大数据技术丛书

Spark

入门与大数据分析实战

迟殿委　李　超　著

清華大學出版社
北京

内 容 简 介

本书基于 Spark 3.3.1 框架展开，系统介绍 Spark 生态系统各组件的操作，以及相应的大数据分析方法。本书各章节均提供丰富的示例及其详细的操作步骤，并配套示例源码、PPT 课件和教学大纲。

本书共分 11 章，内容包括 Scala 编程基础、Spark 框架全生态体验、Spark RDD、Spark SQL、Kafka、Spark Streaming、Spark ML、Spark GraphX、Redis 等技术框架和应用，并通过广告点击实时大数据分析和电影影评大数据分析两个综合项目进行实战提升。

本书适合 Spark 框架初学者，既可以作为大数据分析技术、大数据应用开发工程师的查询手册，也可以作为高等院校或高职高专计算机技术、软件工程、数据科学与大数据科学、智能科学与技术、人工智能等专业大数据课程的教材。

图书在版编目（CIP）数据

Spark 入门与大数据分析实战 / 迟殿委，李超著. －北京：清华大学出版社，2023.6
（大数据技术丛书）
ISBN 978-7-302-63798-1

Ⅰ. ①S… Ⅱ. ①迟… ②李… Ⅲ. ①数据处理软件－教材 Ⅳ. ①TP274

中国国家版本馆 CIP 数据核字（2023）第 105797 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　张：**16.25　　**字　　数：**438 千字
版　　次：2023 年 7 月第 1 版　　**印　　次：**2023 年 7 月第 1 次印刷
定　　价：79.00 元

产品编号：102832-01

前　言

如今大数据技术已广泛应用于金融、医疗、教育、电信、政府等领域，各个行业都积累了大量的历史数据，并不断产生大量新数据，数据计量单位出现 PB、EB、ZB、YB，甚至 BB、NB、DB。大数据的处理方式与传统数据不同，需要通过分布式存储和分布式运算来实现，由此也催生了优秀的大数据处理框架和生态组件。Spark 的特色在于它首先为大数据应用提供了一个统一的平台。从数据处理层面看，模型可以分为批处理、交互式、流处理等多种方式；而从大数据平台层面看，已有成熟的 Hadoop、Cassandra、Mesos 以及其他云的供应商。Spark 整合了主要的数据处理模型，并能够很好地与现在主流的大数据平台集成。

许多大型互联网公司，如谷歌、阿里巴巴、百度、京东等都急需掌握大数据技术的人才，因此大数据相关人才出现了供不应求的状况。Spark 作为继 Hadoop 之后的下一代大数据处理引擎，经过飞跃式发展，现已成为大数据产业中的一股中坚力量：RDD 模型具有强大的表现能力，并通过不断完善自己的功能而逐渐形成了一套自己的生物圈，提供了全栈（full-stack）的解决方案，其中主要包括 Spark 内存中批处理、Spark SQL 交互式查询、Spark Streaming 流式计算、GraphX 图计算和 Spark ML 机器学习算法库。

关于本书

本书基于 Spark 3.3.1 新版本展开，符合企业目前的开发需要。本书全面讲解 Spark 大数据技术的相关知识和实战应用，内容包括 Scala 编程基础、Spark 框架全生态体验、Spark RDD、Spark SQL、Spark Streaming、Kafka、Spark GraphX、Spark ML、Redis 等技术框架及其应用，并通过广告点击实时分析和电影影评分析两个大数据分析综合项目进行实战提升，夯实 Spark 大数据分析的基础知识，提升开发技能。

本书重视实践操作开发，内容安排从框架搭建和开发环境安装、技术框架快速示例引入、技术框架详细案例讲解，到大数据分析综合项目实战提升等，将实战与理论知识相结合，从而加深读者对 Spark 框架应用的理解。

笔者是具有多年大数据分析和处理实战经验的高级工程师，书中融入了笔者多年的实战经验，讲解细致、内容丰富、示例清晰、语言通俗易懂，方便读者提高学习效率，保证学习质量。

配套示例源码、PPT 课件等资源下载

本书配套示例源码、PPT 课件、教学大纲，需要用微信扫描下边二维码获取。如果下载有问题或阅读中发现问题，请用电子邮件联系 booksaga@163.com，邮件主题为“Spark 入门与大数据分析实战”。

适合的读者

- Spark 框架初学者。
- 大数据分析技术人员。
- 大数据应用开发工程师。
- 高等院校或高职高专大数据课程的师生。

笔　者

2023 年 3 月

目　　录

第1章

Spark 开发之 Scala 编程基础

本章详细讲解 Scala 的语法，包括基础语法、函数、控制语句、函数式编程、模式匹配、类和对象、异常处理、Trait（特征）、文件 I/O。掌握本章内容，可以为后续学习 Spark 数据分析奠定编程基础。注意：开发 Spark 代码和应用程序使用的 Scala 版本需要与 Spark 要求的 Scala 版本一致。

本章主要知识点：

- Scala 环境的安装
- Scala 基础语法
- Scala 函数与控制语句
- Scala 面向对象与 Trait 特性
- 异常处理与 I/O 流

1.1 开发环境搭建

可以在 IDEA 集成开发环境中使用 Scala、Java、Python 开发 Spark 应用。本节介绍使用 Scala 搭建 Spark 开发环境。

在启动 Spark（这里只是一个举例，具体可参看 2.2 节）时，会看到如下所示的信息，说明当前 Spark 所使用的 Scala 的版本为 2.13.8，JDK 的版本为 1.8.0。

```
Using Scala version 2.13.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_361)
Type in expressions to have them evaluated.
Type :help for more information.
```

开发 Scala 程序时，需要在本地安装 Scala 环境；如果使用 IDEA 开发环境，还需要在 IDEA 中安装 Scala 插件。安装 Scala 时，就像是安装 JDK 环境一样，也需要配置 Scala 的环境变量。

1. 安装 Scala

步骤 01 下载 Scala 安装包，后面 Spark 3.3.1 使用 Scala 2.13 版本，因此这里的下载地址如下：

```
    https://www.scala-lang.org/download/2.13.8.html
    https://downloads.lightbend.com/scala/2.13.8/scala-2.13.8.zip
```

步骤 02 解压并配置 SCALA_HOME 环境变量：

```
    SCALA_HOME=D:\programfiles\scala-2.13.8
    PATH=%SCALA_HOME%\bin
```

步骤 03 打开 CMD 命令行，查看 Scala 版本：

```
    C:\>scala -version
    Scala code runner version 2.13.8 -- Copyright 2002-2016, LAMP/EPFL and Lightbend,
Inc.
```

步骤 04 运行 scala 命令，进入 Scala 命令行：

```
    D:\a>scala
    Welcome to Scala 2.13.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_361).
    Type in expressions for evaluation. Or try :help.
    scala> 1+1
    res0: Int = 2
```

2. 在 IDEA 中安装 Scala 插件

检查自己安装的 IDEA 的版本，并安装对应的 Scala 插件，如图 1-1 所示。

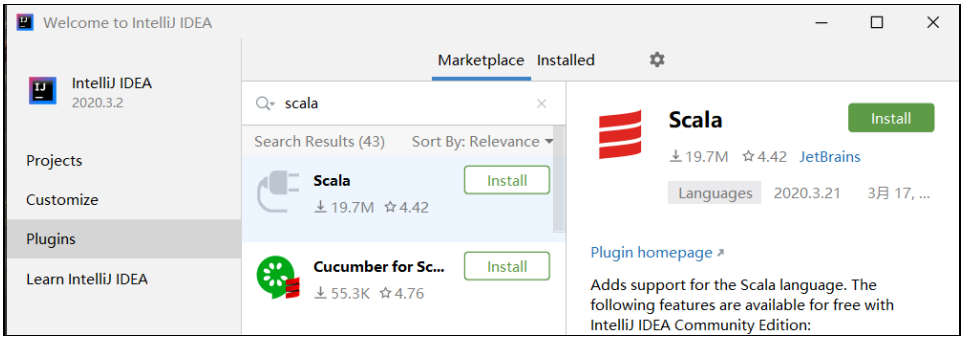

图 1-1

3. 开发 Spark 程序

步骤 01 在 IDEA 中创建项目模块 chapter 1，添加 Scala 的支持，如图 1-2 和图 1-3 所示。

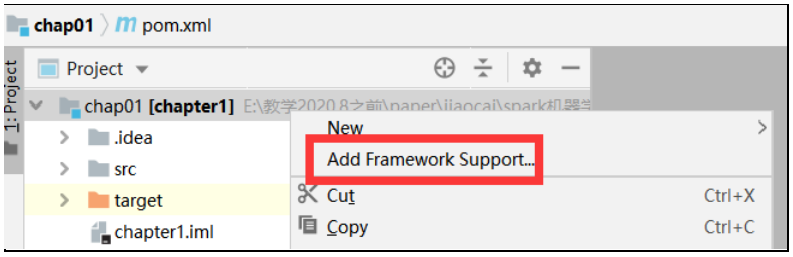

图 1-2

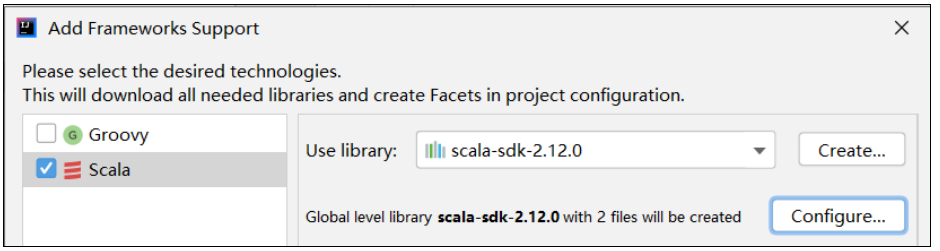

图 1-3

步骤 02 在 main 目录下创建 scala 目录，并设置为 resource root，如图 1-4 所示。

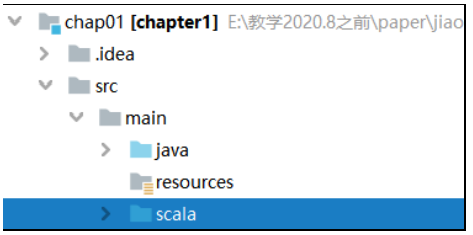

图 1-4

步骤 03 在 pom.xml 中添加依赖：

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core 2.12</artifactId>
    <version>3.3.1</version>
</dependency>
```

步骤 04 在 pom.xml 中添加编译 JDK 为 1.8 的插件（可选）：

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.0</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
    </configuration>
</plugin>
```

步骤 05 在 pom.xml 中添加打包 Scala 到 JAR 文件中的插件：

```
<plugin>
    <groupId>net.alchim31.maven</groupId>
    <artifactId>scala-maven-plugin</artifactId>
    <version>4.4.1</version>
    <executions>
        <execution>
            <goals>
                <goal>compile</goal>
                <goal>testCompile</goal>
```

```
            </goals>
          </execution>
      </executions>
    </plugin>
```

4. 测试 Scala 程序

步骤 01 打开 IDEA，创建一个 Scala 程序 HelloScala，代码如下：

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

步骤 02 在 IDEA 中直接运行 HelloScala 并输出结果：

```
Hello Scala
Process finished with exit code 0
```

步骤 03 直接使用 maven 打包，如图 1-5 所示。

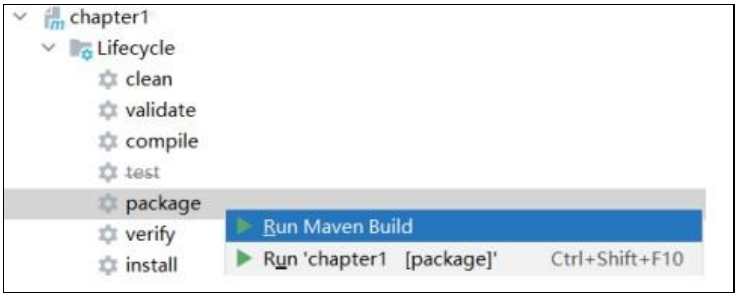

图 1-5

步骤 04 将 JAR 文件放到任意目录下，因为需要 Scala 包的支持，所以使用 java -cp 运行 HelloScala 时必须在命令行上添加 Scala 的支持包再运行：

```
D:\a>java -cp spark-1.0.jar;%SCALA_HOME%\lib\* cn.isoft.HelloScala
Hello Scala..
```

1.2 基 础 语 法

如果之前学过Java语言并了解Java语言的基础知识，那么我们就能很快学会Scala的基础语法。Scala 与 Java 之间有些小的区别，比如 Scala 语句末尾的英文分号（;）是可选的。我们可以认为 Scala 程序是对象的集合，通过调用彼此的方法来实现消息传递。下面将详细介绍 Scala 编程语言的基础语法和编程常识。

1. 注释

注释有单行注释和多行注释。

```
// 单行注释开始于两个斜杠
/*
 *  多行注释，如之前所见，看起来像这样
 */
```

2. 打印

打印分两种：强制换行的打印和没有强制换行的打印。

```
//打印并强制换行
println("Hello world!")
println(10)
// 没有强制换行的打印
print("Hello world")
```

3. 变量

通过 var 或者 val 来声明变量。val 声明是不可变的，var 声明是可变的。不可变变量非常有用。

```
val x = 10      // x 现在是 10
x = 20          // 错误：对 val 声明的变量重新赋值
var y = 10
y = 20          // y 现在是 20
```

4. 数据类型

Scala 与 Java 有着相同的数据类型，表 1-1 列出了 Scala 支持的数据类型。

表 1-1　Scala 支持的数据类型

数 据 类 型	说　　明
Byte	8 位有符号补码整数。数值范围为−128~127
Short	16 位有符号补码整数。数值范围为−32768~ 32767
Int	32 位有符号补码整数。数值范围为−2147483648~2147483647
Long	64 位有符号补码整数。数值范围为−9223372036854775808~9223372036854775807
Float	32 位 IEEE754 标准的单精度浮点数
Double	64 位 IEEE754 标准的双精度浮点数
Char	16 位无符号 Unicode 字符，区间值为 U+0000~ U+FFFF
String	字符序列
Boolean	true 或 false
Unit	表示无值，和其他语言中 void 等同。用作不返回任何结果的方法的结果类型。Unit 只有一个实例值，写成()
Null	null 或空引用
Nothing	Nothing 类型在 Scala 的类层级的最底端；它是任何其他类型的子类型
Any	Any 是所有其他类的超类
AnyRef	AnyRef 类是 Scala 里所有引用类（reference class）的基类

Scala 数据类型设置示例如下：

```
val z: Int = 10
val a: Double = 1.0
```

注意从 Int 到 Double 的自动转型，以下示例结果是 10.0，不是 10：

```
val b: Double = 10.0
```

布尔值：

```
true
False
```

布尔操作：

```
!true           // false
!false          // true
true == false   // false
10 > 5          // true
```

5. 运算符

使用运算符进行数学运算：

```
1 + 1   // 2
2 - 1   // 1
5 * 3   // 15
6 / 2   // 3
6 / 4   // 1
6.0 / 4 // 1.5
```

6. 字符串

Scala 的字符串被英文双引号引起来，不存在单引号字符串。
String 有常见的 Java 字符串方法，例如：

```
"hello world".length
"hello world".substring(2, 6)
"hello world".replace("C", "3")
```

也有一些额外的 Scala 方法，例如：

```
"hello world".take(5)
"hello world".drop(5)
```

改写字符串时留意前缀"s"：

```
val n = 45
s"We have $n apples" // => "We have 45 apples"
```

在要改写的字符串中使用表达式也是可以的：

```
    val a = Array(11, 9, 6)
    s"My second daughter is ${a(0) - a(2)} years old." // => "My second daughter is
5 years old"
    s"We have double the amount of ${n / 2.0} in apples." // => "We have double the
amount of 22.5 in apples."
    s"Power of 2: ${math.pow(2, 2)}" // => "Power of 2: 4"
```

添加"f"前缀对要改写的字符串进行格式化：

```
    f"Power of 5: ${math.pow(5, 2)}%1.0f" // "Power of 5: 25"
    f"Square root of 122: ${math.sqrt(122)}%1.4f" //”Square root of 122: 11.0454"
```

未处理的字符串，忽略特殊字符：

```
    raw"New line feed: \n. Carriage return: \r." // => "New line feed: n. Carriage
return: r."
```

一些字符需要转义，比如字符串中的双引号：

```
    "They stood outside the \"Rose and Crown\"" // => "They stood outside the "Rose
and Crown""
```

三个双引号可以使字符串跨越多行，并包含引号：

```
    val html = """<form id="daform">
    <p>Press belo', Joe</p>
    <input type="submit">
    </form>"""
```

1.3　函　　数

函数是一组一起执行一个任务的语句。我们可以把代码划分到不同的函数中。如何划分代码到不同的函数中是我们自己来决定，但在逻辑上，划分通常是根据每个函数要执行一个特定的任务来进行的。

Scala 有函数和方法，二者在语义上的区别很小：Scala 方法是类的一部分，而函数是一个对象，可以赋值给一个变量。换句话来说，在类中定义的函数即是方法。

我们可以在任何地方定义函数，甚至可以在函数内定义函数（内嵌函数），更重要的一点是 Scala 函数名可以使用这些特殊字符：+、++、~、&、-、--、\、/、:等。

1. 函数声明

Scala 函数声明格式如下：

```
    def functionName ([参数列表]) : [return type] { }
```

如果不写等于号和方法主体，那么方法会被隐式声明为“抽象”（abstract），于是包含它的类型也是一个抽象类型。

2. 函数定义

函数定义由一个 def 关键字开始，紧接着是可选的参数列表、一个英文冒号（:）、函数的返回类型、一个等于号（=），最后是函数的主体。

Scala 函数定义格式如下：

```
    def functionName ([参数列表]) : [return type] = {
       function body
       return [expr]
    }
```

其中 return type 可以是任意合法的 Scala 数据类型，参数列表中的参数可以使用逗号分隔。
以下函数的功能是对两个传入的参数进行相加并求和：

```
object add{
   def addInt( a:Int, b:Int ) : Int = {
      var sum:Int = 0
      sum = a + b
      return sum
   }
}
```

如果函数没有返回值，那么可以返回 Unit，这个关键字类似于 Java 的 void，示例如下：

```
object Hello{
   def printMe( ) : Unit = {
      println("Hello, Scala!")
   }
}
```

3. 函数调用

Scala 提供了多种函数调用方式。
函数调用方法的标准格式如下：

```
functionName( 参数列表 )
```

如果函数使用了实例的对象来调用，那么我们可以使用类似 Java 的调用格式（使用“.”号）：

```
[instance.]functionName( 参数列表 )
```

函数调用示例代码如下：

代码 1-1　TestFunc.scala

```
object TestFunc {
   def main(args: Array[String]) {
      println( "Returned Value : " + addInt(5,7) );
   }
   def addInt( a:Int, b:Int ) : Int = {
      var sum:Int = 0
      sum = a + b
      return sum
   }
}
```

执行以上代码，输出结果为：

```
Returned Value : 12
```

1.4 控 制 语 句

1. 控制语句变量的使用

Scala 对点和括号的要求非常宽松（注意，它们的规则是不同的），这有助于写出读起来像英语的 DSL（领域特定语言）和 API（应用编程接口）。测试代码如代码 1-2 和代码 1-3 所示。

代码 1-2　Test Foreach.scala

```
1 to 5
val r = 1 to 5
r.foreach( println )
r foreach println
```

执行以上代码，输出结果为：

```
1,2,3,4,5,1
2
3
4
5
```

代码 1-3　Test Foreach2.scala

```
(5 to 1 by -1) foreach ( println )
```

执行以上代码，输出结果为：

```
5,4,3,2,1,
```

2. while 循环

while 循环是运行一系列语句，如果条件为 true，就重复运行，直到条件变为 false。测试代码如代码 1-4 所示。

代码 1-4　TestWhile.scala

```
var i = 0
while (i < 10) { println("i " + i); i+=1 }
```

执行以上代码，输出结果为：

```
i 0
i 1
i 2
i 3
i 4
i 5
i 6
i 7
i 8
i 9
```

3. do while 循环

do while 循环类似 while 语句，区别在于判断循环条件之前，do while 循环先执行一次循环的代码块。测试代码如代码 1-5 所示。

代码 1-5 TestDoWhile.scala

```
var x = 0;
do {
   println(x + " is still less than 10");
   x += 1
} while (x < 10)
```

执行以上代码，输出结果为：

```
0 is still less than 10
1 is still less than 10
2 is still less than 10
3 is still less than 10
4 is still less than 10
5 is still less than 10
6 is still less than 10
7 is still less than 10
8 is still less than 10
9 is still less than 10
```

4. for 循环

for 循环允许编写一个执行指定次数的循环控制结构。测试代码如代码 1-6 所示。

代码 1-6 TestFor.scala

```
def main(args: Array[String]) {
   var a = 0;
   // for 循环
   for( a <- 1 to 10){
      println( "Value of a: " + a );
   }
}
```

执行以上代码，输出结果为：

```
value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
value of a: 10
```

5. 条件语句

Scala 的 if...else 语句通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。测试代码如代码 1-7 所示。

代码 1-7　Test If-else.scala

```
val x = 10
if (x == 1) println("yeah")
if (x == 10) println("yeah")
if (x == 11) println("yeah")
if (x == 11) println ("yeah") else println("nay")
println(if (x == 10) "yeah" else "nope")
val text = if (x == 10) "yeah" else "nope"
```

执行以上代码，输出结果为：

```
yeah
nay
yeah
```

6. break 语句

当在循环中使用 break 语句并执行到该语句时，就会中断循环并执行循环体之后的代码块。Scala 语言中默认是没有 break 语句的，但是在 Scala 2.8 版本后可以使用另外一种方式来实现 break 语句。

Scala 中 break 的语法格式如下：

```
// 导入以下包
import scala.util.control.
// 创建 Breaks 对象
val loop = new Breaks;
// 在 breakable 中循环
loop.breakable{
    // 循环
    for(...){
       ...
       // 循环中断
       loop.break;
   }
}
```

测试代码如代码 1-8 所示。

代码 1-8　TestBreak.scala

```
import scala.util.control._
object TestBreak {
  def main(args: Array[String]) {
    var a = 0;
    val numList = List(1,2,3,4,5,6,7,8,9,10);

    val loop = new Breaks;
    loop.breakable {
```

```
            for( a <- numList){
               println( "Value of a: " + a );
               if( a == 4 ){
                  loop.break;
               }
            }
         }
         println( "After the loop" );
      }
   }
```

执行以上代码，输出结果为：

```
Value of a: 1
Value of a: 2
Value of a: 3
Value of a: 4
After the loop
```

1.5 函数式编程

1. Array（数组）

Scala 数组声明的语法格式如下：

```
var z:Array[String] = new Array[String](3)
或
var z = new Array[String](3)
```

数组的元素类型和数组的大小都是确定的，所以当处理数组元素时，我们通常使用基本的 for 循环来遍历数组元素。

以下示例演示了数组的创建、初始化等处理过程。

代码 1-9　TestArray1.scala

```
object TestArray1 {
   def main(args: Array[String]) {
      var myList = Array(1.9, 2.9, 3.4, 3.5)
      // 输出所有数组元素
      for ( x <- myList ) {
         println( x )
      }
      // 计算数组所有元素的总和
      var total = 0.0;
      for ( i <- 0 to (myList.length - 1)) {
         total += myList(i);
      }
      println("总和为 " + total);
      // 查找数组中的最大元素
```

```
      var max = myList(0);
      for ( i <- 1 to (myList.length - 1) ) {
         if (myList(i) > max) max = myList(i);
      }
      println("最大值为 " + max);
   }
}
```

执行以上代码，输出结果为：

```
1.9
2.9
3.4
3.5
总和为 11.7
最大值为 3.5
```

2. List（列表）

List 的特征是其元素以线性方式存储，列表中可以存放重复对象。

以下列出了多种类型的列表：

```
// 字符串列表
val site: List[String] = List("mrchi 的博客", "Google", "Baidu")
// 整型列表
val nums: List[Int] = List(1, 2, 3, 4)
// 空列表
val empty: List[Nothing] = List()
// 二维列表
val dim: List[List[Int]] =
   List(
      List(1, 0, 0),
      List(0, 1, 0),
      List(0, 0, 1)
   )
```

对于 Scala 列表的任何操作都可以使用 head、tail、isEmpty 这 3 个基本操作来表达，示例如下：

代码 1-10　TestList.scala

```
object TestList {
   def main(args: Array[String]) {
      val site = "mrchi 的博客" :: ("Google" :: ("Baidu" :: Nil))
      val nums = Nil
      println( "第一网站是 : " + site.head )
      println( "最后一个网站是 : " + site.tail )
      println( "查看列表 site 是否为空 : " + site.isEmpty )
      println( "查看 nums 是否为空 : " + nums.isEmpty )
   }
}
```

执行以上代码，输出结果为：

```
第一网站是 : mrchi 的博客
最后一个网站是 : List(Google, Baidu)
查看列表 site 是否为空 : false
查看 nums 是否为空 : true
```

3. Set（集合）

Set 是最简单的一种集合。集合中的对象不按特定的方式排序，并且没有重复对象。

对于 Scala 集合的任何操作都可以使用 head、tail、isEmpty 这 3 个基本操作来表达，示例如下：

代码 1-11　TestSet.scala

```
object TestSet {
   def main(args: Array[String]) {
      val site = Set("mrchi 的博客", "Google", "Baidu")
      val nums: Set[Int] = Set()
      println( "第一网站是: " + site.head )
      println( "最后一个网站是: " + site.tail )
      println( "查看列表 site 是否为空: " + site.isEmpty )
      println( "查看 nums 是否为空: " + nums.isEmpty )
   }
}
```

执行以上代码，输出结果为：

```
第一网站是: mrchi 的博客
最后一个网站是: Set(Google, Baidu)
查看列表 site 是否为空: false
查看 nums 是否为空: true
```

4. Map（映射）

Map 是一种映射键对象和值对象的集合，它的每一个元素都包含一对键对象和值对象。

以下示例演示 key、values、isEmpty 这 3 个方法的基本应用。

代码 1-12　TestMap.scala

```
object TestMap {
   def main(args: Array[String]) {
      val colors = Map("red" -> "#FF0000",
                       "azure" -> "#F0FFFF",
                       "peru" -> "#CD853F")
      val nums: Map[Int, Int] = Map()
      println( "colors 中的键为: " + colors.keys )
      println( "colors 中的值为: " + colors.values )
      println( "检测 colors 是否为空: " + colors.isEmpty )
      println( "检测 nums 是否为空: " + nums.isEmpty )
   }
}
```

执行以上代码，输出结果为：

```
colors 中的键为: Set(red, azure, peru)
```

```
colors 中的值为: MapLike(#FF0000, #F0FFFF, #CD853F)
检测 colors 是否为空: false
检测 nums 是否为空: true
```

5. 元组

元组是不同类型的值的集合。与列表一样，元组也是不可变的，但与列表不同的是元组可以包含不同类型的元素。

元组的值是通过将单个的值包含在圆括号中构成的。例如：

```
val t = (1, 3.14, "Fred")
```

表示在元组中定义了 3 个元素，对应的类型分别为[Int, Double, java.lang.String]。

此外也可以使用以下方式来定义元组：

```
val t = new Tuple3(1, 3.14, "Fred")
```

可以使用 t._1 访问第一个元素，t._2 访问第二个元素，以此类推。元组的示例代码如下：

代码 1-13　TestTuple.scala

```
object TestTuple {
   def main(args: Array[String]) {
      val t = (4,3,2,1)
      val sum = t._1 + t._2 + t._3 + t._4
      println( "元素之和为: "  + sum )
   }
}
```

执行以上代码，输出结果为：

```
元素之和为: 10
```

6. Option

Option[T] 表示有可能包含值的容器，当然也可能不包含值。Scala Iterator（迭代器）不是一个容器，更确切地说它是逐一访问容器内元素的方法。Scala Option（选项）类型用来表示一个值是可选的（有值或无值）。

Option[T] 是一个类型为 T 的可选值的容器：如果值存在，那么 Option[T]就是一个 Some[T]；如果不存在，那么 Option[T] 就是对象 None 。

接下来我们来看一段代码：

```
// 虽然 Scala 可以不定义变量的类型，不过为了清楚些，还是把它显示地定义上
val myMap: Map[String, String] = Map("key1" -> "value")
val value1: Option[String] = myMap.get("key1")
val value2: Option[String] = myMap.get("key2")
println(value1) // Some("value1")
println(value2) // None
```

代码解释：

（1）在上面的代码中，myMap 是一个键的类型是 String、值的类型是 String 的 hash map，但

不一样的是它的 get()返回的是一个叫作 Option[String]的类别。

（2）Scala 使用 Option[String]来告诉我们：“我会想办法回传一个 String，但也可能没有 String 给你”。

（3）myMap 里并没有 key2 数据，因此 get()方法返回 None。

Option 有两个子类别，一个是 Some，一个是 None：当它回传 Some 的时候，代表这个函数成功地给了我们一个 String，而我们可以通过 get()函数拿到那个 String；如果它返回的是 None，则代表没有字符串可以给我们。示例代码如下：

代码 1-14　TestOption.scala

```
object Test {
   def main(args: Array[String]) {
      val sites = Map("余辉" -> "mrchi 的博客", "google" -> "www.google.com")
      println("sites.get( \"余辉\" ) : " +  sites.get( "余辉" )) //
Some(www.runoob.com)
      println("sites.get( \"baidu\" ) : " +  sites.get( "baidu" ))  //  None
   }
}
```

执行以上代码，输出结果为：

```
sites.get( "runoob" ) : Some(mrchi 的博客)
sites.get( "baidu" ) : None
```

也可以通过模式匹配来输出匹配值，示例代码如下：

代码 1-15　TestOption2.scala

```
object Test {
   def main(args: Array[String]) {
      val sites = Map("余辉" -> "mrchi 的博客", "google" -> "www.google.com")
      println("show(sites.get( \"余辉\")) : " +
        show(sites.get( "余辉")) )
      println("show(sites.get( \"baidu\")) : " +
        show(sites.get( "baidu")) )
   }
   def show(x: Option[String]) = x match {
      case Some(s) => s
      case None => "?"
   }
}
```

执行以上代码，输出结果为：

```
show(sites.get( "余辉")) : mrchi 的博客
show(sites.get( "baidu")) : ?
```

1.6　模 式 匹 配

1. 模式匹配

Scala 提供了强大的模式匹配机制，应用得也非常广泛。

一个模式匹配包含了一系列备选项，每个备选项都开始于关键字 case，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。

以下是一个简单的整型值模式匹配示例代码。

代码 1-16　TestMach.scala

```
object Test {
   def main(args: Array[String]) {
      println(matchTest(3))
   }
   def matchTest(x: Int): String = x match {
      case 1 => "one"
      case 2 => "two"
      case _ => "many"
   }
}
```

执行以上代码，输出结果为：

```
many
```

示例代码 match 对应 Java 里的 switch，但是写在选择器表达式之后，即选择器 match{备选项}。

match 表达式通过按照代码编写的先后次序尝试匹配每个模式来完成计算，只要发现有一个匹配的 case，剩下的 case 就不会继续匹配。

接下来我们来看一个不同数据类型的模式匹配示例代码。

代码 1-17　TestPattern.scala

```
object TestPattern {
   def main(args: Array[String]) {
      println(matchTest("two"))
      println(matchTest("test"))
      println(matchTest(1))
      println(matchTest(6))
   }
   def matchTest(x: Any): Any = x match {
      case 1 => "one"
      case "two" => 2
      case y: Int => "scala.Int"
      case _ => "many"
   }
}
```

执行以上代码，输出结果为：

```
2
many
one
scala.Int
```

代码解析：

第 1 个 case 对应整型数值 1；第 2 个 case 对应字符串值“two”；第 3 个 case 对应类型模式，用于判断传入的值是否为整型，相比使用 isInstanceOf 来判断类型，使用模式匹配更好；第 4 个 case 表示默认的全匹配备选项，即没有找到其他匹配时的匹配项，类似 switch 中的 default。

2. 样例类

使用了 case 关键字的类定义就是样例类（case classes），样例类是一种特殊的类，经过优化后用于模式匹配。以下是样例类的简单示例代码。

代码 1-18　TestPattern1.scala

```
object TestPattern1 {
   def main(args: Array[String]) {
    val alice = new Person("Alice", 25)
    val bob = new Person("Bob", 32)
    val charlie = new Person("Charlie", 32)
    for (person <- List(alice, bob, charlie)) {
    person match {
          case Person("Alice", 25) => println("Hi Alice!")
          case Person("Bob", 32) => println("Hi Bob!")
          case Person(name, age) =>
             println("Age: " + age + " year, name: " + name + "?")
       }
     }
   }
   // 样例类
   case class Person(name: String, age: Int)
}
```

执行以上代码，输出结果为：

```
Hi Alice!
Hi Bob!
Age: 32 year, name: Charlie?
```

1.7　类 和 对 象

1. 类的定义

类是对象的抽象，而对象是类的具体实例。类是抽象的，不占用内存，而对象是具体的，占用存储空间。类是用于创建对象的蓝图，是一个定义许多具有共性特征和行为的对象的软件模板。

Scala 中的类不声明为 public，一个 Scala 源文件中可以有多个类。我们可以使用 new 关键字来

创建类的对象，示例如下：

```
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}
```

代码解析：

示例中类定义了两个变量 x 和 y；还定义了一个方法 move，方法没有返回值。

Scala 的类定义可以有参数，称之为类参数，如上述示例中的 xc、yc，类参数在整个类中都可以访问。使用 new 来实例化类并访问类中的方法和变量的示例，如代码 1-19 所示。

代码 1-19　TestPoint.scala

```
import java.io._
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}
object TestPoint {
   def main(args: Array[String]) {
      val pt = new Point(10, 20);

      // 移到一个新的位置
      pt.move(10, 10);
   }
}
```

执行以上代码，输出结果为：

```
x 的坐标点: 20
y 的坐标点: 30
```

2. 继承

Scala 使用 extends 关键字来继承一个类。Scala 继承一个基类跟 Java 很相似，但需要注意以下几点：

（1）重写一个非抽象方法时必须使用 override 修饰符。

（2）只有主构造函数才可以往基类的构造函数里写参数。

（3）在子类中重写超类的抽象方法时，不需要使用 override 关键字。

接下来让我们来看个示例。

代码 1-20　TestInherit.scala

```
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}
class Location(override val xc: Int, override val yc: Int,
   val zc :Int) extends Point(xc, yc){
   var z: Int = zc
   def move(dx: Int, dy: Int, dz: Int) {
      x = x + dx
      y = y + dy
      z = z + dz
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
      println ("z 的坐标点 : " + z);
   }
}
```

代码解析：

示例中 Location 类继承了 Point 类，Point 称为父类（基类），Location 称为子类。override val xc 为重写了父类的字段。

继承会继承父类的所有属性和方法，Scala 只允许继承一个父类。示例代码如下：

代码 1-21　TestInherit2.scala

```
import java.io._
class Point(val xc: Int, val yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
   }
}
class Location(override val xc: Int, override val yc: Int,
   val zc :Int) extends Point(xc, yc){
```

```
    var z: Int = zc

    def move(dx: Int, dy: Int, dz: Int) {
      x = x + dx
      y = y + dy
      z = z + dz
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
      println ("z 的坐标点 : " + z);
    }
}
object Test {
    def main(args: Array[String]) {
      val loc = new Location(10, 20, 15);

      // 移到一个新的位置
      loc.move(10, 10, 5);
    }
}
```

执行以上代码，输出结果为：

```
x 的坐标点 : 20
y 的坐标点 : 30
z 的坐标点 : 20
```

Scala 重写一个非抽象方法时，必须用 override 修饰符。示例代码如下：

代码 1-22　TestInherit3.scala

```
class Person {
  var name = ""
  override def toString = getClass.getName + "[name=" + name + "]"
}
class Employee extends Person {
  var salary = 0.0
  override def toString = super.toString + "[salary=" + salary + "]"
}
object TestInherit1 extends App {
  val fred = new Employee
  fred.name = "Fred"
  fred.salary = 50000
  println(fred)
}
```

执行以上代码，输出结果为：

```
Employee[name=Fred][salary=50000.0]
```

1.8 异 常 处 理

Scala 的异常处理与其他语言（比如 Java）类似。Scala 可以通过抛出异常的方式来终止相关代码的运行，而不必通过返回值。

1. 抛出异常

Scala 抛出异常的方法和 Java 一样，使用 throw 方法。例如，抛出一个非法参数异常：

```
throw new IllegalArgumentException
```

2. 捕获异常

Scala 异常捕捉的机制与其他语言的处理方法一样，如果有异常发生，那么 catch 子句按次序捕捉。因此，在 catch 子句中，越具体的异常越靠前，越普遍的异常越靠后。如果抛出的异常不在 catch 子句中，该异常则无法处理，会被升级到调用者处。

捕捉异常的 catch 子句的语法与其他语言中的不太一样。在 Scala 里，借用了模式匹配的思想来做异常的匹配，因此，在 catch 的代码里是一系列 case 字句，如代码 1-23 所示。

代码 1-23　TestException.scala

```
import java.io.FileReader
import java.io.FileNotFoundException
import java.io.IOException
object Test {
   def main(args: Array[String]) {
      try {
         val f = new FileReader("input.txt")
      } catch {
         case ex: FileNotFoundException =>{
            println("Missing file exception")
         }
         case ex: IOException => {
            println("IO Exception")
         }
      }
   }
}
```

执行以上代码，输出结果为：

```
Missing file exception
```

catch 语句里的内容跟 match 里的 case 是完全一样的。由于异常捕捉是按次序的，如果把最普遍的异常 Throwable 写在最前面，则在它后面的 case 都捕捉不到，因此需要将它写在最后面。

3. finally 语句

finally 语句用于执行不管是正常处理还是有异常发生时都需要执行的步骤，如代码 1-24 所示。

代码 1-24　TestFinally.scala

```
import java.io.FileReader
import java.io.FileNotFoundException
import java.io.IOException
object TestFinally {
   def main(args: Array[String]) {
      try {
         val f = new FileReader("input.txt")
      } catch {
         case ex: FileNotFoundException => {
            println("Missing file exception")
         }
         case ex: IOException => {
            println("IO Exception")
         }
      } finally {
         println("Exiting finally...")
      }
   }
}
```

执行以上代码，输出结果为：

```
Missing file exception
Exiting finally...
```

1.9　Trait（特征）

Scala 的 Trait（特征）相当于 Java 的接口，但它比接口的功能还要强大，它还可以定义属性和方法的实现。

一般情况下 Scala 的类只能够继承单一父类，但是如果是 Trait 的话就可以继承多个，从结果来看就是实现了多重继承。

Trait 定义的方式与类类似，但它使用的关键字是 trait，示例如下：

```
trait Equal {
  def isEqual(x: Any): Boolean
  def isNotEqual(x: Any): Boolean = !isEqual(x)
}
```

代码解析：

以上 Trait 由两个方法组成：isEqual 和 isNotEqual。isEqual 方法没有定义方法的实现，isNotEqual 定义了方法的实现。

子类继承特征可以实现未被实现的方法，所以 Scala Trait 其实更像 Java 的抽象类。

特征的完整示例如代码 1-25 所示。

代码 1-25 TestTrait.scala

```
trait Equal {
  def isEqual(x: Any): Boolean
  def isNotEqual(x: Any): Boolean = !isEqual(x)
}
class Point(xc: Int, yc: Int) extends Equal {
  var x: Int = xc
  var y: Int = yc
  def isEqual(obj: Any) =
    obj.isInstanceOf[Point] &&
    obj.asInstanceOf[Point].x == x
}
object Test {
   def main(args: Array[String]) {
     val p1 = new Point(2, 3)
     val p2 = new Point(2, 4)
     val p3 = new Point(3, 3)
     println(p1.isNotEqual(p2))
     println(p1.isNotEqual(p3))
     println(p1.isNotEqual(2))
   }
}
```

执行以上代码，输出结果为：

```
false
true
true
```

1.10 文件 I/O

1. I/O 介绍

Scala 进行文件写操作直接使用的是 Java 中的 I/O 类（java.io.File），如代码 1-26 所示。

代码 1-26 TestFileWriter.scala

```
import java.io._
object TestFileWriter {
   def main(args: Array[String]) {
     val writer = new PrintWriter(new File("test.txt" ))
     writer.write("博客地址为 http://blog.csdn.net/mrchi")
     writer.close()
   }
}
```

执行以上代码，会在当前目录下生成一个 test.txt 文件，文件内容为“mrchi 的博客 http://blog.csdn.net/silentwolfyh”。

2. 从屏幕上读取用户输入

有时候我们需要接收用户在屏幕上输入的指令来处理程序，如代码 1-27 所示。

代码 1-27　TestScreenRead.scala

```
object Test {
   def main(args: Array[String]) {
      print("请输入博客地址: " )
      val line = Console.readLine
      println("谢谢，你输入的是: " + line)
   }
}
```

执行以上代码，屏幕上会显示如下信息:

```
请输入博客地址: http://blog.csdn.net/mrchi
谢谢，你输入的是: http://blog.csdn.net/mrchi
```

3. 从文件上读取内容

从文件读取内容非常简单，我们可以使用 Scala 的 Source 类及伴生对象来读取文件。代码 1-28 演示的是从“test.txt”（代码 1-26 中创建）文件中读取内容。

代码 1-28　TestFileRead.scala

```
import scala.io.Source
object TestFileRead {
   def main(args: Array[String]) {
      println("文件内容为:" )
      Source.fromFile("test.txt" ).foreach{
         print
      }
   }
}
```

执行以上代码，输出结果为：

```
文件内容为:博客地址为 http://blog.csdn.net/mrchi
```