Scala - A Scalable Language

Sven Pfleiderer

HdM Stuttgart, Medieninformatik

18. Mai 2010

Gliederung

Gliederung

Geschichte

- Entwickelt von Martin Odersky
- Erste Version: 2003
- Wahrnehmbare Verbreitung seit 2006 mit Version 2.0
- Beeinflusst von Java, Pizza, Haskell, Erlang, ML, Smalltalk, Scheme
- Aktuelle Versionen: 2.7.7, 2.8 RC2

Konzepte

- Hybridsprache die OOP und funktionale Programmierung kombiniert
- Sowohl kompiliert als auch interpretiert ausführbar
- Kleiner Sprachkern
- APIs/DSLs, die sich wie native Syntax verhalten
- Interoperabel mit Java
- Type Inference
- Ausdrucksstarke Sprachkonstrukte -> Weniger Code
- “Statically typed dynamic language”

Gliederung

Gemeinsamkeiten

- Entwickelt für Virtuelle Maschinen
- Lauffähig in der Java-VM
- Statisch typisiert
- Benutzt Java Klassenbibliothek
- Benutzt Java Datentypen

Unterschiede

- Syntax
- Funktionale Sprache
- “Alles ist ein Objekt”
- Keine statischen Methoden oder Variablen
- Methoden und Variablen im selben Namespace
- Keine Operatoren sondern Methodenaufrufe
- Traits statt Interfaces
- Dateinamen müssen nicht mit Klassennamen übereinstimmen

Gliederung

Hallo Welt

Listing 1: HelloWorld.scala

```
1 object HelloWorld { // Singleton Object
2     def main(args: Array[String]) {
3         var i: Int = 3;
4         i = i + 2
5         var x: Int = 1.+(3)
6         println("Hello, world!")
7         println("i="+i+" ,x="+x)
8     }
9 }
```

Variablen und Wertzuweisung

Listing 2: Vars.scala

```
1  var x = 5
2  x = x + 1
3
4  var y: Int = 42
5
6  val foo = "Infered String"
7  foo = "New String" //won't compile
8
9  lazy val myPair: Pair[Int, String] =
10                 new Pair[Int, String](1, "scala")
11
12 val shortPair = new Pair(1, "scala")
```

Klassen und Methodendefinition

Listing 3: Class.scala

```
1                       // parameters private
2  class Item(val name: String, price: Double) {
3      private var iquantity = 0
4
5      def quantity: Int = iquantity
6      def otherMethod: String = { "Hello Method" }
7      def doNothing: Unit = { /* Nothing */ }
8
9      def add(toAdd: Int) = {
10         iquantity = iquantity + toAdd
11         // implicit return the current quantity
12         quantity
13     }
14 }
```

Gliederung

Allgemeines

- Parametrisierte Klassen
- “Mix-in” von Traits
- Erweitertes Paketsystem
- public/private/protected Members
- public ist Standard
- Kein "package local"

Vererbung in Scala

Listing 4: Inheritance.scala

```
1  class Person(val firstName:String,
2    val lastName:String, val age:Int) {
3      override def toString = "... "+firstName+" ..."
4      def doSomething = {} // wait ... what?
5  }
6
7  class Student(firstName:String, lastName:String,
8    age:Int) extends Person(firstName, lastName, age)
9  {
10   override def doSomething = {
11       System.out.println("I'm studying hard.")
12   }
13 }
```

Verwendung von Traits

Listing 5: Traits.scala

```
1  trait Similarity {
2    def isSimilar(x: Any): Boolean
3    def isNotSimilar(x: Any): Boolean = !isSimilar(x)
4  }
5
6  class TestClass extends Similarity {
7    ...
8    def isSimilar ...
9  }
10
11 class TestClass extends OtherClass with Similarity{
12   ...
13 }
```

Singleton Objects

- Alternative zu statischen Members
- Wird gewöhnlich als “Companion-Object” verwendet
- Kein Zugriff auf Instanzvariablen
- Keine Parameter
- Wird eingeleitet durch Schlüsselwort object
- Syntax: object Objectname { def hello = "hello"}

Gliederung

Function Literals

- Funktionen sind selbst Werte und werden als solche behandelt
- Können anderen Funktionen übergeben werden
- Können von anderen Funktionen zurück gegeben werden
- Können Variablen zugewiesen werden
- Syntax: (parameter: String) => { println(parameter) }

Einfache DSL mit Function Literals

Listing 6: DSL.scala

```
1  dont {
2      println("Hello? Can anyone hear me?");
3  }
4
5  dont {
6      println("Yep, 2 really is greater than 1.")
7  } unless (2 > 1)
8
9  dont {
10     println("Done counting to 5!")
11 } until (nextNumber() == 5)
```

Einfache DSL mit Function Literals

Listing 7: Dont.scala

```
1  def dont(code: => Unit) = new DontCommand(code)
2
3  class DontCommand(code: => Unit) {
4      def unless(condition: => Boolean) =
5          if (condition) code
6
7      def until(condition: => Boolean) = {
8          while (!condition) {}
9          code
10     }
11 }
```

Gliederung

Anwendungsgebiete

- Webentwicklung
- Verteilte Systeme
- Mobile Applikationen
- Twitter: Backend Webservices
- Foursquare: Lift Framework
- Java Ersatz?

Gliederung

Referenzen

- http://www.scala-lang.org/
- http://www.scala-lang.org/node/1658
- Programming in Scala ISBN: 0981531601
- http://www.artima.com/scalazine
- http://liftweb.net
- http://lexandera.com/2009/11/dont-in-scala/

Fragen?