Ausgabe 01/2019

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 01

Java aktuell

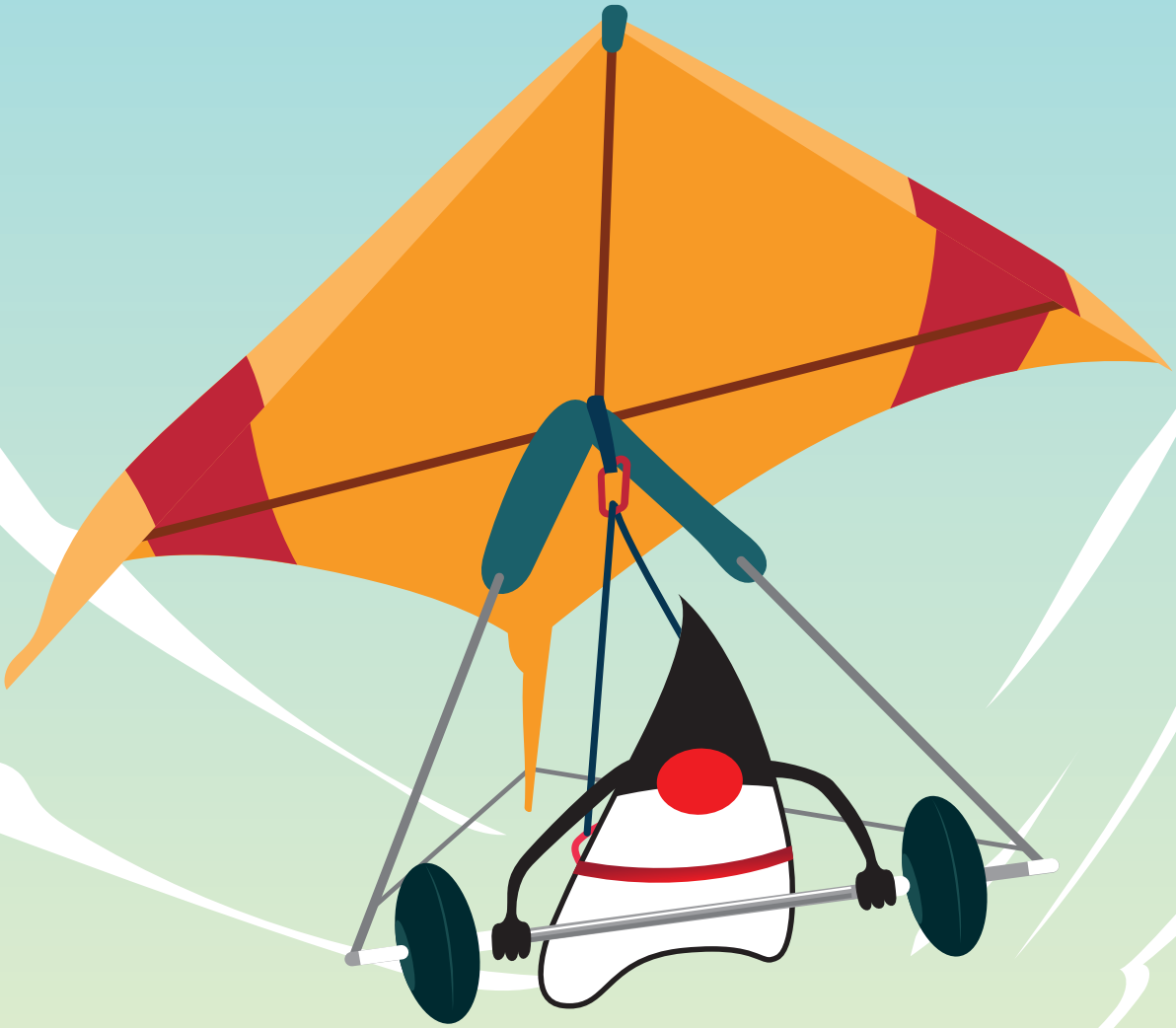
iJUG Verbund
www.ijug.eu

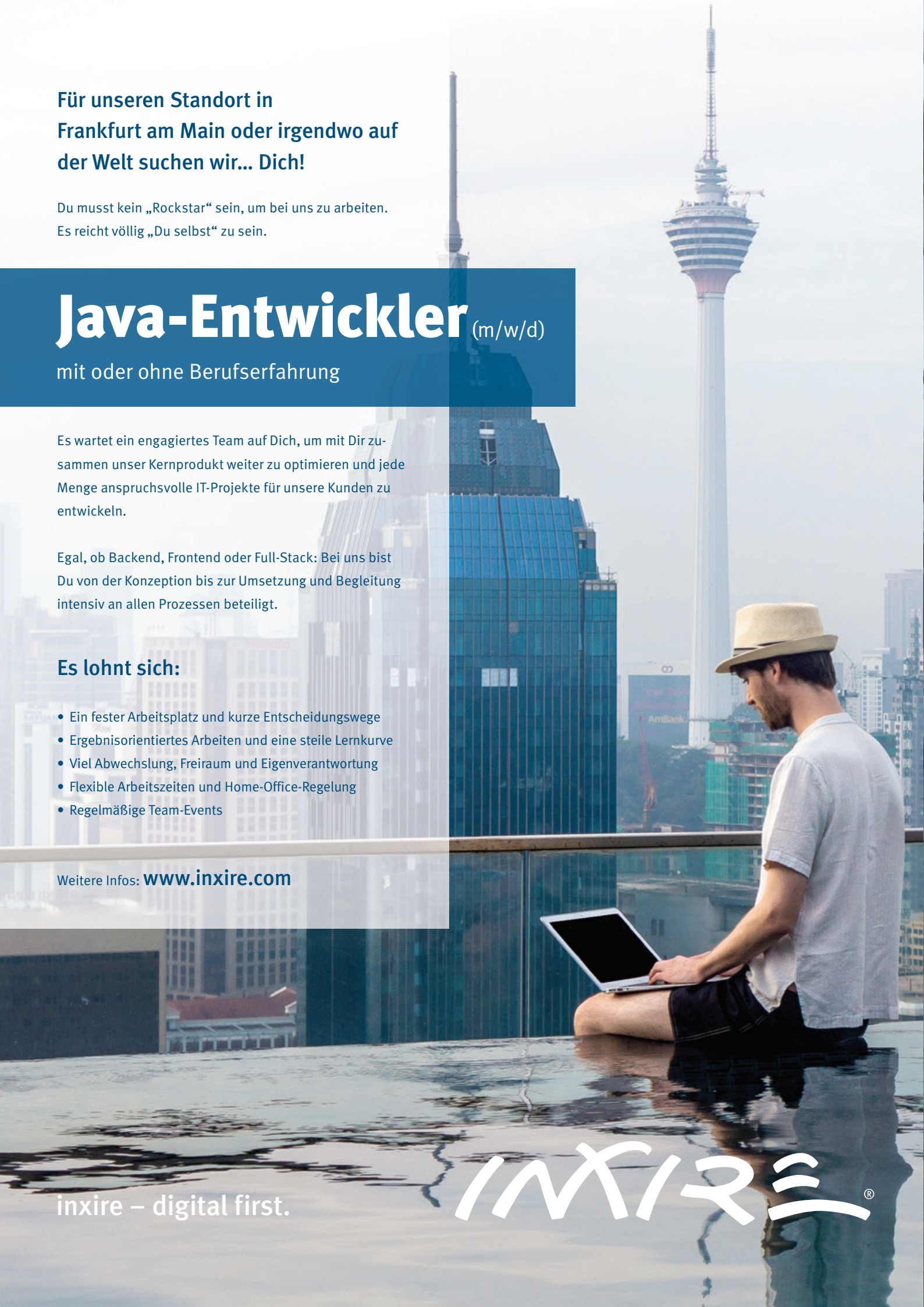
Next Level Java
Erste Eindrücke von der Version 11

Eclipse MicroProfile
Neue Features und Erweiterungen

Blockchain
Implementierung in der IDE seiner Wahl

Java im Aufwind





Für unseren Standort in Frankfurt am Main oder irgendwo auf der Welt suchen wir... Dich!

Du musst kein „Rockstar“ sein, um bei uns zu arbeiten.
Es reicht völlig „Du selbst“ zu sein.

Java-Entwickler (m/w/d)

mit oder ohne Berufserfahrung

Es wartet ein engagiertes Team auf Dich, um mit Dir zusammen unser Kernprodukt weiter zu optimieren und jede Menge anspruchsvolle IT-Projekte für unsere Kunden zu entwickeln.

Egal, ob Backend, Frontend oder Full-Stack: Bei uns bist Du von der Konzeption bis zur Umsetzung und Begleitung intensiv an allen Prozessen beteiligt.

Es lohnt sich:

- Ein fester Arbeitsplatz und kurze Entscheidungswege
- Ergebnisorientiertes Arbeiten und eine steile Lernkurve
- Viel Abwechslung, Freiraum und Eigenverantwortung
- Flexible Arbeitszeiten und Home-Office-Regelung
- Regelmäßige Team-Events

Weitere Infos: **www.inxire.com**

inxire – digital first.

INXIRE®

Eclipse MicroProfile

Thilo Frotscher, Freiberufler

Die bislang unter „Java EE" bekannte Plattform ist mitten im Umbruch. Nach der Entscheidung von Oracle, die alleinige Kontrolle aufzugeben und die Plattform an die Eclipse Foundation zu übertragen, warten Entwickler nun gespannt auf ein erstes Release unter dem neuen Namen „Jakarta EE". Während der Umfang dieser initialen Version identisch mit Java EE 8 sein soll, arbeitet das Eclipse-MicroProfile-Projekt in rasantem Tempo an potenziellen neuen Features, die schon bald in die Plattform einfließen könnten.

Das Jahr 2016 haben Java-EE-Entwickler in keiner guten Erinnerung. Die Arbeiten an Java EE 8 waren praktisch zum Stillstand gekommen und es gab berechtigte Befürchtungen, dass Oracle das Interesse an der Plattform verloren habe und die Weiterentwicklung einstellen könnte. Es wäre das Ende der beliebten und weitverbreiteten Plattform gewesen und hätte zahllose Unternehmen und Entwickler weltweit in erhebliche Schwierigkeiten gebracht. Glücklicherweise ist dies nicht eingetreten. Im Gegenteil, die Übertragung von Java EE an die Eclipse Foundation wird gemeinhin positiv bewertet und als große Chance aufgefasst.

Als Jakarta EE wird die Plattform nun als Open Source weiterentwickelt. Die Entscheidungsprozesse sind offener, eine Beteiligung der Community ist explizit erwünscht und sogar die Technology Compatibility Kits (TCK) sind inzwischen als Open Source verfügbar. Insbesondere letzterer Schritt gilt als bahnbrechender Meilenstein in der Historie der Plattform.

Einziger Wermutstropfen: Während des (noch laufenden) Übertragungsprozesses zur Eclipse Foundation, der neben allerlei technischen vor allem auch viele juristische Herausforderungen mit sich bringt, ist die Weiterentwicklung der Plattform erneut zum Stillstand gekommen. Faktisch ist seit einigen Jahren technologisch nur wenig Fortschritt zu erkennen, wodurch das erste Release von Jakarta EE einen recht angestaubten Eindruck machen wird.

Zwar soll dieses Initial-Release zunächst im Wesentlichen nur den Nachweis erbringen, dass die Plattform auf der Infrastruktur der Eclipse Foundation gebaut werden kann und dass die Kompatibilität zu Java EE 8 erhalten bleibt. Dennoch wird es im Anschluss ganz entscheidend darauf ankommen, ob zeitnah ein weiteres Release veröffentlicht werden kann, das notwendige Erweiterungen und Modernisierungen enthält.

Die MicroProfile-Initiative

Tatsächlich gibt es bereits eine ganze Reihe potenzieller Neuerungen, die sich zur Integration in Jakarta EE anbieten, denn aus der eingangs geschilderten Ungewissheit über die Zukunft von Java EE ist im Herbst 2016 die sogenannte „MicroProfile-Initiative" entstanden. Dabei handelte es sich um den Zusammenschluss mehrerer Hersteller von Application-Servern, die nicht länger passiv bleiben und die weitere Entwicklung bei Oracle abwarten wollten – letztlich ist das Geschäftsmodell dieser Hersteller untrennbar mit dem Fortbestehen der Java-EE-Plattform verbunden. Ziel der Initiative war daher die Erarbeitung von Vorschlägen für neue Features und Erweiterungen, die die Plattform modernisieren und insbesondere für die Entwicklung von Microservices attraktiver machen sollten. Darauf aufbauend soll ein neues Java EE Profile speziell für diesen Einsatzzweck vorgeschlagen und standardisiert werden. Durch die Kombination der Begriffe „Microservice" und „Java EE Profile" kam somit der Name „MicroProfile" zustande.

In einem ersten Schritt wurden jene bestehenden Java-EE-Technologien identifiziert, die als absolutes Minimum oder kleinster gemeinsamer Nenner für die Entwicklung von Microservices anzusehen sind. Somit bestand das MicroProfile 1.0 lediglich aus JAX-RS, JAX-P und CDI. Da diese Technologien ohnehin bereits in den jeweiligen Application-Servern für Java EE 7 enthalten waren, konnten die Hersteller im Anschluss recht schnell spezielle MicroProfile-Editionen ihrer Server bereitstellen. Dabei handelte es sich schlicht um schlankere Versionen der bekannten Produkte.

Dennoch war ein wichtiger, zukunftsweisender Schritt getan: Die Abkehr vom monolithischen Application-Server wurde spätestens zu diesem Zeitpunkt eingeläutet. Die Zukunft gehört dagegen schlanken und modularen Servern, die nur genau jene Technologien enthalten, die für eine jeweilige Anwendung zur Laufzeit auch wirklich benötigt werden. In diesem Zusammenhang entstanden Lösungen, bei denen Anwendungen und die von ihnen benötigten Application-Server-Module während des Build-Prozesses gemeinsam in ein einziges ausführbares JAR gepackt werden, und somit ein Deployment-Modell, das sich vor allem im Kontext von Docker und Cloud erheblich besser einfindet.

Im Anschluss an dieses erste MicroProfile-Release wurde zu-

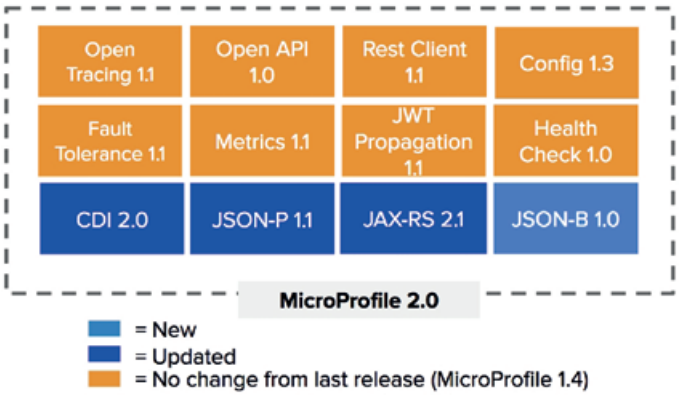


Abbildung 1: Die APIs des MicroProfile 2.0

nächst eine Reihe von Anforderungen gesammelt, die ein modernes Framework erfüllen sollte, um eine gute Unterstützung für die Entwicklung von „Cloud Native"-Anwendungen zu gewährleisten. Unter anderem wurden eine Unterstützung für den Zugriff auf Konfigurations-Parameter aus unterschiedlichen Quellen, Fehlertoleranz und Widerstandsfähigkeit, Monitoring und Sicherheit als besonders dringlich erachtet. In der Folge entstanden Vorschläge für entsprechende APIs, die von den Mitgliedern der MicroProfile-Initiative in (für Java-EE-Verhältnisse) bislang ungekannter Geschwindigkeit implementiert und zum Download bereitgestellt wurden.

In weniger als zwölf Monaten erschienen so die MicroProfile-Versionen 1.1 bis 1.4. Das aktuellste Release im Sommer 2018 ist MicroProfile 2.0, das inzwischen aus insgesamt zwölf APIs besteht. Dazu zählen weiterhin die aus Java EE übernommenen JAX-RS, JAX-P und CDI. Sie wurden auf die jeweils neuesten Releases aus Java EE 8 aktualisiert. Hinzu kam JSON-B 1.0 für das Binding zwischen JSON und fachlichen Java-Klassen. Die verbleibenden acht neuen APIs werden im weiteren Verlauf kurz vorstellt *(siehe Abbildung 1)*.

Konfiguration

Für eine weitverbreitete Plattform wie Java EE, die seit vielen Jahren stetig weiterentwickelt wird, ist es erstaunlich, dass so ein zentraler Aspekt wie das Einlesen von Konfigurations-Informationen bislang nicht durch ein standardisiertes API unterstützt wird. Aus diesem Grund werden häufig Bibliotheken wie Apache Commons Configuration oder Apache Delta Spike eingesetzt. Gerade beim Betrieb von Anwendungen in Containern oder in der Cloud ist es jedoch üblich, unterschiedliche Konfigurationsquellen wie Umgebungsvariablen, System-Properties und Dateien mit Default-Konfigurationen einzusetzen. So kam das Thema gleich zu Beginn der MicroProfile-Initiative auf die Tagesordnung: Das Config-API war das erste neu entwickelte API.

Mit dem Config-API ist es nun möglich, Konfigurationswerte mittels CDI in den Anwendungscode injizieren zu lassen. Dabei spielt es keine Rolle, über welche der unterschiedlichen Konfigurationsquellen der Wert letztlich bezogen werden kann. Das Auffinden des Wertes geschieht durch das API und ist daher transparent für den Anwendungsentwickler.

Hinter den Kulissen werden die unterschiedlichen Konfigurationsquellen in festgelegter Reihenfolge untersucht. Natürlich ist es auch möglich, Default-Werte zu definieren, die zum Einsatz kommen, falls ein Konfigurationswert in keiner der Quellen gefunden werden kann. Alternativ können nicht vorhandene Konfigurationswerte in einem „Optional<T>" gekapselt sein *(siehe Listing 1)*. Anstelle des Einsatzes von CDI kann über einen ConfigProvider auch programmatisch auf die Konfiguration zugegriffen werden *(siehe Listing 2)*.

Sollten die vom Config-API unterstützten Konfigurationsquellen nicht ausreichen, lassen sich eigene Konfigurationsprovider mit einem Plug-in-Mechanismus sehr leicht implementieren. Diese werden zur Laufzeit mithilfe des Service-Loader-Mechanismus erkannt und automatisch in Betrieb genommen.

```
@Inject
private Config config;

@Inject
@ConfigProperty(name="service.url")
private String serviceUrl;

@Inject
@ConfigProperty(name="service.username", defaultValue="Alice")
String username;

@ConfigProperty(name="service.password")
private Optional<String> password;
```

Listing 1

```
Config config = ConfigProvider.getConfig();
String serverUrl =
  config.getValue("service.url", String.class);
Optional<String> username =
  config.getOptionalValue("service.username ", String.class);
String[] codes =
  config.getValue("countryCodes", String[].class);
```

Listing 2

```
@CircuitBreaker
@Fallback(QuoteFallbackHandler.class)
public Quote getQuoteFromSupplier() { … }

@Retry(maxRetries = 3, retryOn = {MyRuntimeException.class})
public Person findPersonWithId(Long personId) { … }

@Timeout(500)
@Fallback(fallbackMethod = "getQuotesFallback")
public Response getQuotes() throws InterruptedException { … }
```

Listing 3

Fault Tolerance

Das Fault-Tolerance-API soll es Entwicklern ermöglichen, auf einfache und standardisierte Weise die Widerstandsfähigkeit ihrer Anwendungen zu verbessern. Eine auf HTTP basierende Kommunikation erfolgt offensichtlich über das Netzwerk – viele kommunizierende Services bilden ein verteiltes System, und da kann bekanntermaßen einiges schiefgehen. Es werden also Strategien benötigt, wie mit Verbindungsproblemen, Timeouts und Fehlerfällen umgegangen werden soll, ob gescheiterte API-Aufrufe etwa zu wiederholen sind und falls ja, wie häufig. Zu diesem Zweck definiert das Fault-Tolerance-API unterschiedliche Annotationen und Interfaces, mit deren Hilfe entsprechende Strategien im Code hinterlegt werden können.

Unterstützt werden die Patterns „Circuit Breaker" und „Bulkhead" sowie Strategien für Timeouts, Retries und Fallbacks. Zudem kann mit einer einzigen Annotation erreicht werden, dass Methoden in einem separaten Thread ausgeführt werden. Als Inspiration für das Fault Tolerance API dienten die bekannten Frameworks Hystrix und Failsafe *(siehe Listing 3)*.

JWT Propagation und Rest Client

Im Bereich der HTTP-APIs und somit auch der Microservices haben sich JSON Web Tokens (JWT) als Mittel zur Authentifizierung etabliert. Das JWT-Propagation-API des MicroProfile schlägt daher eine standardisierte Unterstützung von Open ID Connect auf Basis von JWT vor, mit der sich eine rollenbasierte Zugangskontrolle für Microservices umsetzen lässt. Konkret geschieht dies durch die Einführung der Interfaces „JsonWebToken" und „Claim" zur Repräsentation von Token und der darin enthaltenen Informationen. Zusätzlich wird die Annotation „LoginConfig" vorgeschlagen, die in JAX-RS-Ressourcen-Klassen eingesetzt werden soll, um anzuzeigen, welche Form der Authentifizierung für den Zugriff auf die jeweilige Ressource erforderlich ist. Eine dieser möglichen Authentifizierungsformen wäre dann „MP-JWT", also die vom MicroProfile vorgeschlagene Authentifizierung mittels JSON Web Token.

Das MicroProfile-Rest-Client-API zielt darauf, die Implementierung von JAX-RS-Clients zu vereinfachen. Bisher müssen Anwendungsentwickler recht viel technischen Code schreiben, um mithilfe des JAX-RS-Client-API einen Request zu versenden. Insbesondere müssen sie dafür Sorge tragen, dass zu sendende oder zu empfangene Daten korrekt zwischen eigenen fachlichen Klassen und dem für die Kommunikation verwendeten Format – typischerweise JSON – umgewandelt werden.

Mit dem Rest-Client-API soll nun mehr Typsicherheit erzielt und der Anteil technischen Codes reduziert werden. Hierzu würden Entwickler zunächst ein Java-Interface erstellen, um die Schnittstelle des API-Providers auf Client-Seite zu repräsentieren. Die Parameter und Rückgabewerte der einzelnen Interface-Methoden definieren dabei die fachlichen Klassen für die Datenkonvertierung. Weiterhin wird das Interface mit den (bislang nur serverseitig verwendeten) Annotationen wie „@Path", „@GET", „@POST" etc. versehen, um URL-Pfade und Request-Typen zu definieren *(siehe Listing 4)*.

Zur Laufzeit lassen sich dann auf Client-Seite Proxy-Objekte erzeugen, die das zuvor erstellte Interface implementieren, den technischen JAX-RS-Code aber verbergen und somit den Aufruf der API-Operationen durch fachlich anmutende Methoden ermöglichen. Die Proxy-Objekte können entweder mit CDI injiziert oder mithilfe des ebenfalls vorgeschlagenen „RestClientBuilder" programmatisch erzeugt werden. Zudem lassen sich clientseitige „ResponseExceptionMapper" registrieren, die bestimmte Status-Codes empfangener HTTP-Responses in fachliche Exceptions umwandeln.

OpenAPI

Mit dem Trend zu HTTP-APIs entstanden sehr bald auch unterschiedliche Beschreibungssprachen für die API-Spezifikation. Die populärsten sind sicherlich Swagger, RAML und API-Blueprint. In diesem Kontext formierte sich mit der OpenAPI-Initiative ein Verbund namhafter Hersteller, die sich zum Ziel setzten, eine Beschreibungssprache unter dem Namen „OpenAPI" zu standardisieren. Als Startpunkt für den neuen Standard wurde Swagger 2.0 gewählt und darauf aufbauend inzwischen OpenAPI 3.0 veröffentlicht.

Eine API-Spezifikation kann grundsätzlich auf zwei unterschiedlichen Wegen entstehen. Zum einen kann sie manuell erstellt werden – typischerweise unter Zuhilfenahme eines speziellen Editors. Ein alternativer Weg besteht darin, mit der Implementierung des API-Providers zu beginnen und die API-Spezifikation anschließend aus der Implementierung abzuleiten. In diesem Fall werden Werkzeuge eingesetzt, die den bestehenden Code analysieren, also beispielsweise im Falle des Einsatzes von JAX-RS die jeweiligen Annotationen auswerten, um eine OpenAPI-Datei zu generieren.

In der Regel sind jedoch die Informationen, die allein aus dem Code des API-Providers gewonnen werden können, für eine vollständige API-Spezifikation nicht ausreichend; es ist eine Reihe von Zusatz-Informationen anzureichern. Im Umfeld von Swagger entstand dabei für Java-Entwickler eine Sammlung proprietärer Annotationen, die in JAX-RS-Anwendungen zusätzlich eingebaut werden können, um solche Zusatz-Informationen zu hinterlegen.

Dieser Ansatz erscheint vielen Entwicklern nicht ideal, da die vielen zusätzlichen Annotationen den Blick auf das Wesentliche – den eigentlichen Anwendungscode – doch sehr versperren. Ungeachtet dessen hat MicroProfile OpenAPI zum Ziel, diesen Ansatz zu standardisieren, also Annotationen vorzuschlagen, die künftig zur Jakarta-EE-Plattform gehören könnten und die dazu dienen, API-Spezifikationen nach dem OpenAPI-Standard zu erzeugen.

Health Check, Metrics und Open Tracing

Nicht zuletzt sollen gleich drei unterschiedliche MicroProfile-APIs dazu dienen, Anwendungen im laufenden Betrieb zu überwachen. Das Health-Check-API ermöglicht es Entwicklern, einer Anwendung auf einfache Weise spezielle Endpunkte hinzuzufügen, mit denen ihre grundsätzliche Erreichbarkeit und der Status einzelner System-Komponenten geprüft werden kann. Hier ist insbesondere an den Einsatz in Cloud-Infrastrukturen gedacht, bei denen der Zustand einzelner Knoten automatisiert geprüft wird, um diese gegebenenfalls außer Betrieb zu nehmen oder neu zu starten.

Das Metrics-API richtet sich dagegen eher an den Wunsch, kontinuierlich bestimmte Messwerte wie etwa Antwortzeiten, die Anzahl bearbeiteter Requests oder die Auslastung von System-Ressourcen zu überwachen, um daraus bei Bedarf Statistiken zu erstellen oder Trends für künftige Lastspitzen abzulesen. Zu diesem Zweck werden spezielle Annotationen wie „@Timed", „@Metered" oder „@Counted" definiert, mit denen entsprechende Metriken im Code markieren werden können.

Mit dem Open-Tracing-API wird schließlich eine Unterstützung des Open-Tracing-Standards umgesetzt. Dabei handelt es sich um einen Mechanismus, mit dem sich Vorgänge oder Anwendungsfälle in einem verteilten System, wie etwa einer Microservices-Architektur, verfolgen lassen. Dabei können alle beteiligten Services sogenannte „Trace-Informationen" erzeugen, die anschließend gesammelt werden, um zu erkennen, wie sich die Verarbeitung eines Requests über unterschiedliche Knoten ausgebreitet hat. Open Tracing definiert hierzu herstellerneutrale APIs; die Trace-Informationen lassen sich von entsprechenden Tools verarbeiten. Das MicroProfile-Open-Tracing-API ermöglicht es, auf JAX-RS basierenden Anwendungen an solchen Tracing-Szenarien teilzunehmen.

Aktueller Stand

Mittlerweile ist auch die MicroProfile-Initiative der Eclipse Foundation beigetreten. Die Entwicklung der vorgestellten APIs erfolgt somit innerhalb des neu gegründeten „Eclipse MicroProfile"-Projekts. Dies ist ein sinnvoller Schritt, da sämtliche im MicroProfile entstandenen APIs potenzielle Kandidaten für die Aufnahme in künftige Releases von Jakarta EE sind. Eine Übergabe von Features ist sicherlich leichter zu bewerkstelligen, wenn beide Projekte unter dem Dach der Eclipse Foundation beheimatet sind.

Bislang ist noch unklar, in welcher Beziehung die beiden Eclipse-Projekte mittelfristig zueinander stehen sollen und ob etwa Jakar-

```
@Path("/movies")
public interface MovieReviewService {
  @GET
  Set<Movie> getAllMovies();

  @POST
  @Path("/{movieId}/reviews")
  String submitReview( @PathParam("movieId") String movieId, Review review );

  @GET
  @Path("/{movieId}/reviews")
  Set<Review> getAllReviews( @PathParam("movieId") String movieId );
 }
```

Listing 4

ta EE und MicroProfile zusammengeführt werden. Ein alternativer Vorschlag besteht darin, Eclipse MicroProfile als eigenständiges Projekt beizubehalten. Es könnte als „Incubator" dienen, in dem neue Technologien losgelöst von den organisatorischen Prozessen einer großen Plattform in pragmatischer Code-First-Mentalität entstehen. Sobald eine gangbare Lösung implementiert und ausreichend Feedback eingeholt wurde, sollen formale Spezifikation, Standardisierung und das Einfließen in Jakarta EE erst im letzten Schritt erfolgen.

Obwohl also noch nicht geklärt ist, ob und wann die einzelnen MicroProfile-APIs in die Jakarta-EE-Plattform einfließen, können sie dennoch schon heute eingesetzt werden. Hersteller wie Red Hat, IBM, Payara oder Tomitribe bieten Implementierungen des MicroProfile zum Download an. Entwickler, die erste Experimente mit den neuen APIs starten möchten, sollten daher einen Blick auf die aktuellen Releases von Thorntail, Open Liberty, Payara Micro oder Apache TomEE werfen.

Fazit

Während der Fokus im Jakarta-EE-Projekt aktuell noch darauf liegt, den Übergang zur Eclipse Foundation sowohl technisch als auch organisatorisch abzuschließen und ein erstes, zu Java EE 8 funktional identisches Release zu erstellen, entstehen im MicroProfile gleichzeitig eine ganze Reihe von APIs, die potenzielle Neuerungen für Jakarta EE darstellen.

Es bleibt nun abzuwarten, welche davon tatsächlich in die Plattform übernommen werden. Kritische Stimmen bemerken, dass auch im MicroProfile nur wenig Innovation stattfindet, sondern überwiegend Features kopiert werden, die andere Frameworks schon lange bieten. Dem ist entgegenzuhalten, dass Innovation auch gar nicht im Fokus der MicroProfile-Initiative stand. Stattdessen besteht das momentane Ziel darin, die Plattform derart zu erweitern, dass sie moderne Einsatzgebiete wie Microservices, Cloud und Container besser unterstützt. In diesem Kontext erscheint es sinnvoll, Konzepte zu übernehmen, die sich andernorts bewährt haben und Entwicklern daher in ähnlicher Form bekannt sind.

Es wird spannend sein zu beobachten, wie sich Jakarta EE und MicroProfile weiterentwickeln und ob das Momentum des Neuanfangs genutzt werden kann. In jedem Fall bietet der Übergang in die Open-Source-Welt eine große Chance für die Plattform. Es wird nun auf die Community ankommen, sich einzubringen und die lange geforderte Möglichkeit der Beteiligung und Einflussnahme auch zu nutzen.

Thilo Frotscher
feedback@frotscher.com

Thilo Frotscher arbeitet als freiberuflicher Entwickler und Trainer. Als Experte für Enterprise Java, APIs und System-Integration unterstützt er seine Kunden überwiegend durch Projektarbeit, Reviews oder die Durchführung von Schulungen. Thilo Frotscher ist (Co-) Autor mehrerer Bücher über Java EE, (Web) Services und System-Integration, hat zahlreiche Fachartikel verfasst und spricht regelmäßig auf Fachkonferenzen und Schulungsveranstaltungen oder bei Java User Groups.

JavaLand 2019: Größere Beteiligung als je zuvor

Dass die Java-Community absolut fantastisch ist, ist kein Geheimnis und wurde bereits mehrmals auf der JavaLand im Phantasialand unter Beweis gestellt. Doch in diesem Jahr hat sie sich mit mehr als 600 Vortragseinreichungen selbst übertroffen. Damit konnte die Veranstaltung wieder eine überwältigende Beteiligung und erneutes Wachstum verzeichnen. Sie findet vom 19. bis 21. März 2019 statt.

Für das Konferenzprogramm gab es insgesamt 637 Einreichungen, das sind ein Viertel mehr als im vergangenen Jahr. Auch das Newcomer-Programm für Referenten ohne Bühnenerfahrung fand reges Interesse mit 60 Einreichungen (zum Vergleich: 23 in 2018). Beim Schulungstag haben die Stream-Leiter dieses Mal die Qual der Wahl zwischen 37 Einreichungen von 23 Partnern. Das fertige Programm steht unter *„https://programm.javaland.eu/2019/#/schedule"* bereit.


ORAWORLD
Das e-Magazine für alle Oracle-Anwender!
EOUC
EMEA ORACLE USERGROUP COMMUNITY
• Spannende Geschichten aus der Oracle-Welt
• Technologische Hintergrundartikel
• Leben und Arbeiten heute und morgen
• Einblicke in andere User Groups weltweit
• Neues (und Altes) aus der Welt der Nerds
• Comics, Fun Facts und Infografiken
Jetzt Artikel einreichen oder Thema vorschlagen!
Jetzt e-Magazine herunterladen
www.oraworld.org

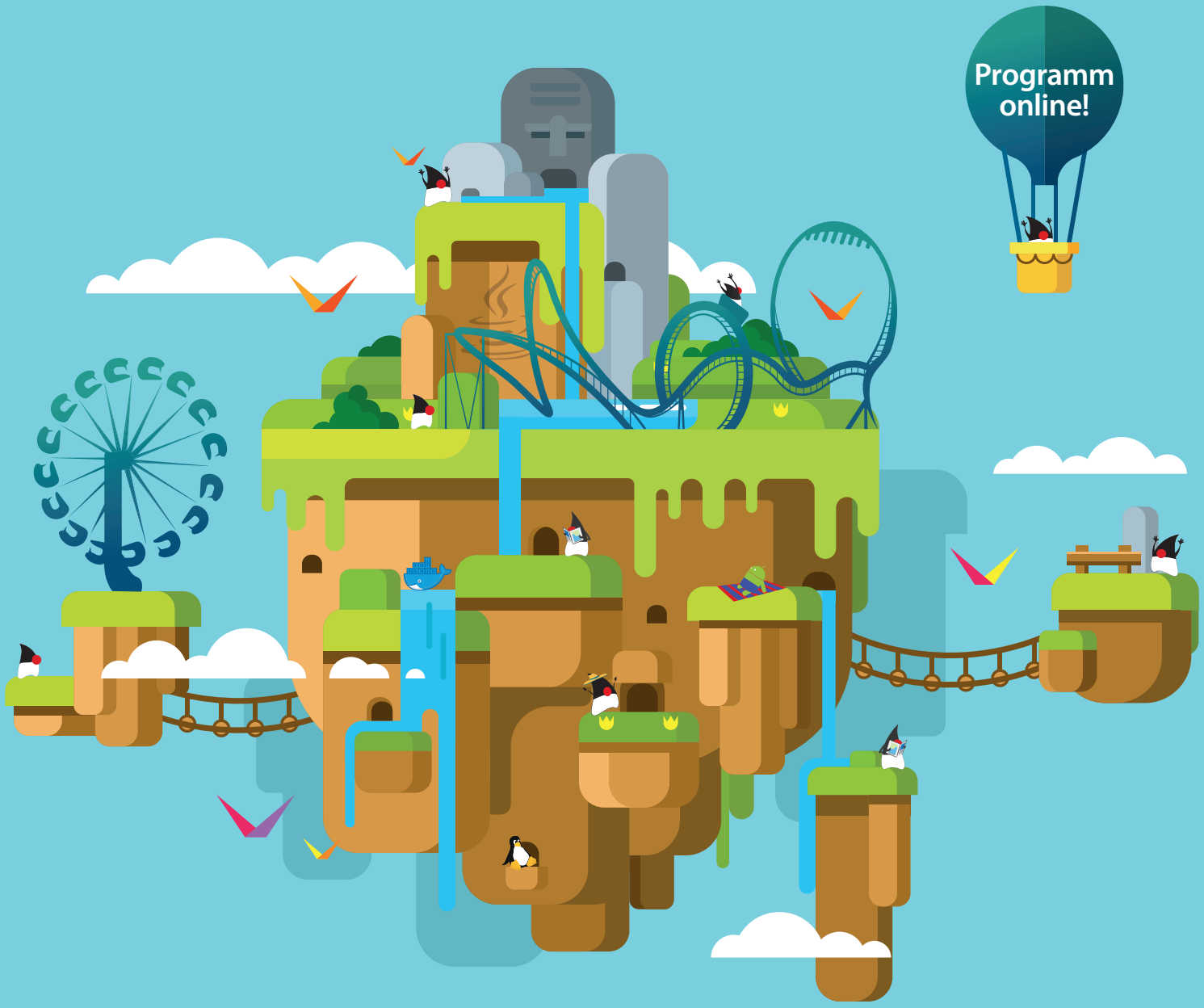
JavaLand

Early Bird
bis 15. Jan. 2019

19. - 21. März 2019 in Brühl bei Köln

Ab sofort Ticket & Hotel buchen!

www.javaland.eu



Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu