04-2015 | Winter | www.ijug.eu

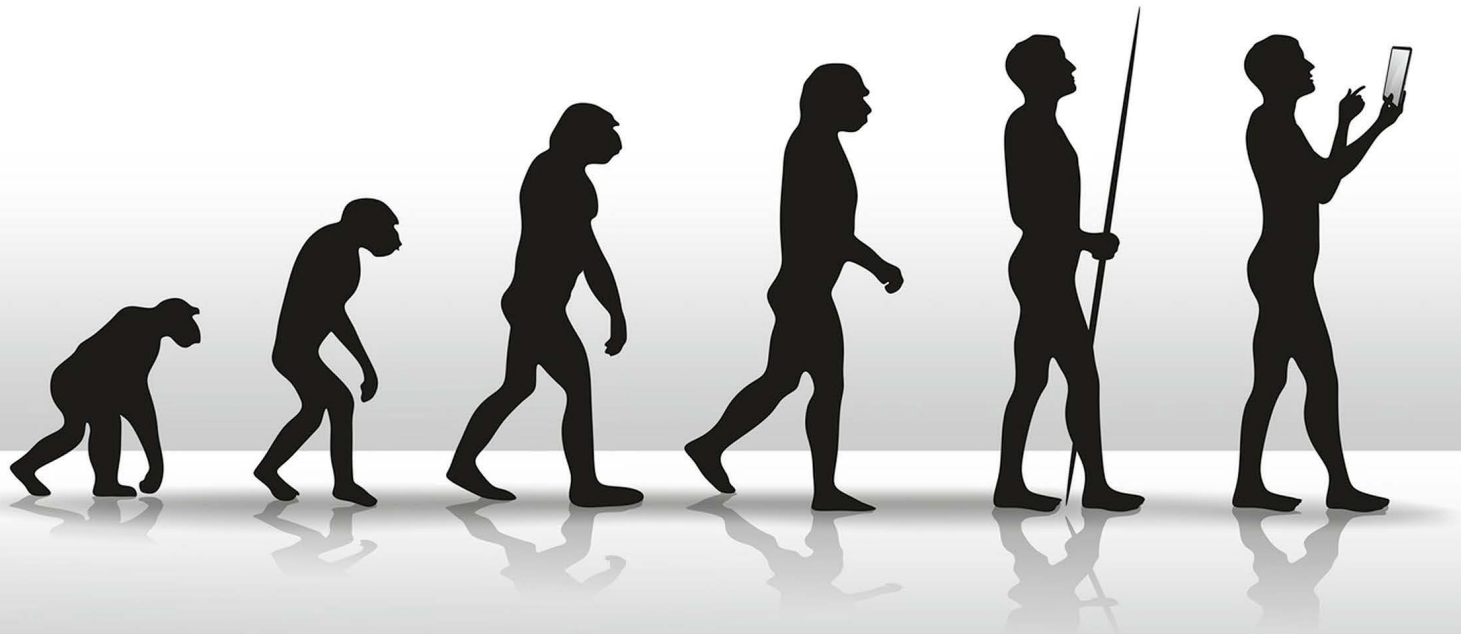
Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

Java entwickelt sich weiter



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

Neue Frameworks
AspectJ, Eclipse Scout, Citrus

Raspberry Pi
Projekte mit Java

Java-Performance
Durch Parallelität verbessern

Web-Anwendungen
Hochverfügbar und performant

iJUG Verbund

4 191978 304903 04

Die Position „Software-Architekt" ist in der Software-Branche etabliert

Java-Champion Kirk Pepperdine gibt Performance-Tipps

Citrus bietet komplexe Integrationstests mit mehreren Schnittstellen

Wo steht CDI 2.0?

Thorben Jannsen und Anatole Tresch, Credit Suisse

Die Arbeiten an Java EE 8 sind in vollem Gange und einige Expert Groups haben mit dem Early Draft Review (EDR) bereits einen ersten offiziellen Zwischenstand veröffentlicht. Der Artikel zeigt den aktuellen Stand der Spezifikation für den Context and Dependency Inject for Java (CDI) 2.0.

Das Early Draft Review (EDR) soll der Entwicklergemeinschaft die Möglichkeit geben, einen Teil der geplanten Änderungen auszuprobieren und mit ihrem Feedback bereits früh die weitere Entwicklung der Spezifikation zu beeinflussen. Zum Zeitpunkt der Erstellung dieses Artikels steht auch die Expert Group für CDI 2.0 *(siehe „http://cdi-spec.org")* kurz vor der Veröffentlichung des EDR.

Die Liste der geplanten Änderungen für die Version 2.0 ist lang und in folgende Themen unterteilt:

- Verbesserung des Event-Systems
- Verwendung im Java-SE-Umfeld
- Unterstützung von Java-8-Sprachfeatures
- Anpassung der Interceptoren und Dekoratoren zur Verbesserung der aspektorientierten Programmierung
- Modularisierung und Anpassungen der SPI
- Kontexte

Das EDR wird allerdings nur die Änderungen beinhalten, die aus Sicht der Expert Group bereits einen ausreichenden Reifegrad erreicht haben, um von der Community ausprobiert und diskutiert zu werden. Das dabei erhaltene Feedback fließt anschließend in die Spezifikation mit ein und kann somit zu weiteren Änderungen führen. Die Hauptthemen des EDRs sind:

- Sortierung von Events
- Asynchrone Verarbeitung von Events
- Unterteilung der Spezifikation in einen Java-EE- und einen Java-SE-Teil
- Ein Bootstrap-Mechanismus für Java-SE-Umgebungen

Sortierung von Events

Die Erweiterung des Event-Systems war eine der von der Community am häufigsten gewünschten Änderungen für CDI 2.0. Dass dies von der Expert Group entsprechend ernst genommen wird, zeigt sich an den im EDR enthaltenen Änderungen. Die Sortierung von Event-Observern war eines der ersten Features, für die ein konkreter Vorschlag erarbeitet und die in das EDR aufgenommen wurden.

Bisher wurden die Observer-Methoden vom Container in unbestimmter Reihenfolge aufgerufen, sodass eine gesteuerte Verkettung mehrerer, aufeinander aufbauender Observer nicht möglich war. Ab Version 2.0 soll der CDI-Container die Event-Observer basierend auf ihrer Priorität sortieren und anschließend in aufsteigender Reihenfolge aufrufen. Die Priorität kann für jede Oberserver-Methode mithilfe der „@Priority"- Annotation definiert werden *(siehe Listing 1)*.

Observer ohne deklarierte Priorität erhalten die Default-Priorität „APPLICATION + 500". Wenn mehrere Observer-Methoden über denselben Prioritätswert verfügen, ist die Aufruf-Reihenfolge nicht definiert und entspricht somit dem bisherigen Verhalten. Die beschriebene Funktionalität wurde bereits im Release 3.0.0.Alpha1 der CDI-Referenz-Implementierung „Weld" prototypisch implementiert und kann dort ausprobiert werden.

Asynchrone Verarbeitung von Events

Ein weiteres, von der Community häufig gewünschtes Feature ist die asynchrone Verarbeitung von Events. Bisher wurden die Event-Observer vom CDI-Container synchron innerhalb des Thread des Event-Producers ausgeführt. Wenn ein Event ausgelöst wurde, stoppte daher die Verarbeitung des Event-Producers, bis alle Observer das Event verarbeitet hatten oder dieses durch eine Exception abgebrochen wurde.

Ab Version 2.0 soll auch eine asynchrone Verarbeitung der Events möglich sein. Dazu verwendet der CDI-Container ein oder mehrere zusätzliche Threads, sodass die Verarbeitung im Event-Producer direkt fortgesetzt werden kann. Wie die asynchrone Verarbeitung im Detail erfolgen soll, wurde lange innerhalb der Expert Group diskutiert und ist zum aktuellen Zeitpunkt noch nicht in allen Details abschließend entschieden worden. Vor allem die Bereitstellung eines intuitiven und einfachen API bei gleichzeitiger Wahrung der Kompatibilität zu CDI 1.1 und die Zugehörigkeit zu den jeweiligen Kontexten stellen hier eine Herausforderung dar.

Im aktuell diskutierten Vorschlag müssen zwei Bedingungen erfüllt sein, damit ein Event asynchron verarbeitet werden kann. Zum einen muss das Event über die zum Event-Interface hinzugefügte „fireAsync"-Methode ausgelöst werden und zum an-

```
void processEvent(@Observes @Priority(100) MyEvent event) {
// do something
}
```

Listing 1

deren der Event-Observer mit der „@ObservesAsync"-Annotation annotiert sein. Diese doppelte Aktivierung der asynchronen Verarbeitung stellt sicher, dass CDI-1.1-Event-Observer durch die Änderung eines Event-Producers nicht plötzlich asynchron ausgeführt werden.

Ebenso können bestehende Event-Producer, bei deren Erstellung eine synchrone Verarbeitung der Events angenommen wurde, keine asynchrone Verarbeitung der Events auslösen. Dies ist wichtig, da das Event-Objekt veränderbar sein kann und in vielen Anwendungen für den Datenaustausch zwischen Event-Producer und -Observer verwendet wird.

Wenn die Verarbeitung des Events in Zukunft asynchron erfolgen soll, muss der Entwickler sich um Nebenläufigkeitsprobleme kümmern und zusätzlich den Event-Producer gegebenenfalls auf die Event-Observer warten lassen. Somit ist die bestehende Code-Basis genau zu analysieren und bei Bedarf anzupassen, bevor eine asynchrone Eventverarbeitung möglich ist. *Listing 2* zeigt, wie asynchrone Events genutzt werden können.

Unterteilung in einen Java-EE- und einen Java-SE-Teil

Die Modularisierung der CDI-Spezifikation bietet im Gegensatz zu den vorher betrachteten Änderungen des Event-Systems nur wenige Vorteile für die Anwendungsentwicklung. Stattdessen unterstützt es die Entwickler anderer Spezifikationen und Frameworks bei der gezielten Verwendung einzelner Teile der Spezifikation.

Der anfängliche Plan war, die CDI-2.0-Spezifikation in drei Teile zu unterteilen: „CDI full mit Java EE", „CDI full" und „CDI light". „CDI full mit Java EE" entspricht dabei dem vollen Funktionsumfang, wie er aus CDI 1.1 bekannt ist, einschließlich der Abhängigkeiten zu anderen Java-EE-Spezifikationen sowie aller im Rahmen von CDI 2.0 neu hinzugefügten Features. „CDI full" und „CDI light" sollen hingegen auch im Java-SE-Umfeld einsetzbar sein und daher keine Abhängigkeiten auf Java-EE-Spezifikationen aufweisen. „CDI light" soll zusätzlich keine aspektorientierte Programmierung (AOP) und keine Kontexte enthalten, womit es unter Umständen sogar Java-ME-kompatibel werden könnte.

Für das EDR-Release ist allerdings nur ein Teil dieser Änderungen umgesetzt. Vorerst wurde die Spezifikation ausschließlich in „CDI full mit Java EE" und „CDI full" unterteilt. Ob die noch fehlende Definition von „CDI light" zu einem späteren Zeitpunkt erfolgen und welchen Umfang diese dann genau haben wird, wird nach Fertigstellung des EDR-Release diskutiert.

Bootstrapping für Java-SE-Umgebungen

In Java-SE-Umgebungen muss das Starten des CDI-Containers durch den Anwendungsentwickler erfolgen. Dies war bisher nicht Bestandteil der Spezifikation, wurde allerdings durch die Implementierungen „Weld" und „Open Web Beans" mithilfe eigener APIs ermöglicht, für die das Apache-Delta-Spike-Projekt eine containerunabhängige Abstraktion anbietet. Ab CDI 2.0 wird der Bootstrapping-Mechanismus durch die Spezifikation definiert *(siehe Listing 3)*.

Zuerst wird durch die statische „getCDIProvider()"-Methode der Utilityklasse „CDI" ein „CDIProvider" geholt. Im Standardfall wird dazu intern ein „ServiceLoader" verwendet. Anschließend startet die Methode „initialize()" den CDI-Container, der dann für die weitere Verwendung zur Verfügung steht. Wird der Container nicht mehr benötigt, kann ihn die „shutdown()"-Methode wieder beenden. Da die CDI-Klasse das „Autocloseable"-Interface implementiert, kann das Beenden des Containers auch mithilfe eines „try with Resource"-Blocks erfolgen.

Auch wenn bereits ein Vorschlag für das neue API ausgearbeitet wurde, sind noch nicht alle Details zur Verwendung von CDI im Java-SE-Umfeld abschließend diskutiert. Ein noch offenes Problem stellt zum Beispiel die Unterstützung impliziter Bean-Archive (ohne „beans.xml"-Datei) dar. Im Java-EE-Umfeld werden dazu per Default alle vorhandenen Archive gescannt. Im Java-SE-Umfeld kann das Durchsuchen aller Archive jedoch zu aufwändig sein. Die aktuell bevorzugte Lösung sieht daher vor, dass die Unterstützung für implizite Bean-Archive durch einen Konfigurations-Parameter aktiviert sein muss. Somit kann der Entwickler entscheiden, ob diese Funktionalität benötigt wird und ob die benötigte Zeit zum Scannen aller Archive aufgewendet werden soll.

Weitere noch zu diskutierende Fragestellungen betreffen unter anderem die Möglichkeit, mehrere Container in einer Anwendung zu starten, sowie die Unterstützung von Kontexten. Hierzu werden über das EDR-Release hinaus noch Lösungen erarbeitet werden müssen.

Weitere Änderungen

Neben den in das EDR eingeflossenen Änderungen wurden in den bisherigen Releases der CDI-Referenz-Implementierung „Weld"

```
// Producer
@Inject
Event<MyEvent> event;
[…]
MyEvent myEvt = new MyEvent();
CompletionStage<MyEvent> completion = event.fireAsynch(myEvt);

// Observer
public void consumeEvent(@ObservesAsynch MyEvent evt){
  // handle the event
}
```

Listing 2

```
// start container
CDIProvider cdiProvider = CDI.getCDIProvider();
CDI<Object> cdi = cdiProvider.initialize();

// get a bean and do something
MyCdiBean bean = cdi.select(MyCdiBean.class).get();
bean.doSomething();

// shutdown container
cdi.shutdown();
```

Listing 3

(siehe „http://weld.cdi-spec.org“) bereits einige weitere Änderungen prototypisch implementiert, die zwar diskutiert, aber noch nicht spezifiziert sind. Ob diese Änderungen in derselben Form in die finale Spezifikation aufgenommen werden, ist zum aktuellen Zeitpunkt nicht abzuschätzen. Allerdings vermitteln die prototypischen Implementierungen bereits einen guten Eindruck über die noch zu erwartenden Erweiterungen der Spezifikation.

So wird für Qualifier und Interceptoren das mit Java 8 eingeführte „repeating annotation“-Feature verwendet, mit dem Methoden und Eigenschaften mehrfach mit derselben Annotation annotiert werden können. Damit besteht beispielsweise die Möglichkeit, eine Producer-Methode mehrfach mit demselben Qualifier und unterschiedlichen Eigenschaften zu annotieren *(siehe Listing 4)*.

Darüber hinaus wurde das „ProcessObserverMethod“-SPI dahingehend erweitert, dass CDI-Extensions nun einzelne Observer-Methoden überschreiben oder sie mithilfe der „veto()“-Methode des Event-Interface deaktivieren können. Durch diese Änderungen erhält der Entwickler, wie von der Community häufig gewünscht, mehr Kontrolle über die Verarbeitung von Events.

```
@Produces
@PaymentMethod("CreditCard")
@PaymentMethod("Paypal")
public PaymentProvider createPaymentProvider() {
     // ...
}
```

Listing 4

Was noch zu erwarten ist

Wie bereits zu Beginn des Artikels gesagt, enthält das EDR nur die Änderungen der CDI-Spezifikation, für die die Expert Group bereits konkrete Vorschläge erarbeitet hat und die über eine ausreichende Reife verfügen, sodass sie von der Community ausprobiert und diskutiert werden können. Daher gibt es auch noch viele Themen, für die eine Lösung erst noch erarbeitet werden muss. Einige davon wurden bereits in den vorhergehenden Abschnitten genannt und einige weitere werden wir nun kurz betrachten. Die Diskussionen zu diesen Änderungen befinden sich in der Regel noch in einem sehr frühen Stadium oder wurden noch gar nicht aufgenommen. Daher ist zum aktuellen Zeitpunkt noch nicht abzusehen, welche dieser Änderungen Bestandteil des finalen Release werden und wie diese genau gestaltet werden.

Wie alle anderen Java-EE-8-Spezifikationen auch soll CDI 2.0 die Verwendung der mit Java 8 neu eingeführten Sprach-Features unterstützen. Bisher wurden noch keine der daraus resultierenden Änderungen innerhalb der Expert Group diskutiert und nur kleinere Änderungen in den Alpha-Releases der Referenz-Implementierung „Weld“ umgesetzt. Der vollständige Umfang der noch zu erwartenden Änderungen ist zurzeit noch nicht abzusehen. Alle neu hinzugefügten Features werden die neuen Java-8-Features verwenden.

Ein weiteres Thema, das bisher nur wenig bearbeitet wurde, ist die Anpassung des SPI. Dadurch soll unter anderem eine bessere Kontrolle der verwendeten Kontexte, die Erzeugung von Beans zur Laufzeit und der Zugriff auf die von CDI erzeugten Metadaten ermöglicht werden.

Die Anpassungen der Interceptoren und Dekoratoren wurden bisher noch nicht spezifiziert. Allerdings wird für die Dekoratoren in der Referenz-Implementierung „Weld“, wie bereits gesagt, das mit Java 8 eingeführte „repeating annotation“-Feature verwendet. Andere geplante Änderungen, wie Verbesserungen des „interceptor chaining“ und der Aufruf von Interceptoren bei internen Methoden-Aufrufen, wurden bisher noch nicht eingehend diskutiert.

Auch die Möglichkeit des Zugriffs auf das durch einen Proxy „gewrappte“ Objekt würde Entwicklern von Extensions das Leben in vielen Fällen vereinfachen. In eine ähnliche Richtung zielt auch der Vorschlag, alle (nicht nur die CDI-relevanten) Annotationen einer „gewrappten“ Klasse auch in der Proxy-Instanz verfügbar zu machen. Interessant ist auch die Idee, bestehende Interceptoren mit „@Alternative“ übersteuern sowie die Reichweite von Events mittels einer Annotation bestimmen zu können. Damit kann aktiv beeinflusst werden, ob ein Event an alle aktuell verfügbaren Observer weitergereicht wird oder zum Beispiel nur an die, die in der gleichen Deployment-Unit („=jar“) enthalten sind.

In welchem Umfang all diese Vorschläge in der finalen Spezifikation enthalten sein werden, ist aber derzeit noch nicht absehbar: stay tuned!

Anatole Tresch
atsticks@gmail.com

Anatole Tresch war nach dem Wirtschaftsinformatik-Studium an der Universität Zürich mehrere Jahre lang als Managing-Partner und -Berater tätig. Er sammelte weitreichende Erfahrungen in allen Bereichen des Java-Ökosystems vom Kleinunternehmen bis zu komplexen Enterprise-Systemen. Schwerpunkte sind verteilte Systeme sowie die effiziente Entwicklung und der Betrieb von Java in der Cloud. Aktuell arbeitet Anatole Tresch als technischer Architekt bei der Credit-Suisse, ab September wird er als Principal Consultant bei Trivadis tätig sein. Anatole ist Star Specification Lead des JSR 354 (Java Money & Currency) und Gründer des Apache-Tamaya-Projekts.

Thorben Jannsen
thjanssen123@gmail.com

Thorben Janssen arbeitet als Senior-Entwickler und Architekt für die Qualitype GmbH und entwickelt seit mehr als zehn Jahren Anwendungen auf Basis von Java EE. Er ist Mitglied der JSR 365 Expert Group und bloggt über Themen rund um Java Enterprise auf *„http://www.thoughts-on-java.org“*.