

Life is short, you need Spark!



# 从零开始

不需要任何基础，带领您无痛入门 Spark

## 云计算分布式大数据 Spark 实战高手之路

王家林著

Spark 亚太研究院系列丛书 版权所有

伴随着大数据相关技术和产业的逐步成熟，继 Hadoop 之后，Spark 技术以其无可比拟的优势，发展迅速，将成为替代 Hadoop 的下一代云计算、大数据核心技术。

## 本书特点

- ▶ 云计算分布式大数据 Spark 实战高手之路三部曲之第一部
- ▶ 网络发布版为图文并茂方式，边学习，边演练
- ▶ 不需要任何前置知识，从零开始，循序渐进

## 本书作者



王家林，Spark 亚太研究院院长和首席专家，中国目前唯一的移动互联网和云计算大数据集大成者。在 Spark、Hadoop、Android 等方面有丰富的源码、实务和性能优化经验。彻底研究了 Spark 从 0.5.0 到 0.9.1 共 13 个版本的 Spark 源码，并已完成 2014 年 5 月 31 日发布的 Spark1.0 源码研究。

Hadoop 源码级专家，曾负责某知名公司的类 Hadoop 框架开发工作，专注于 Hadoop 一站式解决方案的提供，同时也是云计算分布式大数据处理的最早实践者之一。

Android 架构师、高级工程师、咨询顾问、培训专家。

通晓 Spark、Hadoop、Android、HTML5，迷恋英语播音和健美。

“真相会使你获得自由。”

— 耶稣《圣经》约翰 8:32KJV

“所有人类的幸福都来源于不能直面事实。”

— 释迦摩尼

“道法自然”

— 老子《道德经》第 25 章

## 《云计算分布式大数据 Spark 实战高手之路》

### 系列丛书三部曲

《云计算分布式大数据 Spark 实战高手之路---从零开始》：

不需要任何基础，带领您无痛入门 Spark 并能够轻松处理 Spark 工程师的日常编程工作，内容包括 Spark 集群的构建、Spark 架构设计、RDD、Shark/SparkSQL、机器学习、图计算、实时流处理、Spark on Yarn、JobServer、Spark 测试、Spark 优化等。

《云计算分布式大数据 Spark 实战高手之路---高手崛起》：

大话 Spark 源码，全世界最有情趣的源码解析，过程中伴随诸多实验，解析 Spark 1.0 的任何一句源码！更重要的是，思考源码背后的问题场景和解决问题的设计哲学和实现招式。

《云计算分布式大数据 Spark 实战高手之路---高手之巅》：

通过当今主流的 Spark 商业使用方法和最成功的 Hadoop 大型案例让您直达高手之巅，从此一览众山小。



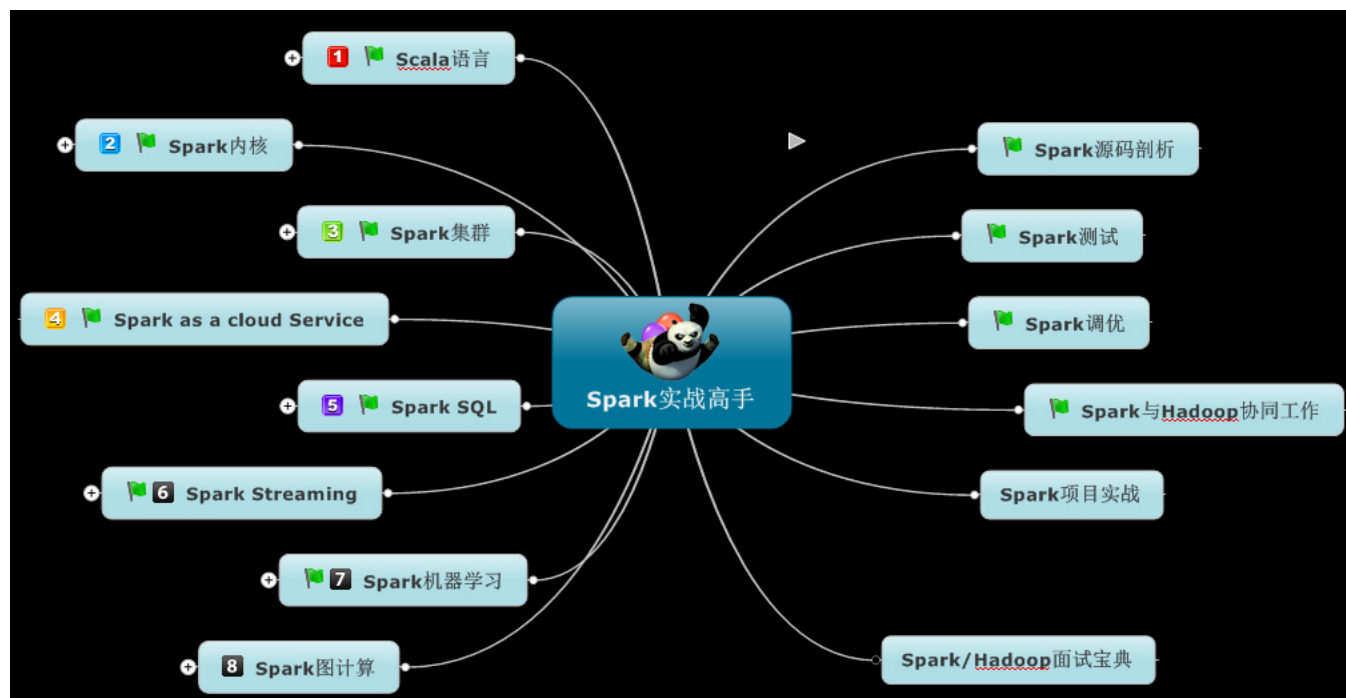
## 《前言》

Spark采用一个统一的技术堆栈解决了云计算大数据的如流处理、图技术、机器学习、NoSQL查询等方面的所有核心问题，具有完善的生态系统，这直接奠定了其统一云计算大数据领域的霸主地位；

要想成为Spark高手，需要经历六大阶段



## Spark 实战高手之核心技能点



### 第一阶段：熟练的掌握Scala语言

1. Spark 框架是采用 Scala 语言编写的，精致而优雅。要想成为 Spark 高手，你就必须阅读 Spark 的源代码，就必须掌握 Scala；
  2. 虽然说现在的 Spark 可以采用多语言 Java、Python 等进行应用程序开发，但是最快速的和支持最好的开发 API 依然并将永远是 Scala 方式的 API，所以你必须掌握 Scala 来编写复杂的和高性能的 Spark 分布式程序；
  3. 尤其要熟练掌握 Scala 的 trait、apply、函数式编程、泛型、逆变与协变等；
- 推荐课程：“精通Spark的开发语言：Scala最佳实践”

### 第二阶段：精通Spark平台本身提供给开发者API

1. 掌握 Spark 中面向 RDD 的开发模式 掌握各种 transformation 和 action 函数的使用；
  2. 掌握 Spark 中的宽依赖和窄依赖以及 lineage 机制；
  3. 掌握 RDD 的计算流程，例如 Stage 的划分、Spark 应用程序提交给集群的基本过程和 Worker 节点基础的工作原理等
- 推荐课程：“18 小时内掌握Spark：把云计算大数据速度提高 100 倍以上!”

### 第三阶段：深入Spark内核

此阶段主要是通过 Spark 框架的源码研读来深入 Spark 内核部分：

1. 通过源码掌握 Spark 的任务提交过程；
2. 通过源码掌握 Spark 集群的任务调度；
3. 尤其要精通 DAGScheduler、TaskScheduler 和 Worker 节点内部的工作的每一步的细节；

推荐课程：[“Spark 1.0.0 企业级开发动手：实战世界上第一个Spark 1.0.0 课程，涵盖Spark 1.0.0 所有的企业级开发技术”](#)

### 第四阶段:掌握基于Spark上的核心框架的使用

Spark 作为云计算大数据时代的集大成者，在实时流处理、图技术、机器学习、NoSQL 查询等方面具有显著的优势，我们使用 Spark 的时候大部分时间都是在使用其上的框架例如 Shark、Spark Streaming 等：

1. Spark Streaming 是非常出色的实时流处理框架，要掌握其 DStream、transformation 和 checkpoint 等；
2. Spark 的离线统计分析功能，Spark 1.0.0 版本在 Shark 的基础上推出了 Spark SQL，离线统计分析的功能的效率有显著的提升，需要重点掌握；
3. 对于 Spark 的机器学习和 GraphX 等要掌握其原理和用法；

推荐课程：[“Spark企业级开发最佳实践”](#)

### 第五阶段:做商业级别的Spark项目

通过一个完整的具有代表性的 Spark 项目来贯穿 Spark 的方方面面，包括项目的架构设计、用到的技术的剖析、开发实现、运维等，完整掌握其中的每一个阶段和细节，这样就可以让您以后可以从容面对绝大多数 Spark 项目。

推荐课程：[“Spark架构案例鉴赏：Conviva、Yahoo！、优酷土豆、网易、腾讯、淘宝等公司的实际Spark案例”](#)

### 第六阶段：提供Spark解决方案

1. 彻底掌握 Spark 框架源码的每一个细节；
2. 根据不同的业务场景的需要提供 Spark 在不同场景的下的解决方案；
3. 根据实际需要，在 Spark 框架基础上进行二次开发，打造自己的 Spark 框架；

推荐课程：[“精通Spark：Spark内核剖析、源码解读、性能优化和商业案例实战”](#)

## 《第二章：动手实战 Scala》

Scala 一门面向对象和函数编程完美结合的语言，特别适合于构建大型项目和密集而复杂的数据处理。

Spark 是基于 Scala 语言开发的大数据通用处理平台，于此同时在 Spark 上虽然可以使用 Scala、Java、Python 三种语言进行分布式应用程序开发，但 Scala 确实支持最好的首选开发语言，所以无论是从阅读源码的角度来讲还是开发程序的角度来讲，对 Scala 的掌握都是必要且至关重要的。

本章的内容主要是从 Spark 的角度来讲解 Scala，以动手实战为核心，从零开始，循序渐进的掌握 Scala 函数式编程和面向对象编程。

### 从零起步，动手实战 Scala 三部曲：

- 第一步：动手体验 Scala
- 第二步：动手实战 Scala 面向对象编程
- 第三步：动手实战 Scala 函数式编程

本讲是动手实战 Scala 三部曲的第三步：动手实战 Scala 函数式编程，具体内容如下所示：

- 1，动手实战 Scala 高阶函数
- 2，动手实战 Scala 中的集合
- 3，动手实战 Scala 中的泛型
- 4，动手实战 Scala 中的隐式转换、隐式参数、隐式类
- 5，动手实战 Scala 中的 apply 方法和单例对象

**不需任何前置知识，从零开始，循序渐进，成为 Spark 高手！**





函数式编程的核心特色之一是把函数作为参数传递给函数、在函数内部可以定义函数等。

## 1，动手实战 Scala 高阶函数

声明一个 List 集合实例：

```
scala> val l = List(1,2,3,4,5,6,7,8,9)
l: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)
scala>
```

List 集合所在的包已经被预定义自动导入，所以此处不需要导入包，同时，这里直接使用 List 实例化对象，其实是用来 List 的 object 对象的 apply 方法；

我们使用 map 函数把 List 中的每个值都乘以 2：

```
scala> val newList = l.map(<x : Int> => 2 * x)
newList: List[Int] = List(2, 4, 6, 8, 10, 12, 14, 16, 18)
scala>
```

在上面的代码中，x 表示 l 中每个一个元素，map 对 l 中的每一个元素进行遍历操作，由于 List 中只有一种类型的元素，所以我们在执行 map 操作的时候可以省略掉其类型，如下所示：

```
scala> l.map(<x> => 2 * x)
res2: List[Int] = List(2, 4, 6, 8, 10, 12, 14, 16, 18)
```

List 集合中只有一个参数的时候，我们可以去掉参数中的括号：

```
scala> l.map(x => 2 * x)
res3: List[Int] = List(2, 4, 6, 8, 10, 12, 14, 16, 18)
```

在只有一个参数的情况下，更简洁和正常的写法如下所示：

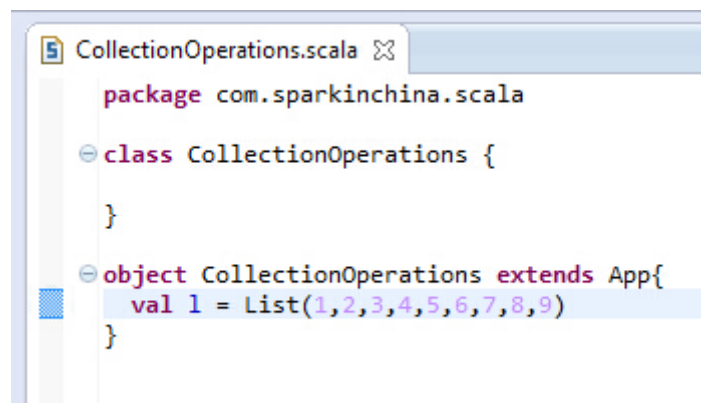
```
scala> l.map(_ * 2)
res4: List[Int] = List(2, 4, 6, 8, 10, 12, 14, 16, 18)
```

常用的高阶函数有 map、filter、reduce 等，我们在稍后会做介绍。

## 2, 动手实战 Scala 中的集合

集合主要有 List、Set、Tuple、Map 等，我们下面以动手实战的方式来学习。

我们在 Eclipse 这个 IDE 中创建一个 List 实例：



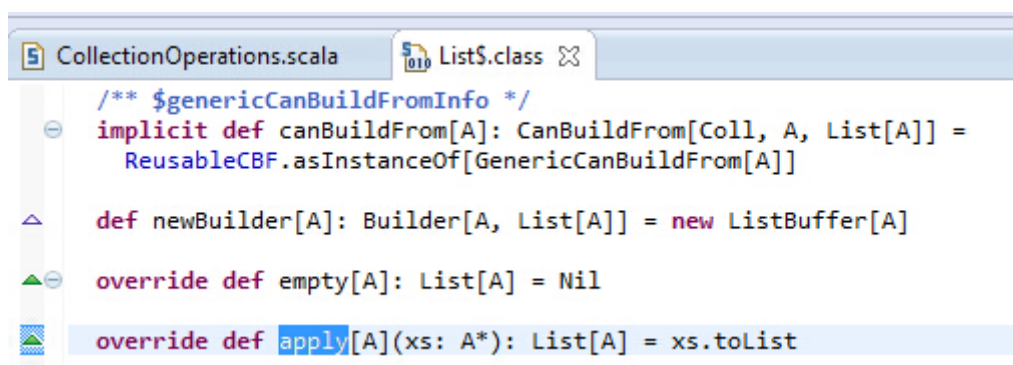
```
CollectionOperations.scala
package com.sparkinchina.scala

class CollectionOperations {

}

object CollectionOperations extends App{
  val l = List(1,2,3,4,5,6,7,8,9)
}
```

此时我们看一下其代码实现：



```
CollectionOperations.scala  List$.class
/** $genericCanBuildFromInfo */
implicit def canBuildFrom[A]: CanBuildFrom[Coll, A, List[A]] =
  ReusableCBF.asInstanceOf[GenericCanBuildFrom[A]]

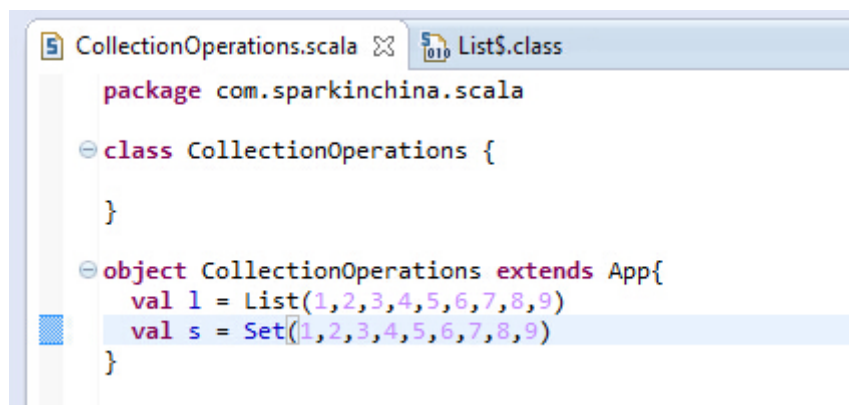
def newBuilder[A]: Builder[A, List[A]] = new ListBuffer[A]

override def empty[A]: List[A] = Nil

override def apply[A](xs: A*): List[A] = xs.toList
```

源代码中说明了其内部是 apply 的方式来完成实例化的；

同样的方式我们可以实例化 Set：



```
CollectionOperations.scala  List$.class
package com.sparkinchina.scala

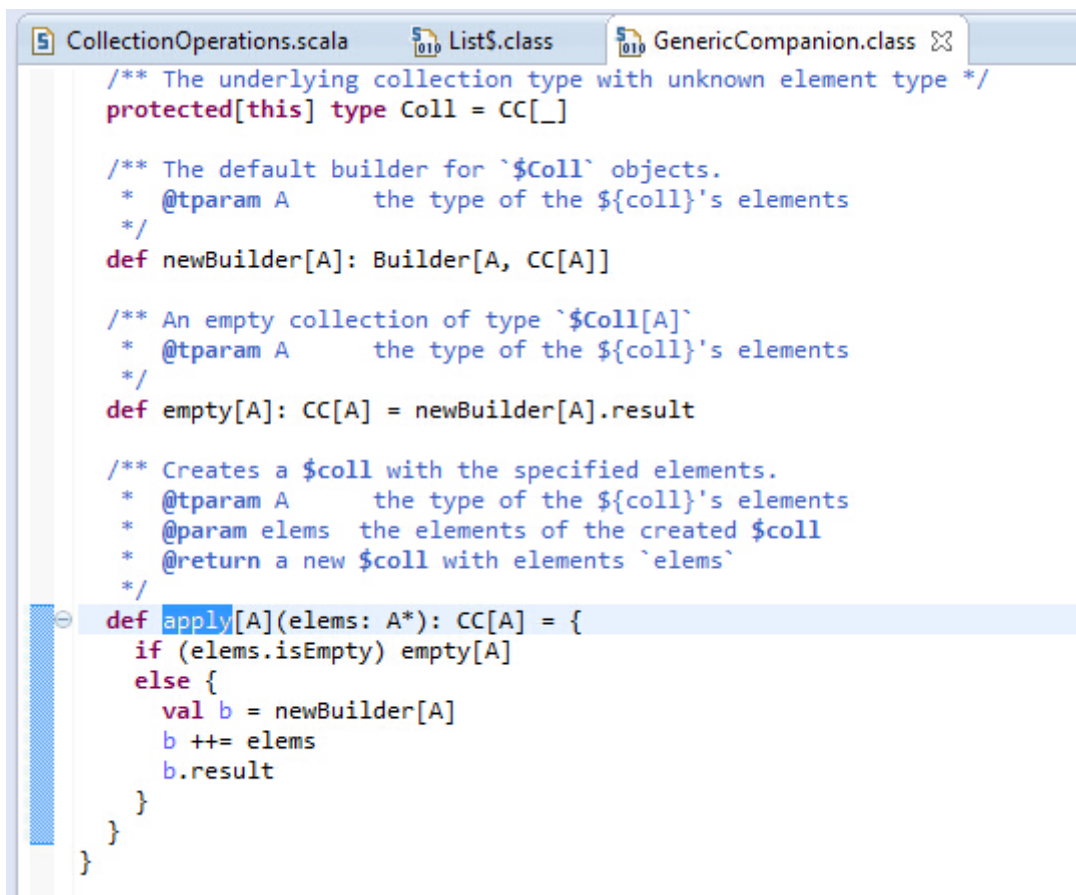
class CollectionOperations {

}

object CollectionOperations extends App{
  val l = List(1,2,3,4,5,6,7,8,9)
  val s = Set(1,2,3,4,5,6,7,8,9)
}
```



此时也可以看一下 Set 实例化对象的实现：



```
CollectionOperations.scala List$.class GenericCompanion.class
/** The underlying collection type with unknown element type */
protected[this] type Coll = CC[_]

/** The default builder for `Coll` objects.
 * @tparam A the type of the coll's elements
 */
def newBuilder[A]: Builder[A, CC[A]]

/** An empty collection of type `Coll[A]`
 * @tparam A the type of the coll's elements
 */
def empty[A]: CC[A] = newBuilder[A].result

/** Creates a coll with the specified elements.
 * @tparam A the type of the coll's elements
 * @param elems the elements of the created coll
 * @return a new coll with elements `elems`
 */
def apply[A](elems: A*): CC[A] = {
  if (elems.isEmpty) empty[A]
  else {
    val b = newBuilder[A]
    b ++= elems
    b.result
  }
}
```

接下来我们在命令行终端中看一下集合的操作，首先看一下 Set：

```
scala> val s = Set(1,2,1)
s: scala.collection.immutable.Set[Int] = Set(1, 2)

scala>
```

可以发现 Set 中不会存在重复的元素。

下面看一下 Tuple 的声明和使用：

```
scala> val hostPort = ("localhost", "8080")
hostPort: (String, String) = (localhost,8080)

scala> hostPort._1
res5: String = localhost

scala> hostPort._2
res6: String = 8080

scala>
```

从上述代码中可以看出源码访问的时候下标是从 1 开始的；

对 Tuple 而言，如果只有两个元素的时候还可以使用下述方式创建：

```
scala> "a" -> "b"
res7: <String, String> = <a,b>
scala>
```

接下来看一下 Map 的定义：

```
scala> Map("a" -> "b")
res8: scala.collection.immutable.Map[String,String] = Map(a -> b)
```

Map 本身使用的是可变参数的方式，所以可以给 Map 赋多个值：

```
scala> Map("a" -> "b", "c" -> "d")
res9: scala.collection.immutable.Map[String,String] = Map(a -> b, c -> d)

scala> Map("a" -> "b", "c" -> "d", "e" -> "f")
res10: scala.collection.immutable.Map[String,String] = Map(a -> b, c -> d, e -> f)

scala>
```

下面看一下 Option 类型，Option 代表了一个可有可无的值：

```
scala> Option
res11: Option.type = scala.Option$@54dccb59
```

Option 有两个子类：Some 和 None，下面我们看 Option 的使用：

```
scala> val m = Map(1 -> 2)
m: scala.collection.immutable.Map[Int,Int] = Map(1 -> 2)

scala> m.get(1)
res14: Option[Int] = Some(2)

scala> m.get(1).get
res15: Int = 2

scala> m.get(2).getOrElse("None")
res16: Any = None

scala> m.get(2).getOrElse(0)
res17: Int = 0

scala>
```

接下来看一下 filter 的处理：

```
scala> val l = List(1,2,3,4,5,6,7,8,9)
l: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> l.filter(x => x % 2 == 0)
res19: List[Int] = List(2, 4, 6, 8)

scala> l.filter(_ % 2 == 0)
res20: List[Int] = List(2, 4, 6, 8)

scala>
```

下面看一下对集合的 zip 操作：

```
scala> val a = List(1,2,3)
a: List[Int] = List(1, 2, 3)

scala> val b = List(4,5,6)
b: List[Int] = List(4, 5, 6)

scala> a zip b
res21: List[(Int, Int)] = List((1,4), (2,5), (3,6))

scala> a.zip(b)
res22: List[(Int, Int)] = List((1,4), (2,5), (3,6))

scala> val b = List(1,2,3,4)
b: List[Int] = List(1, 2, 3, 4)

scala> a zip b
res23: List[(Int, Int)] = List((1,1), (2,2), (3,3))

scala>
```

下面看一下 partition 对集合的切割操作：

```
scala> l.partition(_%2 ==0)
res25: (List[Int], List[Int]) = (List(2, 4, 6, 8), List(1, 3, 5, 7, 9))
```

我们可以使用 flatten 的多集合进行扁平化操作：

```
scala> val l = List(List("a","b"),List("c","d"))
l: List[List[String]] = List(List(a, b), List(c, d))

scala> l.flatten
res26: List[String] = List(a, b, c, d)

scala> val l = List(List("a","b"),List("c","d")).flatten
l: List[String] = List(a, b, c, d)
```

flatMap 是 map 和 flatten 操作的结合，先进行 map 操作然后进行 flatten 操作：

```
scala> val l = List(List(1,2),List(3,4))
l: List[List[Int]] = List(List(1, 2), List(3, 4))

scala> l.flatMap(x => x.map(_ * 2))
res27: List[Int] = List(2, 4, 6, 8)
```

### 3，动手实战 Scala 中的泛型

泛型泛型类和泛型方法，也就是我们实例化类或者调用方法的时候可以指定其类型，由于 Scala 的泛型和 Java 的泛型是一致的，这里不再赘述。

### 4，动手实战 Scala 中的隐式转换、隐式参数、隐式类

隐式转换是很多人学习 Scala 的难点，这是 Scala 的精髓之一：

```
package com.sparkinchina.scala

class Implicit {
}

class A{
}

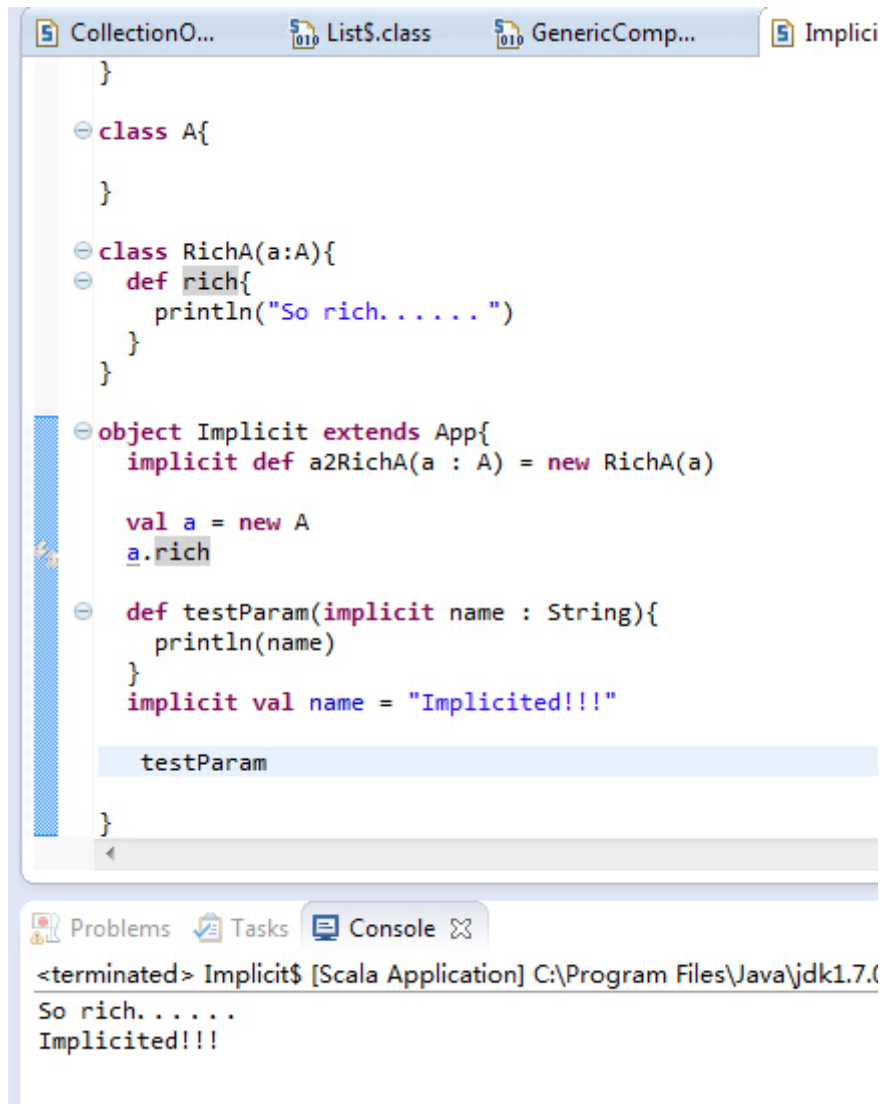
class RichA(a:A){
  def rich{
    println("So rich.....")
  }
}

object Implicit extends App{
  implicit def a2RichA(a : A) = new RichA(a)

  val a = new A
  a.rich
}

<terminated> Implicit$ [Scala Application] C:\Program Files\Java\jdk1.7.0_1
So rich.....
```

下面看一下隐藏参数的例子：



```
CollectionO... List$.class GenericComp... Implici
}
class A{
}
class RichA(a:A){
  def rich{
    println("So rich.....")
  }
}
object Implicit extends App{
  implicit def a2RichA(a : A) = new RichA(a)

  val a = new A
  a.rich

  def testParam(implicit name : String){
    println(name)
  }
  implicit val name = "Implicated!!!"

  testParam
}

<terminated> Implicit$ [Scala Application] C:\Program Files\Java\jdk1.7.0
So rich.....
Implicated!!!
```

上面的例子中使用了隐式参数，当然，你可以显示的指明参数：



The screenshot shows an IDE window with the following Scala code:

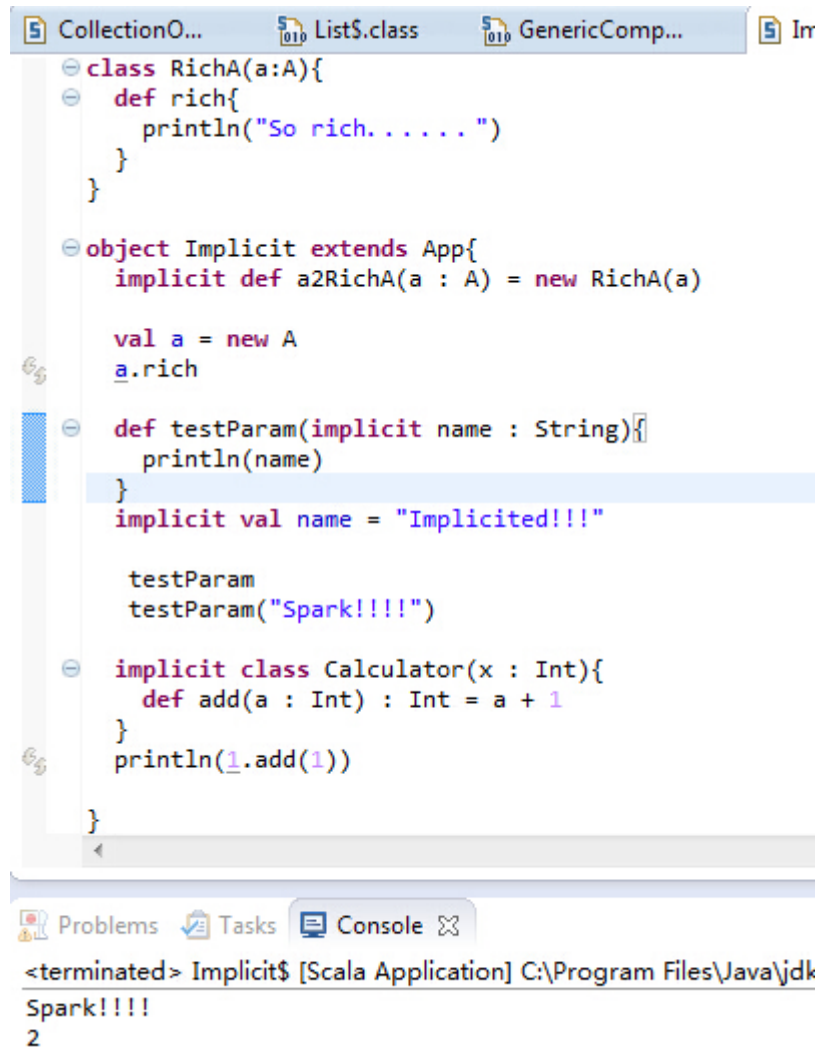
```
class Implicit {  
  }  
  
class A {  
  }  
  
class RichA(a:A){  
  def rich{  
    println("So rich.....")  
  }  
}  
  
object Implicit extends App{  
  implicit def a2Rich(a : A) = new RichA(a)  
  
  val a = new A  
  a.rich  
  
  def testParam(implicit name : String){  
    println(name)  
    implicit val name = "Implicated!!!"  
  
    testParam  
    testParam("Spark!!!!")  
  }  
}
```

The console output at the bottom shows the execution results:

```
<terminated> Implicit$ [Scala Application] C:\Program Files\Java\jdk1.7.0_67\bin\jz  
So rich.....  
Implicated!!!  
Spark!!!!
```



下面看一下隐式类：



```
CollectionO... List$.class GenericComp... In
class RichA(a:A){
  def rich{
    println("So rich.....")
  }
}

object Implicit extends App{
  implicit def a2RichA(a : A) = new RichA(a)

  val a = new A
  a.rich

  def testParam(implicit name : String){
    println(name)
  }
  implicit val name = "Implicated!!!"

  testParam
  testParam("Spark!!!!")

  implicit class Calculator(x : Int){
    def add(a : Int) : Int = a + 1
  }
  println(1.add(1))
}

Problems Tasks Console
<terminated> Implicit$ [Scala Application] C:\Program Files\Java\jdk
Spark!!!!
2
```