尚硅谷大数据技术之 Scala
面向对象编程

官网：www.atguigu.com

ShangGuigu Technologies Co., Ltd.

尚硅谷技术有限公司

一　面向对象编程

"面向对象编程"也是一种"编程范式"（programming paradigm）

结构化编程的思路是从功能结构的角度求解问题，先做什么，后做什么，有一个结构在里面，而面向对象编程是将问题分解成小的问题对象，通过对象之间的通信以及内在的属性变化来求解问题。不同的编程思路有其各自的优缺点。

面向对象编程中，对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件。

具有相同特性（数据元素）和行为（功能）的对象的抽象就是类。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象，类实际上就是一种数据类型。

```
// 声明类
public class User {
    public String username;
}
// 构建对象
User user = new User();
```

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，所以 Java 又不是完全面向对象的语言。Scala 语言来自于 Java，所以天生就是面向对象的语言，在面向对象的学习过程中可以对比着 Java 语言学习

二　基础语法

2.1　类

```
// Java
// [修饰符] class 类名 {类体}
// java 语法中要求一个 java 源码文件中可以声明多个类，但是公共类只能有一个，且必须和源码文件的文件名保持一致。
```

```
// Scala
// [修饰符] class 类名 {类体}
// scala 语法中，类并不声明为 public，一个 Scala 源文件可以包含多个类。所有这些类都具有公有可见性。所以这里的修饰符在后面扩展时再介绍
```

2.2　对象

```
// Java
// 类型 变量 = new 类型();
// User user = new User();
// Java 是强类型语言，声明任何变量的同时都必须声明类型
```

```
// Scala
// val 变量 [：类型] = new 类型();
// val user : User = new User();
// 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
```

2.3　属性

```
// Java(Bean)
// private 类型 属性名称;
// public Setter/Getter 方法
// Java 语法中如果声明了一个属性，但是没有显示的初始化，那么会给这个属性默认初始化
// 通过 setter/getter 方法对属性进行赋值和取值
```

```
// setter/getter 方法一般都是公共访问权限，起到了封装的作用
```

```
// Scala(Bean)
// private var 属性名称 [：类型] = 属性值
// def Setter/Getter 方法
// 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，如果初始化的值设定为 null，也可以使用符号_(下划线)代替
// Scala 为了访问一致性，所以并不推荐由开发人员自行定义 setter/getter 方法
// Scala 中为了简化代码的开发，当声明属性时，本身就自动提供了对应的 setter/getter 方法
//  如果属性声明为 private 的，那么自动生成的 setter/getter 方法也是 private 的
//  如果属性省略访问权限修饰符，那么自动生成的 setter/getter 方法是 public 的
```

2.4 方法

```
// Java
// public 返回值类型 方法名(参数列表) { 方法体 }
//调用方法
// 对象.方法名()
```

```
// Scala
// def 方法名(参数列表) [：返回值类型] = {方法体}
// Scala 中的方法其实就是函数，只不过一般将对象中的函数称之为方法。声明规则请参考函数式编程中的函数声明
// 调用方法
// 对象.方法名()
```

2.5 静态方法

```
// Java
// public static 返回值类型  方法名(参数列表) {方法体}
// Java 中静态方法并不是通过对象调用的，而是通过类对象调用的，所以静态操作并不是面向对象的。
```

```
// Scala
// Scala 语言是完全面向对象(万物皆对象)的语言，所以并没有静态的操作。但是为了能够和 Java 语言交互，就产生了一种特殊的对象来模拟类对象，我们称之为类的伴生对象。这个类的所有静态内容都可以放置在它的伴生对象中声明和调用
// Scala 中伴生对象采用 object 关键字声明，伴生对象中声明的全是“静态”内容，可以通过伴生对象名称直接调用。
// 伴生对象对应的类称之为伴生类，伴生类和伴生对象应该在同一个源码文件中
// 从语法角度来讲，所谓的伴生对象其实就是类的静态方法和成员的集合
// 从技术角度来讲，所谓的伴生对象在编译时，会将对应的代码以静态的方式生成到类的字节码中。
```

2.6 包

```
// Java
// package 包名;
// Java 中的包名主要用于区分不同含义但名称相同的类，通过包名进行管理，使用时，只要增加全包名就不会冲突
// java 中包名和源码所在的系统文件结构息息相关，并且编译后的字节码文件路径也和包名保持一致
```

```
// Scala
// package 包名
// scala 语言也可以使用包管理类
// scala 包名和源码实际的存储位置没有关系
// 从技术角度来讲，Scala 的编译器会将 Scala 中的包编译成符合 Java 语法规则的包结构
```

2.7 继承

和 Java 一样，Scala 也支持类的单继承

```
// Java
// class 子类名 extends 父类名 { 类体 }
```

// 子类继承父类的属性和方法

// Scala
// class 子类名 extends 父类名 { 类体 }
//子类继承父类的属性和方法

2.8 接口

// Java
// 声明接口
// interface 接口名
// 实现接口
// class 实现类类名 implements 接口名
// 在 Java 中，接口并不属于类的体系关系，所以可以多实现，且接口支持多继承
// 接口中属性都是常量
// 接口中的方法都是抽象的。

// Scala
// Scala 语言中，采用特质（特征）来代替接口的概念，也就是说，多个类具有相同的特征（特征）时，就可以将这个特质（特征）独立出来，采用关键字 trait 声明
// 声明特质
// trait 特质名
// 一个类具有某种特质（特征），就意味着这个类满足了这个特质（特征）的所有要素，所以在使用时，也采用了 extends 关键字，如果有多个特质或存在父类，那么需要采用 with 关键字连接
// class 类名 extends 特质 1 名称 with 特质 2 名称 with 特质 3 名称