Java wird Groovy

Eine Einführung in die neue, dynamische Programmiersprache für das Java-Ökosystem

Christof Vollrath, taobits.net
23C3, 12/2006

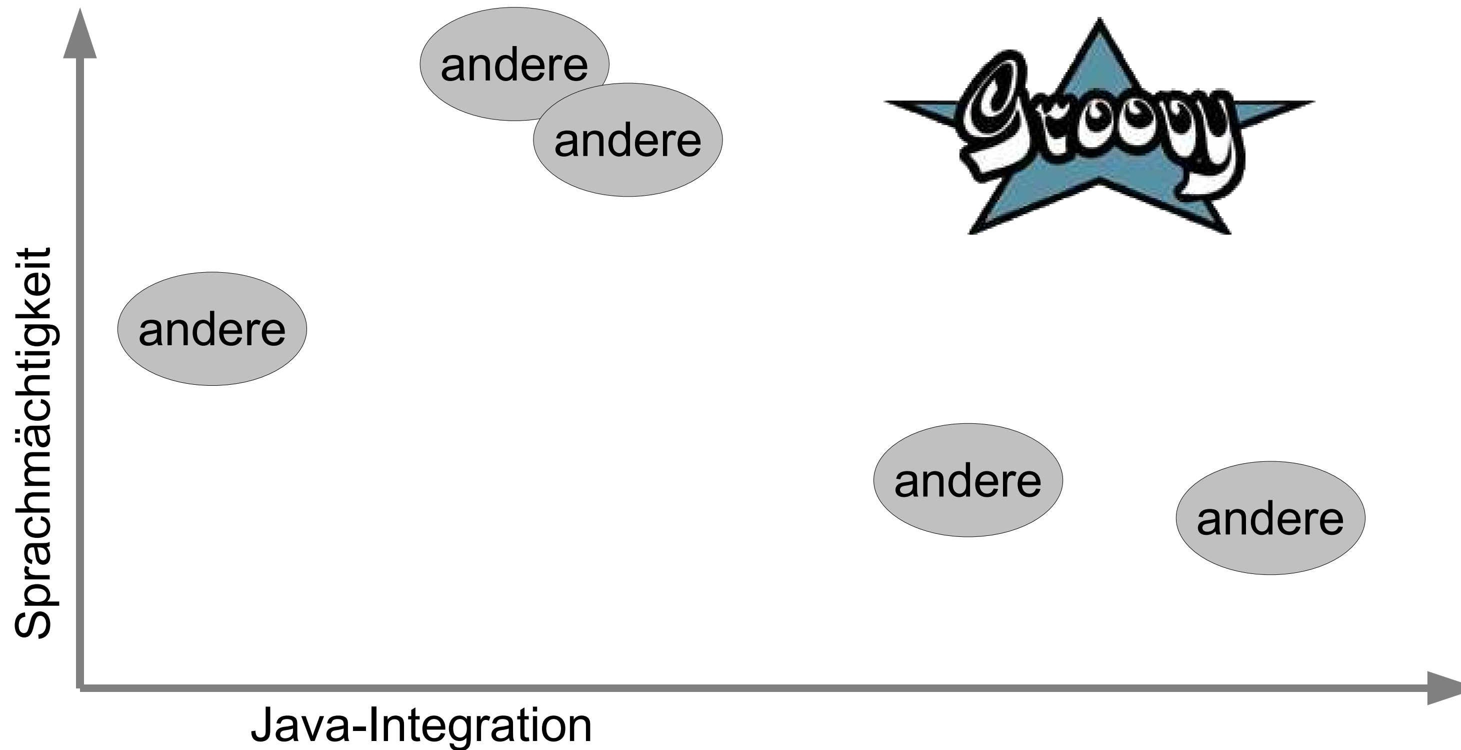
Was ist Groovy?

- Neue, dynamische Sprache für die Java-VM.
- Konzepte von Smalltalk, Python und Ruby
- Hohe Integration mit Java: "Javas bester Freund"
- Open Source-Projekt bei codehaus.org

Javas bester Freund

- Groovy läuft in der Java VM
- Groovy nutzt die Bibliotheken des JDK
- Groovy erweitert das JDK
- Groovy kann problemlos Java-Klassen aufrufen
- Groovy kann aus Java aufgerufen werden
- Groovy kann nach Java Classfiles compiliert werden
- Leichte, risikolose Migration von Java nach Groovy

Groovy in der Sprachlandschaft



aus: Dierk König, Groovy in Action (Manning)

Historie des Groovy Projekts

- Start 2003 durch
  James Strachan und Bob McWriter
  Grundidee: Eleganz von Ruby in Java
- 2004: Beginn der Standardisierung
  mit dem JSR-241
- 2004: GroovyOne
  Treffen von Groovy Entwicklern in London
- Aktuelle Projektleitung: Guillaume LaForge
- Juli 2006: Vorabveröffentlichung von
  "Groovy in Action" (GINA) durch Dierk König
  Vorwort James Gosling!
- Ende 2006: Release 1.0

Verschärfte Syntax (1)

- Semikolon optional
- Klammern bei Methodenaufrufen optional
- Typangaben sind optional
- Deklaration von Exceptions sind optional
- public ist default
- Automatisch importierte Pakete:
  `groovy.lang.*, groovy.util.*, java.lang.*, java.util.*, java.net.*, java.io.*, java.math.BigInteger, BigDecimal`

Verschärfte Syntax (2)

- Alles ist ein Objekt
- Groovy Beans:
  getter- und setter-Methoden werden automatisch erzeugt und beim Attribut-Zugriff automatisch genutzt
- Closures
- Einfache Notation für Listen und HashMaps
- Operator-Overloading
- Erweitertes switch

Seven Ways to Say Hello

```
public class SayHello {
  public static void main(String[] args) {
    System.out.println("hello");
  }
}
```

➡ public ist default

Seven Ways to Say Hello

```
class SayHello {
  static void main(String[] args) {
    System.out.println("hello");
  }
}
```

➞ Argumente und Rückgabewert brauchen keine Typangaben

Seven Ways to Say Hello

```
class SayHello {
  static main(args) {
    System.out.println("hello");
  }
}
```

➜ Jede Klasse kennt die Methode println

Seven Ways to Say Hello

```
class SayHello {
  static main(args) {
    println("hello");
  }
}
```

➡ Semikolon ist (meistens) optional

Seven Ways to Say Hello

```
class SayHello {
  static main(args) {
    println("hello")
  }
}
```

➡ Statt Klasse mit main-Methode echtes Skript

Seven Ways to Say Hello

```
println("hello")
```

➡ Klammern sind optional bei Methodenaufruf

Seven Ways to Say Hello

```
println "hello"
```

Alles ist ein Objekt

- Die primitiven Datentypen aus Java (int, short byte, long, etc.) werden als echte Objekte behandelt:

```
def i = 1
```

- Gleichheit von Objekten wird mit dem Operator == abgefragt (Operator Overloading == entspricht equals)!

```
assert i == new Integer(1)
```

- Identität von Objekten kann mit der Methode is() geprüft werden:

```
def k = i
assert i.is(k)
assert !( i.is(new Integer(1)) )
```

Closure

Closure: Objekt mit ausführbarem Code

```
def c = { println 'hello' }
c()
```

Closure mit Parameter:

```
degree = { rad ->
           return rad / (2*Math.PI) * 360
         }
assert degree(Math.PI) == 180f
```

Oder mit Default-Parameter und ohne return-Anweisung:

```
degree = { it / (2*Math.PI) * 360 }
```

Closure (Anwendungen)

Schleifen mit Closures

```
5.times { println 'hello' }

def sum = 0
1.upto(5) { sum += it }
assert sum == 15
```

Closure (Scope)

Closure arbeitet mit statischer Variablenbindung für lokale Variablen und Felder:

```
class ClosureFactory {
  private field = 'field'

  def create(par) {
    def local = 'local'
    return { par+'_'+field+'_'+local }
  }
}

def c = new ClosureFactory().create('par')
assert c() == 'par_field_local'
```

Groovy Beans (Definition)

Java

```
public class Person {
  private String firstName;
  private String lastName;
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName(
            String name) {
    this.firstName = name;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName(
            String name) {
    this.lastName = name;
  }
}
```

Groovy

```
class Person {
  String firstName;
  String lastName;
}
```

Groovy Beans (Aufruf)

Java

```
Person p = new Person();
p.setLastname("Laforge");
p.setFirstname("Guillaume");

System.out.println(p.getFirstname() + " "
                   + p.getLastname());
```

Groovy

```
def p = new Person()
p.lastname = 'Laforge'
p.firstname = 'Guillaume'

println p.firstname + ' ' + p.lastname
```

Groovy Beans (Zugriffsmethoden)

```
class Person {
  String firstName
  String lastName

  String nickName
  String getNickName() {
    if (!nickName) return firstName
    else return nickName
  }
}
```

```
def p = new Person(firstName: 'Guillaume', lastName: 'LaForge')
assert p.firstName == 'Guillaume'
assert p.nickName  == 'Guillaume'

p = new Person(firstName: 'Jochen', lastName: 'Theodorou',
               nickName: 'blackdrag')
assert p.firstName == 'Jochen'
assert p.nickName  == 'blackdrag'
```

Groovy Beans (Zugriff auf Java-Klassen)

Java

```
Integer i = new Integer(1);


// Zugriff auf Klassenname

String name =
   i.getClass().getSimpleName();
assert name.equals("Integer");


// Auflisten der Methoden

Method[] methods =
   i.getClass().getMethods();
for (Method m: methods)
  System.out.println(m)
```

Groovy

```
def i = 1


def name = i.class.simpleName
assert name == 'Integer'


println i.class.methods
```

Range

Range: Wertebereich von Zahlen als eigener Datentyp

Beispiel: `1..10`
Exklusive obere Grenze: `1..<10`
Umgekehrter Bereich: `5..-5`

```
assert 5 in 1..10
assert 10 in 1..10
assert !(10 in 1..<10)
assert 1.5 in 1.0..2.0
```

Range (Anwendung)

```
// Schleife ueber Range
def sum = 0
for (i in 1..10) sum += i
assert sum == 55

// Schleife mit Closure
sum = 0
(1..10).each { sum += it }
assert sum == 55

// Range als Index
def substr = 'Javas bester Freund'[0..3]
assert substr == 'Java'
```

Listen

```
// Direkte Notation für Listen

def list = [1, 2, 3]

// Listen sind vom Typ ArrayList
assert list instanceof ArrayList


// Iterator über List in for-Schleife
def sum = 0
for (i in list) sum += i
assert sum == 6
```

Listen (Operatoren)

```
// Index auf Listen
def list = [1, 2, 3]
assert list[0]  == 1
assert list[-1] == 3
assert list[1..2] == [2, 3]

// Anhängen von Listen, Elementen
assert list + [4, 5] == [1, 2, 3, 4, 5]
assert list << 9     == [1, 2, 3, 9]

// auch:
assert [1, 2, 3] - [2] == [1, 3]
assert [1, 2] * 2      == [1, 2, 1, 2]
```

Listen und Closures

```
// Schleife mit Closure
def list = [1, 2, 3]
def sum = 0
list.each { sum += it }
assert sum == 6

// Injektion
sum = list.inject(0) { i, res ->
    res + i
}
assert sum == 6
```

Maps

Direkte Notation für Maps

```
def map = [a: 1, b: 2, c: 3]

// Maps sind vom Typ HashMap
assert map instanceof HashMap

// Zugriff auf Elemente
assert map.get('a') == 1
assert map['a'] == 1
assert map.a == 1
```

Operator Overloading

Operator Overloading durch fest zugeordnete Methodenamen.

Operator	*Name*	*Methode*
`a + b`	plus	`a.plus(b)`
`a - b`	minus	`a.minus(b)`
`a * b`	star	`a.multiply(b)`
`a / b`	divide	`a.div(b)`
`a % b`	modulo	`a.mod(b)`
`a++, ++a`	increment	`a.next()`
`a--, --a`	decrement	`a.previous()`
`a**b`	power	`a.power(b)`
`...`		

switch-Anweisung

```
switch(v) {
  // Zahlen
  case 1:         println 'Zahl 1'; break;

  // String
  case 'test':    println 'String test'; break;

  // Range
  case 10..15:    println 'In Range 1..15'; break;

  // Auswahl aus Liste
  case [1, '1', 'Eins']: println 'Irgenwie 1'; break;

  // Regular Expression
  case ~/Ein.*/: println 'Fängt mit Ein an'; break;

  default: println 'Default-Zweig'; break
}
```

Groovy Bonbons

- GDK, Groovy JDK: Groovy Erweiterung der Standardklassen des JDK
  z.B. Einheitliche Methode für die Größe von Arrays, Collections, Strings: size()
- GString: String mit dynamischen Ausdrücken
- Regular Expressions
- GPath: Pfadausdrücke auf Objekt-Netze ähnlich XPath
- Builder: kompakte Syntax zur Erzeugung von Datenstrukturen

Groovy Strings

GStrings erlauben Ausdrücke in Strings:

```
def cal = new GregorianCalendar()
println "Monat: ${cal.get(Calendar.MONTH)+1}"
println "Datum: $cal.time"
```

GStrings über mehrere Zeilen:

```
def gs = """Dieser GString
hat zwei Zeilen.
"""
```

Klassische Java Strings (mehrzeilig mit ''' '''):

```
def s = 'Java String'
```

Strings für Regular Expressions (kein Escape für Backslash nötig):

```
def rexp = /(\w *)*/
```

Regular Expressions

Eigene Operatoren für Regular Expressions, spezielle String-Konstanten

```
// Auf vollständiger Treffer prüfen
assert 'Alles nur Worte' ==~  /((\w*) *)*/

// Auf einzelnen Treffer prüfen
assert 'Alles nur Worte' =~  /\w*/

// Alle Treffer ausgeben
def matcher = 'Alles nur Worte' =~  /\w*/
matcher.each { println it }
```

Builder

Builder erlauben es, komplexe Datenstrukturen aufzubauen. (GOF Pattern)

Builder für:

- Groovy-Objekte: NodeBuilder
- XML, HTML: groovy.xml.MarkupBuilder
- Swing: SwingBuilder
- Ant: AntBuilder

Eigene Builder über BilderSupport

XML-Builder (Beispiel)

Builder

```
def b = new
groovy.xml.MarkupBuilder()

b.persons {
  for (n in ['Dierk',
    'Guillaume', 'Andrew']) {
    person {
      firstName(text: n)
    }
  }
}
```

XML

```
<persons>
 <person>
  <firstName text='Dierk' />
 </person>
 <person>
  <firstName text='Guillaume' />
 </person>
 <person>
  <firstName text='Andrew' />
 </person>
</persons>
```

Meta Object Protocol

MOP stammt von Common Lisp

Abfangen von:

- Methodenaufrufen
- Zugriff auf Properties
- Zugriff auf die Metaklasse

```
public interface GroovyObject {
    public Object invokeMethod(String name, Object args);
    public Object getProperty(String property);
    public void setProperty(String property, Object newValue);
    public MetaClass getMetaClass();
    public void setMetaClass(MetaClass metaClass);
}
```

Dynamische Zaubereien

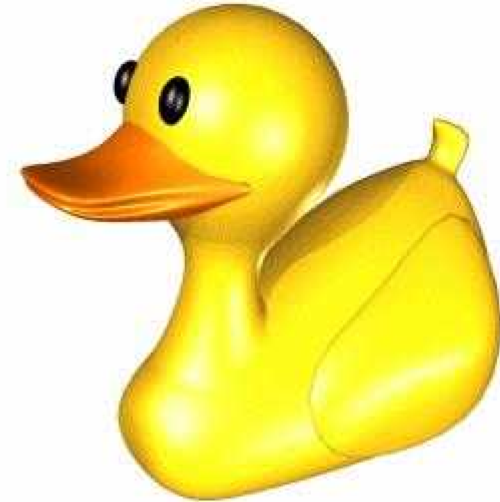
- Attributzugriff auf Elemente einer HashMap
```
def h = [a: 1]
assert h.a == 1
```
- Implementierung eines Builders: Abfangen von Methodenaufrufen
- Expando: dynamisch erweiterbares Objekt

Expando

Dynamische, erweiterbare Objekte

```
def e = new Expando()
e.a = Math.PI
e.m = { println 'Method in Expando' }

e.m()
assert e.a == Math.PI
```

Typsystem



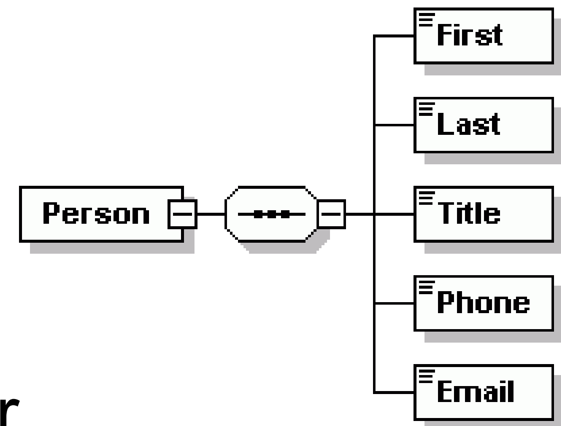
- Strenge, dynamische Typprüfung:
  Typen werden zur Laufzeit geprüft
- Statische Typangaben optional
- Duck Typing:
  Alles was wie eine Ente aussieht,
  watschelt und quakt wie eine Ente,
  ist auch eine Ente.
- Multimethoden: Methodenzuordnung aufgrund dynamischer Typen

Duck Typing

```
class Fish {
  def swim() { println 'I am a fish, I swim' }
}

def fish = new Fish()

def dolphin = new Expando()
dolphin.swim = { println 'I am a swimming dolphin' }

[fish, dolphin].each { it.swim() }
```

Unterstützung für XML

- Einfaches Einlesen durch eingebauten Xml-Parser
- Zugriff auf XML-Daten durch GPath-Ausdrücke
- Erzeugen von XML durch XML-Builder
- Groovy als Alternative zu XSLT



XML-Parser

XML

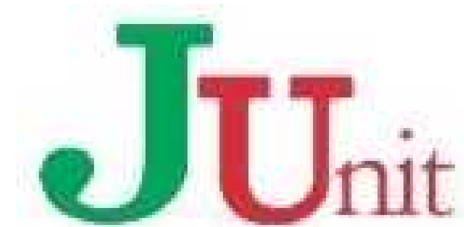
```
<persons>
 <person>
  <firstName
   text='Dierk' />
 </person>
 <person>
  <firstName
   text='Guillaume' />
 </person>
 <person>
  <firstName
   text='Andrew' />
 </person>
</persons>
```

Parser

```
def persons = new XmlParser()
                  .parseText(xmlStr)

// GPath auf das Objekt-Netz
assert persons.person.firstName.'@text'
    == ['Dierk', 'Guillaume', 'Andrew']
```

Unterstützung für JUnit

- Integration von JUnit in Groovy:
  Tests lassen sich direkt von der
  Kommandozeile starten
  oder in Ant integrieren



- Groovy ist durch die höhere Mächtigkeit und die gute Integration zu Java ideal zum Schreiben von JUnit-Tests für Java-Programme
- Guter Einstieg zur Nutzung von Groovy

Unterstützung von Datenbanken

- SQL in Groovy:
  Klasse groovy.sql.SQL

  Einfacher Zugriff auf ResultSet
  durch MOP

- ORM (Object relational mapping): GORM

  Persistenz von Groovy-Objekten durch Hibernate,
  Bestandteil des Grails-Projekt

SQL Beispiel

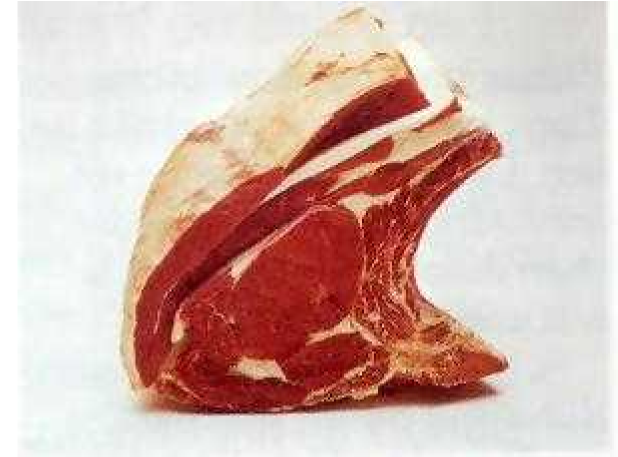
```
// Öffnen der Datenbankverbindung (HSQL)
def sql = Sql.newInstance(
  'jdbc:hsqldb:mem:test',
  'sa', '', 'org.hsqldb.jdbcDriver')

// Datenbankabfrage
sql.eachRow('SELECT * from person') {
  println it.firstName + ' ' + it.lastName
}

// Datenbankabfrage mit Parametern und Ergebnisliste
def rows = sql.rows(
    "SELECT lastName FROM person WHERE firstName = ?",
    ['Dierk'])
assert rows.size() > 0
assert rows[0].lastName == 'Koenig'
```

Web-Anwendungen

- Groovlet: Groovy Servlet
  Aufruf von Groovy-Skripten in Servlet-Engine
  Erzeugung von HTML durch MarkupBuilder

- TemplateBuilder:
  Einfache Template-Engine ähnlich JSP zur Erzeugung von Textdokumenten
- GvTags: Tag-Bibliothek für JSP und TemplateEngine
  www.gvtags.org
- GRAILS: Groovy-Projekt inspiriert von Ruby on Rails
  grails.codehaus.org

Fleisch und Knochen



Aufteilung von Systemen in

- Groovy:
  Dynamische, flexible Komponenten,
  hohe Anpaßbarkeit
  (Fleisch)
- Java:
  Statische, stabile Komponenten,
  Laufzeit-kritisch
  (Knochen)

DSL (Domain Specific Language)

- Operator Overloading
- Meta Object Protocol (MOP)
- Builder

Beispiel: AntBuilder

```
TARGET_DIR = 'target'
CHAPTERS_DIR = 'chapters'

ant = new AntBuilder()

ant.delete(dir:TARGET_DIR)
ant.copy(todir:TARGET_DIR){
      fileset(dir:CHAPTERS_DIR, includes:'*.doc',
                excludes:'~*')
}
```

Der lange Marsch zu Release 1.0

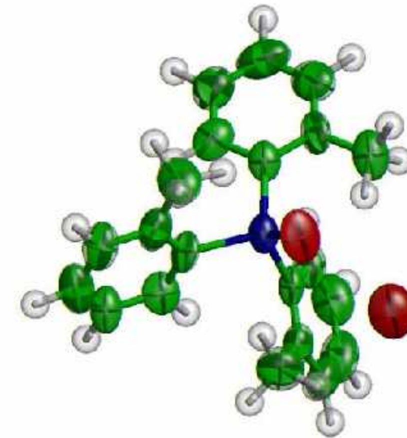
- Guter Start in 2003 mit genialen Ideen
- Immer wieder Veränderungen um neue Ansätze auszuprobieren
  Design-Prinzip: Ist das groovy?
- 2004: Start des JSR-241
- Definition einer formalen Grammatik (ANTLR)
- April 2005: jsr-1 - erste Version basierend auf JSR-241
- Richard Monson-Haefel, Februar 2006:
  "Groovy: The Sleeping Giant"
- @Property-Debakel Mitte 2006 gelöst
- Release 1.0 Ende 2006

Installation und erste Schritte

- Voraussetzung:
  jdk >= 1.4
- Download von Groovy
- Auspacken in ein Verzeichnis
- Environmentvariable GROOVY_HOME auf das Installationsverzeichnis setzen
- Kommandopfad PATH um GROOVY_HOME/bin erweitern

Aufruf eines Groovyskripts:	**`groovy <Skriptname>`**
Groovy-Compiler:	**`groovyc <Skriptname>`**
Interaktiver Groovyinterpreter:	**`groovysh`**
GUI-basierter Interpreter (Swing):	**`groovyConsole`**

Literatur und Links



Links:

- Groovy-Projekt:

http://groovy.codehaus.org

Literatur:

Dierk Koenig, Guillaume Laforge, Andrew Glover:
Groovy in Action
(angekündigt für Feb. 2007, PDF-Vorabversion: http://www.manning.com/koenig, Manning Publications)

Kenneth Barclay, John E. Savage:
Groovy Programming. An Introduction for Java Developers
(angekündigt für Jan. 2007, Elsevier Books)

Deutsches Buchprojekt:
Joachim Baumgart (dpunkt-Verlag)
http://www.groovybuch.de