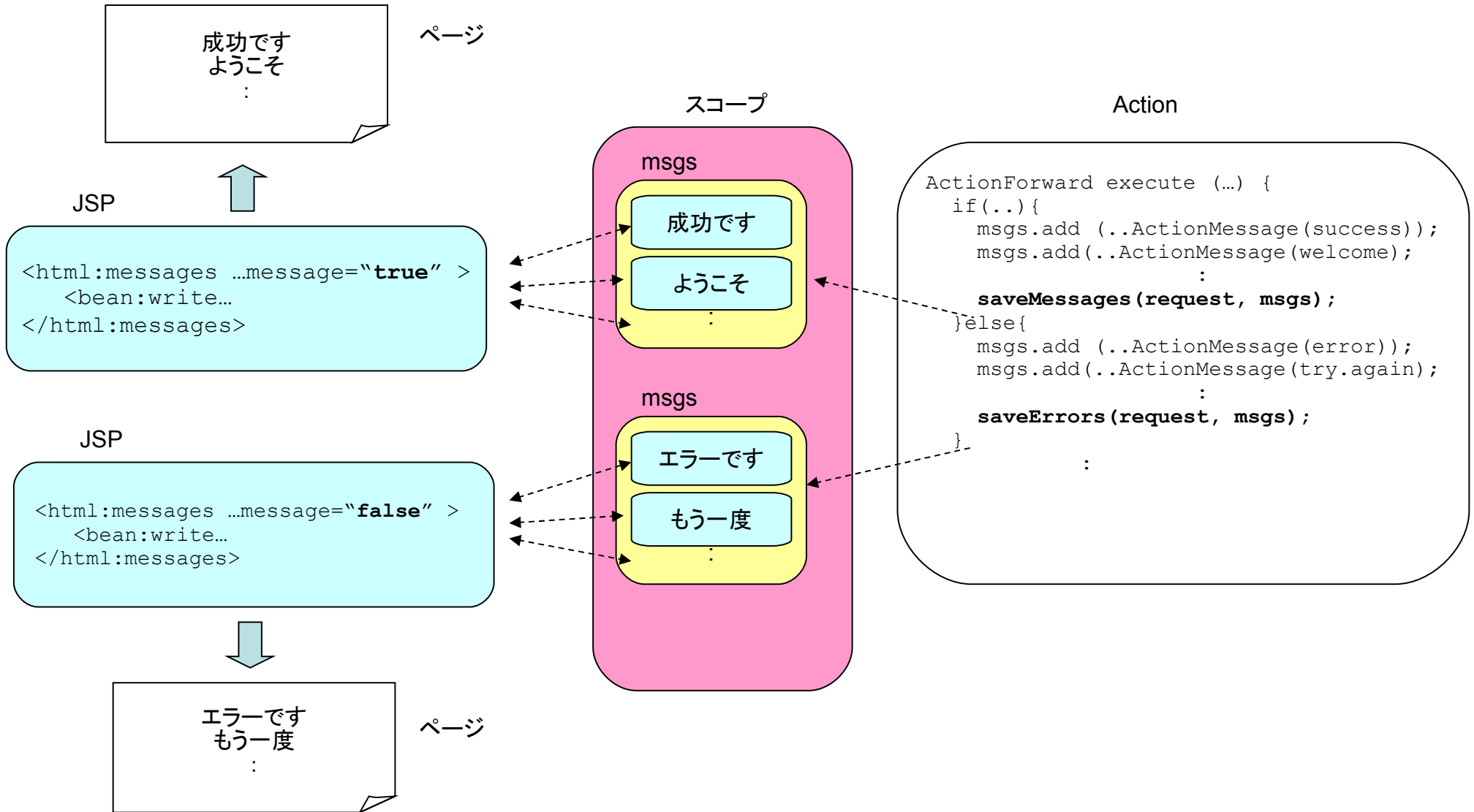
- ActionMessages オブジェクトに保持されている最初のメッセージをmsgという識別名に代入する
- <bean:write>タグで識別名msgを指定し、その値を表示する
- 保持されているすべてのメッセージに対してこの処理が繰り返される

JSPサンプル

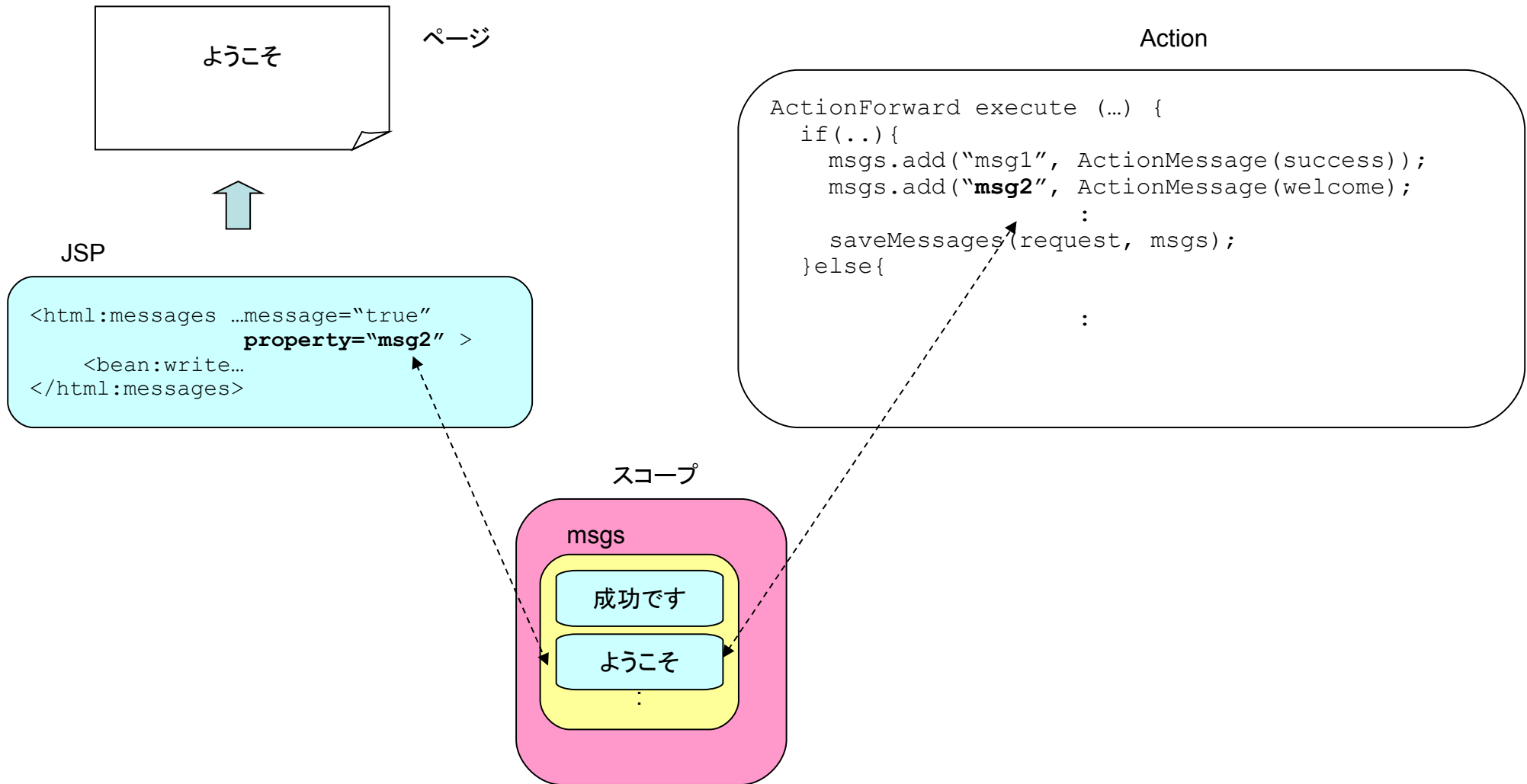
```
<html:messages id="msg" message="true">  
  <bean:write name="msg" />  
</html:messages>
```



# メッセージの流れ



# メッセージの特定

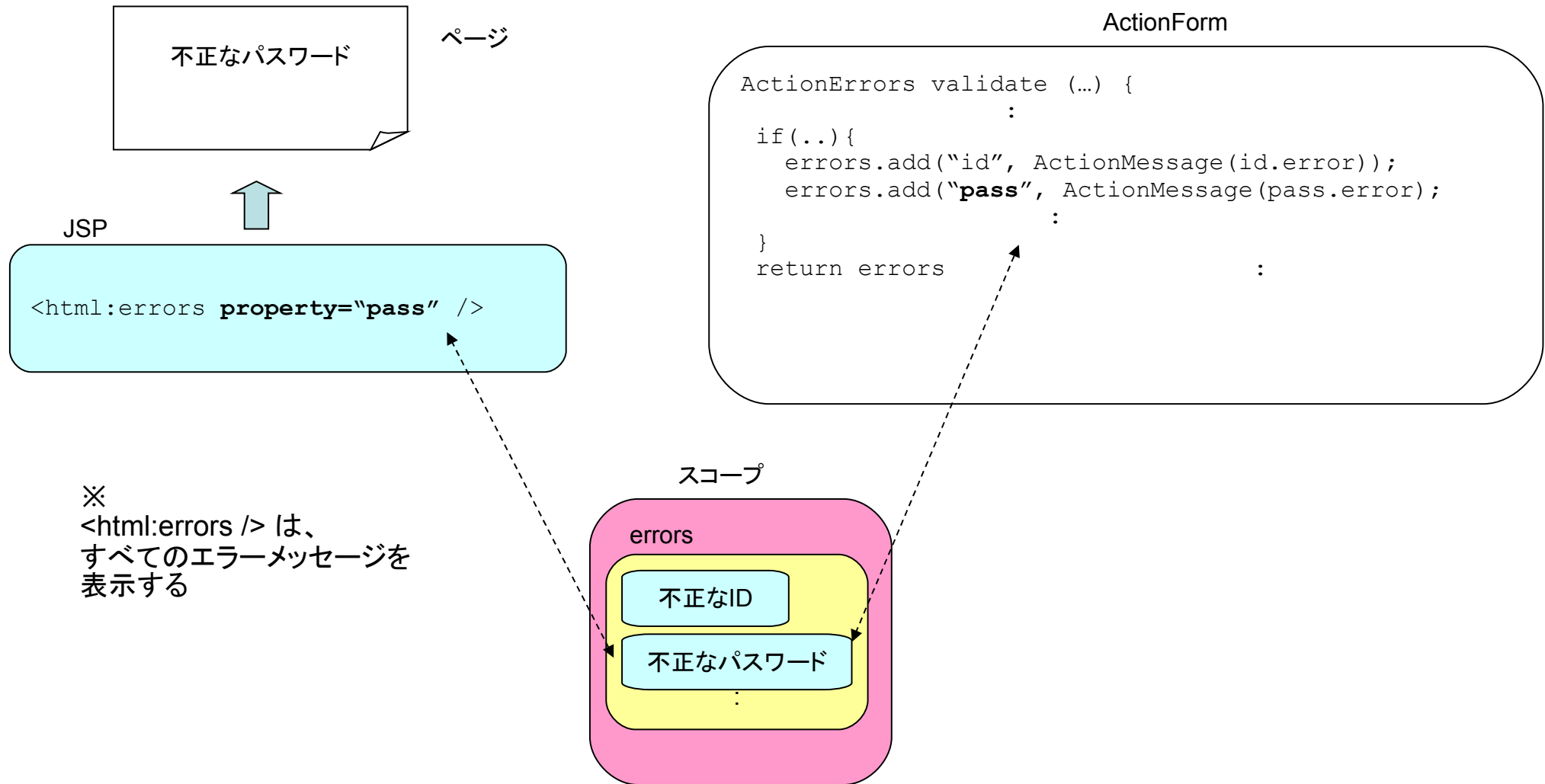


# <html:errors>タグ

## <html:errors>タグの主な属性

属性	説明
property	<ul style="list-style-type: none"><li>• ActionErrorsのadd()メソッドの第1引数で指定したプロパティ名を指定することで指定したメッセージだけを表示できる</li></ul>

# メッセージの特定



※  
<html:errors /> は、  
すべてのエラーメッセージを  
表示する

# まとめ : ActionMessagesと<html:messages>

- Actionのexecute()メソッドでメッセージを設定し表示する
  - ActionMessageでメッセージを取得
  - ActionMessagesのadd()メソッドでメッセージを格納
  - saveMessages()メソッド、saveErrors()メソッドでスコープに登録
  - <html:messages>タグで表示

execute()メソッド

```
msgs.add("id", ActionMessage("key"));  
saveMessages(request, msgs) / saveErrors(request, msgs);
```

※  
saveErrors()メソッドで登録されたエラーメッセージは  
<html:errors />でも表示することができる

JSP

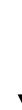
```
<html:messages id="msg" message="true/false" >  
  <bean:write name="msg" />  
</html:messages>
```

# まとめ : ActionErrorsと<html:errors>

- ActionFormのvalidate()メソッドでエラーメッセージを設定し表示する
  - ActionMessageでメッセージを取得
  - ActionErrorsのadd()メソッドでメッセージを格納
  - ActionErrorsを返す
  - <html:errors>タグで表示

validate()メソッド

```
errors.add("id", ActionMessage("key"));  
return errors;
```



JSP

```
<html:errors />
```



# その他のHTML タグ

- <html:html>
- <html:password>
- <html:textarea>
- <html:reset>
- <html:cancel>
- <html:hidden>
- <html:checkbox>
- <html:multibox>
- <html:frame>
- <html:radio>
- <html:select>
- <html:option>
- <html:options>
- <html:img>
- など

# Logic タグライブラリ

## 主なタグ

- `<logic:iterate>`
  - 反復制御を行う

# <logic:iterate>タグ

## <logic:iterate>タグの主な属性

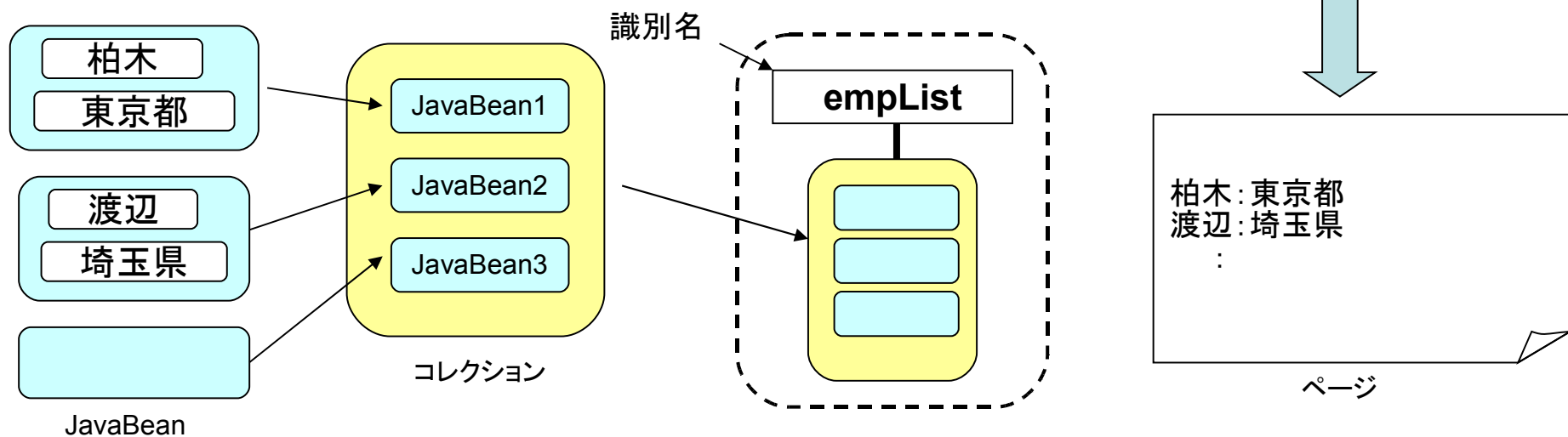
属性	説明
id	<ul style="list-style-type: none"><li>• 反復中に使用する識別名(任意)を指定する</li><li>• この識別名は、反復ごとにコレクションや配列から取り出される要素を保持する<ul style="list-style-type: none"><li>※コレクションとはArrayListのようなオブジェクト</li></ul></li></ul>
name	<ul style="list-style-type: none"><li>• スコープ内の識別名を指定する</li></ul>
property	<ul style="list-style-type: none"><li>• name属性で指定された識別名にバインドされたJavaBeanのプロパティ名を指定する</li></ul>

# <logic:iterate>タグ

- 識別名**empList**でスコープに登録されているコレクションから1番目の要素 (JavaBean1)を取り出し、識別名**emp**に格納する
- 格納された要素に対して処理(<bean:write>)を行う
- すべての要素に対して順番に同じ処理を繰り返す

JSPサンプル

```
<logic:iterate id="emp" name="empList" />
  <bean:write name="emp" property="name" />:
  <bean:write name="emp" property="address" />
</logic:iterate>
```

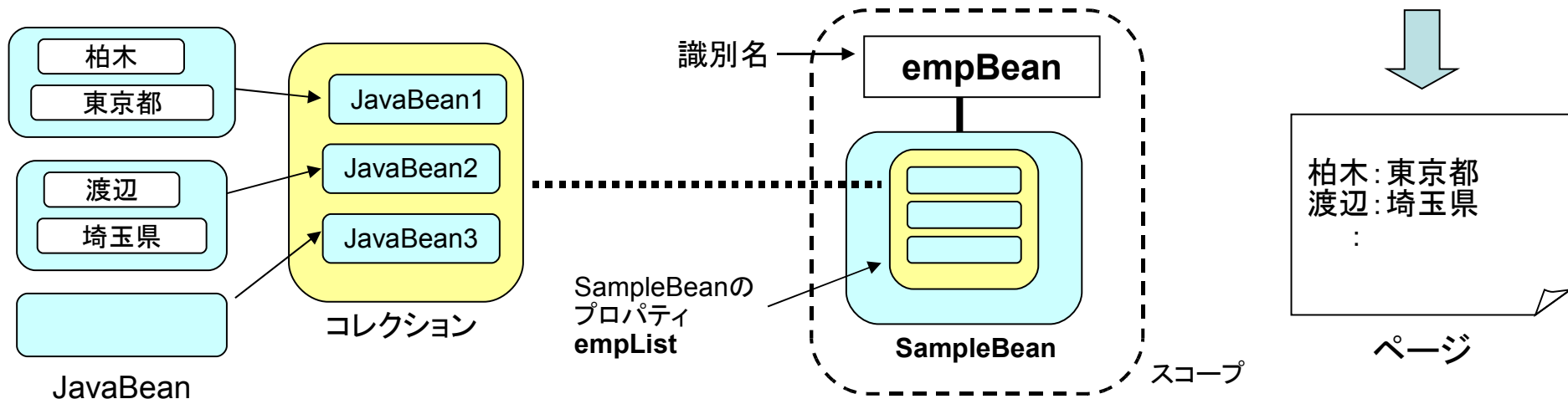


# <logic:iterate>タグ

- 識別名**empBean**でスコープに登録されているJavaBean(例では**SampleBean**)のプロパティ**empList**を指定し、その1番目の要素(JavaBean1)を取り出し、識別名**emp**に格納する
- 格納された要素に対して処理(<bean:write>)を行う
- すべての要素に対して順番に同じ処理を繰り返す

JSPサンプル

```
<logic:iterate id="emp" name="empBean" property="empList">  
  <bean:write name="emp" property="name" />  
  <bean:write name="emp" property="address" />  
</logic:iterate>
```



# その他のLogic タグ

- <logic:equal>
- <logic:notEqual>
- <logic:greaterEqual>
- <logic:greaterThan>
- <logic:lessEqual>
- <logic:lessThan>
- <logic:match>
- <logic:notMatch>
- <logic:empty>
- <logic:notEmpty>
- など

# アクションコンフィグレーションファイル

株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# アクションコンフィギュレーションファイル

- ActionServletがアプリケーションを制御するための情報が定義されたファイル
- XML形式で記述される
- ロケーションとファイル名
  - /WEB-INF/struts-config.xml
    - web.xmlの定義で変更可能



# アクションコンフィギュレーションファイル

- 主な記述内容
  - 共通遷移先管理
  - ActionFormの定義
  - Actionの定義
    - アクション名とActionForm、Actionなどのリソースの関連付け
    - Actionごとの遷移先管理
  - メッセージリソースファイル定義

# アクションコンフィギュレーションファイルの全体像

struts-config.xml

```
<struts-config>  
  <global-forwards>  
    <forward... />  
  </global-forwards>
```

すべてのActionから利用できる  
<global-forwards>の定義

```
<form-beans>  
  <form-bean name="" type="" />  
</form-beans>
```

ActionFormの定義

```
<action-mappings>  
  <action path="" type="" name="" scope=""...>  
    <forward name="" path="" />  
  </action>  
</action-mappings>
```

Actionの定義

```
  <message-resources parameter="" />  
</struts-config>
```

このActionからのみ利用  
できる<forward>の定義

メッセージリソース  
ファイルの定義

# ActionFormの定義

```
<form-beans>
  <form-bean name="sampleForm" type="test.SampleForm" />
  <form-bean name="*****" type="*****" />
  :
</form-beans>
```

複数の<form-bean>定義が可能

## <form-bean>タグの主な属性

属性	説明
name	<ul style="list-style-type: none"><li>• ActionFormの識別名を指定(通常、小文字から始める)</li><li>• 定義した後は、識別名を使って実際のクラスを参照する</li><li>• 実際のクラス名は使わない</li></ul>
type	<ul style="list-style-type: none"><li>• 実際のActionFormクラス名(パッケージ名を含む)</li></ul> 例) test.SampleForm (testパッケージのSampleFormクラス)

# Actionの定義

```
<action-mappings>
```

```
<action path="/sample1"
        type="test.SampleAction"
        name="sampleForm"
        scope="request"
        validate="true"
        input="/test/error.jsp" >
  <forward name="success"
          path="/test/next.jsp" />
</action>
```

```
<action path="/sample2" type="****" ..... >
  <forward name= ..... />
  :
</action>
```

```
</action-mappings>
```

複数の<action>  
定義が可能

# Actionの定義

## <action>タグの主な属性

属性	説明
path	<ul style="list-style-type: none"><li>• Actionの識別名 (/ から始まる)</li><li>• リクエストの処理は、アクション名に一致した識別名を持つ&lt;action&gt;定義に従って実行される</li></ul>
type	<ul style="list-style-type: none"><li>• 実際のActionクラス名 (パッケージ名を含む)</li></ul> 例) test.SampleAction (testパッケージのSampleActionクラス)
name	<ul style="list-style-type: none"><li>• &lt;form-bean&gt;タグで設定したActionForm識別名を指定する</li><li>• このname属性で指定されたActionFormが、type属性で指定されたActionのexecute()メソッドの第2引数として渡される</li></ul>
scope	<ul style="list-style-type: none"><li>• name属性で指定されたActionFormを登録するスコープを指定する (request か session)</li></ul>
validate	<ul style="list-style-type: none"><li>• trueを指定すると、name属性で指定されたActionFormのvalidate()メソッドでリクエストパラメータを検証する</li><li>• 規定値はfalse (validate()メソッドによる検証はしない)</li></ul>
input	<ul style="list-style-type: none"><li>• validate属性がtrueの場合に必要な属性。validate()メソッドによる検証結果がエラーを含む場合に表示するページを指定する</li></ul>

# <forward>タグの定義

```
<action-mappings>
  <action path=".." type=".." ..... >
    <forward name="success" path="/test/next.jsp" />
    <forward name="fail" path="/test/previous.jsp" />
  </action>
</action-mappings>
```

← 複数の<forward>を定義してActionの結果によって遷移先を分けることが可能

## <forward>タグの主な属性

属性	説明
name	<ul style="list-style-type: none"><li>• 遷移先名を指定</li><li>• この&lt;forward&gt;が属している&lt;action&gt;で指定されたActionのexecute()メソッドからのみ遷移先名を参照することができ、pathで指定した遷移先へ移行できる</li></ul>
path	<ul style="list-style-type: none"><li>• 遷移先のパスを指定</li></ul> 例) path="/test/next.jsp" ( / から始める)

# <global-forwards>タグの定義

```
<struts-config>
  <global-forwards>
    <forward name="success" path="/test/login.jsp" />
    <forward name="success" path="/test/menu.jsp" />
    :
  </global-forwards>
  :
```

## <global-forwards>タグの主な属性

属性	説明
name	<ul style="list-style-type: none"><li>・ 遷移先名を指定</li><li>・ すべてのActionのexecute()メソッドからこの遷移先名を参照することができ、pathで指定した遷移先へ移行できる</li></ul> 例) すべてのアクションからmenu.jspへ移行できる
path	<ul style="list-style-type: none"><li>・ 遷移先のパスを指定</li></ul> 例) path="/test/menu.jsp" ( / から始める)

# <message-resources>タグの定義

```
<struts-config>
  :
  <message-resources parameter="MessageResources" />
  :
</struts-config>
```

## <message-resources>タグの主な属性

属性	説明
parameter	• メッセージリソースファイル名を指定する(拡張子は付けない)



# メッセージリソースファイル 実際のファイルと定義名の関係

- 実際のファイルがResources.properties の場合、定義名はResourcesとする(拡張子は.properties という決まり)
- ファイルは、/WEB-INF/classes以下に配置すること
- /WEB-INF/classes以下のサブディレクトリに配置した場合、定義名にもサブディレクトリ名を含まなければならない

```
:  
WEB-INF  
|-- classes  
    |-- msg  
        |--Resources
```



```
<struts-config>  
    :  
    <message-resources  
        parameter="msg.Resources" />  
    :  
</struts-config>
```

# web.xml

株式会社ナレッジエクス  
<http://www.knowledge-ex.jp/>

# web.xml

- ServletコンテナがWebアプリケーションの動作を制御するための設定ファイル
- ActionServletに関する定義を記述する
- /WEB-INF直下に配置する

# web.xmlのServlet定義サンプル

```
<web-app>
  :
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>
      org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  :
</web-app>
```

Servlet識別名  
の定義

アクション  
コンフィギュレーション  
ファイル定義

Servlet起動順

URLパターン  
定義

# Servlet識別名の定義

```
<web-app>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    :
  </servlet>
  :
```

属性	説明
servlet-name	<ul style="list-style-type: none"><li>このアプリケーションで使うServletの識別名を指定する</li><li>&lt;sevlet-mapping&gt;セクションでもここで定義した名前を使用する</li></ul>
servlet-class	<ul style="list-style-type: none"><li>このアプリケーションで使うServlet名を指定する</li><li>完全クラス名(すべてのパッケージ名を含むクラス名)で定義する</li><li>通常、org.apache.struts.action.ActionServletを使用する</li></ul>

# アクションコンフィグレーションファイル定義

```
<web-app>
  <servlet>
    :
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    :
```

属性	説明
init-param	• Servletに渡されるパラメータを定義する
param-name	• パラメータの識別名を指定する • Strutsではconfig という名前で、使用するアクションコンフィグレーションファイルを指定する
param-value	• 使用するアクションコンフィグレーションファイルのパス名を指定する

# Servlet起動順

```
<web-app>
  <servlet>
    :
    <load-on-startup>1</load-on-startup>
  </servlet>
  :
```

属性	説明
load-on-startup	• Servletコンテナ起動時に、<servlet>で定義したServletをロードする順番を指定する

# URLパターン定義

```
<web-app>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    :
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  :
```

属性	説明
servlet-name	• <servlet>セクションで定義したServletの識別名を指定する
url-pattern	• ここで指定したパターンのアクション名を受け取ると、<servlet-name>で指定した識別名で定義されたServletが呼び出される • 上の例では、.doという拡張子をもつアクション名を受け取ると、 <b>action</b> という識別名で定義されたServlet, すなわち <b>ActionServlet</b> が呼び出される



# 参考：その他の機能紹介

- Validator PlugIn
  - このプラグインでvalidate()メソッドでの検証を代替できる
- DynaActionForm
  - Javaコードを使わずにActionFormを作成できる
- DispatchAction
  - 同じ画面の複数のボタンからそれぞれの処理へ移行できる