

Scala (Programmiersprache)

Ein Teaser und allgemeinere Gedanken

Sebastian Eidecker

16. März 2016

*Wer als Werkzeug nur einen Hammer hat,
sieht in jedem Problem einen Nagel.*

— Paul Watzlawick

Worüber reden wir?

IT im Wandel

Worüber reden wir?

IT im Wandel

Herausforderungen

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Worüber reden wir?

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Spannendes

IT im Wandel

Herausforderungen

Software Engineering

Software Engineering

Forderungen an IT

Forderungen an IT

- Stabilität und Resilienz

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businessstreiber

Forderungen an IT

- Stabilität und Resilienz
- Wertbeitrag
- Businessstreiber

— Matthias Magnor – CEO Surface und Contract Logistics

IT im Wandel

Manifeste

Manifeste

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)

Manifeste

- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)

Manifeste

- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Manifeste

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)
- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)
- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Wo stehen wir?

Wo stehen wir im Wettbewerb?

Scala

Management Summary

Scalable Language

Scalable Language

*This means that Scala grows with you. You can play with it by typing **one-line expressions** and observing the results. But you can also rely on it for **large mission critical systems** [...]*

— www.scala-lang.org

Eigenschaften

Eigenschaften

- Objektorientiert

Eigenschaften

- Objektorientiert
- Funktional

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)

Eigenschaften

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default
- Gewohnte Syntax („Java ohne Semikolon“)
- Ausdrucksstark (APIs/DSLs schreiben)
- Jung (2004, Hype 2011)

Versprechen

Versprechen

- Produktivitätssteigerung

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code
- Höheres Abstraktionsniveau

Versprechen

- Produktivitätssteigerung
- Höhere Codequalität
- Mehr Spaß
- durch
- Weniger Code
- Höheres Abstraktionsniveau
- Skalierbarkeit

Scala und die Java-Plattform

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs

Scala und die Java-Plattform

- Java-Bytecode, läuft auf JVM
- Java-Bibliotheken nutzbar
- Bekannte IDEs
- Ähnlicher Paketierungs- und Buildprozess (sbt)

Scala

Ein wenig Code

Eine Java-Klasse

Eine Java-Klasse

```
1 public class Person {  
2     private final String firstName;  
3     private final String lastName;  
4     public Person(String firstName, String lastName) {  
5         this.firstName = firstName;  
6         this.lastName = lastName;  
7     }  
8     public String getFirstName() {  
9         return firstName;  
10    }  
11    public String getLastName() {  
12        return lastName;  
13    }
```

Eine Java-Klasse

```
1  @Override
2  public boolean equals(Object o) {
3      if (this == o) return true;
4      if (o == null || getClass() != o.getClass()) return false;
5      Person person = (Person) o;
6      if (firstName != null ?
7          !firstName.equals(person.firstName) :
8          person.firstName != null) return false;
9      if (lastName != null ?
10         !lastName.equals(person.lastName) :
11         person.lastName != null) return false;
12     return true;
13 }
```

Eine Java-Klasse

```
1  @Override
2  public int hashCode() {
3      int result = firstName != null ? firstName.hashCode() : 0;
4      result =
5          31 * result + (lastName != null ? lastName.hashCode() :
6              0);
7      return result;
8  }
```

Businesslogik?

Dasselbe in Scala

Dasselbe in Scala

```
1 case class Person(firstName:String, lastName:String)
```

Es wird funktional – Quicksort

Es wird funktional – Quicksort

```
1 def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = xs match {  
2   case Nil      => xs  
3   case y :: ys => ys partition (_ <= y) match {  
4     case (l1, l2) => quickSort(l1) ++ (y :: quickSort(l2))  
5   }  
6 }
```


val und var – Immutables

val und var – Immutables

```
1 def addOne(i: Int): Int = { i += 1; i }
```

val und var – Immutables

```
1 def addOne(i: Int): Int = { i += 1; i }
```

Compile-Fehler, da i immutable

Funktionen – Benannte Parameter

Funktionen – Benannte Parameter

1

class

Klassen und Objekte

Klassen und Objekte

1

class

Listen

Listen

1

class

Pattern Matching

Pattern Matching

1

class

Type Inference

Type Inference

1 **def** f() = 3 * 2

2

3 **def** f() : Int = 3 * 2

Implizites return

Implizites return

```
1 def f() = {  
2     if (something)  
3         "A"  
4     else  
5         "B"  
6 }
```

Implizites return

```
1 def f() = {  
2     if (something)  
3         "A"  
4     else  
5         "B"  
6 }
```

Letzte Anweisung wird zurückgegeben, impliziter Typ String.

Type Inference II

Type Inference II

```
1 def f() = {  
2     if (something)  
3         "A"  
4     else  
5         1  
6 }
```

Erste gemeinsame Oberklasse, zur Not Any

Vererbung und Traits

Vererbung und Traits

1

class

Funktionen funktional – Lambdas schön

Funktionen funktional – Lambdas schön

1

class

Flatmap that shit!

Flatmap that shit!

1

class

Arbeit mit Strings

Arbeit mit Strings

1

class

Tupel

Tupel

1

class

Implicits

Implicits

1

class

??? – Mein heimlicher Star

??? – Mein heimlicher Star

```
1 def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = ???
```

??? – Mein heimlicher Star

```
1 def quickSort[A <% Ordered[A]](xs: List[A]): List[A] = ???
```

Kompilierbar, aber nicht gefährlich.

class

Scala

Spannendes

Akka

Akka

- Scalable real-time transaction processing

Akka

- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen

Akka

- Scalable real-time transaction processing
- Will die aktuellen Probleme lösen
-

ScalaTest

ScalaTest

- Fachlich verständliche Tests

ScalaTest

- Fachlich verständliche Tests
- Testdatengenerierung

Scalatest – Beispiel

```
1 "Creating a Time" should {  
2     "throw an IllegalArgumentException for hours less than 0 or  
3     greater equal 24" in {  
4         forAll("hours") { (hours: Int) =>  
5             whenever(hours < 0 || hours >= 24) {  
6                 an[IllegalArgumentException] should be thrownBy Time(  
7                     hours)  
8             }  
9         }  
10    }  
11 }
```

Domain Specific Languages

Domain Specific Languages

-

Domain Specific Languages

-

-

Domain Specific Languages

-
-
-

Meine wenig qualifizierte Meinung

Meine wenig qualifizierte Meinung

Vorteile

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven

Meine wenig qualifizierte Meinung

Vorteile

- Für moderne Architekturen
- Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

- Komplex
- Zukunftssicher?
- Anzahl Entwicklungssklaven
- Binärkompatibilität nicht in alle Ewigkeit

*We've found that Scala has enabled us to
deliver things faster with less code. It's
reinvigorated the team.*

— Graham Tackley, Guardian

Mehr für Nerds

Mehr für Nerds

- Sprecht mich an

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse

Mehr für Nerds

- Sprecht mich an
- Hands on-Termin bei Interesse
- Heiko Seeberger: „Durchstarten mit Scala. Tutorial für Einsteiger (2. Aufl.)“

