Scala第六章节

章节目标

1. 掌握类和对象的定义
2. 掌握访问修饰符和构造器的用法
3. 掌握main方法的实现形式
4. 掌握伴生对象的使用
5. 掌握定义工具类的案例

---

1. 类和对象

Scala是一种函数式的面向对象语言, 它也是支持面向对象编程思想的，也有类和对象的概念。我们依然可以基于Scala语言来开发面向对象的应用程序。

1.1 相关概念

什么是面向对象?

```
面向对象是一种编程思想，它是基于面向过程的，强调的是以对象为基础完成各种操作.
```

面向对象的三大思想特点是什么?

```
1. 更符合人们的思考习惯.
2. 把复杂的事情简单化.
3. 把程序员从执行者变成指挥者.
```

> 面试题: 什么是面向对象? 思路: 概述, 特点, 举例, 总结.

什么是类?

```
类是属性和行为的集合，是一个抽象的概念，看不见，也摸不着.
```

> - 属性(也叫成员变量): 名词, 用来描述事物的外在特征的.
> - 行为(也叫成员方法): 动词, 表示事物能够做什么.
> - 例如: 学生有姓名和年龄(这些是属性), 学生要学习, 要吃饭(这些是行为).

什么是对象?

```
对象是类的具体体现，实现.
```

面向对象的三大特征是什么?

```
封装，继承，多态.
```

1.2 创建类和对象

Scala中创建类和对象可以通过class和new关键字来实现. **用class来创建类, 用new来创建对象.**

1.2.1 示例

创建一个Person类，并创建它的对象, 然后将对象打印到控制台上.

1.2.2 步骤

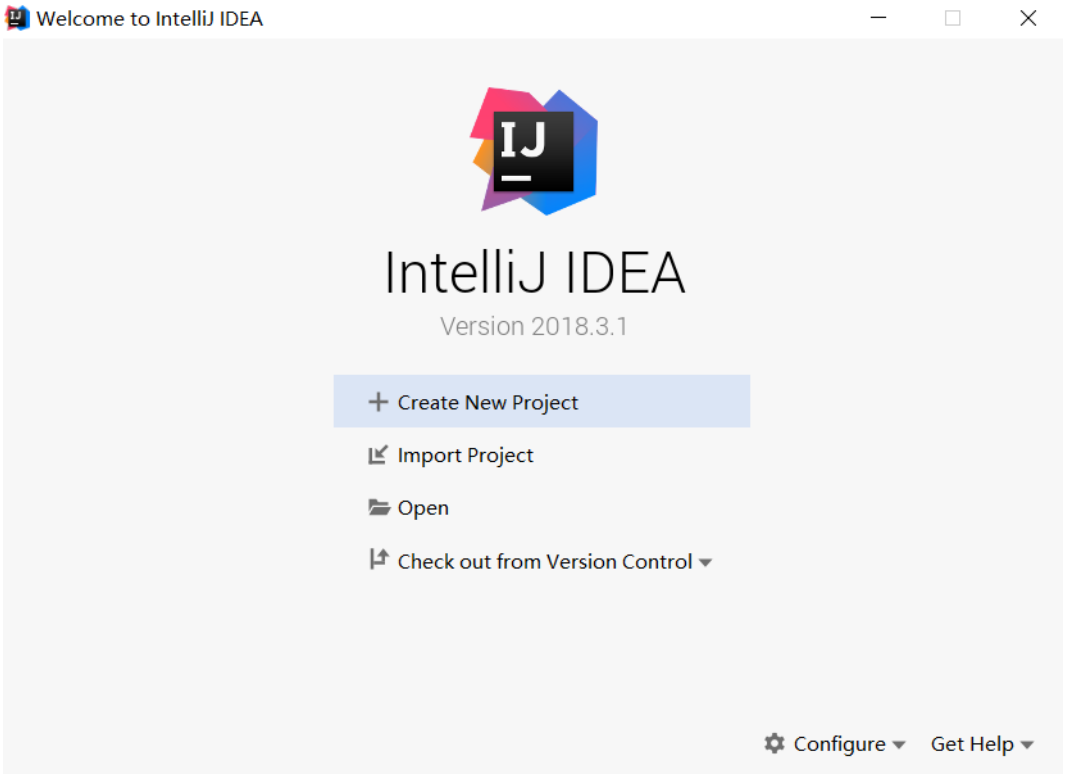
1. 创建一个scala项目，并创建一个object类

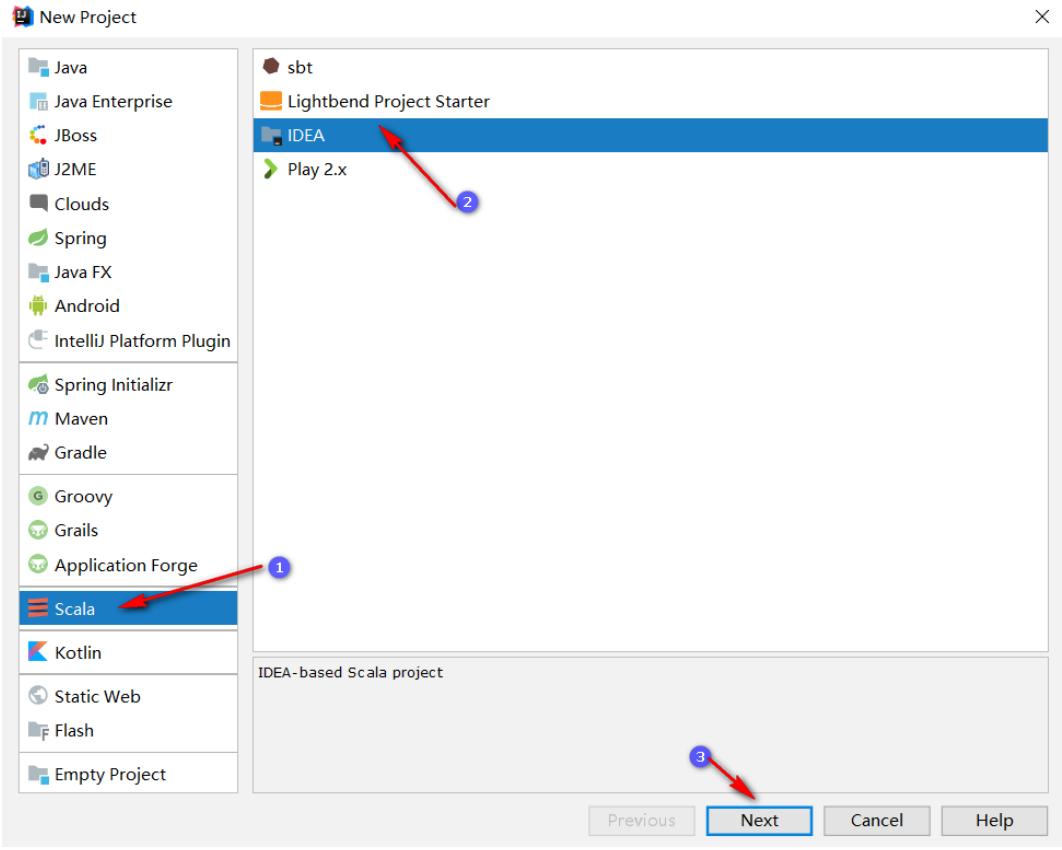
   > 注意: object修饰的类是单例对象，这点先了解即可, 稍后会详细解释.

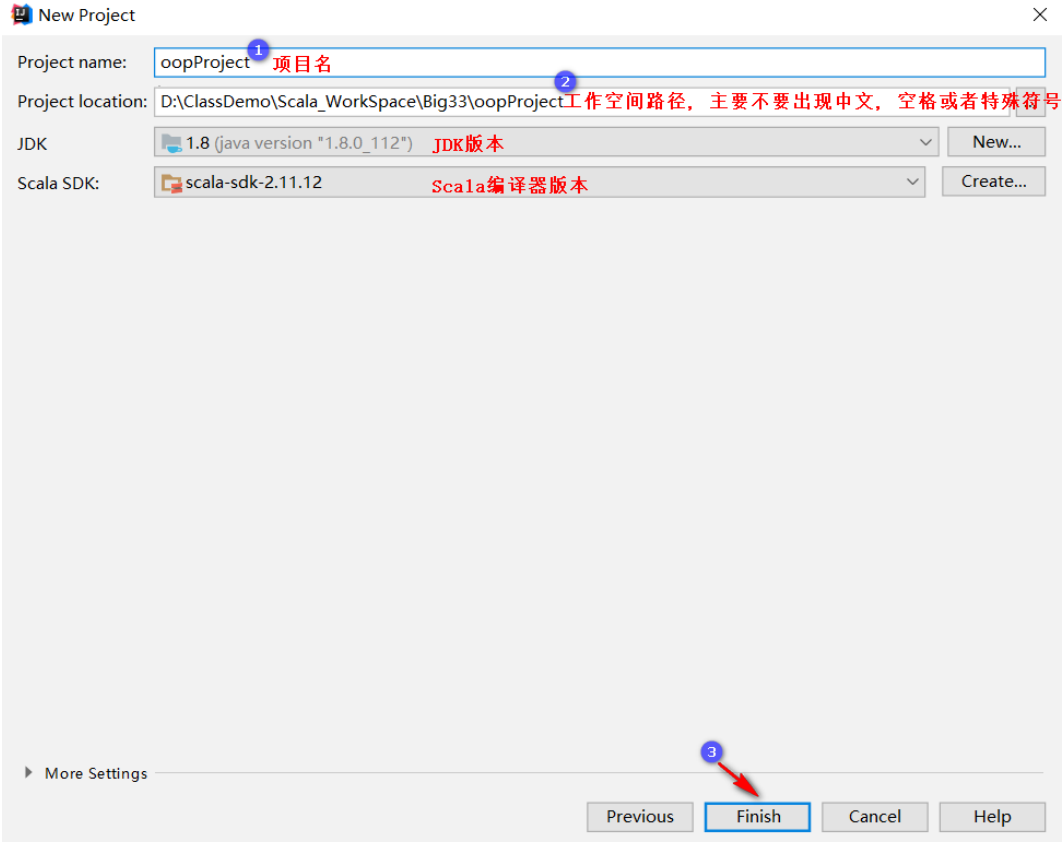
2. 在object类中添加main方法.
3. 创建Person类, 并在main方法中创建Person类的对象, 然后输出结果.

1.2.3 实现

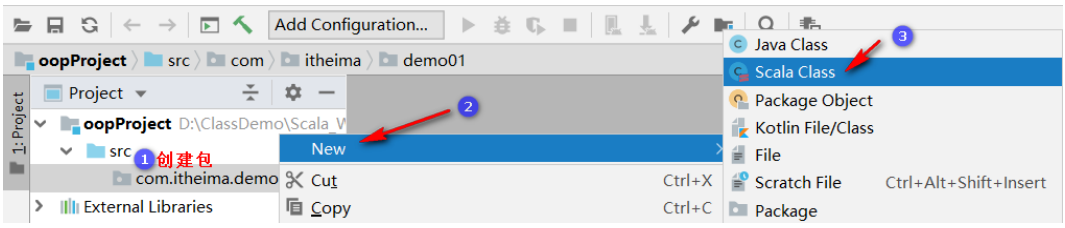
- 在IDEA中创建项目，并创建一个object类（main方法必须放在Object中）
  - 第一步: 创建Scala项目

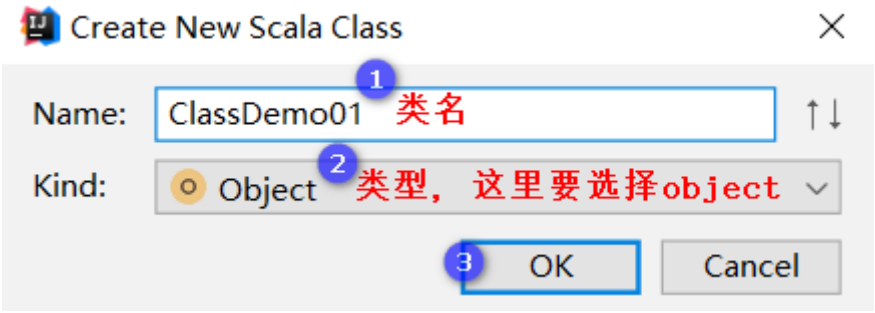






- 第二步: 创建object类.





- 在object类中添加main方法, 并按照需求完成指定代码.

```
//案例: 测试如何在Scala程序中创建类和对象.
object ClassDemo01 {

  //1. 创建Person类
  class Person {}

  //2. 定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 创建Person类型的对象.
    val p = new Person()
    //4. 将对象打印到控制台上
    println(p)
  }
}
```

1.3 简写方式

用法

- 如果类是空的，没有任何成员，可以省略 `{}`
- 如果构造器的参数为空，可以省略 `()`

示例

使用简写方式重新创建Person类和对象, 并打印对象.

参考代码

```
//案例: 通过简写方式在Scala中创建类和对象
object ClassDemo02 {

  //1. 创建Person类
  class Person

  //2. 定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 创建Person类型的对象.
    val p = new Person
    //4. 将对象打印到控制台上
    println(p)
  }
}
```

2. 定义和访问成员变量

一个类会有自己的属性，例如：人类，就有自己的姓名和年龄。我们接下来学习如何在类中定义和访问成员变量。

2.1 用法

- 在类中使用`var/val`来定义成员变量
- 对象可以通过`对象名.`的方式来访问成员变量

2.2 示例

需求

1. 定义一个Person类，包含一个姓名和年龄字段
2. 创建一个名为"张三"、年龄为23岁的对象
3. 打印对象的名字和年龄

步骤

1. 创建一个object类，添加main方法
2. 创建Person类，添加姓名字段和年龄字段，并对字段进行初始化，让scala自动进行类型推断
3. 在main方法中创建Person类对象，设置成员变量为"张三"、23
4. 打印对象的名字和年龄

参考代码

```
//案例: 定义和访问成员变量.
object ClassDemo03 {

  //1. 创建Person类
  class Person {
    //2. 定义成员变量
    //val name:String = ""
    //通过类型推断来实现
    //var修饰的变量, 值是可以修改的.  val修饰的变量, 值不能修改.
    var name = ""    //姓名
    var age = 0      //年龄
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 创建Person类型的对象.
    val p = new Person
    //4. 给成员变量赋值
    p.name = "张三"
    p.age = 23
    //5. 打印成员变量值到控制台上
    println(p.name, p.age)
  }
}
```

3. 使用下划线初始化成员变量

scala中有一个更简洁的初始化成员变量的方式，可以让代码看起来更加简洁, 更优雅.

3.1 用法

- 在定义`var`类型的成员变量时，可以使用`_`来初始化成员变量
  - String => null
  - Int => 0
  - Boolean => false
  - Double => 0.0
  - ...
- `val`类型的成员变量，必须要自己手动初始化

3.2 示例

需求

1. 定义一个Person类，包含一个姓名和年龄字段
2. 创建一个名为"张三"、年龄为23岁的对象
3. 打印对象的名字和年龄

步骤

1. 创建一个object类，添加main方法
2. 创建Person类，添加姓名字段和年龄字段，指定数据类型，使用下划线初始化
3. 在main方法中创建Person类对象，设置成员变量为"张三"、23
4. 打印对象的名字和年龄

参考代码

```
//案例: 使用下划线来初始化成员变量值.
//注意: 该方式只针对于var类型的变量有效. 如果是val类型的变量, 需要手动赋值.
object ClassDemo04 {

  //1. 创建Person类
  class Person {
    //2. 定义成员变量
    var name:String = _   //姓名
    var age:Int = _       //年龄
    //val age:Int = _     //这样写会报错, 因为"下划线赋值"的方式只针对于var修饰的变量有效.
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 创建Person类型的对象.
    val p = new Person
    //4. 给成员变量赋值
    p.name = "张三"
    p.age = 23
    //5. 打印成员变量值到控制台上
    println(p.name, p.age)
  }
}
```

4. 定义和访问成员方法

类可以有自己的行为，scala中也可以通过定义成员方法来定义类的行为。

4.1 格式

在scala的类中，也是使用`def`来定义成员方法的.

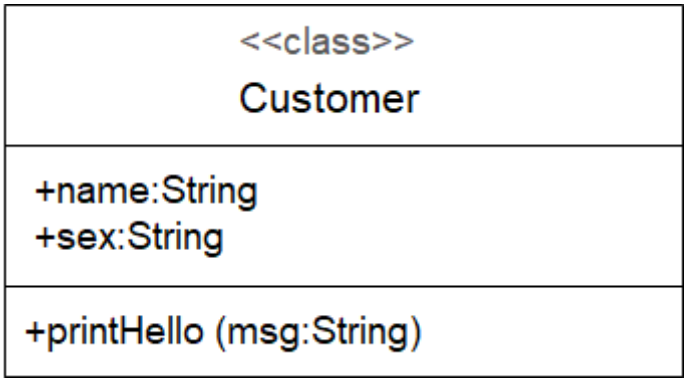
```
def 方法名(参数1:数据类型, 参数2:数据类型): [return type] = {
    //方法体
}
```

> 注意: 返回值的类型可以不写, 由Scala自动进行类型推断.

4.2 示例

需求

1. 创建一个Customer类



2. 创建一个该类的对象，并调用printHello方法

步骤

1. 创建一个object类，添加main方法
2. 创建Customer类，添加成员变量、成员方法
3. 在main方法中创建Customer类对象，设置成员变量值(张三、男)
4. 调用成员方法

参考代码

```
//案例: 定义和访问成员方法
object ClassDemo05 {

  //1. 创建Customer类
  class Customer {
    //2. 定义成员变量
    var name:String = _   //姓名
    var sex = ""          //性别

    //3. 定义成员方法printHello
    def printHello(msg:String) = {
      println(msg)
    }
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //4. 创建Customer类型的对象.
    val c = new Customer
    //5. 给成员变量赋值
```

```
    c.name = "张三"
    c.sex = "男"
    //6. 打印成员变量值到控制台上
    println(c.name, c.sex)
    //7. 调用成员方法
    printHello("你好!")
  }
}
```

5. 访问权限修饰符

和Java一样，scala也可以通过访问修饰符，来控制成员变量和成员方法是否可以被外界访问。

5.1 定义

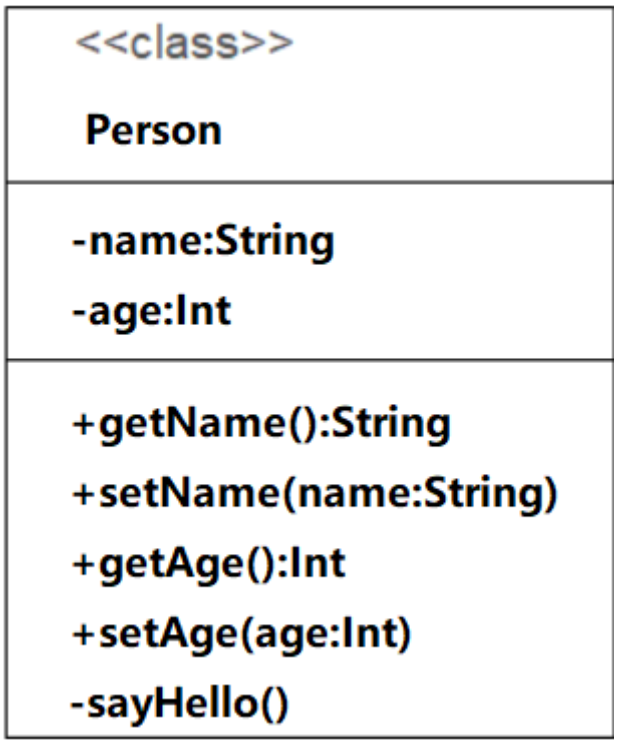
1. Java中的访问控制，同样适用于scala，可以在成员前面添加private/protected关键字来控制成员的可见性。
2. 在scala中，`没有public关键字`，任何没有被标为private或protected的成员都是公共的.

> 注意:
> Scala中的权限修饰符只有: private, private[this], protected, 默认这四种.

5.2 案例

需求

- 定义一个Person类



- 在main方法中创建该类的对象，测试是否能够访问到私有成员

参考代码

```
//案例: 演示Scala中的访问修饰符.
/*
  注意:
```

```
    1. Scala中可以使用private/protected来修饰成员.
    2. 如果成员没有被private/protected修饰, 默认就是公共的(类似于Java中的public).
 */
object ClassDemo06 {

  //1. 创建Customer类
  class Customer {
    //2. 定义成员变量, 全部私有化
    private var name = "" //姓名
    private var age = 0 //年龄

    //3. 定义成员方法
    //获取姓名
    def getName() = name

    //设置姓名
    def setName(name: String) = this.name = name

    //获取年龄
    def getAge() = age

    //设置年龄
    def setAge(age: Int) = this.age = age

    //打招呼的方法, 该方法需要私有化.
    private def sayHello() = println("Hello, Scala!")
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //4. 创建Customer类型的对象.
    val c = new Customer
    //5. 给成员变量赋值
    //c.name = "张三"      //这样写会报错, 因为私有成员外界无法直接访问.
    c.setName("张三")
    c.setAge(23)
    //6. 打印成员变量值到控制台上
    println(c.getName(), c.getAge())
    //7. 尝试调用私有成员方法
    //c.sayHello()        //这样写会报错, 因为私有成员外界无法直接访问.
  }
}
```

6. 类的构造器

当创建对象的时候，会自动调用类的构造器。之前使用的都是默认构造器，接下来我们要学习如何自定义构造器。

6.1 分类

- 主构造器
- 辅助构造器

6.2 主构造器

语法

```
class 类名(var/val 参数名:类型 = 默认值, var/val 参数名:类型 = 默认值){
    //构造代码块
}
```

> 注意:
>
> - 主构造器的参数列表直接定义在类名后面，添加了val/var表示直接通过主构造器定义成员变量
> - 构造器参数列表可以指定默认值
> - 创建实例，调用构造器可以指定字段进行初始化
> - 整个class中除了字段定义和方法定义的代码都是构造代码

示例

1. 定义一个Person类，通过主构造器参数列表定义姓名和年龄字段，并且设置它们的默认值为张三,23
2. 在主构造器中输出"调用主构造器"
3. 创建"李四"对象（姓名为李四，年龄为24），打印对象的姓名和年龄
4. 创建"空"对象，不给构造器传入任何的参数，打印对象的姓名和年龄
5. 创建"测试"对象，不传入姓名参数，仅指定年龄为30，打印对象的姓名和年龄

参考代码

```
//案例: 演示Scala中的类的主构造器.
/*
  注意:
    1. Scala中主构造器的参数列表是直接写在类名后的.
    2. 主构造器的参数列表可以有默认值.
    3. 调用主构造器创建对象时, 可以指定参数赋值.
    4. 整个类中除了定义成员变量和成员方法的代码, 其他都叫: 构造代码.
 */
object ClassDemo07 {

  //1. 创建person类, 主构造器参数列表为: 姓名和年龄.
  class Person(val name: String = "张三", val age: Int = 23) { //这里应该用var修饰.
    //2. 在主构造器中输出"调用主构造器"
    println("调用主构造器!...")
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 创建"空"对象, 什么都不传.
    val p1 = new Person()
    println(s"p1: ${p1.name}...${p1.age}")

    //4. 创建"李四"对象, 传入姓名和年龄
    val p2 = new Person("李四", 24)
    println(s"p2: ${p2.name}...${p2.age}")

    //5. 创建测试对象, 仅传入年龄.
    val p3 = new Person(age = 30)
    println(s"p3: ${p3.name}...${p3.age}")
  }
}
```

6.2 辅助构造器

在scala中，除了定义主构造器外，还可以根据需要来定义辅助构造器。例如：允许通过多种方式，来创建对象，这时候就可以定义其他更多的构造器。我们把除了主构造器之外的构造器称为**辅助构造器。**

语法

- 定义辅助构造器与定义方法一样，也使用`def`关键字来定义
- 辅助构造器的默认名字都是`this`, 且不能修改.

```
def this(参数名:类型, 参数名:类型) {
    // 第一行需要调用主构造器或者其他构造器
    // 构造器代码
}
```

> 注意: 辅助构造器的第一行代码，必须要**调用主构造器或者其他辅助构造器**

示例

需求

- 定义一个Customer类，包含一个姓名和地址字段
- 定义Customer类的主构造器（初始化姓名和地址）
- 定义Customer类的辅助构造器，该辅助构造器接收一个数组参数，使用数组参数来初始化成员变量
- 使用Customer类的辅助构造器来创建一个"张三"对象
    - 姓名为张三
    - 地址为北京
- 打印对象的姓名、地址

> 注意:
>
> 1. 该案例涉及到"数组"相关的知识点, 目前我们还没有学习到.
> 2. Scala中的数组和Java中的数组用法基本类似, 目前能看懂即可, 后续会详细讲解.

参考代码

```
//案例: 演示Scala中的类的辅助构造器.
object ClassDemo08 {

  //1. 创建Customer类, 主构造器参数列表为: 姓名和地址.
  class Customer(var name: String, var address: String) { //这里应该用var修饰.
    //2. 定义一个辅助构造器, 接收一个数组参数
    def this(arr:Array[String]) {
      this(arr(0), arr(1))    //将数组的前两个元素分别传给主构造器的两个参数.
    }
  }

  //定义main函数, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 调用辅助构造器, 创建Customer对象.
    val c = new Customer(Array("张三", "北京"))
    //4. 打印结果.
    println(c.name, c.address)
  }
}
```

7. 单例对象

scala中是没有static关键字的，要想定义类似于Java中的static变量、static方法，就要使用到scala中的单例对象了，也就是object.

7.1 定义单例对象

单例对象表示全局仅有一个对象, 也叫`孤立对象`. 定义单例对象和定义类很像，就是把class换成object.

格式

```
object 单例对象名{ }          //定义一个单例对象.
```

> 注意:
>
> 1. 在object中定义的成员变量类似于Java中的静态变量, 在内存中都只有一个对象.
> 2. 在单例对象中, 可以直接使用`单例对象名.`的形式调用成员.

示例

需求

- 定义一个Dog单例对象，保存狗有几条腿
- 在main方法中打印狗腿的数量

参考代码

```
//案例: 演示Scala中的单例对象之定义和访问成员变量.
object ClassDemo09 {

  //1. 定义单例对象Dog
  object Dog {
    //2. 定义一个变量, 用来存储狗腿子的数量
    val leg_num = 4
  }

  //定义main方法, 它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 打印狗腿子的数量
    println(Dog.leg_num)
  }
}
```

7.2 在单例对象中定义方法

在单例对象中定义的成员方法类似于Java中的静态方法.

示例

需求

- 设计一个单例对象，定义一个能够打印分割线（15个减号）的方法
- 在main方法调用该方法，打印分割线

参考代码

```
//案例: 演示Scala中的单例对象之定义和访问成员方法.
```

```
object ClassDemo10 {

  //1. 定义单例对象PrintUtil
  object PrintUtil {
    //2. 定义一个方法，用来打印分割线
    def printSpliter() = println("-" * 15)
  }

  //定义main方法，它是程序的主入口.
  def main(args: Array[String]): Unit = {
    //3. 调用单例对象中的成员方法
    PrintUtil.printSpliter()
  }
}
```

8. main方法

scala和Java一样，如果要运行一个程序，必须有一个main方法。在Java中main方法是静态的，而在scala中没有静态方法。所以在scala中，这个main方法必须放在一个单例对象中。

8.1 定义main方法

main方法

```
def main(args:Array[String]):Unit = {
    // 方法体
}
```

示例

需求

- 创建一个单例对象，在该单例对象中打印"hello, scala"

参考代码

```
object Main5 {
  def main(args:Array[String]) = {
    println("hello, scala")
  }
}
```

8.2 继承App特质

创建一个object, 继承自App特质(Trait)，然后将需要编写在main方法中的代码，写在object的构造方法体内。

```
object 单例对象名 extends App {
    // 方法体
}
```

示例

需求

- 继承App特质，来实现一个入口。同样输出"hello, scala"

参考代码

```
object Main5 extends App {
  println("hello, scala")
}
```

9. 伴生对象

在Java中，经常会有一些类，同时有实例成员又有静态成员。例如：

```
public class Generals {

    private static String armsName = "青龙偃月刀";

    public void toWar() {
        //打仗
        System.out.println("武将拿着"+ armsName +", 上阵杀敌!");
    }

    public static void main(String[] args) {
        new Generals().toWar();
    }
}
```

在scala中，要实现类似的效果，可以使用**伴生对象**来实现。

9.1 定义伴生对象

一个class和object具有同样的名字。这个object称为**伴生对象**，这个class称为**伴生类**

- 伴生对象必须要和伴生类一样的名字
- 伴生对象和伴生类在同一个scala源文件中
- 伴生对象和伴生类可以互相访问private属性

示例

需求

- 编写一个Generals类，有一个toWar方法，打印

  ```
  武将拿着**武器，上阵杀敌!      //注意：**表示武器的名字.
  ```

- 编写一个Generals伴生对象，定义一个私有变量，用于保存武器名称.
- 创建Generals对象，调用toWar方法

参考代码

```
//案例：演示Scala中的伴生对象
object ClassDemo12 {

  //1. 定义一个类Generals，作为一个伴生类.
  class Generals {    //这里写的都是非静态成员.
    //2. 定义一个toWar()方法，输出一句话，格式为"武将拿着**武器，上阵杀敌!"
```

```
    def toWar() = println(s"武将拿着${Generals.armsName}武器, 上阵杀敌!")
  }

  //3. 定义一个伴生对象, 用来保存"武将的武器".
  object Generals {   //这里写的都是静态成员.
    private var armsName = "青龙偃月刀"
  }

  //定义main方法, 作为程序的主入口
  def main(args: Array[String]): Unit = {
    //4. 创建Generals类的对象.
    val g = new Generals
    //5. 调用Generals类中的toWar方法
    g.toWar()
  }
}
```

9.2 private[this]访问权限

如果某个成员的权限设置为private[this]，表示只能在当前类中访问。伴生对象也不可以访问.

示例

示例说明

- 定义一个Person类，包含一个name字段, 该字段用private[this]修饰
- 定义Person类的伴生对象，定义printPerson方法
- 测试伴生对象是否能访问private[this]权限的成员

示例代码

```
//案例: 测试private[this]的访问权限
object ClassDemo13 {

  //1. 定义一个Person类, 属性为: name
  class Person(private[this] var name: String)

  //2. 定义Person类的伴生对象.
  object Person {
    //3. 定义一个方法printPerson, 用来打印Person#name属性值.
    def printPerson(p:Person) = println(p.name)
  }

  //定义main函数, 它是程序的主入口
  def main(args: Array[String]) = {
    //4. 创建Person类型的对象.
    val p = new Person("张三")
    //5. 调用Person伴生对象中的printPerson方法
    Person.printPerson(p)
  }
}
```

> 注意: 上述代码，会编译报错。但移除掉[this]就可以访问了

9.3 apply方法

在Scala中, 支持创建对象的时候, `免new`的动作, 这种写法非常简便，优雅。要想实现`免new`, 我们就要通过伴生对象的apply方法来实现。

9.3.1 格式

定义apply方法的格式

```
object 伴生对象名 {
    def apply(参数名:参数类型, 参数名:参数类型...) = new 类(...)
}
```

创建对象

```
val 对象名 = 伴生对象名(参数1, 参数2...)
```

> 例如: val p = Person("张三", 23)

9.3.2 示例

需求

- 定义一个Person类，它包含两个字段：姓名和年龄
- 在伴生对象中定义apply方法，实现创建Person对象的免new操作.
- 在main方法中创建该类的对象，并打印姓名和年龄

参考代码

```
//案例: 演示Scala中的apply方法
object ClassDemo14 {

  //1. 定义Person类, 属性为姓名和年龄
  class Person(var name: String = "", var age: Int = 0)

  //2. 定义Person类的伴生对象.
  object Person {
    //3. 定义apply方法, 实现创建Person对象的时候免new.
    def apply(name:String, age:Int) = new Person(name, age)
  }

  //定义main方法, 作为程序的主入口
  def main(args: Array[String]): Unit = {
    //4. 创建Person类型的对象.
    val p = Person("张三", 23)
    //5. 打印Person对象的属性值.
    println(p.name, p.age)
  }
}
```

10. 案例: 定义工具类

10.1 概述

Scala中工具类的概念和Java中是一样的, 都是

1. 构造方法全部私有化, 目的是不让外界通过构造方法来创建工具类的对象.
2. 成员全部是静态化, 意味着外界可以通过"类名."的形式来访问工具类中的内容.

> 综上所述, 在Scala中只有`object单例对象`满足上述的要求.

10.2 示例

需求

- 编写一个DateUtils工具类专门用来格式化日期时间
- 定义一个方法，用于将日期（Date）转换为年月日字符串，例如：2030-10-05
- 定义一个方法，用于将年月日字符串转换为日期(Date).

步骤

- 定义一个DateUtils单例对象
- 在DateUtils中定义日期格式化方法（date2String）和解析字符串方法(string2Date)
- 使用SimpleDateFormat来实现String和Date之间的相互转换

参考代码

```
//案例: 定义DateUtils工具类, 用于实现String和Date之间的相互转换.
object ClassDemo15 {

  //1. 定义DateUtils工具类.    //也就是Scala中的单例对象.
  object DateUtils {
    //2. 创建SimpleDateFormat类型的对象, 用来进行转换操作.
    var sdf: SimpleDateFormat = null

    //3. 定义方法date2String, 用来将Date日期对象转换成String类型的日期.
    //参数1: 日期对象,   参数2: 模板
    def date2String(date: Date, template: String):String  = {
      sdf = new SimpleDateFormat(template)
      sdf.format(date)
    }

    //4. 定义方法string2Date, 用于将String类型的日期转换成Date日期对象.
    def string2Date(dateString: String, template: String) = {
      sdf = new SimpleDateFormat(template)
      sdf.parse(dateString)
    }
  }

  //定义main方法, 作为程序的主入口.
  def main(args: Array[String]): Unit = {
    //5. 调用DateUtils#date2String()方法, 用来格式化日期.
    val s = DateUtils.date2String(new Date(), "yyyy-MM-dd")
    println("格式化日期: " + s)

    //6. 调用DateUtils#string2Date()方法, 用来解析日期字符串.
    val d = DateUtils.string2Date("1314年5月21日", "yyyy年MM月dd日")
    println("解析字符串: " + d)
  }
}
```