

# Scala 2.8.0 – Was gibt's Neues?

Prof. Dr. Oliver Braun  
o.braun@fh-sm.de

Fakultät Informatik  
Fachhochschule Schmalkalden

01. Juli 2010

# Überblick

- ursprünglich nur Collections neu schreiben
- 18 Monate Entwicklungszeit
- parallel viele andere Erneuerungen
  - bessere IDE-Unterstützung
  - neue Tools: scalap, scaladoc2
  - Typ-Inferenz für Typ-Konstrukturen
  - verbesserte Actors
  - Unterstützung für Continuations
  - ...
- Martin Odersky auf den Scala Days 2010 (sinngemäß):

*es wäre eigentlich 3.0 angebracht  
aber 2.8 ist schon announced  
und wird bereits in Büchern etc. referenziert*

# Inhalt

- Named Arguments
- Default Arguments
- Scala 2.8 Collections
- Package-Objekte
- Arrays in Scala 2.8
- Typ-Spezialisierung
- Nested Packages
- Scaladoc 2
- Scala IDE for Eclipse

The screenshot shows the Scala website's announcement for Scala 2.8.0 RC7. The page features the Scala logo at the top, a navigation menu with links for 'About Scala', 'Documentation', 'Code Examples', 'Software', and 'Scala Developers', and a search bar. The main content area is titled 'Scala 2.8.0 RC7' and includes a sub-header 'Created by admin on 2010-09-30. Updated: 2010-09-30, 16:44' and a 'Featured' tag. The text describes the release as a 'new release candidate' and lists several key changes, such as a new inference algorithm, fixed issues with `zescape` and null checks, and updates to `StringBuilder`. A section titled 'The Scala 2.8.0 RC7 distribution' explains that the release is for testing purposes only. A 'What is new?' section highlights the redesigned collection library and the new array implementation. On the right side, there are sections for 'Scala Quick Links' (including download, manuals, API, bug reports, and archives), 'Featured News' (listing the current RC7, a previous RC7, and the 1.0 release), and a 'User login' form with fields for username and password, a 'Log in' button, and links for 'Create new account' and 'Retrieve lost password'.

# Named Arguments

- Parameter in Funktionsdefinitionen haben Namen, z.B.

```
def f(a: Int, b: Boolean) = if (b) a else 0
```

- diese Namen können ab Scala 2.8.0 nicht nur innerhalb der Funktion, sondern auch beim Aufruf der Funktion in der Parameterliste genutzt werden, z.B.

```
f(a = 7, b = false)  
f(b = true, a = 12)  
f(18, b = true)
```

# Named Arguments

(2)

- nicht erlaubt

```
f(b = true, 1) // error: positional after named argument  
f(true, a = 2) // error: parameter specified twice: a
```

- (fast) keine Verwechslung mit Zuweisungen möglich

```
var a = 5  
f(a = a + 1, b = false) // named argument
```

eine Zuweisung der Form `a = a + 1` hat den Typ `Unit`

- bei *by-name*-Argumenten mit dem Ergebnistyp `Unit` kann es Probleme geben  $\rightsquigarrow$  Compiler-Error

# Default Arguments

- bei der Definition einer Funktion können *Default Arguments* angegeben werden, z.B.

```
def f(a: Int, b: Boolean = true) =  
    if (b) a else 0
```

- damit kann das Argument beim Aufruf weggelassen werden

```
f(7)  
f(a = 12)
```

# Default Arguments

(2)

- Named und Default Arguments zusammen machen einfache Überladung obsolet

```
class Rational(val numer: Int = 0,  
              val denom: Int = 1)
```

- 4 „Konstruktoren“

```
new Rational() //  $\frac{0}{1}$   
new Rational(2) //  $\frac{2}{1}$   
new Rational(denom = 2) //  $\frac{0}{2}$   
new Rational(5, 3) //  $\frac{5}{3}$ 
```

# Default Arguments

(3)

- Achtung: Default Arguments ersetzen nicht die Implementierung überladener Methoden
- Beispiel:

```
trait MyTrait {  
  def f(i: Int): Double  
  def f(i: Int, d: Double): Double  
}  
class MyClass extends MyTrait { // error  
  def f(i: Int, d: Double = 1) = i * d  
}
```

- führt zu

```
error: class MyClass needs to be abstract,  
      since method f in trait MyTrait of  
      type (i: Int)Double is not defined
```



# copy-Methode für Case-Klassen

Für Case-Klassen wird `copy`-Methode dank `Named` und `Default` Arguments automatisch generiert.

Beispiel:

```
case class Conference(name: String, year: Int)

val jfs2010 = Conference("Java Forum Stuttgart",
                        2010)

val jfs2011 = jfs2010 copy (year = 2011)
                // ~> Conference(Java Forum Stuttgart,2011)
```

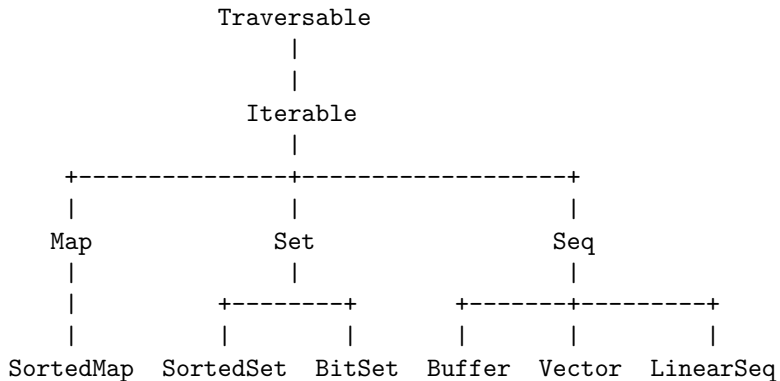
# Scala 2.8 Collections

- mit 2.8.0 Redesign der Scala-Collections-Bibliothek
- alle Collection-Klassen sind in `scala.collection`
- die meisten Basis-Klassen<sup>1</sup> existieren in 3 Formen
  - 1 in `scala.collection.immutable`
  - 2 in `scala.collection.mutable`
  - 3 in `scala.collection`
- einzige Ausnahme ist `Buffer-Trait` (immer mutable)
- die Klassen in `scala.collection` haben das selbe Interface wie die entsprechende in `scala.collection.immutable`
  - ...sind aber nicht garantiert unveränderbar
- die Klassen in `scala.collection.mutable` haben zusätzlich noch Methoden die den Zustand verändern können

---

<sup>1</sup>siehe nächste Folie

# Die wichtigsten Collection-Basisklassen



# Erzeugen einer Instanz

- ... durch Angabe des Klassennamens und der Elemente in Klammern, z.B.

```
Traversable(1, 2, 3)  
Map("B" -> "Berlin", "S" -> "Stuttgart")  
Set("Java", "Forum", "Stuttgart")
```

- genauso für spezifische Implementierungen

```
List(1, 2, 3)  
HashMap(1 -> "one", 2 -> "two")
```

- Repräsentation als Zeichenkette mit `toString` auf die gleiche Weise

# Seq

- Sequenzen sind partielle Funktionen von `Int` zu dem Elementtyp, beginnend bei 0
- Sequenzen definieren eine Methode `apply`<sup>2</sup> zum Indizieren
- Beispiel

```
val list = List("Hallo", "Welt")  
list(1) // ↪ Welt
```

---

<sup>2</sup>Ein Ausdruck der Form `<objectname>(<parameterlist>)` wird in Scala immer expandiert zu `<objectname>.apply(<parameterlist>)`

# Set

- apply ist identisch zur contains-Methode, d.h.

```
val set = Set(1,2,3)
set(2) // ~> true, entspricht set.contains(2)
set(4) // ~> false, entspricht set.contains(4)
```

- Elemente hinzufügen oder entfernen
  - immutable: `Set(1,2) + 3` und `Set(1,2) - 2`
  - mutable: `set += 3` und `set -= 2`
- Argumentliste mit mehreren Argumenten

```
val set2 = set - (3,2) // ~> Set(1)
```

- Mengenoperationen wie Vereinigung (`union, |`), Schnitt (`intersect, &`), Mengendifferenz (`diff, &~`)

# Map

- apply gibt Wert zurück oder wirft Exception

```
map(key) // ↪ value oder Exception
```

- Lookup-Methode get nutzt den Option-Datentyp<sup>3</sup>

```
def get(key): Option[Value]
```

- im Wesentlichen gleiche Methoden wie Set: +, -, +=, -=, ...

---

<sup>3</sup>Option[A] hat die beiden Werte None und Some(x) für ein x: A

- Updates mit verschiedener Syntax möglich<sup>4</sup>

```
map(5) = "five"  
map.update(5, "five")  
map += (5 -> "five")
```

- für immutable Maps

```
map.updated(5, "five")  
map + (5 -> "five")
```

---

<sup>4</sup>Ein Ausdruck der Form

`<objectname>(<parameterlist>) = <expression>`

wird in Scala immer expandiert zu

`<objectname>.update(<parameterlist>, <expression>)`



# Migration zu neuen Collection-Klassen

- die in Scala 2.8.0 neu eingeführten Package-Objekte erleichtern Migration

```
package object scala {  
  type List[+A] =  
    scala.collection.immutable.List[A]  
  val List =  
    scala.collection.immutable.List  
  // ...  
}
```

- das Scala-Package-Objekt muss in `scala/package.scala` gespeichert werden
- die dort definierten Member sind dann Teil des Packages
- damit gibt es neben `scala.collection.immutable.List` auch `scala.List`

# Arrays in Scala

- Spannungsfeld
  - Interoperabilität mit Java / Effizienz von Java-Arrays
- vs.
  - die Vielzahl von Methoden der Scala-Collections soll auch mit Arrays nutzbar sein
- Scala bis 2.7.x nutzt Boxing / Unboxing / Compiler Magic
  - ↪ Probleme und zum Teil schlechte Performanz
- auf Scala-Arrays konnten zwar viele Collection-Methoden angewendet werden, aber das Ergebnis war kein Array mehr
- gleiches Problem mit `String` und `RichString`

# Scala 2.8 Arrays

- entsprechen Java Arrays
- implizite Umwandlung in `ArrayOps` für die Collection-Methoden  $\Rightarrow$  geben Arrays zurück
- modernen VMs können die impliziten Konversionen eliminieren  
 $\rightsquigarrow$  Overhead nahe Null
- zweite implizite Umwandlung in „echte“ `Seq: WrappedArray`
- Auswahl von überladenen Methoden und `Implicits` in 2.8.0 liberalisiert:
  - $\rightsquigarrow$  Priorisierung von `Implicits`
- Analog: `StringOps` und `WrappedString`

# Generische Arrays

- Java erlaubt keine Typparameter im Zusammenhang mit Arrays
- Scala erlaubt aber beispielsweise

```
new Array[T] // für T ist Typparameter
```

⇒ benötigt Runtime-Information über T

- Mechanismus heisst Manifest
- Manifest[T] kann für bekannte Typen vom Compiler generiert werden und wird als impliziter Parameter übergeben
- abgeschwächte Version ClassManifest kann erzeugt werden wenn nur Top-Level-Klasse eines Typen bekannt ist (reicht für Arrays)

# Generische Arrays

(2)

- Beispiel

```
def listToArray[T](list: List[T])
  (implicit m: ClassManifest[T]) = {
  val xs = new Array[T](list.length)
  for (i <- 0 until list.length) xs(i) = list(i)
  xs
}
```

- kürzer mit Context Bound

```
def listToArray[T: ClassManifest]
  (list: List[T]) = { // ...
```

# Generische Arrays

(3)

- Funktion die `listToArray` nutzt und selbst Typparameter hat muss auch `Manifest` bieten:

```
def mkArray[T: ClassManifest](x: T*) =  
  listToArray(x.toList)
```

- `GenericArray` ist immer Java-Referenz-Array und braucht daher kein `Manifest`

# Typ-Spezialisierung

- Scals parametrischer Polymorphismus basiert auf Type Erasure
- für primitive Typen heisst das Un-/Boxing
  - Unit, Boolean, Byte, Short, Char, Int, Long, Float, Double
  - ⇒ ca. 10 mal langsamer
  - ⇒ Programmierer nehmen keine generischen Collections

- Ausweg: Type Specialization in Scala 2.8

```
class Vector[@specialized A] {  
  def apply(i: Int): A = // ...  
  def map[@specialized(Int, Boolean) B]  
    (f: A => B) = // ...  
}
```

- Compiler erzeugt generische Klasse Vector und spezialisierte für jeden primitiven Typ
- map wird für Int und Boolean spezialisiert



# Nested Packages

- Java hat nur absolute Packages, in Scala können Packages verschachtelt werden
- Pre-2.8:

```
package net.obraun  
import java.util.Scanner
```

schlägt fehl, wenn es ein Sub-Package `net.java` gibt

- 2.8:

```
package net.obraun
```

sucht **nicht** in `net`

- 

```
package net  
package obraun
```

sucht in `net`

# Neues Scaladoc

- moderneres Layout
- Tags, wie in Javadoc
  - @author
  - @param
  - @return
  - ...
- Wiki-Syntax in Sourcecode-Kommentaren
- Makros
 

```
@define <name> <body>
```

 nutzbar als \$name

scala

display packages only

scala hide focus

- Annotation
- AnyRef
- Application
- Array
- Call
- ClassfileAnnotation
- cloneable
- Console
- CountedIterator
- deprecated
- Either
- Enumeration
- Equals
- FallbackArrayBuilding
- Function
- Function1
- Function2
- immutable
- inline
- Left
- LowPriorityImplicits
- MatchError
- Math
- Mutable
- native
- nonline
- None
- NotDefinedError
- NotNull
- Option
- PartialFunction
- Predef
- Product
- Product1
- Product2
- Proxy
- remote
- Responder
- Right
- serializable
- SerialVersionUID
- Some
- specialized
- StaticAnnotation
- Symbol
- throws
- transient
- Tuple1
- Tuple2
- TypeConstraint
- unchecked
- UninitializedError
- volatile

scala.actors hide focus

## Predef

object `Predef` extends `LowPriorityImplicits`

The `Predef` object provides definitions that are accessible in all Scala compilation units without explicit qualification.

linear super types: `LowPriorityImplicits`, `AnyRef`, `Any`

source: `Predef.scala`

Ordering:  Alphabetic  By inheritance

Inherited:  Hide All  Show all

Visibility:  Public  All

Impl:  Concrete  Abstract

### Type Members

```
class $$$[-From, +To] extends (From) => To
class $$$[-From, +To] extends (From) => To
class $$$[From, To] extends (From) => To
class ArrowAssoc[A] extends AnyRef
type Class = Class[T]
type ClassManifest = ClassManifest[T]
class BumImplicit extends AnyRef
A type for which there is always an implicit value.
class EnumLike[A] extends AnyRef
type Function = (A) => B
type Manifest = Manifest[T]
type Map = Map[A, B]
type Pair = (A, B)
type Set = Set[A]
type String = String
type Triple = (A, B, C)
```

### Value Members

```
object $$$ extends AnyRef
object $$$ extends AnyRef
object BumImplicit extends AnyRef
val Map: Map
object Pair extends AnyRef
val Set: Set
object Triple extends AnyRef
implicit def any2ArrowAssoc[A](x: A): ArrowAssoc[A]
implicit def any2EnumLike[A](x: A): EnumLike[A]
```

# Eclipse-Plugin

- gemischte Scala/Java-Projekte
- Syntax Highlighting
- Code-Completion
- Hyperlinks zu Definitionen
- Error Markers
- Debugging
- . . .



Download and contribute to the free, open source Scala IDE for Eclipse

[Help Wiki](#) →

## MAIN FEATURES

**Support for mixed Scala/Java projects**  
Support for mixed Scala/Java projects and any combination of Scala/Java project dependencies, allowing straightforward references from Scala to Java and vice versa.

**Editing**  
A Scala editor with syntax highlighting, code completion, inferred type hints, hyperlinking to definitions, error markers and more.

**Debugging**  
Incremental compilation, application launching with integrated debugger, hyperlinking from stacktraces to Scala source, interactive console.

**Navigation**  
Project and source navigation including Scala support in the Package explorer view with embedded outline, outline view, quick outline, open type, open type hierarchy.

[Copy update site URL to clipboard](#)

Current build is master-2.8.0.RC6

Nightly and alternative builds are available.

### Requirements

- Java Developer Toolkit 1.6
- Eclipse Classic 3.5.2

### Help installing



Getting started with the Scala IDE for Eclipse — 3.27

## CONTRIBUTE **Missing Plug-in**

### Get involved

Accurate bug reporting is very welcome as are contributions of code through fixing bugs or adding features. There are comprehensive instructions for setting up a Scala IDE development environment in the developer documentation.

→ [Developer documentation](#)

→ [Sign-up to the developer mailing list](#)

→ [Report a bug](#)

### Recently fixed bugs

Miles Sabin #100042 "source not found" error  
23 Jun 2010

kohtyuk #3184: Support for continuations in  
Eclipse  
22 Jun 2010

Miles Sabin #3184: Support for continuations in  
Eclipse  
21 Jun 2010

kohtyuk #3184: Support for continuations in  
Eclipse  
21 Jun 2010

## TWITTER follow @ScalaIDE

First working installation of the #Scala IDE for #Eclipse from its new #fYcho#Maven build #0001  
2010/06/18

@hircus The #Scala IDE build is moving to #fYcho#Maven ... Maven-friendly version numbers will drop out of that. # 2010/06/18

The #Scala IDE for #Eclipse for Scala 2.8.0.RC6 available now!

## SCALA IDE BLOG 📄

Miles Sabin **The Scala IDE for Eclipse for Scala 2.8.0.RC6 Available Now!**  
Inching towards 2.8.0 final ... The latest release candidate of Scala 2.8.0, RC6 is been released. To match that, a new build of the Scala IDE for Eclipse is now available for update or download.

**News**  
Note that the release cycles for the main Scala toolchain and the IDE are now less tightly coupled. This means that you can continue to follow the development of the IDE without being forced to switch the version of the Scala toolchain that you're building your projects with separate update sites are available for all the supported toolchain releases, as well as the Scala toolchain trunk.

# Vielen Dank für die Aufmerksamkeit – Fragen?

- 02.12.2010
- Die Sprache Scala
- Die Tools
  - Interpreter / Compiler
  - ...
  - Simple Build Tool
  - ScalaDoc / ScalaCheck / ScalaTest / ...
- Die Frameworks
  - Lift
  - Akka
  - ...

