Stripes Framework

...in a comparison with Struts!

Johannes Carlén
johannes.carlen@callistaenterprise.se
www.callistaenterprise.se

CALLISTA

Let's talk about Struts for a minute...

- Pros
  - De facto standard for a couple of years
  - Simple and powerful concept. Easy to understand
  - Stable and mature
  - Documentation. Internet and books
  - Easy to find skilled developers
- Cons
  - Simpler then than now
  - struts-config.xml (form beans, global forwards, action mappings with forwards, message resources, plugins, etc..)
  - Validation
  - Property Binding
  - Confusing JSP-tags
  - Layout

CALLISTA

Ok, so what is this Stripes thing?

- Around since 2005 (Tim Fennell)
- Built upon the same design pattern (MVC) and concepts as Struts.
- An action based framework
- Exisiting skills in Struts is easily transferred to Stripes – no new concepts to learn
- Claim: ”...makes developing web applications in Java easy”

CALLISTA

Tell me in a language I understand, please

- Coding by convention & Annotations
- No external configuration
- URL Binding
- Event Handlers (multiple events form)
- Property Binding / Type Conversion (nested properties)
- Interceptors for cross cutting concerns

CALLISTA

And what's in the box?

- Validation mechanisms
- Exception handling
- JSP-tags
- Layout
- Localization
- Testing (Out of container with mock objects)
- Easy to extend

CALLISTA

The fundamentals

- **Action Beans & Event Handlers**
- URL Binding
- Validation
- Type Conversion and Formatters
- JSP Tags & Layout

CALLISTA

The Struts way - Actions

Action Class

```
public class CustomerAction extends DispatchAction {

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

     CustomerForm customerForm = (CustomerForm) form;
    // Validate form
    if (!isValid(form, mapping, request {
        CustomerForm.
        setCustomers(customerService.findAllCustomers());
      return mapping.getInputForward();
    }
    Customer customer = (Customer)    customerForm.
                          getCustomerBean().extract();
       customerService.saveCustomer(customer,
NumericUtils.toLong(customerForm.getCustomerId()));
      return mapping.findForward(FORWARD_SUCCESS);
}
```

Form Bean

```
public class BaseForm extends ValidatorForm {
private CustomerBean customer;
private String startDate;
public CustomerBean getCustomer() {return customer;}
public void setCustomer(CustomerBean customer)
{this.customer=customer;}
public String getStartDate() {return startDate;}
public void setStartDate(String date){this.startDate=date;}
}
```

View helper bean

```
public class CustomerBean {
private String id;
private String name;
private String email;

public String getId() {return id;}
public void setId(String id) {this.id=id;}
public String getName() {return name;}
public void setName(String name) {this.name=name;}
public String getEmail() {return email;}
public void setEmail(String email) {this.email=email;}
}
```

struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <form-beans>
    <form-bean name="customerForm" type="se.callistaenterprise.exampleweb.CustomerForm" />
    </form-beans>
    <global-exceptions>
    <exception type="se.callistaenterprise.examplecommon.BaseException"
                    handler="se.callistaenterprise.exampleweb.action.BaseExceptionHandler"
key="errors.general"/>
    </global-exceptions>
    <global-forwards>
    <forward name="start" path="/start.do" redirect="true"/>
    </global-forwards>
    <action-mappings>
                    <action path="/customer"
                    type="org.springframework.web.struts.DelegatingActionProxy"
                    scope="request"
                    parameter="task"
                    input=".customer.save"
                    name="customerForm"
                    validate="false">
                    <forward name="success" path="/customer/customer.do?task=edi
                     redirect="true" />
                    <forward name="failure_redirect" path="/customer/list.do" redirect="true" />
    </action>
    </action-mappings>
    <plug-in className="org.apache.struts.tiles.TilesPlugin">
        <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
        <set-property property="moduleAware" value="true" />
        <set-property property="definitions-parser-validate" value="true" />
    </plug-in>
</struts-config>
```

CALLISTA

The fundamentals – Action Beans

- Handles an action **and** encapsulates the model beans
- Implements the ActionBean interface:

```
public interface ActionBean {
    public ActionBeanContext getContext();
    public void setContext(ActionBeanContext context);
}
```


CALLISTA

The Struts way – Dispatch Actions

```
public class CustomerAction extends DispatchAction {

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

    CustomerForm customerForm = (CustomerForm) form;
    if (!isValid(form, mapping, request {
      CustomerForm.setCustomers(customerService.findAllCustomers());
      return mapping.getInputForward();
    }
    Customer customer = (Customer) customerForm.getCustomerBean().extract();
    customerService.saveCustomer(customer,customerForm.getCustomerId()));
    return mapping.findForward(FORWARD_SUCCESS);
}
```


CALLISTA

The fundamentals – Event Handlers

- Stripes responds to a request and invokes a method in the action bean – an event handler.
- Action Bean method
  - has no parameters and returns a *Resolution*

```
public class CustomerActionBean implements ActionBean {
    @Validate
    private Customer customer;
    public Resolution save() {
        customerManager.save(this.customer);
        return new ForwardResolution("/WEB-INF/pages/customers.jsp");
    }
    ...
}
```


CALLISTA

Resolutions

- Tells Stripes what to do next in a request
- Examples:

```
new ForwardResolution(”/WEB-INF/pages/my.jsp”);

new ForwardResolution(AnotherActionBean.class,”someEvent”);

new RedirectResolution(”/some/other.jsp”);

new StreamingResolution(”application/pdf”,myStream);
```


CALLISTA

The fundamentals

- Action Beans & Event Handlers
- **URL Binding**
- Validation
- Type Conversion and Formatters
- JSP Tags & Layout

CALLISTA

The Struts way – URL Binding

struts-config.xml

```
<action-mappings>
   <action path="/customer"
     type="org.springframework.web.struts.DelegatingActionProxy"
     scope="request"
     parameter="task"
     input=".customer.save"
     name="customerForm"
     validate="false">
     <forward name="success" path="/customer.do?task=edit" redirect="true" />
     <forward name="failure" path="/customer.do?task=list" redirect="true" />
     </action>
 </action-mappings>
```


CALLISTA

The fundamentals - URL Binding

- **By convention**

  The action bean class:

  `se.callistaenterprise.web.`**`store.CustomerActionBean`**

  ...defaults to:

  http://myserver/mycontext/**store/Customer.action**

CALLISTA

The fundamentals - URL Binding

- **Override using @UrlBinding**

```
@UrlBinding(”/customer/{$event}.html”)
public class CustomerAction implements ActionBean {
    public Resolution save() {
    ...
    }
}
```

  - Use this URL to trigger event:

    http://myserver/mycontext/**customer/save.html**

CALLISTA

The fundamentals

- Action Beans & Event Handlers
- URL Binding
- **Validation**
- Type Conversion and Formatters
- JSP Tags & Layout

CALLISTA

The Struts way – Validation

- Validation in action method, form bean, validator.xml and validator-rules.xml

Action Class

```
public class CustomerAction extends DispatchAction {

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

      CustomerForm customerForm = (CustomerForm) form;
     // Validate form
     if (!isValid(form, mapping, request {
          CustomerForm.
          setCustomers(customerService.findAllCustomers());
       return mapping.getInputForward();
     }
     Customer customer = (Customer)     customerForm.
                          getCustomerBean().extract();
        customerService.saveCustomer(customer,
NumericUtils.toLong(customerForm.getCustomerId()));
       return mapping.findForward(FORWARD_SUCCESS);
}
```

Form Bean

```
public class CustomerBeann extends ActionForm {

public ActionErrors validate(ActionMapping mapping,
                             HttpServletRequest request) {
    ActionErrors actionErrors = new ActionErrors();
    // validate credit
    if (credit > Credit.LIMIT) {
       actionErrors.add("credte", new ActionMessage("error.name"));
    }
    return actionErrors;
}
...
```

validator-rules.xml

```
<validator name="email"
               classname="org.apache.struts.validator.FieldChecks"
               method="validateEmail"
               methodParams="java.lang.Object,
               org.apache.commons.validator.ValidatorAction,
               org.apache.commons.validator.Field,
          org.apache.struts.action.ActionMessages,
          org.apache.commons.validator.Validator,
          javax.servlet.http.HttpServletRequest"
       depends="" msg="errors.email" />
```

validator.xml

```
<form name="customerForm">
    <field property="customerBean.customerId" depends="required">
        <arg name="required" key="action.customer.customerId" position="0" />
    </field>
    <field property="customerBean.customer.namn" depends="required">
        <arg name="required" key="action.customer.namn" position="0" />
    </field>
    <field property="customerBean.customer.email" depends="email">
        <arg0 key="action.customer.email" />
    </field>
</form>
```

CALLISTA

The fundamentals - Validation

- Using @Validate

  (field,required,on,minlength,maxlength,expression,mask,minvalue,maxvalue, converter,trim,label,ignore,encrypted)

```
@Validate(required=true,minlength=8, maxlength=32)
private String password

@Validate(required=true, expression=”this>=startDate”,
          converter=DateConverter.class)
private Date endDate;

@ValidateNestedProperties({
    @Validate(field=”email” required=true, on=”save”),
    @Validate(field=”name” requried=true, on=”save”)})
private Customer customer;
```


CALLISTA

The fundamentals - Validation

- Custom validation with @ValidationMethod

```
@ValidationMethod(on="save")
public void validateCustomerUsername() {
    if(customerManager.usernameExists(customer.getUsername)){
        // Add validation error
    }
}
```


CALLISTA

The fundamentals

- Action Beans & Event Handlers
- URL Binding
- Validation
- **Type Conversion and Formatters**
- JSP Tags & Layout

CALLISTA

The Struts way – Type conversion

- Conversion in action class.

CustomerAction.java

```
...
CustomerForm customerForm = (CustomerForm) form;
Customer customer = new Customer();
float credit = Float.parseFloat(customerForm.getCredit());
c.setCredit(credit);
...
```

Customer.java

```
private float credit;
public float getCredit() {return credit;}
public void setCredit(float credit) {this.credit=credit;}
```


CALLISTA

The fundamentals – Type Conversion

- Used for parameter binding
- Built in converters for standard types (numericals, booleans, enums, email, credit card, one-to-many)
- Write your own custom converter by implementing the TypeConverter<T> interface

CALLISTA

The fundamentals – Type Conversion

- Example - converting String to PhoneNumber

MyActionBean.java

```
@Validate(required=true, converter=PhoneNumberConverter.class)
private PhoneNumber phoneNumber;
```

PhoneNumberConverter.java

```
public class PhoneNumberConverter implements TypeConverter<PhoneNumber> {
    @Override
    public PhoneNumber convert(String input, Class<? extends PhoneNumber> targetType,
                               Collection<ValidationError> errors) {
        try {
            PhoneNumber phoneNumber = new PhoneNumber(input);
            return phoneNumber;
        } catch (IllegalArgumentException e) {
            errors.add(new ScopedLocalizableError("converter.phonenumber",
                                                  "invalidPhoneNumber"));
            return null;
        }
    }
    ...
}
```

The fundamentals – Formatters

- Type conversion in the opposite direction
- An object is converted to a String to be displayed to the user
- Locale-sensitive
- Write a custom formatter by implementing the Formatter interface
- Best practice - use the same class to implement both TypeConverter and Formatter

CALLISTA

The fundamentals

- Action Beans & Event Handlers
- URL Binding
- Validation
- Type Conversion and Formatters
- **JSP Tags & Layout**

CALLISTA

The Struts way – JSP tags

- Example of a simple JSP:

editCustomer.jsp

```
<html:form action="/customer">
  <html:errors />
  <html:hidden property="customerId" />
  <bean:message key=”customer.company” />
  <html:select property="customerCompanyId">
    <hmtl:optionsCcollection property="companies" value="id" label="name" />
  </html:select>
  <bean:message key=”customer.name.and.phone” />
  <html:text property="customer.name"/>
  <html:text property="customer.phoneNumber"/>
</html:form>
```


CALLISTA

The Fundamentals - JSP tags

- JSP tags equivalent to Struts HTML tags
- Uses the name attribute to bind a value to an action bean property
- Labels from resource bundles
- All HTML attributes

editcustomer.jsp

```
<stripes:form action="/customer.html">
  <stripes:errors />
  <stripes:hidden name="customer.id" />
  <stripes:select name="customer.company.id">
    <stripes:options-collection collection="${actionBean.companies}"
                                value="id" label="name" />
  </stripes:select>
  <stripes:text name="customer.name" style=”customerLabels” />
  <stripes:text name="customer.phoneNumber" />
</stripes:form>
```

CALLISTA

The Tiles way – JSP layout

tiles-def.xml

```
<tiles-definitions>
    <definition name=".layout" path="/WEB-INF/pages/layout/layout.jsp">
        <put name="menu" value="/WEB-INF/pages/topmenu.jsp"/>
        <put name="body" value="/WEB-INF/pages/start.jsp"/>
    </definition>
    <definition name=".customer.view" extends=".layout">
        <put name="body" value="/WEB-INF/pages/customer/view.jsp"/>
    </definition>
    <definition name=".customer.edit" extends=".layout">
        <put name="body" value="/WEB-INF/pages/customer/edit.jsp"/>
    </definition>
<tiles-definitions>
```

struts-config.xml

```
<plug-in
className="org.apache.struts.tiles.TilesPlugin">
        <set-property property="definitions-config"
value="/WEB-INF/tiles-defs.xml" />
        <set-property property="moduleAware"
value="true" />
        <set-property property="definitions-parser-
validate" value="true" />
</plug-in>

<action path="/customer"
type="org.apache.struts.actions.ForwardAction"
        parameter="homepage"/>
```

layout.jsp

```
<html>
<body>
    <tiles:insert attribute="menu"/>
    <tiles:insert attribute="body"/>
</body>
</html>
```

view.jsp

```
<h1>This page outputs the customer</h1>
```

edit.jsp

```
<h1>On this page you can edit the
customer</h1>
```

CALLISTA

The Fundamentals - JSP layout

- No configuration
- Concepts: Layout, Renderer, Component, Attributes
- Three tags: `layout-definition`, `layout-render`, `layout-component`

layout.jsp

```
<stripes:layout-definition>
  <h1>${title}</h1>
  <stripes:layout-component  name="${menu}"/>
          My default menu</stripes:layout-component>
  <stripes:layout-component  name="body"/>
</stripes:layout-definition>
```

view.jsp (renderer)

```
<stripes:layout-render name="/WEB-INF/pages/layout.jsp"
                       title="View customer">
  <stripes:layout-component  name="body"/>
          Output the customer..</stripes:layout-component>
</stripes:layout-render>
```

CALLISTA

More features

- **Web Flows**
- Interceptors
- Localization, Messages and Validation Errors
- Testing

CALLISTA

The Struts Way – No flow

- No mechanism in Struts for wizard-like functionality
- Alternatives: Struts Flow (sandbox), Spring Web Flow
- ..or by hand

CALLISTA

The Stripes Wizard

- The @Wizard annotation
  - generates hidden inputs for already submitted fields
  - keeps track of which fields are to be validated for each step

```
@Wizard
public class CustomerActionBean implements ActionBean {
    public Resolution view() {...}
    public Resolution save() {...}
}
```


CALLISTA

More features

- Web Flows
- **Interceptors**
- Localization, Messages and Validation Errors
- Testing

CALLISTA

The Struts way – Interceptors

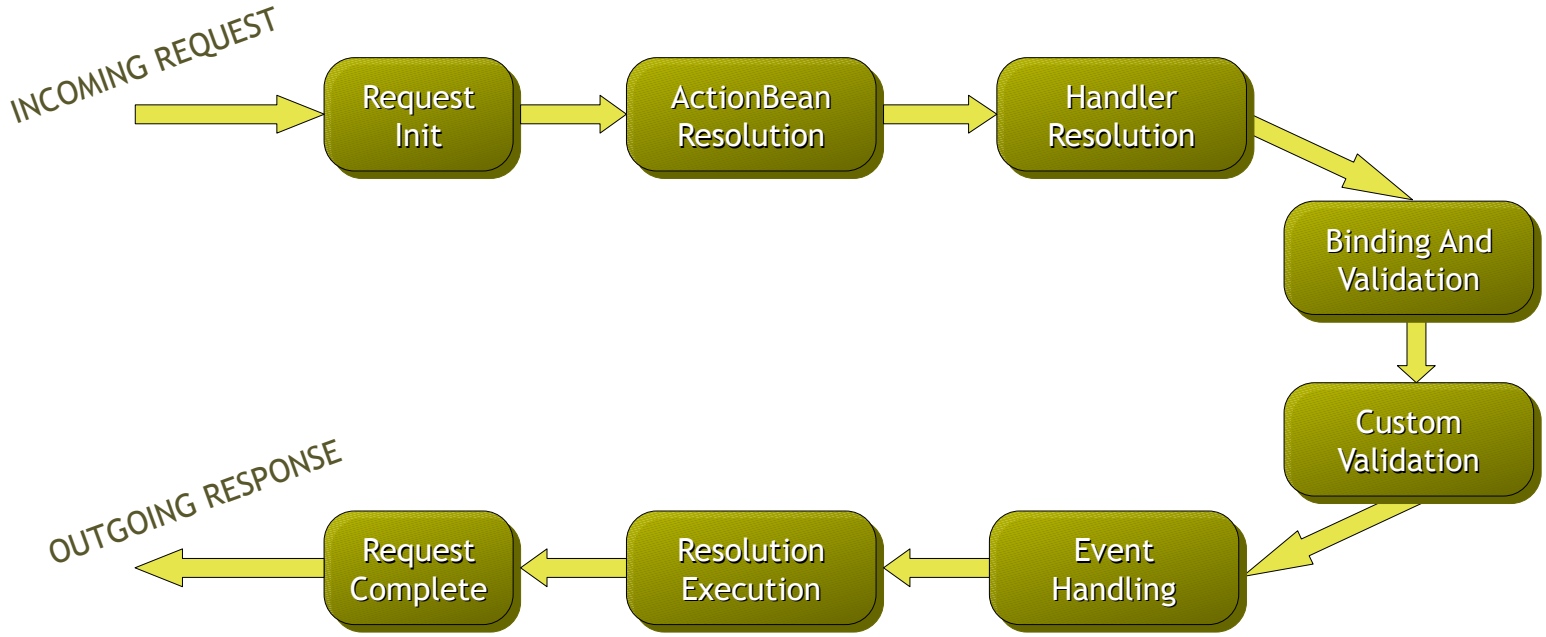
- No real equivalent functionality in Struts

CustomerAction.java

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

   CustomerForm customerForm = (CustomerForm) form;
    customerForm.setCustomer(customerService.findCustomer(customerForm.getId()));
   return mapping.findForward(FORWARD_SUCCESS_VIEW);
}

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

   CustomerForm customerForm = (CustomerForm) form;
   Customer customer = (Customer) customerForm.getCustomerBean().extract();
   customerService.saveCustomer(customer);
   customerForm.setCustomer(customerService.findCustomer(customerForm.getId()));
   return mapping.findForward(FORWARD_SUCCESS_VIEW);
}
```

Interceptors

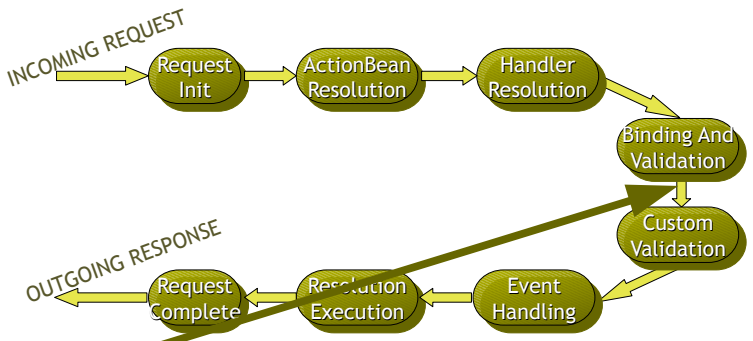
- Code that is executed before and/or after a life cycle stage
- Two ways to intercept:
  - Before / After methods (applies to a specific action bean)
  - Global interceptor (intercepts all requests at given stages)



CALLISTA

Interceptors

- Before and after methods
  - Add a method to an action bean
  - Annotate with @Before or @After

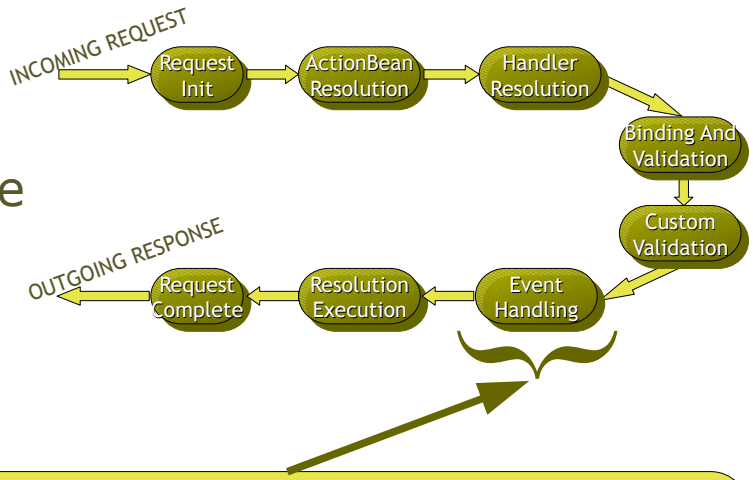



```
@After(stages = LifecycleStage.BindingAndValidation, on = {"view","edit"})
public void populateCustomer() {
    customer = customerManager.findById(customer.getId());
}
```


CALLISTA

Interceptors

- Global Interceptor
  - Implement the Interceptor interface and indicate life cycle stages




```
@Intercepts(LifecycleStage.EventHandlingResolution)
public class AuditLogInterceptor implements Interceptor {
    public Resolution intercept(ExecutionContext context) throws Exception
    {
        logEventStarted(context);
        Resolution resolution = context.proceed();
        logEventCompleted(context);

        return resolution;
    }
}
```

More features

- Web Flows
- Interceptors
- **Localization, Messages and Validation Errors**
- Testing

CALLISTA

The Struts way – I18n and messages

- ApplicationResources.properties – single resource bundle
- Use the struts-bean taglib:

```
<bean:message key="app.name" />
```

- Use <html:errors/> and <html:messages/> to display validation errors and messages

editCustomer.jsp

```
<html:form action="/customer">
  <html:errors /><br />
  <bean:message key=”customer.label.name” />
  <html:text property="customer.name"/>
</html:form>
```


CALLISTA

Localization, Messages & Validation Errors

- Localization relies on standard ResourceBundles
- JSTL with <fmt:message> for text
- Application messages and validation errors by key:

CustomerActionBean.java

```
@ValidationMethod(on=”save”)
public void validateCustomerUsername() {
    if(customerManager.usernameExists(customer.getUsername)){
        getContext.getValidationErrors.
          add(new LocalizableError(”customer.invalid.username”,customer.getUsername());
    }
}
```

editCustomer.jsp

```
<stripes:form action="/customer.html">
  <stripes:errors /><br />
  <fmt:message key=”customer.name” />
  <stripes:text name="customer.name" style=”customerLabels” />
</stripes:form>
```

More features

- Web Flows
- Interceptors
- Localization, Messages and validation errors
- **Testing**

CALLISTA

The Struts way – Out of container testing

- Struts TestCase on Sourceforge
- Mock objects
- Unit tests extends MockStrutsTestCase

```
public class TestCustomerAction extends MockStrutsTestCase {

    public TestCustomerAction(String testName) { super(testName); }

    public void testSave() {
       setConfigFile("mymodule","/WEB-INF/struts-config.xml");
       setRequestPathInfo("/customer.do");
       addRequestParameter("task","save");
       addRequestParameter("customer.email","NO_VALID_EMAIL_ADRESS");
       actionPerform();
       verifyForward("edit");
       verifyActionErrors(new String[] {"error.invalid.email"});
    }
}
```


CALLISTA

Automated testing

- What?
  - Submit a form, validating response with any validation errors
  - Type conversion
  - URL Bindings
  - Interceptors
- How?
  - Using mock objects for Session, Request, Response, ServletContext objects etc)
  - MockRoundtrip simulates requests to action beans

CALLISTA

Test example

```
@Test
public void testEmailRequired() throws Exception {

    MockRoundtrip trip = new MockRoundtrip(mockServletContext,
                                           CustomerActionBean.class);
    trip.setParameter("customer.email", "NO_VALID_EMAIL_ADRESS");
    trip.execute("save");

    assertEquals(1, trip.getValidationErrors().size());
}
```


CALLISTA

Wrapping up

- Easy but still powerful MVC framework without the Struts caveats
- Short learning curve for existing Struts developers
  - Manage in hours – control in a couple of days
- 30% less code (at least)
- Simplicity saves time
- Good documentation
- And yes the Stripes team's claim holds – Stripes really makes web application development easier.

CALLISTA

As simple as that...

Q&A

Johannes Carlén
johannes.carlen@callistaenterprise.se
www.callistaenterprise.se

CALLISTA