在线实验，请到PC端体验

类层次关系和底层类型

一、实验介绍

1.1 实验内容

前面我们介绍了 Scala 的类的继承，本节我们将介绍 Scala 语言自身定义的类的层次关系和底层类型。

1.2 实验知识点

- Scala 的类层次关系
- 底层类型

1.3 实验环境

- Scala 2.11.7
- Xfce 终端

1.4 适合人群

本课程难度为一般，属于初级级别课程，适合零基础或具有 Java 编程基础的用户。

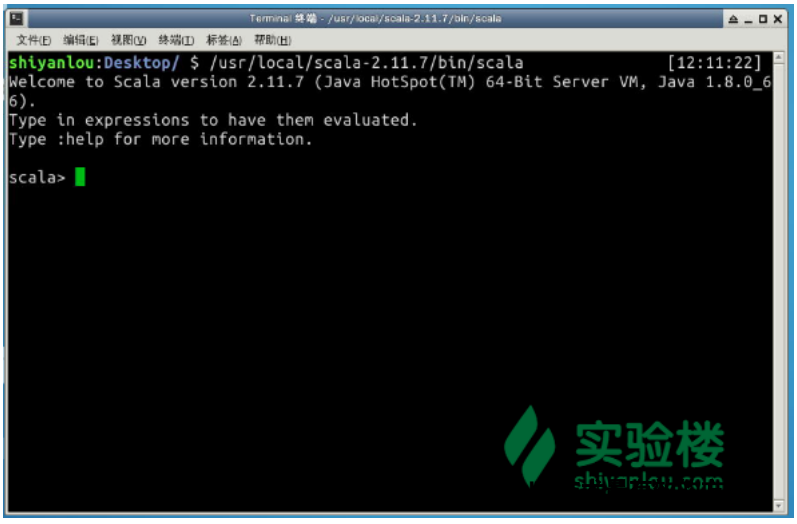
二、开发准备

为了使用交互式 Scala 解释器，你可以在打开的终端中输入命令：

```
cd /usr/local/scala-2.11.7/bin/

scala
```

当出现 `scala>` 开始的命令行提示符时，就说明你已经成功进入解释器了。如下图所示。

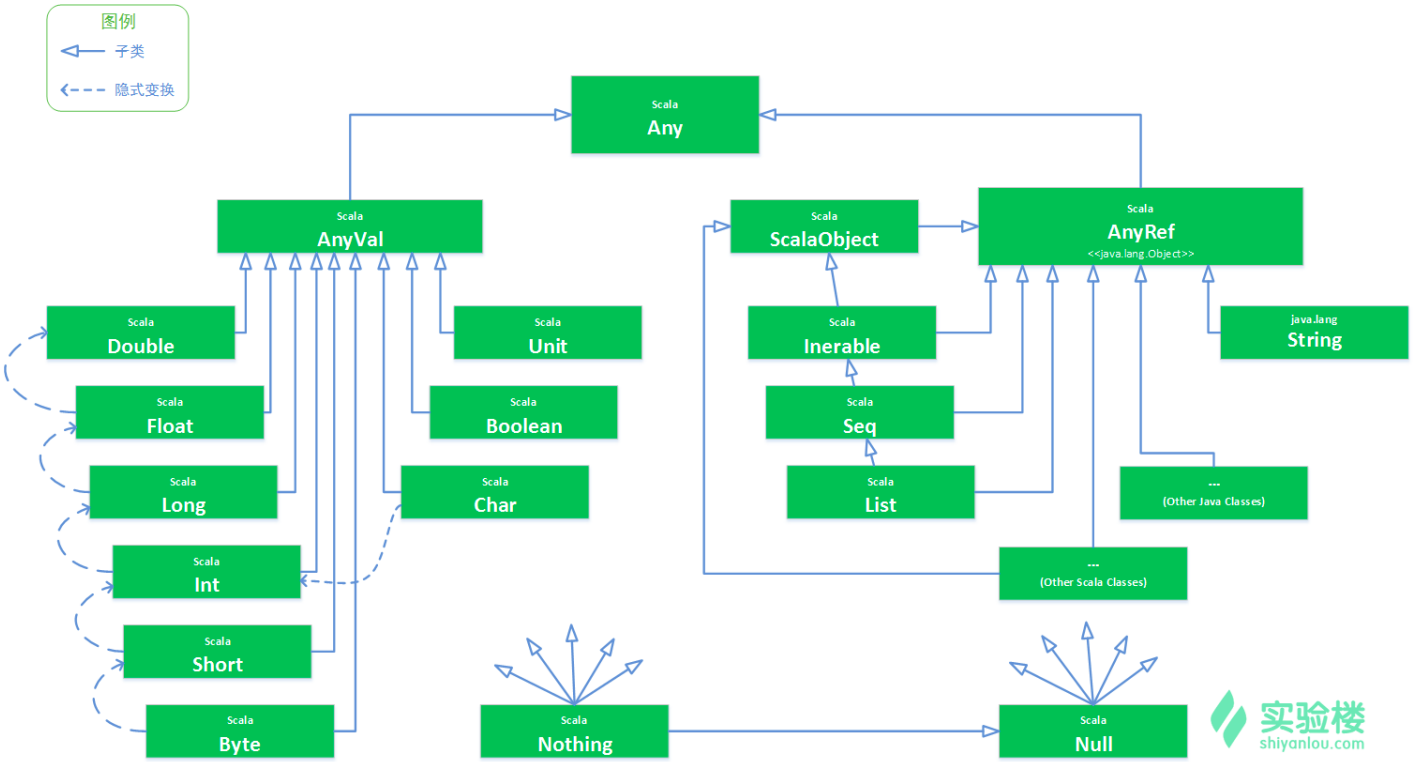


开始实验

三、实验步骤

3.1 Scala 的类层次关系

在 Scala 中，所有的类都有一个公共的基类称为 `Any` ，此外还定义了所有类的子类 Nothing ，下面的图给出的 Scala 定义的类层次关系的一个概要：



由于所有的类都继承自 `Any` ，因此 Scala 中的对象都可以使用 `==` 、 `!=` 或 `equals` 来比较，使用 `##` 或 `hashCode` 给出 `hash` 值，使用 `toString` 转为字符串。 `Any` 的 `==` 和 `!=` 定位为 `fianl` ，因此不可以被子类重载。 `==` 实际上和 `equals` 等价， `!=` 和 `equals` 的否定形式等价，因此重载 `equals` 可以修改 `==` 和 `!=` 的定义。

根类 `Any` 有两个子类： `AnyVal` 和 `AnyRef` 。 `AnyVal` 是 `Scala` 里每个内建值类型的父类。有九个这样的值类型： `Byte` ， `Short` ， `Char` ， `Int` ， `Long` ， `Float` ， `Double` ， `Boolean` 和 `Unit` 。其中的前八个对应到 Java 的基本数值类型，它们的值在运行时表示成 Java 的类型。

Scala 里，这些类的实例都写成字面量。例如， `42` 是 `Int` 的实例，“ `x` ”是 `Char` 的实例， `false` 是 `Boolean` 的实例。值类型都被定义为即是抽象的又是 `final` 的，你不能使用 `new` 创造这些类的实例。

```
scala> new Int
<console>:8: error: class Int is abstract; cannot be instantiated
            new Int
            ^
scala>
```

另一个值类是 `Unit` ，大致对应于 Java 的 `void` 类型；它被用作不返回任何有趣结果的方法的结果类型。 `Unit` 只有一个实例值，被写作 `()` 。

值类支持作为方法的通用的数学和布尔操作符。例如， `Int` 有名为 `+` 和 `*` 的方法， `Boolean` 有名为 `||` 和 `&&` 的方法。值类也从类 `Any` 继承所有的方法。你可以在解释器里测试如下代码：

```
scala> 42 toString
res3: String = 42
scala> 42.hashCode
res6: Int = 42
```

可以看到，Scala 的值类型之间的关系是扁平的。所有的值类都是 `scala.AnyVal` 的子类型，但是它们不是互相的子类。代之以它们不同的值类类型之间可以隐式地互相转换。例如，需要的时候，类 `scala.Int` 的实例可以自动放宽（通过隐式转换）到类 `scala.Long` 的实例。隐式转换还用来为值类型添加更多的功能。例如，类型 `Int` 支持以下所有的操作：

```
scala> 42 max 43
res0: Int = 43

scala> 42 min 43
res1: Int = 42

scala> 1 until 5
res2: scala.collection.immutable.Range = Range(1, 2, 3, 4)

scala> 1 to 5
res3: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5)

scala> 3.abs
res4: Int = 3

scala> (-3).abs
res5: Int = 3
```

这里解释其工作原理：方法 `min`、`max`、`until`、`to` 和 `abs` 都定义在类 `scala.runtime.RichInt` 里，并且有一个从类 Int 到 RichInt 的隐式转换。当你在 `Int` 上调用没有定义在 `Int` 上但定义在 `RichInt` 上的方法时，这个转换就被应用了。

类 `Any` 的另一个子类是类 `AnyRef` 。这个是 Scala 里所有引用类的基类。正如前面提到的，在 Java 平台上 `AnyRef` 实际就是类 `java.lang.Object` 的别名。因此 Java 里写的类和 Scala 里写的都继承自 `AnyRef` 。

如此说来，你可以认为 `java.lang.Object` 是 Java 平台上实现 `AnyRef` 的方式。因此，尽管你可以在 Java 平台上的 Scala 程序里交换使用 `Object` 和 `AnyRef` ，推荐的风格是在任何地方都只使用 `AnyRef` 。

Scala 类与 Java 类不同在于它们还继承自一个名为 `ScalaObject` 的特别的 `Marker Trait` （ `Trait` 我们在后面再进一步解释）

3.2 所有类的公共子类——底层类型

在本实验的上一小节中（Scala 的类层次关系），图的最下方我们可以看到有两个类， `scala.Null` 和 `scala.Nothing` 。这两个类的作用是： Scala 支持统一方式用来处理面向对象的一些边角情况。因为它们在类层次图的下方，因此也称为 底层类型 。

类 `Null` 代表 `null` 引用，它是所有引用类（每个由 `AnyRef` 派生的类）的子类。 `Null` 和值类型不兼容，也就是说，你不能把 `null` 赋值给一个整数类型变量：

```
scala> val i:Int=null
<console>:7: error: an expression of type Null is ineligible for implicit conversion
       val i:Int=null
```

`Nothing` 类型为图中类层次关系的最下方，它是所有其他类的子类。然而，这个类型没有任何实例（也就是没有任何值对应 `Nothing` 类型）。前面提到， `Nothing` 类型的一个用法是示意应用程序非正常终止，比如 `Predef` 的有一个 `error` 方法：

```
def error(message:String) :Nothing =
  throw new RuntimeException(message)
```

`error` 的返回类型就是 `Nothing` ，告诉调用者该方法没有正常退出（抛出异常）。正因为 `Nothing` 是所有其它类型的子类，你可以灵活使用如 `error` 这样的函数。比如：

```
def divide(x:Int,y:Int):Int=
  if(y!=0) x/y
  else error("Cannot divide by Zero")
```

`if` 的“`then`”分支的类型为 `Int(x/y)` ，而 `else` 分支的类型为 `error` 返回值，其类型为 `Nothing` 。因为 `Nothing` 为所有类型的子类，它也是 `Int` 的子类，因此 `divide` 的类型为 `Int` 。

四、实验总结

在本实验中，我们学习了 Scala 的类层次关系和底层类型。你对于这些内容可能有些生疏，但它对于一些科学计算的工作是非常有帮助的。

课程教师

引路蜂

共发布过**6**门课程

CSDN 专家博主，擅长Java ME, Blackberry ,LWUIT , iPhone, Android, Windows Mobile, Mono , Windows Phone 7等平台开发，主页 http://www.imobilebbs.com/

查看老师的所有课程 > (/teacher/164063)

进阶课程

Scala 专题教程 - Case Class和模式匹配 (/courses/514)

Scala 专题教程 - 隐式变换和隐式参数 (/courses/515)

Scala 专题教程 - 抽象成员 (/courses/516)

Scala 专题教程 - Extractor (/courses/526)

动手做实验，轻松学IT

(http://weibo.com/shiyanlou2013)

公司

关于我们 (/aboutus)
联系我们 (/contact)
加入我们 (http://www.simplecloud.cn/jobs.html)
技术博客 (https://blog.shiyanlou.com)

合作

我要投稿 (/contribute)
教师合作 (/labs)
高校合作 (/edu/)
友情链接 (/friends)
开发者 (/developer)

服务

企业版 (/saas)
实战训练营 (/bootcamp/)
会员服务 (/vip)
实验报告 (/courses/reports)
常见问题 (/questions/?tag=%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98)
隐私条款 (/privacy)

学习路径

Python学习路径 (/paths/python)
Linux学习路径 (/paths/linuxdev)
大数据学习路径 (/paths/bigdata)
Java学习路径 (/paths/java)
PHP学习路径 (/paths/php)
全部 (/paths/)

动手实践是学习 IT 技术最有效的方式！ **开始实验**