# 类、对象、继承、特质

Scala的类与Java、C++的类比起来更简洁，学完之后你会更爱Scala！！！

## 类

### 类的定义

|  |
| --- |
| *//在Scala中，类并不用声明为public。*  *//Scala源文件中可以包含多个类，所有这些类都具有公有可见性。* **class** Student {  *//用val修饰的变量是只读属性，有getter但没有setter*  *//（相当与Java中用final修饰的变量）* **val** *id* = 666  *//用var修饰的变量既有getter又有setter* **var** *age*: Int = 20  *//类私有字段,只能在类的内部使用* **private var** *name*: String = **"tom"**   *//对象私有字段,访问权限更加严格的，Person类的方法只能访问到当前对象的字段* **private**[**this**] **val** *pet* = **"小强"**  } |

### 构造器

注意：主构造器会执行类定义中的所有语句

|  |
| --- |
| */\*\*  \*每个类都有主构造器，主构造器的参数直接放置类名后面，与类交织在一起  \*/* **class** Person(**val** name: String, **val** age: Int){  *//主构造器会执行类定义中的所有语句  println*(**"执行主构造器"**)   **try** {  *println*(**"读取文件"**)  **throw new** IOException(**"io exception"**)  } **catch** {  **case** e: NullPointerException => *println*(**"打印异常Exception : "** + e)  **case** e: IOException => *println*(**"打印异常Exception : "** + e)  } **finally** {  *println*(**"执行finally部分"**)  }   **private var** *gender* = **"male"** *//用this关键字定义辅助构造器* **def this**(name: String, age: Int, gender: String){  *//每个辅助构造器必须以主构造器或其他的辅助构造器的调用开始*  **this**(name, age)  *println*(**"执行辅助构造器"**)  **this**.*gender* = gender  } } |

|  |
| --- |
| *//在类名后面加private就变成了私有的* **class** People **private**(**val** name: String, **private var** age: Int = 18){   **object** People{  **def** main(args: Array[String]) {  *//私有的构造器，只有在其伴生对象中使用* **val** q = **new** People(**"hatano"**, 20)  } } |

## 对象

### 单例对象

在Scala中没有静态方法和静态字段，但是可以使用object这个语法结构来达到同样的目的

1. 存放工具方法和常量
2. 高效共享单个不可变的实例
3. 单例模式

|  |
| --- |
| **import** scala.collection.mutable.ArrayBuffer **object** SingletonDemo {  **def** main(args: Array[String]) {  *//单例对象，不需要new，用【类名.方法】调用对象中的方法* **val** session = SessionFactory.*getSession*()  println(session)  } } **object** SessionFactory{  *//该部分相当于java中的静态块* **var** *counts* = 5  **val** *sessions* = **new** ArrayBuffer[Session]()  **while**(*counts* > 0){  *sessions* += **new** Session  *counts* -= 1  }   *//在object中的方法相当于java中的静态方法* **def** getSession(): Session ={  *sessions*.remove(0)  } }  **class** Session{  } |

### 伴生对象

在Scala的类中，与类名相同的对象叫做伴生对象，类和伴生对象之间可以相互访问私有的方法和属性

|  |
| --- |
| **class** Dog {  **val** *id* = 1  **private var** *name* = **"xiaoqing"   def** printName(): Unit ={  *//在Dog类中可以访问伴生对象Dog的私有属性  println*(Dog.*CONSTANT* + *name* )  } }  */\*\*  \* 伴生对象  \*/* **object** Dog {   *//伴生对象中的私有属性* **private val** *CONSTANT* = **"汪汪汪 : "   def** main(args: Array[String]) {  **val** p = **new** Dog  *//访问私有的字段name* p.*name* = **"123"** p.printName()  } } |

### apply方法

通常我们会在类的伴生对象中定义apply方法，当遇到类名(参数1,...参数n)时apply方法会被调用

|  |
| --- |
| **object** ApplyDemo {  **def** main(args: Array[String]) {  *//调用了Array伴生对象的apply方法  //def apply(x: Int, xs: Int\*): Array[Int]  //arr1中只有一个元素5* **val** arr1 = *Array*(5)  *println*(arr1.toBuffer)   *//new了一个长度为5的array，数组里面包含5个null* **var** arr2 = **new** Array(5)  } } |

### 应用程序对象

Scala程序都必须从一个对象的main方法开始，可以通过扩展App特质，不写main方法。

|  |
| --- |
| **object** AppObjectDemo **extends** App{  *//不用写main方法  println*(**"I love you Scala"**) } |

## 继承

### 扩展类

在Scala中扩展类的方式和Java一样都是使用extends关键字

### 重写方法

在Scala中重写一个非抽象的方法必须使用override修饰符

### 类型检查和转换

|  |  |
| --- | --- |
| **Scala** | **Java** |
| obj.isInstanceOf[C] | obj instanceof C |
| obj.asInstanceOf[C] | (C)obj |
| classOf[C] | C.class |

### 超类的构造

|  |
| --- |
| **object** ClazzDemo {  **def** main(args: Array[String]) { } }  **trait** Flyable{  **def** fly(): Unit ={  *println*(**"I can fly"**)  }   **def** fight(): String }  **abstract class** Animal {  **def** run(): Int  **val** name: String }  **class** Human **extends** Animal **with** Flyable{   **val** *name* = **"abc"** *//打印几次"ABC"?* **val** *t1*,*t2*,(*a*, *b*, *c*) = {  *println*(**"ABC"**)  (1,2,3)  }   *println*(*a*)  *println*(*t1*.\_1)   *//在Scala中重写一个非抽象方法必须用override修饰* **override def** fight(): String = {  **"fight"** }  *//在子类中重写超类的抽象方法时，不需要使用override关键字，写了也可以* **def** run(): Int = {  1  } } |

# 模式匹配和样例类

Scala有一个十分强大的模式匹配机制，可以应用到很多场合：如switch语句、类型检查等。

并且Scala还提供了样例类，对模式匹配进行了优化，可以快速进行匹配

## 匹配字符串

|  |
| --- |
| **import** scala.util.Random **object** CaseDemo01 **extends** App{  **val** *arr* = *Array*(**"YoshizawaAkiho"**, **"YuiHatano"**, **"AoiSola"**)  **val** *name* = *arr*(Random.nextInt(*arr*.length))  *name* **match** {  **case "YoshizawaAkiho"** => *println*(**"吉泽老师..."**)  **case "YuiHatano"** => *println*(**"波多老师..."**)  **case** \_ => *println*(**"真不知道你们在说什么..."**)  } } |

## **匹配类型**

|  |
| --- |
| **import** scala.util.Random **object** CaseDemo01 **extends** App{ *//val v = if(x >= 5) 1 else if(x < 2) 2.0 else "hello"* **val** *arr* = *Array*(**"hello"**, 1, 2.0, CaseDemo)  **val** *v* = *arr*(Random.nextInt(4))  *println*(*v*)  *v* **match** {  **case** x: Int => *println*(**"Int "** + x)  **case** y: Double **if**(y >= 0) => *println*(**"Double "**+ y)  **case** z: String => *println*(**"String "** + z)  **case** \_ => **throw new** Exception(**"not match exception"**)  } } |

**注意：**case y: Double if(y >= 0) => ...

模式匹配的时候还可以添加守卫条件。如不符合守卫条件，将掉入case \_中

## 匹配数组、元组

|  |
| --- |
| **object** CaseDemo03 **extends** App{   **val** *arr* = *Array*(1, 3, 5)  *arr* **match** {  **case** *Array*(1, x, y) => *println*(x + **" "** + y)  **case** *Array*(0) => *println*(**"only 0"**)  **case** *Array*(0, \_\*) => *println*(**"0 ..."**)  **case** \_ => *println*(**"something else"**)  }   **val** *lst* = *List*(3, -1)  *lst* **match** {  **case** 0 *:: Nil* => *println*(**"only 0"**)  **case** x *::* y *:: Nil* => *println*(**s"x: $**x **y: $**y**"**)  **case** 0 *::* tail => *println*(**"0 ..."**)  **case** \_ => *println*(**"something else"**)  }   **val** *tup* = (2, 3, 5)  *tup* **match** {  **case** (2, x, y) => *println*(**s"1, $**x **, $**y**"**)  **case** (\_, z, 5) => *println*(z)  **case** \_ => *println*(**"else"**)  } } |

注意：在Scala中列表要么为空（Nil表示空列表）要么是一个head元素加上一个tail列表。

9 :: List(5, 2) :: 操作符是将给定的头和尾创建一个新的列表

注意：:: 操作符是右结合的，如9 :: 5 :: 2 :: Nil相当于 9 :: (5 :: (2 :: Nil))

## 样例类

在Scala中样例类是一中特殊的类，可用于模式匹配。case class是多例的，后面要跟构造参数，case object是单例的

|  |
| --- |
| **import** scala.util.Random  **case class** SubmitTask(id: String, name: String) **case class** HeartBeat(time: Long) **case object** CheckTimeOutTask  **object** CaseDemo04 **extends** App{  **val** *arr* = *Array*(CheckTimeOutTask, *HeartBeat*(12333), *SubmitTask*(**"0001"**, **"task-0001"**))   *arr*(Random.nextInt(*arr*.length)) **match** {  **case** *SubmitTask*(id, name) => {  *println*(**s"$**id**, $**name**"**)  }  **case** *HeartBeat*(time) => {  *println*(time)  }  **case** CheckTimeOutTask => {  *println*(**"check"**)  }  } } |

## Option类型

在Scala中Option类型样例类用来表示可能存在或也可能不存在的值(Option的子类有Some和None)。Some包装了某个值，None表示没有值

|  |
| --- |
| **object** OptionDemo {  **def** main(args: Array[String]) {  **val** map = *Map*(**"a"** -> 1, **"b"** -> 2)  **val** v = map.get(**"b"**) **match** {  **case** *Some*(i) => i  **case** None => 0  }  *println*(v)  *//更好的方式* **val** v1 = map.getOrElse(**"c"**, 0)  *println*(v1)  } } |

## 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

|  |
| --- |
| **object** PartialFuncDemo {   **def** func1: PartialFunction[String, Int] = {  **case "one"** => 1  **case "two"** => 2  **case** \_ => -1  }  **def** func2(num: String) : Int = num **match** {  **case "one"** => 1  **case "two"** => 2  **case** \_ => -1  }  **def** main(args: Array[String]) {  *println*(*func1*(**"one"**))  *println*(*func2*(**"one"**))  } } |