02-2017 | Sommer | www.ijug.eu

Java aktuell

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java im Mittelpunkt

Big Data
Predictive Analytics mit Apache Spark

Microservices
Lagom, das neue Framework

Achtung, Audit
Nutzung und Verteilung von Java SE

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

4 191978 304903 02

iJUG Verbund

Thymeleaf – eine Template-Engine für Entwickler und Designer

Gerrit Meier

In den letzten Jahren wurde mit Single Page Applications auf JavaScript-Basis immer mehr der Schritt weg vom serverseitigen Rendering gemacht. Auch die Aussage von Oracle im Rahmen der JavaOne, dass die meisten Anwendungen in der Cloud gänzlich ohne Oberfläche auskommen werden, gibt diesem Ansatz wenig Rückenwind. Warum aber erfreut sich, ganz gegen diesen Trend, die Template-Engine Thymeleaf immer weiter steigender Nutzerzahlen? Der Artikel zeigt, was Thymeleaf anders macht und warum es gerade bei neuen Projekten eine interessante Alternative sein kann.

Bei der Entscheidung, ob eine Web-Anwendung server- oder clientseitig gerendert werden sollte, gibt es viele möglichen Faktoren, die man berücksichtigen und bewerten muss. Dabei sollte, wie immer, das grundlegende (Kunden-)Problem und dessen Lösung die Hauptrolle spielen. Ist einmal die Entscheidung getroffen, sich nicht auf JavaScript-Frameworks wie ReactJS oder Angular zu verlassen, sondern die HTML-Repräsentation auf dem Server zu generieren, steht man vor der nächsten Frage: Mit welcher Engine soll die Seite verarbeitet werden?

An dieser Stelle kann zu traditionellen Bibliotheken wie JavaServer Pages (JSP), Velocity oder Freemarker gegriffen und das Rendering wie immer gelöst werden. Ein Blick über den Tellerrand beziehungsweise eine Google-Suche nach Alternativen in der Java-Welt bringt einen weiteren Kandidaten zum Vorschein: Thymeleaf *[1]*.

Das Open-Source-Projekt, das sich vor allem durch eine übersichtliche, aber dennoch nicht einschränkend wirkende Syntax auszeichnet, existiert seit dem Jahr 2011. Dazu gibt es eine hervorragende Integration in das Spring-Ökosystem, unter anderem durch einen Spring Boot Starter *[2]*. Auf diesen Weg lassen sich sehr schnell Anwendungen unter Verwendung von Thymeleaf-Templates erstellen und die ersten Schritte mit der Engine unternehmen.

Natural Templating

Vor der Besprechung der Syntax und der Funktionsweise von Thymeleaf wird ein einfaches JSP- mit einem Thymeleaf-Template verglichen *(siehe Listings 1 und 2)*. Beide Dateien erzeugen die gleiche Ausgabe, wenn die Seiten aus einer laufenden Anwendung aufgerufen werden. Das Besondere an Thymeleaf ist, dass das HTML-Template auch statisch geöffnet werden kann und ein gemocktes, grafisch korrektes Ergebnis zu sehen ist *(siehe Abbildung 1)*. Im Vergleich dazu die statisch geöffnete JSP-Quelldatei *(siehe Abbildung 2)*.

Wer den Sourcecode beider Template-Engines genauerer betrachtet, erkennt schnell, dass Thymeleaf im Gegensatz zu JSP keine Custom Tags für die Logik verwendet, sondern mit Pseudo-Attributen arbeitet. Dieser Ansatz, auch als „Natural Templating" bezeichnet, beschreibt vor allem die Nutzung der Quelldateien als Prototyp/Mocks.

Ein großer Vorteil besteht darin, dass die Entwickler diese Quelldateien etwa an einen Grafiker geben können, der diese ohne eine laufende Anwendung auf dem Server weiter stylen kann. Solange die Datei direkt geöffnet und nicht durch die Engine verarbeitet wird, werden die unbekannten Attribute vom Browser verworfen und die gemockten Daten aus den bekannten Attributen und Texten in den HTML-Tags angezeigt. Wird die View durch einen Server ausgeliefert,

ersetzt Thymeleaf die Attribute, für die entsprechende Werte definiert wurden.

Alternativ zur Standard-Syntax können auch Data Attributes verwendet werden (also „data-th-href" statt „th:href"). So lässt sich anstelle mit unbekannten HTML-Attributen, die vom Browser ignoriert werden, mit korrekten HTML5-Attributen arbeiten. Dies betrifft nur den statischen Zustand der Datei und ist aus Sicht des Autors als „nice to have" einzuordnen. Im weiteren Verlauf gilt die Konvention, die Attribute aus dem Thymeleaf-Namespace zu nutzen.

Einfache Syntax

Wie man schon am Beispiel erkennen kann, ist die Syntax auch beim ersten Kontakt mit der Engine leicht zu verstehen. Das liegt vor allem daran, dass sich Thymeleaf lediglich auf zwei Kern-Komponenten im Markup konzentriert: Expressions und Processors. In *Listing 3* sind einige grundlegenden Expressions zu sehen, die nun genauer beschrieben werden. Als Expression bezeichnet man den Ausdruck im Wert des Thymeleaf-HTML-Attributs.

In der ersten Zeile wird die wahrscheinlich bekannteste Form angetroffen, Variablen in einer (Java-)Web-Anwendung auszugeben: das Einschließen der Variablen in „${...}". Mit diesem Ausdruck werden Werte oder Objekte aus unserem Controller ausgegeben beziehungsweise verwendet. Als weiterer Bekannter gehört auch der Ausdruck „#{...}" zum Sprachumfang, der für die Ausgabe von fixen Strings/Messages wie „i18n"-Properties verwendet werden kann.

Bis jetzt wurde für die meist verwendeten Zugriffsarten auf Werte oder Objekte keine neue Syntax eingeführt, sodass hier schon zu erkennen ist, dass die Lernkurve von Thymeleaf erfreulich flach ist. Soll es ein wenig mehr Komfort sein, kann man auf die Expression *{...} zurückgreifen. Wie in den Zeilen drei bis fünf zu sehen ist, wird mit ihr auf ein umschließendes Objekt zurückgegriffen, dessen Attribute werden direkt referenziert. Falls kein äußeres Objekt vorhanden ist, verhält sich dieser Ausdruck wie der genannte Variablen-Zugriff per ${...}.

Sehr nützlich ist die URL-Expression „@{...}". In *Listing 3* sind mehrere Beispiele zu sehen, allen voran die Verwendung von relativen Pfaden in der Anwendung. Hier wird der Context-Pfad der Anwendung automatisch ermittelt und ergänzt, sodass ein komplexes beziehungsweise unschönes Präfixen per Hand nicht notwendig ist. URL-Expressions lassen sich auch parametrisieren. Dies bietet zum Beispiel den Vorteil, dass mit relativ elegantem Code in Schleifen bei jedem Eintrag auf die gleiche URL mit unterschiedlichen Parametern gemappt werden kann. Die Parametrisierung kann dabei mit oder ohne Platzhalter erfolgen. Dies führt im ersten Beispiel zu einem Link mit Query-Parameter und im zweiten zur Ersetzung des Platzhalters.

Soll ein Link doch einmal auf eine andere

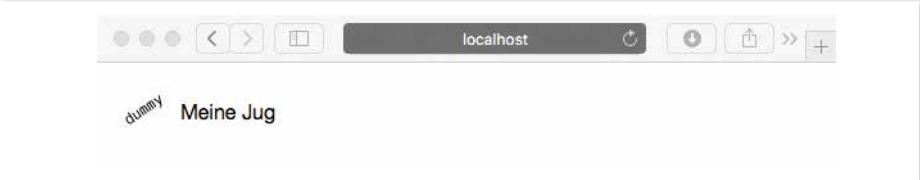

Abbildung 1: Statische Ausgabe von Thymeleaf

Abbildung 2: Statische Ausgabe von JSP

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
    <title></title>
    <link href="${pageContext.request.contextPath}/css/jug_style.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
<table>
<c:forEach items="${jugs}" var="jug">
    <tr class="jug_entry">
        <td><a href="<c:url value="/jug/"/>${jug.id}"><img src="<c:out
value="img/${jug.image}"/>"/></a></td>
        <td><a href="<c:url value="/jug/"/>${jug.id}">${jug.name}</a></td>
    </tr>
</c:forEach>
</table>
</body>
</html>
```

Listing 1

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8"/>
  <title>JUG List</title>
  <link th:href="@{/css/jug_style.css}" href="css/jug_style.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
<table>
  <tr class="jug_entry" th:each="jug : ${jugs}">
    <td><a href="#"><img th:src="@{'/img/' + ${jug.image}}" src="img/dummy.
jpg"/></a></td>
    <td><a href="#" th:text="${jug.name}">Meine Jug</a></td>
  </tr>
</table>
</body>
</html>
```

Listing 2

```
<span th:text="${jug.name}">Platzhalter</span>
<span th:text="#{welcome.message}">Platzhalter</span>

<div th:object="${jug}">
  <span th:text="*{name}">Platzhalter</span>
</div>

<a th:href="@{/jugs}">JUGs</a>
<!-- Parameter ohne Platzhalter: /jug/details?jugId=1 -->
<a th:href="@{/jug/details(jugId=${jug.id})}">JUG Details</a>
<!-- Parameter mit Platzhalter: /jug/1/details -->
<a th:href="@{/jug/{jugId}/details(jugId=${jug.id})}">JUG Details</a>
<a th:href="@{~/otherApplication}">eine andere Anwendung</a>
<a th:href="@{//otherApplication}">ein anderer Server</a>
```

Listing 3

```
<span th:text="${jug.name}">Reintext</span>
<span th:utext="${jug.name}">unescaped Text</span>
<span th:inline="text">Meine JUG: [[${jug.name}]]!</span>
```

Listing 4

```
<tbody>
<tr th:each="jug : ${jugs}" th:object="${jugs}">
  <td th:text="*{name}">erste JUG</td>
</tr>
<tr>
  <td>zweite JUG</td>
</tr>
<tr>
  <td>dritte JUG</td>
</tr>
</tbody>
```

Listing 5

```
<tbody th:remove="all-but-first">
<tr th:each="jug : ${jugs}" th:object="${jugs}">
  <td th:text="*{name}">erste JUG</td>
</tr>
<tr>
  <td>zweite JUG</td>
</tr>
<tr>
  <td>dritte JUG</td>
</tr>
</tbody>
```

Listing 6

```
<table>
<!--/*/ <th:block th:each="jug : ${jugs}"> /*/-->
<tr th:object="${jug}">
  <td th:text="*{name}">erste JUG</td>
</tr>
<tr>
  <td>zweite JUG</td>
</tr>
<tr>
  <td>dritte JUG</td>
</tr>
<!--/*/ </th:block> /*/-->
</table>
```

Listing 7

Anwendung auf dem Server oder ein ganz anderes System zeigen, kann dieses auch in den URL-Expressions mit angegeben werden. Dabei wird der URL ein „~“ (Server relativ) beziehungsweise ein „//“ (Protocol relative) vorangestellt. Im zweiten Fall kann alternativ auch die vollständige URL inklusive Protokoll in der Expression verwendet werden. Dies schafft jedoch eine Bindung an das Protokoll und erzwingt den Einsatz von Environment-Configs, um beispielsweise im Testbetrieb jeglichen Datenverkehr mit anderen Systemen über HTTP anstatt HTTPS laufen zu lassen.

Processors

Unter „Processor“ wird im Thymeleaf-Kontext nicht nur das verwendete HTML-Attribut verstanden, sondern auch seine dahinterliegende Logik und das Resultat bei der Verarbeitung durch die Engine. In Thymeleaf steht eine theoretisch überschaubare Anzahl von Processors zur Verwendung bereit. Theoretisch aus dem Grund, weil es für jedes denkbare Standard-HTML5-Attribut einen Processor gibt. Diese sind intuitiv so benannt, dass man nur vor das Attribut ein „th:“ einfügen muss, um den passenden Processor zu bekommen.

Die wichtigste Funktion einer Template-Engine ist wahrscheinlich die Ausgabe von Informationen, also Text. Um Text in ein HTML-Tag unter der Verwendung von Processors auszugeben, werden die Attribute „th:text“ beziehungsweise „th:utext“ verwendet *(siehe Listing 4)*. Dies ist, wenn es sich als geeigneter erweist, auch durch den Processor „th:inline“ und das Schreiben des auszugebenden Texts in einer Inline-Form möglich. Es ist bei dieser Expression egal, ob die Ausgabe direkt als Kind oder in einer tieferen Ebene erscheinen soll. Dabei sollte beachtet werden, dass in diesem Fall die Ausgabe des Platzhalters beim Öffnen der Quelldatei immer zu sehen ist und nicht durch einen prototypischen Wert ersetzt werden kann.

Schleifen und Beispiel-Daten

Viele Anwendungen basieren auf der Anzeige von Listen, Tabellen oder ähnlichen Darstellungsformen, die meist in einer Schleife ausgegeben werden. Hier bietet Thymeleaf mit „th:each“ *(siehe Listing 5)* genau eine Lösung, um Daten aus einer Sammlung (Iterable, Map, Array etc.) zu durchlaufen. Nun besteht eine der beworbenen Besonderheiten der Engine darin, dass mit gemockten Daten

gearbeitet werden kann.

Wenn das Beispiel in der Form auf dem Server aufgerufen wird, bekommt man nicht nur die Werte geliefert, die unsere Anwendung bereitstellt, sondern am Ende auch noch die Mock-Daten. Um dieses Problem zu umgehen, wurde ein weiterer Processor („th:remove") bereitgestellt *(siehe Listing 6)*. Wie der Name schon erraten lässt, entfernt dieser je nach gewähltem Wert etwa den ganzen Tag („all"), alle Kinder („body") oder alle Kinder bis auf das erste („all-but-first"). Bei Verwendung des letztgenannten Werts kann genau das gewünschte Verhalten erreicht werden: Nach dem Rendern werden nur noch die echten Daten aus der Anwendung angezeigt und die gemockten Elemente sind entfernt.

Bedingungen

Für die Abfrage von Bedingungen gibt es in Thymeleaf wiederum eine überschaubare Anzahl von Processors. Bedingungsabfragen lassen sich durch „th:if" beziehungsweise invers durch „th:unless" realisieren. Bei der Verwendung der Expression ist jedoch etwas Vorsicht geboten, da hier nicht nur pauschal auf einen booleschen Wert im Java-Sinn zu achten ist. Abhängig von der Art des Objekts beziehungsweise seines Werts ergeben sich folgende Möglichkeiten für true: Boolean mit dem Wert „true", eine Zahl oder ein Character ungleich „0" oder ein String, der nicht „no", „false" oder „off" als Wert hat.

Soll es zu einer Fall-Unterscheidung kommen, sollte man anstelle von komplexen „if"-Kaskaden die Processors „th:switch" und „th:case" verwenden. In Thymeleaf gibt es keinen Fallthrough, sondern der erste zutreffende Case wird verwendet. Um den Default-Fall abzufangen, gibt es den speziellen Wert „th:case="*"".

Der Block-Prozessor

Da Ausnahmen die Regel bestätigen, hat auch Thymeleaf seine ganz eigene Ausnahme, den Blockprozessor. Wie in *Listing 7* zu sehen ist, ist dieser Processor kein HTML-Attribut, sondern ein – genau genommen das einzige – HTML-Tag in der Thymeleaf-Welt. Er wird vor allem verwendet, wenn Schleifen realisiert werden, aber kein Eltern-Element verwendet werden kann, das als Iterations-Anfang dient. Das Element wird nach der Verarbeitung nicht an den Browser ausgeliefert, erzeugt jedoch in der statischen Ansicht durch das unbekannte HTML-Tag Fehler im HTML-Source. Wen diese stören, der kann diesen Block mit einem bestimmten Kommentar sowohl vom Browser ignorieren als aber auch von der Engine erkennen lassen.

Templating

Durch die Verwendung von Fragments, einem weiteren Processor, ist es möglich, einfaches Templating mit Thymeleaf vorzunehmen *(siehe Listing 8)*. Die Definition der einzufügenden Inhalte erfolgt dabei in einer ausgelagerten HTML-Datei *(siehe Listing 9)*.

So ergibt sich wieder der Vorteil, dass das zu inkludierende Fragment in ein vollständiges HTML-Dokument eingebettet werden kann, um es statisch zu bearbeiten. Bevor das Einbetten der Fragmente gezeigt wird, sei an dieser Stelle der Hinweis gegeben, dass es bei komplexeren Fragmenten auch

zu einem komplexen Mocking in der Hauptseite kommen kann.

Das Hinzuziehen von HTML-Bestandteilen aus anderen Dateien ist intuitiv mit der Wahl einer von drei unterschiedlichen Einbindungsarten möglich. Zur Auswahl stehen „th:insert" (ab Thymeleaf 3), „th:replace" und „th:include", die dazu führen, dass ein Fragment entweder komplett als Kind eingefügt, das Tag in der Hauptseite ersetzt oder nur der Inhalt des Fragments in die Hauptseite übernommen wird. Dabei bezieht man sich beim Referenzieren immer auf den Datei- und definierten Fragmentnamen.

Fragmente sind darüber hinaus auch parametrisierbar und lassen sich somit sehr flexibel einsetzen. Ein möglicher Anwendungsfall wäre, eine Breadcrumb-Bar in ein Fragment auszulagern und den Besuchs- beziehungsweise Navigationsverlauf über Parameter für die Visualisierung mitzugeben.

```
<div th:include="footer :: footer"></div>
<div th:replace="footer :: footer"></div>
<div th:insert="footer :: footer"></div>
```

Listing 8

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8"/>
  <title>Footer dummy</title>
</head>
<body>
<div th:fragment="footer">
  <span>footer text</span>
</div>
</body>
</html>
```

Listing 9

Kleine Helfer

Damit das Hantieren mit unterschiedlichen Objekt-Typen mehr Komfort bietet, stehen unterschiedliche Helfer-Objekte („Expressions Utility Objects") bereit, um dem Entwickler unter die Arme zu greifen. Da wären zum einen Methoden, die mit Collections (Lists, Set, Maps, Arrays) arbeiten und praktische Funktionalität wie „contains", „isEmpty" oder „sort" anbieten. Hinzu kommt ein guter Support für Datums-, Zahlen- und String-Objekte, der viele üblichen Zugriffe oder Operationen abdeckt. Zwei String-Methoden, die hier als Beispiel des Umfangs erwähnt sind, lauten „listJoin" und „listSplit". Damit lassen sich Listen-Inhalte in einen konkatenierten String verwandeln beziehungsweise anhand eines Trennzeichens eine Liste anlegen.

Gerade an solchen Stellen erkennt man, dass die Entwickler der Engine nicht nur das Nötigste anbieten, sondern auch dem Nutzer helfen, Probleme auf eine elegante Weise zu lösen. Richtig eingesetzt, können die Utility Objects somit sowohl zur Funktionalität als auch zur Lesbarkeit des Codes beitragen.

Ausblick

Dieser Artikel basiert größtenteils auf der Version 2.1 von Thymeleaf, obwohl schon seit Mai letzten Jahres die Version 3 verfügbar ist. Grund dafür ist, dass es bei den angesprochenen Features wenig bis gar keine Änderungen in der Nutzung gibt. Ein weiterer und eigentlich der ausschlaggebende Faktor ist, dass bei der Erstellung dieses Artikels der Support für die aktuelle Version in Spring Boot, mit dem man sehr schnell Thymeleaf-Projekte erstellen kann, noch nicht existierte *(siehe „https://github.com/spring-projects/spring-boot/issues/7450)"*. Man muss jedoch keine Angst davor haben, Templates unter Verwendung der Version 2.1 zu erstellen. Sie lassen sich später auf Version 3 migrieren.

Thymeleaf 3 zeichnet sich vor allem durch ein annähernd komplettes Rewrite der Engine, unter Rücksichtnahme auf die aktuellen Anforderungen, aus. So wurden zum Beispiel die Processor- und Dialect-APIs, die in erster Linie den Erweiterungen der Engine dienen, noch zugänglicher gestaltet. Es existieren sowohl offizielle als auch Community-getriebene Extensions *[3]*, die an dieser Stelle ansetzen. Beispielsweise kann man mit dem thymeleaf-spring-data-dialect *[4]* komfortabel Pagination in den Views ohne viel Eigenaufwand realisieren.

Fazit

Wer sauberen und wartbaren Code in seinen Views haben und zeitgemäße Template-Engines einsetzen möchte, dem sei Thymeleaf ans Herz gelegt. Basiert das aktuelle Projekt auf Spring Boot, bekommt man unter Zuhilfenahme eines Thymeleaf-Starters die passende Konfiguration mitgeliefert. Will man sich nicht an Spring Boot binden, die Engine aber dennoch in einer Spring-Anwendung nutzen, so bietet Thymeleaf Support für die Versionen 3, 4 und 5 (Milestone Releases). Wenn die Entwickler an ihre CSS-Grenzen stoßen, kann die Datei einfach von einem Designer bearbeitet werden, ohne dass Thymeleaf die Views zuerst verarbeitet haben muss. Vor allem im Gegensatz zu dem doch etwas angestaubten JSP bringt Thymeleaf wieder frischen Wind in die Entwicklung von serverseitig gerenderten Web-Anwendungen.

Weiterführende Links

[1] Thymeleaf: http://www.thymeleaf.org
[2] Spring Boot: http://start.spring.io
[3] Thymeleaf Extensions: http://www.thymeleaf.org/ecosystem.html
[4] Spring Data Extension: https://github.com/jpenren/thymeleaf-spring-data-dialect

Gerrit Meier
gerrit.meier@posteo.de

Gerrit Meier beschäftigt sich beruflich hauptsächlich mit Themen aus dem Umfeld der (Java-)Web-Entwicklung. Neue Technologien, Ideen und Ansätze – auch abseits seines technologischen Schwerpunkts – motivieren ihn immer wieder zum Ausprobieren und Verstehen. Da er dies als eine wichtige Kompetenz von Entwicklern sieht, bemüht er sich, dieses Wissen und den Spaß am „Einfach-mal-Machen" weiterzuvermitteln. Gerrit ist Co-Organisator der JUG Ostfalen.