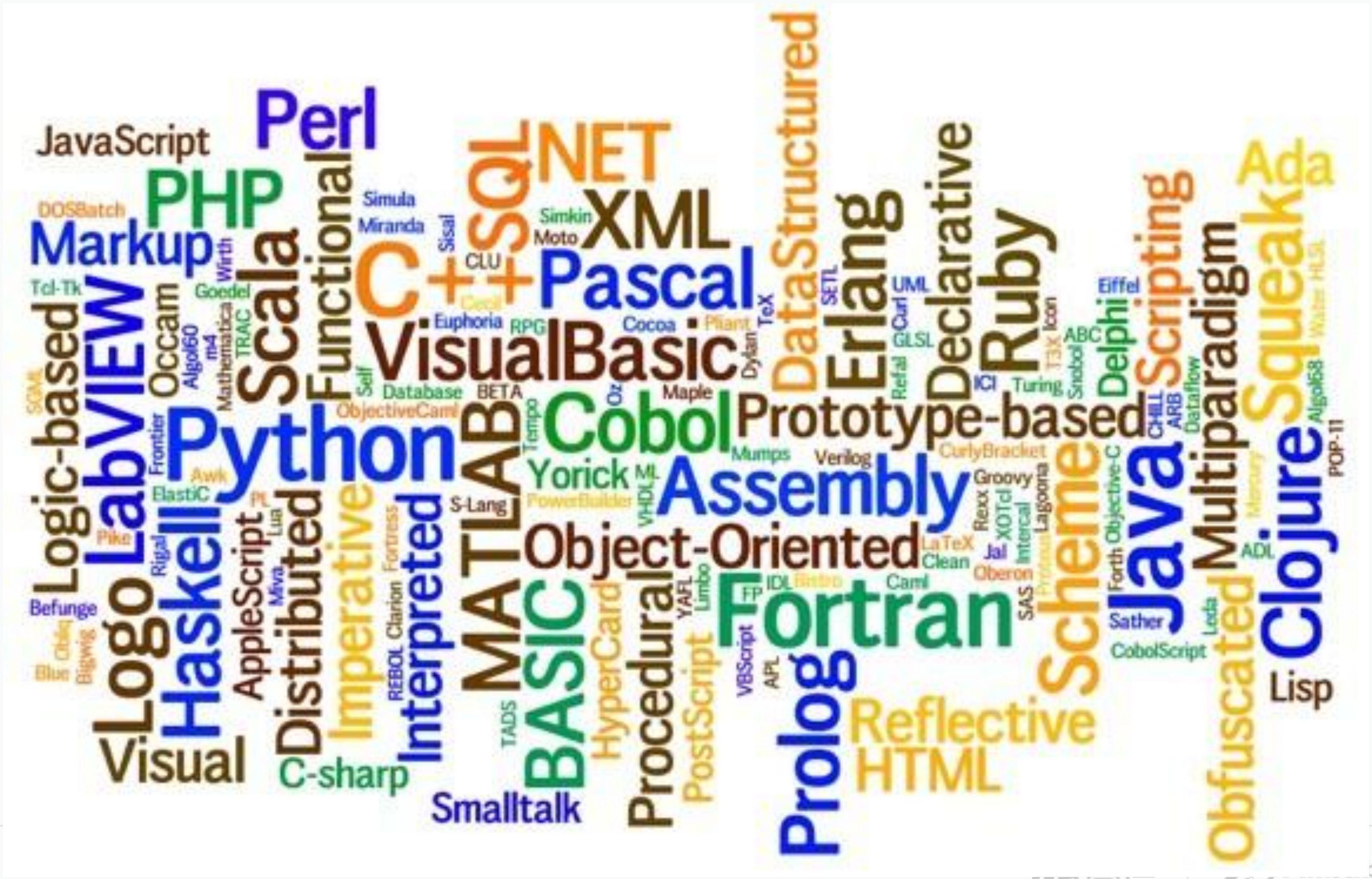

JavaScript
Perl
.NET
SQL
PHP
XML
DOSBatch
Markup
C++
Pascal
Simula
Miranda
Sisal
Simkin
Moto
CLU
DataStructured
Erlang
Declarative
Ruby
Scripting
Multiparadigm
Squeak
Ada
Tcl-Tk
Wirth
Goedel
Occam
Algol60
m4
Mathematica
TRAC
Scala
Functional
Euphoria
RPG
Cocoa
Pliant
TeX
SETL
UML
Curl
GLSL
Refal
Icon
ABC
T3X
Snobol
Delphi
Eiffel
Dataflow
Water
HLSL
Algol68
VisualBasic
Dylan
Self
Database
BETA
Maple
ICI
Turing
ObjectiveCaml
Logic-based
LabVIEW
Python
MATLAB
Cobol
Prototype-based
CHILL
ARB
POP-11
Frontier
Tempo
Oz
Mumps
Verilog
CurlyBracket
Awk
ElasticC
Yorick
ML
Assembly
Groovy
Java
Clojure
Mercury
PL
Lua
Pike
S-Lang
PowerBuilder
VHDL
Rexx
XOTcl
Intercal
Lagoona
Objective-C
Forth
ADL
Befunge
Logo
Haskell
Rigal
AppleScript
Miva
Distributed
Imperative
Fortress
REBOL
Clarion
Interpreted
Object-Oriented
LaTeX
Clean
Jal
Oberon
Protocol
Scheme
Sather
Lisp
Obfuscated
Blue
Cilk
Bigwig
IDL
Bistro
Caml
FP
Limbo
YAFL
SAS
Fortran
Lexico
Leda
CobolScript
BASIC
HyperCard
Procedural
PostScript
VBScript
APL
Prolog
TADS
Visual
C-sharp
Smalltalk
Reflective
HTML

01

Scala Intro

xiaofeng.cui@autonavi.com

提纲

- 函数式/命令式
- Scala简介/特点
- Scala 语法
  - 变量/表达式/数据结构
  - **函数**
    - **匿名函数**
    - **闭包**
    - **柯里化**
  - 模式匹配
  - Option
  - 类
- 小结

命令式/函数式

命令式

- 执行命令序列
- How

函数式

- 函数一等公民
- What

命令式/函数式

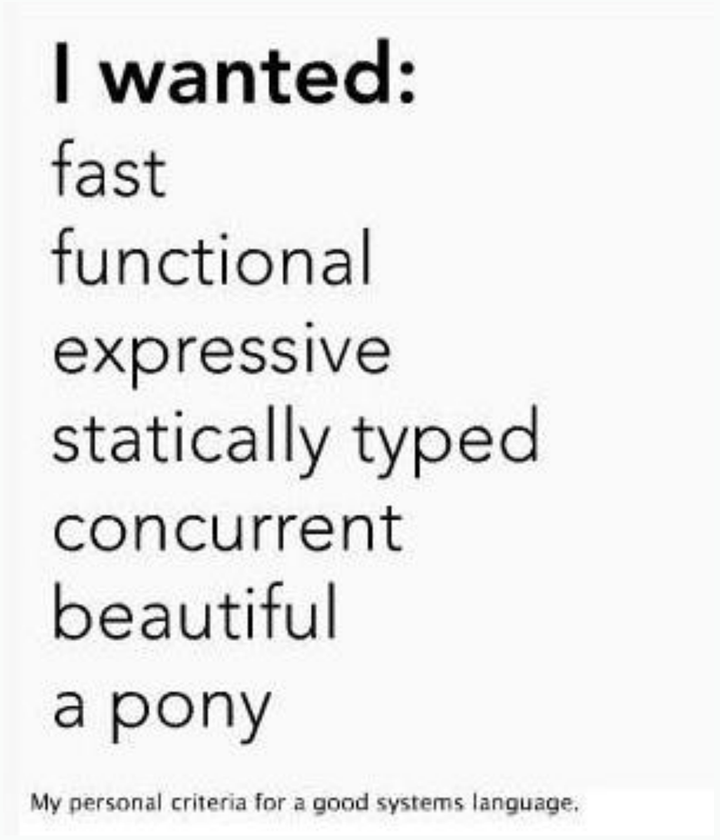
- 函数式编程 WordCount

```
   val file = List("warn 2013 msg 2014", "warn 2013 msg 2013", "error 2013
msg");
   var count = 0;
   for (item <- file) {
     val tmp = item.split(" ");
     for(str <- tmp) {
       if (str == "2013") count = count + 1;
     }
   }
```

```
val file = List("warn 2013 msg", "warn 2012 msg", "error 2013 msg")

val wordNum = file.map(_.split(" ").count("2013" == _)).reduceLeft(_ + _)
```

Scala



I wanted:

fast

functional

expressive

statically typed

concurrent

beautiful

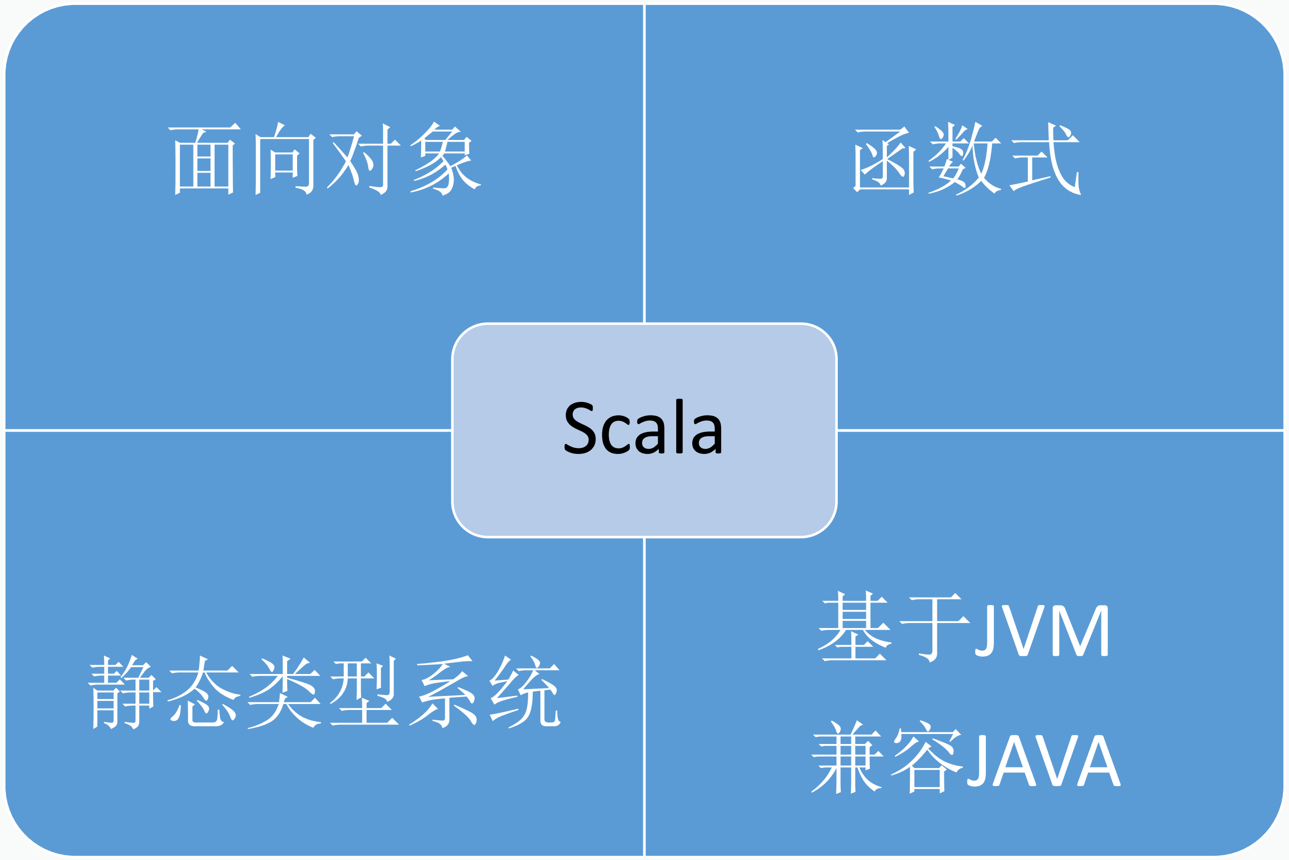
a pony

My personal criteria for a good systems language.

Scala (Unifier)

$$\begin{tikzcd}
\text{Lisp} \arrow[d, "\text{functional programming}"] & \text{C} \arrow[ddr, "\text{syntax}"] & & \text{Simula} \arrow[d, "\text{objects}"] & \text{Prolog} \arrow[dd, "\text{pattern matching}"] \\
\text{ML} \arrow[d] & & & \text{Smalltalk} \arrow[dl] & \\
\text{Haskell} \arrow[ddrr] & & \text{C++} \arrow[d] & & \text{Erlang} \arrow[ddll, "\text{Actors}"'] \\
& & \text{Java} \arrow[d] & & \\
& & \text{Scala} & &
\end{tikzcd}$$

Scala (Unifier)



Scala简介

- 面向对象 + 　函数式
- 兼容Java：
  - 和Java兼容，运行于JVM，可以访问java的类库；
  - import java.util.Date
- 静态类型系统：
  - 可以根据计算的值的类型确定变量和表达式的类型；消除运行时的错误，保证bool不和整数相加等

Scala简介

- **简洁，优雅**
  - Java

```
class Person {
    private String firstName;
    private String lastName;
    private int    age;

    public Person(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName  = lastName;
        this.age   = age;
    }
}
```

  - Scala

```
class Person(var firstName: String, var lastName: String, var age: Int)
```

HelloWorld

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

变量定义

- 常量val，变量var
- 类型推断

```
var num: Int = 123
var num = 123 //类型推断
val num = 123 //常量，类似java的final
var args = new Array[String](3)
```

表达式

- 几乎所有的语言元素都是表达式，都有值
  - If，while，match，= 等

```
val result = if (x > y) x else y
```

  - 块表达式 { }

```
  //val result = (1 + 2) * ( 3 + 4)
  val result = {
    var a = 1 + 2
    var b = 3 + 4
    a * b
  }
```

数据结构

```
Array
     var args1 = Array("1", "2", "3")
     var args = new Array[String](3);
     args(0) = "1"

List
       //List只能包括同类型的元素：
     var list = List(1, 2, 3)
     list = 4 :: list // 将4加到list的头部
     println(list)

Tuple
       //Tuple可以包括不同类型的元素：
     val tuple = (1, "1")

Set
     var set = Set('1', '2')
     set += '3'

Map
     var map1 = Map[String, Int]()
     map1 += ("jj" -> 20 )
     var map = Map("1" -> 1, "2" -> 2)
     map += ("jj" -> 20 )
```

数据结构

- Java初始化集合

```
      ArrayList<String> list = new ArrayList(Arrays.asList("Ryan", "Julie", 
"Bob")) ;

      // 第一层括弧实际是定义了一个内部匿名类 (Anonymous Inner Class)
      // 第二层括弧 实际上是一个实例初始化块 (instance initializer block)，这个块在
内部匿名类构造时被执行。这个块之所以被叫做¡实例初始化块¡是因为它们被定义在了一个类的实例范
围内。
      List list1 = new ArrayList(){
        {
          add("A");
          add("B");
        }
      };

      Map<String, String> hashMap = new HashMap<String, String>(){
        {
          put("A", "a");
          put("B", "b");
        }
      };
```

函数（一等公民）

- 作为参数传递
- 作为返回值
- 可赋值给其他变量

```
def max(x : Int, y : Int) : Int = {
    // x = 6 wrong
    if (x > y) x
    else y
}
```

```
def max(x : Int, y : Int) : Int = if (x > y) x else y //省略大括号
def max(x : Int, y : Int) = if (x > y) x else y //省略返回值,可以推断
```

函数

- 作为参数，返回值
- 函数类型: (输入参数) => 输出参数，（Int, Int） => Int

```
//函数作为参数
def getValue(func: (Int, Int) => Int, x: Int, y: Int) = {
   func(x, y)
}
getValue(max, 1, 2)

// 函数作为返回值
def getFunc(): (Int, Int) => Int = {
  val func = (x: Int, y: Int) => if (x > y) x else y
  func
}
val func = getFunc()
func(1, 2)
```

匿名函数

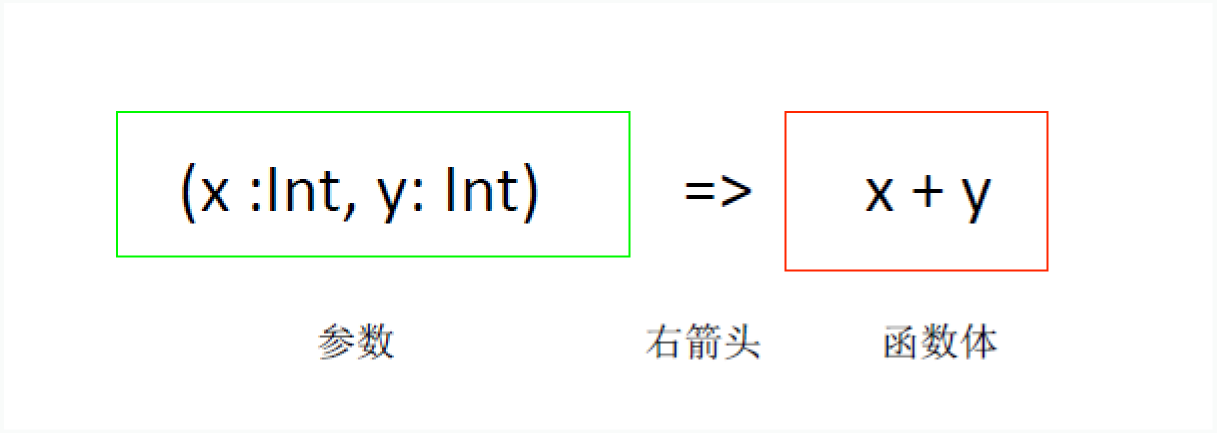
- python lambda

```
list = [1, 2, 3, 4]

print map(lambda x : x + 1, list)
```

匿名函数

- Lambda： 函数字面量



```
val max1 = (x: Int, y: Int) => if (x > y) x else y
println(max1(1, 5))
```

匿名函数

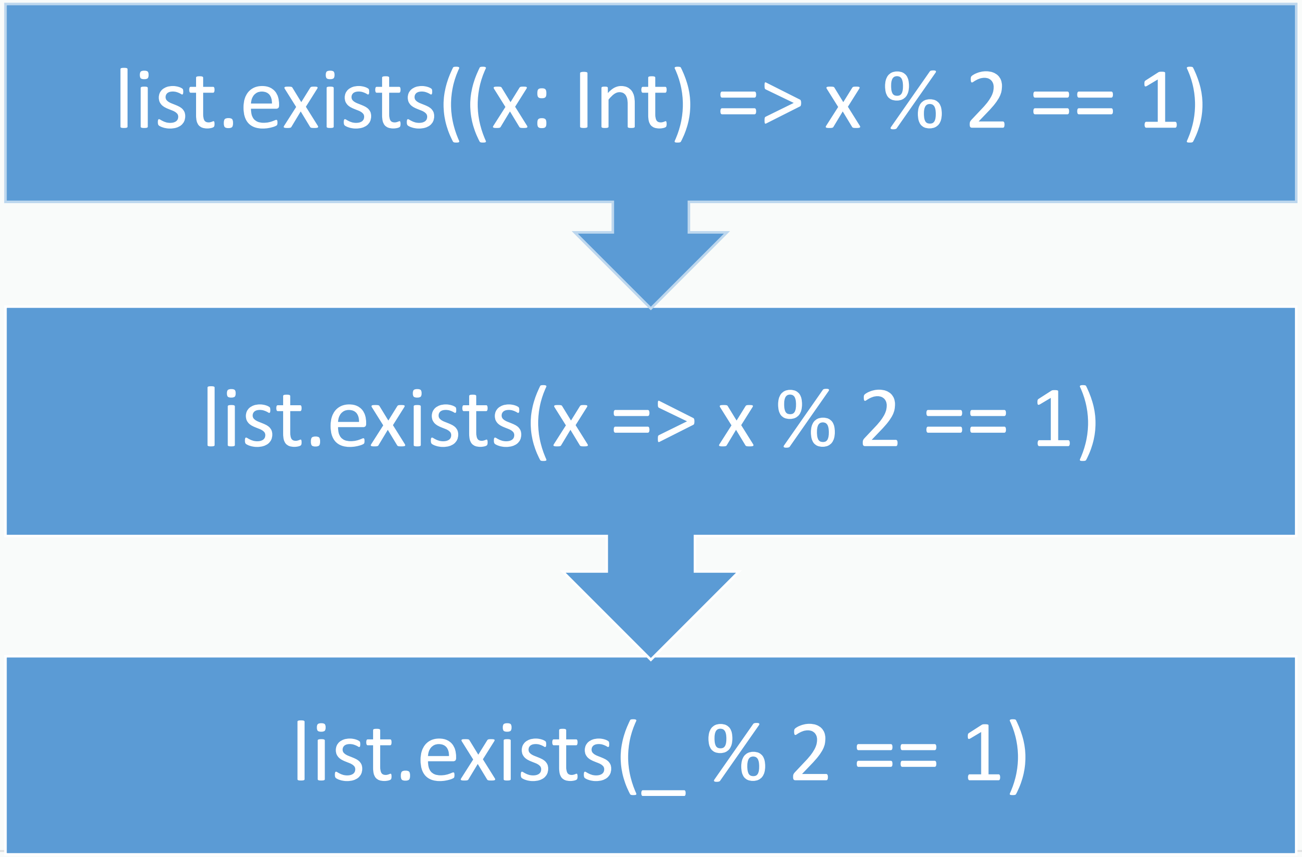
- 简洁，优雅

```
//判断列表中是否包含奇数
val list = List(1, 2, 3, 4)
def containsOdd(list: List[Int]): Boolean = {
  for (i <- list) {
    if (i % 2 == 1)
      return true;
  }
  return false;
}
```

```
list.exists((x: Int) => x % 2 == 1) // 要什么，而不是怎么做
```

匿名函数中神秘的_

- 进一步简化代码



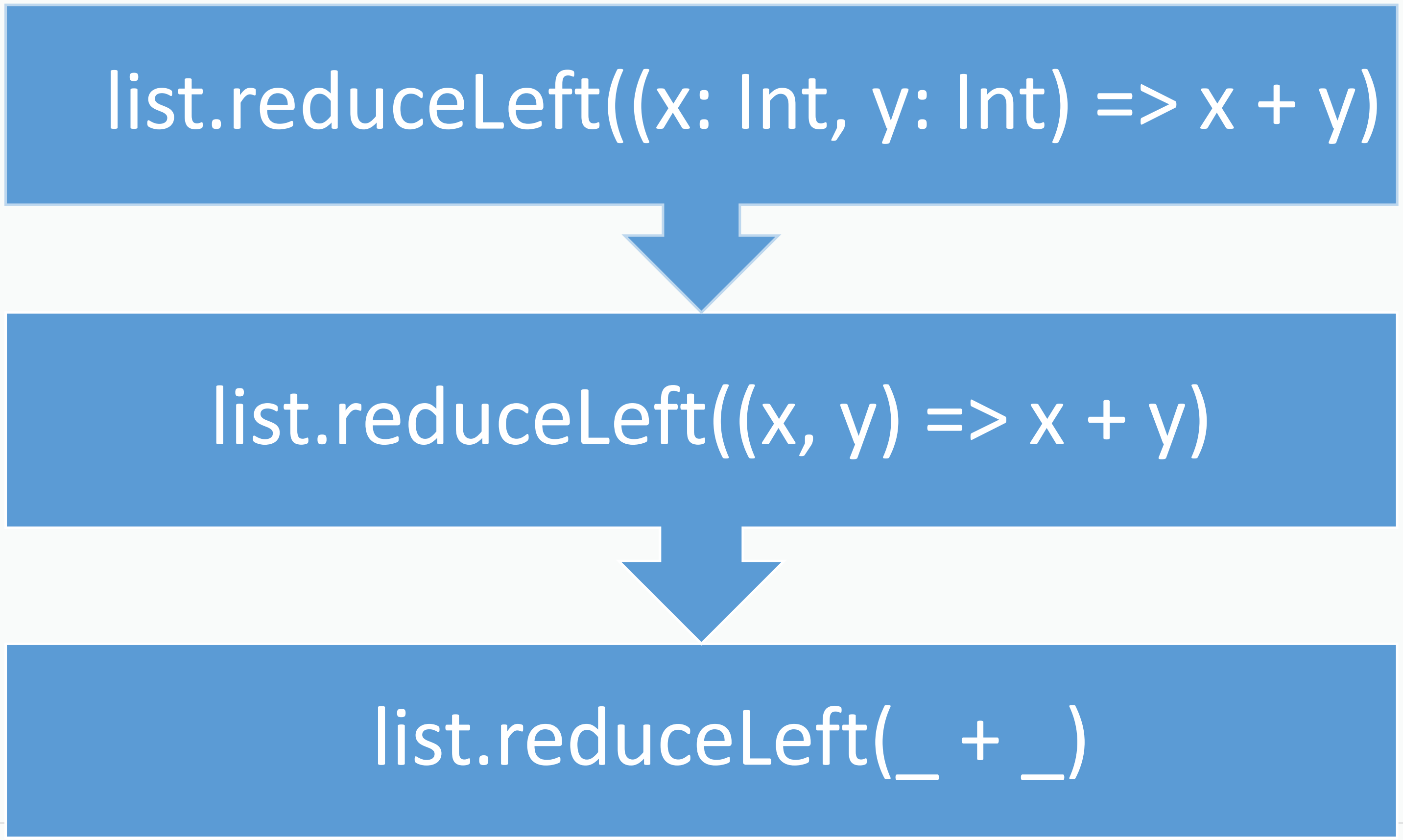
```
list.exists((x: Int) => x % 2 == 1)
```

```
list.exists(x => x % 2 == 1)
```

```
list.exists(_ % 2 == 1)
```

匿名函数中神秘的_

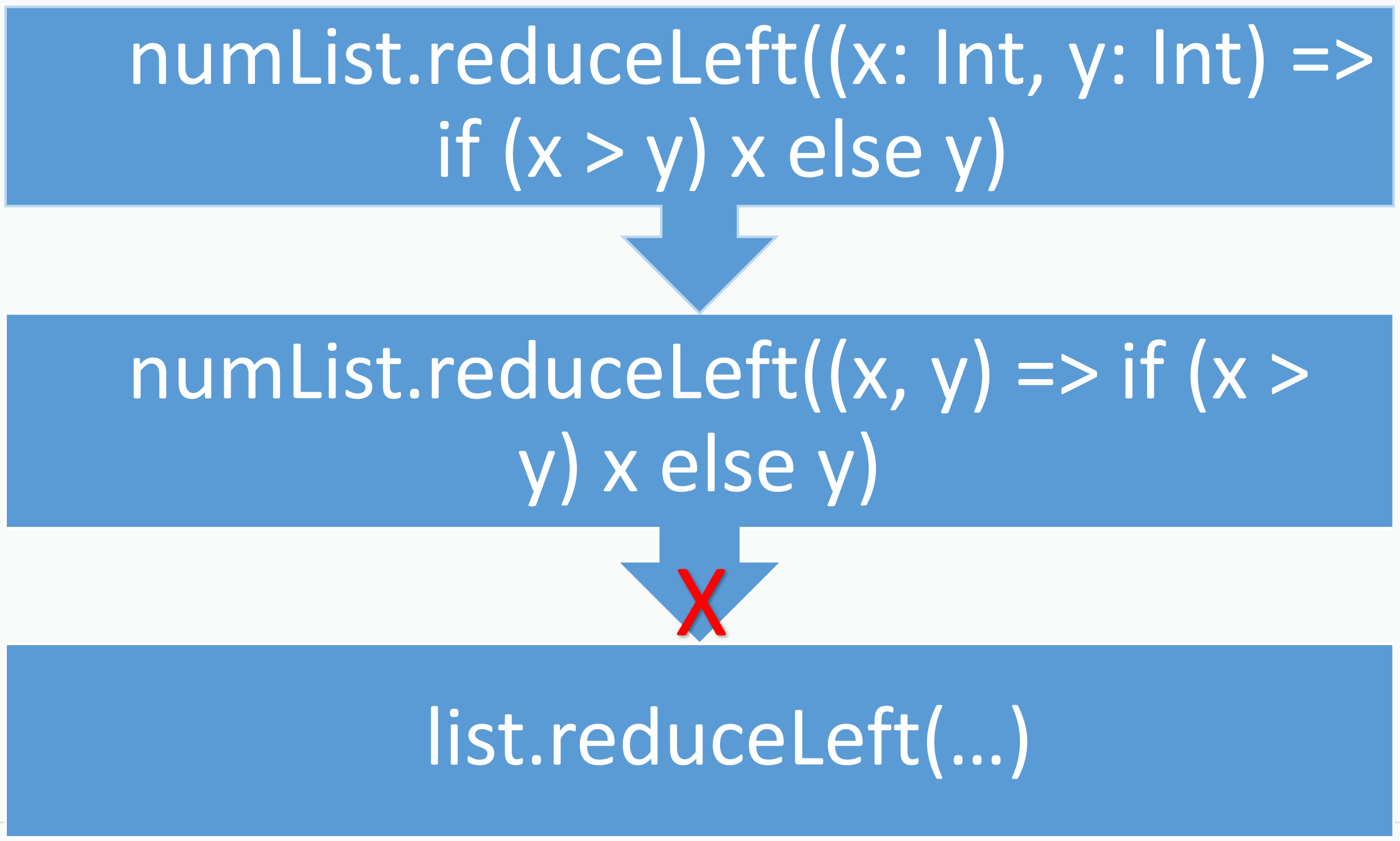
- 复杂一点，求列表中所有元素的和



```
list.reduceLeft((x: Int, y: Int) => x + y)
list.reduceLeft((x, y) => x + y)
list.reduceLeft(_ + _)
```

匿名函数中神秘的_

- Max例子
  -



```
numList.reduceLeft((x: Int, y: Int) => if (x > y) x else y)
```

```
numList.reduceLeft((x, y) => if (x > y) x else y)
```

```
list.reduceLeft(...)
```

闭包(closure)

- 代码块 +上下文，关于引用环境的绑定

```
def increase(more: Int) = {
  (x : Int) => x + more
}
```

```
var inc = increase(100);
println(inc(1))  //101

inc = increase(200);
println(inc(1))  //201
```

柯里化(currying)

柯里化

- 把一个带有多个参数的函数，转换为多个只有一个参数的函数来执行

```
def sum(x:Int, y:Int) = x + y
```

```
def sum(x:Int)(y:Int) = x + y
```

```
sum(1)(2)
```

柯里化

```
def first(x:Int) = (y:Int) => x+y
```

```
var second = first(1)
```

```
var ret = second(2)
```

柯里化

- 控制抽象，改变代码风格

```
var a = 1 + 2
var b = 3 + 4
var c = a * b
sum(1, c);
```

```
var a = 1 + 2
var b = 3 + 4
var c = a * b
sum(1)(c)
```

```
sum1(1) {
  var a = 1 + 2
  var b = 3 + 4
  a * b
}
```

柯里化

- 控制抽象

```
def until(condition: => Boolean)(block: => Unit) {
  if (!condition) {
    block
    until(condition)(block)
  }
}

// 使用
var x = 10
until (x == 0) {
  x -= 1
  println(x)
}
```

柯里化

- 偏函数（部分应用函数）

```
val sum = (a: Int, b: Int, c: Int) => a + b + c
val sum2 = sum(1, _: Int, _: Int)
sum2(2, 3)
```

- C++: bind1st，bind2nd

模式匹配

```
//模式匹配
def activity(day:Any){
  day match {
    case "monday" => println(day)
    case "saturday" => println(day)

    //元组既可以是明确值的  也可是元组是变量形式的
    case ("sunday","friday")=> println(day)
    case (a, b) => println(day)

    case List("sunday","friday") =>  println(day)
    //只要开头是sunday，friday的list都可以匹配
    case List("sunday","friday", _*) =>  println(day)

    //类型匹配
    case a: Int => println(a)
    //类型和guard匹配
    case a: Long if a > 10 => println(a)
    case msg: String => println(msg)
    //元组中的一个元素也可以
    case (a: Int, b: Int) if a > 10 => println(a)

    case _ => //匹配通配符
  }
}
```

Option

- 如何判断Java函数是否返回NULL
  - 依靠 JavaDoc 上的说明
  - 查看那个函式的源码来看
  - 黑盒测试
  - 爆NullpointException才知道

```
HashMap<String, String> myMap = new HashMap<String, String>();
myMap.put("key1", "value1");
String value1 = myMap.get("key1");  // 返回 "value1"
String value2 = myMap.get("key2");  // 返回 null
if (value1 != null) {
  System.out.println(value1.length());
}
if (value2 != null) {
  System.out.println(value2.length());
}
```

Option

- Option：没办法法回传一个有意义的东西
  - Some/None

```
val myMap = Map("key1" -> "value")
val value1: Option[String] = myMap.get("key1")
var length = value1 match {
  case Some(content) => content.length
  case None => 0
}
val value2: Option[String] = myMap.get("key2")
length = value2 match {
  case Some(content) => content.length
  case None => 0
}
```

```
value1.getOrElse("").length
value2.getOrElse("").length
```

尾递归

- 尾递归是指递归调用是函数的最后一个语句，而且其结果被直接返回

```
def factorial(n: Int): Int = {
  if( n <= 1 ) 1
  else n * factorial(n-1)
}
```

↓

```
def factorialTailrec(n: BigInt, acc: BigInt): BigInt = {
    if(n <= 1) acc
    else factorialTailrec(n-1, acc * n)
}
```

factorialTailrec(5, 1)
factorialTailrec(4, 5) // 1 * 5 = 5
factorialTailrec(3, 20) // 5 * 4 = 20

类

```
class Person(name: String, age: Int) {
  //从构造器
  def this(name: String) = this(name, 20)

  def getName() = name
  def getAge() = age
}

//伴生对象
object Person {
  var num = 0
  // 静态函数
  def test =  println("Person test")
  // 执行入口
  def main(args : Array[String]): Unit = {
    println(new Person("name").getAge())
    println(Person.num)
    Person.test
  }
}
```

其他

- Trait
- 泛型
- Actor

小结

- 优点
  -
- 缺点
  - 社区小
  - 偏复杂
  - 可读性稍差

01

Thanks！

命令式/函数式

- 内容

标题-左右结构