



TOKYO IT SCHOOL

Validator

目次

| | |
|---------------------------|----|
| 1. validate メソッドによる入力チェック | 1 |
| 2. Validator を準備する | 9 |
| 3. DynaValidatorForm | 23 |
| 4. 定数 | 24 |
| 5. 独自の検証ルールを追加する | 26 |

1. validate メソッドによる入力チェック

1 validate()メソッドとは

リクエストパラメータの値は検証が必要な場合が多々あります。例えば、数値を入力すべき箇所に、文字列が入力される場合などです。このような値がビジネスロジックで処理できるものであるかどうかを事前にチェックするために、ActionForm には validate ()メソッドが用意されています。validate()メソッドは、ActionForm のプロパティにリクエストパラメータの値が格納され、Action に渡される前に呼び出されます。また、記述した validate()メソッドを Struts に実行させるには、Struts 設定ファイル内で、この ActionForm と関連付けられた action 要素の validate 属性に true を指定し、エラーがあったときに表示される JSP または HTML を input 属性に指定する必要があります。

2 validate()メソッドの内容

validate()メソッドは、ActionMapping と HttpServletRequest を引数に持ち、ActionErrors を戻り値として作成します。また、validate()メソッドの中には以下のような処理を記述していきます。

- (1) ActionErrors クラスのインスタンスの生成
- (2) 検証ロジックの記述。エラーがある場合は、ActionMessage クラスのインスタンスを生成して、ActionErrors に追加
- (3) Action Errors を戻り値として返す

ActionErrors はエラーメッセージを保持するクラスで、ActionMessage の集合体です。検証ロジックにて異常値が見つかった場合は、ActionErrors クラスの add()メソッドを使用して、ActionMessage のインスタンスを ActionErrors に追加していきます。

add()メソッドの第 1 引数には、エラーを識別するためのプロパティ名を指定します。このプロパティ名は html:errors カスタムタグの property 属性の値として使用することができます。

第 2 引数には ActionMessage のコンストラクタを指定します。ActionMessage のコンストラクタの第 1 引数にはメッセージリソースのキー、第 2 引数以降にはメッセージのパラメータを必要に応じて指定します。たとえば、メッセージリソースファイルに以下の記述がある場合、

```
errors.required={0}を入力してください。
```

上の例のように

```
new ActionMessage("errors.required", "パスワード");
```

とすると、エラーメッセージを表示する際に、{0}の部分が第 2 引数の文字列"パスワード"と置き換えられて、「パスワードを入力してください。」と表示されます。同じように{1}は第 3 引数に、{2}は第 4 引数に置き換えられていきます。

そして validate()メソッドの戻り値の ActionErrors が空または null である場合、ActionForm は Action に渡されて処理されます。それ以外のときにはエラーとして扱われます。

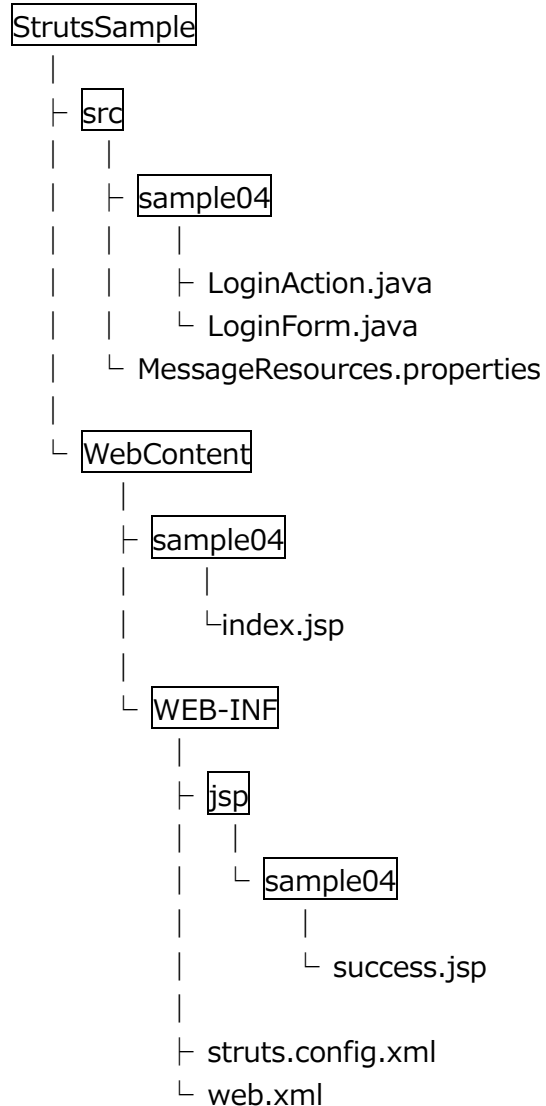
3 validate()メソッドの実装

それでは、実際に validate()メソッドを実装してみます。作成、および変更するファイルの一覧を以下に示します。

| ファイル名 | 解説 | ロケーション |
|-----------------------------|---------------|------------------------|
| struts-config.xml | Struts 設定ファイル | /WEB-INF/ |
| LoginAction.java | Action | /src/sample04/ |
| LoginForm.java | ActionForm | /src/sample04/ |
| success.jsp | 結果表示用 JSP | /WEB-INF/jsp/sample04/ |
| index.jsp | 入力フォーム | /sample04/ |
| MessageResources.properties | リソースファイル | /src/ |

Eclipse 上から見たディレクトリ構成を以下に示します。

【ディレクトリ構成】



まずは、リソースファイルの作成です。必須入力用のエラーメッセージを用意します。

【MessageResources.properties】

errors.required={0}を入力してください。

続いて ActionForm への validate()メソッドの実装です。

【LoginForm.java】

```
package sample04;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;

public class LoginForm extends ActionForm {

    private String id = null;
    private String password = null;

    public String getId() {
        return id;
    }

    public String getPassword() {
        return password;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public ActionErrors validate(ActionMapping mapping,
        HttpServletRequest request) {

        // ActionErrors クラスのインスタンスを生成
        ActionErrors errors = new ActionErrors();
```

```
// ID の必須チェック
if ("".equals(getId())) {
    // エラーメッセージを登録
    errors.add("id", new ActionMessage("errors.required", "ID"));
}

// パスワードの必須チェック
if ("".equals(getPassword())) {
    // エラーメッセージを登録
    errors.add("password",
        new ActionMessage("errors.required", "パスワード"));
}

// ActionErrors を返す
return errors;
}
}
```

Action クラスには特別な処理は記述しておりません。

【LoginAction.java】

```
package sample04;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LoginAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {

        LoginForm loginForm = (LoginForm) form;

        String id = loginForm.getId();
        String password = loginForm.getPassword();
```

```
String message = " ID : " + id + " パスワード : " + password;

request.setAttribute("message", message);

return mapping.findForward("success");
}
}
```

続いて入力フォームの JSP の作成です。返されたエラーメッセージを表示するには、HTML タグライブラリの errors タグを使用します。

【index.jsp】

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>ログイン画面</title>
</head>
<body>
<h2>ログイン画面</h2>
<html:form method="POST" action="/sample04/login.do">
  ID: <html:text property="id" />
  <html:errors property="id" /><br>
  PASSWORD: <html:password property="password" />
  <html:errors property="password" /><br>
  <html:submit>ログイン</html:submit>
</html:form>
</body>
</html:html>
```

html:errors タグには以下の属性を指定できます。

html:errors タグの属性

| 属性名 | 必須 | 記述 |
|--------|----|---|
| bundle | | 使用するメッセージリソースを取得するための application スコープのキーを指定します。 デフォルトは、Globals.MESSAGES_KEY です。 |
| locale | | 表示するメッセージを選択するために使用する Locale オブジェクトが格納されている session スコープのキーを指定します。 デフォルトは、Globals.LOCALE_KEY です。 |

| | | |
|----------|--|--|
| name | | request スcope中の、エラーメッセージが保存されている Bean のキーを指定します。デフォルトは、Globals.ERROR_KEY です。 |
| property | | 表示するエラーメッセージのプロパティ名を指定します。指定しない場合、(プロパティに関わらず) すべてのエラーメッセージが 表示されます。 |

また、エラーメッセージは個別ではなく、全てをまとめて表示することもできます。その際は、以下のように html:errors タグの属性を省略します。

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

また、メッセージリソースに以下の特殊なキーを設定することにより、エラーメッセージの前後に共通的な HTML タグを出力させることもできます。

メッセージリソース中の特殊なキー

| キー | 説明 |
|---------------|---|
| errors.header | errors タグの先頭に埋め込むメッセージ (HTML 要素) を指定します。などを想定しています。 |
| errors.footer | errors タグの最後に埋め込むメッセージ (HTML 要素) を指定します。などを想定しています。 |
| errors.prefix | 各エラーメッセージの先頭に埋め込むメッセージ (HTML 要素) を指定します。などを想定しています。 |
| errors.suffix | 各エラーメッセージの最後に埋め込むメッセージ (HTML 要素) を指定します。や などを想定しています。 |

エラーメッセージの表示に関する説明は以上となります。
続いて、結果表示用の JSP を作成します。

【success.jsp】

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>ログイン結果</title>
  </head>
  <body>
    <h2>ログイン結果</h2>
    <%= (String)request.getAttribute("message") %>
  </body>
</html>
```


最後に Struts 設定ファイルの作成です。

【struts-config.xml】

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
    <form-beans>
        <form-bean name="sample01_loginForm" type="sample01.LoginForm"/>
        <form-bean name="sample02_loginDynaForm"
            type="org.apache.struts.action.DynaActionForm">
            <form-property name="id" type="java.lang.Integer" initial="0"/>
            <form-property name="password" type="java.lang.String" />
        </form-bean>
        <form-bean name="sample03_htmlTagForm"
            type="sample03.HtmlTagForm"/>
        <form-bean name="sample04_loginForm" type="sample04.LoginForm"/>
    </form-beans>

    <action-mappings>
        <action path="/sample01/login"
            type="sample01.LoginAction"
            name="sample01_loginForm"
            scope="request">
            <forward name="success" path="/WEB-INF/jsp/sample01/success.jsp"/>
        </action>
        <action path="/sample02/login"
            type="sample02.LoginAction"
            name="sample02_loginDynaForm"
            scope="request">
            <forward name="success" path="/WEB-INF/jsp/sample02/success.jsp"/>
        </action>
        <action path="/sample03/htmlTag"
            type="sample03.HtmlTagAction"
            name="sample03_htmlTagForm"
            scope="request">
            <forward name="success"
                path="/WEB-INF/jsp/sample03/htmlTag.jsp"/>
        </action>
        <action path="/sample03/beanTag"
            type="sample03.BeanTagAction">
```



```
scope="request">
  <forward name="success"
path="/WEB-INF/jsp/sample03/beanTag.jsp"/>
  </action>
  <action path="/sample04/login"
    type="sample04.LoginAction"
    name="sample04_loginForm"
    scope="request"
    validate="true"
    input="/sample04/index.jsp">
    <forward name="success" path="/WEB-INF/jsp/sample04/success.jsp"/>
  </action>
</action-mappings>
<message-resources parameter="MessageResources"/>
</struts-config>
```

それでは、動作を確認しましょう。Tomcat を起動し、下記 URL をブラウザで表示します。

<http://localhost:8080/StrutsSample/sample04/index.jsp>

■ ログイン画面



■ ログイン画面（エラー時）



■ ログイン結果画面



入力エラー時には入力画面へエラーメッセージが出力され、エラー未検出時には次画面へと遷移する動きが確認できました。

2. Validator を準備する

1 Validator とは

前項では、フォームに入力された値の検証を `ActionForm` で `validate()` メソッドをオーバーライドすることにより行いました。しかし、入力値のチェックが文字数の制限、数値の範囲など非常に単純な場合は、`validate()` メソッドに処理を記述するのは面倒です。

そこで Struts には、XML ベースの設定ファイルにルールを定義することで、検証作業を一手に引き受ける仕組み「Validator」が提供されています。以下に Validator の特徴を列挙します。

豊富な検証ルール

Validator には、フォームを作成する際によく使用される検証ロジックが、標準検証ルールとしてあらかじめ多数用意されています。これらは、検証ロジックをコーディングすることなく、すぐに使用ができます。また、独自の検証ルールを作成することも可能です。

`validate()` メソッドと同じエラーメッセージハンドリング

Validator による検証の結果として出力されるエラーメッセージは、`validate()` メソッドを使用した場合と同じく、`ActionMessage`、`ActionErrors`、そして `html:errors` タグでハンドリングしますので、メッセージを独自に管理する必要がありません。

DynaActionForm の検証

Struts 設定ファイルから動的に生成される `DynaActionForm` は、`validate()` メソッドが使用できませんが、Validator を使用すれば検証可能です。

JavaScript によるクライアントサイド検証

Validator ではサーバサイドで行う検証の他に、JavaScript によるクライアントサイドで入力値の検証を行う機能も提供されています。これにより、サーバの負荷を低減することが可能です。

2 Validator 利用の流れ

Validator を利用するには次のような準備が必要となります。

- JAR ファイルの配置
- Struts 設定ファイルで Validator プラグインを利用できるようにする
- 検証ルール設定ファイルの作成と配置
- Validator により拡張された `ActionForm` クラスを継承したクラスの作成

(1) Validator プラグインの登録

まず、Validator はプラグインとして提供されていますので、`commons-validator.jar` と `jakarta-oro.jar` というファイルが必要です。これは、Struts をインストールしたディレクトリの `lib` ディレクトリに含まれます。このファイルを、`WEB-INF/lib` ディレクトリにコピーしてください。

そして、同じく Struts の lib ディレクトリから validator-rules.xml というファイルを ¥WEB-INF にコピーしておきます。

次に、Struts 設定ファイルに設定を行います。次のように記述してください。

【struts-config.xml】

```
<struts-config>
  . . .
  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
      value="/WEB-INF/validator-rules.xml,
      /WEB-INF/validation.xml"/>
  </plug-in>
</struts-config>
```

plug-in 要素で、使用するプラグインを指定します。同時に、入力値の検証に使用する設定ファイルも指定します。この例では、validator-rules.xml と validation.xml を ¥WEB-INF¥ の直下に置くよう指定しています。validation.xml はこの後で作成します。

(2) 検証ルール設定ファイルの準備

Validator では 2 つの検証ルール設定ファイルを使用します。それぞれの役割は以下の通りです。

| ファイル | 役割 |
|---------------------|-----------------------------|
| validator-rules.xml | 検証の方法を具体的に設定します。 |
| validation.xml | フォームのパラメータの値ごとに検証の方法を設定します。 |

validator-rules.xml には、検証ルールそのものを指定します。これに対し、validation.xml では、ActionForm と適用する検証ルールを指定します。

validator-rules.xml は独自で作成することも可能ですが、一般的な検証ルールについては、その検証ロジックが commons-validator.jar の中に用意されており、検証ルールを記述済みの validator-rules.xml が Struts にあらかじめ添付されています。

validation.xml は ActionForm との関連性の設定を行うので、開発者によって作成する必要があります。

(3) ActionForm の作成

Validator の機能を利用する場合は、通常の ActionForm ではなく、Validator が拡張した ActionForm である ValidatorForm のサブクラスを作成します。このクラスは、フォームのパラメータについて、getter/setter メソッドを作成するだけで実装は完了です。validate() メソッドを実装する必要はありません。

(4) 検証ルール設定ファイルの作成

validator-rules.xml は Struts から提供されているものを利用しますので、通常は特に手を加える必要はありませんが、以下にどのようなルールが定められているのかを示します。

提供されている標準検証ルール

| ルール名 | 説明 | パラメータ | メッセージリソースのデフォルトキー |
|------------|--|----------|-------------------|
| required | このフィールドに入力されているかどうかを検証します。 | 不要 | errors.required |
| validwhen | 指定した条件（たとえば「XX のフィールドの値が XXX だった場合」など）を満たす場合に required ルールで 検証します。 | 必須 | errors.required |
| minlength | 入力された文字列の長さが指定された長さ以上であるかどうかを検証します。 | 必須 | errors.minlength |
| maxlength | 入力された文字列の長さが指定された長さ以下であるかどうかを検証します。 | 必須 | errors.maxlength |
| mask | 入力された文字列が指定した正規表現にマッチするかどうかを検証します。 | 必須 | errors.invalid |
| byte | byte 型に変換できるかどうかを検証します。 | 不要 | errors, byte |
| short | short 型に変換できるかどうかを検証します。 | 不要 | errors.short |
| integer | int 型に変換できるかどうかを検証します。 | 不要 | errors.integer |
| long | long 型に変換できるかどうかを検証します。 | 不要 | errors.long |
| float | float 型に変換できるかどうかを検証します。 | 不要 | errors.float |
| double | double 型に変換できるかどうかを検証します。 | 不要 | errors.double |
| date | 指定した日付の型にマッチするかどうかを検証します。 | デフォルト値あり | errors.date |
| intRange | 入力された数値が指定した int 型の最小値、最大値の 範囲に入るかどうかを検証します。 | 必須 | errors.range |
| floatRange | 入力された数値が指定した float 型の最小値、最大値の範囲に入るかどうかを検証します。 | 必須 | errors.range |
| creditCard | クレジットカード番号として適切かどうかを検証します。ハイフンを使用すると、不正な番号として扱われます。 | 不要 | errors.creditcard |
| email | メールアドレスとして適切かどうかを検 | 不要 | errors.email |

| | | | |
|-----|----------------------|----|------------|
| | 証します。 | | |
| url | URL として適切かどうかを検証します。 | 不要 | errors.url |

上記のルールを ActionForm に提供するためには validation.xml を作成し、Struts 設定ファイルで指定したディレクトリに置く必要があります。以下に validation.xml の例を示します。

【validation.xml】

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules
    Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<form-validation>
  <formset>
    <form name="sample05_sampleValidatorForm">
      <field property="userName"
        depends="required">
        <msg name="required" key="errors.empty"/>
        <arg name="required" key="labels.userName" position="0"/>
      </field>
      <field property="zipCode"
        depends="mask">
        <arg name="mask" key="labels.zipCode" position="0"/>
        <var>
          <var-name>mask</var-name>
          <var-value>^¥d{3}(-¥d{4})?$</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>
```

formset 要素の中で、各フォームのパラメータに対する検証方法を定義します。検証の対象となる ActionForm の名前を、form 要素の name 属性に指定します。ここで指定する ActionForm は、Struts 設定ファイルの form-bean 要素で指定されている必要があります。

次に検証を行う ActionForm のプロパティの設定を行います。field 要素の property 属性に、検証ルールを設定するプロパティを指定します。上記の例では、userName プロパティ、zipCode プロパティに検証ルールの適用を行っています。

field 要素の depends 属性では、適用する検証ルールの名前を指定します。この例では userName プロパティは値が設定されているかどうか、zipCode プロパティでは指定した正規表現にマッチするかどうかを検証されます。ここには複数のルールをカンマ区切りで指定することができます。その際の検証は、記述した順に行われていきます。

field 要素の子要素には、エラーメッセージの設定を行うものと、検証を行う際に必要となる

パラメータの指定を行うものがあります。

msg 要素は、検証失敗時に表示するエラーメッセージのキーを指定します。name 属性でどの検証ルールに対するメッセージなのかを指定し、key 属性でメッセージリソースを検索する際のキーを指定します。もし、key 属性で指定されるメッセージにパラメータを渡すときは、arg 要素で指定します。

arg 要素にはエラーメッセージに埋め込むフィールド名を示すメッセージリソースのキーを指定します。name 属性には msg 要素と同じようにどの検証ルールなのかを指定し、key 属性でキー名を、position 属性でメッセージに渡すパラメータの格納位置を指定します。たとえばメッセージリソースファイルに次の記述がある場合、

```
labels.userNameee = 氏名
errors.empty = {0}を入力してください
```

この required ルールによる検証に失敗すると、「氏名を入力してください」というエラーメッセージを表示できるようになります。

また、msg 要素でキーを指定しなかった場合は、デフォルトのキーが割り当てられます。たとえば、zipCode の場合（検証ルール mask）は、errors.invalid となります。その他のルールのデフォルトキーは、表「提供されている標準検証ルール」の「メッセージリソースのデフォルトキー」にて記述していますので参照してください。

最後に、検証を行う際に必要となるパラメータの指定を var 要素により行います。この要素の子要素である var-name 要素でどのルールに渡すパラメータなのか、var-value 要素でその値を指定します。ここでは、検証ルール mask に、郵便番号の正規表現を渡しています。なお、検証の際にパラメータが必要かどうかは検証ルールによって違います。

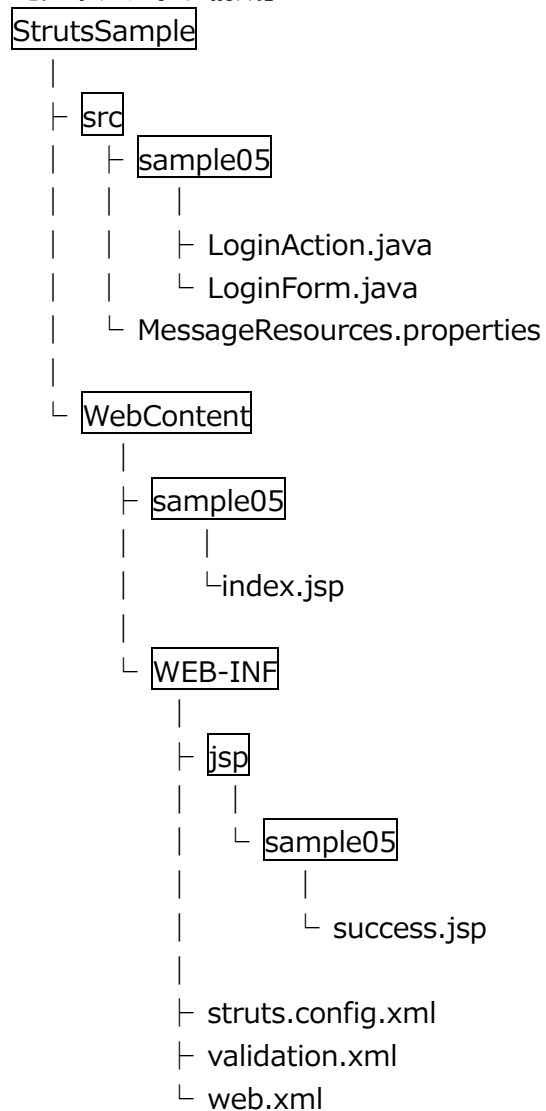
3 Validator の実装

それでは、実際に Validator を使用した簡単なログイン画面を実装してみます。作成、および変更するファイルの一覧を以下に示します。

| ファイル名 | 解説 | ロケーション |
|-----------------------------|-------------------|------------------------|
| struts-config.xml | Struts 設定ファイル | /WEB-INF/ |
| LoginAction.java | Action | /src/sample05/ |
| LoginForm.java | ActionForm | /src/sample05/ |
| success.jsp | 結果表示用 JSP | /WEB-INF/jsp/sample05/ |
| index.jsp | 入力フォーム | /sample05/ |
| validation.xml | Validation 設定ファイル | /WEB-INF/ |
| MessageResources.properties | リソースファイル | /src/ |

Eclipse 上から見たディレクトリ構成を以下に示します。

【ディレクトリ構成】



まずは、リソースファイルの作成です。項目ラベルと正規表現用のエラーメッセージを用意します。

【MessageResources.properties】

```
labels.id = ID
labels.password = パスワード

errors.required={0}を入力してください。
errors.invalid={0}は不正です。
```

続いて ActionForm の作成です。org.apache.struts.validator.ValidatorForm を継承して作成しています。

【LoginForm.java】

```
package sample05;

import org.apache.struts.validator.ValidatorForm;

public class LoginForm extends ValidatorForm {

    private String id = null;
    private String password = null;

    public String getId() {
        return id;
    }

    public String getPassword() {
        return password;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```

Action クラスには特別な処理は記述しておりません。

【LoginAction.java】

```
package sample05;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LoginAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {

        LoginForm loginForm = (LoginForm) form;

        String id = loginForm.getId();
        String password = loginForm.getPassword();
        String message = " ID : " + id + " パスワード : " + password;

        request.setAttribute("message", message);

        return mapping.findForward("success");
    }
}
```

入力フォーム、および正常時の JSP の作成です。sample04 と遷移先以外は変更していません。

【index.jsp】

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>ログイン画面</title>
</head>
<body>
```

```
<h2>ログイン画面</h2>
<html:form method="POST" action="/sample05/login.do">
  ID: <html:text property="id" /><html:errors property="id" /><br>
  PASSWORD: <html:password property="password" /><html:errors
property="password" /><br>
  <html:submit>ログイン</html:submit>
</html:form>
</body>
</html:html>
```

【success.jsp】

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>ログイン結果</title>
  </head>
  <body>
    <h2>ログイン結果</h2>
    <%= (String)request.getAttribute("message") %>
  </body>
</html>
```

続いて Struts 設定ファイルの作成です。Validator の読み込みを行っています。

【struts-config.xml】

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
  <form-beans>
    <form-bean name="sample01_loginForm" type="sample01.LoginForm"/>
    <form-bean name="sample02_loginDynaForm"
      type="org.apache.struts.action.DynaActionForm">
      <form-property name="id" type="java.lang.Integer" initial="0"/>
      <form-property name="password" type="java.lang.String" />
    </form-bean>
    <form-bean name="sample03_htmlTagForm"
      type="sample03.HtmlTagForm"/>
  </form-beans>
</struts-config>
```

```
<form-bean name="sample04_loginForm" type="sample04.LoginForm"/>
<form-bean name="sample05_loginForm" type="sample05.LoginForm"/>
</form-beans>

<action-mappings>
  <action path="/sample01/login"
    type="sample01.LoginAction"
    name="sample01_loginForm"
    scope="request">
    <forward name="success" path="/WEB-INF/jsp/sample01/success.jsp"/>
  </action>
  <action path="/sample02/login"
    type="sample02.LoginAction"
    name="sample02_loginDynaForm"
    scope="request">
    <forward name="success" path="/WEB-INF/jsp/sample02/success.jsp"/>
  </action>
  <action path="/sample03/htmlTag"
    type="sample03.HtmlTagAction"
    name="sample03_htmlTagForm"
    scope="request">
    <forward name="success"
path="/WEB-INF/jsp/sample03/htmlTag.jsp"/>
  </action>
  <action path="/sample03/beanTag"
    type="sample03.BeanTagAction"
    scope="request">
    <forward name="success"
path="/WEB-INF/jsp/sample03/beanTag.jsp"/>
  </action>
  <action path="/sample04/login"
    type="sample04.LoginAction"
    name="sample04_loginForm"
    scope="request"
    validate="true"
    input="/sample04/index.jsp">
    <forward name="success" path="/WEB-INF/jsp/sample04/success.jsp"/>
  </action>
  <action path="/sample05/login"
    type="sample05.LoginAction"
    name="sample05_loginForm"
    scope="request"
    validate="true"
```

```
        input="/sample05/index.jsp">
        <forward name="success" path="/WEB-INF/jsp/sample05/success.jsp"/>
    </action>
</action-mappings>
<message-resources parameter="MessageResources"/>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"
    />
</plug-in>
</struts-config>
```

最後に validation.xml の作成です。ID に対しては必須チェック、パスワードに対しては半角英数字チェックを行っています。

【validation.xml】

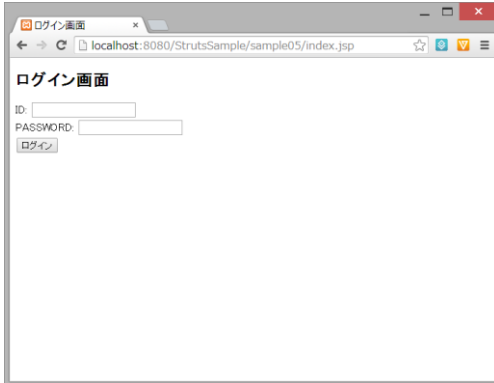
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules
    Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<form-validation>
    <formset>
        <form name="sample05_loginForm">
            <field property="id"
                depends="required">
                <arg name="required" key="labels.id" position="0"/>
            </field>
            <field property="password"
                depends="mask">
                <arg name="mask" key="labels.password" position="0"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[0-9a-zA-Z]*$</var-value>
                </var>
            </field>
        </form>
    </formset>
</form-validation>
```

作成、および修正するファイルは以上です。

それでは、動作を確認しましょう。Tomcat を起動し、下記 URL をブラウザで表示します。

<http://localhost:8080/StrutsSample/sample05/index.jsp>

■ ログイン画面



ログイン画面

ID:

PASSWORD:

■ ログイン画面（エラー時）



ログイン画面

ID:

- IDを入力してください。

PASSWORD:

- パスワードは不正です。

■ ログイン結果画面



ログイン結果

ID: abode パスワード: 12345

Validator による入力チェックが確認できました。

4 他のフィールドの状態により検証を行う

あるフィールドの検証を行う際、他のフィールドの入力状態により、検証するかしないかを選択したい場合があります。たとえば、パスワードの値と確認用パスワードの値が一致しているかを検証する場合などです。このような場合にも対応できるよう、Struts には `validwhen` という検証ルールが用意されています。

`validwhen` を利用するには `antlr.jar` というファイルが必要です。Struts のライブラリに含まれていますので、これを `WEB-INF/lib` ディレクトリにコピーします。

`validwhen` を使用するにはパラメータを指定する必要があります。`var-name` 要素には `"validwhen"` ではなく `"test"` を指定します。そして `var-value` 要素には評価結果が真偽値となる式を指定します。式は評価結果が真の場合に検証が成功するように記述します。式で使用できる値は以下の通りです。

| | |
|---------------------|--|
| 文字列 | シングルクォーテーションがダブルクォーテーションで囲んで使用します。 |
| 整数 | 10 進法、16 進法、8 進法が使用できます。 |
| null | リテラルの <code>"null"</code> は null か空文字列を表します。 |
| 他のフィールドの値 | フィールド名、たとえば <code>userName</code> などで指定できます。 |
| <code>*this*</code> | 検証対象となっているフィールドを表します。 |

では、以下のような JSP のパスワードと確認用パスワードの値が一致しているかを検証する設定を以下に示します。

【JSP】

```
<form action="/sampleValidWhen" method="POST">
  パスワード : <html:password property="password" /><br>
  パスワード（確認用） : <html:password property="passconf" /><br>
  <html:submit> 送信</html:submit>
</html:form>
```

【validation.xml】

```
<formset>
  <form name="sampleValidwhenForm">
    <field property="password" depends="validwhen">
      <arg name="validwhen" key="labels.password" position="0"/>
      <var>
        <var-name>test</var-name>
        <var-value>(*this* == passconf)
      </var-value>
      </var>
    </field>
  </form>
</formset>
```


式を記述する際に注意しなければならないのは以下の点です。

- ・ 比較式は"()"（かっこ）で囲う必要があります
- ・ 比較式を連結する and や or は 1 度のみ使用可能です。たとえば以下のような式は無効です。
`<var-value>((*this* ==1) or (*this* == 3) or (*this* == 5))</var-value>`
- ・ 比較するものがいずれも整数に変換可能である場合は数値比較が行われ、そうでない場合は文字列比較が行われます。

3. DynaValidatorForm

1 DynaValidatorForm

DynaValidatorForm は、Validator による検証が可能な DynaActionForm です。DynaValidatorForm を使用して ActionForm を作成する場合、ActionForm に関する設定を Struts 設定ファイルに、検証ルールを validation.xml ファイルに定義しておくだけで、ActionForm の定義、生成、検証までの設定が完了します。

次の Struts 設定ファイルにおける例は、プロパティとして、userName と zipCode を持つ DynaValidatorForm を定義しています。

type 属性に org.apache.struts.validator.DynaValidatorForm を指定することに注意してください。

【struts-config.xml:DynaValidatorForm の使用例】

```
・・・(略)・・・  
<form-bean name="sampleDynaValidatorForm"  
    type="org.apache.struts.validator.DynaValidatorForm"> ----- ①  
    <form-property name="userName" type="java.lang.String" />  
    <form-property name="zipCode" type="java.lang.String" />  
</form-bean>  
・・・(略)・・・
```

DynaValidatorForm のプロパティの検証を行う検証ルールの定義は、ValidationForm の場合と同様です。

4. 定数

1 定数の定義と参照

Validator では、検証ルール適用をよりスマートにすることが可能な、定数が用意されています。複数の検証ルールの中で同じ値を使用する場合などに利用すると良いでしょう。

たとえば mask ルールを用いて、複数のプロパティに対して同じ正規表現を使用する場合などは、正規表現を定数として定義します。定数にはグローバル定数とローカル定数の 2 種類あり、グローバル定数が設定ファイル内すべてで参照できるのに対し、ローカル定数は定義した formset 要素内でしか参照できません。

以下はグローバル定数を用いた例です。

validation.xml (グローバル定数の使用例)

```
... (略) ...  
<form-validation>  
  <global>  
    <constant>  
      <!-- グローバル定数の定義 -->  
      <constant-name>ZIP_MASK</constant-name>  
      <constant-value>^¥d{3}(-¥d{4})?$</constant-value>  
    </constant>  
  </global>  
</formset>  
... (略) ...  
  <form>  
    <field property="zip" depends="mask">  
      <arg name="mask" key="string.zip" position="0"/>  
      <var>  
        <var-name>mask</var-name>  
        <var-value>${ZIP_MASK}</var-value>  
      </var>  
    </field>  
  </form>
```

グローバル定数は、global 要素内で constant 要素を用いて定義します。constant-name 要素が定数名、constant-value 要素がその値となります。global 要素は form-validation 要素内の先頭に定義する必要があるため、注意しましょう。複数のグローバル定数を定義する場合には、その数だけ constant 要素を global 要素内で繰り返します。

上記の例では、郵便番号を検証するための正規表現を「ZIP_MASK」という定数名で定義しています。定義した定数の参照は\${定数名}で行います。

ローカル定数にするには、constant 要素を formset 内に記述することで可能です。以下に例を示します。

validation.xml (ローカル定数の定義例)

```
... (略) ...  
<formset>  
  <constant>  
    <!-- ローカル定数の定義 -->  
    <constant-name>ZIP_MASK</constant-name>  
    <constant-value>^¥d{3}(-¥d{4})?</constant-value>  
  </constant>  
... (略) ...  
</formset>
```

5. 独自の検証ルールを追加する

アプリケーションによっては、独自の検証ルールを追加したい場合があります。Struts の Validator では独自の検証ルールを追加することができます。ここでは、その方法について説明します。

独自の検証ルールを追加するには、以下の手順で行います。

- (1) 検証クラス（メソッド）の作成
- (2) 検証ルールの設定

1 検証クラスの作成

まずは検証クラスおよび検証メソッドの作成について説明します。検証メソッドは任意のクラスに任意の名前で作成できます。また、検証メソッドの戻り値は自由ですが、引数は次の表にあるオブジェクト、またはいずれかの組み合わせでなければなりません。

検証メソッドの引数オブジェクト

| オブジェクト | 説明 |
|--|--|
| java.lang.Object | 検証が行われる JavaBean（ここでは ActionForm）のこと。 |
| java.util.Locale | 現ユーザのロケール。Struts1.2 以降。 |
| javax.servlet.ServletContext | アプリケーションのサーブレットコンテキスト。 |
| javax.servlet.http.HttpServletRequest | 現在のリクエストオブジェクト。 |
| org.apache.commons.validator.Field | 検証が行われるフィールドオブジェクトのこと。JavaBean のプロパティ。 |
| org.apache.commons.validator.Validator | 現在の Validator オブジェクト。Struts1.2 以降。 |
| org.apache.commons.validator.ValidatorAction | 現在実行されている ValidatorAction オブジェクト。 |

また、上記の他にエラーメッセージを作成するために、引数として org.apache.struts.action. ActionMessages の指定も必要となります。

以下に検証クラスの例を示します。

【MyChecks.java】

```
public class MyChecks {  
    public static boolean myVvalidate (Object bean,  
                                       ValidatorAction va,  
                                       Field field,  
                                       ActionMessages errors,  
                                       HttpServletRequest request) {  
        . . .  
        チェック処理を記述  
        . . .  
    }  
}
```

作成した検証メソッドによるルールを使用するためには、XML ファイルに定義しなければなりません。validator-rules.xml に定義を書き加えてもよいですし、新たに設定ファイルを用意し、そこに定義してもかまいません。

以下に検証ルールを my-validation-rules.xml として作成する例を示します。

【my-validator-rules.xml】

```
<!DOCTYPE form-validation PUBLIC  
    "-//Apache Software Foundation//DTD Commons Validator Rules  
    Configuration 1.1.3//EN"  
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">  
<form-validation>  
    <global>  
    <validator  
        name="myCheck" classname="MyChecks"  
        method="myValidate"  
        methodParams="java.lang.Object,  
                     org.apache.commons.validator.ValidatorAction,  
                     org.apache.commons.validator.Field,  
                     org.apache.struts.action.ActionMessages,  
                     javax.servlet.http.HttpServletRequest"  
        msg="errors.myValidate"/>  
    </global>  
</form-validation>
```

検証ルールの定義は、validator 要素に指定します。上の例では、検証ルールの名前を myCheck とし、検証メソッドを MyChecks クラスの myValidate ()としています。このとき、検証メソッドの引数に与えるオブジェクトの型も指定します。

validator 要素の属性は以下の通りです。

validator 要素の属性

| 属性名 | 説明 |
|----------------|---|
| name | 検証ルール名を指定します。ここで指定した名前が validation.xml から参照されます。 |
| className | 検証メソッドを持つクラス名を完全限定名で指定します。 |
| method | 検証メソッド名を指定します。 |
| methodParams | 検証メソッドの引数の型を、カンマ", "で区切って指定します。 |
| msg | 検証に失敗したときに、メッセージリソースからメッセージを検索するためのキーのデフォルトを指定します。 |
| depends | この検証ルールが依存する検証ルール名を指定します。既に定義されている検証ルール名でなければなりません。 |
| jsFunctionName | JavaScript が生成される場合に、使用される名前を指定します。 |

新たに作成したファイルは、Validator プラグインにより読み込まれるように、Struts 設定ファイルに変更を加える必要があります。

以下に例を示します。

【struts-config.xml】

```
・・・(略)・・・
<plug-in className="org.apache.struts.validator.ValidatorPlugin">
  <set-property property="pathnames"
    value="/WEB-INF/validator-rules.xml,
          /WEB-INF/my-validator-rules.xml,
          /WEB-INF/validation.xml"/>
</plug-in>
・・・(略)・・・
```

独自の検証メソッドの定義方法は以上となります。