broadleaf commerce

Using Thymeleaf as a Templating Language for Java Web MVC

Jay Aisenbrey

Broadleaf Commerce

- Open source, Java-based eCommerce framework
- Enterprise features
  - Order Management System
  - Content Management System
  - Multi-Tenant
  - Powerful offers engine
- Fortune 500 and IR Top 100 clients
  - The Container Store
  - Vology
  - Pep Boys
- Spring MVC
- Customizable, flexible, and scalable

broadleaf
commerce

Thymeleaf

- Open source server-side Java template engine
- Write pure HTML, CSS, and JS that can be displayed without processing
- Integrates well with a Spring environment
- Uses SpEL and OGNL
  - Therefore you can use static class function and object functions
- Full internationalization support
- Highly customizable
- Configurable parsed template cache
  - Cache where the template is and, possibly, what it evaluates to

broadleaf commerce

Thymeleaf vs Other Templating Languages

- Pros
  - Natural Templating unlike FreeMarker and JSP
    - Great for prototyping so that designers don’t need to know a template language
  - Bottom-up Spring support
    - Along with the Spring dialect you get great I18n support
  - Server side rendering
    - Great for mobile support since servers have a lot more power than a smartphone
  - XHTML and HTML5 support which adds in room for specific optimizations
  - Arguably better/cleaner syntax and great documentation
  - No dependency on the Servlet API

Thymeleaf vs Other Templating Languages

- Cons
  - Limited to XHTML and HTML5 unlike Velocity and FreeMarker
  - Not as popular as JSP and other templating languages
  - No JSP tag library support

Basic Example

HomeController.java

```
@RequestMapping(“/”)
public String home(Model model) {
    model.addAttribute(“firstName”, “Jay”);
    model.addAttribute(“lastName”, “Aisenbrey”);
    return “home”;
}
```

th:text - Escapes HTML for when you want to display “<” as “<”

th:utext - Doesn’t escape HTML for when you want to inject HTML into the tag

/WEB-INF/templates/home.html

```
<span th:text=”${‘Hi’+ firstName + ‘ ‘ + lastName}”>Welcome</span>
```

Without Rendering

Welcome

With Rendering

Hi Jay Aisenbrey

broadleaf commerce

Basic Example (i18n)

/WEB-INF/templates/home.html

```
<span th:text="${#{home.hi} + firstName + ' ' + lastName}">Welcome</span>
```

messages_en.properties

```
home.hi=Hi
```

messages_es.properties

```
home.hi=Hola
```

Rendered (en)

Hi Jay Aisenbrey

Rendered (es)

Hola Jay Aisenbrey

Basic Syntax

- Simple Expressions
  - Model variable expression: ${...}
    - Reference global model variables
  - Selection variable expression: *{...}
    - Reference local model variables
  - Message expression: #{...}
  - Link URL expression: @{...}
    - Creates urls with correct path
  - Fragment Expression: ~{...}
    - References a piece of a template

Scoping Model Variables

```
<span>
    [[${user.firstName}]] [[${user.lastName}]] <br/>
    [[${user.address.line1}]] <br/>
    [[${user.address.line2}]] <br/>
    [[${user.address.city}]], [[${user.address.state}]] [[${user.address.zipcode}]]
</span>
```

```
<span th:object=”${user.address}”>
    [[${user.firstName}]] [[${user.lastName}]] <br/>
    [[*{line1}]] <br/>
    [[*{line2}]] <br/>
    [[*{city}]], [[*{state}]] [[*{zipcode}]]
</span>
```

Inline syntax - Great for using model variables in a large block of static text.

Scoping All Variables

```
<span th:object="${user.address}>
    [[${user.firstName}]] [[${user.lastName}]] <br/>
    [[*{line1}]] <br/>
    [[*{line2}]] <br/>
    [[*{city}]], [[*{state}]] [[*{zipcode}]]
</span>
```

```
<span th:object="${user.address} th:with="fullName=${user.firstName + ‘ ‘ + user.lastName}">
    [[${fullName}]]
    [[*{line1}]]<br/>
    [[*{line2}]]<br/>
    [[*{city}]], [[*{state}]] [[*{zipcode}]]
</span>
```

Conditionals and Loops

```
<th:block th:each=”product : ${products}”>
    <div>
        <span th:if=”${product.isOnSale()}” th:text=”${product.salePrice}”></span>
        <span th:unless=”${product.isOnSale()}” th:text=”${product.retailPrice}”></span>
    </div>
</th:block>
```

```
<th:block th:each=”product : ${products}”>
    <div>
        <span th:text=”${product.isOnSale() ? product.salePrice : product.retailPrice}”></span>
    </div>
</th:block>
```

Includes

/WEB-INF/templates/catalog/search.html

```
<th:block th:each=”product : ${searchResults}”>
	<div th:replace=”catalog/partials/productResult” ></div>
</th:block>
```

/WEB-INF/templates/catalog/partials/productResult.html

```
<div>
	<img th:src=”@{${product.image?.url}}” />
	<span th:text=”${product.name}”></span>
	<span th:text=”${product.isOnSale() ? product.salePrice : product.retailPrice}”></span>
</div>
```

th:replace - replace this tag with the results of the partial.

th:include - put the results of the partial inside of the tag

broadleaf commerce

Variable Expressions

- Just a Java class with some methods
- Has a name that correlates with it → “lists” and “strings” in the example
- Good easy way to make helper functions
- Does not have access to model, only access to the class and parameter

HomeController.java

```
@RequestMapping(“/”)
public String home(Model model) {
        List<String> strings = new ArrayList<>();
        strings.add(“apple”);
        strings.add(“oranges”);
        strings.add(“bananas”);
        model.add(“strings”, strings):
        return “home”;
}
```

WEB-INF/tempates/home.html

```
<span th:text=”${#lists.size(strings)}”></span>
<span th:each=”string : ${strings}”>
        th:text=”${#strings.size(string)}”></span>
```

broadleaf commerce

Processors

- All Processors
  - A keyword tells Thymeleaf when to run certain Java code. Usually to modify the DOM.
  - Some come with Thymeleaf but custom ones can be made
- Attribute Processor
  - The keyword is an attribute on the tag
    - th:text
    - th:src
    - th:attr - ex. <span th:attr=”data-id=${something}” …> → <span data-id=”result” …>
- Tag Processor
  - The keyword is the actual tag name
    - th:block
- Generally use variable expressions over processors
  - Variable expressions are easier to create, support, and upgrade
  - Only use processors if you’re modifying the DOM

Built-in Utilities

- #dates
  - #dates.format(date), #dates.day(date)
- #numbers
  - #numbers.formatInteger(num, 3), #numbers.sequence(from, to, step)
- #strings
  - #strings.isEmpty(str), #strings.contains(str, “hi”)
- #arrays
  - #arrays.length(arr), #arrays.isEmpty(arr)
- #lists
  - #lists.size(lis), #lists.isEmpty(lis)
- #sets
  - #sets.contains(set, element), #sets.isEmpty(set)

How Broadleaf Uses Thymeleaf

- Created over 40 custom processors
  - Form Processor
    - looks for tag keyword “blc:form”, if the method is “POST” then a CSRF token is added
  - Price Text Display Processor
    - looks for attribute “blc:price” and then casts the value to a double that has two decimal places and adds a “$” in front
  - Cache Processor
    - looks for attribute “blc:cache” and then uses an object that is set on that tag as a cache key and then caches the rendered block of HTML
- Created over 10 variable expressions
  - Do various tasks from looking up properties out of the database to retrieving data about the current request
  - We usually create variable expressions for clients since that’s when we need to do more complicated processes that usually return a string or object

broadleaf commerce

How Broadleaf Uses Thymeleaf

- Use a tiered system of template resolvers
  - This is for our admin since we have a default set of templates set in the core framework
  - Modules that override partials and/or templates set up resolvers that resolve the same file but in its classpath.
    - Also set the precedence higher so that the module resolver is used first
  - Looks at the servlet path first to see if the client overwrote a custom template or partial
- Utilize different types of template resolvers
  - Database resolver
    - We have templates that are managed through our admin
  - Dynamic Servlet and Classpath resolvers
    - Depending on a theme we change the directories to look in so that a website can easily be re-skinned based on a database property

broadleaf commerce

Thymeleaf 3 Features

- Ability to send fragments as parameters to other templates
  - Send a fragment from current template (template A) to another template (template B)
  - Template B uses the fragment sent from template A to evaluate its fragment
  - The resulting fragment from B is returned to A who then uses that result
  - This effectively gets rid of layout dialects
    - Effectively a way to always wrap your content with the same head, header, foot, footer.
- Inline expression no longer requires th:inline=”text” attribute in parent tags
  - Previously → <span th:inline=”text”>[[${product.name}]]</span>
  - Now → <span>[[${product.name}]]</span>
  - Mostly just a quality of life change

Thymeleaf 3 Features

- No longer uses XML parser in favor of DOM parser
  - Previously used a SAX parser but now uses a parser, written by the creator of Thymeleaf, called AttoParser 2.0
- Better performance
  - Entire backend parsing system rewritten to use an event-based parsing system
  - Switch to AttoParser 2.0
- Decoupled templating from HTML
  - Designers can write pure HTML with no Thymeleaf
  - All Thymeleaf can be written 100% separately

broadleaf commerce

Layouts Using Fragments

```
<head th:fragment="common_header(title,links)">

  <title th:replace="${title}">The awesome
application</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css"
          media="all" th:href="@{/css/awesomeapp.css}">

  <link rel="shortcut icon"
          th:href="@{/images/favicon.ico}">

  <script type="text/javascript"
             th:src="@{/sh/scripts/codebase.js}"></script>

  <!--/* Per-page placeholder for additional links */-->
  <th:block th:replace="${links}" />

</head>
```

```
...
<head th:replace="base :: common_header(~{::title},~{::link})">

  <title>Awesome - Main</title>

  <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
  <link rel="stylesheet"
th:href="@{/themes/smoothness/jquery-ui.css}">

</head>
...
```

broadleaf
commerce

Layouts Using Fragments

Result

```
...
<head>

  <title>Awesome - Main</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

  <link rel="stylesheet" href="/awe/css/bootstrap.min.css">
  <link rel="stylesheet" href="/awe/themes/smoothness/jquery-ui.css">

</head>
...
```

broadleaf commerce

Decoupled Templating

Normal Templating (home.html)

```
<!DOCTYPE html>
<html>
 <body>
  <table id="usersTable" th:remove="all-but-first">
   <tr th:each="user : ${users}">
    <td class="username" th:text="${user.name}">Jeremy Grapefruit</td>
    <td class="usertype" th:text="#{|user.type.${user.type}|}">Normal User</td>
   </tr>
   <tr>
    <td class="username">Alice Watermelon</td>
    <td class="usertype">Administrator</td>
   </tr>
  </table>
 </body>
</html>
```

broadleaf commerce

Decoupled Templating

HTML (home.html)

```
<!DOCTYPE html>
<html>
  <body>
    <table id="usersTable">
      <tr>
        <td class="username">Jeremy Grapefruit</td>
        <td class="usertype">Normal User</td>
      </tr>
      <tr>
        <td class="username">Alice Watermelon</td>
        <td class="usertype">Administrator</td>
      </tr>
    </table>
  </body>
</html>
```

Templating (home.th.xml)

```
<?xml version="1.0"?>
<thlogic>
  <attr sel="#usersTable" th:remove="all-but-first">
    <attr sel="/tr[0]" th:each="user : ${users}">
      <attr sel="td.username"
                th:text="${user.name}" />
      <attr sel="td.usertype"
                th:text="#{|user.type.${user.type}|}" />
    </attr>
  </attr>
</thlogic>
```

broadleaf commerce

Thymeleaf Ecosystem

Thymol - http://www.thymoljs.org/

- Client-Side JS implementation of Thymeleaf
- Doesn’t need a web server to work
- Takes static prototyping even further
- So far only supports Thymeleaf 2

Thymeleaf Ecosystem

Thymeleaf Testing Library - github.com/thymeleaf/thymeleaf-testing

- Used to test the view layer's processing and results
- Includes benchmarking utilities
- Works with many testing frameworks including JUnit
- Spring Framework and Spring Security integration
- Supports Thymeleaf 3
- Managed by the creators of Thymeleaf

broadleaf commerce

Configuration

```xml
<bean id="thymeleafSpringStandardDialect" class="org.thymeleaf.spring4.dialect.SpringStandardDialect" />
<bean id="webTemplateResolver" class="org.thymeleaf.spring4.templateresolver.SpringResourceTemplateResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".html" />
    <property name="cacheable" value="${cache.page.templates}"/>
    <property name="cacheTTLMs" value="${cache.page.templates.ttl}" />
    <property name="characterEncoding" value="UTF-8" />
    <property name="order" value="200"/>
</bean>
<bean id="templateResolvers" class="java.util.ArrayList">
    <constructor-arg>
        <list>
            <ref bean="webTemplateResolver" />
        </list>
    </constructor-arg>
</bean>
```

broadleaf commerce

Configuration

```
<bean id="webTemplateEngine" class="org.thymeleaf.spring4.SpringTemplateEngine">
    <property name="messageResolvers">
        <set>
            <bean class="org.thymeleaf.spring4.messageresolver.SpringMessageResolver" />
        </set>
    </property>
    <property name="templateResolvers" ref="templateResolvers" />
    <property name="dialects">
        <set>
            <ref bean="thymeleafSpringStandardDialect" />
        </set>
    </property>
</bean>
<!-- Set up the view resolver to be used by Spring -->
<bean id="viewResolver" class="org.thymeleaf.spring4.view.ThymeleafViewResolver">
    <property name="templateEngine" ref="webTemplateEngine" />
    <property name="order" value="1" />
    <property name="cache" value="${thymeleaf.view.resolver.cache}" />
    <property name="characterEncoding" value="UTF-8" />
</bean>
```

broadleaf commerce

Resources

- Tutorial
  - http://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
- AttoParser 2.0
  - http://www.attoparser.org/
- Ten minute migration guide
  - http://www.thymeleaf.org/doc/articles/thymeleaf3migration.html
- My blog about Broadleaf's upgrade from 2.1 to 3.0
  - http://www.broadleafcommerce.com/blog/broadleaf-commerce-upgrade-to-thymeleaf-3
- Me
  - email : jaisenbrey@broadleafcommerce.com
  - GitHub : cja769

broadleaf commerce

Q&A

To contact Broadleaf:

- EMAIL: sales@broadleafcommerce.com
- TWITTER: @broadleaf
- YOUTUBE: Broadleaf Commerce
- WEBSITE: www.broadleafcommerce.com