# 课堂笔记

## 继承

关键字：extends

目的：扩展类的功能，父类**代码复用，不可以集成多个父类。**

**可以extends多个trait（相当于Java里的接口）。**

**没有Java里的implement关键词**

extends trait1 with trait2 with trait3

案例：

父类代码：

**class** Father2(name:*String*, age:Int) {

**def** doEat(food:*String*)

{

println("eatting .."+food)

}

**def** printInfo2()

{

println("name:"+name+",age:"+age)

}

}

子类：

**class** Person2(name:*String*,age:Int,addr:*String*) **extends** Father2(name,age)

{

**override** **def** doEat(food:*String*)

{

println("my eatting .."+food)

}

**def** printInfo()

{

**super**.printInfo2()

println("name:"+name+",age:"+age+",addr:"+addr)

}

}

子类把name和age传入到父类中。

调用：

**var** p = **new** Person2("张三",22,"北京")

p.printInfo()

规则梳理：

· 继承关键词extends

· 某类如果不想被继承，可定义为final类型

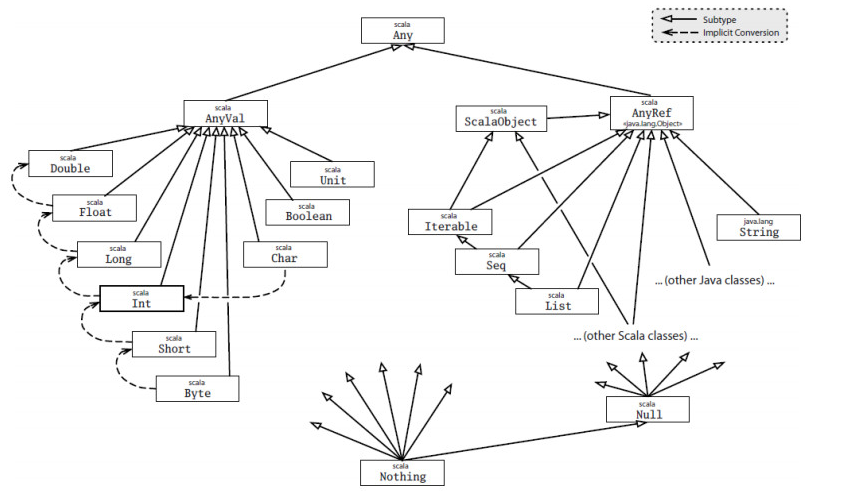
· 用super调父类的方法或属性。

· 重写方法时必须用override，可以重写字段，不想被override则定义为final类型。

· 只有主构造器可以调父类的主构造器，如上案例中。

## Scala类层级结构

Scala里，每个类都继承自通用的名为Any的超类。因为所有的类都是Any的子类，所以定义在Any中的方法就是“共同的”方法：它们可以被任何对象调用。



**顶级父类Any类的内容：**

final def ==(that: Any): Boolean

final def !=(that: Any): Boolean

def equals(that: Any): Boolean

def hashCode(): Int

def toString(): String

因为每个类都继承自Any，所以Scala程序里的每个对象都能用==、!=或equals比较，用hashCode做散列，以及用toString转为字符串。Any类里的等号和不等号方法被声明为final，因此他们不能再子类里重写。实际上，==总是与equals相同，!=总是与equeal相反。因此，独立的类可以通过重写equals方法改变==或!=的意义。

Any有两个子类：AnyVal和AnyRef(相当于Java里的Object)

AnyVal是Scala里每个内建值类的父类。有9个这样的值类：Byte、Short、Char、Int、Long、Float、Double、Boolean和Unit。其中的前8个都对应到Java的基本类型。这些值类都被定义为既是抽象的又是final的，不能使用new创造这些类的实例。Unit被用作不返回任何结果的方法的结果类型。Unit只有一个实例值，写成()。

AnyRef类是Scala里所有引用类(reference class)的基类。它其实是Java平台上java.lang.Object类的别名。因此Java里写的类和Scala里写的都继承自AnyRef。

Scala类与Java类的不同在于它们还继承自一个名为ScalaObject的特别trait。是想要通过ScalaObject包含的Scala编译器定义和实现的方法让Scala程序的执行更高效

scala.Null和scala.Nothing是用统一的方式处理Scala面向对象类型系统的某些“边界情况”的特殊类型。

Null类是null引用对象的类型，它是每个引用类（继承自AnyRef的类）的子类。Null不兼容值类型。

Nothing类型在Scala的类层级的最低端；它是任何其他类型的子类型。然而，根本没有这个类型的任何值。Nothing的一个用处是它标明了不正常的终止。

### 对比操作

Scala程序里的每个对象都能用==、!=或equals比较，用hashCode做散列，以及用toString转为字符串。Any类里的等号和不等号方法被声明为final，因此他们不能再子类里重写。实际上，==总是与equals相同，!=总是与equeal相反。因此，独立的类可以通过重写equals方法改变==或!=的意义。

另外测试某个对象是否属于某个类时，还可以用isInstanceOf方法，而asInstanceOf方法是转换引用。

### 抽象类，很少用

抽象类很少用，了解下即可。

如果不希望某类被实例化的话，用abstract定义为抽象类，通常用作父类，定义些公共函数或字段默认是abstract型的，不提供函数体实现，而且子类必须全部override。