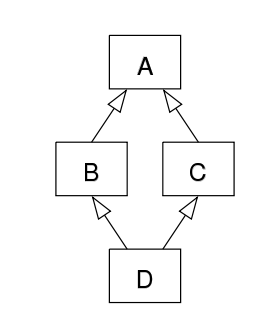
课前准备:

多重继承的钻石问题,也是为什么java和c都不支持多重继承的原因







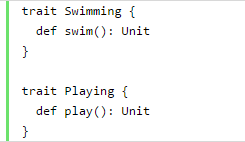


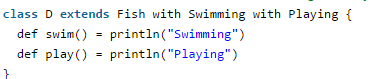
Trait网上文章

一、因为多重继承产生的钻石问题,因此java不支持多继承,只支持一个类别的基础,但是提供了接口功能。但是接口不能实现,要子类去实现,因此相同功能的代码可能在不同子类要重复写多次,因此不够完美。因此scala追加了一个trait功能,因此他与接口是有区别的。

二、最简单的trait—当作接口使用

基本上trait可以理解成是java的接口,因此trait可以当作java的接口使用





即D是一个fish类的子类,可以游泳和玩耍功能

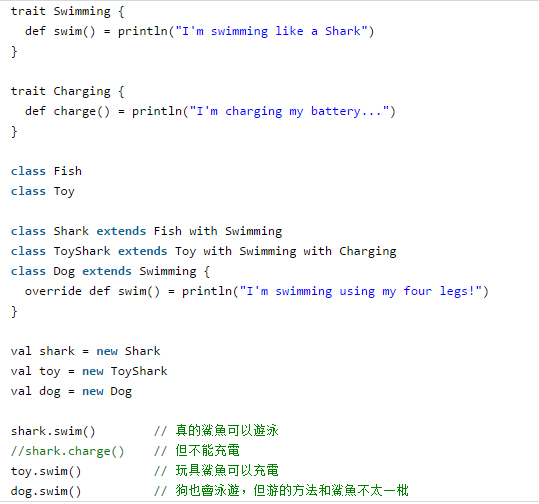


上面这个是有问题的,因为没有实现游泳的具体实现,因此E要被定义成abstract的

三、有具体实现的trait,即与java接口不同,trait可以有具体实现

因为有具体的实现,因此类mix一个trait的时候,可以拥有了该trait的实现的功能,因此可以间接的实现了多重继承功能,因此也就相同代码实现一次即可被不同类使用了,

从而提高了代码的使用率。

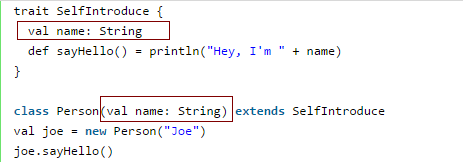


四、trait如何限制,不让任意一个类都使用该trait

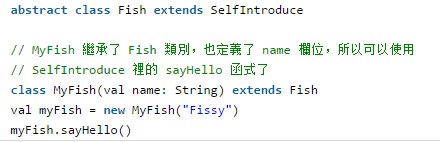
与java接口不同的是,trait可以设置一个限制符,允许某一个特定的子类才能使用该trait。

1．未定义的占位符号形式

Trait的某些函数和属性是有特定条件下的变数的,因此要子类去实现,因此该trait可以相当于java的抽象类功能。



特例



说明fish没有实现父类trait的name,因此他必须是抽象类

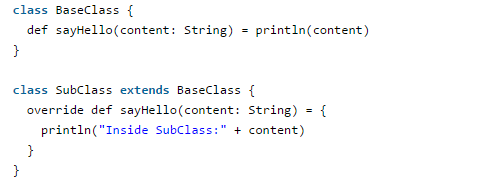
2．限制trait只能被某些类或者该类的子类mix,其他类不允许使用

语法trait TraitA extends BaseClass

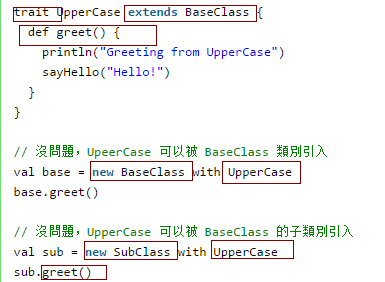
注意: 这里的extends不是继承关系,而是表示该trait只能被纳入的限制范围。

因此trait只能在BaseClass类中被mix混入,因此该trait可以使用BaseClass类中的成员函数和变量.

例如,定义两个基类

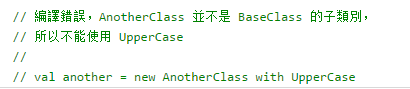


定义trait,只能用在基类上



这两个类都是可以mix一个trait的

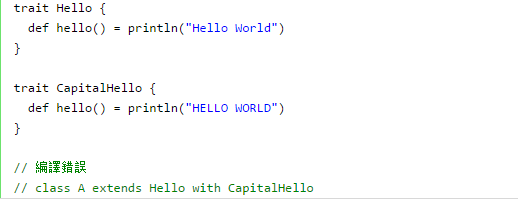




这样就不允许,因为anotherClass不是baseclass,因此upperCase这个trait不能混入到该类中,因此编译错误。

五、这样trait那算是多重继承么?没有钻石问题产生么?

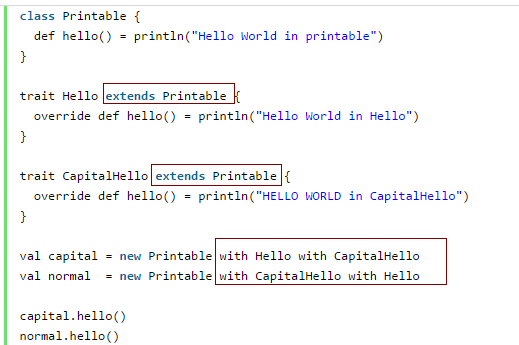
1.即如果一个类引入的两个trait有相同的函数名字,那么产生钻石问题,该类要执行哪一个trait呢?



可以看到相同名字的时候,编译就错误了,因此不会产生钻石问题

2．但是另外,如果相同的函数名字的不同trait都使用的extends关键字限定量范围,那么还是可以编译通过的,并且执行最右边的trait。

例如



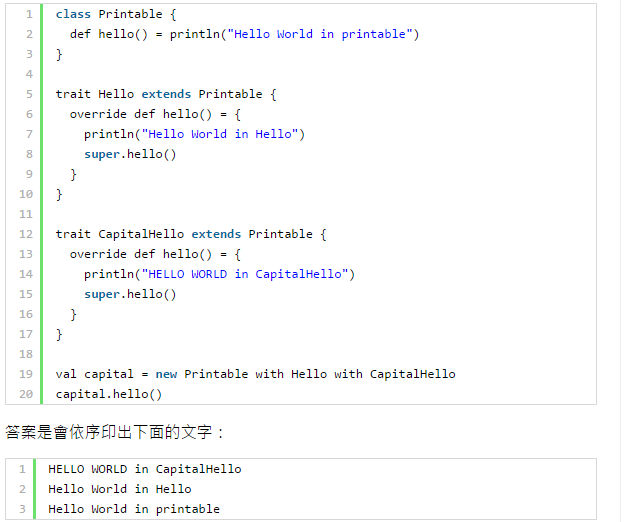
Capital会执行CapitalHello里面的hello,

Normal会执行Hello里面的hello

3.trait中的super

Super在trait中使用的时候,不是执行父类的方法,而是执行多个trait中相同方法,从右往左边执行。

例如

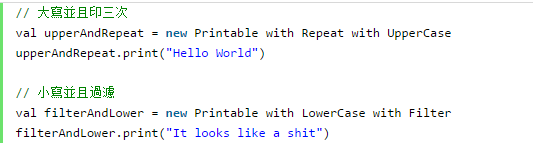


六、使用trait可以很容易实现包装模式

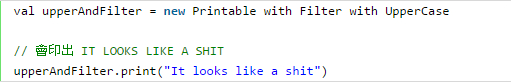
1．有如下集中trait实现方式,重复三次、全部编写大写、全部小写、过滤关键字四种装饰者。



不同的组合trait,可以产生不同的效果



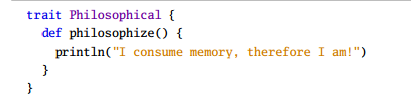
下面的组合,因为先进行大写,在进行过滤关键字,因此不会真的过滤掉关键字



第12章 Traits

1. Trait是对方法和属性的封装,用于代码重用
2. 重用trait可以将trait混用mix到一个class中
3. How traits work

1．一个trait定义和class其实一样,只是关键字是trait



2．该trait默认继承AnyRef.类

3．使用extends或者with 关键字,将该trait混入到其他class中

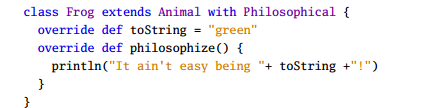
4．使用混入mix的意思是,他可以让一个class类混入多个trait,而不是被限制与继承一个父类了,同时与其他语言,比如java的多实现也不同,因此用混入mix含义

5．例如

class Frog extends Philosophical { override def toString = "green" }

可以看到frog就有了Philosophical的方法了

6．可以返回类型设置为父trait类型





7．Trait像java的接口,但是他可以实现具体的方法,也可以有属性。

事实上你可以在trait上做任何事情,class能做的,trait都可以做,只有两点与class不同.

a.trait不能像class一样有构造函数参数

class Point(x: Int, y: Int) 这个是允许的

trait NoPoint(x: Int, y: Int) 这个是不允许的

b.第二点不同是super方法在class和trait中的含义,

calss的super是静态的,创建calss的时候就已知的,

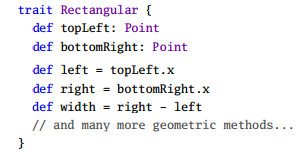
trait是动态的,在class进行mix混入trait的时候才能根据trait的顺序,知道super是哪个trait。

第二节 Thin versus rich interfaces 瘦与富接口

1. Trait就是丰富一个类,让其添加一些新的功能,因此trait能够丰富一个瘦接口。
2. Scala可以定义公共方法,通过mix的方式将其添加到不同的类中,而java添加接口,需要不同的具体类实现不同的接口,可能会造成重复代码工作。
3. 针对有可能变化的方法,trait可以定义成抽象的,子类覆盖即可,即具体的子类是可以覆盖trait中的方法的

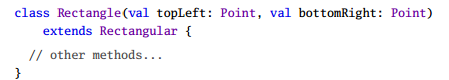
第三节 一个例子--- Rectangular objects 长方形对象

1. 图像库有许多不同的方法代表长方形,比如窗口,图片,鼠标画长方形等。
2. 用scala的trait定义一个抽象类,让子类就可以共同拥有同一个功能了。



子类只要传递两个具体的点即可

1. 具体实现



第四节 The Ordered trait ----有比较含义的trait

1. 元素之间是有比较需求的,因此提供> < >=等方法比较简单明了,因此scala提供了

scala.math. Ordered

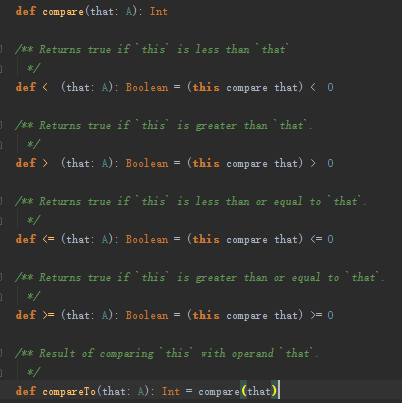
即 trait Ordered[A] extends Any with java.lang.Comparable[A] {

而public interface Comparable<T> {

public int compareTo(T o);

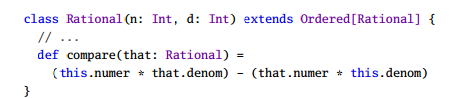
注意:有泛型需求

1. 看一下具体定义



因此子类只要实现了compare方法即可使用> <等triat方法了

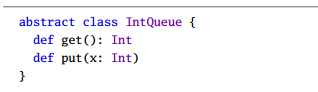
1. 两个要求



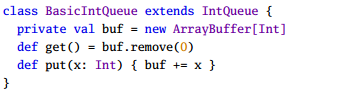
1. 要定义一个泛型,其中泛型就表示要比较的两个元素,因此compare就是对应的元素
2. 实现了compare方法

第五节 Traits as stackable modifications 可以叠加一组trait,并且可以修改叠加顺序,产生作用不一样的结果

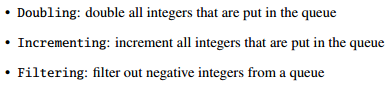
1. 先定义一个基类,用于表示一个队列,有put和get方法



2.定义一个具体实现

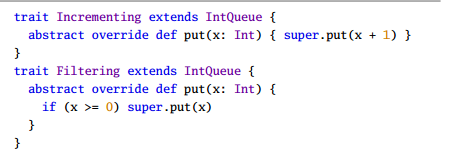


3.定义3个trait



分别让进入队列的int扩大一倍、增加1个,过滤负数





1. 上面trait的几点说明

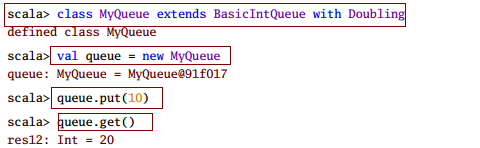
a. extends,表示这三个trait只能被IntQueue或者子类进行mix,其他类是不允许mix这三个trait的

b.super前面有abstract关键字,正常class这么写是非法的,因为他们运行期调用父类的方法.但是父类IntQueue是没有实现put方法的,是一个抽象方法,因此要加入abstract.表示具体要子类去实现该方法,即混用该trait的类必须实现put方法。

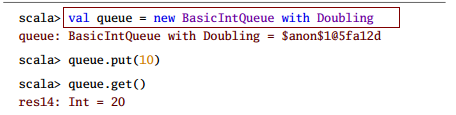
c.super,他指代的不是父类,而是从右边开始,挨着他最近的左边的拥有同样函数名字的trait的实现。

5．例子

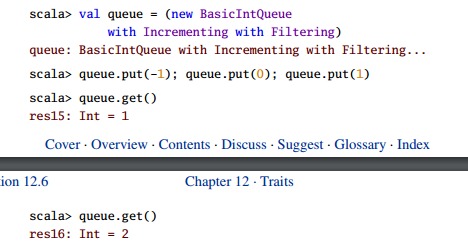
下面的例子中basicIntQueue已经实现了put和get功能了,因此可以mix Doubling



可以看到上面的例子定义的myQueue没有任何代码,但是却产生的强大的功能.但是这么产生一个单独的myQueue类有时候是没意义的,因此可以改写。

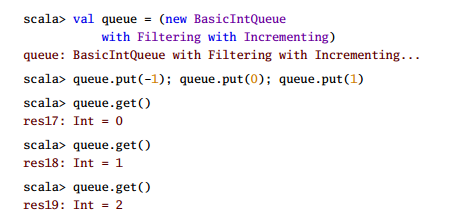


1. Trait的顺序问题,即super的影响



可以看到先执行filter,然后执行increment,因此第一个-1被过滤掉了,get不到数据。

反过来看:



就可以产生3次get了

第六节 Why not multiple inheritance 为什么这样就不会是多次继承了

1. Super不是真正调用父类,而是线性的方式从右到左的执行,因此不是多继承。

而是按照顺序,执行多种实现方式

1. 如果没有super,则找到最右边第一次出现的函数名后,就停止调用了,即从多个继承中选择了一个方式去执行,选择的顺序是固定的.因此也不属于多次继承问题。

第七节 To trait, or not to trait? 是否要用trait,即用triat还是用抽象类等其他方式呢

1. 如果你的行为不会被重用,那么就实现具体的class,这个时候不用使用trait
2. 如果一个行为被重复使用多次,并且不同类都可能被重复使用,那么就使用trait,但是不能用extends限制使用的类的范围。
3. 等等 有机会看一下吧，应该意义并不是那么大。