Scala核心编程

-面向对象编程(基础部分)

尚硅谷研究院

类与对象

- 类与对象

看一个养猫猫问题

张老太养了只**猫**猫:一只**名字**叫小白,今年**3岁**,白色。还有一只叫小花,今年10岁,花色。**请编写一个程序**，当用户输入小猫的名字时，就显示该猫的名字，年龄，颜色。如果用户输入的小猫名错误，则显示 张老太没有这只猫猫。

//问题

1. 猫有三个属性，类型不一样.
2. 如果使用普通的变量就不好管理
3. 使用一种新的**数据类型**((1) 可以管理多个不同类型的数据 [**属性**]) (2) 可以对属性进行操作-**方法**
4. 因此类与对象

- 类与对象

使用现有技术解决

1) 单独的定义变量解决

现有技术解决的缺点分析

1) ?
2) ?

- 类与对象

一个程序就是一个世界，有很多对象

- 类与对象

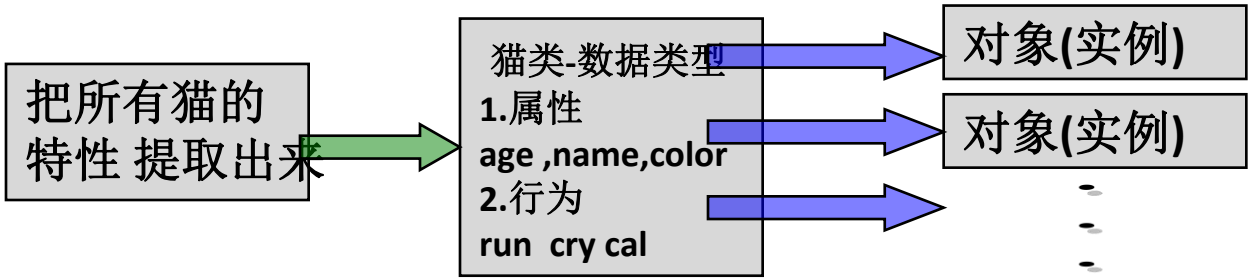
Scala语言是面向对象的

- Java是面向对象的编程语言，由于历史原因，Java中还存在着非面向对象的内容:基本类型 ，null，静态方法等。
- Scala语言来自于Java，所以天生就是面向对象的语言，而且Scala是纯粹的面向对象的语言，即在Scala中，一切皆为对象。

- 在面向对象的学习过程中可以对比着Java语言学习

- 类与对象

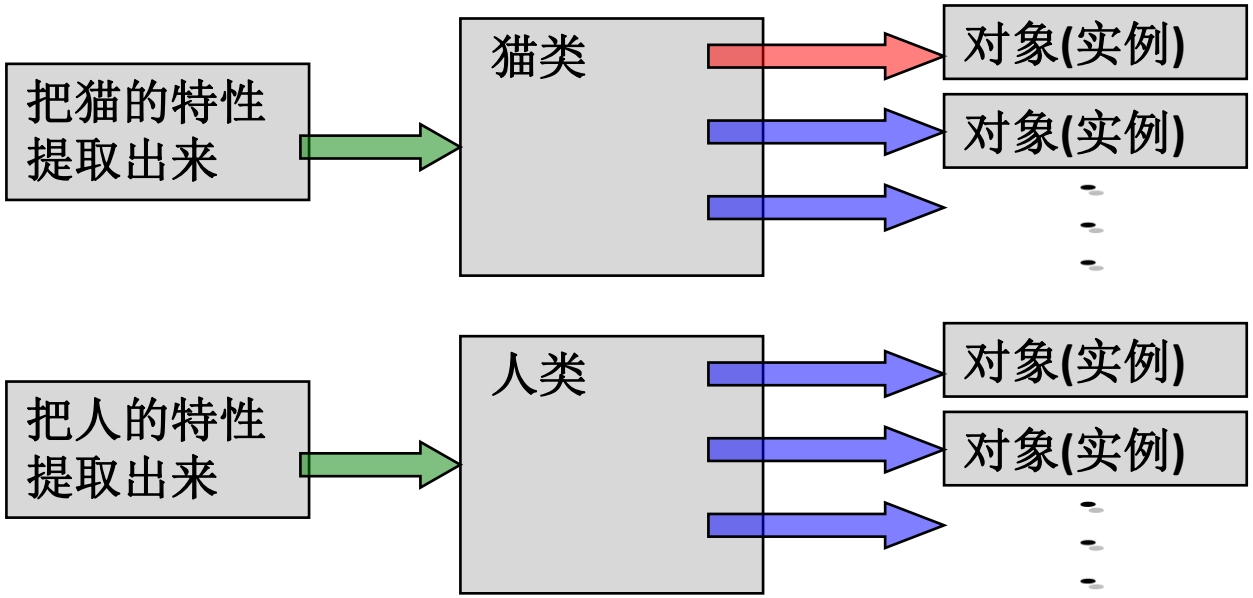
类与对象的关系示意图



- 对上图说明

- 类与对象

类与对象的关系示意图



说明：Scala中类与对象的关系，和Java语言中的类与对象的关系一样。

- 类与对象

快速入门-面向对象的方式解决养猫问题

- 类与对象

类和对象的区别和联系

通过上面的案例和讲解我们可以看出:

1) 类是抽象的，概念的，代表**一类事物**,比如人类,猫类..
2) 对象是具体的，实际的，代表**一个具体**事物
3) 类是对象的模板，对象是类的一个个体，对应一个实例
4) Scala中类和对象的区别和联系 和 Java是一样的。

- 类与对象

如何定义类

➢ 基本语法

[修饰符] class 类名 {
类体
}

➢ **定义类的注意事项**

1) scala语法中，类并不声明为public，所有这些类**都具有公有可见性(即默认就是public)**,[修饰符在后面再详解].
2) 一个Scala源文件可以包含多个类.

```
class Tiger {
  var name = ""
  var age = 10
}
```

```
class Dog {
  var name = ""
  var age: Int = _
}
```

- 类与对象

属性

- 基本介绍

1) 案例演示：

2) 属性是类的一个组成部分，一般是**值数据类型**,也**可是引用类型**。比如我们前面定义猫类 的 age 就是属性

- 类与对象

属性/成员变量

➢ 注意事项和细节说明

1) 属性的定义语法同变量，示例：**[访问修饰符] var 属性名称 [：类型] = 属性值**
2) 属性的定义类型可以为任意类型，包含值类型或引用类型[案例演示]

3) Scala中声明一个属性,必须**显示的初始化**，然后根据初始化数据的类型自动推断，属性类型可以省略(这点和Java不同)。[案例演示]

4) 如果赋值为null,则一定要加类型，因为不加类型, 那么该属性的类型就是Null类型.

```
class Person {
  var age : Int = 10
  var sal = 8090.9  //给属性赋初值，省略类型，会自动推导
  var Name  = null  // Name 是什么类型?
  var address : String = null // address 是什么类型?
}
```

- 类与对象

属性/成员变量

➢ 注意事项和细节说明

5) 如果在定义属性时，暂时不赋值，也可以使用符号**_(下划线)**，让系统分配默认值.

类型	_ 对应的值
Byte Short Int Long	0
Float Double	0.0
String 和 引用类型	null
Boolean	false

6) 不同对象的属性是独立，互不影响，一个对象对属性的更改，不影响另外一个。
案例演示+图(Monster) //这点和java完全一样

- 类与对象

属性的高级部分

说明： **属性的高级部分**和构造器(构造方法/函数)相关，我们把属性高级部分放到构造器那里讲解。

- 类与对象

如何创建对象

➢ 基本语法

val | var 对象名 [：类型] = new 类型()

➢ 说明

1) 如果我们不希望改变对象的引用(即：内存地址), 应该声明为val 性质的，否则声明为var, scala设计者推荐使用**val** ,因为一般来说，在程序中，我们只是改变对象属性的值，而不是改变对象的引用。
2) scala在声明对象变量时，可以根据创建对象的类型自动推断，所以类型声明可以省略，但当类型和后面new 对象类型有继承关系即多态时，就必须写了

- 类与对象

如何访问属性

- 基本语法

对象名.属性名;

案例演示赋值和输出

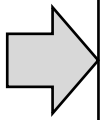
- 类与对象

类和对象的内存分配机制(重要)

- 看一个思考题
  我们定义一个Person类(包括 名字,年龄)， 创建一个对象(ObjectDemo.scala)

我们看看下面一段代码：

```
val p1 = new Person
p1.name = "jack"
p1.age = 30
val p2 = p1     //内存布局?
```



```
println("p2.age=" + p2.age)
println("p1.age="+ p1.age)
```

请问：p2.age究竟是多少?

- 类与对象

类和对象的内存分配机制

➢ Scala对象的内存布局

对象是存在内存中的, 那么对象在内存中究竟是怎样存在的?

这里我们给大家伙画一个图来说明一下, 这个图对我们理解对象在内存中如何存在是非常重要的.

对象在内存的

scala对象内存布局(

- 方法

基本说明

Scala中的方法其实就是函数，声明规则请参考函数式编程中的函数声明。

基本语法

def 方法名(参数列表) [：返回值类型] = {

方法体

}

方法案例演示

给Cat类添加cal方法,可以计算两个数的和

```
class Dog {
  private var sal: Double = _
  var food : String = _

  def cal(n1: Int, n2: Int): Int = {
    return n1 + n2
  }}
```

- 方法

方法的调用机制原理

提示：程序调用方法过程+说明


方法调用机

1) 当我们scala开始执行时，先在栈区开辟一个main栈。main栈是最后被销毁
2) 当scala程序在执行到一个方法时，总会开一个新的栈。
3) 每个栈是独立的空间，变量（基本数据类型）是独立的，相互不影响（引用类型除外）
4) 当方法执行完毕后，该方法开辟的栈就会被jvm机回收。

- 方法

课堂练习题

1) 编写类(**MethodExec**)，编程一个方法，方法不需要参数，在方法中打印一个10*8 的矩形，在main方法中调用该方法。【案例演示】

2) 修改上一个程序，编写一个方法中，方法不需要参数，计算该矩形的面积，并将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印(结果保留小数点2位)。【案例演示】

3) 修改上一个程序，**编写一个方法**，提供m和n两个参数，方法中打印一个m*n的矩形，**再编写一个方法**算该矩形的面积(可以接收长len，和宽width)， 将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印。【课堂】

4) 编写方法：判断一个数是奇数odd还是偶数 【课堂】 [10min]

- 成员方法

课堂练习题

4) 编写方法：判断一个数是奇数odd还是偶数 【课堂】
5) 根据行、列、字符打印 对应行数和列数的字符，比如：行：3，列：2，字符*,则打印相应的效果【课后】
6) 定义小小计算器类(Calcuator)，实现加减乘除四个功能 【课后】
   实现形式1：分四个方法完成:
   实现形式2：用一个方法搞定

- 类与对象应用实例

小狗案例

1) 编写一个Dog类，包含name(String)、age(Int)、weight(Double)属性
2) 类中声明一个say方法，返回String类型，方法返回信息中包含所有属性值。

3) 在另一个TestDog类中的main方法中，创建Dog对象，并访问say方法和所有属性，将调用结果打印输出。

盒子案例(学员，课后)

1) 编程创建一个Box类，在其中定义三个变量表示一个立方体的长、宽和高，长宽高可以通过控制台输入。
2) 定义一个方法获取立方体的体积(volumn)。长*宽*高
3) 创建一个对象，打印给定尺寸的立方体的体积。

- 创建和使用类与对象应用实例

景区门票案例(学员课后)

1) 一个景区根据游人的年龄收取不同价格的门票。
2) 请编写游人类，根据年龄段决定能够购买的门票价格并输出
3) 规则：年龄>18 , 门票为20元，其它情况免费。
4) 可以循环从控制台输入名字和年龄，打印门票收费情况, 如果名字输入 n , 则退出程序。

- 构造器

看一个需求

我们来看一个需求：前面我们在创建Person的对象时，是先把一个对象创建好后，再给他的**年龄**和**姓名**属性赋值，如果现在我要求，在创建人类的对象时，就直接指定这个对象的年龄和姓名，该怎么做？这时就可以使用**构造方法/构造器**。

回顾-Java构造器基本语法

```
[修饰符] 方法名(参数列表){
        构造方法体
}
```

- 构造器

基本介绍

构造器(constructor)又叫构造方法，是类的一种特殊的方法，它的主要作用是完成对**新对象的初始化**。

- 构造器

回顾-Java构造器的特点

- 在Java中一个类可以定义多个不同的构造方法，构造方法重载
- 如果程序员**没有定义构造方法**，系统会自动给类生成一个**默认无参构造方法**(也叫默认构造器)，比如 Person (){}
- 一旦定义了自己的构造方法（构造器）,默认的构造方法就覆盖了，就不能再使用默认的无参构造方法，除非**显示的定义一下**,即：Person(){};

- 构造器

回顾-Java构造器的案例

在前面定义的Person类中添加两个构造器：

第一个无参构造器：利用构造器设置所有人的age属性初始值都为18

第二个带name和age两个参数的构造器：使得每次创建Person对象的同时初始化对象的age属性值和name属性值。

```
class Person{
    public String name;
    public int age;
    public String getInfo(){
        return name+"\t"+age;
    }
    public Person(){
        age = 18;
    }
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }}
```

- 构造器

Scala构造器的介绍

和Java一样，Scala构造对象也需要调用构造方法，并且可以有任意多个构造方法（即scala中构造器也支持重载）。
Scala类的构造器包括： 主构造器 和 辅助构造器

Scala构造器的基本语法

```
class 类名(形参列表) { // 主构造器
  // 类体
  def this(形参列表) { // 辅助构造器
  }
  def this(形参列表) { //辅助构造器可以有多个...
  }
}
//1. 辅助构造器 函数的名称this, 可以有多个，编译器通过不同参数来区分.
```

- 构造器

Scala构造器的快速入门

创建Person对象的同时初始化对象的age属性值和name属性值，案例演示。

```
var name: String = inName
var age: Int = inAge
```


图片3.z

- 构造器

Scala构造器注意事项和细节

1) Scala构造器作用是完成对**新对象的初始化，构造器没有返回值。**
2) **主构造器的声明直接放置于类名之后** [反编译]

3) **主构造器会执行类定义中的所有语句**，这里可以体会到Scala的函数式编程和面向对象编程融合在一起，**即：构造器也是方法（函数）**，传递参数和使用方法和前面的函数部分内容没有区别【案例演示+反编译】
4) 如果主构造器无参数，小括号可省略，构建对象时调用的构造方法的小括号也可以省略

```
class AA {

}

val a = new AA
val b = new AA()
```


图片4.z

- 构造器

Scala构造器注意事项和使用细节

5) 辅助构造器名称为this（这个和Java是不一样的），多个辅助构造器通过不同参数列表进行区分， 在底层就是f构造器重载。【案例演示+反编译】

```
class Person() {
  var name: String = _
  var age: Int = _
  def this(name : String) {
    //辅助构造器无论是直接或间接，最终都一定要调用主构造器，执行主构造器的逻辑
    //而且需要放在辅助构造器的第一行[这点和java一样，java中一个构造器要调用同类的其它构造器，也需要放在第
    this() //直接调用主构造器
    this.name = name
  }
  def this(name : String, age : Int) {
    this() //直接调用主构造器
    this.name = name
    this.age = age
  }
  def this(age : Int) {
    this("匿名") //简介调用主构造器,因为 def this(name : String) 中调用了主构造器!
    this.age = age
```

main函数中:

```
val p1 = new Person("scott")
p1.showInfo()
```

- 构造器

Scala构造器注意事项和使用细节

6) 如果想让主构造器变成私有的，可以在()之前加上private，这样用户只能通过辅助构造器来构造对象了【反编译】

7) 辅助构造器的声明不能和主构造器的声明一致,会发生错误(即构造器名重复)

图片5.z

```
class Person private() {
  var name: String = _
  var age: Int = _
  def this(name : String) {
    this() //Error:(13, 5) ambiguous reference to overloaded
    this.name = name
  }
  def this() {
    this()
  }
}
```

这里的this() 和 主构造器一致，导致13行在调用this() 不明确ambiguos

```
es Build
Error:(13, 5) ambiguous reference to overloaded definition,
both constructor Person in class Person of type ()com.atguigu.chapter02.Person
and  constructor Person in class Person of type ()com.atguigu.chapter02.Person
match argument types ()
```

- 属性高级

前面我们讲过属性了，这里我们再对属性的内容做一个加强.

构造器参数

1) Scala类的主构造器的形参未用任何修饰符修饰，那么这个参数是局部变量。

```
class Person( name : String ) {
```

2) 如果参数使用val关键字声明，那么Scala会将参数作为类的私有的只读属性使用 【案例+反编译】
3) 如果参数使用var关键字声明，那么那么Scala会将参数作为类的**成员属性**使用,并会提供属性对应的xxx()[类似getter]/xxx_$eq()[类似setter]方法，即这时的**成员属性是私有的，但是可读写**。【案例+反编译】

- 属性高级

Bean属性

JavaBeans规范定义了Java的属性是像getXxx（）和setXxx（）的方法。许多Java工具（框架）都依赖这个命名习惯。为了Java的互操作性。将Scala字段加@BeanProperty时，这样会自动生成**规范的 setXxx/getXxx 方法**。这时可以使用 对象.setXxx() 和 对象.getXxx() 来调用属性。

注意:给某个属性加入@BeanPropetry注解后，会生成getXXX和setXXX的方法并且对原来底层自动生成类似xxx(),xxx_$eq()方法，**没有冲突，二者可以共存**。

```
import scala.beans.BeanProperty
class Car {
  @BeanProperty var name: String = null
}
```

案例演示+反编译

- 对象创建的流程分析

看一个案例

```
class Person {
  var age: Short = 90
  var name: String = _
  def this(n: String, a: Int) {
    this()
    this.name = n
    this.age = a
  }}
var p : Person = new Person("小倩",20)
```

流程分析(面试题-写出)

1) 加载类的信息(属性信息，方法信息)
2) 在内存中(堆)开辟空间
3) 使用父类的构造器(主和辅助)进行初始
4) 使用主构造器对属性进行初始化 【age:90, naem nul】
5) 使用辅助构造器对属性进行初始化 【 age:20, naem 小倩 】
6) 将开辟的对象的地址赋给 p这个引用

谢谢！ 欢迎收看！