Doppelplusgut

Das Grails-Webframework – Teil 1: Grundlagen und produktiver Einsatz

Manuel Breitfeld, Tobias Kraft, Peter Soth

Grails wirbt mit schnelleren Entwicklungszeiten im Vergleich zu Java EE. Hierfür wird das DRY-Prinzip (DRY steht für „Don't repeat yourself"), das Andy Hunt und Dave Thomas in ihrem Buch „Der pragmatische Programmierer" formulierten, als elementar angesehen. Die Zeit ist natürlich auch bei Java EE nicht stehen geblieben. Wo liegen angesichts dessen die zusätzlichen Stärken des Grails-Frameworks? In der zweiteiligen Artikelserie schreiben wir über Erfahrungen, die wir in den letzten Jahren mit Grails sammeln konnten. Im ersten Teil werden die Grundlagen beleuchtet, im zweiten wird die Realisierung eines komplexen Portal-Frameworks mit Grails beschrieben.

Ein Java-basiertes Framework muss her

Eine Kundenanfrage brachte den Stein ins Rollen: Wir sollten eruieren, ob es möglich wäre, einen Excel-Prozess in eine Webapplikation zu überführen. Java EE kam nicht in Frage, da das Budget auf Kundenseite begrenzt war und dafür nicht ausreichte.

Alle Java-basierten Frameworks, die wir bis dato kannten, erschienen uns zu schwerfällig. Daher zogen wir zunächst „Ruby On Rails" in Betracht, das wir ausgesprochen interessant fanden, mussten die Idee jedoch wieder aufgeben, da unsere Kunden eine Java-Infrastruktur wie Tomcat oder Oracle WebLogic Server einsetzen.

Weitere Nachforschungen führten uns schließlich zum Grails-Framework. Es versprach alles, was uns an Ruby On Rails gefiel, allerdings mit einem unschätzbaren Pluspunkt: Grails-Webapplikationen sind Java-Applikationen, die auf einem Java EE-Applikationsserver als WAR-Datei installiert werden können. Des Weiteren punktet Grails mit einer starken Community, einer sehr guten Dokumentation – auch in Form von Büchern – und dem Einsatz von bewährten Industriestandards wie Spring, Hibernate, SiteMesh, Quartz usw.

Zu guter Letzt fallen für Grails auch keine Lizenzkosten an, da es auf der Apache License 2.0 basiert und eine Open-Source-Software ist.

Grails gibt es bereits mehr als sieben Jahre. Man kann daher nicht von einem Hype reden, der kommt und wieder geht. Es werden jährlich mehrere Minor- und Major-Releases veröffentlicht, die neben weiteren Features auch jeweils neue Library-Versionen von beispielsweise Groovy integrieren. Für Grails 3.0, dem nächsten großen Release, sind die zwei großen Themenblöcke „Modularisierung" und „Umstellung" des Builds auf Gradle vorgesehen.

Verbesserungen für den Entwickler durch Grails

Dem DRY-Prinzip folgend, bildet bei Grails die Domain-Klasse den zentralen Bezugspunkt. Diese wird von der Datenbank über die Service-Schicht bis zur Benutzungsoberfläche durchgereicht. Die aus der früheren Java EE-Zeit bekannten Infrastruktur-Muster wie „Service Locator", „Transfer Object" und „Business Facade" sind nicht nötig. Das befreit den Entwickler von unnötiger Schreibarbeit und bei Modifikationen, die speziell bei agilen Projekten oft auftreten, muss eine Änderung nur noch an der Domain-Klasse und nicht mehr in allen Hilfsschichten darüber durchgeführt werden. Natürlich gibt es auch in Grails Anwendungsfälle, bei denen es sinnvoll ist, ein Entwurfsmuster einzusetzen. Die „CommandObjects" sind in Grails beispielsweise das Pendant zu „Transfer Objects" und bieten sich bei komplexeren Dialogen zur Abstraktion von Domain-Objekten an.

Ein weiteres zentrales Prinzip ist „Convention over Configuration". Hierbei wird beispielsweise eine Controller-Klasse mit dem Suffix `Controller` beendet. Es ist nicht nötig, dies zusätzlich in der web.xml-Datei zu konfigurieren. „Convention over Configuration" bedeutet aber nicht, dass man dadurch Flexibilität verliert. Es bedeutet viel mehr, dass der Standard vordefiniert ist, man bei Bedarf aber auch davon abweichen kann.

Wie eingangs erwähnt, hielten diese Prinzipien auch nach und nach Einzug in Java EE. Grails verfügt jedoch noch über weitere interessante Funktionen.

Grails bietet noch mehr

Die Sprache Groovy kommt in vielen Fällen mit weniger Code-Zeilen aus als Java. Wird aus Gründen wie Performance oder Wiederverwendung Java-Code gewünscht, so kann dieser ohne Probleme eingebunden werden. Groovy besticht beispielsweise auch mit einem einfacheren Handling von Maps und Collections. In Java müssen diese zuerst deklariert und initialisiert werden, bevor damit gearbeitet werden kann.

Mit Closures bietet Groovy eine weitere hilfreiche Funktion an, um den Quellcode übersichtlicher und kürzer zu halten. Unter Closures versteht man Code-Blöcke, denen Variablen zugewiesen und ähnlich wie Objekte als Parameter übergeben werden können. In Listing 1 werden einfache Anwendungsfälle für Closures vorgestellt. Eine ähnliche Funktionalität wird Java mit der Version 8 mit den Lambda-Ausdrücken [GüLe13] erhalten.

```
// einfache Closure-Definiton mit Argumenten
def add = { a, b -> a + b }
assert 55 == add(33, 22)
assert 'GroovyGrails' == add('Groovy ', 'Grails')

// direkte Anwendung von Closures für Listen
def list = ['Groovy', 'Grails', 'Java', 'GORM']
assert 'Java' == list.find { e -> e == 'Java' }
assert ['Groovy', 'Grails'] == list.findAll { it.size() > 4 }

// Closure für das Dateihandling
new File('/data/').eachFileMatch( ~".*xml" ) { f ->
  if ( f.lastModified() <= yesterday ) {
    f.delete()
  }
}
```

Listing 1: Einfache Anwendungsfälle für Closures

Des Weiteren eignet sich Groovy auch sehr gut, um eine DSL (DSL steht für Domain Specific Language) zu entwickeln.

Grails bietet als Full-Stack-Webframework alles aus einer Hand. Nach dem Herunterladen und Entpacken kann man sofort mit der Entwicklung beginnen, ohne dass weitere Komponenten installiert oder konfiguriert werden müssen. Ein ab-

gespeckter Tomcat-Server liegt der Grails-Distribution bei und nach dem Ausführen des Befehls `create-app <Name>` in der Grails-Kommandozeile lässt sich die neu erstellte Applikation direkt über den Befehl `run-app` sofort starten. Auch das Testen auf Basis von JUnit ist direkt integriert.

Neben den beiden Kommandos zum Erstellen und Starten einer Applikation gibt es eine Vielzahl weiterer Kommandos, mit denen sich Basis-Code-Fragmente generieren und erstellen lassen.

GORM, die Persistenzschicht von Grails, ist ein flexibler Layer, der neben Hibernate auch Unterstützung für weitere Persistenztechnologien wie NoSQL-Datenbanken bietet. In Kombination mit Groovy ergeben sich hier ungeahnte Möglichkeiten wie etwa dynamische Finder-Methoden. Mit dem Befehl `Articles.findAllByTitle("JavaSpektrum")` können beispielsweise alle Artikel mit dem Titel `"JavaSpektrum"` ermittelt werden, ohne eine Zeile SQL zu schreiben. Die Finder-Methoden mit den Entitäten der Domain-Klasse werden hierbei erst zur Laufzeit gebunden. Verwendet man eine fortschrittliche Entwicklungsumgebung wie IntelliJ IDEA, dann werden dem Entwickler alle möglichen Finder-Methoden angezeigt. Ein sehr nützliches Feature. Listing 2 zeigt neben einigen Finder-Methoden auch die mit der Grails-Version 2.0 eingeführten Where-Queries. Diese Art von Abfragen bietet noch mehr Flexibilität und Möglichkeiten zum Erstellen komplexer Abfragen.

```
// Finder auf das Attribut login der Domain-Klasse Author
def person = Author.findByLogin("soth")

// UND verknüpfter Finder mit Assoziation
Articles.findAllByTitleAndAuthor("JavaSpektrum", person)

// OR verknüpfter Finder für Listen mit Blätterung
Articles.findAllByTitleOrTitle("JavaSpektrum", "ObjektSpektrum",
  [offset:0, max:10, sort: "title", order: "asc"])

// where-Abfrage mit Attributen aus der zugehörigen Domain-Klasse
def query = Author.where {
  (hobby != "Reading") || (gender == "female" && age > 80)
}
def specialAuthors = query.list(sort:"lastName")

// where-Abfrage mit Beziehung
def query = Author.where {
  articles { title ==~ "Grails" } || firstName == "Peter"
}
```

Listing 2: Einige Finder-Methoden und Where-Queries (Grails, Version 2.0)

Scaffolding ermöglicht es dem Entwickler, eine Benutzungsoberfläche aus Domain-Klassen auf einfachem Weg zu erzeugen. Dies ist ausgesprochen angenehm für das Prototyping oder die Erstellung von Administrations-Oberflächen. Mit Grails ist dynamisches und statisches Scaffolding möglich. Ersteres generiert den Code der Views und Controller immer zur Laufzeit aus Templates. Die statische Methode generiert beide Komponenten einmalig und speichert sie in den entsprechenden Dateien. Damit kann der Quellcode im Nachhinein an die eigenen Bedürfnisse angepasst werden. Ändern sich allerdings Attribute der Domain-Klasse, dann müssen diese natürlich auch manuell in den generierten View-Dateien nachgezogen werden.

URL-Mapping gepaart mit einer exzellenten Unterstützung für JSON ermöglicht den einfachen Aufbau REST-basierter Schnittstellen. In einem aktuellen Projekt konnten wir mit nur wenigen Zeilen Code Domain-Klassen als JSON-Objekte anbieten und über einen ähnlichen Mechanismus auch Schreiboperationen erlauben. Tabelle 1 veranschaulicht die Funktionsweise der Datei UrlMappings.groovy, in der die Definitionen abgelegt werden. Dabei wird deutlich, dass Grails von Haus aus SEO-freundliche (Search Engine Optimization) URLs anbietet. Ohne spezielle Konfiguration ist eine Action stets über den Pfad `/controller/action` aufrufbar, beispielsweise führt die URL `http://exensio.de/user/list` einen Aufruf der `list`-Action im UserController durch.

Der Plug-in-basierte Entwicklungsansatz ermöglicht die projektübergreifende Wiederverwendung von Komponenten. Es stehen über 500 Plug-ins zum Herunterladen bereit, die jeweils einfach durch eine Konfigurationszeile eingebunden werden können. Wir benutzen manche Out-Of-The-Box, wie beispielsweise das Spring-Security-Plug-in. Andere wie das Elasticsearch-Plug-in müssen für unsere Anforderungen angepasst werden. Des Weiteren überlegen wir uns, inwiefern wir Teile unseres Quellcodes durch Plug-ins wiederverwenden können, was Zeit und Kosten spart. Weitere Informationen zu diesem Thema finden sich auf dem Blog von Burt Beckwith [Plugins].

Produktivitätssteigerung

Wir sehen in unseren Projekten eine Produktivitätssteigerung zwischen 20 und 30 Prozent im Vergleich zur Java-Welt. Das ermöglicht vor allem der kompaktere Code von Groovy und der Einsatz von Plug-ins.

Konfiguration in UrlMappings.groovy	Beschreibung
`"/admin/system"(controller: "system")`	Der Aufruf `http://exensio.de/admin/system` leitet auf die Default-Action im SystemController
`"/$client/portal/$action?" { controller="portal" constraints { bu( inList: ['bmw','audi'] ) } }`	Es können auch Variablen innerhalb des Mappings definiert werden, die an die Action durchgereicht werden. Hierbei kann schon direkt bei der Konfiguration der Wertebereich von Variablen beschränkt werden Der Aufruf `http://exensio.de/bmw/portal/login` leitet auf die Action `login` im PortalController. Der `login`-Action steht dabei der Parameter `client` mit dem Wert `bmw` zur Verfügung
`"500"(controller:'globalException',action:'error')`	Im URL-Mapping kann auch einfach definiert werden, was bei Server-Fehlern passieren soll. Dieses Beispiel definiert für den HTTP-Error 500 eine Umleitung auf einen eigenen Controller, der dann die Fehlerbehandlung übernimmt

Tabelle 1: Funktionsweise der Datei UrlMappings.groovy

Angenehm sind insbesondere für Entwickler die kürzeren Zeiten für das Neustarten des Servers oder auch die Startzeiten zum Durchführen der Tests im Vergleich zu Java EE.

Produktivbetrieb

Wir betreiben Grails-Applikationen erfolgreich sowohl auf reinen Open-Source-Umgebungen (Apache WS, Tomcat, MySql) sowie auf kommerziellen Umgebungen (IIS, Oracle Weblogic Server, Oracle DB). Während der Entwicklung sollten Tests auf der Zielplattform jedoch schon früh durchgeführt werden. So hat sich gezeigt, dass sich eine spätere Migration von MySql auf eine Oracle DB – abhängig von den verwendeten Abfragen – aufwendig gestaltet. Der geneigte Leser findet auf unserem Blog [GrailsProd] zum Thema „Grails in Produktion – mit Apache, Tomcat und MySQL" weitere Informationen.

Performance

Es ist ab und an in Foren zu lesen, dass Grails langsam sei. Hierbei wird auf Groovy verwiesen, das bekanntlich Funktionen zur Laufzeit dynamisch bindet. Unsere Erfahrung aus vielen Projekten sieht anders aus. Durch den Einsatz von Standard-Caches läuft die Performance analog wie die einer Java-Applikation. Neben dem Caching für Hibernate-Abfragen kommen insbesondere auch Caches auf Service-Ebene zum Einsatz. Des Weiteren besteht die Möglichkeit, ressourcenintensive Operationen direkt als Java-Code zu integrieren. Dies mussten wir bisher noch nicht umsetzen. Wie auch in anderen Projekten sollten Lasttests ausgeführt werden, um eine optimale Performance zu erzielen. Hierbei zeigt sich häufig, dass der Engpass nicht unbedingt im Code zu suchen ist, sondern beispielsweise in nicht optimierten Indizes für die eingesetzte Datenbank. Wir benutzen Apache JMeter für die Durchführung der Lasttests.

Warum kommt man bei Grails nicht ohne Tests aus?

Da Grails auf der dynamischen Sprache Groovy basiert, bereitete es uns am Anfang Sorge, dass Fehler erst zur Laufzeit auftreten könnten, die bei Java bereits während des Kompilierens gefunden worden wären. Grails kann über Plug-ins viele Test-Frameworks wie JUnit, Spock [Spock] oder Geb [Geb] integrieren. So benutzen wir beispielsweise gerne Spock, da es Data-Driven-Testing so sagenhaft einfach macht. Es brachte auch den schönen Nebeneffekt mit sich, dass Projektbeteiligte auf einmal ähnlich viel Spaß daran hatten, Tests zu schreiben, wie neue Anforderungen zu entwickeln. Rückblickend lautet unser Resümee: Mit dem Einsatz von Grails verbesserte sich unsere Testabdeckung erheblich. Neben dem Schreiben der Tests ist es erforderlich, sie regelmäßig automatisiert auszuführen. Hierzu verwenden wir Jenkins. Er fungiert auch gleichzeitig als zentraler Build-Server, um eine konstante Generierung der WAR-Dateien sicherzustellen.

Fazit

Der Einsatz von Grails zeigte sich in der Praxis als durchweg positiv. Die anfängliche Lernkurve ist für einen erfahrenen Java-Entwickler steil. Die Ängste, die in Foren bezüglich Performance geschürt werden, oder die Behauptung, dass sich dieses Framework nur für kleine Projekte eignet, können wir nicht bestätigen und setzen Grails mittlerweile in fast 80 Prozent unserer Projekte ein. Im nächsten Teil unseres Artikels werden wir beschreiben, wie wir ein Portal-Framework mit Grails umgesetzt haben, das bereits in mehreren Kundenprojekten Verwendung findet.

Literatur und Links

[Geb] `http://gebish.org/`

[GrailsProd] P. Soth, Grails in Produktion – mit Apache, Tomcat und MySQL, 13.2.2012, Exensio, `http://blog.exensio.de/2012/02/teil-1-grails-in-produktion-mit-apache.html`

[GüLe13] M. Günther, M. Lehmann, Lambda-Ausdrücke in Java 8, in: JavaSPEKTRUM, 3/2013, s. a. `http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2013/03/guenther_lehmann_JS_03_13_va54.pdf`

[HuTh03] A. Hunt, D. Thomas, Der pragmatische Programmierer, Hanser Fachbuch, 2003

[Plugins] Converting Grails Applications to Plugins and vice versa, Blog von Burt Beckwith, 22.7.2013, `http://burtbeckwith.com/blog/?p=1973`

[Spock] `http://spockframework.org`

[UrlMapping] `http://grails.org/doc/latest/ref/Plug-ins/URL%20mappings.html`

Manuel Breitfeld ist nach seinem Studium der Softwaretechnik seit 2010 als Consultant für die exensio GmbH tätig. Dabei umfasst sein Tätigkeitsfeld sowohl Entwicklungsprojekte von Enterprise-Portalen und Webapplikationen als auch Beratungsprojekte im Usability- und Intranet-Bereich.
E-Mail: manuel.breitfeld@exensio.de

Tobias Kraft war nach seinem Studium des Wirtschaftsingenieurwesens in mehreren Software- und IT-Beratungshäusern als Consultant und Softwarearchitekt tätig (u. a. sd&m – heute Cap Gemini). Seit 2009 beschäftigt er sich bei der exensio GmbH mit der Architektur und Umsetzung von Enterprise-Portalen sowie Webapplikationen basierend auf Java-Technologien und dem Grails-Framework.
E-Mail: tobias.kraft@exensio.de

Peter Soth arbeitete nach seinem Studium der Elektrotechnik und Informatik mehrere Jahre als Software-Engineer und Senior Consultant bei international tätigen Unternehmen, wie Hewlett Packard, SAS Institute und BEA Systems (nun Oracle). Im Jahre 2006 gründete er die exensio GmbH mit. Seine Tätigkeitsschwerpunkte umfassen die Architekturberatung und Realisierung von Enterprise-Portal-Projekten mittels Java-Technologien.
E-Mail: peter.soth@exensio.de