



Painless Java Web Development



About me

- **Cosmin Marginean**, senior Java Developer at Evozon Systems (evozon.com).
- **Technical background**: mostly Java, but also some C/C++, .NET and others
- **Open source enthusiast and contributor**
- **Contact**: cosminaru@gmail.com

Why another Java Web framework?

Because popular doesn't always mean the best.

Because modern applications need modern solutions based on the most powerful features of the Java platform.

Because we need to easily integrate with the standards.

Because XML is over-rated

And last, but not least: Because it's good to learn some new stuff every once in a while.

Stripes - stripesframework.org

Key notes

- **An annotation-centric, MVC Java web framework.**
- **Author and lead developer: Tim Fennell (<http://www.jroller.com/tfenne/>)**
- **Most concepts are similar to other frameworks (Struts, WebWork, etc) => small ramp up time / efficient learning curve for both seniors and newbies.**
- **Based on many auto-discovery features.**
- **Zero configuration.**
- ***Tons* of defaults out of the box.**
- **Extensibility/plugability as easy as you can get.**
- **Extremely compact (one ~400KB .jar and an additional ~56Kb dependency)**
- **Open source, Apache 2.0 licensed**



OK. It sounds both interesting and buzzy. Let's see which one is it!

We'll go through the following

- **Setting up**
- **Mapping action beans**
- **Event handling**
- **Binding and validation**
- **Execution intercepting**
- **Integrating (with) 3rd party tools/frameworks**
- **Adding your own extensions**

/// Setting up Stripes

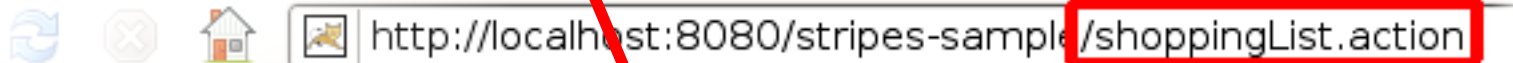
```
<filter>
  <filter-name>StripesFilter</filter-name>
  ...
  <init-param>
    <param-name>ActionResolver.Packages</param-name>
    <param-value>org.example.stripes.web</param-value>
  </init-param>
</filter>
...
<servlet>
  <servlet-name>StripesDispatcher</servlet-name>
  ...
  <load-on-startup>1</load-on-startup>
</servlet>

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

/// Mapping action beans “manually”

```
package org.example.stripes.web.shopping;
...

@UrlBinding("/shoppingList.action")
public class ShoppingListAction implements ActionBean {
    ...
}
```



http://localhost:8080/stripes-sample/shoppingList.action

```
<stripes:form action="/shoppingList.action"/>
...
</stripes:form>
```

/// Default mappings for action beans

```
<param-name>ActionResolver.Packages</param-name>  
<param-value>org.example.stripes.web</param-value>
```

```
package org.example.stripes.web.shopping;  
...  
public class ShoppingListAction implements ActionBean {  
    ...  
}
```

```
<stripes:form action="/shopping/ShoppingList.action"/>  
    ...  
</stripes:form>
```

/// Neat, but what's that ActionBean interface?

- A marker interface used by Stripes to discover action implementations.
- A way for the user code to interact with the state of the current request/session, through the ActionBeanContext entity.

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

```
public class ActionBeanContext {  
    public HttpServletRequest getRequest() {...}  
    public ServletContext getServletContext() {...}  
    public String getEventName() {...}  
    public ValidationErrors getValidationErrors() {...}  
    ...  
}
```

/// Event handlers and resolutions

```
@UrlBinding("/shoppingList.action")
public class ShoppingListAction implements ActionBean {
    @DefaultHandler
    public Resolution listItems() throws Exception {...}

    @HandlesEvent("save")
    public Resolution addNewItem() throws Exception {...}
}
```

- **Event** - in most cases the equivalent of a request (ex: a form submit).
- **Event handler** - a method responsible for handling a certain type of request (associated with a certain event name).
- **Resolution**
 - the result of handling an event
 - responsible for defining the view (in the MVC sense).
 - commonly consists in a forward or redirect to a JSP view.

/// Submitting to an event handler

```
@HandlesEvent("save") public Resolution addItem() {..}
```

```
<stripes:form action="/shopping/ShoppingList.action"/>
  ...
  <input type="submit" name="save" ...
  <!-- OR -->
  <input type="hidden" name="_eventName" value="save"/>
</stripes:form>
```

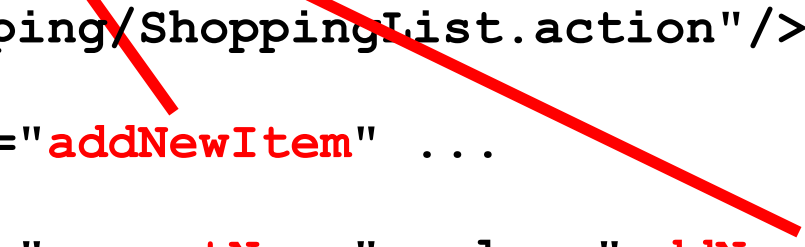
```
<stripes:form action="/shopping/ShoppingList.action"/>
  ...
  <input type="submit" value="Update"/>
</stripes:form>
```

```
@DefaultHandler
public Resolution listItems() {...}
```

/// I smell more fancy-shmancy defaults...

```
public Resolution addNewItem() {...}
```

```
<stripes:form action="/shopping/ShoppingList.action"/>  
  ...  
  <input type="submit" name="addNewItem" ...  
  <!-- OR -->  
  <input type="hidden" name="_eventName" value="addNewItem" />  
</stripes:form>
```



/// Doing it ...*cleaner*

- Clean URLs – previously available as a plugin, but since 1.5 is built-in the core

```
@UrlBinding("/action/shoppingList/{$event}/{itemId}")
public class ShoppingListAction implements ActionBean {

    private Long itemId;

    public Resolution delete() {...}
    ....
}
```



http://localhost:8080/stripes-sample/action/shoppingList/delete/12

/// Binding

- Classic JavaBeans-style binding, automatic nested properties binding, etc:

```
private ShoppingListItem newItem;  
private List<ShoppingListItem> shoppingList;  
...  
public ShoppingListItem getNewItem() {...}  
public List<ShoppingListItem> getShoppingList() {...}
```

- Automatically instantiates beans (and even collections) for you. No `NullPointerException` when submitting any of these:

```
<input name="newItem.name" ..
```

```
<input name="shoppingList[0].name" value="newVal" ..
```

```
<input name="mapWithEnumKeys[SOME_ENUM_VALUE].name"  
value="newValue" ..
```

/// Page code for binding

- HTML standards

```
<input name="newItem.name"  
       value="{actionBean.newItem.name}" />
```

- Stripes custom tags

```
<stripes:form action="...">
```

```
  <stripes:text name="newItem.name" />
```

```
  <stripes:checkbox name="someBooleanProperty" />
```

```
  <stripes:select>
```

```
    <stripes:options-collection collection="items"  
                              value="itemId" label="name" />
```

```
  </stripes:select>
```

```
</stripes:form>
```

/// How about validating user input?

Commonly used validation features (required, min/max length, mask, etc) are offered out of the box by using the `@Validate` annotation:

```
@Validate(required=true, minlength=3)
private String firstName;
```

More complex validation checks can be achieved with the support for JSP EL-like expressions:

```
private String firstName;
@Validate(expression="this != firstName")
private String lastName;
```

Enable validation only for certain events:

```
@Validate(required=true, on={"addNewItem"})
private String name;
```

/// Validating complex objects

- Validating a bean's inner properties:

```
@ValidateNestedProperties ({  
    @Validate (field="firstName", minlength=3) ,  
})  
private Person person;
```

- Custom type converters:

```
@Validate (converter=MyConverter.class)  
private MyObject someObject;
```

The TypeConverter interface:

```
public interface TypeConverter<T> {  
    void setLocale(Locale locale);  
    T convert(String input, Class<? extends T> targetType,  
        Collection<ValidationError> errors);  
}
```

/// Other validation solutions

- Using custom validation methods (by default called after, and only when, the other validations passed):

```
@ValidationMethod  
public void validate(ValidationErrors errors) {...}
```

- Direct manipulation of validation errors list:

```
ValidationErrors errors = ...  
errors.add("quantity", new LocalizableError(".."));  
getContext().setValidationErrors(errors);
```

/// So how do I let the user know when stuff like this fails?

- Display everything:

```
<stripes:errors/>
```

- Show only specific errors:

```
<stripes:errors field="newItem.name"/>
```

- Flexible localization. You can use any of the following keys:

- `org.example.stripes.web.shopping.ShoppingListAction.newItem.name.valueNotPresent`
- `/shoppingList.action.newItem.name.errorMessage`
- `newItem.name.errorMessage`
- `validation.required.valueNotPresent`

... and many others.

/// Intercepting action execution

- Very simple pre/post processing with `@Before` and `@After`:

```
@Before(stages=LifecycleStage.BindingAndValidation)
public void prepareSomeStuff() {
    //load some stuff from DB
}
```

Other life-cycle stages: `ActionBeanResolution`,
`CustomValidation`, `EventHandling`, etc.

- AOP-style interception of event execution:

```
public interface Interceptor {
    Resolution intercept(ExecutionContext context)
        throws Exception;
}
```

```
public class ExecutionContext {
    public ActionBean getActionBean() {...}
    public Resolution proceed() throws Exception {...}
    ...
}
```

/// OK dude, but in my webapps I also use stuff like Spring ...

- Built-in Spring support.

```
<init-param>  
  <param-name>Interceptor.Classes</param-name>  
  <param-value>  
net.sourceforge.stripes.integration.spring.SpringInterceptor  
  </param-value>  
</init-param>
```

```
@SpringBean  
private IShoppingListService myService;
```

- Spring's `@Autowired` is also supported in a Stripes extension – plans to be supported out of the box in a future release.

/// Extending the framework

- Interceptors, Type Converters (and others) are determined using auto-discovery – no need for explicit configuration:

```
<init-param>  
  <param-name>Extension.Packages</param-name>  
  <param-value>  
    org.example.stripes.web.myextensions  
  </param-value>  
</init-param>
```

- The package `org.example.stripes.web.myextensions` is scanned recursively for any implementation of an extension interface: `TypeConverter<T>`, `Formatter<T>`, `Interceptor`, etc.

/// I think I got the picture. What other Stripes stuff I should know? ...

- **Exception handling**
- **Localization**
- **File upload / multipart form handling**
- **Redirect-after-post (FlashScope)**
- **Built-in mechanism for page layout reuse**
- **FreeMarker integration**
- **Handling Wizards**
- **EJB plugin (<http://code.google.com/p/stripes-ejb3/>)**
- **Plugins for persistence (Stripersist, Stripernate), security, AJAX, etc.**

... and where can I read about it?

- Project page
<http://www.stripesframework.org/>
- Author blog
<http://www.jroller.com/tfenne/>
- The book: *"Stripes: ...and Java web development is fun again"*,
by Frederic Daoud,
available from "The Pragmatic Programmers"
(<http://www.pragprog.com/titles/fdstr/stripes>)
- User mailing list
<https://lists.sourceforge.net/lists/listinfo/stripes-users>
- Issue tracking
<http://www.stripesframework.org/jira>

