Stripes

Stripes & Stripernate

Zbigniew Skowron
13 kwietnia 2007

Stripes & Stripernate

Stripes:

- Framework do budowania aplikacji webowych.
- Założenia:
  prostota i minimum konfiguracji.
- Zbudowany na technologiach Javy 5.0:
  Serlvet 2.4 i JSP 2.0.

Stripernate:

Biblioteka integrująca Stripes'y z Hibernate'm.

Własności

- Konfiguracja poprzez anotacje.
- Automatycznie konfigurowane ActionBean'y.
- Automatyczna walidacja i konwersja typów.
- Indeksowane właściwości.
- Łatwa lokalizacja.
- Obsługa pobierania plików.
- Rozszerzalność.

"JavaDoc, TagDoc and reference documentation that doesn't suck."

Część I: Wprowadzenie

Użycie

Stripes:

Należy umieścić poniższe w classpath:

- /WEB-INF/classes
  - StripesResources.properties
- /WEB-INF/lib
  - stripes.jar
  - commons-logging.jar (1.1)
  - cos.jar (com.oreilly.servlets)

Stripernate:

- stripernate.jar
- jar'y Hibernate'a...

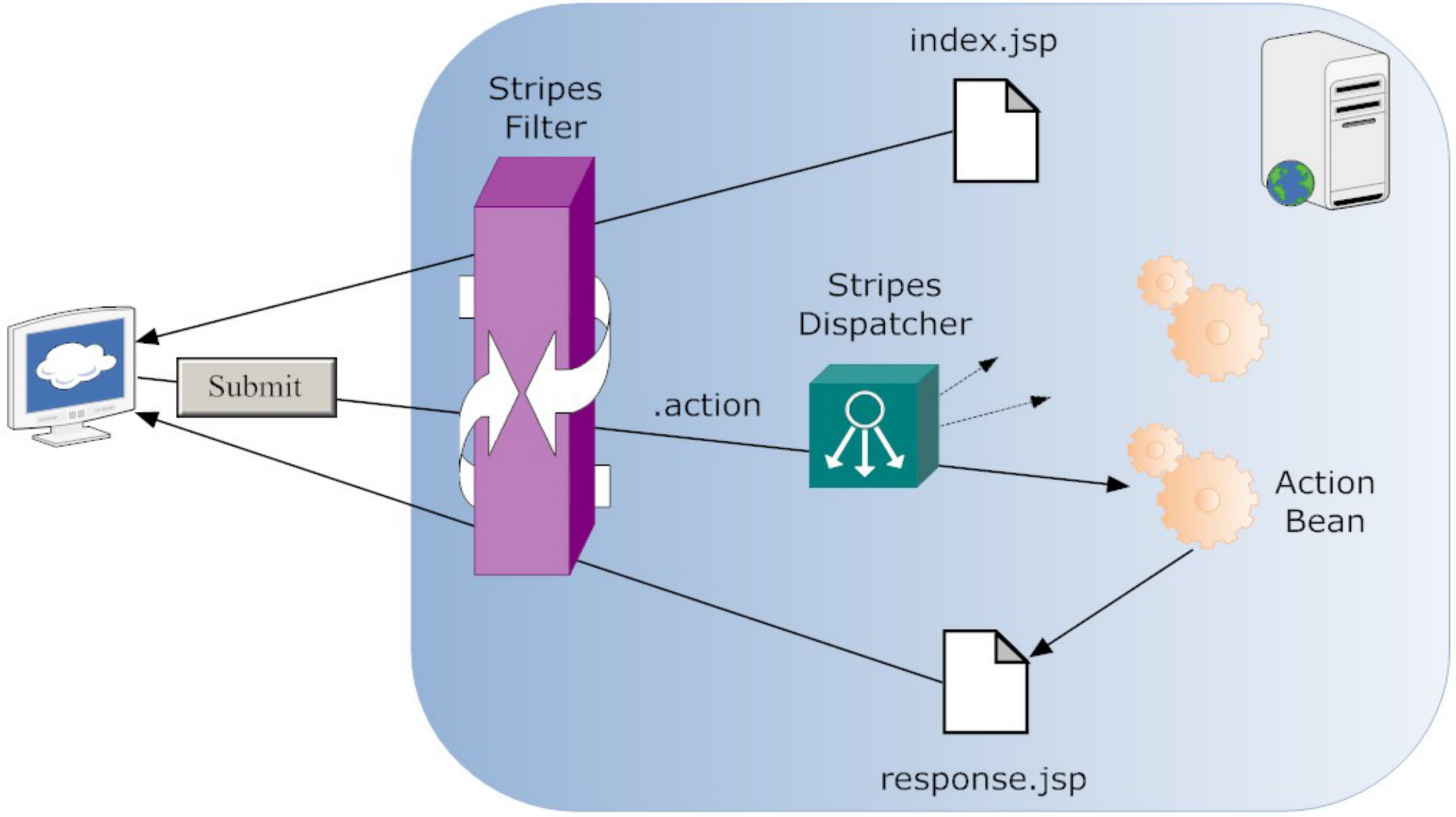
web.xml

```
<filter>
    <display-name>Stripes Filter</display-name>
    <filter-name>StripesFilter</filter-name>
    <filter-class>net.sourceforge.stripes.controller.StripesFilter</...>
</filter>

<filter-mapping>
    <filter-name>StripesFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>

<filter-mapping>
    <filter-name>StripesFilter</filter-name>
    <servlet-name>StripesDispatcher</servlet-name>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>

<servlet>
    <servlet-name>StripesDispatcher</servlet-name>
    <servlet-class>net.sourceforge.stripes.controller.DispatcherServlet</...>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>StripesDispatcher</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

Stripes Filter

Stripes Dispatcher Servlet

Stripes Filter & Dispatcher



Przykład: Calculator.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes.tld"%>

<html>
  <head>
      <title>My First Stripe</title>
  </head>
  <body>
    <h1>Stripes Calculator</h1>

    <stripes:form action="/calc/Calculator.action" focus="">

        <stripes:errors/>

        Number 1:
        <stripes:text name="numberOne"/>
        <br/>
        Number 2:
        <stripes:text name="numberTwo"/>
        <br/>

        <stripes:submit name="addition" value="Add"/>
        <stripes:submit name="division" value="Divide"/>

        Result: ${actionBean.result}

    </stripes:form>
  </body>
</html>
```

Stripes Taglib

Action Bean

Pola Action Bean'a

Metody Action Bean'a

Zwykły EL

ActionBean

```
public class CalculatorActionBean implements ActionBean {
    private ActionBeanContext context;

    public ActionBeanContext getContext() {
        return context;
    }
    public void setContext(ActionBeanContext context) {
        this.context = context;
    }

    private double numberOne;
    private double numberTwo;
    private double result;

    //... gettery i settery pominięte ...


    public Resolution addition() {
        result = getNumberOne() + getNumberTwo();
        return new ForwardResolution("/calc/index.jsp");
    }

    @DefaultHandler
    public Resolution foo() {
        return new RedirectResolution("/calc/foo.jsp");
    }
}
```

Zwykły interfejs

Kontekst

Właściwości

Obsługa zdarzenia "addition"

Obsługa domyślna

Polecenie do wykonania

ActionBeanContext

Kontekst zawiera informacje o bieżącym żądaniu:

```
ServletContext      getServletContext()
HttpServletRequest  getRequest()
HttpServletResponse getResponse()
```

Dostęp do Servlet API

```
String              getEventName()
List<Message>       getMessages()
ValidationErrors    getValidationErrors()
Resolution          getSourcePageResolution()
```

Nazwa przetwarzanego zdarzenia

Pozwala wrócić do wywołującej strony

Znajdowanie ActionBean'a

Stripes sam znajduje klasy implementujące interfejs ActionBean i przypisuje je do URL-i.

Jeśli znajdzie bean'a:

pl.mimuw.www.math.calc.CalculatorActionBean

- Obcina wszystko do pakietu web, www, action lub stripes włącznie.
- Usuwa słowa Action i Bean z końca nazwy klasy.
- Zamienia . na / i dodaje .action
- Przypisuje bean'a do powstałego URL-a:
  /math/calc/Calculator.action

Ręcznie: anotacja dla klasy
@UrlBinding("/qs/calc")

Użycie:

```
<stripes:form action="/math/calc/Calculator.action" focus="">
    ...
</stripes/form>
```

Obsługa zdarzeń

Obsługuje zdarzenie "addition"

Event handler każda metoda publiczna zwracająca Resolution.

```
public Resolution addition() {
    return new ForwardResolution("/calc/index.jsp");
}
```

Obsługuje zdarzenie "subtraction"

```
@HandlesEvent("subtraction")
public Resolution foo() {
    return new RedirectResolution("/calc/index.jsp");
}
```

```
new ForwardResolution("/calc/index.jsp");

new RedirectResolution("/calc/index.jsp");

new JavaScriptResolution(fooObject);

new StreamingResolution("text/xml", new StringReader("<ala/>"));
```

serializuje obiekt do JavaScript'a

wysyła dowolne dane

StreamingResolution & AJAX

```
function invoke(form, event, container) {

      var params = event + '&' + Form.serialize(form);

      new Ajax.Updater(container, form.action,
                       {method:'post', postBody:params});
}
```

Zdarzenie

```
<stripes:button value="Div" name="divide"
         onclick="invoke(this.form, this.name, 'result');"/>

<div id="result"></div>
```

Zdarzenie

```
public Resolution divide() {
        String result = String.valueOf(numberOne / numberTwo);
        return new StreamingResolution("text", result);
}
```

Wysłanie zwykłego napisu

JavaScriptResolution & AJAX

JavaScriptResolution:

- serializuje obiekty Javy do JavaScript'a,
- radzi sobie z cyklami i duplikatami,
- obsługuje także kolekcje (Collection, Map i Array).

```
return new JavaScriptResolution(fooObject);

function update(xmlResponse) {
    var output = eval(xmlResponse.responseText);
    $('result').innerHtml = output;
}

function invoke(form, event) {
    var params = event + '&' + Form.serialize(form);
    new Ajax.Request(form.action, {method:'post', postBody:params,
                                   onSuccess: update});
}
```

Serializacja

Deserializacja

Resolution robione ręcznie

Resolution to zwykły interfejs, mający tylko jedną metodę:

```
public Resolution getLotsOfData() {

    return new Resolution() {

        public void execute(HttpServletRequest request,
                            HttpServletResponse response) throws Exception
        {
            response.setContentType("text/html");

            response.getOutputStream().print("<html>");
            response.getOutputStream().print("Ala ma kota.");
            response.getOutputStream().print("</html>");
        }
    }
}
```

Klasa anonimowa

Jak zwykły servlet

Walidacja

Walidacja jest sterowana przy pomocy anotacji:

```
@Validate(required=true) private double numberOne;
@Validate(maxvalue=25.5) private double numberTwo;
```

Stripes sam już dodaje sprawdzenie, czy wpisane wartości dają się zkonwertować na typ double

Możliwości:

- required=true/false,
- on={events},
- minlength, maxlength, minvalue, maxvalue,
- mask=".*@regex.pl",
- expression="this < elVar"

wyrażenie EL (dostępne sa pola bean'a, request i session scope)

Wyświetlanie komunikatów

Podobnie jak w JSF:

```
<stripes:form action="/calc/calc.action">
    ...
    <stripes:errors/>
    <stripes:messages/>

    <stripes:errors globalErrorsOnly="true"/>
    <stripes:text name="username"/>
    <stripes:errors field="username"/>
    ...
</stipes:form>
<stripes:errors action="/calc/calc.action"/>

<style type="text/css">
    input.error { background-color: red; }
</style>
```

Podobnie jak stripes:errors

Można też umieścić poza formularzem

Każde pole z błędami walidacji dostaje dodatkowo css-ową klasę "error"

Walidacja pól obiektów

Większe możliwości jak w JSF:

`@ValidateNestedProperties`

```
@ValidateNestedProperties({
    @Validate(field="username", required=true, minlength=3),
    @Validate(field="email", mask="[\\w\\.]+@[\\w\\.]+\\.\\w+"),

    @Validate(field="employer.id", required=true),
    @Validate(field="pets.name", required=true, maxlength=30)
 })

private List<Person> people = new ArrayList<Person>();
```

wielokrotne zagnieżdżenie

walidacja wszystkich elementów kolekcji

Walidacja wybiórcza

```
<stripes:submit name="division" value="Divide"/>
```

```
public Resolution division() {
    result = numberOne / numberTwo;
    return new ForwardResolution("/calc/index.jsp");
}
```

Można też bez parametru

```
@ValidationMethod(on="division")
public void avoidDivideByZero(ValidationErrors errors) {
    if(this.numberTwo == 0)
        errors.add("numberTwo",
                   new SimpleError("Dividing by zero is not allowed."));
}
```

Pomiń walidację dla tego zdarzenia

```
@DontValidate
public Resolution redirect() {
    return new RedirectResolution("/calc/index.jsp");
}
```

Ręczna obsługa błędów walidacji

Zwykły interfejs

```
public class CalculatorActionBean implements
                                    ActionBean, ValidationErrorHandler {

    Resolution  handleValidationErrors(ValidationErrors errors) {
        int count = errors.size();

        errors.clear();
        errors.addGlobalError(
            new SimpleError("There were " + count + " errors."));

        return new RedirectResolution("/calc/errors.jsp");
    }
}
```

Można anulować wszystkie błędy

Można przekierować do dowlolnej strony

Konwersje

Stripes automatycznie używa konwerterów.

```
@Validate(converter=MoneyTypeConverter.class)
private Money balance;


<s:link xhref="/Update.action">
    <s:param name="man" value="${man.id}" />
    ${man.name}
</s:link>


public class ManConverter implements TypeConverter<Man>
{
    public void setLocale(Locale locale) {}
    public Man convert(String formVal, Class targetClass,
   Collection<ValidationError> errors)
    {
        return DAO.findById( Long.parseLong( formVal ) );
    }
}
```

Zmiana konwertera

Dla własnych typów można zarejestrować konwerter w fabryce konwerterów...

...ale wystarczy też konstruktor przyjmujący Stringa

Lokalizacja komunikatów

StripesResources.properties zawiera m. in. komunikaty dla błędów:

Można użyć innych źródeł

```
converter.number.invalidNumber=The value ({1}) entered in field {0} must be a valid number
converter.byte.outOfRange=The value ({1}) entered in field {0} was out of the range {2} to {3}
converter.enum.notAnEnumeratedValue=The value "{1}" is not a valid value for field {0}
converter.date.invalidDate=The value ({1}) entered in field {0} must be a valid date
converter.email.invalidEmail=The value ({1}) entered is not a valid email address
```

Można je lokalizować zgodnie z konwencjami Javy.

Lokalizacja komunikatów

```
new ScopedLocalizableError(
    "converter.integer", "outOfRange", ...);
```

Kolejność wyszukiwania:

```
/cats/KittenDetail.action.age.outOfRange
/cats/KittenDetail.action.age.errorMessage
age.outOfRange
age.errorMessage
/cats/KittenDetail.action.outOfRange
/cats/KittenDetail.action.errorMessage
converter.integer.outOfRange
```

Kolekcje obiektów

- Większe możliwości jak w JSF.
- Bezproblemowa integracja z JSTL.

```
private Map<Date, List<Appointment> > appointments;

<stripes:text name="appointments[${date}][${idx}].note"/>
```

Stripes stworzy nowe słowniki i listy w miarę potrzeby

Stripes sam stworzy obiekty, których pola ustawiamy

Wielokrotne zagnieżdżenie

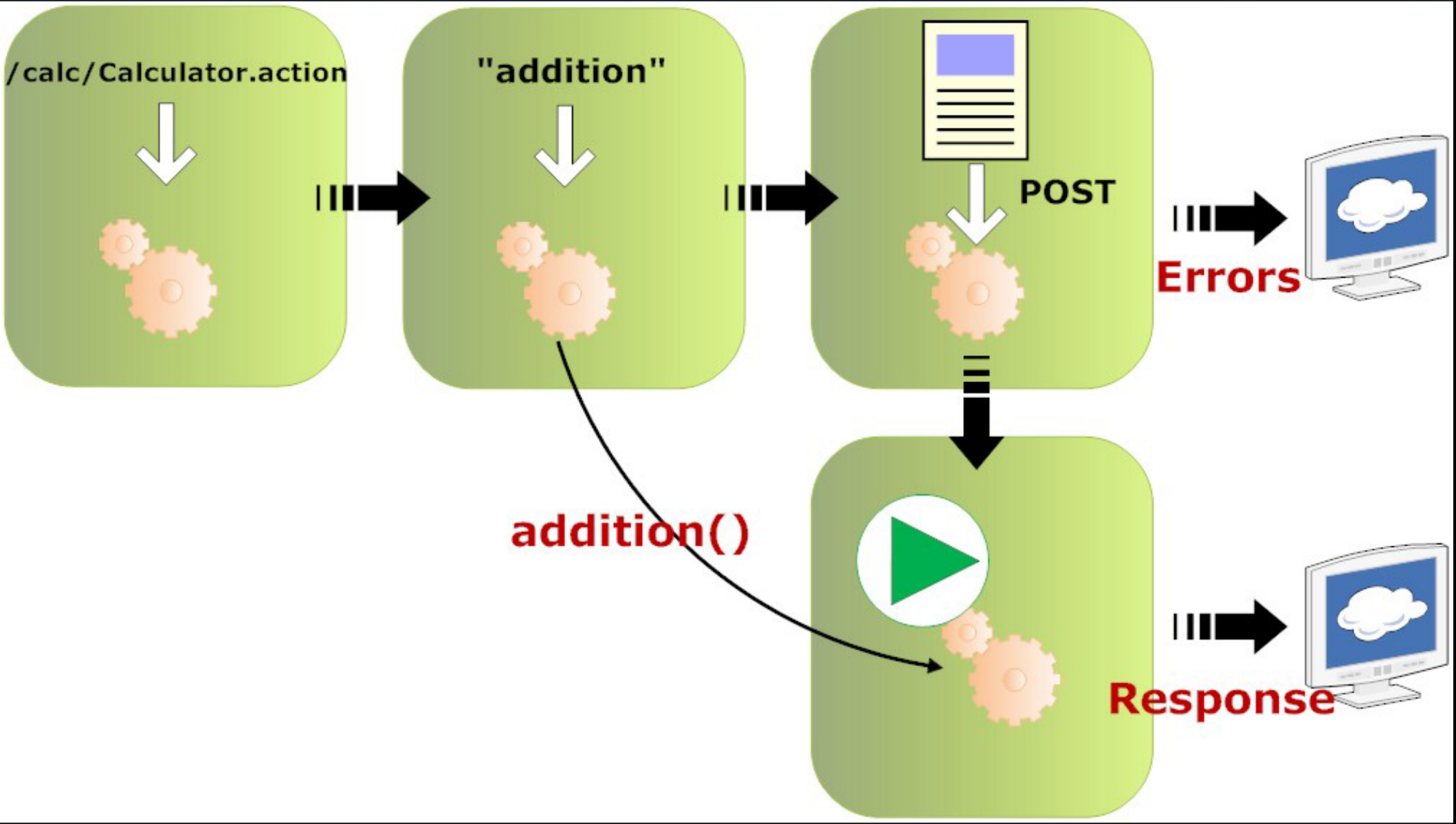
```
private List<Person> people;

<c:forEach items="${actionBean.people}" var="man" varStatus="loop">
    ${man.name}
    <stripes:hidden name="people[${loop.index}].id"/>
</c:forEach>
```

Część II: Szczegóły

Cykl życia: znajdowanie ActionBean'a

1. Dopasuj klasę ActionBean'a do URL'a.
2. Jeśli klasa ma anotację @SessionScope to:
   - zwróć instancję zapisaną w HttpSession.getAttribute(UrlBinding),
   - jeśli jej nie ma, to utwórz nową.
3. Wpw (domyślnie):
   - utwórz nową instancję ActionBean'a.
4. Stwórz kontekst bean'a i wywołaj setContext().
5. Zapisz ActionBean'a w odpowiednio request lub session pod kluczem UrlBinding.
6. Zapisz ActionBean'a w request pod kluczem 'actionBean'.

Cykl życia ActionBean'a



Cykl życia: przypisania i walidacja

@Validate(required=true)

1. Sprawdzenie czy wartości wymagane są ustawione.
2. Walidacja przed konwersją: minlength, maxlength, mask...
3. Konwersja na docelowy typ.
4. Przypisanie polom ActionBean'a nowych wartości.
5. Walidacja po konwersji: minvalue, maxvalue, expression...
6. Jeśli były błędy to przerwij.
7. Uruchom walidację użytkowika.
8. Jeśli były błędy to przerwij.

Błędy są zgromadzone w ActionBeanContext.getValidationErrors()

Interceptory

Może być też @After

```
@Before(LifecycleStage.BindingAndValidation)
public void rehydrate() {
    this.domainObject = getHibernateSession().load(DomainObject.class,
                                context.getRequest().getParameter("id"));
}

@Intercepts({LifecycleStage.ActionBeanResolution,
             LifecycleStage.HandlerResolution,
             LifecycleStage.BindingAndValidation,
             LifecycleStage.CustomValidation,
             LifecycleStage.EventHandling,
             LifecycleStage.ResolutionExecution})
public class NoisyInterceptor implements Interceptor {
    public Resolution intercept(ExecutionContext ctx) throws Exception {
        System.out.println("Before " + ctx.getLifecycleStage());
        Resolution resolution = ctx.proceed();
        System.out.println("After " + ctx.getLifecycleStage());
        return resolution;
    }
}
```

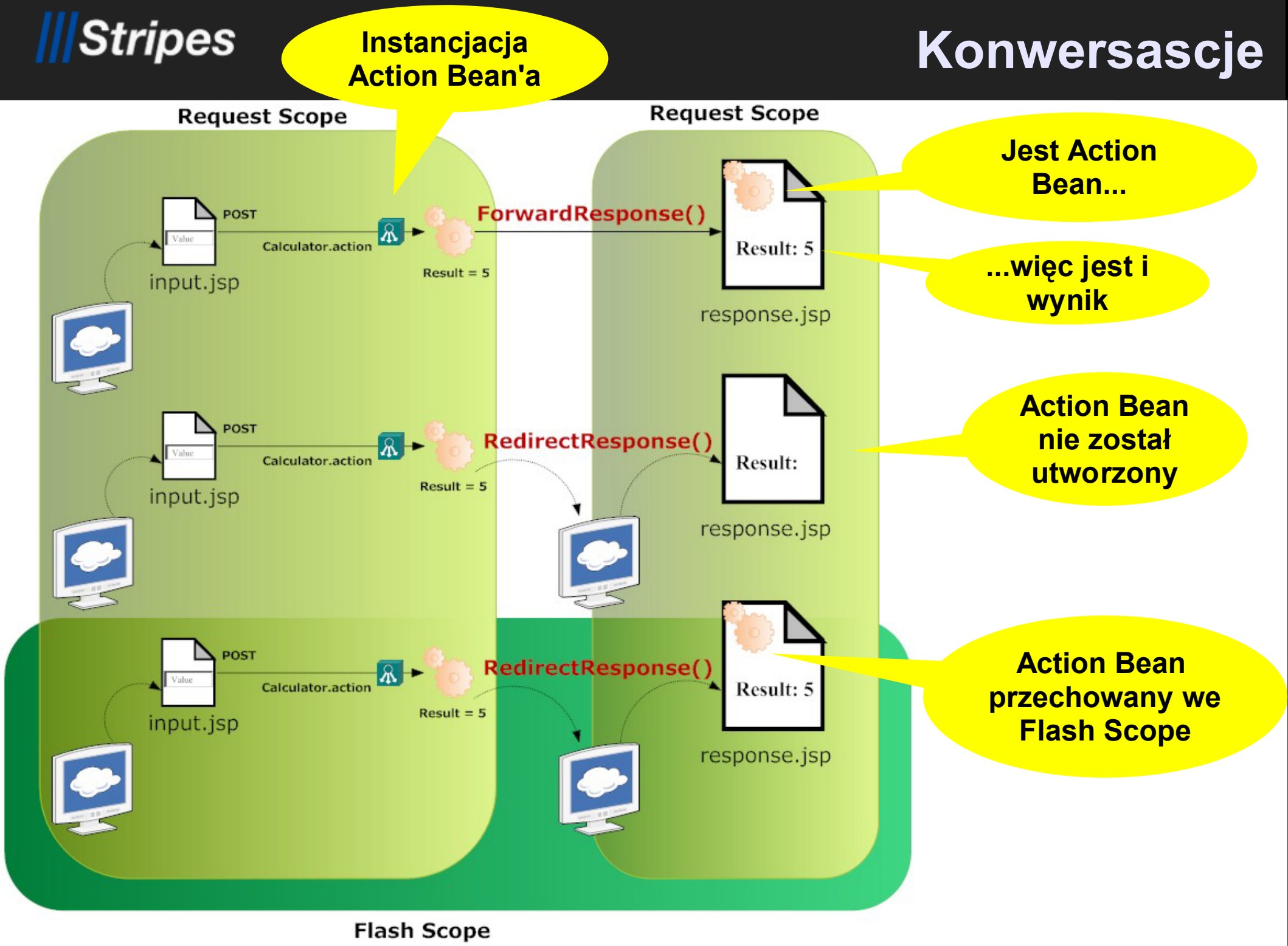
Etapy życia

Można przerwać cykl życia zwracając resolution != null

Parametry dla StripesFilter

```
<init-param>
    <param-name>Interceptor.Classes</param-name>
    <param-value>
        com.myco.NoisyInterceptor,
        net.sourceforge.stripes.controller.BeforeAfterMethodInterceptor
    </param-value>
</init-param>
```

Konwersascje



Flash Scope

Flash Scope istnieje w ciągu bieżącego i następnego żądania.

- Pozwala się uniezależnić od różnicy pomiędzy ForwardResolution() i RedirectResolution().
- Nie psuje się przy używaniu wielu okien przeglądarki.
- Zaimplementowany jako tymczasowy obiekt w Session Scope.

```
RedirectResolution("/some/page.jsp").flash(this);
FlashScope.getCurrent(getContext().getRequest(),
  true).put(this);
```

Wizard

Lista zdarzeń startowych

```
@Wizard(startEvents="begin")
public class RegisterActionBean extends BugzookyActionBean {

    public Resolution begin() {
        return new RedirectResolution("/bugzooky/Register.jsp");
    }

    public Resolution gotoStep2() throws Exception {
        return new ForwardResolution("/bugzooky/Register2.jsp");
    }

    public Resolution register() {
        new PersonManager().saveOrUpdate(this.user);
        getContext().setUser(this.user);
        getContext().getMessages().add(
            new LocalizableError("/bugzooky/Register.action.successMessage",
                                 this.user.getFirstName(),
                                 this.user.getUsername()));

        return new RedirectResolution("/bugzooky/BugList.jsp");
    }
}
```

Przetwarzanie formularza podzielone na dwa zdarzenia

Alternatywnie:

```
<wizard-fields/>
```

Wpisuje do strony jako 'hidden' wszystkie pola z żądania, które nie mają odpowiadającego pola formularza

Użyteczne znaczniki

```
<stripes:useActionBean binding="/db/Blob.action"/>

<stripes:hidden name="numberOne">


<link href="/my/actions/cannon" event="fire">
    <param name="yaw" value="north-east-north"/>
    <param name="pitch" value="45 deg"/>
</link>



<stripes:select name="bugs[${loop.index}].component.id">
    <stripes:option value="">Select One</stripes:option>
    <stripes:options-collection
            collection="${componentManager.allComponents}"
            label="name" value="id"/>
</stripes:select>



<stripes:select name="bugs[${loop.index}].priority">
    <stripes:option value="">Select One</stripes:option>
    <stripes:options-enumeration
            enum="net.sourceforge.stripes.examples.bugzooky.biz.Priority"/>
</stripes:select>
```

Utworzenie ActionBean'a dla strony

Link do zdarzenia

Wybór elementu kolekcji

Wybór wartości wyliczeniowej

Przesyłanie plików

```
<stripes:file name="newAttachment"/>
```

Znacznik

```
private FileBean newAttachment;
```

Pole ActionBean'a

Można albo tak:

```
FileBean.save(File file);
```

Albo tak:

```
FileBean.getInputStream();
FileBean.delete();
```

Nie można łączyć tych sposobów

Stripernate = Stripes + Hibernate

Dostępne dla każdego żądania HTTP

```
HibernateProvider.getInstance().getSession();
HibernateProvider.getInstance().commit();
```

Dla bezpieczeństwa trzeba wykonywać ręcznie.

- **HibernateFilter** - wyszukuje obiekty zaanotowane jako **@Entity** i udostępnia sesję Hibernate'a.
- **HibernateInterceptor** - inicjalizuje sesję Hibernate'a.
- **HibernatePropertyBinder** - łapie wyjątki walidacji Hibernate'a i wycofuje transakcje.
- **HibernateTypeConverter** - konwersja id -> obiekt.
- **HibernateFormatter** - konwersja obiekt -> id.

Łączenie formularzy z bazą

Jeśli request będzie miał parametr **user=5**, to Stripernate wyciągnie z bazy User'a o **id=5** i przypisze na pole user.

```
@Entity
class User {
    @Id
    int id;
}

class ActionBean {
    User user;
}
```

Stripernate znajdzie tą klasę automatycznie

Wystarczy jej użyć jako pole ActionBean'a

Koniec

Źródła:

http://stripes.mc4j.org/

http://www.mongus.com/Topics/Web%20Development/Stripernate/