A TOUR OF MODERN TEMPLATING FRAMEWORKS WITH SPRING MVC

Rob Winch, Sr Software Engineer
@rob_winch

About Me

- Spring Security Lead at Spring Source / VMware
- 15+ years web experience
- 10+ years of Java experience
- Previous Employment
  - Health Care Security at Cerner (7 years)
  - Grid Computing at Argonne National Labs (1 year)
  - Proteomics Research at Loyola University Chicago (1 year)
  - Started professional career as PERL contractor in High School

Agenda

- Introduce Message Application
- Why not JSP?
- Alternatives to JSPs
  - Mustache
  - Thymeleaf
    - w/ Tiles
  - Scalate
    - Scala Server Pages (SSP)
    - Scala Markup Language (Scaml)
    - Jade

Introduce Message Application

Message.java

```
public class Message {
    private Long id;
    private String text;
    private String summary;
    private Calendar created;

    // accessors omitted

}
```

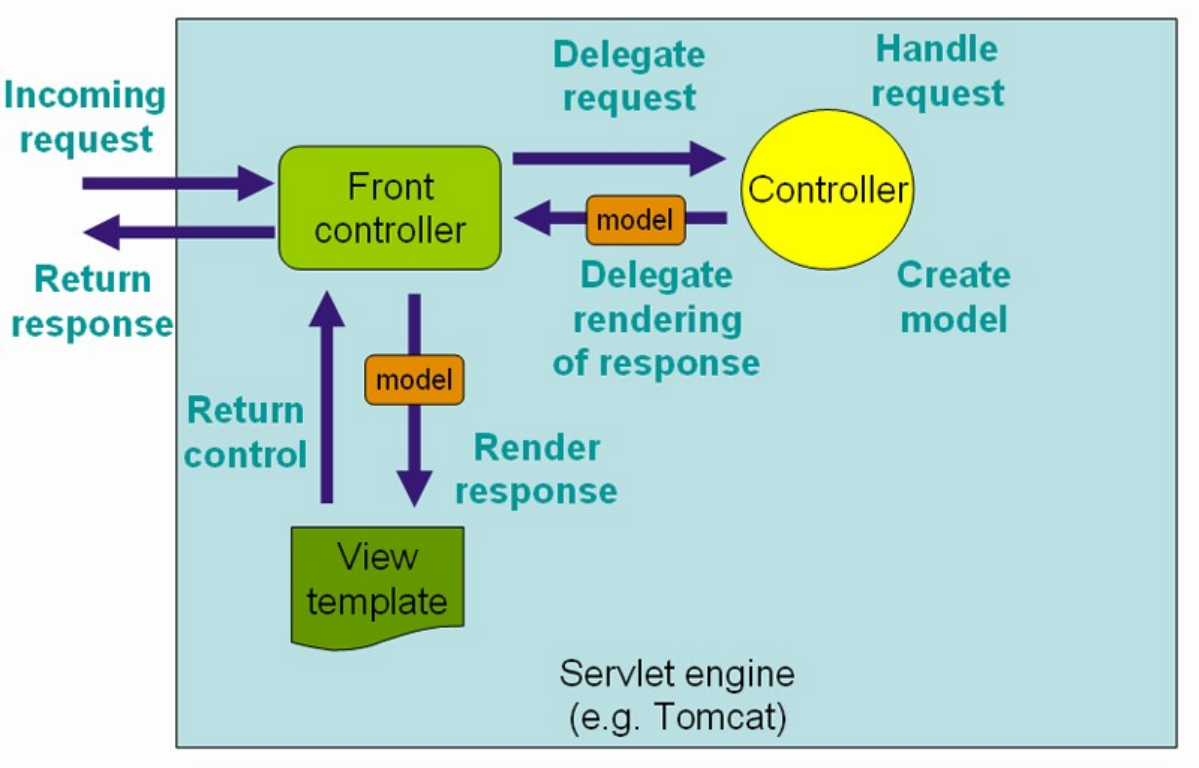
MessageRepository.java

```
public interface MessageRepository extends
        CrudRepository<Message, Long> {

}
```

MessageController.java

```
@Controller
@RequestMapping("/")
public class MessageController {
    @RequestMapping
    public ModelAndView list() {
        Iterable<Message> messages =
            messageRepository.findAll();
        return new ModelAndView("messages/list",
            "messages", messages);
    }
    ...
}
```

Spring MVC



http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html

Why not JSP?

- Cannot be used outside of a container
  - Send emails
  - Testing
  - Commandline
- More elegant (concise, better separation of concerns, etc) ways exist
- Templates can be resolved in other ways
- Output is servlet container specific

{{ mustache }}

Mustache Highlights

- Available on https://mustache.github.com
- “logic-less” because there are no control statements (i.e. if, else, loops, etc)
- Tags are specified using {{ }} (looks like a mustache)
- Implemented by many languages
  - Java, Ruby, JavaScript, Python, Erlang, PHP, Perl, Objective-C, .NET, Android, C++, Go, Lua, occ, ActionScript, ColdFusion, Scala, Closure, Fantom, CoffeeScript, D, node.js

Java Mustache Implementations & Spring MVC Integrations

- https://github.com/spullara/mustache.java
  - https://github.com/ericdwhite/mustache.java-spring-webmvc
- https://github.com/samskivert/jmustache
  - https://github.com/sps/mustache-spring-view
  - https://gist.github.com/1603296 (Keith Donald)

Mustache Variables

Hash:

```
{
  “greeting” : “Hello”,
  “message” : “World”
}
```

Template:

```
{{greeting}} {{message}}
```

Mustache Variables

Hash:

```
{
  “greeting” : “Hello”,
  “message” : “World”
}
```

Template:

```
{{greeting}} {{message}}
```

Output:

```
Hello World
```

Mustache Sections

- Notated as {{#var}} {{/var}}
- Behavior depends on the type
  - Boolean / Empty Lists
  - Non-Empty Lists
  - Lambda
  - Non-False Values

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “messages” : []
}
```

Template:

```
top
{{#messages}}
  Hidden
{{/messages}}
bottom
```

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “messages” : []
}
```

Template:

```
top
{{#messages}}
  Hidden
{{/messages}}
bottom
```

Output:

```
top
bottom
```

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “hidden” : false
}
```

Template:

```
top
{{#hidden}}
  Not Visible
{{/hidden}}
bottom
```

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “hidden” : false
}
```

Template:

```
top
{{#hidden}}
  Not Visible
{{/hidden}}
bottom
```

Output:

```
top
bottom
```

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “showme” : true
}
```

Template:

```
top
{{#showme}}
  Visible
{{/showme}}
bottom
```

Mustache Sections – Booleans / Empty Lists

Hash:

```
{
  “showme” : true
}
```

Template:

```
top
{{#showme}}
  Visible
{{/showme}}
bottom
```

Output:

```
top
Visible
bottom
```

Mustache Sections – Non-Empty Lists

Hash:

```
{
  “messages” : [
    {“summary”:“short”},
    {“summary”:“values”}
  ]
}
```

Template:

```
{{#messages}}
  <li>{{summary}}</li>
{{/messages}}
```

Mustache Sections – Non-Empty Lists

Hash:

```
{
  “messages” : [
    {“summary”:“short”},
    {“summary”:“values”}
  ]
}
```

Template:

```
{{#messages}}
  <li>{{summary}}</li>
{{/messages}}
```

Output:

```
<li>short</li>
<li>values</li>
```

Mustache Sections – Lambdas

Hash:

```
{
  “greeting” : “Hello”,
  “boldme” : function() {
    return function(text) {
      return “<b>” + render(text) + “</b>”
    }
  }
}
```

Mustache Sections – Lambdas

Template:

```
{{#boldme}}
  {{greeting}} World
{{/#boldme}}
```

Mustache Sections – Lambdas

Template:

```
{{#boldme}}
  Hello World
{{/#boldme}}
```

Mustache Sections – Lambdas

Template:

```
{{#boldme}}
  Hello World
{{/#boldme}}
```

Output:

```
<b>Hello World</b>
```

Mustache Sections – Non-False Values

Hash:

```
{
  “message” : [
    {“summary”: “short”},
    {“text”: “longer”}
  ]
}
```

Template:

```
{{#message}}
  Summary is {{summary}}
{{/#message}}
```

Mustache Sections – Non-False Values

Hash:

```
{
  “message” : [
    {“summary”: “short”},
    {“text”: “longer”}
  ]
}
```

Template:

```
{{#message}}
  Summary is {{summary}}
{{/#message}}
```

Output:

```
Summary is short
```

Mustache Sections – Inverted Sections

Hash:

```
{
  “messages” : []
}
```

Template:

```
{{^messages}}
  No Results
{{/messages}}
```

Mustache Sections – Inverted Sections

Hash:

```
{
  “messages” : []
}
```

Template:

```
{{^messages}}
  No Results
{{/messages}}
```

Output:

```
No Results
```

Mustache Sections – Partials

messages.html:

```
{{#messages}}
  {{> message}}
{{/messages}}
```

message.html:

```
<li>{{summary}}</li>
```

Mustache Sections – Partials

messages.html:

```
{{#messages}}
  {{> message}}
{{/messages}}
```

message.html:

```
<li>{{summary}}</li>
```

Output:

```
<li>short</li>
<li>values</li>
```

Mustache - Spring MVC

Thymeleaf

NOTE: Some of the Thymeleaf slides content was borrowed (with permission) from *Natural templating in Spring MVC with Thymeleaf*
http://www.thymeleaf.org/springio2012.html

Thymeleaf Highlights

- Available on http://www.thymeleaf.org
- Extensible XML / XHTML / HTML5 Template engine
- Excellent documentation
- Excellent Spring MVC integration
- Active Codebase
- Natural Templates

Thymeleaf Natural Templates

- Allows static prototypes that look like real content
- UI usually starts with static prototypes
- Prototype-to-working-UI usually the hard path
- A new approach: NATURAL TEMPLATING!

Thymeleaf Natural Templates

From Wikipedia: Template Engine (web)

“Natural Templates = the template can be a document as valid as the final result, the engine syntax doesn't break the document's structure”

Thymeleaf Natural Templates

- *“valid document, don't break structure”*
- Templates should be statically displayable
- Static = Open in browser no web server
- Templates should work as prototypes

Thymeleaf
Demo Natural Templates

Thymeleaf Namespace

Template

```
<html xmlns:th="http://www.thymeleaf.org">
```

Thymeleaf Text

Template

```
<dd th:text="${message.text}">
  A short summary...
</dd>
```

Model

```
message = new Message();
message
  .setText("is <encoded>");
```

Thymeleaf Text

Template

```
<dd th:text="${message.text}">
  A short summary...
</dd>
```

Output

```
<dd>
  is &lt;encoded&gt;
</dd>
```

Model

```
message = new Message();
message
  .setText("is <encoded>");
```

Thymeleaf Expressions & Spring

- When using Thymeleaf Spring Integration the expressions are actually using Spring Expression Language (SpEL)
- Using expressions in Thymeleaf is the same as Spring's JSP tags

Thymeleaf Formatting

```
<dd th:text="${#calendars.format(message.created)}">
  July 11, 2012 2:17:16 PM CDT
</dd>
```

```
<dd th:text="${#strings.capitalize(message.text)}">
  MANY DETAILS...
</dd>
```

Thymeleaf URLs

Template

```
<a th:href="@{/form}" href="../form.html">
  Create
</a>
```

Thymeleaf URLs

Template

```
<a th:href="@{/form}" href="../form.html">
  Create
</a>
```

Output

```
<a href="/messages/form">
  Create
</a>
```

Thymeleaf i18n

Template

```
<dt th:text="#{summary.label}">
  Summary
</dt>
```

Thymeleaf
Spring MVC Demo

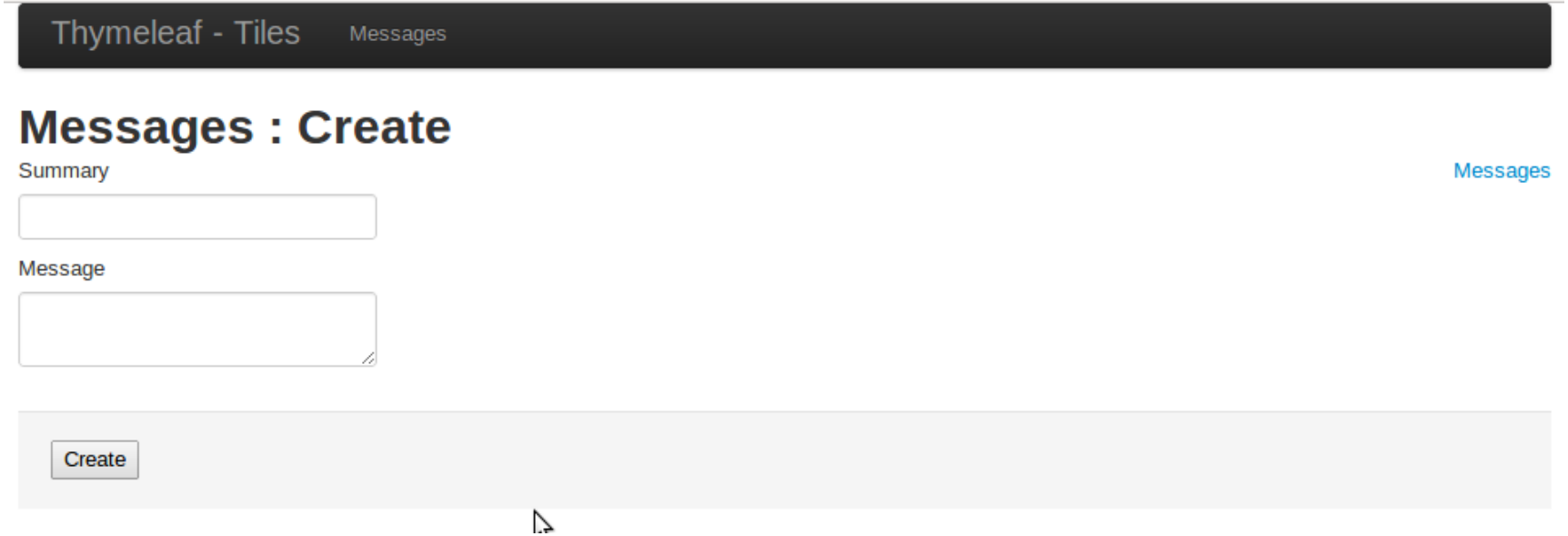
Thymeleaf & Tiles

Tiles Highlights

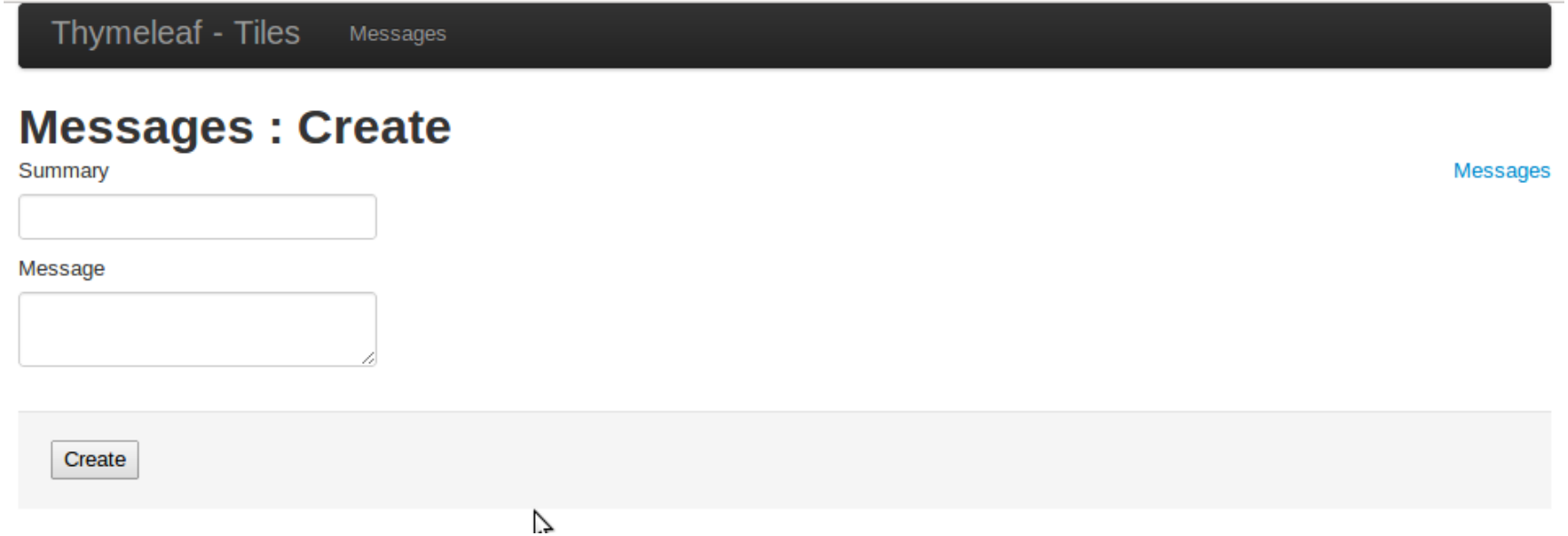
- Available on http://tiles.apache.org
- Used to support layouts (i.e. common components can be positioned and reused)
- Developed for Struts, but now is framework agnostic
- Excellent Spring Support
- Integration with a number of view technologies: JSP, Freemarker, Velocity, Thymeleaf...

Tiles Layout

Tiles Layout



Tiles Layout



Tiles Layout

- Much of the content remains the same
- Do not want to replicate it
- Tiles can help

Tiles Layout

layout

```
Fancy Header

 tiles:include=”content”

Fancy Footer
```

Tiles Layout

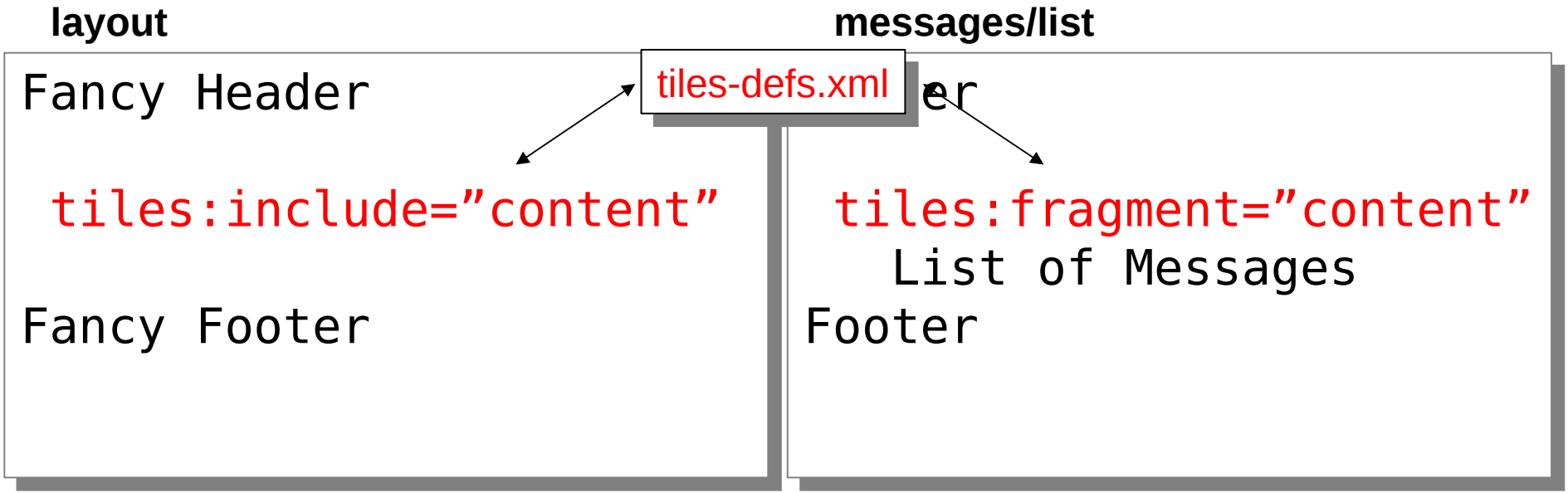
layout

```
Fancy Header

 tiles:include=”content”

Fancy Footer
```

messages/list

```
Header

 tiles:fragment=”content”
   List of Messages
Footer
```

Tiles Layout



Tiles Layout

layout

```
Fancy Header

 List of Messages

Fancy Footer
```

messages/list

```
Header

 tiles:fragment=”content”
   List of Messages
Footer
```

Tiles Layout

View Name:
messages/list

```
<definition name="messages/*"
    template="layout">
  <put-attribute name="content"
      value="content/{1}"/>
  <put-attribute name="title"
      value="title/{1}"/>
</definition>
```

Tiles Layout

View Name:
messages/list

```
<definition name="messages/list"
    template="layout">
  <put-attribute name="content"
      value="content/{1}"/>
  <put-attribute name="title"
      value="title/{1}"/>
</definition>
```

Tiles Layout

View Name:
messages/list

```
<definition name="messages/list"
    template="layout">
  <put-attribute name="content"
      value="content/list"/>
  <put-attribute name="title"
      value="title/list"/>
</definition>
```

Tiles Layout

View Name:
content/list

```
<definition name="content/*"
  template="messages/{1} :: content"/>
```

Tiles Layout

View Name:
content/list

```
<definition name="content/list"
  template="messages/{1} :: content"/>
```

Tiles Layout

View Name:
content/list

```
<definition name="content/list"
  template="messages/list :: content"/>
```

Tiles Layout

View Name:
content/list

```
<definition name="content/list"
  template="messages/list :: content"/>
```

WEB-INF/templates/**messages/list**.html

```
<html ... xmlns:tiles="http://www.thymeleaf.org">
  ...
    <div tiles:fragment="content">
      ...
    </div>
  ...
</html>
```

Tiles Layout

View Name:
content/list

```
<definition name="content/list"
  template="messages/list :: content"/>
```

WEB-INF/templates/messages/list.html

```
<html ... xmlns:tiles="http://www.thymeleaf.org">
  ...
    <div tiles:fragment="content">
      ...
    </div>
  ...
</html>
```

Tiles Layout

```
<definition name="messages/list"
    template="layout">
```

WEB-INF/templates/layout.html

```
<html ... xmlns:tiles="http://www.thymeleaf.org">
  … (banner, etc) …
    <div tiles:substituteby="content">
      … insert messages/list's content …
    </div>
  ...
</html>
```

Tiles Layout

```
<definition name="messages/list"
    template="layout">
```

WEB-INF/templates/layout.html

```
<html ... xmlns:tiles="http://www.thymeleaf.org">
  … (banner, etc) …
    <div tiles:substituteby="content">
      … insert messages/list's content …
    </div>
  ...
</html>
```

Thymeleaf, Tiles, Spring MVC Demo

Scalate

Scalate Highlights

- Available at http://scalate.fusesource.org/
- Scala based template engine which supports:
  - Mustache
  - Scala Server Pages (SSP)
  - Scaml (Scala Markup Language)
  - Jade
- Good Documentation
- Support for many frameworks:
  - Spring MVC, Play, Jog (JaxRS/Jersey), Lift, Scalatra, OSGi, custom
- Scalate Tool

Scala Server Pages (SSP)

Scalate SSP

- Good if you know Velocity, JSP, or ERB (Embedded RuBy)
- Simple – use markup and <% … %> and ${ … } for dynamic content

SSP Expressions

Hash:

```
{
  “greeting” : “Hello”
}
```

Attributes:

```
<%@ val greeting : String %>
<%@ val message : String = “default”%>
```

SSP Expressions

Hash:

```
{
  “greeting” : “Hello”
}
```

Template:

```
${greeting} <%= message %>
```

Attributes:

```
<%@ val greeting : String %>
<%@ val message : String = “default”%>
```

SSP Expressions

Hash:

```
{
  “greeting” : “Hello”
}
```

Template:

```
${greeting} <%= message %>
```

Output:

```
Hello default
```

Attributes:

```
<%@ val greeting : String %>
<%@ val message : String = “default”%>
```

SSP Attributes

Hash:

```
{
  “message” : message
}
```

Template:

```
<%@ import val message : Message %>
Summary ${summary}
```

Access
message.summary

SSP Expressions

Template:

```
<%
  var m = “Hello”
  m += “ World”
 %>
<p>${m}</p>
```

SSP using Scala Code

Template:

```
<%
  var m = “Hello”
  m += “ World”
 %>
<p>${m}</p>
```

Output:

```
<p>Hello World</p>
```

Scalate
SSP Spring MVC Demo

SSP Layouts

Template:

```
Header
${unescape(body)}
Footer
```

Template:

```
<% layout("../layouts/default.ssp") {%>
  Content
<% } %>
```

SSP Layouts

Output:

```
Header
  Content
Footer
```

Scaml

Scaml

- Scala version of Haml
- DRY way of writing XHTML templates
- Is very concise by using whitespace to determine open close tags

Scaml Doctype

Template:

```
!!! 5
```

Output:

```
<!DOCTYPE html>
```

Scaml Doctype

- Other Doctypes include (but not limited to)
  - !!!
    - XHTML Transitional
  - !!! Strict
    - XHTML 1.0 Strict
  - !!! Mobile
    - XHTML Mobile 1.2
  - !!! Basic
    - XHTML Basic 1.1

Basic Scaml

Template:

```
%div
  %span
    <p>Hello World</p>
```

Basic Scaml

Template:

```
%div
  %span
    <p>Hello World</p>
```

Output:

```
<div>
  <span>
    <p>Hello World</p>
  </span>
</div>
```

Scaml Attributes

Template:

```
%div(class=”message”)
  %span{:id=>”greet”}
    <p>Hello World</p>
```

Scaml Attributes

Template:

```
%div(class=”message”)
  %span{:id=>”greet”}
    <p>Hello World</p>
```

Output:

```
<div class=”message”>
  <span id=”greet”>
    <p>Hello World</p>
  </span>
</div>
```

Scaml Class/ID

Template:

```
%div.message
  %span#greet
    <p>Hello World</p>
```

Scaml Class/ID

Template:

```
%div.message
  %span#greet
    <p>Hello World</p>
```

Output:

```
<div class=”message”>
  <span id=”greet”>
    <p>Hello World</p>
  </span>
</div>
```

Scaml Implicit Divs

Template:

```
.message
  #greet
    <p>Hello World</p>
```

Scaml Implicit Divs

Template:

```
.message
  #greet
    <p>Hello World</p>
```

Output:

```
<div class=”message”>
  <div id=”greet”>
    <p>Hello World</p>
  </div>
</div>
```

Scaml Implicit Divs

Template:

```
.message.alert
  #greet
    <p>Hello World</p>
```

Scaml Implicit Divs

Template:

```
.message.alert
  #greet
    %p Hello World
```

Output:

```
<div class=”message alert”>
  <div id=”greet”>
    <p>Hello World</p>
  </div>
</div>
```

Scaml Attributes

Hash:

```
{
  “greeting” : “Hello”,
  “message” : “World”
}
```

Template:

```
-@ val greeting : String
-@ val message : String = “”
```

Expressions

Hash:

```
{
  “greeting” : “Hello”,
  “message” : “World”
}
```

Template:

```
%p Hi #{message}
```

Attributes:

```
-@ val greeting : String
-@ val message : String = “”
```

Scaml Expressions

Hash:

```
{
  “greeting” : “Hello”,
  “message” : “World”
}
```

Template:

```
%p Hi #{message}
```

Output:

```
<p>Hi World</p>
```

Attributes:

```
-@ val greeting : String
-@ val message : String = “”
```

Scaml Attributes

Hash:

```
{
  “message” : message
}
```

Template:

```
-@ import val message : Message
Summary ${summary}
```

Access
message.summary

Scaml Expressions

Template:

```
-
  var m = “Hello”
  m += “ World”

%p ${m}
```

Scaml using Scala Code

Template:

```
-
  var m = “Hello”
  m += “ World”

%p ${m}
```

Output:

```
<p>Hello World</p>
```

Scalate
Scaml Spring MVC Demo

Scaml Layouts

Template:

```
Header
!~~ body
Footer
```

Template:

```
- layout("../layouts/default.scaml")
  Content
```

Scalate Layouts

Output:

```
Header
  Content
Footer
```

Jade vs Scaml

Scaml:

```
%div
  %span
    Hello World
```

Jade:

```
div
  span
    | Hello World
```

Jade vs Scaml

- Scaml
  - Elements start with %
  - Text is not prefixed
  - More concise with a lot of content
- Jade
  - Elements are not prefixed
  - Text is prefixed with |
  - More concise with a lot of markup

Learn More. Stay Connected.

At SpringOne 2GX:

- Client-Side UI Smackdown

Web: springsource.org

- Github: github.com/rwinch/spring-modern-templating
- Newsletter: springsource.org/news-events
- Twitter: twitter.com/rob_winch
- YouTube: youtube.com/user/SpringSourceDev
- LinkedIn: springsource.org/linkedin