1、scala语言有什么特点？什么是函数式编程？有什么优点？

1、scala语言集成面向对象和函数式编程

2、函数式编程是一种典范，将电脑的运算视作是函数的运算。

3、与过程化编程相比，函数式编程里的函数计算可以随时调用。

4、函数式编程中，函数是一等功明。

2、scala中的闭包

1、定义：你可以在任何作用域内定义函数:包，类甚至是另一个函数或方法。在函数体内，可以访问到相应作用域内地任何变量。(重点)函数可以在变量不再处于作用域内时被调用。

例如：

```
def mulBy(factor:Double) = (x:Double) => factor * x
//开始调用
val tripe = mulBy(3)
val half = mulBy(0.5)
println(tripe(14) + " " + half(14))
```

这就是一个闭包

3、scala中的柯里化

定义：柯里化指的是将原来接受两个参数的函数变成新的接受一个参数的函数的过程。新的函数返回一个以原有的第二个参数作为参数的函数

例如：

```
def mul(x:Int,y:Int) = x * y  //该函数接受两个参数
def mulOneAtTime(x:Int) = (y:Int) => x * y  //该函数接受一个参数生成另外一个接受单个参数的函数
这样的话，如果需要计算两个数的乘积的话只需要调用：
mulOneAtTime(5)(4)
```

这就是函数的柯里化

4、scala中的模式匹配

scala的模式匹配包括了一系列的备选项，每个替代项以关键字大小写为单位，每个替代方案包括一个模式或多个表达式，如果匹配将会进行计算，箭头符号=>将模式与表达式分离

例如：

```
obj match{
    case 1 => "one"
    case 2 => "two"
    case 3 => "three"
    case _ => default
}
```

5、case class和class的区别

case class:

是一个样本类，样本类是一种不可变切可分解类的语法糖，也就是说在构建的时候会自动生成一些语法糖，具有以下几个特点：

1、自动添加与类名一致的构造函数(也就是半生对象，通过apply方法实现)，也就是说在构造对象的时候不需要使用new关键字

2、样本类中的参数默认是val关键字，不可以修改

3、默认实现了toString，equals,hashcode,copy方法

4、样本类可以通过==来比较两个对象，不在构造方法内地二属性不会用在比较上

class:

class是一个类

1、class在构造对象的时候需要使用new关键字才可以。

6、谈谈scala中的隐式转换

所谓的隐式转换函数(implicit conversion function)指的事那种以implicit关键字申明的带有单个参数的函数，这样的函数会被自动的应用，将值从一种类型转换为另一种类型

比如：需要把整数n转换成分数n/1

implicit def int2Fraction(n:Int) = Fraction(n,1)

这样就可以做如下表达式求职：

val result = 3 * Fraction(4,5)

此时，隐士转换函数将整数3转换成一个Fraction对象，这个对象接着又被乘以Fraction(4,5)

7、scala中的伴生类和伴生对象是怎么一回事

1. 在scala中，单例对象与类同名时，该对象被称为该类的伴生对象，该类被称为该对象的伴生类。
2. 伴生类和伴生对象要处在同一个源文件中
3. 伴生对象和伴生类可以互相访问其私有成员
4. 不与伴生类同名的对象称之为孤立对象

```
代码示例：
//定义一个类
class MyClass(number: Int, text: String) {
  private val classSecret = 42
  def x = MyClass.objectSecret + "?"  // MyClass.objectSecret->在类中可以访问伴生对象的方法，在
}
//定义一个伴生对象
object MyClass { // 和类名称相同
  private val objectSecret = "42"
   def y(arg: MyClass) = arg.classSecret -1 // arg.classSecret -> 在伴生对象中可以访问类的常量
}
MyClass.objectSecret // 无法访问
MyClass.classSecret // 无法访问
new MyClass(-1, "random").objectSecret // 无法访问
new MyClass(-1, "random").classSecret // 无法访问
```

8、scala和java 的区别

1、变量申明：

scala：只需要申明是val或是var，具体的类型(比如String，Int，Double等等)，由编译器自行推断

java： 需要在变量前面先注明变量的类型

2、返回值：

scala：申明返回值是在后面，并且不需要return语句，非要用，也不是不可以

java： 如果有返回值，需要return语句

3、结束符

scala：不需要使用分号作为结束符

java： 每个语句结束后需要分号作为结束符

4、循环

scala：循环语句可以用于守卫

java： 不可以这么写

5、通配符：

scala：_

java： *

6、构造器

scala：构造器名称为this，scala的辅助构造器之前需要有一个主构造器或者其他辅助构造器，并且scala的

构造器参数可以直接放在类的后面

java： 构造器名称需要与类名称一样

7、内部类

scala：scala实例化的内部类是不同的，可以使用类型投影，例如 Network#Person表示Network的Person类

java：内部类从属于外部类

8、接口

scala：scala中接口称为特质(trait)，特质中是可以写抽象方法，也可以写具体的方法体以及状态。且类是可以实现多个特质的。

特质中未被实现的方法默认就是抽象的

子类的实现或继承统一使用的事extends关键字，如果需要实现或继承多个使用with关键字

特质中可以有构造器

特质可以继承普通的类，并且这个类称为所有继承trait的父类

java： java中的接口(interface)，接口中的方法只能是抽象方法，不可以写具体包含方法体的方法

接口中不能有抽象的属性，且属性的修饰符都是public static final

类实现接口需要使用implements关键字，实现多个接口，需要用逗号隔开

接口中不可以有构造器

接口不可以继承普通的类

9、赋值

scala：scala中的赋值语句返回结果是unit的不可以串联，例如x=y=1,这样是有问题的，x并没有被赋值为1

java： x=y=1，这样是没问题的

9、谈谈scala的尾递归

1. 正常得递归，每一次递归步骤，需要保存信息到堆栈中去，当递归步骤很多的时候，就会导致内存溢出

2. 尾递归，就是为了解决上述的问题，在尾递归中所有的计算都是在递归之前调用，编译器可以利用这个属性避免堆栈错误，尾递归的调用可以使信息不插入堆栈，从而优化尾递归

例如：

```
正常递归: def sum(n:Int):Int = {
    if (n == 0){
      n
    }else{
      n + sum(n - 1)
    }
  }//执行结果sum(5)
```

5 + sum(4) // 暂停计算 => 需要添加信息到堆栈

5 + (4 + sum(3))

5 + (4 + (3 + sum(2)))

5 + (4 + (3 + (2 + sum(1))))

5 + (4 + (3 + (2 + 1)))

15

```
尾递归
@tailrec    //告诉编译器，强制使用尾递归
  def tailSum(n:Int,acc:Int = 0):Int = {
    if (n ==0 ){
      acc
    }else{
      tailSum(n - 1,acc + n)
    }
  }//执行结果tailSum(5) // tailSum(5, 0) 默认值是0
```

tailSum(4, 5) // 不需要暂停计算

tailSum(3, 9)

tailSum(2, 12)

tailSum(1, 14)
tailSum(0, 15)
15

10、 var，val和def三个关键字之间的区别?

1. var是变量声明关键字，类似于Java中的变量，变量值可以更改，但是变量类型不能更改。
2. val常量声明关键字。
3. def 关键字用于创建方法（注意方法和函数的区别）
4. 还有一个lazy val（惰性val）声明，意思是当需要计算时才使用，避免重复计算

```
代码示例:
var x = 3 //  x是Int类型
x = 4      //
x = "error" // 类型变化，编译器报错'error: type mismatch'
val y = 3
y = 4        //常量值不可更改，报错 'error: reassignment to val'
def fun(name: String) = "Hey! My name is: " + name
fun("Scala") // "Hey! My name is: Scala"
//注意scala中函数式编程一切都是表达式
 lazy val x = {
   println("computing x")
   3
 }
 val y = {
   println("computing y")
   10
 }
 x+x  //
 y+y  // x 没有计算，打印结果"computing y"
```

11、 trait（特质）和abstract class（抽象类）的区别?

（1）一个类只能集成一个抽象类，但是可以通过with关键字继承多个特质；
（2）抽象类有带参数的构造函数，特质不行（如 trait t（i：Int）{}，这种声明是错误的）

12、object和class的区别?

object是类的单例对象，开发人员无需用new关键字实例化。如果对象的名称和类名相同，这个对象就是伴生对象（深入了解请参考问题Q7）

```
代码示例
//声明一个类
class MyClass(number: Int, text: String) {
  def classMethod() = println(text)
}
//声明一个对象
object MyObject{
  def objectMethod()=println("object")
}
 new MyClass(3,"text").classMethod() //打印结果test，需要实例化类
 Myclass.classMethod()  //无法直接调用类的方法
 MyObject.objectMethod() //打印结果object，对象可以直接调用方法
```

13、 case class （样本类）是什么?

样本类是一种不可变且可分解类的语法糖，这个语法糖的意思大概是在构建时，自动实现一些功能。样本类具有以下特性：

（1）自动添加与类名一致的构造函数（这个就是前面提到的伴生对象，通过apply方法实现），即构造对象时，不需要new；

（2）样本类中的参数默认添加val关键字，即参数不能修改；

（3）默认实现了toString，equals，hashcode，copy等方法；

（4）样本类可以通过==比较两个对象，并且不在构造方法中定义的属性不会用在比较上。

```
代码示例
//声明一个样本类
case class MyCaseClass(number: Int, text: String, others: List[Int]){
 println(number)
}
//不需要new关键字，创建一个对象
val dto = MyCaseClass(3, "text", List.empty) //打印结果3
//利用样本类默认实现的copy方法
dto.copy(number = 5) //打印结果5
 val dto2 = MyCaseClass(3, "text", List.empty)
 pringln(dto == dto2) // 返回true，两个不同的引用对象
 class MyClass(number: Int, text: String, others: List[Int]) {}
 val c1 = new MyClass(1, "txt", List.empty)
 val c2 = new MyClass(1, "txt", List.empty)
 println(c1 == c2 )// 返回false,两个不同的引用对象
```

14、 unapply 和apply方法的区别， 以及各自使用场景?

先讲一个概念——提取器，它实现了构造器相反的效果，构造器从给定的参数创建一个对象，然而提取器却从对象中提取出构造该对象的参数，scala标准库预定义了一些提取器，如上面提到的样本类中，会自动创建一个伴生对象（包含apply和unapply方法）。

为了成为一个提取器，unapply方法需要被伴生对象。

apply方法是为了自动实现样本类的对象，无需new关键字。

15、 Scala类型系统中Nil, Null, None, Nothing四个类型的区别?

1、Null是一个trait（特质），是所以引用类型AnyRef的一个子类型，null是Null唯一的实例。

2、Nothing也是一个trait（特质），是所有类型Any（包括值类型和引用类型）的子类型，它不在有子类型，它也没有实例，实际上为了一个方法抛出异常，通常会设置一个默认返回类型。

3、Nil代表一个List空类型，等同List[Nothing]

4、None是Option monad的空标识（深入了解请参考问题Q11）

16、Unit类型是什么?

Unit代表没有任何意义的值类型，类似于java中的void类型，他是anyval的子类型，仅有一个实例对象"( )"

17、 call-by-value和call-by-name求值策略的区别?

（1）call-by-value是在调用函数之前计算；

（2）call-by-name是在需要时计算

```
 示例代码
//声明第一个函数
def func(): Int = {
  println("computing stuff....")
  42 // return something
}
//声明第二个函数，scala默认的求值就是call-by-value
```

```
def callByValue(x: Int) = {
  println("1st x: " + x)
  println("2nd x: " + x)
}
//声明第三个函数，用=>表示call-by-name求值
def callByName(x: => Int) = {
  println("1st x: " + x)
  println("2nd x: " + x)
}
//开始调用
//call-by-value求值
callByValue(func())
//输出结果
//computing stuff....
//1st x: 42
//2nd x: 42
//call-by-name求值
callByName(func())
//输出结果
//computing stuff....
//1st x: 42
//computing stuff....
//2nd x: 42
```

18、Option类型的定义和使用场景?

在Java中，null是一个关键字，不是一个对象，当开发者希望返回一个空对象时，却返回了一个关键字，为了解决这个问题，Scala建议开发者返回值是空值时，使用Option类型，在Scala中null是Null的唯一对象，会引起异常，Option则可以避免。Option有两个子类型，Some和None（空值）

```
代码示例:
val person: Person = getPersonByIdOnDatabaseUnsafe(id = 4) // 如果没有id=4的person时，返回nu
println(s"This person age is ${person.age}") //如果是null，抛出异常
val personOpt: Option[Person] = getPersonByIdOnDatabaseSafe(id = 4) // 如果没有id=4的person
personOpt match {
  case Some(p) => println(s"This person age is ${p.age}")
  case None => println("There is no person with that id")
}
```

19、yield如何工作?

yield用于循环迭代中生成新值，yield是comprehensions的一部分，是多个操作（foreach, map, flatMap, filter or withFilter）的composition语法糖。（深入了解请参考问题Q14）

```
代码示例:
// <-表示循环遍历
scala> for (i <- 1 to 5) yield i * 2
res0: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 4, 6, 8, 10)
```

20、解释隐示参数的优先权

在Scala中implicit的功能很强大。当编译器寻找implicits时，如果不注意隐式参数的优先权，可能会引起意外的错误。因此编译器会按顺序查找隐式关键字。顺序如下：

（1）当前类声明的implicits；

（2）导入包中的 implicits；
（3）外部域（声明在外部域的implicts）；
（4）inheritance
（5）package object
（6）implicit scope like companion objects

21、comprehension（推导式）的语法糖是什么操作?

comprehension（推导式）是若干个操作组成的替代语法。如果不用yield关键字，comprehension（推导式）可以被forech操作替代，或者被map/flatMap，filter代替。

```
示例代码：
// 三层循环嵌套
for {
  x <- c1
  y <- c2
  z <- c3 if z > 0
} yield {...}
//上面的可转换为
c1.flatMap(x => c2.flatMap(y => c3.withFilter(z => z > 0).map(z => {...})))
```

22、谈谈对Scala的Streams的理解?

```
// Stream:Stream is lazy List;
// Stream惰性求值指它只确定第一个值，后面的值用到再求值，这样可以防止数据过大全部加载导致内存溢出
// 将Range转化成Stream
val stream = (1 to 1000).toStream
println(stream)     // Stream(1, ?)
println(stream.head) // 1
println(stream.tail)  // Stream(2, ?)
```

23、什么是vaule class?

开发时经常遇到这个的问题，当你使用integer时，希望它代表一些东西，而不是全部东西，例如，一个integer代表年龄，另一个代表高度。由于上述原因，我们考虑包裹原始类型生成一个新的有意义的类型（如年龄类型和高度类型）。

Value classes 允许开发者安全的增加一个新类型，避免运行时对象分配。有一些 必须进行分配的情况 and 限制,但是基本的思想是：在编译时，通过使用原始类型替换值类实例，删除对象分配。

24、Option，Try 和 Either 三者的区别?

1、这三种monads允许我们显示函数没有按预期执行的计算结果。
2、Option表示可选值，它的返回类型是Some（代表返回有效数据）或None（代表返回空值）。
3、Try类似于Java中的try/catch，如果计算成功，返回Success的实例，如果抛出异常，返回Failure。
4、Either可以提供一些计算失败的信息，Either有两种可能返回类型：预期/正确/成功的 和 错误的信息。

```
代码示例：
//返回一个Either类型//返回一个E
def personAge(id: Int): Either[String, Int] = {
  val personOpt: Option[Person] = DB.getPersonById(id) //返回Option类型，如果为null返回None,
personOpt match {
    case None => Left(s"Could not get person with id: $id")  //Left 包含错误或无效值
    case Some(person) => Right(person.age)                    //Right包含正确或有效值
}
```

25、什么是高阶函数?

高阶函数指能接受或者返回其他函数的函数，scala中的filter map flatMap函数都能接受其他函数作为参数。