# 语法

|  |
| --- |
| scala 中所有的对象的超类是Any。Any又分为AnyVal和AnyRef。  AnyVal 相当于java中基础数据类型。  AnyRef 相当于对象引用。可以引用scala的对象，也可以引用java的对象  var 变量 val 常量  val name : String =”lisi”;  val name = “lisi”;  println（）  println(f”name= $name”); f允许使用插值器。通过$+变量直接在字符串中取值。  Println(s”1 + 1= { 1 + 1}”); s与f类似。允许使用插值器。还可以使用 表达式  Println(s” name = ${stu.name}”) $+表达式 获取对象的属性  大括号的最后一行 为返回值  条件表达式  val s = if ( i > 10) {  i  }else {  0  }  循环  类似java增强for循环  for( item <- array){  println(item)  }  类似java普通循环  for(index <- 0 to array.length -1 ){  println(array(index))  }  to可换成until，表示左闭右开区间  数组取数，通过（）+下标来取 数据的元素  yield 关键字  将 循环中的每次的返回 形成一个新的数组。  如  for（item <- 1 to 10）yeild item+1  返回 2-11的数组  操作符重载成方法  如： + - \* / to until map  var a=1+2 可写成var a=1.+2  1 to 10 可写成 1.to(10)  array map (\_.1)  定义方法  1、def 方法名 （传参）：返回类型=｛方法体｝  2、def sayHello = println(“hello”) 可以在sayHello在定义时没有加（），调用时也不能加（）  方法转函数: 方法名 \_  如：add \_  定义函数  1、（传参名：类型 ...） => 方法体  （a : int, b : int ）=> a+b  如：val add =（a : int, b : int ）=> a+b  2、（传参类型） => 返回类型 =>(传参名) => 方法体  如： val add : (Int, Int) => Int =(x,y) => x+y  可变参数  Def (a :int , b Int \*):Int ={  for(item :<- b){  }  }  参数默认值，调用时，若不传参数则使用默认值  def add (a:Int = 6,b:Int=6) ={}  或者指定传名  add(b=9,a=9);  add(b=9);  高阶函数：  将其它函数作为参数 或 函数的返回值是一个函数  部分参数应用函数：  如  def add(a:Int,b:Int) = a+b  var add1 =(x:Int) =>add(1,x)  此时，add1也是一个函数。调用 add1(4) 就等 调用 aad(1+4)  函数柯里化  将一个函数的多个参数 拆开。如：  def add(a:Int ,b:Int)=a+b 柯里化 def add(a:Int)(b:Int)=a+b ,调用时add(1)(2)  以上柯里化函数可以 写成 def add(a:Int)=(b:Int) =>a+b  调用时，var add1=add(1) 返回一个函数，再调用add1(3) 返回4.  偏函数，是一个类 PartialFunction[String,Int], String，int是可以是其它类型，String是传入的类型，Int是返回的类型  def f1: PartialFunction[] ={ case “a” => 97 case \_ => 0}  使用 f1(“a”) 返回97 |

数组

|  |
| --- |
| 定义  var x:Array[String] =new Array[String](3)  或 var y = new Array[String](3)  Val z= Array[Int](1,2,3)  修改元素内容  Z(0)= 100  添加内容  z += (200) 报错  数组长度不可变，内容可变  数组的map方法 映射：即对每个元素进行操作 ，返回新的数组 flatten将数组里面的多个数组元素转化一个数组  flatmap  foreach 遍历  groupBy 分组，返回一个map ，map[String,Array]如：array.groupBy( x => x)  第一个x表示每个元素。第二个x，表示返回的分组，即返回map的Key |
| mapValues( ) 对每个value进行映射。返回一个map，该map的key仍是原map的key，value是映射后的值  Array 内容都可变，长度不可变  长度可变数组(ArrayBuffer)和长度不可变数组（Array） |

# 二、集合

## 2.1 序

|  |
| --- |
| 三大集合：Seq、Set、Map  在scala中，集合分为可变集合（Mutalbe）和不可变集合（immutable）  分别在 scala.collection.mutable.\_ 和scala.collection.immutable.\_包下  可变集合指：长度可变，内容可变  不可变集合：长度不可变，内容也不可变 |

2.2 Seq (即list)

|  |
| --- |
| 特殊：  Nil 表示 一个空的List 如：var list1= Nil  :: 这是一个List的方法。 使用 var list2 = 9::Nil 向一个空的List中添加9这个元素。这个方法是后向前的。 如2::3::4::list1 就是 List(4,3,2)  增  ++ 将一个集合中的所有 元素向另一个集合 **在后**我追加  ++：将一个集合中的所有 元素向另一个集合 在前我追加  ：： 将一个元素向集合中添加 如 8：：nill  Count 统计。 传入一个函数。 带条件的统计。  Sorted  sortBy 根据某个元素排序  sortWith 自定义排序  grouped 传入一个int。将，几个数分为一组  fold 叠加 传入两个参数。一个初始值和一个表达式  如 var list = List(1,2,3);list.fold(0)(\_+\_) 结果为6；  第一\_ 表示初始值或每次叠加前，上一次的叠加结果。  第二个\_ 表示每次要叠加的元素  foldLeft foldRight 区别 将初始值放元素的左边还是右边，然后依次将才表达式  filter 传入一个表达式，过滤出满足条件的元素。  aggregate  非分布式情况下，底层调用 foldRight  union inersct diff 并集 交集 差集  mkString() 同java的split  slice list截取操作 |

## 2.2 Set

|  |
| --- |
| Val hset =collection.mutalbe.HashSet(1,3,5) //新建  hset.add(4) //新增 hset +=4 |
| hset.remove(5) //删除 hset -= 5 |
|  |

## 2.3 Map

|  |
| --- |
| Val map = Map[String,Int](“a” -> 1 ,”b” -> 2) //新建  map.put(“c”,3) //新增 map += “c” -> 3 |
| map.remove(“a”) //删除 |
| map.get(“b”)  map.getOrElse(“a”,0) // |

# 三、面向对象

## 3.1 Object

|  |
| --- |
| 该关键词定义的对象，是一个单例对象，里面的方法和属性都是静态的，通过对象名.方法名（属性名）调用  apply方法，调用对象名（）时，默认调用这个方法。 |

## 3.2 class

|  |
| --- |
| 用class关键字定义类，在类后面的定义的是主构造器,  主构造器中定义 的参数名，会默认为 该类的成员变量。不用另外声明。  主构造器中定义参数用 var修饰，相当于提供了get/set方法。  用val修饰相当于只提供get方法。  不用var/val修改，则不提供get。  class Teacher(name :String,var sex:String){  }  还可以定义辅助构造器,辅助构造器中，需先调用主构造器  class Teacher(name :String,sex:String){  var age :int =\_  def this(name:String , ,sex:String ,age:Int){  this(name,sex);  this.age=age  }  }  类构造器的访问权限。  在构造器前添加private即可。  属性的访问权限  若在主构造器中的变量量用private修饰，则该属性的get/set方法是私有的，外部不能访问  类的访问权限，  在class 关键字之前添加private[this]则表示该类，只能在当前包可见，只包也不可见。  Private[包名] 表示，这个类可在这个包及子包下可见 |

## 3.3 伴生类/对象

|  |
| --- |
| class teacher{  }  object teacher{  }  前者是后者的伴生类，  后者是前者的伴生对象  在伴生对象中，可以访问伴生类的私有成员变量、方法 |

## 3.4 访问权限

|  |
| --- |
| 用private修饰  成员变量 外部 没法访问get/set  方法 外部没法访问这个方法  构造器 外部无法访问  类 外部不可见 |
|  |

## 3.5 trail特质

|  |
| --- |
| 特质（相当于java中的interface）  在特质中可以 定义实现的方法，也可以定义没有实现的方法 |
| 动态混入  创建时，用with关键字  var student = new Student() with TrailTest  这样student即可调用TrailTest的方法  可以用多个with，混入N个特质。  也可以在类后面使用with混入特质 |

## 3.6 抽象类

|  |
| --- |
| 使用abstract定义抽象类  可以具体实现方法  可以有没有实现的方法 |
|  |

## 3.7 final

|  |
| --- |
| 成员定义成final，不允许修改  方法定义成final ,不允许重写  类定义成final，不允许被继承。 |

## 3.8 type类型别名

|  |
| --- |
| class student {  type T = String  def method1 (msg : T ){  //do something  }  }  有些像java中的泛型 |

## 3.9 case与模式匹配

|  |
| --- |
| 支持模式 匹配，默认实现了Serializable接口  样例类 ：case class 类名（属性...）  用于封装数据  样例对象：case object 对象名  不能封装数据 |

|  |
| --- |
| 模式匹配  def stringMatch(str:String) = str match{  case “a” => println(“aa”)  case “b” => println(“aa”)  case “c” => println(“cc”)  }  感觉就是些 判断与匹配  可以匹配 字符 类型 List Array 元组 对象 |

# 四、隐式转换implicit

## 4.1 隐式参数

|  |
| --- |
| 方法如果有多个隐式参数 的话，只需要 上个图一个implicty关键字即可  隐式参数 列表 必须放在参数 列表后面  def say (implicit content :String) = println(content)  say方法的参数 是隐式参数。  如果没有传递参数编译器在编译的时候会自动从当前 的上下文中找一个隐式值（符合参数 的类型的隐式值）  上下文中，隐式参数不能有歧义（相同类型的隐式值 只能有一个）  Implicit val msg = “你好帅”  say |

## 4.2 隐式类型转换

|  |
| --- |
| 1、  implicit val double2Int = (double :Double) =>{  double.toInt  }  var a:Int =3.14;  此时3.14不过报错。因为已经调用了 隐式函数 double2Int.将3.14转成了int  2、  Implicit def file2RichFie(File file) =new RichFile(file)  //RichFile是一个自定义的一个类，扩展了File，添加了一个count方法，计算文件中的行数 。  使用  val file =new File(“...”);  var count =file.count() //此时也，file也被转型成了RichFile，具有了count的方法 |
|  |

## 4.3 隐式类

|  |
| --- |
| 隐式类，只能在Object中使用。作为Object的内部类  Object xxx{  Implicit class (file :File ){  def read = Source.fromFile(file).mkString  }  def main(args:Array[String]):Unit ={  File file =new File(“...”)  var content= file.read(); //此时file具有了read方法  }  } |