04-2016 | Winter | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

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

Java ist vielseitig

JUnit 5
Das nächste große Release steht vor der Tür

Ansible
Konfigurationsmanagement auch für Entwickler

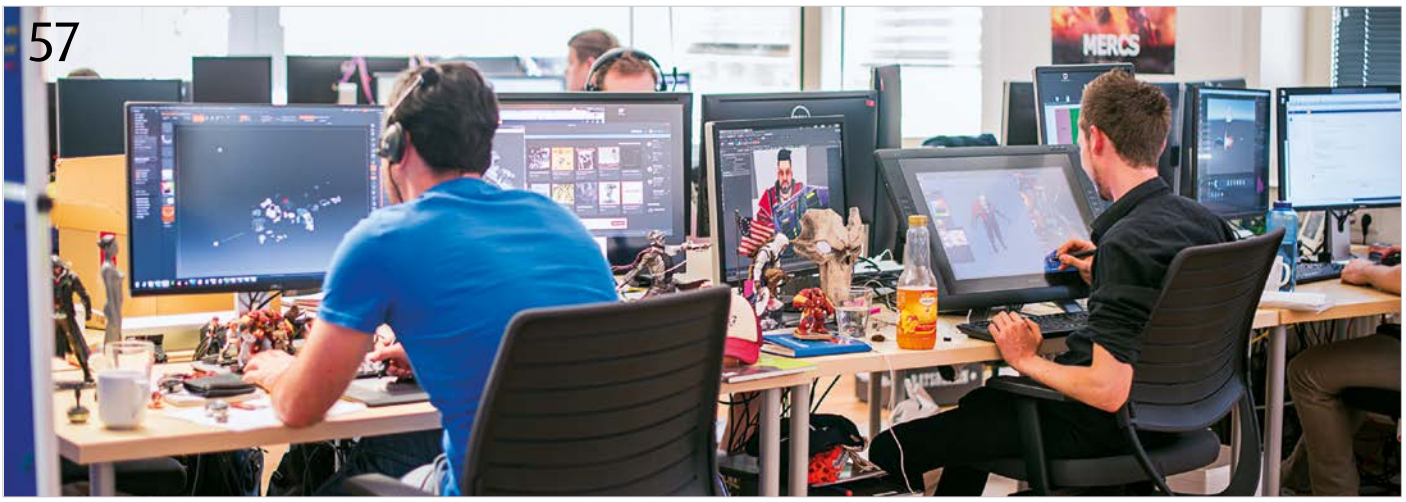
Spring Boot Starter
Komfortable Modularisierung und Konfiguration

4 191978 304903 04

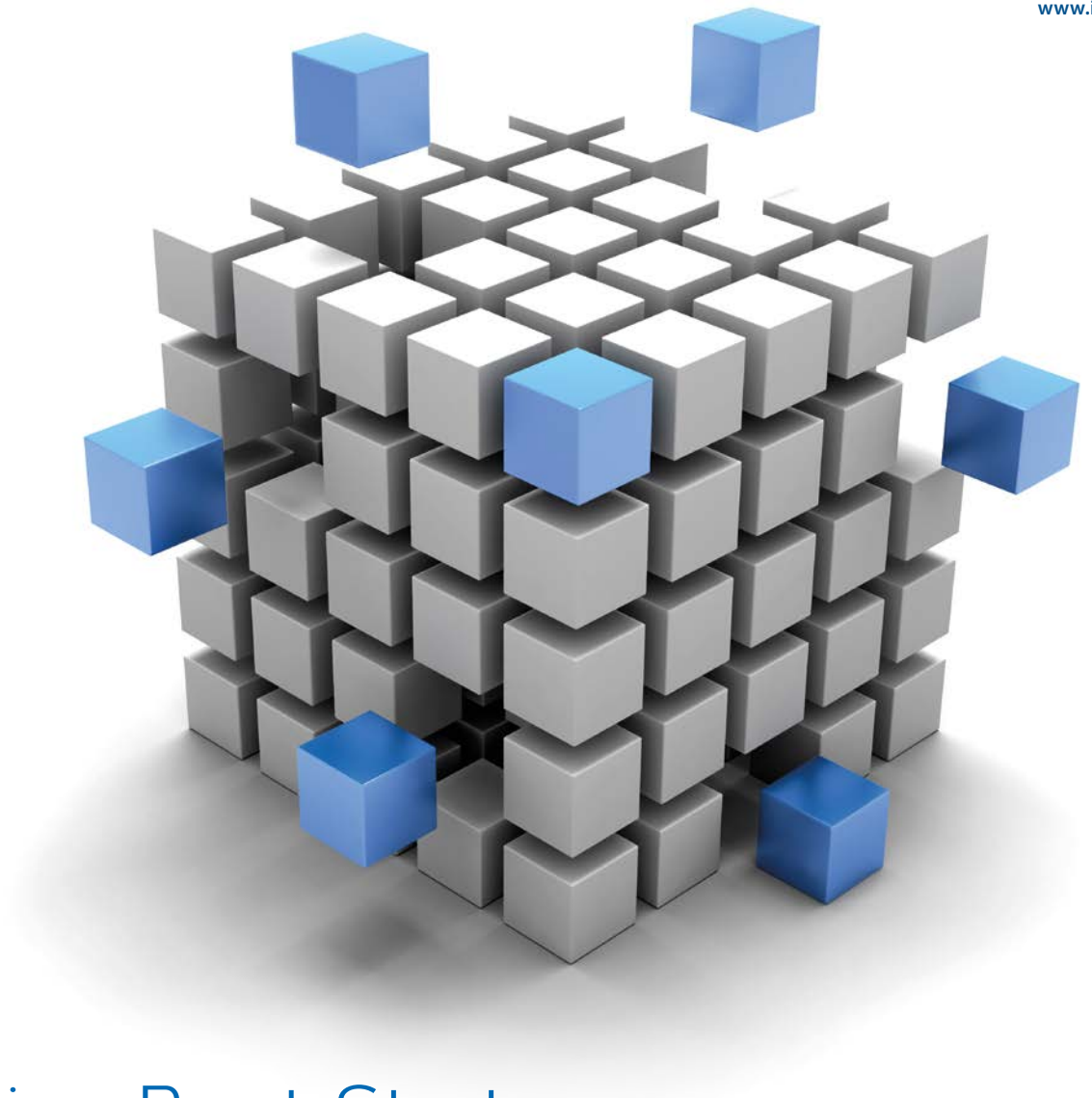
iJUG Verbund

Anwendung im laufenden Betrieb verwalten

Business Process Management auf Open-Source-Basis



Ein Blick hinter die Kulissen eines Spiele-Unternehmens



Spring Boot Starter – komfortable Modularisierung und Konfiguration

Michael Simons, ENERKO Informatik GmbH

Dieser Artikel zeigt anhand eines konkreten Beispiels, der Optimierung von Webressourcen, wie eigene Spring Boot Starter dabei helfen können, Anwendungen sauber zu modularisieren.

Auch in Zeiten von Microservices ist Architektur und sinnvolle Modularisierung wichtig. Ein Microservice ohne innere Struktur ist ebenso wenig wartbar wie ein Monolith. Auch wenn Architekturen immer feiner zergliedert werden, gibt es dennoch Module, die in mehreren Microservices genutzt werden, ohne selber wiederum ein Microservice zu sein. Konkrete Beispiele sind:

- Datenstrukturen (Entitäten) und zugehörige Repositories für wiederkehrende Aufgaben
- Logging, Status (Healthchecks) und ähnliche querschnittliche Funktionen, die gerade zum Management der immer mehr werdenden Anwendungen unerlässlich sind
- Integration (Caches, Datenbanken und vieles mehr)

Natürlich müssen diese Module nicht nur bereitgestellt werden, sie sind auch zu konfigurieren. In vielen Fällen kann dabei allerdings eine Vorgabe mit sinnvollen Standardwerten manuellen Mehraufwand erheblich verringern. Eine Möglichkeit ist es, eigene Module zur Bereitstellung von Funktionen und zur Autokonfiguration für Spring-Boot-basierte Anwendungen zu erstellen.

Spring Boot

Spring Boot *[1]* kann im weitesten Sinne als Schirmprojekt im Spring-Eco-System betrachtet werden, das mit folgenden Zielen geschaffen wurde:

- Einen schnelleren und fehlerfreieren Zugang zur Anwendungsentwicklung mit dem Spring Framework *[2]* zu ermöglichen
- Sinnvolle Vorbelegung der am häufigsten benutzen Funktionen, die aber ohne Klimmzüge überschrieben werden können, zu bieten
- Eine große Auswahl nicht-funktionaler Eigenschaften, die in annähernd jedem Projekt benötigt werden, zur Verfügung zu stellen

Während des Bootstrapping von Anwendungen und deren Autokonfiguration wird zu keinem Zeitpunkt Code generiert. Alle Mittel, die von Spring Boot genutzt werden, um festzustellen, welche Bibliotheken verfügbar sind, in welchem Umfeld eine Anwendung läuft und wie sie konfiguriert ist, stehen auch außerhalb des Frameworks zur Entwicklung eigener Starter zur Verfügung.

JavaScript im Griff mit Maven, WebJars und Wro4j

Auch im Jahr 2016 soll es den Bedarf geben, Webanwendungen mit serverseitig gerendertem HTML-Code, JavaScript als progressive Verbesserung und einfacher Verwaltung von JavaScript zu erstellen, ohne das „Universal Install Script" benutzen zu müssen *(siehe Abbildung 1)*.

Wir haben alle schon über Maven geflucht, aber in der Regel funktioniert Maven unauffällig und ist ziemlich gut darin, Abhängigkeiten zu verwalten, aufzulösen und widersprüchliche Versionen (wie mit dem „enforcer-plugin") zu entdecken.

WebJars *[3]* ist ein Projekt von James Ward *[4]* und ermöglicht die explizite und einfache Verwaltung von JavaScript, CSS und ähnlichen Abhängigkeiten mit Maven, SBT, Gradle, ivy oder anderen Build-Tools. Spring Boot unterstützt WebJars unmittelbar und Abhängigkeiten können direkt genutzt werden *(siehe Listing 1)*.

Wro4j *[5]* ist ein Werkzeug zur Analyse und Optimierung von Webressourcen, das viele moderne Werkzeuge aus der Webentwicklung zusammenbringt. Ziel ist es, Abhängigkeiten im pom.xml zu definieren *(Listing 2)*.

Daraus entstehen dann logische Einheiten. In diesem kleinen Beispiel soll eine Library mit allen fremden Ressourcen zusammen mit den eigenen JavaScript-Programmen ein gemeinsames Modul bilden. Dabei wird die Wro4j-XML-Notation genutzt, es steht darüber hinaus auch eine Groovy-DSL zur Verfügung *(siehe Listing 3)*.

Diese Konfiguration sollte mit dem zu erstellenden Starter ausreichen, um jQuery, AngularJS und eigene Funktionen in eine Webseite über ein einziges Element einzubinden.

Spring Boot Starter

Zu einem Starter gehören folgende Komponenten:

- Ein „autoconfigure"-Modul, das die automatische Konfiguration mit ihren entsprechenden Standardwerten beinhaltet
- Das eigentliche „starter"-Modul, das vom „autoconfigure"-Modul abhängig ist und gegebenenfalls weitere Abhängigkeiten mitbringt

In der eigentlichen Anwendung ist das „starter"-Modul eingebunden. Natürlich können beide Module in einem Modul kombiniert werden.

Pivotals Spring Developer Advocate Josh Long *[6]* spricht sich explizit dagegen aus, die Autokonfiguration durch Spring Boot als Magie zu bezeichnen, und er hat Recht damit. Die automatische Konfiguration der Umgebung basiert auf zwei Kernkomponenten

- Profilen („@Profile", verfügbar seit Spring 3.1)
- Bedingungen („@Conditional" im Zusammenspiel mit „org.springframework.context.annotation.Condition", verfügbar seit Spring 4)

Profile ermöglichen die Bereitstellungen von Beans auf Basis aktivierter Umgebungen („spring.profiles.active"), Bedingungen ermöglichen einfache bis komplexe Bedingungen, unter denen Beans instanziiert und konfiguriert werden:

- „@Conditional" aus dem Kernframework selber ermöglicht mit der Implementierung eigener Bedingungen beliebige Abfragen

```
<link rel='stylesheet' href='/webjars/bootstrap/3.1.0/css/bootstrap.min.css'>
```

Listing 1

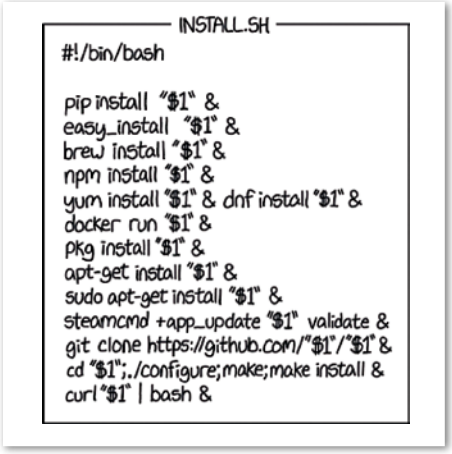


Abbildung 1: The Universal Install Script [18]

```
<properties>
     <jquery.version>1.11.3</jquery.version>
     <angularjs.version>1.2.29</angularjs.version>
</properties>
<!-- ... -->
<dependencies>
     <dependency>
          <groupId>org.webjars</groupId>
          <artifactId>jquery</artifactId>
          <version>${jquery.version}</version>
     </dependency>
     <dependency>
          <groupId>org.webjars</groupId>
          <artifactId>angularjs</artifactId>
          <version>${angularjs.version}</version>
     </dependency>
</dependencies>
```

Listing 2

```
<groups xmlns="http://www.isdc.ro/wro">
     <group name="lib">
          <js minimize="false">/webjars/jquery/@jquery.version@/jquery.min. js</js>
          <js minimize="false">/webjars/angularjs/@angularjs.version@/angular.min.js</js>
          <js minimize="false">/webjars/angularjs/@angularjs.version@/angular-route.min.js</js>
     </group>

     <group name="biking2">
          <group-ref>lib</group-ref>
          <css>/css/stylesheet.css</css>
          <js>/js/app.js</js>
          <js>/js/controllers.js</js>
          <js>/js/directives.js</js>
     </group>
</group>
```

Listing 3

Aus Spring Boot selbst einige Highlights:

- „@ConditionalOnBean / @ConditionalOnMissingBean": Konfiguration oder Beans werden nur erzeugt, wenn bestimmte andere Beans vorliegen oder fehlen
- „@ConditionalOnClass / @ConditionalOnMissingClass": Konfiguration oder Beans werden nur erzeugt, wenn bestimmte Klassen auf dem Klassenpfad verfügbar sind
- „@ConditionalOnResource": Tritt nur in Kraft, wenn spezifische Ressourcen verfügbar sind
- „@ConditionalOnProperty": Eine mächtige Bedingung zur Abfrage auf konkrete Eigenschaften in Konfigurationsdateien

So sehr die automatische Umgebung magisch wirkt, sie ist es nicht. Spring Boot stellt mit dem ConditionEvaluationReport *[7]* sogar eine Reporting-Lösung zur Verfügung, mit der genau nachvollzogen werden kann, warum, wie und in welcher Reihenfolge automatische Konfiguration durchgeführt wurde. Übrigens nutzt auch Spring seine eigenen Features intensiv: Die „@Profile"-Annotation wurde im Rahmen von Spring 4 neu geschrieben und basiert nun selber auf „Conditions".

Erstes Beispiel: Bereitstellung des Wro4j-Servlet-Filters

Der Kern des Wro4j-Pakets ist der Servlet-Filter, der unter einem bestimmten Mapping Anfragen für CSS- und JavaScript-Dateien entgegennimmt und entsprechend seiner Konfiguration verarbeitet. Durch Hinzufügen des wro4j-spring-boot-starter soll dieser Filter ohne weitere Nacharbeit bereitgestellt werden *(siehe Listing 4)*.

Die „Wro4jAutoConfiguration" ist eine gewöhnliche Spring-„@Configuration"-Klasse, in der eine Bean programmatisch über Java-Code instanziiert und konfiguriert werden kann. Die in ihr vorgenommene Konfiguration soll dann ausgeführt werden, wenn die „WroFilter"-Klasse verfügbar ist und es noch keine Bean entsprechenden Typs gibt. Sie soll nach der automatischen Cache-Konfiguration durchgeführt werden, da diese benötigt wird. Ferner stellt sie alle Eigenschaften, die mit „wro4j.*" beginnen, als Objekt mit entsprechenden Attributen zur Verfügung.

```
@Configuration
@ConditionalOnClass(WroFilter.class)
@ConditionalOnMissingBean(WroFilter.class)
@EnableConfigurationProperties(Wro4jProperties.class)
@AutoConfigureAfter(CacheAutoConfiguration.class)
public class Wro4jAutoConfiguration {
}

@ConfigurationProperties("wro4j")
public class Wro4jProperties {
}
```

Listing 4

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration = ac.simons.
spring.boot.wro4j.Wro4jAutoConfiguration
```

Listing 5

```
@Bean
ConfigurableWroFilter wroFilter(WroManagerFactory wroManagerFactory,
Wro4jProperties wro4jProperties) {
     ConfigurableWroFilter wroFilter = new ConfigurableWroFilter();
     wroFilter.setProperties(wroFilterProperties(wro4jProperties));
     wroFilter.setWroManagerFactory(wroManagerFactory);
return wroFilter;
}
```

Listing 6

Entsteht der eigene Spring Boot Starter innerhalb des Projektes, in dem er zuerst benutzt wird, übersieht man sehr schnell „META-INF/spring.factories", da alle „@Configuration"-Klassen mit ihren Bedingungen in einem Spring-Boot-Projekt automatisch ausgewertet werden. „spring.factories" ähnelt nicht ganz zufällig dem eingebauten Java Service Provider Interface. Die „Factories"-Definition für den Wro4j-Starter sieht ganz einfach aus *(siehe Listing 5)*.

Diese Aufzählung von Konfigurationsklassen führt beim Laden der Bibliothek zu ihrer Auswertung und vermeidet dadurch sowohl ein manuelles Aufzählen als auch einen vollständigen Scan des Klassenpfads nach „@Component"-Beans. Doch zurück zum Beispiel. Der eigentliche Filter ist nichts anderes als eine normale Spring Bean und wird wie folgt in „Wro4jAutoConfiguration" erzeugt *(siehe Listing 6)*.

Gut erkennbar: Der Filter hängt von einer WroManagerFactory und zusätzlichen Eigenschaften ab, die in diese Methode injiziert werden. Am Beispiel der automatischen Konfiguration zeigt sich die Mäch-

```
@Bean
@ConditionalOnBean(CacheManager.class)
@ConditionalOnProperty("wro4j.cacheName")
@ConditionalOnMissingBean(CacheStrategy.class)
@Order(-100)
<K, V> CacheStrategy<K, V> springCacheStrategy(CacheManager cacheManager, Wro4jProperties wro4jProperties) {
     return new SpringCacheStrategy<K, V>(cacheManager, wro4jProperties.getCacheName());
}

@Bean
@ConditionalOnMissingBean(CacheStrategy.class)
@Order(-90) <K, V> CacheStrategy<K, V> defaultCacheStrategy() {
     return new LruMemoryCacheStrategy<K, V>();
}
```

Listing 7

tigkeit der Java-basierten Konfiguration von Spring Beans: Komplexe Bedingungen lassen sich in Quelltext ausdrücken und mit Werkzeugunterstützung aller aktuellen IDEs nachvollziehen.

Zweites Beispiel: Unterschiedliche Laufzeitumgebung

Wro4j unterstützt Caching von optimierten Ressourcen in Form einer einfachen, speicherbasierten „Least Recently Used Strategy". Eine Implementierung dieser „CacheStrategy" ist schnell geschrieben und mit wenig Code lässt sich „org.springframework.cache.CacheManager" nutzen, sofern zur Laufzeit eine Instanz vorliegt. Das ist zum Beispiel immer dann der Fall, wenn Spring „@EnableCaching" genutzt wird; Spring unterstützt eine Vielzahl von Caches, unter anderem „ehcache" und „Redis". Ist für die jeweilige Applikation kein Cache konfiguriert, wird die standardmäßige Wro4j-Implementierung genutzt *(siehe Listing 7)*.

Die „springCacheStrategy"-Methode wird nur dann ausgeführt, wenn ein „CacheManager" aktiv, der Name des zu nutzenden Cache konfiguriert und noch keine anderweitig konfigurierte Strategie vorhanden ist. Durch „@Order"-Annotation wird sichergestellt, dass die Methode vor „defaultCacheStrategy" ausgewertet wird, die die Default-Implementierung von Wro4j nutzt.

Es gibt sicherlich Stimmen, die dieses Beispiel gerne auf „@Annotatiomania" *[8]* sehen würden, aber gerade zur Konfiguration außerhalb von Geschäftslogik und fachlichen Objekten hält der Autor Annotationen für ein probates und gut lesbares Werkzeug, um Bedingungen zu definieren.

Das Deployment

Das Beispiel ist ein normales Maven-Projekt, in dem ein gemeinsames „autoconfigure"- und „starter"-Modul implementiert wurde. Es bringt daher die Abhängigkeiten zu Wro4j direkt mit *(siehe Listing 8)*.

```
<dependencies>
     <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter</artifactId>
     </dependency>

     <dependency>
          <groupId>ro.isdc.wro4j</groupId>
          <artifactId>wro4j-core</artifactId>
          <version>${wro4j.version}</version>
     </dependency>

     <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
     </dependency>
</dependencies>
```

Listing 8

```
<resources>
     <resource>
          <directory>src/main/resources</directory>
          <filtering>true</filtering>
          <includes>
               <include>wro.xml</include>
          </includes>
     </resource>
</resources>
```

Listing 9

```
wro4j.filterUrl = /owr
wro4j.managerFactory.preProcessors = removeSourceMaps, cssUrlRewriting, cssImport, cssMinJawr, semicolonAppender, jsMin
wro4j.cacheName = wro4j
```

Listing 10

Bei zwei separaten Projekten werden die Abhängigkeiten des „autoconfigure"-Moduls in der Regel als „optional" gekennzeichnet und erst im „starter"-Modul konkretisiert. Die Besonderheit des hier erstellten Starters liegt darin, dass er als Child-Artefakt vom offiziellen „spring-boot-starters"-Projekt erbt, weil der Autor sich bei diesem Projekt an die Kodierungsrichtlinien des Spring-Teams halten wollte. Erleichternd dabei war die Tatsache, dass NetBeans *[9]* Checkstyle-Regeln ebenso direkt unterstützt wie JaCoCo. Der vollständige Quelltext des Projekts ist auf GitHub *[10]* verfügbar.

Fazit

Der hier beschriebene Spring Boot Starter wird in mehreren Projekten produktiv ein-

gesetzt, unter anderem bei der Euregio JUG *[11]*. Durch Einbinden des Starters entfallen das manuelle Einbinden von Abhängigkeiten, zusätzliche Konfiguration und vieles mehr.

Es gibt wenige Alternativen zu Spring Boot Startern im Spring-Umfeld, wenn gemeinsame Module für Anwendungen erstellt werden sollen. Wird Annotations-basierte Konfiguration in diesen Modulen eingesetzt (zum Beispiel „@Service" und Ähnliche), so muss Sorge dafür getragen werden, dass diese Komponenten auch im Klassenpfad gefunden werden. XML-basierte Konfiguration ist aufwändig und oft fehlerträchtig.

Das „spring-boot-starters"-Repository *[12]* listet zahlreiche Starter auf, von Datenbank-Anbindungen über Template-Sprachen bis hin zu Remote Shells ist viel Nützliches dabei. Der Autor nutzt weitere Starter, zum Beispiel zur Bereitstellung von Entitäten und Repositories zur Speicherung von Oauth-Credentials.

Das Ziel des „wro4j-starter" wurde vollständig erfüllt. In *Listing 3* sieht man, dass die „wro.xml"-Konfigurationsdatei Maven-Platzhalter wie zum Beispiel „@jquery.version@" enthält, die während des Build-Vorgangs durch die entsprechenden Maven-Properties ersetzt werden *(siehe Listing 9)*.

Die automatische Konfiguration liest „application.properties" aus *(siehe Listing 10)*. Das Ergebnis ist jeweils eine minimierte CSS- und JavaScript-Datei, die mit einem Statement eingebunden werden kann. Diese Optimierung funktioniert auch problemlos mit AngularJS-SPA-Anwendungen.

Ausblick

Der Starter lässt sich dahingehend erweitern, dass je nach Umgebung die URL der optimierten Ressourcen oder die Informationen aus dem Modell in die Seite generiert werden. Das Vorgehen wäre identisch: Feststellen, ob Unterstützung für Template-Sprachen wie Thymeleaf *[13]* oder andere auf dem Klassenpfad sind, und entsprechende zusätzliche Tags registrieren.

Quellen und Links

[1] Spring Boot: *http://projects.spring.io/spring-boot*
[2] Spring Framework: *https://projects.spring.io/spring-framework*
[3] WebJars: *http://www.webjars.org*
[4] James Ward: *https://twitter.com/_JamesWard*
[5] wro4j: http://alexo.github.io/wro4j
[6] Josh Long: *https://twitter.com/starbuxman/status/708579187087511552*
[7] ConditionEvaluationReport: *http://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/autoconfigure/condition/ConditionEvaluationReport.html*
[8] @Annotatiomania: *http://www.annotatiomania.com*
[9] NetBeans: *http://www.netbeans.org*
[10] GitHub: *https://github.com/michael-simons/wro4j-spring-boot-starter*
[11] Euregio JUG: *http://www.euregjug.eu*
[12] spring-boot-starters: *https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters*
[13] Thymeleaf: *http://www.thymeleaf.org*
[14] ENERKO Informatik: *http://www.enerko-informatik.de*
[15] info.michael-simons.eu: *http://info.michael-simons.eu*
[16] The primary goals of Spring Boot: *https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone*
[17] Creating your own starter: *http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-custom-starter*
[18] Universal install script: *http://xkcd.com/1654*
[19] Maven Enforcer Plugin - The Loving Iron Fist of Maven: *http://maven.apache.org/enforcer/maven-enforcer-plugin*

Michael Simons
michael@simons.ac

Michael Simons ist Software-Architekt (CPSA-F) bei ENERKO Informatik [14] in Aachen und entwickelt dort GIS-, EDM- und Vertriebsmanagement-Systeme für Stromnetzbetreiber und Energielieferanten. Michael ist Mitgründer der Euregio JUG [11] und schreibt über ihre Entwicklung und andere Dinge unter info.michael-simons.eu [15].

Veranstaltungen der im iJUG organisierten JUGs auf Erfolgskurs

Von Java User Groups organisierte Fachkonferenzen sind voll im Trend und erfreuen sich stark steigender Besucherzahlen. Mit Java Forum Stuttgart, Berlin Expert Days, Java Forum Nord und JavaLand sind die führenden deutschen Java-Konferenzen von der Community organisiert.

Mit einer Rekordbeteiligung von rund 1.700 Teilnehmern fand zum 19. Mal das Java Forum in Stuttgart statt. Die Java User Group Stuttgart hatte 49 Vorträge in sieben parallelen Tracks organisiert. Zudem waren 34 Aussteller vor Ort, darunter auch der Interessenverbund der Java User Groups e.V. (iJUG).

Am 15. und 16. September 2016 waren in Berlin die Berlin Expert Days. Der Verein Berlin Expert Days e.V. wurde im Jahr 2010 mit dem Ziel gegründet, eine Plattform zum Informationsaustausch anzubieten. Als offen geführter Verein steht eine solide Basis bereit, um die Unabhängigkeit von Herstellern und Dienstleistern zu gewährleisten. Auf der diesjährigen Veranstaltung waren mehr als 500 Teilnehmer auf den 44 Vorträgen.

Das Java Forum Nord öffnet am 20. Oktober 2016 seine Pforten. Die eintägige, nicht-kommerzielle Konferenz in Norddeutschland mit Themenschwerpunkt Java ist für Entwickler und Entscheider. Mit mehr als 25 Vorträgen in parallelen Tracks und einer Keynote wird ein vielfältiges Programm zu einem unschlagbaren Preis geboten, der regionale Bezug bietet zudem interessante Networking-Möglichkeiten. Auch das Datum für die JavaLand 2017 steht bereits fest. Sie findet vom 28. bis 30. März 2017 an gewohnter Stätte im Phantasialand Brühl statt. Mit mehr als 1.200 Teilnehmer gehört die JavaLand zu den größten Java-Konferenzen Europas. Für drei Tage wird der Freizeitpark zum Zentrum der Java-Community.