Grails – Die Suche ist vorbei

Stefan Glase, Christian Metzler
OPITZ CONSULTING Gummersbach GmbH

Schlüsselworte:

Groovy, Grails, Java, Java EE, Spring, Hibernate, Rapid Prototyping, Webanwendung, Webentwicklung

Einleitung

Grails ist ein Framework für Webapplikationen auf Basis der dynamisch typisierten Programmiersprache Groovy sowie bewährten Technologien wie dem Spring Framework und Hibernate. Eine Vielzahl von Plug-Ins für Grails (derzeit gibt es mehr als 600) macht es möglich, wiederkehrende Problemstellungen mit bewährten Lösungen umzusetzen.

Dieser Vortrag startet mit einem kurzen Überblick über die Grundlagen von Groovy & Grails und zeigt im Anschluss an Beispielen und echtem Code eine Auswahl der interessantesten Funktionen und Möglichkeiten.

Groovy

Groovy ist eine agile und dynamisch typisierte Programmiersprache, die aber auch statische Typisierung unterstützt. Groovy-Code wird zu Java Virtual Machine Bytecode kompiliert und ist interoperabel mit normalem Java Code und Java Bibliotheken. Mit wenigen Ausnahmen ist Java-Code zugleich auch syntaktisch korrekter Groovy-Code.

Groovy…

- baut auf den Stärken von Java auf.
- bietet für viele Java-Sprachmittel ausdrucksstärkere Groovy-Pendants.
- erlaubt Java-Entwicklern einen sehr schnellen und schrittweisen Einstieg.
- integriert sich nahtlos in die Java Welt aus Java-Objekten und Java-Bibliotheken.

Das typische „Hello World“-Beispiel gibt es für nahezu jede Programmiersprache und so soll auch eine Groovy-Variante nicht fehlen. Das folgende Listing zeigt gültigen und ausführbaren Groovy-Code:

```
class Greeter {
    def name
    def greet() { "Hello $name!" }
}

helloGroovy = new Greeter(name: 'Groovy')
println helloGroovy.greet()
```

Dem Java-Entwickler fallen hier sofort diverse Unterschiede zu altbekanntem Java-Code auf:

- Mit dem Schlüsselwort `def` werden dynamische Typen deklariert.
- Die Sichtbarkeit von Methoden ist standardmäßig `public`.

- Felder sind automatisch mittels Setter und Getter zugreifbar.
- Semikolons am Zeilenende eines Ausdrucks sind optional.
- Variablen innerhalb von Groovy Strings werden aufgelöst.
- Das `return` Schlüsselwort ist optional.
- Es gibt Map-Konstruktoren für die Felder einer Groovy Bean.
- Für bestimmte Java-Ausdrücke wie `System.out.println()` gibt es Kurzformen.

Collections

Für eine bessere Ausdrucksstärke des Groovy-Codes gegenüber herkömmlichem Java-Code sorgen unter anderem die in Groovy stark verbesserten Collections.

Listen

- Java:

```
int[] zahlen = new int[] { 1, 2, 3 }
```

- Groovy:

```
zahlen = [1, 2, 3]
```

Maps

- Java:

```
Map<String, String> laender = new HashMap<String, String>();
laender.put("de", "Deutschland");
laender.put("gb", "Großbritannien");
```

- Groovy:

```
laender = [de: 'Deutschland', gb: 'Großbritannien']
```

Ranges

- Groovy:

```
buchstaben = 'a'..'d' // entspricht ['a','b','c','d']
zahlen = 22..27 // entspricht [22,23,24,25,26,27]
```

Closures

Ein weiteres Sprachmittel, was seinen Weg zwar schon seit längerem in die Java-Welt sucht aber noch nicht gefunden hat, sind Closures, welche in Groovy zur Lesbarkeit von Code beitragen können.

```
def zahlen = 1..10
zahlen.each{ println it }
zahlen.findAll{ it % 2 == 0 }.each{ println it }
```

In der ersten Zeile wird hier eine Range definiert, wobei eine Range immer auch eine Liste ist. Alle Werte der Liste werden in der Anweisung in Zeile zwei zeilenweise ausgegeben. In der dritten Zeile werden nur alle geraden Zahlen zeilenweise ausgegeben.

Grails

Grails ist ein Fullstack-Webapplication-Framework vergleichbar mit Oracle ADF (Oracle Application Developer Framework). Grails basiert auf Java und Groovy und sitzt dabei auf den Schultern von etablierten Java-Bibliotheken wie dem Spring Framework, Hibernate und SiteMesh.

- Spring Framework als führendes Java EE Framework bei Grails verantwortlich für Dependency Injection, Inversion of Control und das webbasierte MVC Framework.
- Hibernate als führendes objektrelationales Mapping Framework bei Grails verantwortlich als Grundstein für GORM, also die Persistenzschicht von Grails.
- SiteMesh als mächtiges Layout und Dekorator-Framework bei Grails verantwortlich für ein konsistentes Look & Feel der erstellten Webapplikation.

Grails propagiert stark die Idee von „Convention over Configuration“ und bietet so eine gesunde Mischung aus Freiheit bestimmte Belange durch Konfiguration beeinflussen zu können und Einfachheit durch die Bereitstellung von sinnvollen Standardkonfigurationen.

Grails…

- macht intensiven Gebrauch von Groovy-Sprachmitteln.
- reduziert die Komplexität und erhöht die Produktivität.
- vereinfacht die Kommunikation mit der Datenbank.
- ermöglicht ein Rapid Prototyping mit Hilfe der Scaffolding-Funktionalität.
- kann über die Konsole als auch über die IDE entwickelt werden.
- kann mit Plug-Ins wiederkehrende Problemstellungen lösen.

Eine erste Applikation

Mit wenigen Konsolenbefehlen und Zeilen Code kann bereits eine erste Version einer Anwendung entwickelt und betrieben werden (Rapid Prototyping).

```
# Anlegen einer neuen Webanwendung
grails create-app grails-quiz

# Erstellen der Domänenobjekte
grails create-domain-class quiz.Question
grails create-domain-class quiz.Answer

# Implementierung der Domänenobjekte
class Question {
    String text
    static hasMany = [answers: Answer]

    static constraints = {
        text(blank: false, unique: true)
    }
    String toString() { text }
}

class Answer {
    String text
    boolean correct
    static belongsTo = [question: Question]

    static constraints = {
        text(blank: false)
    }
    String toString() { text }
}
```

```
# Erstellen der Controller
grails create-controller quiz.Question
grails create-controller quiz.Answer

# Scaffolding der Controller
class QuestionController {
    static scaffold = true
}

class AnswerController {
    static scaffold = true
}

# Starten der Applikation
grails run-app
```

Mit den oben dargestellten Befehlen und dem gezeigtem Quellcode haben wir an dieser Stelle eine funktionierende Anwendung, mit der sich Fragen und Antworten hinzufügen, ändern, löschen und auflisten (CRUD) lassen.

Testdaten und Bootstrapping

Damit nicht bei jedem Neustart Testdaten erzeugt werden müssen, kann nun ein erstes Plug-In bemüht werden (http://www.grails.org/plugin/build-test-data) oder aber über den Bootstrapping-Mechanismus von Grails manuell Testdaten erzeugt werden. Wir benutzen den `XmlSlurper` um aus einer vorbereiteten XML-Datei die Fragen und zugehörigen Antworten zu lesen.

```
# Testdaten in der Datenbank speichern
import quiz.Question
import quiz.Answer

class BootStrap {
  def init = { servletContext ->
    environments {
      development {
        def quizData = new XmlSlurper().parse("quiz-data.xml")
        quizData.question.each { question ->
          def q = new Question(text: question.@text.text()).save()
          question.answer.each { answer ->
            new Answer(question: q, text: answer.text(), correct:
answer.@correct.text()).save()
          }
        }
      }
    }
  }
}
```

Quiz Implementierung

Zunächst gilt es einen Controller anzulegen, der sich um die Darstellung der Fragen und Antwortmöglichkeiten kümmert. Dieser Controller soll auf der URL `quiz/index` (Standard: `controllerName/actionName/id`) zuerst die Liste der Fragen und deren Antworten aus der Datenbank laden und diese der View, welche für die Anzeige verantwortlich ist, bereitstellen.

```
grails create-controller quiz.Quiz

class QuizController {
    def index = {
        [questionList: Question.list()]
    }
}
```

Im zweiten Schritt muss die passende View für die Präsentation erzeugt und implementiert werden. Hier kann intensiv Gebrauch gemacht werden von der in allen GSP-Dateien standardmäßig verfügbaren Grails Taglib.

```
<html>
<head>
    <meta name="layout" content="main">
    <g:javascript library="prototype"/>
</head>
<body>
<div class="body">
    <h1>Grails Quiz</h1>
    <div id="results" class="message" style="display:none"></div>
    <g:each in="${questionList}" var="question" status="index">
    <p>
        <h2>${question.text}</h2>
        <ul>
            <g:each in="${question.answers}" var="answer">
            <li><g:remoteLink action="answer" id="${answer.id}"
update="results"
onSuccess="showResults()">${answer.text}</g:remoteLink></li>
            </g:each>
        </ul>
    </p>
    </g:each>
</div>
</body>
</html>
```

Zuletzt müssen wir noch die aufgerufene JavaScript-Methode `showResults()` sowie die Action `answer` implementieren, um unsere Quiz-Applikation zu komplettieren. Hierfür spendieren wir der Datei application.js die Funktion `showResults()`:

```
#application.js
function showResults() {
    document.getElementById('results').style.display = 'block'
}
```

Im QuizController ergänzen wir die fehlende Action `answer`:

```
    def answer = {
        def answer = Answer.get(params.int('id'))
        render answer.correct ? "Hurra, die Antwort ist richtig!" : "Die
Antwort '$answer' ist leider falsch!"
    }
```

Die Applikation ist zu diesem Zeitpunkt fertig und beweist uns, dass selbst in kürzester Zeit und mit wenig aber sehr ausdrucksstarkem Code eine voll funktionsfähige Applikation erstellt werden kann.

Fragen und Antworten

In der verbleibenden Zeit des Vortrags-Slots besteht für die Teilnehmer die Möglichkeit mit Fragen zu Groovy & Grails das weitere Programm selber zu bestimmen.

Mögliche Themen könnten beispielsweise sein (Vorschläge aus dem Publikum bevorzugt!):

- GORM – Grails Object Relational Mapping
- Testgetriebene Entwicklung mit Grails (Mocking, Unit- und Integrationtests, Specifications)
- Oberflächengestaltung mit Plug-Ins und Tag-Bibliotheken
- Vorstellung der Neuerungen in Grails 2.0 (Interaktive Grails Console, Agent Based Reloading, Test-Mix-Ins, Scaffolding UI, Resource Handling…)

Weitere Informationsquellen

- http://www.grails.org – Offizielle Grails Webseite
- http://grails.org/doc/latest – Dokumentation der aktuellen Grails Version
- http://www.grailspodcast.com – Regelmäßige Podcasts zu Grails
- http://groovy.codehaus.org – Offizielle Groovy Webseite
- http://www.groovyblogs.org – Sammlung von Blogs zu Groovy

Kontaktadresse:

Stefan Glase, Christian Metzler
OPITZ CONSULTING Gummersbach GmbH
Kirchstr. 6
D-51647 Gummersbach

Telefon:	+49 (0) 2261-6001 0
Fax:	+49 (0) 2261-6001 4200
E-Mail	stefan.glase@opitz-consulting.com, christian.metzler@opitz-consulting.com
Internet:	www.opitz-consulting.com