DVD auf S.3

BONUS ÜBER 150 MINUTEN VIDEOMATERIAL + BUCHAUSZÜGE + TOOLS!

S&S

Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 10.2012

JAVA Mag

Java magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

W-JAX 12
Programm im Heft!
▸ 35

COOL TOOLS

Java-Werkzeuge, die Ihr tägliches Leben erleichtern können ▸ 39

4 194586 709801 10

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

DSL-Komposition
Integration einer Groovy-DSL ▸ 16

GWT meets Ajax und PHP
Wie GWT mit seinen Spielkameraden redet ▸ 88

Windows Azure
PaaS auch für Java-Entwickler ▸ 72

Kolumne: EnterpriseTales

von Lars Röwekamp und Matthias Weßendorf

Einer für alle – alle für einen: Bean Validation 1.1

Es gibt wohl kaum eine andere Java-Spezifikation, die so eng mit der Community verbunden ist wie das Bean-Validation-API. Erwachsen aus einem Open-Source-Projekt, sind auch weiterhin die Anforderungen und Wünsche der Anwender maßgeblicher Treiber für Neuerungen und Verbesserungen. Wie die kommende Version in etwa aussehen könnte, zeigt ein Blick in die Kristallkugel aka JSR 349 Early Draft.

Das Bean-Validation-API ist deutlich mehr als nur eine Spezifikation inklusive Referenzimplementierung. Ein Blick auf die zugehörige Webseite [1] macht klar, wie auch in einen offiziellen JSR die Community gewinnbringend mit eingebunden werden kann und so der ursprüngliche Open-Source-Gedanke erhalten bleibt. So wundert es auch nicht, dass ein Großteil der bisher für die aktuell als Early Draft vorliegende Version 1.1 [2] angedachten Neuerungen und Erweiterungen das direkte Feedback der Nutzergemeinde darstellt. Bevor wir das eine oder andere zukünftige Feature im Detail anschauen, soll kurz ein Überblick über die wichtigsten zu erwartenden Neuerungen gegeben werden:

- *Integration mit weiteren APIs*: bessere Integration in weitere JSRs wie JAX-RS, JAXB, EJB sowie JPA und CDI
- *Method-Level Validation*: Validierung von Methodenparametern und -rückgabewerten
- *Dependency Injection*: CDI-Support für Bean Validation Components (*MessageInterpolator*, *ConstraintValidatorFactory* und *ConstraintValidator*)
- *Constraint Composition*: Erweiterung der Komposition von bestehenden Constraints durch OR-Verknüpfungen. Aktuell sind Kompositionen automatisch AND-Verknüpfungen.
- *Group Propagation*: Überführung einer Constraint-Gruppe in eine andere Gruppe bei kaskadierenden Validierungen
- *Constraints für Collection-Elemente*: Aktuell werden lediglich Constraints auf einer Collection selbst oder kaskadierende Constraints auf den enthaltenen Elementen unterstützt. Zukünftig soll auch die direkte Angabe von Constraints auf Collection-Element-Ebene möglich sein, wodurch zum Beispiel die Validierung von nativen Typen unterstützt werden wird. Dies ist eigentlich nur ein Workaround für die nach wie vor fehlende Unterstützung von Annotationen auf Java-Typen (JSR 308 – Early Draft).

Integration

Wenn auch nicht offizieller Bestandteil des Bean-Validation-API, legt die Spezifikation großen Wert darauf, sich optimal in die verschiedenen Lifecycles der Java-SE/EE-Plattformen einzubetten. Ein gutes Beispiel hier-

Porträt

W-JAX Speaker

Lars Röwekamp ist Geschäftsführer der open knowledge GmbH und berät seit mehr als zehn Jahren Kunden in internationalen Projekten rund um das Thema Enterprise Computing (Twitter: @mobileLarson).

Matthias Weßendorf beschäftigt sich mit WebSocket, HTML5 und weiteren Themen rund um das Next Generation Web. Er bloggt gelegentlich auf http://matthiaswessendorf.wordpress.com (Twitter: @mwessendorf).

für ist die bereits heute schon bestehende Integration in JSF 2 und JPA 2. Angedacht für die Version 1.1 ist insbesondere eine zusätzliche Integration in JAX-RS zur besseren Validierung von *WebService*-Parametern und *Return*-Werten, sowie eine Integration in JAXB zur Konvertierung von *BeanValidation* Annotation in XML-Schema-Deskriptoren (vice versa).

Aber auch bereits vorhandene Integrationen sollen verbessert werden. Innerhalb von JPA soll zukünftig die Generierung von DDL-Fragmenten mittels Bean Validation Constraints am Entity-Modell gesteuert werden können. EJB und CDI werden neben der aktuell vorhandenen Property-Level-Validation zukünftig auch Method-Level-Validation unterstützen (siehe unten). Ebenfalls in Planung sind JSF Class Level Constraints, mit deren Hilfe clientseitige Validierung auf Basis von Java Constraints ermöglicht werden soll, wie man es heute bereits von einigen Third-Party Libraries her kennt.

Listing 1

```
public class OrderService {

   public OrderService(@NotNull CreditCardProcessor creditCardProcessor) {
      //...
   }

   public void placeOrder(
      @NotNull @Size(min=3, max=20) String customerCode,
      @NotNull Item item,
      @Min(1) int quantity) {
      //...
   }
}
```

Listing 2

```
public class OrderService {

   private CreditCardProcessor creditCardProcessor;

   @ValidOnlineOrderService
   public OrderService(OnlineCreditCardProcessor creditCardProcessor) {
      this.creditCardProcessor = creditCardProcessor;
   }

   @ValidBatchOrderService
   public OrderService(BatchCreditCardProcessor creditCardProcessor) {
      this.creditCardProcessor = creditCardProcessor;
   }

   @NotNull
   @Size(min=1)
   public Set<CreditCardProcessor> getCreditCardProcessors() { ... }

   @NotNull
   @Future
   public Date getNextAvailableDeliveryDate() { ... }
}
```

Method-Level-Validation

Eine der wohl wesentlichsten und übergreifenden Neuerungen in der Bean-Validation-1.1-Spezifikation stellt die Method-Level-Validation dar. Sie erlaubt zukünftig das Platzieren von Constraints an Methoden- und Konstruktor-Parametern sowie an Methodenrückgabewerten.

Das als „Programming by Contract“ (kurz: *PbC*) bekannte Pattern erlaubt über diesen Weg, Pre- und Post-Conditions für Methodenaufrufe zu deklarieren. Neben dem Vorteil der besseren Lesbarkeit – auch ohne Zusatzdokumentation – werden so redundante Prüfungen im aufrufenden Code obsolet. Potenziell kann Method-Level-Validation auf jeder *non-static*-Methode angewendet werden. Es ist aber durchaus erlaubt, dass einzelne Spezifikationen zusätzliche Restriktionen – zum Beispiel *public* und/oder *non-final* – mit ins Feld führen. Listing 1 zeigt ein einfaches Beispiel für Method-Level-Parameter-Validation aus der Spezifikation.

Damit eine möglichst sinnvolle Darstellung von Validierungsfehlern ermöglicht wird, stellt Bean Validation einen *ParameterNameProvider* zur Verfügung, der den Namen des fehlerhaften Parameters liefert. Da dieser in der Regel – zum Beispiel für den Endbenutzer einer Webanwendung – wenig Aussagekraft haben dürfte, soll es zukünftig ebenfalls möglich sein, eigene *ParameterNameProvider* zu implementieren und via Bootstraping-API- oder XML-Konfiguration (via *parameter-name-provider*-Tag) einzubinden.

Mit sehr hoher Wahrscheinlichkeit wird es für die Parameter-Validation auch einen Mechanismus geben, der Cross-Parameter Constraints erlaubt. Den aktuellen Stand der Diskussion dazu findet man unter [3] und [4]. Derzeit sind drei unterschiedliche Varianten im Gespräch. Die erste Variante legt nahe, auf einen Support vollständig zu verzichten. Die zweite Variante sieht ein Interface namens *MethodConstraintValidator* vor, dessen Methode *isValid(...)* beliebige Cross-Parameter-Validierungen ermöglicht und einen Zugriff auf die einzelnen Parameter via Signatur-Position erlaubt. Die dritte und letzte Variante verzichtet auf das Interface der zweiten Variante und setzt stattdessen eine *isValid(...)*-Methode voraus, die in ihrer Signatur mit der zu prüfenden Methode übereinstimmt. Auch eine Kombination aus der zweiten und dritten Variante ist durchaus denkbar. Neben den Parametern einer Methode kann – wie bereits beschrieben – auch deren Rückgabewert mit Constraints versehen werden. Das in Listing 2 aufgeführte Beispiel zeigt, dass neben „built-in“ Constraints selbstverständlich auch eigene Varianten zur Prüfung herangezogen werden können.

Eine Frage, die sich bei der Method-Level-Validation fast automatisch stellt, ist, wie es sich bei der Validierung von Methoden bei Vererbung verhält. Hier verweist die Spezifikation auf das Liskovsche Substitutionsprinzip [5] – es ist keine Schande, wenn man dieses erst nachschlagen muss – und legt fest, dass Parameter-Constraints in Subtypen nicht verstärkt und Return Value Constraints in Subtypen nicht geschwächt werden sollten.

Fazit

Dank Bean Validation ist eine einheitliche Validierung im gesamten Java EE/SE Stack möglich. Schichtenspezifische Lösungen werden obsolet. Eine derartige Lösung hat allerdings nur dann Aussicht auf Erfolg, wenn eine gute Integration mit anderen APIs gewährleistet ist. Dass dies funktioniert, hat Bean Validation bereits in der Version 1.0 am Beispiel von JSF und JPA bewiesen. JAX-RS, JAXB und weitere werden in der Version 1.1 folgen.

Ein weiterer Baustein für den zukünftigen Erfolg von Bean Validation stellt die Möglichkeit zur Validierung von Methoden-Parametern und -rückgabewerten dar. „Programming by Contract“ inklusive Pre- und Post-Conditions erlauben einen zentralen Validierungsansatz und ersparen so eine Menge redundanten Code. Ob dieses Feature allerdings bereits in Java EE 7 in den zugehörigen APIs integriert werden wird, steht noch in den Sternen. Der aktuelle Stand der Spezifikation – Early Draft – macht einen sehr guten Eindruck. Wir dürfen gespannt sein, was sich bis zur Final Version noch alles tut. In diesem Sinne: Stay tuned ...

Links & Literatur

[1] Bean-Validation-Specification-Website: http://beanvalidation.org/

[2] Bean Validation 1.1 (JSR 349): http://jcp.org/en/jsr/detail?id=349

[3] Cross-Parameter Constraints Diskussion: http://beanvalidation.org/proposals/BVAL-241/#cross_parameter

[4] Cross-Parameter Constraints: https://hibernate.onjira.com/browse/BVAL-232

[5] Liskovsches Substitutionsprinzip: http://de.wikipedia.org/wiki/Liskovsches_Substitutionsprinzip

Anzeige

ANZEIGE