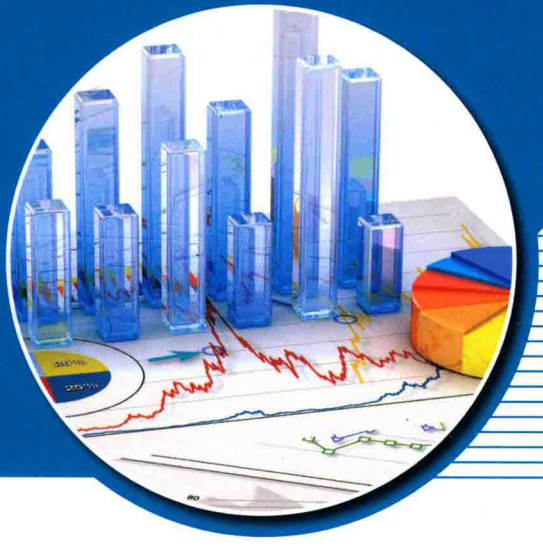
金融市场与风险管理系列教材

分布式统计计算

冯兴东 著

Distributed Computing in Statistics



上海财经大学出版社

金融市场与风险管理系列教材

分布式统计计算

冯兴东　著

上海财经大学出版社

图书在版编目(CIP)数据

分布式统计计算/冯兴东著.—上海:上海财经大学出版社,2018.4
金融市场与风险管理系列教材
ISBN 978-7-5642-2969-6/F·2969

Ⅰ.①分… Ⅱ.①冯… Ⅲ.①统计数据—分布式数据处理—高等学校—教材 Ⅳ.①O212②TP274

中国版本图书馆 CIP 数据核字(2018)第 040382 号

□ 责任编辑 台啸天
□ 封面设计 张克瑶

FENBUSHI TONGJI JISUAN

分布式统计计算

冯兴东 著

上海财经大学出版社出版发行
(上海市中山北一路 369 号 邮编 200083)
网 址:http://www.sufep.com
电子邮箱:webmaster @ sufep.com
全国新华书店经销
上海崇明裕安印刷厂印刷装订
2018 年 4 月第 1 版 2018 年 4 月第 1 次印刷

787mm×960mm 1/16 8.5 印张 181 千字
定价:39.00 元

前　言

人类各项科学技术的发展带来了海量数据，大数据的概念铺天盖地。统计学这一专注于数据分析的学科理应适应这一时代的变革和发展。显然大数据带给统计学的冲击是全方位的，不只局限于理论或者计算。国际上众多统计学家都在思考统计学在大数据时代应该扮演的角色。因此，提高统计学专业的学生相关计算机编程能力刻不容缓。在这一背景下，上海财经大学统计与管理学院开设了专业统计学硕士“数据科学与商务统计”方向，力图增强相关硕士生从事大数据分析的计算能力以及分析商务数据（包括营销数据、信用数据等）的应用能力。在这一指导思想下，学院还开设了一系列相关课程。本教材就是针对该专业方向的“分布式统计计算”课程。这门课程主要向学生介绍分布式计算的思想以及在统计学上的应用，将统计学传统方法和分布式计算方法相结合，通过不同的统计学问题来强化学生的分布式统计计算的编程能力和对统计计算问题的理解。

目前，有不同的计算平台来实现分布式计算，本教材将会依托 Apache Spark 来作为介绍课程内容的计算机语言平台。统计学专业的学生比较熟悉一些结构化的矩阵计算语言，比如 R 和 Matlab。而 Apache Spark 计算平台是基于 Scala 语言开发，可处理非结构化的数据，因此，与统计学专业方向的学生熟悉的编程环境有着极大不同。我们将通过介绍该平台上的 Breeze 软件包来让读者熟悉如何结合矩阵计算和分布式计算，从而帮助大家尽快入门。此外，我们还介绍了传统统计计算的方法在分布式计算平台 Apache Spark 上的实现方法，努力在统计计算和分布式计算之间搭建起一座桥梁。

目 录

1 Apache Spark 简介

对数据分析而言，有几个事实必须要严肃应对：第一，绝大多数成功的数据分析案例都需要可靠的数据预处理，尤其对于大规模的数据，合格的预处理必不可少。第二，迭代重复是数据科学的基础步骤，也就是说，我们需要不断迭代使用数据集进行建模分析。第三，即使一个成功的建模过程已经结束，数据分析任务也仍未完成。面对非专业客户，我们不能只满足于提供一个模型系数之类的东西。在真实的应用场景中，数据科学家需要提供真实的决策依据，需要追踪模型的运行情况并想方设法提高其执行效率和预测精度等。

对于探索性的数据分析而言，R 语言具有强大的软件包，可以帮助我们初步分析数据。然而，当我们确定算法之类的事情之后，我们往往通过 C++或者 Java 等语言来加以实现。这是由于 R 语言在运算上的低效且和真正商业平台间融合的困难，而 C++或者 Java 并不适合探索性的分析。因此，一种能够简化建模并能和运行系统良好契合的分析框架就会非常有用。本章中，我们将介绍这一种框架平台：Apache Spark。

1.1 Apache Spark 的历史与现状

Apache Spark 是一种开源软件。这款软件起源于 2009 年美国加州大学伯克利分校 RAD 实验室的一个研究项目。RAD 实验室后来更名为著名的 AMPLab。在这个项目中，研究人员发现 Hadoop MapReduce 在处理循环类型和交互较多的计算任务时比较耗时耗力，因此，他们决定自己设计一个计算平台来充分利用内存提高运算效率和纠错率。这款软件从 2010 年对公众开源，并在 2013 年转移至 Apache Software Foundation，并成为一款被公共社区共同维护开发的顶尖项目。这一公共开发社区由超过 200 个开发人员和超过 50 个公司组成。事实上，Apache Spark 可能是首款能让数据科学家实现分布式计算的开源软件。

Apache Spark 自诞生不久，就在一些运算任务上比 Hadoop MapReduce 快 10～20 倍，目前号称比 Hadoop MapReduce 要快 100 倍。我们在表 1.1 中展示了它们在执行排序任务时的一个简单比较。

表 1.1　　运算效率比较

	Hadoop MapReduce	Spark	Spark 1PB
数据量	102.5 TB	100 TB	1 000 TB
释放时间	72 mins	23 mins	234 mins
节点个数	2 100	206	190
核数	50 400(物理)	6 592(虚拟)	6 080(虚拟)
集群磁盘吞吐量	3 150 GB/s	618 GB/s	570 GB/s
Sort Benchmark Daytona 规则(通用目的排序)	是	是	否
网络	专用数据中心 10 Gbps	虚拟(EC2)10 Gbps	虚拟(EC2)10 Gbps
排序效率	1.42 TB/min	4.27 TB/min	4.27 TB/min
每节点排序效率	0.67 GB/min	20.7 GB/min	22.5 GB/min

Apache Spark 可以很方便地利用 Scala，Python，甚至 R 语言来启动。Spark 主要由几个重要的程序库组成：SQL，MLlib，GraphX 以及 Spark Streaming。SQL 用来处理表格类型的结构化数据，和 Hive 数据库可以兼容使用；MLlib 实现了一些诸如聚类、分类、线性回归等统计学习功能；GraphX 可以实现一些图模型，从而帮助大家分析网络数据；Spark Streaming 可以用来分析流数据。从 Apache Spark 1.4 版本开始，SparkR 被包含在安装包之中，我们实际上可以通过熟悉的 R 语言环境来实现 Spark 分布式计算，本书中的所有程序都在 Apache Spark 1.6 版本之上测试可用。

1.2　安装和运行 Apache Spark

Apache Spark 既可以在单机上安装，也可以在计算机集群上安装。在单机上运行的时候，所有的 Spark 进程都在同一个 Java 虚拟机(JVM)上运行。单机版可以方便大家的学习、程序开发和调试以及测试等。当然，在实际的运行过程中，我们也可以使用单机版 Spark 在一台计算机上利用多核进行运算。单机版可以去 Apache Spark 官方网页(http://spark.apache.org/downloads.html)下载。Spark 需要在一个 Hadoop(用于文件的分布式管理，HDFS)平台上运行，因此我们在安装单机版的时候，需要下载一个整合了 Hadoop 平台的版本。需要注意的是，想要成功运行 Spark，我们还需要在计算机上安装 Java Runtime Environment(JRE)或者 Java Development Kit(JDK)。从官方网站下载下

来的是一个压缩文件，无论是在 Linux 系统下还是在 Windows 系统下，我们都只需要简单地将其解压。我们以 Linux 为例，可以运行以下命令来解压：

```
>tar xfvz spark-1.6.0-bin-hadoop2.6.tgz
```

然后进入目录：

```
>cd spark-1.6.0-bin-hadoop2.6/bin
```

我们可以通过运行下面的程序来检验 Spark 程序：

```
>run-example org.apache.spark.examples.SparkPi
```

如果要通过 Scala 启动 Spark，我们可以运行以下命令：

```
>spark-shell
```

在经过一段时间之后，就可以进入如图 1.1 所示的界面。这个时候，我们就可以输入相应命令进行编程。我们还可以通过运行比如 SparkR 来进入 R 语言环境实现 Spark 编程。

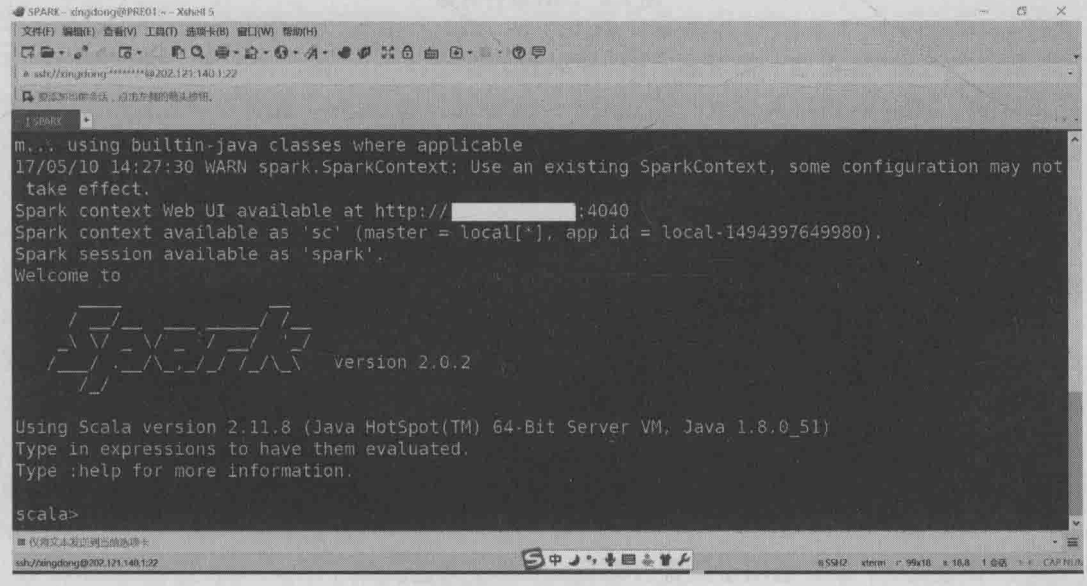


图 1.1　软件界面

对于 Spark 计算集群(Cluster),我们需要使用诸如 SSH 之类的软件通过计算机终端连入系统,比如我们可以使用免费的 xshell 软件(http://www.netsarang.com/download/down_ xsh.html)来实现连接,见图 1.2。一个 Spark 计算集群由两种进程组成,即主程序(Driver program)和执行程序(Executor)。在计算机集群中,主程序和各个执行程序会在不同的计算节点(Node)上运行。

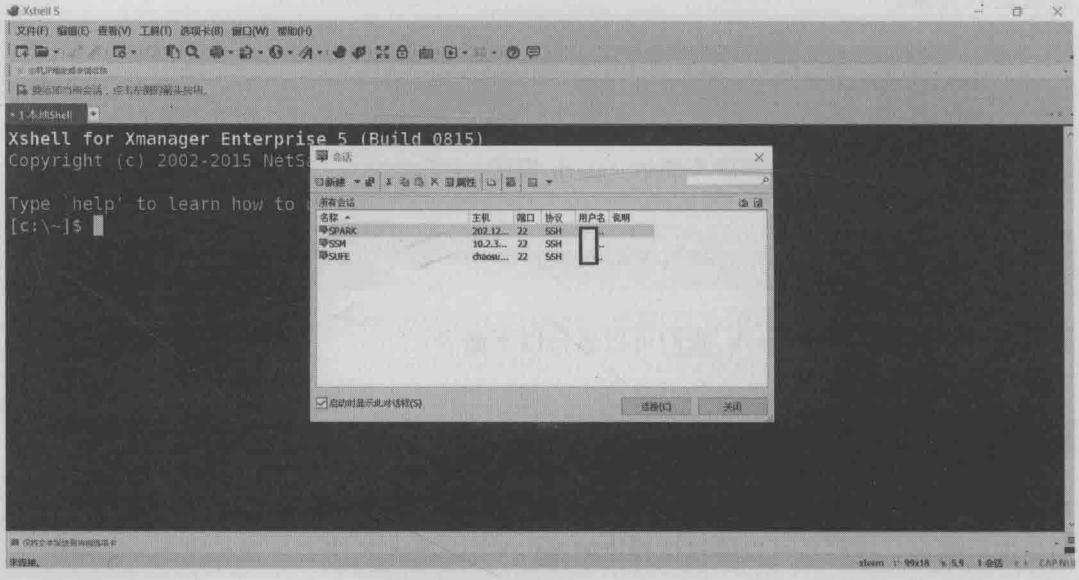


图 1.2 SSH 软件界面

需要注意的是,由于系统资源有限,如果我们在登录 Apache Spark 系统的时候对于资源分配不加约束,将可能导致一部分用户远程登录之后,另外一部分用户无法登录的问题,因此我们可以用以下语句来启动 Apache Spark 系统(见图 1.2):

```
>spark-shell--conf spark.port.maxRetries=100--master spark://x.x.x.x:7077--total-executor-cores 2
```

通过上面的语句来启动 Apache Spark 系统的时候,最大尝试的次数是 100 次,"x.x.x.x"是主机的 ip 地址(由系统管理员提供),"7077"是连接接口,"--total-executor-cores 2"表示申请分配两个核用于该用户的计算。

实际上,在启动 Spark 的时候,可以设置若干启动参数,如:

- total-executor-cores

  参数说明:该参数用于设置启动 Spark 作业对总共用于 Executor 进程的 CPU 数量,仅限于 Spark Alone、Spark on Mesos 模式。

- num-executors NUM
  参数说明：该参数用于设置启动的 Executor 进程的数量，默认是 2 个，仅限于 Spark on Yarn 模式。
- executor-cores
  参数说明：该参数用于设置每个 Executor 进程使用的 CPU 个数，缺失值为 1，仅限于 Spark on Yarn 模式。
- executor-memory
  参数说明：该参数用于设置每个 Spark Executor 进程的内存大小。
- executor-cores
  参数说明：该参数用于设置每个 Spark Executor 进程的 CPU core 数量。
- driver-memory
  参数说明：该参数用于设置 Spark Driver 进程的内存大小。

1.3 Apache Spark 编程简介

Apache Spark 来源于 Scala 语言，因此，在介绍 Apache Spark 之前，我们会稍微花点篇幅简单介绍一下 Scala 编程语言，这样可以帮助本书的读者对 Apache Spark 的语言规则有更深的理解。

1.3.1 Scala 语言

目前，绝大部分大数据研究平台，分布式平台都是建立在 JVM 平台基础上的。JVM 平台所对应的语言主要有两个，即 Java 和 Scala。本书中分布式计算基于的 Spark 平台就是利用 Scala 编写完成的。Scala 可以调用 Java 语言编写的包，这也就是为什么在 Spark 平台上可以调用 hadoop 文件管理系统。

相比较 Java 语言支持面向对象，Scala 是一种面向对象和函数式的语言，首先在 Scala 中一切都是对象，并且对象皆有方法。这样的结构使得这种语言可以更好地胜任大规模的项目，便于模块化，易于管理。其次，作为一种函数语言，函数可以出现在编程语句的任何地方，包括可以作为参数进行传递，类似编程语言有 R 语言。同时，Scala 语言也是一种面向对象的编程语言，类似编程语言包括 Java，C++等。本书只对 scala 做简要介绍，向大家介绍基本的处理数据所用到的变量和类型，以及一些表达式和条件式，紧接着会更深一步地和大家介绍函数和类，本书有关 Scala 语言的语法讲解及实例可参见《Scala 编程手册》(Scala Cookbook)一书，有兴趣的同学可以阅读这本书并进行实际演练。

- **Scala 开发环境配置**

Intellij IDEA 是一个在 Java/Scala/Groovy 开发中很流行的 IDEA。经过使用发现，在 IDEA 上面直接写 Scala，并调用单机版 Spark 进行直接调试给实际开发带来了诸多

便利，并且这一 IDEA 支持程序直接打包成 jar，也使得我们可以更方便地在集群中运行。

若要完成环境的搭建，我们首先需要如下四个条件：

1. JDK 安装。前往 Oracle 官方网站下载安装，并在 command 命令行窗口确认 Java-version 可以返回版本号，否则要在系统环境变量设置中添加 Java 到路径中。

2. Scala 下载安装。前往官网(http://www.scala-lang.org/)下载并安装。同样也需要确认环境变量已配置妥当。

3. Spark 源代码下载。前往官方网站(http://spark.apache.org/downloads.html)下载。

4. Intellij IDEA 下载安装。网站(https://www.jetbrains.com/idea)上可以下载免费的 community 版本。

打开 Intellij，在首界面上右下角选择 Configure-Plugins，在弹出页面中输入 Scala，安装 jetbrains 提供的连接 scala 和 intellij 的插件，若已下载则可以通过 Install plugin from dist ...选项从本地导入。

安装好 Scala 插件后，我们在首页选择 Creat new project，由此建立一个 Scala 的项目，在建立过程中注意在 Project SDK 中选择我们配置好的 JDK，Scala SDK 也要在下拉列表中选择对应的 sdk。

建立好新的项目之后，我们需要加载一些集合环境，选择 File - Project Structure - Project Settings - Libraries，点击加号选择 Java，选择我们下载好的 Spark 源码中的 spark-assembly-x.x.x-hadoop x.x.x.jar 文件。这时返回我们项目的页面，可以发现左侧 External Libraries 中此时有三个东西，第一个是关于 jdk 的，第二个是 scala-sdk，第三个是 spark-assembly 压缩包。至此，我们完成了基本配置，接下来开始测试写代码。

在左侧 src 上右击鼠标选择新建 scala class，类型选择 Object，如果发现找不到 scala class，则说明 src 这个文件夹的属性不对，一定要确认 src 被选中为 Sources 文件夹属性。这一属性在刚刚的 Project Structure 中的 Modules 中对应项目的 sources 选项卡中，选中 src 文件夹，单击上面的 Sources 按钮即可，这时在右侧可以看到这一文件夹变为蓝色显示。如果当前的 Module 中 src 下面有很多文件夹，那么我们选择在 scala 文件夹下建立新的类。

若想把我们写好的项目打包，首先我们要完成一些基本设置，在 Project Structure - Project Settings 中选择 Artifacts，选择添加 JAR-From modules with dependencies ...，在弹出的 Create JAR from Modules 中配置好对应的 Main Class，选择 extract to the target JAR。全部配置完成后，选择 OK 保存配置。

在 Build - Build Artifacts ...中选择要打包的项目，点击 build，如果没有特殊设置，压缩完成的包会存放在项目的 out-artifacts 中。

接下来我们讲解下如何配置 IDEA 才可以使我们可以直接连接单机的 Spark 平台运行测试代码。在 Run - Edit Configurations 中左侧选择对应的项目，在 Configuration 选项卡中选择对应的 main 函数，并在 vm option 中填写-Dspark.master＝local，并在 Program arguments 添加参数，也就是 main 函数里面用 args 接收到的参数。至此，在 IDEA 中单击 Run 可以直接运行基于 Spark 平台的代码，所有的输出都会显示在 IDEA 中。

· 值和变量

值(value)是不可变的，有对应类型的存储单元。变量(variable)表示一个唯一的标识符，对应一个已分配或保留的内存空间，这一空间的内容是动态的(variable)。其各自的声明方式为：

```
val<identifier>[: <type>]=<data>
var<identifier>[: <type>]=<data>
```

需要注意的是，如果我们在变量建立之初对变量类型有指定，那么在后面重新赋值时的赋值类型只能是指定的类型或指定类型的子类型。在实际项目中，由于 Spark 会涉及网络中大量的数据传输，模块间的通信，所以建议尽量使用值，避免错误的更改。值得注意的是，Scala 语言中如果某种方法不适用于一种类型，这种类型会自动进行隐式转换，转换成配合此方法的类型。

```
0.to(5)
```

我们知道 Int 数据类型是没有 to 的函数方法的，这里把 Int 自动转换成 Range.Integer 的对象类型。具体的核心数据类型见表 1.2。具体一些运算操作同其他语言大致相同，若要完成一些复杂的数学运算，如矩阵运算等，则需要引用相关的包。如：

```
import scala.math._
min(20,4)
```

· 表达式和条件式

表达式是返回一个值的代码单元，多个表达式用大括号即可构造一个表达式块，值得注意的是，表达式有自己的作用域，可以作用于所属表达块中的局部(或全局)值和变量。表达块中最后一个表达式即为整个表达式块的返回值。

- if ... else 表达式块

```
if (<Boolean expression>)<expression>
```

if ... else 块是编写条件逻辑的一种很简单的方法，并且这种方法的结果可以被直接赋值给变量，如表 1.2 所示。

表 1.2　　核心数据类型

类型名	描　　述	大小(字节)	取值范围
Byte	有符号整数	1	[−127,128]
Short	有符号整数	2	[−32 768,32 767]
Int	有符号整数	4	$[-2^{31},2^{31}-1]$
Long	有符号整数	8	$[-2^{63},2^{63}-1]$
Float	有符号浮点数	4	n/a
Double	有符号浮点数	8	n/a
Any	所有类型的根		
AnyVal	Scala 中所有值类型的根		
AnyRef	所有引用(非值)类型的根		
Nothing	所有类型的子类		
Null	所有指示 null 值得 AnyRef 类型的子类		
Char	Unicode 字符		
Boolean	true 或 false		
String	字符串		
Unit	指示没有值		

```
scala>val A_age=15; val B_age=20
scala>val Older_age=if (A_age>B_age) A_age else B_age
```

若 if 表达式要执行的有很多条语句，可以用大括号来构建表达式块。

· **循环**

循环是一个基于表达式的控制结构，本书介绍 for 循环、While 循环和 Do/While 循环。

```
for (<identifier><-<iterator>) [yield][<expression>]
```

for 循环中比较关键的有迭代器哨位和嵌套迭代器，迭代器哨位即为迭代器增加一个 if 表达式，嵌套迭代器即增加一个额外的迭代器，如：

```
scala>for {t<-0 until 4 if t%4==0} println(t)
0
scala>for {x<-0 until 3
     | y<-0 until 3}
     | {println(x* y)}
0
0
0
0
1
2
0
2
4
```

while 循环和 do/while 循环会重复一个语句，直到布尔值表达式返回 false。在实际应用中，这一循环方法并没有 for 循环那么常用，这里给出标准格式。

```
while (<Boolean expression>) statement
```

· 函数

函数（又称方法，即面向对象版本的函数）便于我们可以重复实现某一特定功能。函数式编程语言特别强调创建可重用、可组合的函数，可以帮助开发人员共同组建一个易维护的大型项目。当然，对于初学者而言，一个最明显的优势就是使你编写的代码更短，可读性更强。在标准函数式编程方法论中，建议编写者尽可能地构建纯函数，以便于最大可能地发挥优势，纯函数是指：

-有一个或多个输入参数；

-只使用输入参数完成计算；

-对于相同的输入总返回相同的值；

-不使用影响函数之外的任何数据；

-只返回一个值；

-不受函数之外的任何数据的影响。

这种结构的函数无状态，与外部数据正交，所以更为稳定。这里默认大家均掌握了基本的函数编程知识，只对 Scala 的语法做出说明。

```
def<identifier>(<identifier>: <type>[, ... ]): <type>=<expression>
```

在具体创建中，这些函数的函数体基本由表达式或表达式块组成，并且最后一行将成为函数的返回值。若需要在函数的表达式块结束前退出并返回一个值，可以使用 return 来指定返回值并退出。如果有一个函数，没有显式的返回类型，则编译器会推导出这个函数的返回类型为 Unit，表示没有值。这里要给大家介绍一种比较特殊的函数，叫作递归函数，因为它们为迭代处理数据结构或计算提供了一种很好的方法，而且不必使用可变的数据，因为每个函数调用自己的栈来存储函数参数。如：

```
scala>def power(x: Int, n: Int): Long={
     | if (n>=1) x* power(x, n-1)
     | else 1
     | }
power: (x: Int, n: Int)Long
scala>power(2,8)
res0: Long=256
```

这一结构有时会带来“栈溢出”的问题，在一些情况下可以用函数注解方法来标志一个函数将完成尾递归优化来解决这一问题，即@annotation.tailrec。这一改进只能在最后一个语句是递归调用的函数才可以使用。当我们需要在一个方法中重复某个逻辑，但把它作为外部方法又没有太多意义时，可以选择嵌套函数。这里建议大家从目的和方法方面比较 Scala 或 Python 中高阶函数学习，更利于大家掌握比较高阶的代码编写技巧。在实际定义函数参数时，若指定了部分参数的默认值，则要注意实际调用中的参数调用顺序。以及当我们要设置调用可变数目的参数时，可以用 vararg 参数来解决这一问题。正如我们之前所说的，函数我们有时也把它称为方法，方法是类中定义的一个函数，这个类的所有实例都可以调用这一方法，具体调用的做法就是使用中缀点记法。还有一些更为复杂的结构，如高阶函数、Lambda 表达式、柯里化、偏函数等，若正确运用这些复杂的结构，可以使得我们的代码更加高效，更具有可读性。

· 类

类是面向对象语言的核心构件，普通的类可以根据需要实例化多次，每一个实例都有自己的数据初始化，利用继承可以实现扩展，多态使得子类可以代表父类，封装则使得我们可以更好地管理类的外在表现。我们可以按需求参考下面的语法定义类。

```
class<identifier>[type-parameters]
                 ([var|val]<identifier>: <type>=<expression>[, ... ])
                 {extends<identifier>[type-parameters](<input parameters>)}
                 [{fields and methods}]
```