依照尚硅谷b站Scala视频教程和tour of scala中文文档进行学习记录；感谢。。。

视频链接：https://www.bilibili.com/video/BV1Xh411S7bP

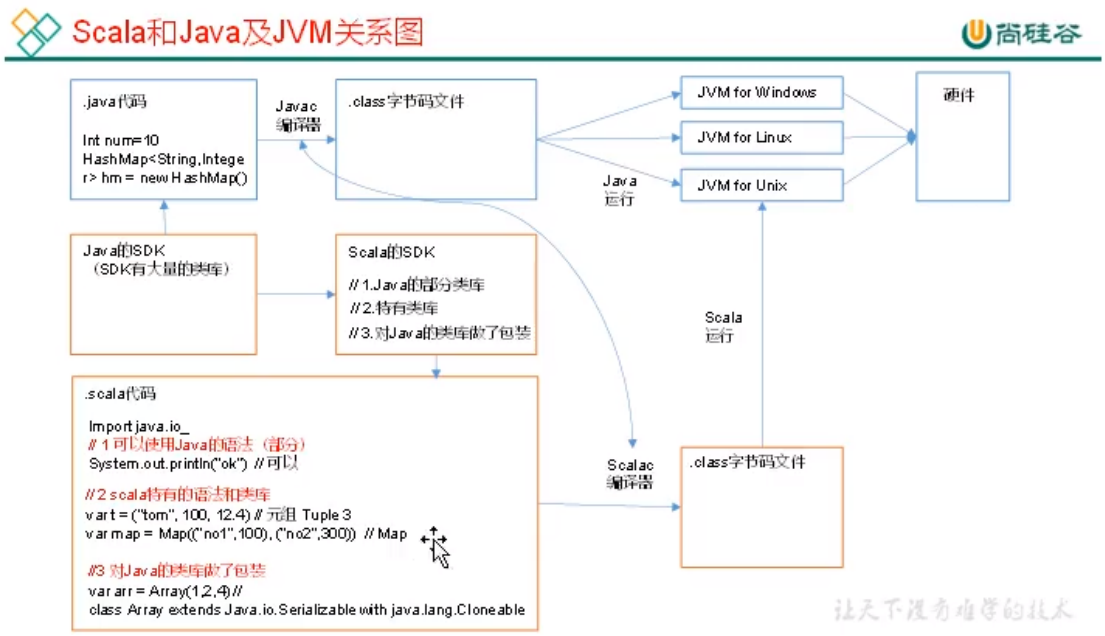
文档链接：https://docs.scala-lang.org/zh-cn/tour/tour-of-scala.html

文档整理记录了Scala一些基础概念和基本语法以及Scala最关键的函数式编程的一些关键点和代码

1.scala简介

- 基于jvm，java++;
- 更加面向对象
- 函数式编程
- 更适合大数据处理：对集合数据类型的支持，spark底层（**暂不能清出的做出解释**）

Java和Scala两者无缝连接：.java文件和.scala文件都是**编译**为.class字节码文件通过JVM**解释**运行，而且Scala可以直接调用Java库



图来源是b站尚硅谷Scala教程

2.scala基础知识

基础知识的简单介绍！

基础

表达式：和普通编程语言一样

常量`val`，变量`var`，代码块{}

函数

所有的函数都必须有返回，Unit可以做空返回。

在scala里函数是一个表达式， 函数是带有参数的表达式，也是一个对象。如：

```
val addOne = (x:Int) => x+1;
```

也可以是匿名函数

```
(x:Int) => x+1;
```

`=>`的左边是参数列表，右边是一个包含参数的表达式。

方法

方法在功能上和函数类似；方法由`def`关键字定义。如：

```
def addOne(x:Int) : Int ={ x + 1};
```

和一般编程习惯不同，`def`后面跟着名字、参数列表、返回类型和方法体。参数列表可以是多个，也可以没有。scala有return关键字，但一般不用，方法体的最后一个表达式的值就是方法的返回值。

类

普通类:class

实用class定义一个类，后面跟着类名和构造参数；如：

```
class Student(name:String, age:Int){

    def greet(): Unit = {

        println("Hello, I'm " + name);

    }

}
```

你可以使用`new`关键字创建一个类的实例。如：

```
val std = new Student("chenhuan", 18)
```

样例类:case class

默认情况下，样例类一般用于不可变对象，并且可作值比较。

```
case class Point(x: Int, y: Int)
```

object

object翻译为对象，区别于一般编程语言的对象，这里的对象并不是Java、C++等面向对象编程语言里的类的实例；实际上在Scala里object的出现很大程度是因为，Scala取消了Java里的static关键字。object相当于一个单实例（**这里并不是很清楚**），可以简单的理解：object定义出的对象全局仅有一个，类似Java的static功能。实例：

```
object StudentIdFactory {
  private var counter = 0
  def create(): Int = {
    counter += 1
    counter
  }
}
```

上述代码块相当于声明了一个全局static变量counter，实现了ID自增；可以通过引用对象的名字来访问一个对象。

```
val newId: Int = IdFactory.create()
```

trait

暂且和Java的虚基类这种东西类比理解吧，后续学了用了再做详细更新

更新：相当于把一系列类所共有的特征抽象出来了，比如大部分都可以比较大小，定义一个比较大小的trait，抽象出比较大小的函数；让这些类都继承这个trait来实现各自比较大小的细节。

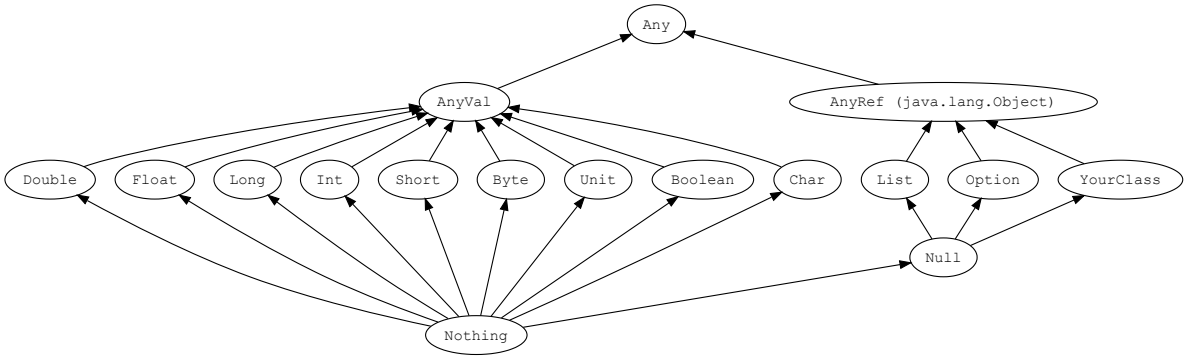
统一类型

基本类型

`Any`是所有类型的超类型，也称为顶级类 型。它定义了一些通用的方法如`equals`、`hashCode`和`toString`。`Any`有两个直接子类：`AnyVal`和`AnyRef`。

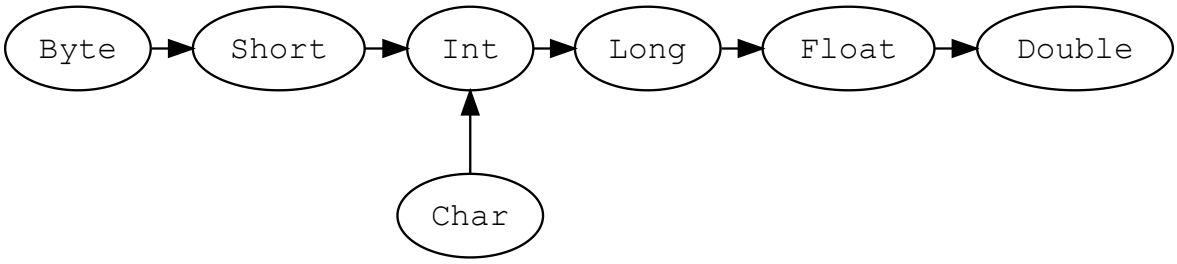
`AnyVal`代表值类型。有9个预定义的非空的值类型分别是：`Double`、`Float`、`Long`、`Int`、`Short`、`Byte`、`Char`、`Unit`和`Boolean`。`Unit`是不带任何意义的值类型，它仅有一个实例可以像这样声明：`()`。**所有的函数必须有返回**，所以说有时候`Unit`也是有用的返回类型。

`AnyRef`代表引用类型。所有非值类型都被定义为引用类型。在Scala中，每个用户自定义的类型都是`AnyRef`的子类型。如果Scala被应用在Java的运行环境中，`AnyRef`相当于`java.lang.Object`。



类型转换

类型间可以进行转换，但有一定规则，一些转换是无法通过编译的；总结起来就是，只能从小转到大，因为大到小会造成数据丢失；小到大补0就可以了。



Nothing

`Nothing`是所有类型的子类型，也称为底部类型。没有一个值是`Nothing`类型的。它的用途之一是给出非正常终止的信号，如抛出异常、程序退出或者一个无限循环（可以理解为它是一个不对值进行定义的表达式的类型，或者是一个不能正常返回的方法）。

3.函数式编程

3.1函数的定义

Scala中函数可以写定义在一个函数里或一个类里，这里提到的函数就是第2部分提到的方法，是通过def关键字定义的，后面所用到的函数若无特殊声明都是指方法。有下面的两种种定义和调用方式，一个是在类里定义，一个是在main函数里定义；

```
object Test01_FunctionAndMethod {
  def main(args: Array[String]): Unit = {
    // 在main函数里定义函数
    def sayHi(name: String): Unit = {
      println("hi, " + name)
    }

    // 调用函数
    sayHi("alice")

    // 调用对象方法
    Test01_FunctionAndMethod.sayHi("bob")

    // 获取方法返回值
    val result = Test01_FunctionAndMethod.sayHello("cary")
    println(result)
  }

  // main函数外定义对象的方法
  def sayHi(name: String): Unit = {
    println("Hi, " + name)
  }

  def sayHello(name: String): String = {
    println("Hello, " + name)
    return "Hello"
  }
}
```

3.2函数的参数

可变参数：动态给定参数列表，不确定参数个数，可变参数一般放在最后；

参数默认值：在调用时可以不给定参数；

带名参数：函数调用时通过名字给指定的参数传值；

```
object Test03_FunctionParameter {
  def main(args: Array[String]): Unit = {
    //    （1）可变参数
    def f1(str: String*): Unit = {
      println(str)
    }

    f1("alice")              //WrappedArray(alice)
    f1("aaa", "bbb", "ccc")//WrappedArray(aaa, bbb, ccc)

    //    （2）如果参数列表中存在多个参数，那么可变参数一般放置在最后
    def f2(str1: String, str2: String*): Unit = {
      println("str1: " + str1 + " str2: " + str2)
    }
    f2("alice")//str1: alice str2: WrappedArray()
    f2("aaa", "bbb", "ccc")//str1: aaa str2: WrappedArray(bbb, ccc)

    //    （3）参数默认值，一般将有默认值的参数放置在参数列表的后面
    def f3(name: String = "atguigu"): Unit = {
      println("My school is " + name)
    }

    f3("school")//My school is school
    f3()//My school is atguigu

    //    （4）带名参数
    def f4(name: String = "atguigu", age: Int): Unit = {
      println(s"${age}岁的${name}在尚硅谷学习")
    }

    f4("alice", 20)
    f4(age = 23, name = "bob")
    f4(age = 21)
  }
}
```

3.3函数至简原则

3.3.1省略return

函数默认以最后一行的表达式的值作为返回：

```
def f0(str:String) : String ={
  return str;
}
def f1(str:String) : String ={
  str;
}
println(f0("chenhuan"))//chenhuan
println(f1("chenhuan"))//chenhuan
```

3.3.2省略花括号

函数只有一行代码可以直接省略花括号：

```
def f0(str:String) : String = str;
```

3.3.3省略返回类型

如果编译器能够推断出返回值类型则可以直接删除返回值类型，但是如果用了return则返回类型不能省略；

```
def f0(str:String) = str;
```

3.3.4省略括号

如果函数是空参数，则参数括号可以省略；

```
def f1(): Unit = {
  println("chenhuan")
}
f1()//chenhuan
f1//chenhuan
```

3.3.5省略名称*

如果只关心处理逻辑，可以省略def关键字和名称；

```
//(str:String) => str 是匿名函数;这就事lambda表达式
(str:String) => {return str};
//相当于把(str:String) => str;这个逻辑处理过程看做一个val 赋值给f1，这个val的类型就是函数，之后就可以通过f1来调用这个逻辑过程。这里就和第2部分的函数统一了，所以Scala里函数方法在都可称做函数
val f1 = (str:String) => {return str};
```

3.3.6函数作为参数

(str:String) => str;这个逻辑处理过程可以看做一个val值，那么它其实可以作为函数的形参进行传递；也就是把一个lambda表达式作为一个函数的参数。这就是高阶函数了，

```
val fun = (name: String) => {
  println(name)
}

// 定义一个函数，以函数作为参数输入，数据"chenhuan"写死了，逻辑操作作为参数
//def 函数名（参数名：类型）：返回值 = {执行体 }；String => Unit表示一个输入为String返回为Unit的函数
```

```
def f(func: String => Unit): Unit = {
  func("chenhuan")
}

//下面是f的三种调用方式
f(fun)   //传穿函数变量
f(name => println(name))//传lambda表达式，省略了一些括号、类型
f( println )//直接传入println函数，对数据进行println操作；println是一个输入为String返回为Unit的函数
```

3.4高阶函数

scala底层其实是把函数或者一个lambda表达式作为一个函数对象，完全可以把它们按对象的使用方法来使用，下面三种情形都是把函数整体作为一个变量或常量来使用。

- 函数可以作为值进行传递
- 函数可以作为参数进行传递
- 函数可以作为函数的返回值

```
def fun(num:Int): Int = num+1


val res = fun(2) //调用函数，把返回值赋值给res
val f0 = fun _//函数作为值传递给常量f
val f1 :Int=>Int = fun //确定f是一个Int参数一个Int返回值的函数后可以省略下划线_


def f2(func:Int=>Int) : Int = func(3)  //
f2(f1)// 函数作为参数传递


def f3(): Int=>Int = {
    return f0
}
println(f3()(2))  //f3()执行后返回值是一个函数，也就是f0，这里相当于执行了f0(2)
```

3.5函数柯里化&闭包

3.5.1闭包

函数能够访问它外部（局部）变量的值；

```
    def func1(num1:Int): Int = {
      def func2(num2:Int): Int = num1 + num2// func2内部访问到了func1的参数num1

      func2(3)
    }
    println(func1(2))
```

3.5.2柯里化

函数参数列表化；

3.6其他知识点

3.6.1尾递归

Scala编译器支持尾递归，在函数调用时是直接覆盖函数指针，而不是压栈；

```
@tailrec   //也可以通过注解的方式来告诉编译器这里是一个尾递归
def tailfact(n:Int): Int ={
    def loop(n:Int, cur_res:Int): Int = {
        if (n<=1) return cur_res;
        loop(n-1,cur_res * n)//编译器自动识别尾递归
    }
    loop(n, 1)
}
```

3.6.2控制抽象

意思就是把一部分控制逻辑抽象成为一个对象，可以像一个对象一样去使用它

值调用：直接传递计算后的值，和普通的用法一样

名调用：传递一个可执行代码块

```
def f1(a: =>Int): Int = a//'=>Int'表示一个代码块，返回值是Int，只要在函数a出现一次，代码块a就执行一次

//代码块可以是：函数，23，

while(condition){
    print('')
}

def myWhlie(condition: =>Boolean): (=>Unit)=>Unit = {
    def doLoop(codeblock: =>Unit): Unit = {
        if(condition){
            codeblock
            myWhile(condition)(codeblock)
        }
    }
    doLoop _
}
```

3.6.3惰性函数，惰性加载

在一个函数被使用时才去做加载执行；

```
lazy val res: Int = sum(12,12)   //这里不会执行sum

println("这个println执行之前才会调用执行sum：" + res)
println("这个println不会调用执行sum：" + res)

def sum(a: Int, b: Int): Int = {
  println("sum执行")
  a+b
}
```