

## scala里的List/Stream/View机制浅析

原创 2017年10月21日 19:23:47 标签： scala (http://so.csdn.net/so/search/s.do?q=scala&t=blog) 67

### List机制浅析

scala里的List就是单向链表，一般通过下面方式来组装一个List：

```
1 val l = 1 :: 2 :: 3 :: Nil
```

Nil是空List，::是右结合操作符，所以上述写法相当于：

Nil::(3)::(2)::(1)

我们来看看::连接符是如何实现的：

```
1 sealed abstract class List[+A] ..... {
2   ...
3   def isEmpty: Boolean
4   def head: A
5   def tail: List[A]
6
7   def ::[B >: A](x: B): List[B] =
8     new scala.collection.immutable.::(x, this)
9   .....
10 }
11
12 final case class ::[B](override val head: B, private[scala] var tl: List[B]) extends
13 List[B] {
14   override def tail : List[B] = tl
15   override def isEmpty: Boolean = false
16 }
```

可见List是由Cons节点（即::类的实例）链接而成，每个Cons节点除了包含值（即head），还有一个指向尾List的tail指针。注意::类的类参数列表里val head前加override修饰符是因为::类重载了基类的head方法，在scala里成员方法和成员变量是被一视同仁的，也就是说，定义了一个成员方法后，我们不能再定义一个同名的成员变量。

另外，元素在加入List之前是要立即计算的，什么意思呢，像下面的语句：

```
1 val l = 1 :: {println("haha");2} :: {println("hehe");3} :: Nil
2 println(l(0))
```

会输出：

haha  
hehe

1  
虽然我们只打印第1个元素，但第2、3个元素里的println动作也执行了，说明在List就绪之时，所有元素都必须计算出来。这样的话，List就无法去表示一个无限序列了。要表达一个无限序列，必须用Stream。

### Stream机制浅析

Stream的写法是这样的：

```
1 val s = 1 #:: {println("haha");2} #:: {println("hehe");3} #:: Stream.empty
2 println(s)
```



李宝胖 (http://blog.csdn...)

+ 关注

(http://blog.csdn.net/tlxamulet)

码云

未开通

(https://gi  
utm\_sourc

原创

13

粉丝

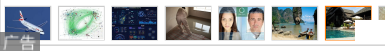
0

喜欢

2



泰国清迈攻略



#### 他的最新文章

更多文章 (http://blog.csdn.net/tlxamulet)

pthread包的mutex实现分析 (http://blog.csdn.net/tlxamulet/article/details/79047717)

python的GIL锁的是什么 (http://blog.csdn.net/tlxamulet/article/details/78965404)

写文件操作探微 (http://blog.csdn.net/tlxamulet/article/details/78825396)

spark小结 (http://blog.csdn.net/tlxamulet/article/details/78765040)

scala里如何中断for循环 (http://blog.csdn.net/tlxamulet/article/details/78236641)

#### 文章分类

java技术 (http://blog.csdn.net/ 1篇

DSL (http://blog.csdn.net/tl... 1篇

scala (http://blog.csdn.net/t... 8篇

操作系统 (http://blog.csdn.net/ 2篇

python (http://blog.csdn.net/ 1篇

展开 ∨

#### 文章存档

2018年1月 (http://blog.csdn... 2篇

2017年12月 (http://blog.csd...

输出：

Stream(1, ?)

说明Stream就绪时，仅有head元素是计算了的，其他元素均是未知。也就是说，元素在加入Stream之前是无需立即计算的，只在要用时才会计算，比如我们这样写：

```
1 println(s(1))
```

输出：

haha

2

那么，Stream是如何实现这种元素的惰性计算机制的呢？来看代码：

```
1 class ConsWrapper[A](tl: => Stream[A]) {
2     /** Construct a stream consisting of a given first element followed by elements
3     *   from a lazily evaluated Stream.
4     */
5     def #::(hd: A): Stream[A] = cons(hd, tl)
6     .....
7 }
8
9 object cons {
10
11     /** A stream consisting of a given first element and remaining elements
12     *   @param hd   The first element of the result stream
13     *   @param tl   The remaining elements of the result stream
14     */
15     def apply[A](hd: A, tl: => Stream[A]) = new Cons(hd, tl)
16 }
17
18 final class Cons[+A](hd: A, tl: => Stream[A]) extends Stream[A] {
19     override def isEmpty = false
20     override def head = hd
21     @volatile private[this] var tlVal: Stream[A] = _
22     @volatile private[this] var tlGen = tl _
23     def tailDefined: Boolean = tlGen eq null
24     override def tail: Stream[A] = {
25         if (!tailDefined)
26             synchronized {
27                 if (!tailDefined) {
28                     tlVal = tlGen()
29                     tlGen = null
30                 }
31             }
32
33         tlVal
34     }
35 }
```

我们发现，Stream跟List有点类似，也是一个单向链表，head指向元素值，但与List不同的是，tail指针并不指向尾队列，而是指向一个生成尾队列的函数：

tl: => Stream[A]

既然tail传递的是函数，而非尾队列，说明Stream除了首元素外，其他元素都不是立即计算的。

我们再看Cons类的tail方法，这个方法的实现有两个关键点：

- 1、尾队列的值tlVal是按需计算出来的，见tlVal = tlGen()这一句
  - 2、一旦tlVal计算出来后，再次调用tail，就直接返回tlVal，不会再重复计算，这是通过tailDefined的检查来保证的。这说明，Stream具有记忆能力，可以缓存中间计算结果，以空间换时间。
- 所以，Stream适合作为无限序列的生成器，且可用于累积计算场景。

scala里的Stream与java8的Stream（流式操作）名字虽同，可却是全然不同的概念，scala里真正与java8的Stream对应的，其实是View。下面我们来分析View。

2篇

2017年10月 (<http://blog.csdn.net/tlxamulet/article/details/78236641>)

2篇

2017年9月 (<http://blog.csdn.net/tlxamulet/article/details/77871194>)

5篇

展开▼

## 他的热门文章

scala里如何中断for循环 (<http://blog.csdn.net/tlxamulet/article/details/78236641>)

🔖 212

scala函数的定义 (<http://blog.csdn.net/tlxamulet/article/details/77871194>)

🔖 165

java下的证书访问 (<http://blog.csdn.net/tlxamulet/article/details/77142500>)

🔖 90

scala下DSL的设计与开发 (<http://blog.csdn.net/tlxamulet/article/details/77387579>)

🔖 75

scala的trait之我见 (<http://blog.csdn.net/tlxamulet/article/details/77870918>)

🔖 73

## View机制浅析

我们看一个具体的例子：

```
1 @Test
2   def testView: Unit ={
3     val l = 0 to 5
4     println(l.map(x => x * x).zip(10 to 15))
5     println(l.view.map(x => x * x).zip(10 to 15))
6   }
```

输出：

Vector((0,10), (1,11), (4,12), (9,13), (16,14), (25,15))

SeqViewMZ(...)

未调用view的操作序列输出一个Vector，而调用了view的操作序列仅输出一个SeqViewMZ对象（这里的M是Mapped，Z是Zipped的意思），并未真正计算，若要看到view的结果，需调force强制计算：

```
1 println(l.view.map(x => x * x).zip(10 to 15).force)
```

输出：

Vector((0,10), (1,11), (4,12), (9,13), (16,14), (25,15))

我们来看看view及map的实现：

```
1   override def view = new SeqView[A, Repr] {
2     protected lazy val underlying = self.repr
3     override def iterator = self.iterator
4     override def length = self.length
5     override def apply(idx: Int) = self.apply(idx)
6   }
7
8   override def map[B, That](f: A => B)(implicit bf: CanBuildFrom[This, B, That]): Th
9   at = {
10     newMapped(f).asInstanceOf[That]
11   }
12
13   protected def newMapped[B](f: A => B): Transformed[B] = new { val mapping = f } wi
14   th AbstractTransformed[B] with Mapped[B]
15
16   trait Mapped[B] extends Transformed[B] {
17     protected[this] val mapping: A => B
18     def foreach[U](f: B => U) {
19       for (x <- self)
20         f(mapping(x))
21     }
22   }
```

view中underlying用lazy修饰，确保SeqView对应的真实容器是按需使用的。

map方法在SeqView基础上创建了一个MappedSeqView型实例，该实例所做的事情就是把 $x \Rightarrow x * x$ 函数保存起来，即`val mapping = f`这句。MappedSeqView的foreach方法是要立即计算的，我们看到，它针对SeqView集合的每个元素都要先调一次 $x \Rightarrow x * x$ ，接着才调foreach的f函数，如下面代码所示：

```
1 for (x <- self)
2   f(mapping(x))
```

再来看zip调用：

```

1  override def zip[A1 >: A, B, That](that: GenIterable[B])(implicit bf: CanBuildFrom[T
2  his, (A1, B), That]): That = {
3      newZipped(that).asInstanceOf[That]
4
5  protected def newZipped[B](that: GenIterable[B]): Transformed[(A, B)] = new { val ot
6  her = that } with AbstractTransformed[(A, B)] with Zipped[B]
7
8  trait Zipped[B] extends Transformed[(A, B)] {
9      protected[this] val other: GenIterable[B]
10     def iterator: Iterator[(A, B)] = self.iterator zip other.iterator
11     final override protected[this] def viewIdentifier = "Z"
12 }

```

zip方法在MappedSeqView的基础上又创建了一个ZippedSeqView型实例，该实例将zip的对端序列（这里是10 to 15）缓存到other成员，即val other = that这句。ZippedSeqView的iterator方法则将上一个集合（即MappedSeqView）的迭代器与对端序列（即other成员）的迭代器做zip结合，如下面代码所示：

```

1  def iterator: Iterator[(A, B)] = self.iterator zip other.iterator

```

所以，view的调用过程像这样（foreach最终会转到iterator）：

```

ZippedSeqView.foreach(
zip(other, MappedSeqView.foreach(
map(SeqView.foreach(
underlying.foreach))))))

```

等价于：

```

underlying.foreach(zip(other.item, map(underlying.item)))

```

也就是说，view是将操作累积起来了，它不像非view版本那样会生成临时集合。就我们这个例子而言，非view版本的处理过程是这样的：

```

Collection(0,1,2,3,4,5) -> Collection(0,1,4,9,16,25) -> Collection((0,10), (1,11), (4,12), (9,13), (16,14), (25,15))

```

而view版本则是这样的：

```

Collection(0,1,2,3,4,5) -> Collection((0*0,10), (1*1,11), (2*2,12), (3*3,13), (4*4,14), (5*5,15))

```

可见非view版本生成了额外的临时集合，且对原始集合(0,1,2,3,4,5)和临时集合(0,1,4,9,16,25)各做了一次遍历，最终生成结果集合((0,10), (1,11), (4,12), (9,13), (16,14), (25,15))。

而view版本由于不依赖临时集合，只需对原始集合(0,1,2,3,4,5)做一次遍历即可生成结果集合((0,10), (1,11), (4,12), (9,13), (16,14), (25,15))。这样的处理在原始集合数据量很大时，能有效节省内存、提升效率。

最后说明一下，view的这种处理方式有一个专有叫法：惰性化计算，什么意思呢？打个比方，就是在我们提交map计算给集合的时候，集合说：“知道了，我等会做”，其实它并没做，只有你真正需要结果时它才会不紧不慢的去执行，此谓之“惰性”。

## 对比

List与Stream：前者用于有限集合，后者用于无限集合。比如下面代码：

```

1  def constList(n:Int):List[Int] = n :: constList(n)
2
3  def constStream(n:Int):Stream[Int] = n #:: constStream(n)
4
5  @Test
6  def testListStream: Unit ={
7      println(constStream(10))
8      println(constList(10))
9  }

```

println(constStream(10))会输出Stream(10, ?)

而println(constList(10))会栈溢出。

List和View：前者在做集合转换操作（如zip、map、flatMap等）时会生成中间集合，后者则不会，只在集合行为操作（如foreach）时一下子计算出中间累积操作的结果，后者在大集合时更省内存。

Stream和View：两者都会做惰性计算，但关注的维度不一样，前者是集合里元素计算的惰性化，后者则是集合本身计算的惰性化。事实上，Stream是有一个view方法的。



版权声明：本文为博主原创文章，未经博主允许不得转载。



发表你的评论

(<http://my.csdn.net/xuhao7891>)

## Scala中Stream的应用场景及其实现原理



cuipengfei1 2014年10月26日 11:00 4981

欢迎访问我的独立博客：<http://cuipengfei.me/blog/2014/10/23/scala-stream-application-scenario-and-how-its-impl-em...>

(<http://blog.csdn.net/cuipengfei1/article/details/40475201>)

## Scala进阶源码实战之四——模式匹配



Crystal\_Zero 2016年04月19日 22:22 774

basicpackage PatternMatchobject patternmatch { println("Welcome to Scala worksheet") //>...

([http://blog.csdn.net/Crystal\\_Zero/article/details/51194671](http://blog.csdn.net/Crystal_Zero/article/details/51194671))



返回顶部

## 区块链概念股大揭秘！这些股值得入手！



【网易官方股票交流群】添加微信好友，进群免费领牛股→

## scala筑基篇-01-List操作



hylexus 2016年09月13日 19:08 578

List简介 特性 创建列表 操作 list的基本操作 list类的一阶方法 连接 长度 reverse apply indices zip mkString list类的高阶方法 foreach ...

(<http://blog.csdn.net/hylexus/article/details/52528498>)

## scala List集合的用法



u010666884 2016年07月22日 17:23 17982

一、前言：人们常说，Scala是一个难掌握的语言，一是其面向函数和面向对象结合的原因，二是其丰富的语法和内置函数。对于Conllection 这一章的内容，更多的是利用内置函数灵活地运用，避免...

(<http://blog.csdn.net/u010666884/article/details/51994461>)

## Scala教程(十二)List操作高级进阶实战



yuan\_xw 2015年10月13日 16:43 1887

List组成结构：数组由head tail两部分组成：head表示第一个元素，tail表示其它元素。 :::操作符：list与list之间进行连接符::: List的foldLeft、 foldRigh...

([http://blog.csdn.net/yuan\\_xw/article/details/49100627](http://blog.csdn.net/yuan_xw/article/details/49100627))

## 不再死记硬背，一个公式学懂英文



英语长难句解读，记住这个公式就够了！

## Scala List一阶函数操作



yyywyr 2015年12月15日 22:05 1217

Scala List一阶函数操作

(<http://blog.csdn.net/yyywyr/article/details/50321167>)



99元特价机票



### 联系我们



网站客服  
(<http://wpa.qq.com/msgrd?v=3&uin=2431299880&site=qq&webmaster@csdn.net>)  
webmaster@csdn.net  
(mailto:webmaster@csdn.net)



微博客服  
(<http://e.weibo.com/csdnsupport/p400-660-0108>)  
400-660-0108

关于 招聘 广告服务



© 2018 CSDN 京ICP证09002463号

(<http://www.miibeian.gov.cn/>)



经营性网站备案信息  
(<http://www.hd315.gov.cn/beian/view.asp?bianhao=010202001032100010>) 网络  
110报警服务 (<http://www.cyberpolice.cn/>)

## Scala集合操作—List



u013514928 2016年10月11日 18:17 1874

Scala常用操作：[http://www.yiibai.com/scala/scala\\_lists.html](http://www.yiibai.com/scala/scala_lists.html)

Scala中列表是非常类似于数组，这意味着，一个列表的所有元素都具有相同的类...

(<http://blog.csdn.net/u013514928/article/details/52789453>)

## Scala入门到精通——第三节 Array、List



lovehuangjiaju 2015年07月20日 09:05 26503

本节主要内容 数组操作实战 列表List操作实战 数组操作实战1 定长数组//定义一个长度为10的数值数组 scala> val number Array=new Array[Int](10) numb...

(<http://blog.csdn.net/lovehuangjiaju/article/details/46963721>)

## 大数据学习之Scala中列表（List）的使用学习（5）

在上篇文章当中（[http://blog.csdn.net/poison\\_h/article/details/50456398](http://blog.csdn.net/poison_h/article/details/50456398)），我们学习了Scala中的Array,我想新大家对Scala有稍稍的认识了...

([http://blog.csdn.net/Poison\\_H/article/details/50474092](http://blog.csdn.net/Poison_H/article/details/50474092))

## Scala中常见的容器 List



Winterto1990 2016年08月03日 16:28 1433

为何选择Scala？Scala是一门混合了函数式和面向对象的语言。用Scala创建多线程应用时，你会倾向于函数式编程风格，用不变状态编写无锁代码。Scala提供一个基于actor的消息传递模型，消...

(<http://blog.csdn.net/Winterto1990/article/details/52100714>)

## list字符串去重的三种方式 list去重 字符串去重

list字符串去重的三种方式求List< String >中元素去重，并且 求出去重后的个数 采用原始的for循环遍历 采用set集合的特点 采用Java8流处理方式 package sun.rain....

(<http://blog.csdn.net/sunrainamazing/article/details/71597561>)

## Java8 Stream流操作在用户系统中的妙用



lvshaorong 2016年07月04日 16:06 13034

本文主要介绍了Java8 新api Stream的使用案例。实现排序，去重，对比，筛选，分组，收集，聚集等等功能。主要包含sort()函数，distinct()函数，map()函数，collect()...

(<http://blog.csdn.net/lvshaorong/article/details/51810288>)

## Scala深入浅出实战经典：34,对List进行高效的排序和倒排序代码实战

Scala深入浅出实战经典：34,对List进行高效的排序和倒排序代码实战

sd637 2015年09月19日 13:39 264

(<http://blog.csdn.net/sd637/article/details/48574571>)

## [scala--基础]--Java和Scala容器的转换



high2011 2016年08月14日 15:33 5533

Java和Scala容器的转换 和Scala一样，Java同样提供了丰富的容器库，Scala和Java容器库有很多相似点，例如，他们都包含迭代器、可迭代结构、集合、映射和序列。但是...

(<http://blog.csdn.net/high2011/article/details/52204625>)

## Scala深入浅出实战经典：35,List的map、flatMap、foreach、filter操作代码实战

Scala深入浅出实战经典：35,List的map、flatMap、foreach、filter操作代码实战

sd637 2015年09月19日 13:41 960

(<http://blog.csdn.net/sd637/article/details/48574587>)

## scala剖析PriorityQueue，权值的使用



cjuexuan 2015年11月30日 20:23 1063



基于堆实现的优先级队列：PriorityQueue创建：new PriorityQueue()(implicit ord:Ordering[A])这里涉及到Ordering特质，看一个demoimpo...

(<http://blog.csdn.net/cjuexuan/article/details/50118135>)

---

## Scala学习笔记02【数组、列表、元组、集合和映射】

1、使用类型参数化数组【Array】Scala使用new实例化对象或类实例。 当在Scala里实例化对象，可以使用值和类型把它参数化：parameterize。参数化的意思是在你创建实例的时候” ...



 y396397735 2015年10月02日 18:23  956

(<http://blog.csdn.net/y396397735/article/details/48861671>)

---

## Scala集合，序列(可变和不可变List),List各种函数的使用,不可变Set和可变Set,Map

1．集合Scala的集合有三大类：序列Seq、集Set、映射Map，所有的集合都扩展自Iterable特质 在Scala中集合有可变（mutable）和不可变（immutable）两种类型，imm...



 toto1297488504 2017年06月28日 02:56  2714

(<http://blog.csdn.net/toto1297488504/article/details/73825623>)

---

## java8 lambda小试牛刀，利用Stream把list转map,并将两个list的数据对象合并起来



java8 lambda小试牛刀，利用Stream把list转map,并将两个list的数据对象合并起来

 liangrui1988 2017年03月17日 16:07  9485

(<http://blog.csdn.net/liangrui1988/article/details/62889900>)

---

## scala数据结构之List列表

 lyzx\_in\_csdn 2017年12月12日 08:48  19

XML Code 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ...

([http://blog.csdn.net/lyzx\\_in\\_csdn/article/details/78778519](http://blog.csdn.net/lyzx_in_csdn/article/details/78778519))