千锋教育
www.mobiletrain.org

千 锋 强 力 推 出

逆战班

停课不停学　品质不打折

鼓励全国学子在疫情中坚持学习

非常时期，通过网络，我们相聚在逆战班！
停课不停学，我必领先一步
逆战，待到疫情解除时，我必笑傲群芳
加油，加油，加油！

课程时间安排

在线上课时间安排	周一到周五	周六
上午第一节课	9:30-10:30	自习
间休	10:30-10:50	
上午第二节课	10:50-11:50	
间休	11:50-14:30	
下午第一节课	14:30-15:30	
间休	15:30-15:50	
下午第二节课	15:50-16:50	
晚休	16:50-18:30	
晚自习	19:00-21:30	
辅导时间	9:30-24:00	

好程序员
从平凡到卓越，为梦想而拼搏
GoodProgrammer.org

好程序员大数据学院出品

Spark 阶段知识串讲之Scala

DESIGN BY Goodprogrammer

好程序员
从平凡到卓越·为梦想而拼搏
GoodProgrammer.org

课程目标

COURSE CONTENTS

01 Scala回顾

02 Spark core 回顾

03 SparkSQL 回顾

04 Spark Streaming 回顾

05 Spark原理回顾

06 Spark调优回顾

07 Kafka、Redis回顾

第一章

Scala 回顾

概述 →

1、什么是函数式编程

2、基本概念：匿名函数、lambda表达式、闭包、高阶函数、柯里化、偏函数、偏应用函数、递归与尾递归

3、伴生类、伴生对象

4、case class 与 class 的区别

5、case class 与 case object 的区别

6、隐式转换及其应用场景

7、模式匹配

8、Option、Some、None

9、None、Unit、Null、Nothing、Nil、Any、AnyRef

10、+:、:+、++、::、:::、/:、:\

11、WordCount 程序

12、Scala的面试题串讲

1、什么是函数式编程 或 谈谈你对函数式编程的理解

函数式编程是一种编程的范式，函数式编程强调程序执行的结果而非执行的过程。函数式编程的特点有：

- 函数是一等公民(first class functions)。可以让函数像变量一样来使用，也就是说，函数可以像变量一样被创建，修改，并当成变量一样传递，返回或是在函数中嵌套函数。

- 不可变数据(immutable data)。默认变量是不可变的，推荐使用不可变的变量。因为程序中的状态不好维护，在并发的时候更不好维护。

- 没有副作用(No Side Effect)。副作用，指的是函数内部与外部互动（最典型的情况，就是修改全局变量的值），产生运算以外的其他结果。函数式编程强调没有”副作用”，意味着函数要保持独立，**所有功能就是返回一个新的值**，没有其他行为，尤其是不得修改外部变量的值。

- 少用循环和遍历，多使用 map 、 reduce等函数。函数式编程最常见的技术就是对一个集合做Map、Reduce等操作。这比起过程式的语言来说，在代码上要更容易阅读；

- 相关的技术包括：匿名函数、柯里化、高阶函数、递归与尾递归；

函数式编程的优点：

- 代码简洁，开发速度快
- 接近自然语言，易理解
- 易于代码管理
- 适合并发编程

2、基本概念：匿名函数、lambda表达式、闭包

➢ 匿名函数(lambda表达式)

在 Scala 中，不需要给每一个函数命名，没有将函数赋给变量的函数叫做匿名函数。

匿名函数又被称为 Lambda 表达式。 匿名函数的形式如下：

(参数) => 表达式

匿名函数实例：(x: Int) => x * 2

➢ 闭包

闭包是一种特殊的函数，返回值依赖于声明在函数外部的一个或多个变量。

```
var more = 10

// 闭包。返回值依赖于声明在函数外部的一个或多个变量
val z = (x: Int) => x + more
println(z(10))

more = 100
println(z(10))

more = 1000
println(z(10))
```

2、基本概念：高阶函数

- 高阶函数

高阶函数：接收一个或多个函数作为输入 或 输出一个函数。

函数的参数可以是变量，而函数又可以赋值给变量，即函数和变量地位一样，所以函数参数也可以是函数。

常用的高阶函数：map、reduce、fold、filter … …

2、基本概念：柯里化

➢ 柯里化

柯里化是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的技术。

```
def addCurrying1(x: Int)(y: Int) = x + y
def addCurrying2(x: Int) = (y: Int) => x + y

val result1 = addCurrying1(10)(10)
println(s"addCurrying1 = $result1")

val result2 = addCurrying2(20)(20)
println(s"addCurrying2(100)(100) = $result2")
```

常见的柯里化实现的算子：

```
def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1 = foldLeft(z)(op)

def sortBy[B](f: A => B)(implicit ord: Ordering[B]): Repr = sorted(ord on f)
```

2、基本概念：偏函数、偏应用函数

➢ 偏函数

被包在花括号内没有match的一组case语句是一个偏函数；

它是PartialFunction[A, B]的一个实例，A代表参数类型，B代表返回类型，常用作输入模式匹配；

```
val arr = (100 to 110).toArray
arr.zipWithIndex.map{case (elem, idx) =>
  println(s"elem = $elem; idx = $idx")
}
```

➢ 偏应用函数

偏应用函数也叫部分应用函数，与偏函数从仅有一字之差，但它们二者之间却有天壤之别。

部分应用函数, 是指一个函数有n个参数, 而为其提供少于n个参数, 那就得到了一个部分应用函数。

```
// 偏应用函数
def add(x: Int, y: Int, z: Int) = x + y + z
def addX = add(10, _: Int, _: Int)
val resultX = addX(20, 30)
println(s"addX(20, 30) = $resultX")
arr.foreach(println)
```

2、基本概念：递归与尾递归

在函数式编程中，递归的出镜率大大高于其他类型的语言。

递归函数的核心是设计好递归表达式，并且确定算法的边界条件。

递归算法需要保持调用堆栈，效率较低，如果调用次数较多，会耗尽内存或栈溢出。

尾递归可以克服这一缺点。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。由于递归结果总是直接返回，尾递归比较方便转换为循环，因此编译器容易对它进行优化。

```
// 递归与尾递归
val lst = (1 to 10).toList
def lstSum1(lst: List[Int]): Int = {
  lst match {
    case Nil => 0
    case head :: tail => head + lstSum1(tail)
  }
}
val result5 = lstSum1(lst)
println(s"lstSum1(lst) = $result5")

// 尾递归
def lstSum3(lst: List[Int], result: Int): Int = {
  lst match {
    case Nil => result
    case head :: tail => lstSum3(tail, result + head)
  }
}
```

3、伴生类 与 伴生对象

单例对象与类同名时，这个单例对象被称为这个类的伴生对象；

这个类被称为这个单例对象的伴生类。不与伴生类同名的单例对象称为孤立对象;

- 伴生类和伴生对象要在同一个源文件中定义；
- 伴生对象和伴生类可以互相访问其私有成员；

4、case class 与 class

class是普通的类；

case class称为样例类，在scala中主要用于模式匹配；

样例类是scala中特殊的类。当声明样例类时，如下事情会自动发生：

- 构造器中每一个参数都成为val。除非它被显示的声明为var（不建议这样做）
- 提供apply方法。不用new关键字就能够构造出相应的对象
- 提供unapply方法。让模式匹配可以工作
- 将生成toString、equals、hashCode和copy方法。除非你显示的给出这些方法的定义
- 继承了Product和Serializable接口，也就是说实现了序列化 和 可以应用Product的方法；

case class 在其他方面与 class 基本一样，可以为它们添加方法和字段，扩展它们；

5、case class 与 case object

case class是多例的，后面要跟构造参数；

case object是单例的，后面不能跟参数；

6、隐式转换(implicit)及其应用

隐式函数：是指那种以implicit关键字声明的函数。(注意：可以带有多个参数)

隐式参数：隐式参数有点类似缺省参数，如果在调用方法时没有提供某个参数，编译器会在当前作用域查找是否有符合条件的 implicit 对象可以作为参数传入；

隐式转换：最为核心的就是定义隐式函数，在程序的一定作用域中scala会自动调用。Scala会根据隐式函数的签名，在程序中使用到隐式函数参数定义的类型对象时，会自动调用隐式函数将其传入隐式函数，转换为另一种类型对象并返回，这就是scala中的“隐式转换”。

- 隐式转换函数的函数名可以是任意的，与函数名称无关，只与函数签名（函数参数和返回值类型）有关；
- 如果当前作用域中存在函数签名相同但函数名称不同的两个隐式转换函数，则在进行隐式转换时会报错；
- 隐式转换函数定义在在伴生对象中；

在Spark Sql中，隐式转换大量的应用到了DSL风格语法中。并且在Spark 2.0版本以后，DataSet里面如果进行转换RDD或者DF的时候，那么都需要导入必要的隐式转换操作。

7、模式匹配

Scala 提供了强大的模式匹配机制，应用也非常广泛。

一个模式匹配包含了一系列备选项，每个都开始于关键字 case。每个备选项都包含了一个模式及一到多个表达式。箭头符号 => 隔开了模式和表达式。

Scala中模式匹配最常用于match语句中。Java风格的switch可以很自然地用match表达式表达。但它们之间还是有三个显著的区别：

- Scala的match是一个表达式（即有返回值）；
- Scala的可选分支不会贯穿到下一个case（即匹配上了就立即返回）；
- 如果没有任何一个模式匹配上，会抛出异常（MatchError）；

8、Option、Some、None

在Scala语言中，Option类型是一个特殊的类型，它是代表有值和无值的体现，内部有两个对象，一个是Some，一个是None；

- Some代表有返回值，内部有值
- 而None恰恰相反，表示无值

比如，我们使用Map集合进行取值操作的时候，当我们通过get取值，返回的类型就是Option类型，而不是具体的值。

9、None、Unit、Null、Nothing、Nil、Any、AnyRef

Null是一个trait（特质），是所以引用类型AnyRef的一个子类型，null是Null唯一的实例；

Nothing也是一个trait（特质），是所有类型Any（包括值类型和引用类型）的子类型，它不再有子类型，它也没有实例，实际上为了一个方法抛出异常，通常会设置一个默认返回类型；

Nil代表一个List空类型，等同List[Nothing]；

None是Option monad的空标识；

Scala中的Unit类型类似于java中的void，无返回值。主要的不同是在Scala中可以有一个Unit类型值，也就是（ ），然而java中是没有void类型的值的。除了这一点，Unit和void是等效的。一般来说每一个返回void的java方法对应一个返回Unit的Scala方法。

10、+:、:+、++、::、:::、/:、:\

```
val list = List(1,2,3)
// :: 用于的是向队列的头部追加数据,产生新的列表, x::list,x就会添加到list的头部
println(4 :: list)  //输出: List(4, 1, 2, 3)
// .:: 这个是list的一个方法;作用和上面的一样,把元素添加到头部位置; list.::(x);
println( list.:: (5)) //输出: List(5, 1, 2, 3)
// :+ 用于在list尾部追加元素; list :+ x;
println(list :+ 6)  //输出: List(1, 2, 3, 6)
// +: 用于在list的头部添加元素;
val list2 = "A"+:"B"+:Nil //Nil Nil是一个空的List,定义为List[Nothing]
println(list2)  //输出: List(A, B)
// ::: 用于连接两个List类型的集合 list ::: list2
println(list ::: list2) //输出: List(1, 2, 3, A, B)
// ++ 用于连接两个集合, list ++ list2
println(list ++ list2) //输出: List(1, 2, 3, A, B)

println("_____")
// :\ 等价于 foldRight
println(List(1,2,3) .:\ (0)((x:Int,y:Int)=>x+y))
println(List(1,2,3) .:\ (0)(_+_))

println("_____")
// /: 等价于 foldLeft
println(List(1,2,3) ./: (0)((x:Int,y:Int)=>x+y))
println(List(1,2,3) ./: (0)(_+_))
}
```

经验：若是使用grammar sugar方式总是报错，就是用普通方式。

如：

/: <=>foldLeft

:\ <=> foldRight

11、WordCount 程序

```
import scala.io.BufferedSource

// 计算词频，并按词频降序排列
object WordCount {
  def main(args: Array[String]): Unit = {
    val source: BufferedSource = scala.io.Source.fromFile("spark.log")
    val lst = source.getLines()
      .toList

    lst.flatMap(_.split("\\s+"))
      .filter(_.nonEmpty)
      .groupBy(x => x)
      .mapValues(_.length)
      .toList
      .sortBy(_._2)
      .reverse
      .foreach(println)

    source.close()
  }
}
```

Scala面试题

1、 介绍一下 scala

Scala 是一种多范式语言，它一方面吸收继承了多种语言中的优秀特性，一方面又没有抛弃 Java 这个强大的平台，它运行在 Java 虚拟机 (Java Virtual Machine) 之上，轻松实现和丰富的 Java 类库互联互通。它既支持面向对象的编程方式，又支持函数式编程。它写出的程序像动态语言一样简洁，但事实上它确是严格意义上的静态语言，相对于Java而言，Scala的代码更为精简（减低犯错），而且功能更为广泛（Scala其实是 Scalable Language 的简称，意为可扩展的语言），许多Scala的特性和语法都是针对Java的不足和弱点来设计的。

2、 关于scala，你有什么想说的吗？scala可以用在哪些方面呢？

Scala的特点是有很多函数程式语言的特性（例如ML，Miranda, Scheme，Haskell），譬如惰性求值，list comprehension, type inference, anonymous function, pattern matching 等等，同时也包含 Object-Oriented 的特性（OO 能与 FP 混合使用是 Scala 的亮点）。此外，许多相似于高级编程语言的语法也渗入其中（例如 Python），不仅提高了 Scala 代码的可读性，维护、修改起来也较为省时省力。

scala语言主要在于Spark开发中进行使用，代码编写简洁方便，并且执行效率很高，相比较Java语言来说，代码可以减少几倍，还很灵活。

3、scala懒加载问题怎么处理？

使用Lazy关键字进行懒加载操作

在一些情况中经常希望某些变量的初始化要延迟，并且表达式不会被重复计算。就像我们用Java实现一个懒汉式的单例。如：打开一个数据库连接。这对于程序来说，执行该操作，代价是昂贵的，所以我们一般希望只有在使用其的引用时才初始化。（当然实际开发中用的是连接池技术）

为了缩短模块启动时间，可以将当前不需要的某些工作推迟执行。保证对象中其他字段的初始化能优先执行。

Scala面试题

4、 Scala有break吗，Case class了解吗，哪里用到过？

Scala没有break操作，但是可以实现break原理，需要创建Breaks对象实现内部的break方法就可以像java一样跳出语句，但是在模式匹配过程中不需要跳出匹配模式，因为模式匹配只能匹配其中一个结果值。

case class代表样例类，它和class类比较来说，可以不需要序列化，而class需要序列化操作，和object很类似，但是不同的是object不能传入参数，而case class可以带入参数，一般在做转换操作传参使用，比如DataSet操作的时候，转换RDD或者DataFream操作时候，可以使用case class进行参数的传递。

5、 元组

→ 元组的创建

```
val tuple1 = (1, 2, 3, "呵呵哒")

println(tuple1)
```

5、 元组（续）

→ 元组数据的访问，注意元组元素的访问有下划线，并且访问下标从1开始，而不是0

```
val value1 = tuple1._4

println(value1)
```

→ 元组的遍历

方式1：

```
for (elem <- tuple1.productIterator ) {

   print(elem)

}
```

方式2：

```
tuple1.productIterator.foreach(i => println(i))

tuple1.produIterator.foreach(print(_))
```

Scala面试题

6、 隐式转换

隐式转换函数是以implicit关键字声明的函数。这种函数将会自动应用，将值从一种类型转换为另一种类型。

```
implicit def a(d: Double) = d.toInt

// 当执行这句代码的时候，内部会自动调用我们自己编写好的隐式转换方法

val i1: Int = 3.5

println(i1)
```

7、 隐式转换应用场景

在scala语言中，隐式转换一般用于类型的隐式调用，亦或者是某个方法内的局部变量，想要让另一个方法进行直接调用，那么需要导入implicit关键字，进行隐式的转换操作，同时，在Spark Sql中，这种隐式转换大量的应用到了我们的DSL风格语法中，并且在Spark2.0版本以后，DataSet里面如果进行转换RDD或者DF的时候，那么都需要导入必要的隐式转换操作。

8、 什么叫闭包

闭包是一个函数，返回值依赖于声明在函数外部的一个或多个变量。

闭包通常来讲可以简单的认为是可以访问一个函数里面局部变量的另外一个函数。

→案例1：

```
def minusxy(x: Int) = (y: Int) => x - y
```

这就是一个闭包：

1) 匿名函数(y: Int) => x -y嵌套在minusxy函数中。

2) 匿名函数(y: Int) => x -y使用了该匿名函数之外的变量x

3) 函数minusxy返回了引用了局部变量的匿名函数

→案例2

```
def minusxy(x: Int) = (y: Int) => x - y

val f1 = minusxy(10)

val f2 = minusxy(10)

println(f1(3) + f2(3))
```

此处f1,f2这两个函数就叫闭包。

Scala面试题

9、 解释一下Scala内的Option类型

在Scala语言中，Option类型是一个特殊的类型，它是代表有值和无值的体现，内部有两个对象，一个是Some一个是None，Some代表有返回值，内部有值，而None恰恰相反，表示无值，比如，我们使用Map集合进行取值操作的时候，当我们通过get取值，返回的类型就是Option类型，而不是具体的值。

10、 解释一下什么叫偏函数

偏函数表示用{}包含用case进行类型匹配的操作，这种操作一般用于匹配唯一的属性值，在Spark中的算子内经常会遇到，例：

```
val rdd = sc.textFile(路径)

rdd.map{

    case (参数)=>{返回结果}

}
```

11、 手写Scala单例模式

单例模式是一种常用的软件设计模式。在它的核心结构中只包含一个被称为单例的特殊类。通过单例模式可以保证系统中一个类只有一个实例。

```
/**

 * scala中关于单例的模拟

 * object中的属性和方法都可以当做类似java中的静态成员，都可以通过

 * object.成员来进行调用

 */

object SingletonOps {

 def main(args: Array[String]): Unit = {

  val singleton1 = Singleton.getInstance

  val singleton2 = Singleton.getInstance

  println(singleton1 == singleton2)

  singleton1.index = 5

  println("singleton1.index：" + singleton1.index)

  println("singleton2.index：" + singleton2.index)

 }

}
```

11、 手写Scala单例模式（续）

```
object Singleton {

 private val singleton = Singleton;

 def getInstance = singleton

 var index = 1

}
```

Scala面试题

12、 解释一下柯里化

定义：柯里化指的是将原来接受两个参数的函数变成新的接受一个参数的函数的过程。新的函数返回一个以原有的第二个参数作为参数的函数

例如：

```
def mul(x:Int,y:Int) = x * y  //该函数接受两个参数

def mulOneAtTime(x:Int) = (y:Int) => x * y  //该函数接受一个参数生成另外一个接受单个参数的函数
```

这样的话，如果需要计算两个数的乘积的话只需要调用：

```
mulOneAtTime(5)(4)
```

这就是函数的柯里化

13、 Scala中的模式匹配和Java的匹配模式的区别

一个模式匹配包含了一系列备选项，每个都开始于关键字case。每个备选项都包含了一个模式及一到多个表达式。箭头符号 => 隔开了模式和表达式。

例如:

```
obj match{
        case 1 => "one"
        case 2 => "two"
        case 3 => "three"
        case _ => default
}
```

而Java的匹配模式是switch case匹配方式，它内部匹配的类型有局限性，并且需要用Break跳出匹配模式，而Scala中只会匹配其中一个结果，同时匹配类型居多，如String、Array、List、Class等..

14、 Scala中的伴生类和伴生对象是怎么一回事

在scala中，单例对象与类同名时，该对象被称为该类的伴生对象，该类被称为该对象的伴生类。

伴生类和伴生对象要处在同一个源文件中

伴生对象和伴生类可以互相访问其私有成员

不与伴生类同名的对象称之为孤立对象

15、 谈谈Scala的尾递归

正常的递归，每一次递归步骤，需要保存信息到堆栈中去，当递归步骤很多的时候，就会导致内存溢出

而尾递归，就是为了解决上述的问题，在尾递归中所有的计算都是在递归之前调用，编译器可以利用这个属性避免堆栈错误，尾递归的调用可以使信息不插入堆栈，从而优化尾递归

例如:

```
5 + sum(4) // 暂停计算 => 需要添加信息到堆栈
5 + (4 + sum(3))
5 + (4 + (3 + sum(2)))
5 + (4 + (3 + (2 + sum(1))))
5 + (4 + (3 + (2 + 1)))
15

tailSum(4, 5) // 不需要暂停计算
tailSum(3, 9)
tailSum(2, 12)
tailSum(1, 14)
tailSum(0, 15)
15
```

Scala面试题

16、 函数中 Unit是什么意思

Scala中的Unit类型类似于java中的void，无返回值。主要的不同是在Scala中可以有一个Unit类型值，也就是（ ），然而java中是没有void类型的值的。除了这一点，Unit和void是等效的。一般来说每一个返回void的java方法对应一个返回Unit的Scala方法。

17、Scala中的to和until 有什么区别

例如

1 to 10，它会返回Range（1,2,3,4,5,6,7,8,9,10），

而1 until 10 ，它会返回Range（1,2,3,4,5,6,7,8,9）

也就是说to包头包尾，而until 包头不包尾！

18、var，val和def三个关键字之间的区别

var是变量声明关键字，类似于Java中的变量，变量值可以更改，但是变量类型不能更改。 val常量声明关键字。
def 关键字用于创建方法（注意方法和函数的区别） 还有一个lazy val（惰性val）声明，意思是当需要计算时才使用，避免重复计算

代码示例：

```
var x = 3 //  x是Int类型

x = 4     //

x = "error" // 类型变化，编译器报错'error: type mismatch'

val y = 3

y = 4       //常量值不可更改，报错 'error: reassignment to val'

def toGreet(name: String) = "Hey! My name is: " + name

toGreet("Scala") // "Hey! My name is: Scala"

//注意scala中函数式编程一切都是表达式

lazy val x = {

  println("computing x")

  3

}

val y = {

  println("computing y")

  10

}

 // x 没有计算, 打印结果"computing y"
```

Scala面试题

19、trait(特质)和abstract class（抽象类）的区别

（1）一个类只能继承一个抽象类，但是可以通过with关键字继承多个特质；

（2）抽象类有带参数的构造函数，特质不行（如 trait t（i：Int）{} ，这种声明是错误的）

20、unapply 和apply方法的区别， 以及各自使用场景

提取器是从传递给它的对象中提取出构造该对象的参数。

Scala 标准库包含了一些预定义的提取器。

Scala 提取器是一个带有unapply方法的对象。unapply方法算是apply方法的反向操作：unapply接受一个对象，然后从对象中提取值，提取的值通常是用来构造该对象的值。

提取器在模式匹配语句中自动被调用

21 Scala类型系统中Nil, Null, None, Nothing四个类型的区别

Null是一个trait（特质），是所以引用类型AnyRef的一个子类型，null是Null唯一的实例。

21 Scala类型系统中Nil, Null, None, Nothing四个类型的区别（续）

Nothing也是一个trait（特质），是所有类型Any（包括值类型和引用类型）的子类型，它不在有子类型，它也没有实例，实际上为了一个方法抛出异常，通常会设置一个默认返回类型。

Nil代表一个List空类型，等同List[Nothing]

None是Option monad的空标识

22、 call-by-value和call-by-name求值策略的区别

（1）call-by-value是在调用之前计算；

（2） call-by-name是在需要时计算

示例代码 →

```
//声明第一个方式
def func(): Int = {
  println("computing stuff....")
  42 // return something
}
```

Scala面试题

22、 call-by-value和call-by-name求值策略的区别（续）

```
//声明第二个方法，scala默认的求值就是call-by-value
def callByValue(x: Int) = {
  println("1st x: " + x)
  println("2nd x: " + x)
}
//声明第三个方法，用=>表示call-by-name求值
def callByName(x: => Int) = {
  println("1st x: " + x)
  println("2nd x: " + x)
}
```

```
//开始调用
//call-by-value求值
callByValue(func())
//输出结果
//computing stuff....
//1st x: 42
//2nd x: 42


//call-by-name求值
callByName(func())
//输出结果
//computing stuff....
//1st x: 42
//computing stuff....
//2nd x: 42
```

Scala面试题

23、 yield如何工作？comprehension（推导式）的语法糖是什么操作？

yield →

①yield用于循环迭代中生成新值，yield是comprehensions的一部分，是多个操作（foreach, map, flatMap, filter or withFilter）的composition语法糖。

②针对每一次 for 循环的迭代, yield 会产生一个值，被循环记录下来 (将内部实现上可以看看作一个缓冲区)；当循环结束后, 会返回所有 yield 的值组成的集合；返回集合的类型与被遍历的集合类型是一致的；

comprehension（推导式）→

是若干个操作组成的替代语法。如果不用yield关键字，comprehension（推导式）可以被forech操作替代，或者被map/flatMap，filter代替。

示例代码：

```
def scalaFiles =
  for {
    file <- List(new File("a_file/jd.log"))
    if file.isFile
    if file.getName.endsWith(".log")
  } yield file


scalaFiles.foreach(println)
```

等价于：

```
List(new File("a_file/jd.log"))
.filter(f=>f.isFile && f.getName.endsWith(".log"))
.foreach(println)
```

Scala面试题

24、什么是高阶函数？

高阶函数指能接受或者返回其他函数的函数，scala中的filter map flatMap函数都能接受其他函数作为参数。

25、scala全排序过滤字段，求 1 to 4 的全排序， 2不能在第一位， 3,4不能在一起

```
import util.control.Breaks._
// 1 to 4 的全排序
// 2不能在第一位
// 3,4不能在一起
 object LocalSpark extends App{
    override def main(args: Array[String]): Unit = {
```

25、scala全排序过滤字段，求 1 to 4 的全排序,2不能在第一位, 3,4不能在一起（续）

```
    List(1, 2, 3, 4).permutations.filter(list => list(0) != 2).map(list => {
      var num = 0
      breakable {
        for (x <- 0 to (list.size - 1)) {
          if (list(x) == 3 && x < 3 && list(x + 1) == 4) break
          if (list(x) == 3 && x > 0 && list(x - 1) == 4) break
          num += 1
        }
      }
      if (num < 4) {
        List()
      } else {
        list
      }
    }).filter(list => list.size > 3).foreach(println(_))
  }
}
```

Scala面试题

25、scala全排序过滤字段，求 1 to 4 的全排序， 2不能在第一位，3,4不能在一起（续）

结果 →

List(1, 3, 2, 4)

List(1, 4, 2, 3)

List(3, 1, 2, 4)

List(3, 1, 4, 2)

List(3, 2, 1, 4)

List(3, 2, 4, 1)

List(4, 1, 2, 3)

List(4, 1, 3, 2)

List(4, 2, 1, 3)

List(4, 2, 3, 1)