Scala阶段问题

1.静态语言与动态语言

奥义：

- 动态语言（弱类型语言）：在运行时，才确定数据类型，变量在使用之前无需申明类型，通常变量的值是被赋值的那个值的类型。如php、asp、javascript、python、perl...
- 静态语言（强类型语言）：在编译时，变量的数据类型就可以确定的语言，大多数静态语言要求在使用变量之前必须声明数据类型。如Java、C、C++、C#...

补充：

- **弱类型语言**是数据类型可以被忽略的语言。它与强类型语言相反，一个变量可以赋不同数据类型的值。一个变量的类型是由其上下文决定的，效率更高。
- **强类型语言**是必须强制确定数据类型的语言，一旦一个变量被指定了某个数据类型，如果不经过强制转换，那么它就永远是这种数据类型。一个变量的类型是申明的时候就已经确定的，更安全。

区别：

- 静态语言由于强制声明数据类型，让开发工具（IDE）对代码有很强的判断能力，在实现复杂的业务逻辑和开发大型商业系统、以及那些声明周期很长的应用中，开发者可以依托强大的IDE来更高效、更安全地开发。
- 动态语言思维不受约束，可以任意发挥，把更多的精力放在产品本身上；集中思考业务逻辑实现，思考过程就是实现过程。

2.option

2.1 java中

optional是Java 8 引入的特性

解决：空指针异常（NullPointerException）

本质：这是一个包含有可选值的包装类，这意味着 Optional 类既可以含有对象也可以为空。

https://www.oschina.net/translate/understanding-accepting-and-leveraging-optional-in?lang=chs&page=2#

2.2 Scala中

比较特殊的`None`，是`Option`的两个子类之一，另一个是`Some`，用于安全的函数返回值。

`scala`推荐在可能返回空的方法使用`Option[X]`作为返回类型。如果有值就返回`Some[X]`，否则返回`None`

```
def get(key: A): Option[B] = {
    if (contains(key))
        Some(getValue(key))
    else
        None
}
```

`Nil`表示`长度为0的List`。

3.高阶函数与匿名函数

代码：

```
/**
 * f函数表示：接收两个Int类型的参数，返回一个int类型的函数
 * 函数：看什么时候被调用
 * 参数是函数
 * @param a
 * @param f
 * @return
 */
def f7(a:Int,f:(Int,Int)=>Int)={
    val result = f(1, 2)
    a*result
}
def f8(a:Int,b:Int):Int=a+b
println(f7(1, f8))

val sum: (Int, Int) => Int = (a:Int, b:Int)=>a+b

//用匿名函数写，因为重要的是body
println(f7(1, (a:Int,b:Int)=>a+b))
println(f7(1, sum))
```

匿名函数用`val`接收

4.闭包概念

一句话解释就是：

- 存在自由变量的函数就是闭包。
- 一个持有外部环境变量的函数就是闭包。
- 在这个例子里函数b因为捕获了外部作用域（环境）中的变量a，因此形成了闭包。 而由于变量a并不属于函数b，所以在概念里被称之为「自由变量」。

关键点：

- 函数
- 自由变量
- 环境

例子：

```
let a = 1
let b = function(){
    console.log(a)
}
```

在这个例子里函数b因为捕获了外部作用域（环境）中的变量a，因此形成了闭包。 而由于变量a并不属于函数b，所以在概念里被称之为「自由变量」。

情况：

- 函数作为值返回
- 直接或间接执行内部函数

特点

- 基于词法作用域的查找规则
- 在一个函数内部定义一个内部函数，然后将内部函数作为值返回，或直接or间接的立即执行内部函数
- 拥有更长的生命周期，保持对当前词法作用域的引用

5.对None进行get

会报错：`NoSuchElementException`

6.isInstanceOf和asInstanceOf

如果实例化了子类的对象，但是将其赋予了父类类型的变量，

在后续的过程中，又需要将父类类型的变量转换为子类类型的变量，应该如何做?

Ø 首先，需要使用isInstanceOf 判断对象是否为指定类的对象，如果是的话，则可以使用asInstanceOf 将对象转换为指定类型；

Ø 注意：p.isInstanceOf[XX] 判断 p 是否为 XX 对象的实例；p.asInstanceOf[XX] 把 p 转换成 XX 对象的实例

Ø 注意：如果没有用isInstanceOf 先判断对象是否为指定类的实例，就直接用 asInstanceOf 转换，则可能会抛出异常；

Ø 注意：如果对象是 null，则isInstanceOf 一定返回 false， asInstanceOf 一定返回 null；

Ø Scala与Java类型检查和转换 Scala Java obj.isInstanceOf[C] obj instanceof C obj.asInstanceOf[C] (C)obj classOf[C] C.class

Ø 举例说明：

```
package cn.itcast.extends_demo

class Person3 {}
class Student3 extends Person3
object Student3{
  def main (args: Array[String] ) {
  val p: Person3 = new Student3
  var s: Student3 = null
  //如果对象是 null，则 isInstanceOf 一定返回 false
  println (s.isInstanceOf[Student3])
```

```
  // 判断 p 是否为 Student3 对象的实例
 if (p.isInstanceOf[Student3] ) {
  //把 p 转换成 Student3 对象的实例
   s = p.asInstanceOf[Student3]
 }
 println (s.isInstanceOf[Student3] )
 }
}
```

Scala中getClass 和 classOf

Ø isInstanceOf 只能判断出对象是否为指定类以及其子类的对象，而不能精确的判断出，对象就是指定类的对象；

Ø 如果要求精确地判断出对象就是指定类的对象，那么就只能使用 getClass 和 classOf 了；

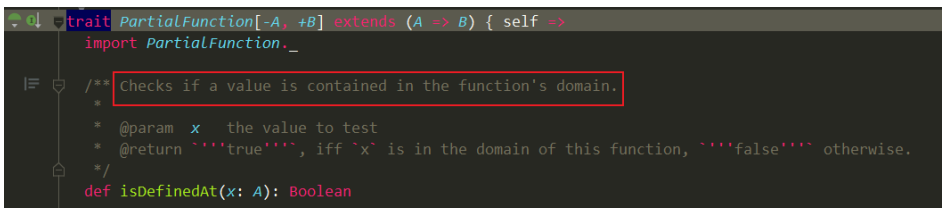
Ø p.getClass 可以精确地获取对象的类，classOf[XX]可以精确的获取类，然后使用 == 操作符即可判断；

Ø 举例说明：

```
package cn.itcast.extends_demo

class Person4 {}
class Student4 extends Person4
object Student4{
 def main(args: Array[String]) {
  val p:Person4=new Student4
  //判断p是否为Person4类的实例
  println(p.isInstanceOf[Person4])//true
  //判断p的类型是否为Person4类
  println(p.getClass == classOf[Person4])//false
  //判断p的类型是否为Student4类
  println(p.getClass == classOf[Student4])//true
 }
}
```

7.偏函数中isDefinedAt函数

1. 定义接收数据的范围 isDefinedAt()
   - 

   - 检查值是否包含在函数的域中
2. 定义好后如何处理值，即具体处理 apply()

```scala
def MyTest:PartialFunction[Any,Int] = {
  case i: Int => i + 1
}
```

重点看这两处

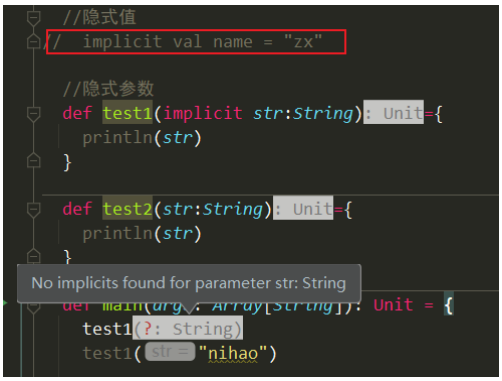
```scala
def main(args: Array[String]): Unit = {
  val list = List(1, 3, 5, "seven")
  //用map会报错，seven不匹配
//    list.map(MyTest).foreach(println)

  list.collect(MyTest).foreach(println)



  //逻辑简单例子
  println(MyTest1("hello"))
  def MyTest1:PartialFunction[String,String]={
    case "hello" => "hello"
    case "hello1" => "hello1"
    case "hello2" => "hello2"
    case _ => "no match"
```

8.缺少隐式值



问题可能情况：

- 没有隐式值
- 隐式值和隐式参数类型不一致
- 有多个隐式值

解决：

- 添与隐式参数同类型的隐式值（同类型唯一）

9.关于并发

Java中

并发基于数据共享

锁的概念

锁的关键字

Scala中

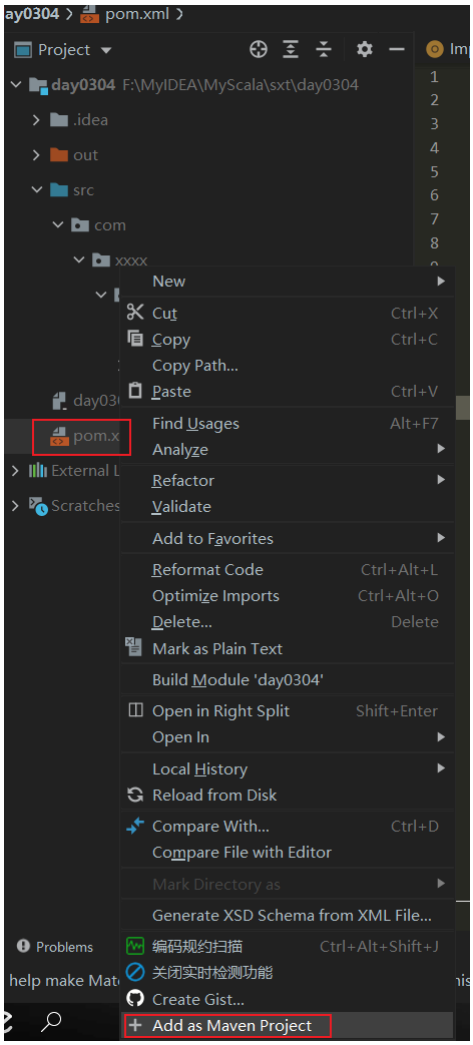
actor是传递信息，不是共享

2.12.x 需要单独导lib包

通信框架：akka（Scala）是库与netty（Java）比较

10.普通项目转maven项目

1. 导入pom.xml
    1. 项目->new->File->pom.xml
    2. ```
       <?xml version="1.0" encoding="UTF-8"?>
       <project xmlns="http://maven.apache.org/POM/4.0.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       http://maven.apache.org/xsd/maven-4.0.0.xsd">
           <modelVersion>4.0.0</modelVersion>

           <groupId>com.xxxx</groupId>
           <artifactId>项目名字</artifactId>
           <version>1.0-SNAPSHOT</version>
       <dependencies>
       </dependencies>
       </project>
       ```
    3. 

2. 建包格式
       ```

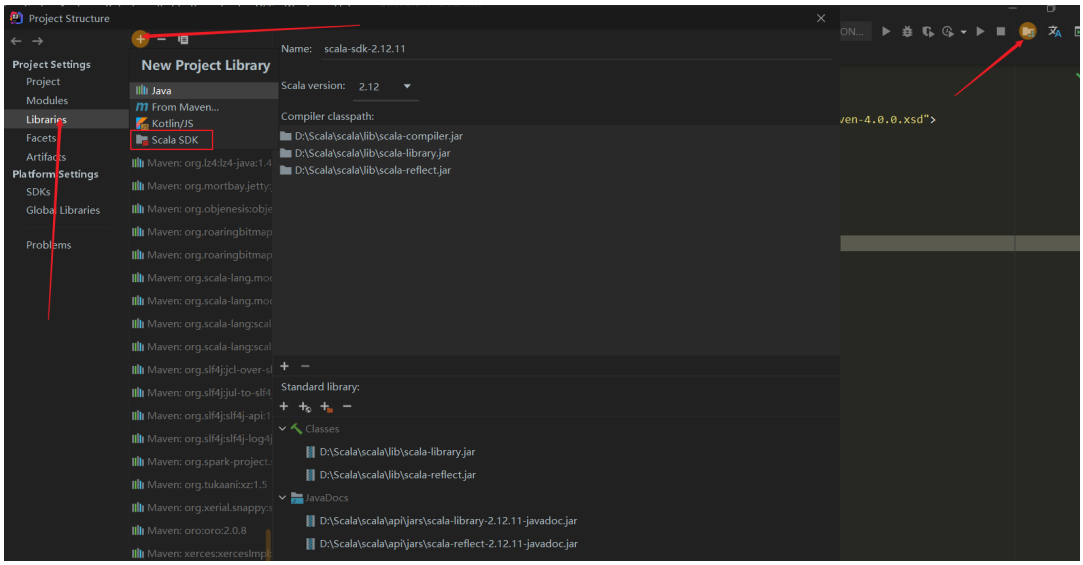
1. java:**src/main/java**
2. resources:**src/main/resources**
3. scala:**src/main/scala**

mark Directory as -->

3. 导入依赖
    1. lib

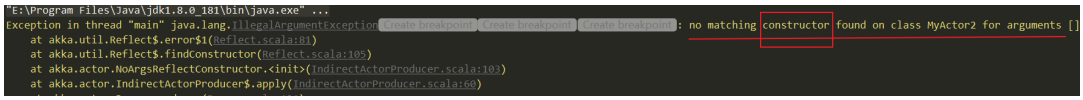
11.创建maven导入Scala库

方法:

- 

12.actor中通信未传值错误

报错:

- 

关键代码:

- ```
  class MyActor1 extends Actor{
    override def receive: Receive = {
      case msg:String =>{
        println(msg + " 1: " + self.path.name)
        println("已读")
      }
      case _:Int =>{
        println("number"+ " 1: "+ self.path.name)
      }
      case _ =>{
        println("no"+ " 1: "+ self.path.name)
      }
    }
  }
  ```
  ```

- ```
  class MyActor2(actor11: ActorRef) extends Actor{
    override def receive: Receive = {
      case msg:String =>{
        println(msg + " 2: " + self.path.name)
        actor11 ! "好"
      }
      case _ =>{
        println("ohohoh"+ " 2: "+ self.path.name)
      }
    }
  }
  ```
- ```
  private val actor21: ActorRef = system.actorOf(Props[MyActor2], "actor21")
  ```
- ```
  actor21 ! "你好"
  ```

解决：

- ```
  private val actor21: ActorRef = system.actorOf(Props(new MyActor2(actor11)), "actor21")
  ```

结果：

-
  ```
  你好 2: actor21
  你好 1: actor11
  已读
  ```

13.hadoop环境变量

报错：

-
  ```
  SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
  Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
  21/03/04 21:40:33 INFO SparkContext: Running Spark version 2.4.6
  21/03/04 21:40:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
  21/03/04 21:40:34 ERROR Shell: Failed to locate the winutils binary in the hadoop binary path
  java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
      at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:378)
      at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:393)
      at org.apache.hadoop.util.Shell.<clinit>(Shell.java:386)
      at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:79)
      at org.apache.hadoop.security.Groups.parseStaticMapping(Groups.java:116)
  ```

问题：

- 未设置win端hadoop环境变量

解决：

- 配置win端hadoop环境变量

准备：

1. hadoop 3.1.2解压
2. winutils-master.zip 解压 找到3.1.2的bin对前一步的bin进行覆盖
3. 将winutils.exe和hadoop.dll拷贝到 C:\Windows\System32

配置：

1. HADOOP_HOME-->D:\software\hadoop-3.1.2
2. HADOOP_USER_NAME-->root
3. Path --> %HADOOP_HOME%\bin;%HADOOP_HOME%\sbin;

4. 注意JDK需要和服务器版本一致