Day30_Scala中的模式匹配

大数据-张军锋 Day30 Scala 模式匹配

scala中的模式匹配match case

Java中的switch case

- 只能够对基础数据类型进行匹配
- 每一个case都需要加break跳出匹配
- 只能够对值进行匹配
- 只是一种流程控制语法，无返回值

Scala中的match case

- 可以对任意类型进行匹配
- 不需要加break跳出，所有case表达式中，只要有一个被匹配就马上跳出
- 除了能对值进行匹配外，还有多种匹配方式（如：String、Int、Douuble……）
- 有返回值

常量模式匹配

```
def constantMatchconstantMatch(x: Any) = {
  val specialValue = "123abc"
  x match {
    //case s if s == specialValue => "幸运值"
    case `specialValue` => "幸运值"
    case "abc" => "常量值abc"
    case true => "常量值true"
    case 123 => "常量值123"
    case 22.30 => "常量值22.30"
    case _ => "其他值"
  }
}
```

变量模式匹配

```
def variableMatch(x: Any) = {
  x match {
    case z => println(s"$z")
    case _ => println("其他")
  }
}

//在变量后面加过滤条件进行匹配
def scoreAnaylize(score: Int) = {
  score match {
    case x if x > 90 => s"优，分数：$x"
    case x if x > 80 => s"良，分数：$x"
    case x if x > 70 => s"中，分数：$x"
    case x if x > 60 => s"及格，分数：$x"
    case _ => s"差，分数：$score"
  }
}
```

类型匹配

```
def typeMatchCase(x: Any) = {
  x match {
    case z: Boolean => s"${z}是一个Boolean类型的变量"
    case z: Int => s"${z}是一个Int类型的变量"
    case z: String => s"${z}是一个String类型的变量"
    case _ => s"${x}是一个其他类型的变量"
  }
}
```

Option Some None

```
def osnMatchCase(x: Option[Int]) = {
  x match {
    case Some(1) => "结果是1"
    case Some(2) => "结果是2"
    case Some(y) => s"结果是$y"
    case None => "没有结果值"
  }
}
```

序列匹配，集合匹配

```
def listMatchCase(x: List[Int]) = {
  x match {
    case List(9, _*) => "9开头的List"
    case List(a, b) => s"list有两个元素，第一个值是$a，第二个值是$b"
    case List(a, b, _) => s"list有三个元素，第一个值是$a，第二个值是$b"
    case List(_, _, _, _) => "list有四个元素"
    case List(a, b, _*) => s"list有至少三个元素，第一个值是$a，第二个值
是$b"
    case Nil => "这是一个空的List"
    case _ => "其他List"
  }
}
```

集合类型匹配

array可以匹配集合类型，List在类型匹配上不能匹配到泛型

```
class Parent {}
class Child extends Parent {}

def arrayTypeMatchCase(x: Any) = {
  x match {
    case v: Array[Child] => "x是一个Child的集合类型"
    case v: Array[Parent] => "x是一个Parent的集合类型"
    case v: Array[Int] => "x是一个Int的集合类型"
    case _ => "x是其他类型的集合类"
  }
}

//不可以
def listTypeMatchCase(x: List[Any]) = {
  x match {
    case v: List[Child] => "x是一个Child的集合类型"
    case v: List[Parent] => "x是一个Parent的集合类型"
    case v: List[Int] => "x是一个Int的集合类型"
    case _ => "x是其他类型的集合类"
  }
}
```

元组匹配

match case一次只能匹配一个值,如果想一次匹配多个值,那么需要用元组把多个值封装起来
_*不适合用于元组，只适用于序列

```
def tupleMatchCase(x: Any, y: Any) = {
  (x, y) match {
    case (1, 2) => "x是1，y是2"
    case (3, 4) => "x是3，y是4"
    case (5, _) => "x是5，y随意"
    //case (one,_*) => one
    case _ => "其他"
  }
}
```

正则表达式匹配

```
def regularRegexMatch(x: String) = {
  val regExp = "(\\d+)\\.(\\d+)\\.(\\d+)\\.(\\d+)".r
  x match {
    case regExp(one, two, three, four) => s"第一段：$one,第二段：$tw
o，第三段：$three，第四段：$four"
    case _ => "不是ip"
  }
}
```

case class

case class 是scala的特殊类型,它是专门用于承载数据的类型,类似java中的JavaBean类,在声明case class 的时候主构造函数必须要显示声明,属性都写在主构造方法中

case class由编译器自动给我们生成其伴生对象里面，并且在伴生对象里自动帮我们实现apply方法

case class主构造方法中的参数可以用var或val声明，自动会被当做类的属性，相当于有一个默认的val，如果想把属性声明

case class 自动帮我们重写美化的toString方法,还有hashCode方法和equal方法

```
case class Student(id: String, name: String, var age: Int)
case class SchoolClass(id: String, student: Student)

//构造方法匹配，case class来使用
def caseClassMatch(x: Any) = {
  x match {
    case Student(id,name,_) => s"x是是student类型，id是$id，姓名是：$na
me"
    //输出18岁的
    //case Student(id, name, 18) => s"x是是student类型，id是$id，姓名
是：$name，年龄是18岁"
    case _ => "其他类型"
  }
}

//嵌套匹配
def caseClassMultiplyMatch(x: Any) = {
  x match {
    case SchoolClass(id, Student(sid, sname, 18)) => s"x是是student
类型，id是$sid，姓名是：$sname，年龄是18岁"
    case _ => "其他类型"
  }
}
```

如果在match case中没有把sealed类的所有子类都case判断一下的话,编译器会报一个提示,提示我们把所有情况考虑在内

```
//case class类型匹配
class MStudent

case class LittleStudent(id: String, name: String) extends MStudent

case class HighStudent(id: String, name: String) extends MStudent

case class CollageStudent(id: String, name: String) extends MStuden
t

case class OtherStudent(name: String) extends MStudent

def caseClassMatchMStudent(x: Any) = {
  x match {
    case LittleStudent(id, name) => "小学生"
    case HighStudent(id, name) => "中学生"
    case CollageStudent(id, name) => "大学生"
    case OtherStudent(name) => "其他学生类型"
  }
}
```

异常

```
object ExceptionTest {
  def main(args: Array[String]): Unit = {
    try{
      //throw new NullPointerException("这里抛出异常")
      throw new ClassNotFoundException("有个:xxxClass类找不到")
    }catch {
      case e:NullPointerException =>println("抛出了空指针异常")
      case e:ClassNotFoundException =>println("抛出了未找到类定义异常")
      case e:Exception =>println("其他类型异常");e.printStackTrace()
    }
  }
}
```