Scala 面向对象

目录

1、Scala 类

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

1.1、定义类

定义一个 Student 类：

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */

//在 Scala 中，类并不用声明为 public。
//在 Scala 文件中，文件名可以不用和类名一致
//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 Person {

  //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 = "100"

  //用 var 修饰的变量既有 getter 又有 setter
  var age: Int = 33

  //类私有字段,只能在当前类的内部或伴生对象中使用
  private var name: String = "黄渤"

  //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
  //伴生对象都不能访问
  private[this] val nickname = "影帝"
}
```

通过反编译工具反编译之后看到：

```
package com.mazh.scala.util;

public class Person {
   private final String id = "100";
   private int age = 33;
   private String name = "黄渤";
   private final String nickname = "影帝";

   // 相当于 id 属性的 getId() 方法
   public String id() {
      return this.id;
   }

   // 相当与 age 属性的 getAge() 方法
```

```
    public int age() {
        return this.age;
    }

    // 相当与 age 属性的 setAge 方法
    public void age_$eq(int x$1) {
        this.age = x$1;
    }

    // name 属性的私有 get 方法
    private String name() {
        return this.name;
    }

    // name 属性的私有 set 方法
    private void name_$eq(String x$1) {
        this.name = x$1;
    }
}
```

通过 JDK 自带的反编译工具反编译：

```
C:\IdeaProjects-2017-02-07\ScalaTest\out\production\ScalaTest\com\mazh\scala\day2\oop>javap -private Person
警告: 二进制文件 Person 包含 com.mazh.scala.day2.oop.Person
Compiled from "Person.scala"
public class com.mazh.scala.day2.oop.Person {
  private final java.lang.String id;
  private int age;
  private java.lang.String name;
  private final java.lang.String nickname;
  public java.lang.String id();
  public int age();
  public void age_$eq(int);
  private java.lang.String name();
  private void name_$eq(java.lang.String);
  public com.mazh.scala.day2.oop.Person();
}
```

下面这张图给出了 setter 和 getter 方法的生成关系：

Scala Field	Generated Methods	When to Use
val/var name	public name name_= (var only)	To implement a property that is publicly accessible and backed by a field.
@BeanProperty val/var name	public name getName() name_= (var only) setName(...) (var only)	To interoperate with JavaBeans.
private val/var name	private name name_= (var only)	To confine the field to the methods of this class, just like in Java. Use private unless you really want a public property.
private[this] val/var name	none	To confine the field to methods invoked on the same object. Not commonly used.

1.2、定义构造器

注意：主构造器会执行类定义中的所有语句

定义一个 Student 类：

```
package com.mazh.scala.day2.oop

import java.io.IOException

/**
  * 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
  * 主构造器会执行类定义中的所有语句
  * 当在创建对象时，需要进行相关初始化操作时，可以将初始化语句放在类体中
  * 同样也可以在类中添加或重写相关方法
  */
class Student(val name: String, val age: Int){
  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
  }
}
```

定义辅助构造器：

如果禁用掉了主构建器（使用 private 声明），则必须使用辅助构造函数来创建对象。

辅助构造函数具有两个特点：

1、辅助构建器的名称为 this，Java 中的辅助构造函数与类名相同，这常常会导致修改类名时出现不少问题，scala 语言避免了这样的问题；

2、调用辅助构造函数时，必须先调用主构造函数或其它已经定义好的构造函数。

定义一个 Queen 类：

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */

//构造器参数可以不带 val 或 var，
//如果不带 val 或 var 的参数至少被一个方法所使用，那么它将会被提升为字段
//在类名后面加 private 就变成了私有的，相当于禁用了主构造器
class Queen private(val name: String, prop: Array[String], private var age: Int = 18){

  println(prop.size)

  //prop 被下面的方法使用后，prop 就变成了不可变的对象私有字段，
  //等同于 private[this] val prop
  //如果没有被方法使用该参数将不被保存为字段
  //仅仅是一个可以被主构造器中的代码访问的普通参数
  def description = name + " is " + age + " years old with " + prop.toBuffer
}

object Queen{
```

```
  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 Queen("刘亦菲", Array("女神", "女神经"), 20)
    println(q.description())
  }
}
```

总结：

主构造方法：

1、与类名交织在一起

2、主构造方法运行，导致类名后面的大括号里面的代码都会运行

辅助构造方法：

1、名字必须叫 this

2、必须以调用主构造方法或者是其他辅助构造方法开始

3、里面的属性不能写修饰符

2、Scala 对象

2.1、单例对象

在某些应用场景下，我们可能不需要创建对象，而是想直接调用方法，但是 Scala 语言并不支持静态成员，没有静态方法和静态字段，Scala 通过单例对象 object 来解决该问题

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

```
package com.mazh.scala.day2.oop

import scala.collection.mutable.ArrayBuffer

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
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{

}
```

总结：

1、object 里面的方法都是静态方法

2、object 里面的字段都是静态字段

3、它本身就是一个单例，(因为不需要去 new)

2.2、伴生对象

在 Scala 的类中，与类名相同的单例对象叫做伴生对象，也就是说如果我们在 object Dog 所在的文件内定义了一个 class Dog，此时：

1、 object Dog 被称为 class Dog 的伴生对象

2、 class Dog 被称为 object Dog 的伴生类

3、 类和伴生对象之间可以相互访问私有的方法和属性

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
class Dog {
  val id = 100
  private var name = "旺财"

  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:String = "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()
  }
}
```

总结：

伴生类 和 伴生对象 之间可以互相访问对方的私有属性

2.3、Apply 方法

在讲集合和数组的时候，可以通过 val intList=List(1,2,3)这种方式创建初始化一个列表对象，其实它相当于调用 val intList=List.apply(1,2,3)，只不过 val intList=List(1,2,3)这种创建方式更简洁一点，但我们必须明确的是这种创建方式仍然避免不了 new，它后面的实现机制仍然是 new 的方式，只不过我们自己在使用的时候可以省去 new 的操作。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。

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
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)
    println(arr2.toBuffer)
```

```
  }
}
```

2.4、应用程序对象 App

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。

```
package com.mazh.scala.day2.oop


/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  */
object AppObjectDemo extends App{
  //不用写main 方法
  println("I love Scala")
}
```

缺点：

无法为该类的执行，从外部传入参数，也既没有 main 方法的参数可用

2.5、抽象类

抽象类是一种不能被实例化的类，抽象类中包括了若干不能完整定义的方法，这些方法由子类去扩展定义自己的实现。

1、如果在父类中，有某些方法无法立即实现，而需要依赖不同的子类来覆盖，重写实现自己不同的方法实现。此时可以将父类中的这些方法**不给出具体的实现，只有方法签名**，这种方法就是抽象方法。

2、而一个类中如果有一个抽象方法，那么类就必须用 abstract 来声明为**抽象类**，此时抽象类是**不可以实例化**的

3、在子类中**覆盖抽象类的抽象方法时，不需要使用 override 关键字**

```
package com.mazh.scala.day2.oop


/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  定义抽象类
  */
abstract class Animal {
  //抽象字段(域)
  //前面我们提到，一般类中定义字段的话必须初始化，而抽象类中则没有这要求
  var height:Int
  //抽象方法
  def eat:Unit
}
```

```
//Person 继承Animal，对eat 方法进行了实现
//通过主构造器对height 参数进行了初始化
class Person(var height:Int) extends Animal{

  //对父类中的方法进行实现，注意这里面可以不加override 关键字
  def eat()={
    println("eat by mouth")
  }
}
```

总结：

Scala 抽象类的使用方式和 Java 中的抽象类的概念一致

3、Scala 继承

3.1、扩展类

在 Scala 中扩展类的方式和 Java 一样都是使用 extends 关键字

1、Scala 中，让子类继承父类，与 Java 一样，也是使用 extends 关键字

2、继承就代表，子类可以从父类继承父类的 field 和 method；然后子类可以在自己内部放入父类所没有，子类特有的 field 和 method；使用继承可以有效复用代码

3、子类可以覆盖父类的 field 和 method；但是如果父类用 final 修饰，field 和 method 用 final 修饰，则该类是无法被继承的，field 和 method 是无法被覆盖的

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  * 描述:  继承 测试
  */
object ExtendsDemo{
  def main(args: Array[String]): Unit = {
    val student=new Student ("黄渤",33,"1024")
    println(student.studentNo)
  }
}

//Person 类
class Person (name:String,age:Int){
  println("Person : "+ name + "\t" + age)
}

//Student 继承Person 类
class Student(name:String,age:Int,var studentNo:String) extends Person(name,age){
```

```
  println("Student : "+ name + "\t" + age + "\t" + studentNo)
}
```

问题：
Java 不支持多继承，为什么？

3.2、重写方法（Override 和 Super）

1、**Scala 中，如果子类要重写一个父类中的非抽象方法，则必须使用 override 关键字**
2、override 关键字可以帮助我们尽早地发现代码里的错误，比如：**override** 修饰的父类方法的方法名我们拼写错了；比如要覆盖的父类方法的参数我们写错了；等等
3、此外，在子类覆盖父类方法之后，如果我们在子类中就是要调用父类的被覆盖的方法呢？**那就可以使用 super 关键字，显式地指定要调用父类的方法**

```
package com.mazh.scala.day2.oop


/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  定义抽象类
  */
object OverrideDemo {
  def main(args: Array[String]): Unit = {
    val student = new Student222
    student.eat()
  }
}

abstract class Person222{
  def eat()
}

class Student222 extends Person222{
  // 因为父类是抽象类，所以 override 可以加可以不加
  override def eat(): Unit = {
    println("我要吃成金三胖")
  }
}
```

3.3、类型检查和转换

操作	Scala	Java
判断	obj.isInstanceOf[C]	obj instanceof C
转换	obj.asInstanceOf[C]	(C)obj
获取	classOf[C]	C.class

类型检查测试代码：

```
package com.mazh.scala.day2.oop

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  类型检查测试
  */
object TypeCheckDemo {

  def main(args: Array[String]): Unit = {

    val p = new Student_11()
    val p1 = new People_11()

    println(p.isInstanceOf[Student_11])
    println(p1.isInstanceOf[Student_11])
    println(classOf[Student_11])
    println(p.asInstanceOf[Student_11])
  }
}

class People_11{
}

class Student_11 extends People_11{
}
```

3.4、超类的构造

```
package com.aura.mazh.day2.oop

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  */
object ClazzDemo {
  def main(args: Array[String]) {
    val h = new Human
    println(h.fight)
  }
}

trait Flyable{
```

```
  def fly(): Unit ={
    println("I can fly")
  }

  def fight(): String
}

abstract class Animal1 {
  def run(): Int
  val name: String
}

class Human extends Animal1 with Flyable{

  val name = "abc"

  // 五个变量分别都赋值一次，那么{}也就相当于要执行五次
  val t1,t2,(a, b, c),t3,t4 = {
    println("ABC")
    (1,2,3)
  }

  println(a)
  println(t1._1)
  println(t1.hashCode() + "\t" + t2.hashCode())

  //在Scala 中重写一个非抽象方法必须用override 修饰
  def fight(): String = {
    "fight with 棒子"
  }
  //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
  }
}
```

4、特质 Trait

4.1、特质的定义

Scala 和 Java 语言一样，采用了很强的限制策略，避免了多继承的问题。在 Java 语言中，只允许继承一个超类，该类可以实现多个接口，但 Java 接口有其自身的局限性：接口中只能

包括抽象方法，不能包含字段、具体方法。Scala 语言利用 trait 解决了该问题，**在 Scala 的 trait 中，它不但可以包括抽象方法还可以包含字段和具体方法**。

trait 的示例如下：

```
package com.mazh.scala.day2.oop

trait DAO{
  //抽象字段
  val id:Int
  //具体字段
  val name:String = "huangbo"
  //带实现的具体方法
  def delete(id:String):Boolean = true
  //定义一个抽象方法，注意不需要加abstract,加了abstract 反而会报错
  def add(o:Any):Boolean
  def update(o:Any):Int
  def query(id:String):List[Any]
}
```

4.2、Trait 的使用

4.2.1、Trait 使用概述

Java 中：
1、接口是一个特殊的抽象类
2、里面所有的方法都是抽象方法。

Scala 中：
1、 特质里面的方法既可以实现，也可以不实现
那么跟抽象类有什么区别，两点原因：
1、优先使用特质。一个类扩展多个特质是很方便的，但却只能扩展一个抽象类。
2、如果你需要构造函数参数，使用抽象类。因为抽象类可以定义带参数的构造函数，而特质不行。例如，你不能说 trait t(i: Int) {}，参数 i 是非法的。
2、抽象类，我们用的是 extends，我们只能单继承，但是我们可以多实现
3、实现特质，如果没有继承其它类，那么使用第一个特质使用 extends，后面的使用 with，所以如果有考题说实现特质只能使用 with，这是不对的。

Trait 的几种不同使用方式
1、当做 Java 接口使用的 trait，全是抽象字段和抽象方法
2、带实现方法的 trait
3、带具体字段的 trait
Traits 的底层实现就是采用的 Java 的抽象类

4.2.1、将特质作为接口使用

Scala 中的 Triat 是一种特殊的概念
首先我们可以将 Trait 作为接口来使用，此时的 Triat 就与 Java 中的接口非常类似
在 Triat 中可以定义抽象方法，就与抽象类中的抽象方法一样，不给出方法的具体实现
在 Triat 中也可以定义具体方法，给出方法的具体实现
在 Triat 中可以定义具体字段，也可以定义抽象字段
类可以使用 extends 关键字继承 trait，注意，这里不是 implement，而是 extends，在 scala 中没有 implement 的概念，无论继承类还是 trait，统一都是 extends
类继承 trait 后，必须实现其中的抽象方法，实现时不需要使用 override 关键字
scala 不支持对类进行多继承，但是支持多重继承 trait，使用 with 关键字即可

```
package com.mazh.scala.day2.oop

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  定义抽象类
  */
trait MySQLDAO{
  val id:Int
  def add(o:Any):Boolean
  def update(o:Any):Int
  def query(id:String):List[Any]
}

//如果有多个trait 的话，则使用with 关键字即可
class DaoImpl extends MySQLDAO with Serializable{
  // 给父类中的抽象字段赋值。
  override val id = 12
  // 实现抽象方法
  def add(o:Any):Boolean = true
  def update(o:Any):Int = 1
  def query(id:String):List[Any] = List(1,2,3)
}
```

4.2.2、在 Trait 中定义具体方法和属性

```
package com.mazh.scala.day2.oop

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  定义抽象类
  */
```

```
trait MySQLDAO{

  val id:Int
  val name:String = "huangbo"

  def add(o:Any):Boolean
  def update(o:Any):Int
  def query(id:String):List[Any]
  def delete(id:Int) = {println("delete one record")}
}

//如果有多个trait 的话，则使用with 关键字即可
class DaoImpl extends MySQLDAO with Serializable{

  // 实现父类中的抽象方法，必须的
  def add(o:Any):Boolean=true
  def update(o:Any):Int= 1
  def query(id:String):List[Any]=List(1,2,3)

  // 给父类中的抽象字段赋值，必须的
  override val id = 12
}
```

4.3.5、为实例对象混入 Trait

有时我们可以在创建类的对象时，指定该对象混入某个 Trait，这样，就只有这个对象混入该 Trait 的方法，而类的其他对象则没有

```
package com.mazh.scala.day2.oop

trait MyLogger {
  def log(msg:String){}
}
trait Logger_A extends MyLogger{
  override def log(msg:String): Unit ={
    println("test:"+msg)
  }
}
trait Logger_B extends MyLogger{
  override def log(msg:String): Unit ={
    println("log:"+msg)
  }
}
```

```
class Person123(val name:String) extends Logger_A{
  def sayHello(): Unit ={
    println("Hi ,i'm name")
    log("sayHello is invoked!")
  }
}
object  MyLogger_Trait_Test{
  def main(args: Array[String]) {
    val p1=new Person123("liudehua")
    p1.sayHello()
    val p2=new Person123("zhangxueyou") with Logger_B
    p2.sayHello()
  }
}
```

4.3.6、Trait 调用链

Scala 中支持让类继承多个 Trait 后，依次调用多个 Trait 中的同一个方法，只要让多个 Trait 的同一个方法中，在最后都执行 **super.方法** 即可
类中调用多个 Trait 中都有的这个方法时，首先会从最右边的 Trait 的方法开始执行，然后依次往左执行，形成一个调用链条
这种特性非常强大，其实就相当于设计模式中的责任链模式的一种具体实现依赖

```
package com.mazh.scala.day2.oop

trait Handler {
  def handler(data:String){}
}
trait Handler_A extends Handler{
  override def handler(data:String): Unit = {
    println("Handler_A :"+data)
    super.handler(data)
  }
}
trait Handler_B extends Handler{
  override def handler(data:String): Unit = {
    println("Handler_B :"+data)
    super.handler(data)
  }
}
trait Handler_C extends Handler{
  override def handler(data:String): Unit = {
    println("Handler_C :"+data)
```

```
    super.handler(data)
  }
}
class  Person_TraitChain(val name:String) extends Handler_C with Handler_B with
Handler_A{
  def sayHello={
    println("Hello "+name)
    handler(name)
  }
}

object TraitChain_Test{
  def main(args: Array[String]) {
    val p=new Person_TraitChain("zhangxiaolong");
    p.sayHello
  }
}
```

运行结果：

```
Hello lixiaolong
Handler_A :zhangxiaolong
Handler_B :zhangxiaolong
Handler_C :zhangxiaolong
```

5、Scala 的模式匹配

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

5.1、匹配字符串

```
package com.mazh.scala.day2.oop

import scala.util.Random

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object CaseDemo01 extends App{
  val arr = Array("huangbo", "xuzheng", "wangbaoqiang","xxxx")
  val name = arr(Random.nextInt(arr.length))
```

```
  // name 模式匹配字符串
  name match {
    case "huangbo" => println("影帝来也...")
    case "xuzheng" => println("喜剧笑星来了...")
    case "wangbaoqiang" => println("实力干将来也...")
    case _ => println("谁...？？？")
  }
}
```

5.2、匹配类型

```
package com.mazh.scala.day2.oop

import scala.util.Random

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  */
object CaseDemo02 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)

  // 模式匹配
  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")
  }
}

case class CaseDemo()
```

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

5.3、匹配数组、元组、集合

```
package com.mazh.scala.day2.oop

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
```

```
  * 描述：  匹配数组和元组和集合
  */
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, 7)
  tup match {
    case (1, 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))

5.4、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

```
package com.mazh.scala.day2.oop


import scala.util.Random


/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：
  */
```

```
object CaseDemo04 extends App{

  val arr = Array(CheckTimeOutTask, HeartBeat(123), SubmitTask("0001", "task1"))

  arr(Random.nextInt(arr.length)) match {
    case SubmitTask(id, name) => {
      println(s"$id, $name")
    }
    case HeartBeat(time) => {
      println(time)
    }
    case CheckTimeOutTask => {
      println("check")
    }
  }
}

case class SubmitTask(id: String, name: String)
case class HeartBeat(time: Long)
case object CheckTimeOutTask
```

当一个类被声名为 case class 的时候，scala 会帮助我们做下面几件事情：

1、构造器中的参数如果不被声明为 var 的话，它默认的话是 val 类型的，但一般不推荐将构造器中的参数声明为 var

2、自动创建伴生对象，同时在里面给我们实现子 apply 方法，使得我们在使用的时候可以不直接显示地 new 对象

3、伴生对象中同样会帮我们实现 unapply 方法，从而可以将 case class 应用于模式匹配 apply 方法接受参数返回对象，unapply 方法接收对象返回参数

4、实现自己的 toString、hashCode、copy、equals 方法

5、case class 主构造函数里面没有修饰符，默认的是 val

除此之此，case class 与其它普通的 scala 类没有区别

```
package com.aura.mazh.day2.oop


/**
  * 作者:  马中华   https://blog.csdn.net/zhongqi2513
  * 时间:  2018/7/16 8:15
  *
  * 描述:  测试 unapply 方法
  */
object UnapplyTest {

  // 接收参数，返回对象，  一般用作工厂
  def apply(value: Double, unit: String): Currency = new Currency(value, unit)

  // 接收对象，返回参数，  一般用于模式匹配
```

```
  def unapply(currency: Currency): Option[(Double, String)] = {
    if (currency == null){
      None
    }
    else{
      Some(currency.value, currency.unit)
    }
  }
}

class Currency(val value: Double, val unit: String) {
}
```

5.5、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

```
package com.mazh.scala.day2.oop

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  * 描述:
  */
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)
  }
}
```

5.6、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 代表返回值类型**，常用作输入模式匹配

```
package com.mazh.scala.day2.oop


/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  偏函数测试Demo
  */
object PartialFuncDemo  {
  //偏函数
  def func1: PartialFunction[String, Int] = {
    case "one" => 1
    case "two" => 2
    case _ => -1
  }
  //普通函数的模式匹配实现
  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"))
  }
}
```

总结：

偏函数，就是用来做模式匹配的