Scala基础 101

- 介绍
- 语言特点
- 基础入门
- Q&A

由来

- Scala = Scalable(Sca)+Language(la)
- 2001年由Martin Odersky在EPFL开始
- Pizza > Scala
- Scala更有野心

为什么是Scala

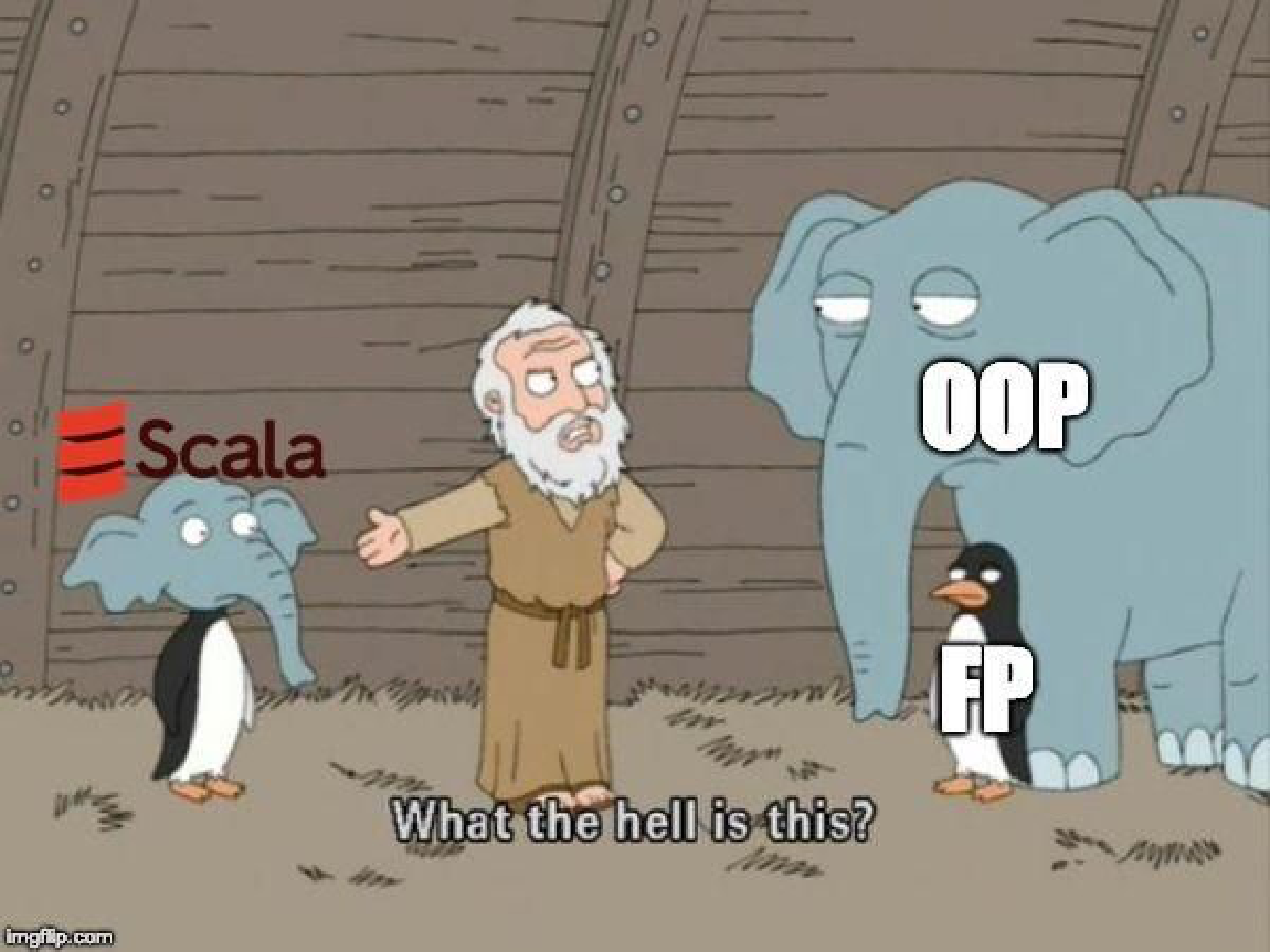
- Scala最终会编译成Java字节码运行在JVM
- Scala可以轻松使用Java生态系统
- Scala非常简洁（分号,getter/setter,return..）
- Scala是高级的，结合OOP和FP
- Scala是静态类型语言，但是看起来很动态

“If I were to pick a language to use today other than Java, it would be Scala”

James Gosling

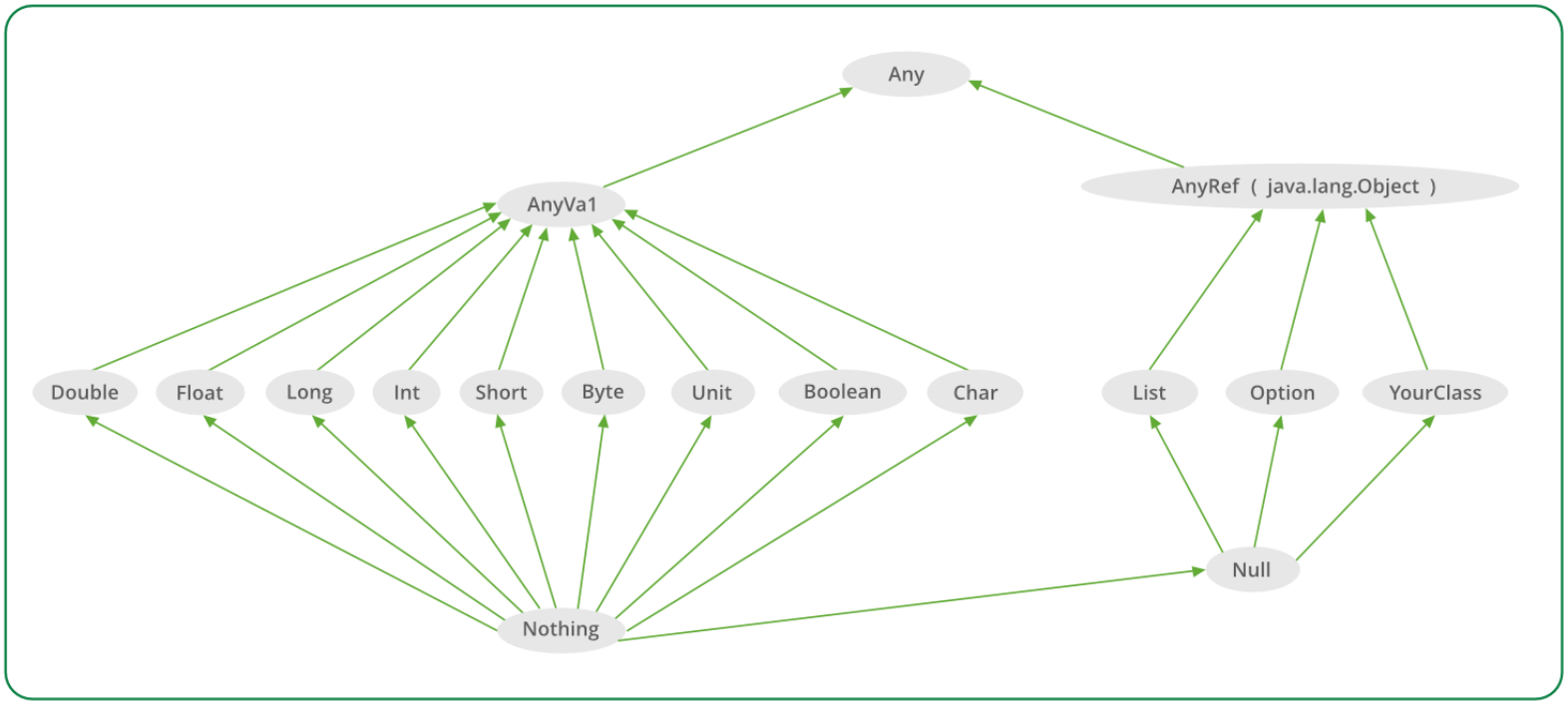
Scala生态与案例

- 大数据（Spark/Flink/Kafka……）
- Web服务端（Play/Lift/Spray……）
- 分布式、高并发、并行（Akka、ZIO）
- 数据科学（Scala NLP……）
- 跨平台（Scala Native/Scala.js/Scala Android）
- 硬件（SpinalHDL）


Scala
OOP
FP
What the hell is this?
imgflip.com

类型系统

- 类型推断
- 编译期暴露潜在异常
- Null/Nil/Nothing/None 拒绝NPE/undefined



特性

- 单例 `object`
- 不可变 `immutable`
- 惰性 `lazy, by-name`
- 样例类和模式匹配 `case class`
- 特质 `trait`
- 高阶函数 `map, flatMap, filter, fold, zip...`
- 标点符号自由
- 并发控制 `future, actor`

Hello Scala

创建 `HelloScala.scala`

```
object HelloScala {
    def main(args: Array[String]): Unit = {
        println("Hello World!")
    }
}
```

编译

```
scalac HelloScala.scala
```

运行

```
scala HelloScala
```

数据类型

Scala的一些内置类型，他们都是对象，注意他不是 基本类型

```
val b: Byte = 1
val x: Int = 1
val l: Long = 1
val s: Short = 1
val d: Double = 2.0
val f: Float = 3.0
```

String and Char

```
// 16-bit unsigned Unicode character (0 to 2^16-1, inclusive)
0 to 65,535
val c: Char = 'a'
// a sequence of `Char`
val name: String = "Bill"
```

> 可以在Scala REPL练习

表达式

> `if` 表达式是有返回值的

```
val a = 1
val b = 2
// 定义Local变量
val minValue = if (a < b) a else b
// 定义函数
def ifThenElseExpression(aBool: Boolean) = if (aBool) 38 else 0
```

> for循环和for表达式

```
val nums = Seq(1,2,3)
for (n <- nums) println(n)
val doubledNums = for (n <- nums) yield n * 2
```

`match` 表达式 -> 模式匹配

```
val monthName = i match {
    case 1  => "January"
    case 2  => "February"
}

def isTrue(a: Any) = a match {
    case 0 | "" => false
    case _ => true
}
```

`try/catch/finally` 表达式

```
try {
    // your scala code here
} catch {
    case foo: FooException => handleFooException(foo)
    case bar: BarException => handleBarException(bar)
    case _: Throwable => println("Got some other kind of Throwable exception")
} finally {
    // your scala code here, such as closing a database connection or file handle
}
```

集合

Scala中有两种类型的集合, 可变的和不可变的

List & Set & Map

```
// List
val numbersList: List[Int] = List(1, 2, 3 ,4)
List(1,2) ::: List(3,4) // List(1, 2, 3, 4)
List(1,2,3,4).reverse // List(4, 3, 2, 1)
// Set
Set(1, 2, 3, 4).head // 1
// Map
val immutableMap = Map(1 -> "a", 2 -> "b")
val mutableMap = collection.mutable.Map(1 -> "a", 2 -> "b")
```

Tuple

```
val t1 = (1, "A")   // val t1 = Tuple2(1, "A")
t1._1   // 1
t1._2   // A
```

类、方法

定义一个Person，构造函数有两个参数，还有两方法

> 不需要显示声明getter/setter等

```
class Person(var firstName: String, val lastName: String) {
    def growing(): Unit = {
        println(s"$firstName is growing")
    }
    def sleeping(h: Int = 8): String = {
        s"I'm sleeping $h hours"
    }
}

val p = new Person("Bill", "Panner")
println(p.firstName)
p.firstName = "Bob" // 可以
p.lastName = "James"    // 不可以，因为是val

p.growing
p.sleeping(9)
p.sleeping()    // 默认是8
```

Trait Mixins

`trait` 很像 `interface`，但是 `Minxin` 起来很强大

```
trait Speaker {
    def speak(): String
}

trait TailWagger {
    def startail(): Unit = println("tail is wagging")
    def stopTail(): Unit = println("tail is stopped")
}

class Dog extends Speaker with TailWagger {
    // Speaker
    def speak(): String = "Woof!"
}

class Cat extends Speaker {
    def speak() = "Meowing"
}

val dog = new Dog
val cat = new Cat with TailWagger
val cat2 = new Cat
```

Be Functional

如果一首歌既不是新歌又不会过时，那它就是民谣。

> 高阶函数 - 函数作为参数或返回值

```
val meat = Seq("pork", "beef", "chicken")
val map = meat.map(name => s"great $name")
val filter = meat.filter(_ == "pork")
val fold = meat.foldLeft(0)((acc, x) => acc + 1)

def mathOperation(name: String): (Int, Int) => Int = (x: Int, y: Int) => {
  name match {
    case "addition" => x + y
    case "multiplication" => x * y
    case "division" => x/y
    case "subtraction" => x - y
  }
}

def add: (Int, Int) => Int = mathOperation("addition")
def mul: (Int, Int) => Int = mathOperation("multiplication")
def div: (Int, Int) => Int = mathOperation("division")
def sub: (Int, Int) => Int = mathOperation("subtraction")
```

闭包

```
var factor = 3
val multiplier = (i:Int) => i * factor
scala> multiplier(10)
scala> res: Int = 100
```

Curring（柯里化/咖喱） & Partial Application（部分应用）

```
val sum: (Int, Int) => Int = (x, y) => x + y
// function def
val curriedSum: Int => Int => Int = x => y => x + y


sum(1,2) shouldBe 3
curriedSum(1)(2) shouldBe 3

// partial
def curriedSum(x: Int)(y: Int): Int = x + y
val increment: Int => Int = curriedSum(1)
val inc = curriedSum(1) _
scala> inc(2)
```

标点符号自由 - Infix Notation

```
val x = 1
val y = 2
scala> x + y
// `+`是一个函数名称
scala> x.+(y)
// 定义在这里
final abstract class Int() extends scala.AnyVal {
    def +(x : scala.Int) : scala.Int
    //...
}
```

不可变数据类型 - 语义拷贝/流畅

```
val update = device
    .filter(_.hierarchyId === hierarchyId)
    .map(d => (d.name))
    .update(name)
```

Enjoy scala! ✌