Scala

让Java平台上的编程重现生机

Sparkle

大纲

- Scala是什么
- 为什么选择Scala
- 语法特色，与Java对比
- 缺点
- 实际使用

Scala是什么

"Which Programming Language would you use *now* on top of JVM, except Java?". The answer was surprisingly fast and very clear: - Scala.

http://www.adam-bien.com/roller/abien/entry/java_net_javaone_which_programming

I can honestly say if someone had shown me theProgramming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.

http://macstrac.blogspot.com/2009/04/scala-as-long-term-replacement-for.html

The Scala research group at EPFL is excited to announce that they have won a 5 year European Research Grant of over 2.3 million Euros to tackle the "Popular Parallel Programming" challenge.

http://www.scala-lang.org/node/8579

Scala Creator Launches Typesafe to Commercialize Modern Application Platform for Multicore and Cloud Computing.

Typesafe also named to its board of advisors Java creator James Gosling and Java concurrency expert Doug Lea.

http://www.marketwire.com/press-release/scala-creator-launches-typesafe-commercialize-modern-application-platform-multicore-1513767.htm

TIOBE 2012年4月份

- 45名

TIOBE 2012年4月份

- 45名
- 3月份不在前50名

TIOBE 2012年4月份

- 45名
- 3月份不在前50名
- (Go也不在前50名）

TIOBE 2012年4月份

- 45名
- 3月份不在前50名
- (Go也不在前50名）
- (曾经进入前30名）

Scala实现

- Play 2.0
- Akka(很像Erlang的并发库)
- Apollo(下一代ActiveMQ)
- Spark(Hadoop竞争者)
- Kafka(MQ)
- Apache Camel支持Scala DSL

Scala是……

- 又一个JVM上的语言(编译型）
- 强类型
- “动态的”静态语言
- 面向对象
- 函数式
- 并发

Scala不是……

- 杀手
- 取代JVM
- 突破JVM限制
- Java实现不了的功能

Scala不是……

- 杀手
- 取代JVM
- 突破JVM限制
- Java实现不了的功能

你可以认为Scala是大量语法糖的Java

谁在用

- Twitter
- LinkedIn
- FourSquare
- Tumblr
- Bump
- Xerox、Sony、Siemens

为什么选择Scala

Java语法落后了

JDK 1.0 (January 23, 1996)
JDK 1.1 (February 19, 1997)
J2SE 1.2 (December 8, 1998)
J2SE 1.3 (May 8, 2000)
J2SE 1.4 (February 6, 2002)
J2SE 5.0 (September 30, 2004)
Java SE 6 (December 11, 2006)
Java SE 7 (July 28, 2011)

世间还有这些语言

- Erlang Python Ruby Lua C # PHP C++ Go
- JRuby Groovy Clojure

为什么选择Scala

为什么选择Scala

- JVM平台——技术积累

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化
- 编译成Bytecode——性能保证

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化
- 编译成Bytecode——性能保证
- 函数式和OO并存——无痛切换

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化
- 编译成Bytecode——性能保证
- 函数式和OO并存——无痛切换
- 高级语法——高层抽象

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化
- 编译成Bytecode——性能保证
- 函数式和OO并存——无痛切换
- 高级语法——高层抽象
- 并发——多核的挑战

为什么选择Scala

- JVM平台——技术积累
- 静态语言——工程化
- 编译成Bytecode——性能保证
- 函数式和OO并存——无痛切换
- 高级语法——高层抽象
- 并发——多核的挑战
- 其实Scala的学习难度并不高

语法特色，与Java对比

Hello, world!

```
//Java
class Test {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
//Scala
object Test {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

Hello, world!

```
//Scala
object Test extends Application {
  println("Hello, world!")
}
```

Java Bean

```
//Java
class MyBean {
  private int x, y;
  public int getX() { return x; }
  public void setX(int x) { this.x = x; }
  public int getY() { return y; }
  public void setY(int y) { this.y = y; }
}
//Scala
class MyBean(var x: Int, var y: Int) {}
```

多返回值

```
//Java
class MyResult { //xxx }

public MyResult getResult() {
  //xxx
  return new MyBean(x, y);
}

MyResult result = getResult();
result.getX();
```

多返回值

```
//Java
public Object[] getResult() {
  //xxx
  return new Object[]{ x, y };
}

Object result = getResult();
(String)result[0];
```

多返回值

```
//Scala
def getResult = {
  //xxx
  (x, y, z)
}

val (x, _, z) = getResult
```

模式匹配

```
def matchTest(x: Any) = x match {
  case 1 => "number one"
  case y: Int if y > 0 => y + 1
  case head :: tail => tail
  case Send(a, _) => a
  case ("Send", _, b: String) => b
  case _ => x
}
//强化版switch
//元素提取
```

模式匹配

```
//Java
public boolean equals(Object obj) {
  if (obj instanceof Point) {
    Point point = (Point) obj;
    return x == point.x && y == point.y;
  } else {
    return false;
  }
}
```

模式匹配

```
//Scala
def equals(any: Any) = any match {
  case point: Point => x == point.x && y == point.y
  case _ => false
}
```

函数式

```
//Java 当然你不会这样写
list.filter(new F<Integer, Boolean>() {
  public Boolean f(Integer i) {
    return i % 2 == 0;
  }
});
//你可能经常这样写
executor.execute(new Runnable() {
  public void run() {
    //xxx
  }
});
```

函数式

```
//Scala 三种写法
list.filter(e => e % 2 == 0)

list.filter(_ % 2 == 0)

val f = (e: Int) => e % 2 == 0
list.filter(f)
```

函数式

```
//Scala 这里有三个循环，不一定是好习惯
list.filter(_ % 2 == 0).map(_ / 2).sum
```

匿名函数

```
//Scala
execute {
  println("check")
}
```

```
def execute(runnable: => Unit) {
  executor.execute(new Runnable() {
    def run = runnable
  })
}
```

匿名函数

```
//slf4j
logger.error("some error: {}, {}, {}", new Object[]
{x, y, z});

//Scala msg在访问的时候才会计算!
def error(msg: => String) {
  if(logger.isErrorEnabled) { logger.error(msg) }
}
error("some error: " + x + ", " + y + ", " + z)
```

open class(ruby)

```
class String
  def foo
    "foo"
  end
end
puts "".foo # prints "foo"
```

为什么open class

```
//这不是一个OO行为
StringUtils.isBlank("some str")

//这才是OO
"some str".isBlank

//我们需要打开别人的类添加方法，继承并不能满足
```

为什么open class

```
//junit hamcrest
assertThat(theBiscuit, is(equalTo(myBiscuit)));

//mockito
verify(mockedList).add("one");

//Scala specs2
"Hello world" must startWith("Hello")
//DSL !!
//让第三方框架打开我们的类
```

隐式转换

```
class MyStr(str: String) {
  def isBlank = str.trim.length == 0
}
implicit def myStrW(str: String) =
  new MyStr(str)

println("some str".isBlank)
```

隐式转换

```
class MyStr(str: String) {
  def isBlank = str.trim.length == 0
}
implicit def myStrW(str: String) =
  new MyStr(str)

println("some str".isBlank)

println(new MyStr("some str").isBlank)
```

open class的效果让大家觉得Scala是动态语言，但选择隐式转换来实现，正好证明了Scala是静态语言

Traits

```
//这是一个经典的Java模式
interface IAnimal {}

abstract class Animal implements IAnimal {
   //公共代码
}

class Bird extends Animal implements CanFly {}

//问题来了，CanFly的公共代码写在哪里
```

Traits

```
//Scala
abstract class Animal { xxx }
trait CanFly {
  def fly {
    println("fly")
  }
}
class Bird extends Animal with CanFly {}
```

Traits

```
class Fish { xxx }
trait CanFly {
  def fly {
    println("fly")
  }
}
val flyFish = new Fish with CanFly
```

Traits

```
//按照具体需求装配不同的功能
val order = new Order(customer)
  with MailNotifier
  with ACL
  with Versioned
  with Transactional
```

Duck Typing

```
class Duck {
  def quack = "呱...呱..."
}
def doQuack(d: {def quack: String}) {
  println(d.quack)
}
doQuack(new Duck)
```

Duck Typing

```
class Duck {
  def quack = "呱...呱..."
}
type DuckLike = {
  def quack: String
}
def doQuack(d: DuckLike) {
  println(d.quack)
}
doQuack(new Duck)
//是不是有点像Go语言的感觉?
```

尾递归优化

```
def factorial(n: Int): Int = {
  @tailrec
  def factorialAcc(acc: Int, n: Int): Int = {
    if (n <= 1) acc
    else factorialAcc(n * acc, n - 1)
  }
  factorialAcc(1, n)
}
//Scala会自动优化成循环，@tailrec只是确保发生优化
```

所有都是表达式

```
//Java
int i;
try { i = Integer.parstInt(str);
} catch(NumberFormatException e) {
  i = 0;
}
//Scala
val i = try { Integer.parseInt(str)
} catch {
  case _:NumberFormatException => 0
}
```

XML支持

```
println(<date>{new java.util.Date()}</date>)
```

并发

是什么让我们觉得一个语言是并发的语言

是什么让我们觉得Go和Erlang是并发语言，而C++和Python不是

是什么让我们觉得Go和Erlang是并发语言，而C++和Python不是

- Go内置Channel和Goroutine
- Erlang的一次赋值、轻量级进程

是什么让我们觉得Scala是并发语言

- case class
- 模式匹配
- 大量的immutable类
- Actor
- Akka

Actor

```
//从Erlang学来
val myActor = actor {
  loop { react {
    case "Send" => {
      println("Get")
    }
  }}
}
myActor ! "Send"
```

Actor

```
//增加阻塞的方案
val future = actor !! "msg"
```

```
val reply = actor !? "msg"
```

Akka

更多Erlang的功能

- STM & Transactors
- Fault-tolerance
- Transparent Remoting
- Scala & Java API

Akka

50 million msg/sec on a single machine. Small memory footprint; ~2.7 million actors per GB of heap.

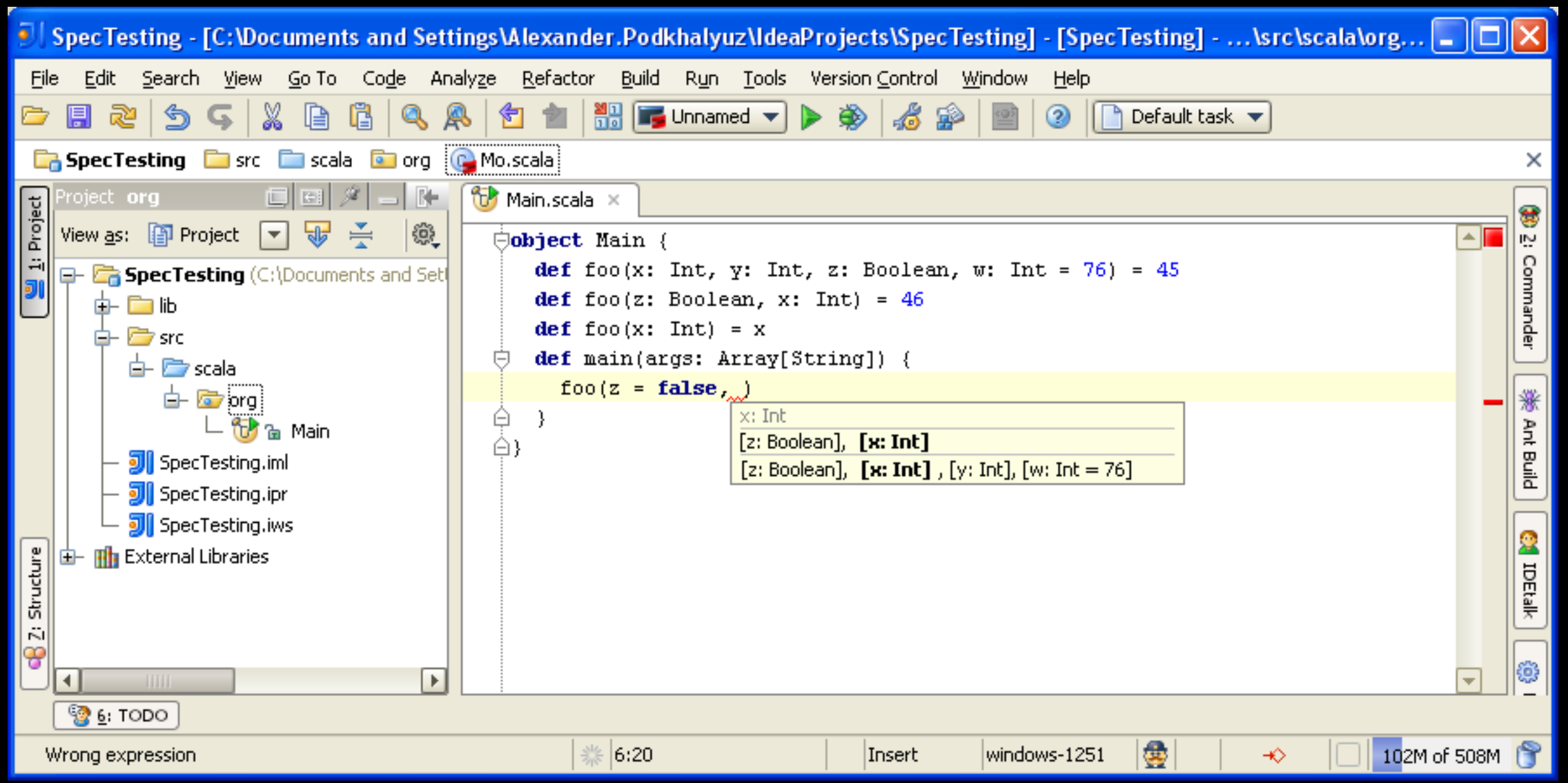
更多改进

- Unchecked exception
- 所有都是对象
- 所有都是方法（包括基础运算符）
- ==的语义正确了
- 多行字符串
- lazy

与Java互相调用

- Scala能调用绝大部分的Java
- 集合转换
  JavaConverters、JavaConversions
- Java调用Scala独有的东西比较困难

IDE支持


SpecTesting - [C:\Documents and Settings\Alexander.Podkhalyuz\IdeaProjects\SpecTesting] - [SpecTesting] - ...\src\scala\org...
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
Unnamed
Default task
SpecTesting src scala org Mo.scala
Project org
View as: Project
SpecTesting (C:\Documents and Set
lib
src
scala
org
Main
SpecTesting.iml
SpecTesting.ipr
SpecTesting.iws
External Libraries
Main.scala
object Main {
def foo(x: Int, y: Int, z: Boolean, w: Int = 76) = 45
def foo(z: Boolean, x: Int) = 46
def foo(x: Int) = x
def main(args: Array[String]) {
foo(z = false,)
}
}
x: Int
[z: Boolean], [x: Int]
[z: Boolean], [x: Int] , [y: Int], [w: Int = 76]
1: Project
2: Commander
Ant Build
IDEtalk
Z: Structure
6: TODO
Wrong expression
6:20
Insert
windows-1251
102M of 508M

缺点

It took about 15 hours to recreate the publishing daemon in Clojure and get it to pass all our tests. Today we ran a "soak test" publishing nearly 300,000 profiles in one run. The Scala code would fail with OoM exceptions if we hit it with 50,000 profiles in one run (sometimes less).

http://groups.google.com/group/clojure/browse_thread/thread/b18f9006c068f0a0

The e-mail confirms that Yammer is moving its basic infrastructure stack from Scala back to Java, owing to issues with complexity and performance.

http://www.infoq.com/news/2011/11/yammer-scala

```
flatMap [B, That] (f: (A) ⇒ Traversable[B])(implicit bf:
CanBuildFrom[List[A], B, That]) : That
```

http://goodstuff.im/yes-virginia-scala-is-hard

jetbrains出了新JVM语言Kotlin后，Scala社区终于开始正视Scala太复杂的问题了

http://groups.google.com/group/scalacn/browse_thread/thread/cbbe5997009db919

缺点

- 编译速度
- 二进制不兼容
- 语法越来越复杂
- 不能突破Bytecode限制
- 太少人使用，招聘和培训

实际使用

适用场景

- 单元测试
- 工具
- Socket开发
- 并发
- 任何Java实现的代码

实际遇到的问题

- 不用sbt会非常痛苦，但有学习成本
- int和Integer的转换问题依然存在
- 编译器启动太慢了
- 混合Java使用比较多问题需要解决
- 集合框架不给力，并且有转换问题
- 高级语法太晦涩了

使用建议

- 你应该首先是一个Java高手
- 不要用高级特性，除非你非常清楚
- 优先使用Java的类库，因为他们更成熟
- 立刻开始使用

开始使用

- 使用稳定版(2.9.1)
- 小工具、单元测试
- Java/Scala混合项目，逐步迁移
- sbt
- idea scala插件

谢谢

- weibo&twitter: sparklezeng
- email: popeast@gmail.com
- website: http://weavesky.com

ArchSummit

中国·深圳 2012.08

INTERNATIONAL ARCHITECT SUMMIT

全 球 架 构 师 峰 会

详情请访问：architectsummit.com

- 3天 • 6场主题演讲
- 3场圆桌论坛 • 9场专题会议
- 国内外30余家IT、互联网公司的50多位来自一线的讲师齐聚一堂

主办方：InfoQ 战略合作伙伴：Tencent 腾讯 特别支持：

http://architectsummit.com


QCon
杭州站 · 2012年10月25日~27日
www.qconhangzhou.com（6月启动）
QCon北京站官方网站和资料下载
www.qconbeijing.com