Im Maschinenraum

Java-Web-Frameworks von innen?

Phillip Ghadir, Michael Vitz

Web-Frameworks für Java gibt es nun bereits seit fast 20 Jahren. Diese nehmen dem Anwendungsentwickler eine Menge Arbeit ab und sorgen dafür, dass dieser sich nicht auf Infrastruktur konzentrieren muss, sondern die Anwendungslogik im Vordergrund steht. Doch was genau tut so ein Web-Framework eigentlich? Dieser Artikel zeigt, was heutige Web-Frameworks leisten und wo diese sich dann doch in Nuancen unterscheiden.

Obwohl [REST] heute als Integrationsarchitektur gesetzt ist, besitzen viele Java-Entwickler noch immer recht wenig Verständnis für HTTP. Nur aus der Ferne betrachtet ist HTTP ein sehr einfaches Protokoll: Der Client schickt einen Request an den Server, den dieser verarbeitet und mit einer Response beantwortet.

Ein Blick in die Request for Comments (s. Kasten „HTTP RFCs") macht deutlich, dass HTTP nicht ganz trivial ist. Abbildung 1 zeigt, welche Entscheidungen in der Request-Verarbeitung getroffen werden müssen, bevor die eigentliche Anwendungslogik aufgerufen werden darf.

Das richtige Web-Framework?

Häufig begegnet uns die Frage, welches Web-Framework für die Entwicklung von komplexen Systemen am Besten geeignet ist. Wie beantwortet man diese Frage, die so einfach klingt, aber so viele Facetten hat?

Wir drücken uns um eine Antwort, indem wir klären, welche Abstraktionen und Funktionen man sich von einem Web-Framework wünschen sollte. Aus dieser Perspektive heraus sollte jeder in der Lage sein, ein konkretes Framework auf dessen Eignung für die eigenen Zwecke auszuwählen.

Welche Abstraktionen braucht man?

Angenommen, wir müssten einen einfachen Internet-Service programmieren, der nur die denkbar einfachste Interaktion zulässt: Auf jede Anfrage antwortet er unmittelbar. Darüber hinaus gibt es keine weitere Interaktion. In diesem Fall genügt es, wenn unsere Funktionalität aufgerufen werden kann. Es gibt also keine speziellen Anforderungen an das Framework. Eigentlich könnte man alles HTTP-Spezifische ausblenden.

Aber wenn der Internet-Service komplexere Interaktionen unterstützen muss, und dafür gegebenenfalls mehrere Dialoge oder Zustände bietet, müssen die Dialogschritte oder Zustände identifiziert werden können.

HTTP nutzt zur Adressierung bekanntlich URIs. Ein Web-Server und gegebenenfalls auch das Web-Framework müssen zu diesen URIs die passende Implementierung finden können. Dies wird allgemein als *Routing* bezeichnet.

Das Routing liefert für einen HTTP-Request die aufzurufende Methode in der Laufzeitumgebung. Dazu betrachten wir zuerst einmal ein Beispiel für einen HTTP-Request:

```
GET /service/customers/1234 HTTP/1.1
Host: www.example.com
Accept: text/html
```

Grundsätzlich erlaubt HTTP die Adressierung beliebiger Dinge durch:

- die Angabe des HTTP-Verbs (Zeile 1 des Requests),
- die Adressierung der Ressource (Zeile 1),
- die Angabe des Hosts (Zeile 2),
- die Angabe des Formats (Content-Type des Request-Body),
- die Angabe der zulässigen Antwortformate (Zeile 3).

Darüber hinaus bestehen zwei weitere Möglichkeiten:

- beliebige eigene Header anzufügen, die dann möglicherweise nicht in jeder Infrastruktur funktionieren und daher stets nur optionale Informationen enthalten sollten, oder aber
- sogenannte Cookies zu vergeben: Diese dienen im Guten wie im Schlechten zur Identifizierung von Browsern, Sessions und Nutzern im World Wide Web.

Cookies

„Cookies" ist ein spezieller HTTP-Header, der (zum Teil größenbeschränkt auf 4 KB) selbst Schlüssel-Wert-Paare enthalten kann. Das Besondere an Cookies ist, dass sie für einen Host (mit einem Ablaufdatum) gesetzt werden können, und dann innerhalb des Gültigkeitszeitraums automatisch bei jedem Request vom Browser an den Server mitgeschickt werden.

Arten von Web-Frameworks

Wir versuchen die Web-Frameworks anhand der Merkmale zu unterscheiden, von denen sie abstrahieren, und nach ihren zentralen Konstruktionselementen zu benennen.

Zuallererst gibt es hier *Action-basierte Web-Frameworks*. Diese bilden den von HTTP geforderten Request-Response-Zyklus ab, indem der Anwendungsentwickler eine vom Framework vorgegebene Schnittstelle implementiert, die vom Framework aufgerufen wird, wenn ein passender Request ankommt. Als zusätzlichen Input bekommt man ein Objekt, welches den HTTP-Request abbildet. Das Ergebnis dieses Aufrufes ist eine HTTP-Response. Aktuelle Vertreter dieser Art sind [Grails], [JAX-RS], das [PlayFramework] und [SpringMVC].

HTTP RFCs

Lange Zeit (genau genommen seit 1999) war RFC2616 (`https://tools.ietf.org/html/`) der relevante Request for Comment für HTTP. Im Jahr 2007 startete jedoch die HTTP Working Group [httpbis] damit, diese Spezifikation zu überarbeiten und bisherige Unklarheiten zu beseitigen. Neue Features wurden dabei nicht eingeführt. 2014 war es schließlich so weit und die Working Group veröffentlichte die überarbeitete Fassung, aufgeteilt auf die folgenden sechs RFCs:

- RFC7230 - HTTP/1.1: Message Syntax and Routing
- RFC7231 - HTTP/1.1: Semantics and Content
- RFC7232 - HTTP/1.1: Conditional Requests
- RFC7233 - HTTP/1.1: Range Requests
- RFC7234 - HTTP/1.1: Caching
- RFC7235 - HTTP/1.1: Authentication

Als zweite Art haben sich *komponentenbasierte Web-Frameworks* etabliert. Diese sind von den klassischen Rich-Client-Programmiermodellen inspiriert und legen ihren Fokus darauf, dass eine Oberfläche aus mehreren Komponenten besteht und die Anwendung auf Änderungen dieser Komponenten reagiert. Der Request-Response-Zyklus wird dabei im Allgemeinen vor dem Anwendungsentwickler versteckt. Dieser implementiert lediglich Callbacks der Komponenten. Das Framework übernimmt die komplette Verarbeitung von Request und Response, ohne dem Entwickler Einflussmöglichkeiten zu geben.

Komponentenbasierte Frameworks halten oder rekonstruieren typischerweise den Dialogzustand für jeden Request-Response-Zyklus. Neben dem JavaEE-Standard [JSF] sind [ApacheWicket] und [Vaadin] populäre Vertreter dieser Art.

Neben diesen beiden Arten tritt in letzter Zeit eine dritte Art von Web-Frameworks in Erscheinung: *ressourcenbasierte Web-Frameworks*. Sie sind den Action-basierten Frameworks nicht unähnlich, allerdings unterscheiden sich die Programmiermodelle teilweise.

Eine definierte Ressource wird an eine URI gebunden. Wird diese URI aufgerufen, durchläuft das Framework strikt die in den HTTP RFCs beschriebene State-Maschine. Bei jeder Entscheidung wird auf der Ressource ein Callback aufgerufen, der den weiteren Verlauf des Requests beeinflusst. Somit entstehen Webanwendungen, die sich sehr genau an die HTTP-Spezifikation halten, und die Verarbeitung eines Requests wird auf mehrere Methoden aufgeteilt. Jede Methode muss hierbei allerdings nur einen einzelnen Aspekt der HTTP-Verarbeitung betrachten.

Leider gibt es auf der JVM für diese Art von Web-Frameworks noch nicht viele Implementierungen, das Konzept ist jedoch aufgrund der populären Erlang-Implementierung [webmachine] ausgereift. Auf der JVM kann man sich mit [webster] eine Java-basierte oder mit [liberator] eine Clojure-basierte Implementierung ansehen.

Routing – hin und zurück

Das Routing findet zur Laufzeit die richtige Implementierung für einen ankommenden Request. Je nach Anforderungen sind für das Routing beliebige Abstraktionen zulässig.

Action-basierte Web-Frameworks ziehen dafür zumindest mal die URI und in vielen Fällen auch das Verb und den Content-Type heran, während komponentenbasierte Web-Frameworks typischerweise vom Content-Type abstrahieren und den Fokus eher darauf legen, den Dialogzustand zwischen Client und Server zu synchronisieren.

Jedes Action-basierte Web-Framework besteht somit in seinem Kern aus einem Teil, der einen HTTP-Request entgegennimmt und anschließend den für diesen Request relevanten Code ausführt. Damit dies funktioniert, muss der Entwickler Routen definieren und diese dem Framework bekannt machen.

Routen registrieren

Die Registrierung von Routen und der Support der verschiedenen Routing-Merkmale unterscheidet sich je nach Framework. Servlets beispielsweise werden nur anhand der URI registriert – entweder per XML oder per Annotation. Gleiches gilt für SpringMVC. Bei SpringMVC lassen sich die Routen jedoch durch eine Kombination aus URI, HTTP-Methode, Accept- und Content-Type-Header registrieren:

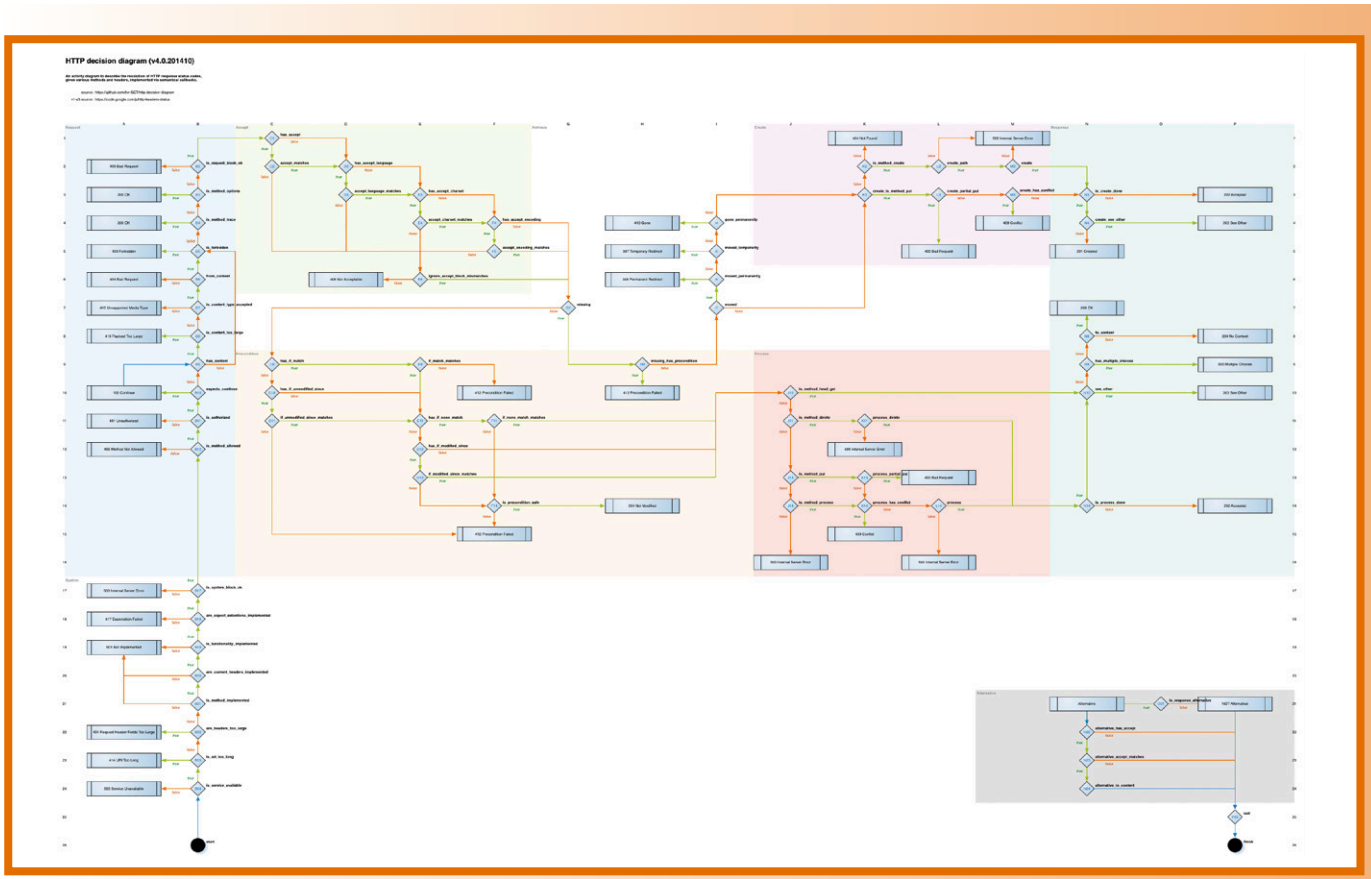

Abb. 1: HTTP-Decisions – unabhängig von der Fachlogik [httpdecisions]

```
...
@RequestMapping(method = RequestMethod.GET
                path="/clients/{id}")
public String show(@PathVariable Long id, Model model) {
  Client client = clientService.findById(id);
  model.addAttribute("client", client);
  return "showClient";
}
...
```

Das Play-Framework geht einen anderen Weg: Hier werden die Routen in einer DSL definiert, welche anschließend zu Scala-Code kompiliert wird:

```
# Verb Path         Action
GET  /clients/:id controllers.Clients.show(id: Long)
```

Die angegebene Action hierzu sieht folgendermaßen aus:

```
package controllers

import play.api._
import play.api.mvc._

object Clients extends Controller {
  def show(id: Long) = Action {
    Client.findById(id).map { client =>
      Ok(views.html.Clients.display(client))
    }.getOrElse(NotFound)
  }
}
```

Das Routing erfolgt auch hier per HTTP-Methode und URI.

Reverse-Routing

Wenn ein Service Antworten generieren soll, die per Link auf andere Serviceelemente oder Ressourcen verweisen, möchte man als Entwickler sich nicht selbst die URI zusammenbauen müssen, sondern die geeignete Handler-Funktion oder Klasse angeben und für diese automatisch einen Link vom Framework generieren lassen. Eine manuelle String-Konkatenation für den Zusammenbau von URIs würde nur zu großem Änderungsaufwand bei Anpassungen führen. Das Erzeugen eines Links für eine bestehende Implementierung nennt man *Reverse-Routing*.

Auch für diese Funktionalität gibt es unterschiedliche Ansätze und Unterstützung durch die aktuell verfügbaren Web-Frameworks. SpringMVC bietet mit dem MvcUriComponents-Builder die Möglichkeit zu sagen, welche Controllermethode man aufrufen möchte, und baut anschließend die passende URI hierfür:

```
UriComponents uriComponents = MvcUriComponentsBuilder
      .fromMethodCall(on(ClientsController.class).show(1)).buildAndExpand();

URI uri = uriComponents.encode().toUri();
```

Einen anderen Weg geht das Play-Framework, indem es aus den Routen spezielle Klassen generiert, die für das Reverse-Routing genutzt werden können:

```
...
def showFirstClient = Action {
  Redirect(routes.Clients.show(1))
}
...
```

Andere Frameworks bieten einem hierfür keinerlei Unterstützung und der Entwickler muss sich seine eigene Abstraktion hierfür schaffen.

Ist das Verlinken innerhalb des Systems gelöst, treten ganz andere Fragen in den Vordergrund.

Session-Management

HTTP ist zustandslos. Ein Client sendet einen Request und erhält eine Response. Das Protokoll selbst sieht keinen Aufbau eines Dialoggedächtnisses vor. Jeder Request soll alle Informationen enthalten, die für dessen Antwort erforderlich sind.

In HTTP ist nicht definiert, wie ein Server erkennen kann, ob zwei Requests zusammengehören oder unabhängig sind. Das trägt wesentlich dazu bei, dass das Gesamtsystem World Wide Web so skalierbar und damit auch verfügbar ist.

Für die Anwendungsentwicklung würde man sich – je nach Anwendung – aber schon eine zustandsbehaftete Konversation wünschen.

Das Konzept einer Benutzer-Sitzung lässt sich gut Framework-seitig unterstützen, da es üblicherweise unabhängig von der Fachlogik ist.

Ein Web-Framework, das die Grundideen des World Wide Web trägt, sollte sich nicht darauf verlassen, dass sich die mit dem Client ausgetauschten Formate dazu eignen, eine Session-ID mitzuführen.

Somit bleiben in HTTP nur noch zwei Stellen, die eine Session-ID enthalten können:

- in der URI oder
- als Header.

Die Übertragung im Body entfällt, wenn alle HTTP-Verben unterstützt werden sollen, da nicht jedes Verb einen Request-Body unterstützt.

Nutzt man die URI zur Übertragung einer Nutzeridentifikation, so geschieht dies in der Regel im „query"-Teil, da der Pfad zum Routing benötigt wird und man meistens auch nicht für jeden Nutzer eine eigene URI für dieselbe Ressource haben möchte. Dies ist möglich und wird auch teilweise genutzt (z. B. per jsessionid).

Die Session sollte nicht (ausschließlich) in der URI codiert werden. Wenn die URI ausreicht, auf den gesamten Zustand einer Session zuzugreifen, kann jeder, der in den Besitz dieser URI kommt (durch Versenden per Copy&Paste oder Mitschneiden des Netzwerkverkehrs), die Session übernehmen und von da an als legitim angemeldeter User handeln.

Daher empfiehlt sich für das Prüfen der Authentizität eines Requests eigentlich nur eines.

Session-Cookies

Somit bleibt nur noch die Option, das Merkmal per Header zu übertragen. Hierzu nutzt man Cookies. Der Server sendet hierbei in einer Response die Aufforderung an den Client, einen Wert in einem Cookie abzulegen. Bei jedem weiteren Request sendet der Client anschließend den Wert des Cookies als HTTP-Header mit. Hieraus entstehen folgende Anforderungen:

- Der Wert sollte möglichst klein sein, um nicht bei jedem Request unnötig viel Bandbreite zu verbrauchen.
- Der Wert sollte signiert sein, da dieser potenziell vom Client verändert werden könnte.
- Das ganze funktioniert nur mit HTTPS, da ansonsten der Wert des Cookies im Netzwerk mitgeschnitten werden könnte.

Ein gutes Web-Framework versteckt all diese Funktionalität vor dem Anwendungsentwickler. Dieser kann demnach so vie-

le Objekte wie er möchte in die Session packen. Das Framework speichert diese jedoch serverseitig (im Speicher oder zum Beispiel in einer Datenbank) und sendet an den Client lediglich eine signierte Session-ID.

Response-Rendering

Nachdem die Geschäftslogik ausgeführt wurde, muss häufig das Ergebnis dieser Logik an den Client übertragen werden. Hierzu dient der Body innerhalb einer HTTP-Response. Webanwendungen liefern hier klassischerweise HTML aus. Aber natürlich müssen auch andere Formate, wie JSON für APIs, ausgeliefert werden. Die Frameworks erfinden hierbei nichts Neues, sondern bedienen sich bei bestehenden Technologien und Libraries und bieten hierfür eine Integration an.

Spring beispielsweise erlaubt eine ganze Menge an Rückgabetypen in seinen Controllern (s. [springreturntypes]). Das Framework reagiert anschließend unterschiedlich je nach Rückgabewert. Für HTML wird in der Regel eine Template Engine (z. B. [Thymeleaf], [JSP] oder [Freemarker]) genutzt, wohingegen JSON-Responses durch Serialisieren des Rückgabewertes über [jackson] erzeugt werden.

Das Play-Framework lässt nur einen validen Rückgabewert für Controller zu. Hier muss der Anwendungsentwickler seine Objekte vorher in das passende Format konvertieren. Play liefert hierzu allerdings für HTML eine eigene Templatesprache und empfiehlt, für JSON jackson zu nutzen [playjson].

Andere Frameworks wiederum bieten dem Entwickler hier jede Freiheit, indem diese keinerlei Unterstützung mitbringen und den Entwickler wählen lassen.

Request-Binding

Gerade bei Action-basierten Web-Frameworks wünscht man sich häufig die mit dem Request geschickten (Meta-)Daten als Objektmodell. Dafür bieten Frameworks unterschiedliche Möglichkeiten an.

SpringMVC zum Beispiel erlaubt eine fast unendliche Anzahl an Methodenargumenten (s. [springarguments]). Das Play-Framework hingegen bietet innerhalb einer Controller Action Zugriff auf den Request durch die implizite Variable `request`:

```
...
def index() = Action {
  request.get("myHeader").orNull
}
...
```

Komponentenbasierte Frameworks brauchen kein für den Entwickler sichtbares Request-Binding. Diese geben dem Entwickler Zugriff auf die UI-Elemente der Seite und abstrahieren somit komplett vom Request.

Ressourcenbasierte Frameworks sind da anders. Sie geben dem Entwickler direkten Zugriff auf ein Request-Objekt. Dieser ist anschließend selbst dafür verantwortlich, die benötigten Daten aus dem Request zu extrahieren.

Sicherheitsaspekte

Von einem Web-Framework wünscht man sich, dass die typischen Angriffsvektoren auf Webanwendungen verhindert werden. Auf Anhieb fallen hier Dinge wie Cross-Site-Scripting, XSRF oder auch SQL-Injection ein.

Ein Framework sollte bei der Verwendung der HTTP-Parameter und beim Verarbeiten von Requests und Request-Parametern sowie beim Wiederherstellen von Sitzungsdaten sicherstellen, dass dies zuverlässig sicher funktioniert und illegale Zugriffe unterbunden werden.

Fazit

Die Frage nach dem geeigneten Web-Framework lässt sich nicht einfach beantworten. Je nach Anforderungen genügt zum Teil das einfachste, aber je nach Anforderungen steigt der Bedarf nach Funktionalität, die HTTP mit sich bringt.

Komponentenbasierte Web-Frameworks bieten ein Programmiermodell, das zwar Rich-Client-Entwicklern vertraut erscheinen mag, aber die Vorzüge des Webs so kapselt, dass sie sich nur schwerlich nutzen lassen. Zudem leidet bei diesen häufig die Skalierbarkeit, da sie meistens eine serverseitige Session voraussetzen.

Action-basierte und ressourcenbasierte Frameworks sind da schon besser geeignet, auf zukünftige Qualitätsanforderungen mit HTTP-Mitteln zu reagieren.

In dieser Ausgabe haben wir ein paar der wesentlichen Eckpfeiler von Web-Frameworks betrachtet, um hier zumindest für die Serverseite eine Entscheidungshilfe zu skizzieren.

Links

[ApacheWicket] `http://wicket.apache.org`
[Freemarker] `http://freemarker.incubator.apache.org`
[Grails] `https://grails.org`
[httpbis] Hypertext Transfer Protocol, `https://httpwg.github.io`
[httpdecisions] `https://camo.githubusercontent.com/4e15cccf2a9277dcca2c8824092547dee7058744/68747470733a2f2f726177676976746875622e636f6d2f666f722d4745542f687474702d6465636973696f6e2d6469616772616d2f6d-61737465722f6874747064642e706e67`
[jackson] `https://github.com/FasterXML/jackson`
[JAX-RS] Java API for RESTful Services, `https://jax-rs-spec.java.net`
[JSF] JavaServer Faces Technology, `http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html`
[JSP] JavaServer Pages Technology, `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`
[liberator] `http://clojure-liberator.github.io/liberator/`
[PlayFramework] `https://www.playframework.com`
[playjson] `https://www.playframework.com/documentation/2.4.x/JavaJsonActions`
[REST] Representational State Transfer, Kapitel 5 aus der Dissertation von R. J. Fielding von 2000 an der University of California, `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`
[RFC2616] Request for Comments 2616: Hypertext Transfer Protocol – HTTP/1.1, `https://tools.ietf.org/html/rfc2616`
[RFC7230] Request for Comments 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, `https://tools.ietf.org/html/rfc7230`
[RFC7231] Request for Comments 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, `https://tools.ietf.org/html/rfc7231`
[RFC7232] Request for Comments 7232: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests, `https://tools.ietf.org/html/rfc7232`
[RFC7233] Request for Comments 7233: Hypertext Transfer Protocol (HTTP/1.1): Range Requests,

https://tools.ietf.org/html/rfc7233
[RFC7234] Request for Comments 7234: Hypertext Transfer Protocol (HTTP/1.1): Caching, https://tools.ietf.org/html/rfc7234
[RFC7235] Request for Comments 7235: Hypertext Transfer Protocol (HTTP/1.1): Authentication, https://tools.ietf.org/html/rfc7235
[springarguments] Supported method argument types, Spring Framework Reference Documentation, http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-arguments
[SpringMVC] http://projects.spring.io/spring-framework/
[springreturntypes] Supported method return types, Spring Framework Reference Documentation, http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-return-types
[Thymeleaf] http://www.thymeleaf.org
[Vaadin] https://vaadin.com/home
[webmachine] https://github.com/webmachine/webmachine
[webster] https://github.com/pschirmacher/webster

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com

Michael Vitz ist Consultant bei innoQ und verfügt über mehrjährige Erfahrung in der Entwicklung und im Betrieb von JVM-basierten Systemen. Zurzeit beschäftigt er sich vor allem mit den Themen DevOps, Continuous Delivery und Cloud-Architekturen sowie Clojure. E-Mail: michael.vitz@innoq.com

Java User Groups Termine

Mitte März bis 12. Mai 2016

Bitte senden Sie Ihre Termine an: susanne.herl@sigs-datacom.de

Augsburg; Infos unter http://www.jug-augsburg.de

21.04.2016, 19:00 Uhr – Microservices mit Spring Cloud und Netflix OSS,
mit Thomas Letsch. Ort: SHI GmbH & Co. KG, Curt-Frenzel-Str. 12
Microservices sind ein sehr interessanter Architekturstil, wenn es um verteilte und hoch skalierbare Anwendungen geht. Ein Vorreiter ist hier Netflix. Nicht nur, dass Netflix intern auf Microservices setzt, sie haben auch ihre Infrastrukturkomponenten als Open Source veröffentlicht. Spring Cloud dient dazu als Integrationsmechanismus und Abstraktionsschicht.

Bern; Infos unter http://www.jug.ch

29.03.2016, 18:00 – 19:00 Uhr - Wie Weltklasse Testing aussieht und warum es Manuelles Testing nicht gibt,
mit Ilari Henrik Aegerter. Ort: Schmiedstube, Schmiedenplatz 5, 3011 Bern
Software Testing ist eine junge Disziplin und wie alles Neue noch weitgehend unverstanden. Ist Software Testing ein technisches Problem, das man mit Engineering-Ansätzen lösen kann? Was genau ist das Ziel von Software Testing? Was kann man tun, um ein wirklich guter Tester zu werden?

Bremen; Infos unter http://www.meetup.com/de-DE/jugbremen/event

13.04.2016, 18:30 Uhr – Jigsaw: The Module Platform arrived in JDK 9,
mit Wolfgang Weigend. Ort: CTS Eventim Contrescarpe 75a, Bremen
Das Projekt Jigsaw hat die primäre Aufgabe, das Design und die Implementierung eines Standard-Modulsystems für die Java-SE-Plattform und für das JDK 9 bereitzustellen. Dabei soll die durchgängige, dynamische und einfache Anpassbarkeit von Plattform und JDK, auch für kleine Endgeräte, berücksichtigt sowie die Verbesserung von Sicherheit und Wartbarkeit von Java-SE-Plattform-Implementierungen, speziell vom JDK, verwirklicht werden. Die Verfügbarkeit vom JDK 9 ist für Ende März 2017 geplant.

Darmstadt; Infos unter http://www.jug-da.de

21.04.2016, 18:30 – 20:30 Uhr - Getting started with Spring Cloud,
mit Josh Long. Ort: Accso GmbH, Berliner Allee 58, 64295 Darmstadt
Spring Cloud is here! This is a deep dive on how to make connect and consume microservices, and is a natural next step after the introduction to building microservices with Spring Boot.

12.05.2016, 18:30 – 20:30 Uhr – Robuste Cloud-Architekturen,
mit Agim Emruli. Ort: IT FOR WORK, c/o IHK Darmstadt, Rheinstrasse 89, Raum S 18 (Untergeschoss), 64295 Darmstadt
Dieser Vortrag geht auf die Herausforderungen von dynamischen, verteilten und potenziell instabilen Architekturen und auf neue Patterns sowie Möglichkeiten hin zu einer robusten und adaptierenden Softwarearchitektur ein. Gängige Entwurfsstrategien sowie Patterns wie Circuit-breakers, Bulkheads und Handshakes werden besprochen.

Dresden; Infos unter http://www.jugsaxony.de

07.04.2016, 19:00 – 21:00 Uhr - Model-View-ViewModel mit JavaFX,
mit Manuel Mauky und Max Wielsch. Ort: BA Dresden, Raum 305, Hans-Grundig-Str. 25, 01307 Dresden
MVVM ist ein Frontend-Architektur-Pattern, bei dem der UI-Zustand und die Präsentationslogik im sogenannten ViewModel gekapselt werden. Der Vortrag stellt die Idee dieses Musters vor und zeigt, wie JavaFX-Oberflächen nach diesem Muster testgetrieben umgesetzt werden können. Außerdem wird das Open-Source-Framework „mvvmFX“ vorgestellt, welches die Entwicklung von JavaFX mit MVVM vereinfacht.

Düsseldorf; Infos unter http://www.rheinjug.de

07.04.2016, Über Faulheit, Feigheit, Unfähigkeit und Clean Code,
mit Dr. Reik Oberrath und Jörg Vollmer. Ort: Heinrich-Heine Universität Düsseldorf, Hörsaal 5C
Im Jahr 2008 erschien das Buch „Clean Code" von R. C. Martin. Vor sechs Jahren wurde die Clean-Code-Developer-Bewegung von R. Westphal und S. Lieser gegründet. Es ist Zeit, der Frage nachzugehen, inwieweit die Appelle an unsere Professionalität Wirkung zeigen und wie es mit unserem Entwickler-Wertesystem bestellt ist.

Frankfurt; Infos unter https://sites.google.com/site/jugffm

30.03.2016, 18:30 Uhr - JDK 9 und die modulare Plattform Jigsaw,
mit Wolfgang Weigend. Ort: Deutsche Nationalbibliothek – s. Bremen

27.04.2016, 18:30 Uhr - So viel Zeit muss sein: Der Umstieg auf die neue Date Time API in Java 8,
mit Dipl.-Inf. Jens-Hagen Syrbe. Ort: Deutsche Nationalbibliothek
Es werden die Designprinzipien der neuen Date Time API in Java 8 vorgestellt. Die API orientiert sich an der Java-Bibliothek JodaTime und lässt zahlreiche, fast schon lieb gewonnene Schwächen der alten Abstraktionen hinter sich. In der neuen API wird zwischen der kontinuierlichen Zeit (bzw. Maschinenzeit) und der menschlichen Zeit unterschieden.

Göttingen; Infos unter http://www.java.de/stammtisch-goettingen

23.03.2016, 08:00 - 18:45 Uhr – CDI Pattern, mit Sven Ruppert. Ort: Bunsenstr. 3-5, Math. Institut, 37073 Göttingen
CDI (Context Dependency Injection) ermöglicht es dem Entwickler, skalierbare und flexible Architekturen aufzubauen, die sowohl auf einem Java(EE)-Server als auch auf dem Desktop laufen. Wie aber sollten Entwurfsmuster aufgebaut werden unter Verwendung von CDI? Was für einen Einfluss hat Java 8 auf die Muster, wie wird es mit der neuen Streams API kombiniert?

Kassel; Infos unter http://www.jugh.de

31.03.2016, 18:00 Uhr - Mut zur Fachlichkeit,
mit Lars Röwekamp. Ort: Ing.-Schule Universität Kassel, Wilhelmshöher Allee 71-73, 34121 Kassel, Raum HS 0315
Die Session zeigt einfache, aber sehr effektive Strategien, wie bestehende Anwendungen in ein fachlich getriebenes Architektur- und API-Design überführt werden können – Aha-Effekte garantiert.

28.04.2016, 18:00 Uhr - Event Sourcing: Einführung und Best Practices,
mit Michael Plöd. Ort: voraussichtlich: Ing.-Schule Universität Kassel, Wilhelmshöher Allee 71-73, 34121 Kassel, Raum HS 0315
Der Vortrag führt in das Thema Event Sourcing ein und stellt gängige Patterns vor. Des Weiteren werden Themen wie Rollback von Events (Kompensation-Services), Möglichkeiten der Erstellung von Daten-Snapshots und -Änderungen durch externe Systeme vorgestellt. Abgerundet wird der Vortrag durch eine Diskussion möglicher Einsatzszenarien für Event Sourcing.

Köln; Infos unter https://www.xing.com

21.03.2016, 18:30 – 21:00 Uhr – JUGC: Git from the inside (Powersession). Ort: anderScore GmbH Köln
Alles, was Sie schon immer über Git wissen wollten ...

Luzern; Infos unter http://www.jug.ch

31.03.2016, 18:00 – 19:15 Uhr - Wir trimmen unsere Monolithen fit für die Zukunft!
mit Anatole Tresch. Ort: Hochschule Luzern - Wirtschaft, Zentralstrasse 9, Raum 3.24
Der Vortrag soll zeigen, welche Vorteile Microservices bieten und warum dieser Architekturansatz in den Rucksack jedes modernen Software-Ingenieurs und -Architekten gehört. Und wie lose Kopplung konkret in technischer, betrieblicher und organisatorischer Hinsicht im Kleinen, wie im Großen erreicht werden kann.

Münster; Infos unter http://www.jug-muenster.de

23.03.2016, 18:00 Uhr - Custom WebComponents, mit Hendrik Ebbers. Ort: LVM Versicherung, 3.01.08 Konferenzraum II Links

27.04.2016, 18:00 Uhr - Graylog, mit Kai Roepke. Ort: LVM Versicherung, 3.01.08 Konferenzraum II Links

Stuttgart; Infos unter http://www.jugs.org

21.03.2016, 18.30 Uhr - ObjektForum Stuttgart: Mit AngularJS Projekte schnell an die Wand fahren,
mit Rouven Röhrig. Ort: Agnes-Kneher-Platz, Große Falterstraße 6a, 70597 Stuttgart-Degerloch
In diesem Vortrag werden wir uns häufige Fehler anschauen und Alternativen aufzeigen, die zu einem wartbareren Code führen. Voraussetzungen hierfür sind u.a. ein hohes Maß an Automatisierung, testbarer Code, Modularisierung und eine strikte Trennung von Repräsentations- und Business-Logik.

12.05.2016, 18:30 Uhr - Model-View-ViewModel mit JavaFX,
mit Manuel Mauky und Max Wielsch. Ort: Agnes-Kneher-Platz, Große Falterstraße 6a, 70597 Stuttgart-Degerloch, s. Dresden

Zürich; Infos unter http://www.jug.ch

22.03.2016, 16:00 – 17:00 Uhr - Anti-fragile Cloud-Architekturen,
mit Agim Emruli. Ort: Technopark Zürich, Technoparkstr. 1, Zürich, Raum Cobol
Dieser Talk gibt eine Einführung in die neuesten Entwurfsmuster und Technologien, die auf der Basis von Open-Source-Stacks wie Spring Cloud, Netflix OSS, Apache Zookeeper sowie Consul basieren.

30.03.2016, 18:00 – 19:00 Uhr - Wie Weltklasse Testing aussieht und warum es Manuelles Testing nicht gibt,
mit Ilari Henrik Aegerter. Ort: PH Zürich, Lagerstrasse 2, 8090 Zürich, Gebäude LAC, Raum E071, s. Bern