1.本章讲述类型参数化type parameterization的细节

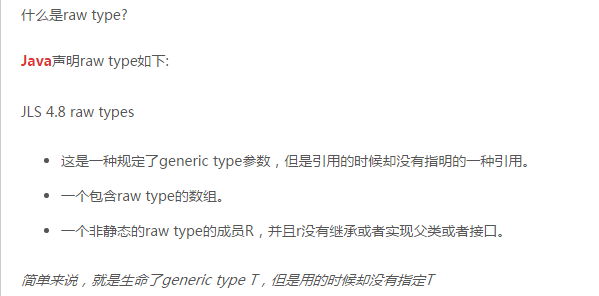
2.信息参数化可以让你写一些更通用的class和trait

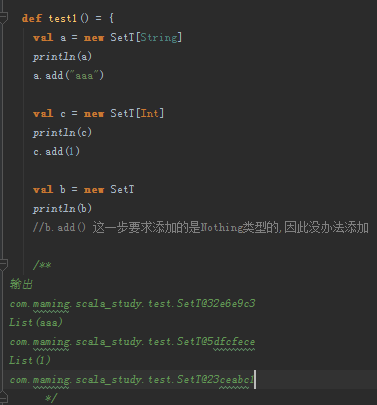
比如定义Set[T]拿到一个类型参数,使得集合更加通用。

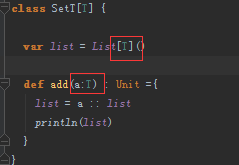
scala不像java,java允许raw types,

即class Box<T>{} Box b = new Box(); //这个Box就是Box<T>的原始类型 。

但是scala要求必须指定参数类型,比如Set[String]







3. Variance:定义了一个参数化类型的继承关系,比如Set[String]是Set[AnyRef]的子类

1. Functional queues 纯粹的一个功能的队列

一.该队列提供三个方法 head、tail、enqueue

Head返回集合的第一个元素

Tail返回除了第一个元素外的其他集合

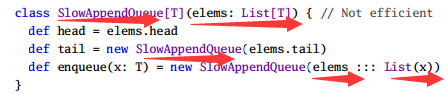
Enqueue是插入一个元素到结尾

二.该队列有点类似list

1.与list不同的是list是在前面追加,queue是在后面追加元素

2.要求queue要在常数时间内完成三个操作.

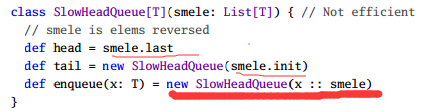
3.使用List去实现



a.可以看到head和tail是常数范围内完成的,因为scala的list是一个链表,因此追加一个很容易,只要把两个连续的对象用指针连接即可.

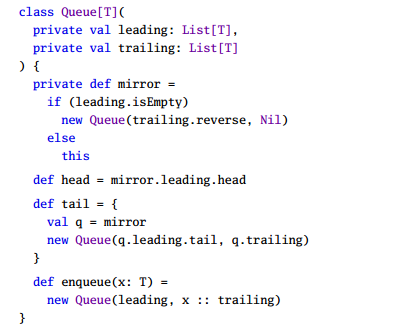
b.enqueue因为是尾部追加,因此是用x在list之后,即表示list在前面,因此是list的最后一个元素要跟x连接,而寻找到list的最后一个元素是需要线性时间的,根据list的元素数量是有关系的。

c.改版



可以看到enqueue是常数范围内完成了,但是head和tail却花费线性时间了。

d.升级版本,使用两个list



a.所有向后追加方法,都是用list的向前追加方法实现的,即向trailing的list追加,因此enqueue就是x在trailing之前设置,符合list的常数时间的性能。

b.每次追加都保持leading队列不变

c.而每次get数据的时候都是从leading获取,而保持trailing队列不变。

d.上面都是保证了常数范围内完成的

e.耗时的是镜像方法.每次当leading队列没有数据的时候,他要花时间将trailing队列的数据倒序插入到leading队列中,然后trailing队列为nil.

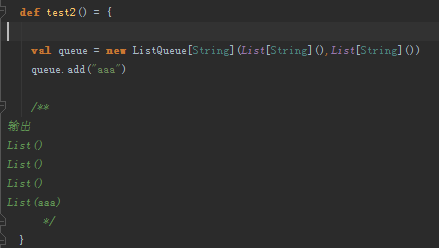
f.每一次都是产生新的Queue对象。

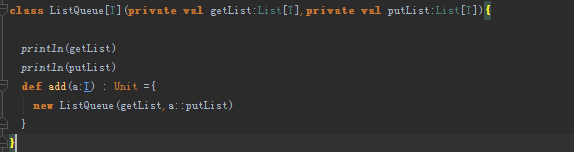
因此如果三个方法的使用频率相同,则效率会很高。

1. Information hiding 信息隐藏

一.上一节讲述的是很好的效率方面实现,但是你可能会反对在效率的得到上,却付出了暴露太多不必要的详细的实现细节。

实现细节是构造队列的时候需要提供两个list参数,其实这个应该是被省略的,客户端代码不应该关注两个list的设置。

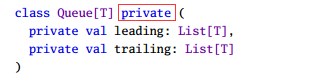




二. Private constructors and factory methods 私有的构造函数和工厂方法

1.java为构造函数提供了private 构造函数.因此外界没办法new 一个Queue,并且提供参数、

而scala让外界隐藏主构造函数中的参数,也是有办法的。

  
即在class的name和参数之间,加入private.

因此

**val** queue = **new** ListQueue[String]()



提示不能访问class

2.如果设置private后不能构造一个对象了,那么如何才能创建一个对象呢

该class依然是public级别的,外面是可以访问的,但是构造函数只能是他的伴随对象或者class自己可以访问.因此可以这么创建对象

a.附加副构造函数

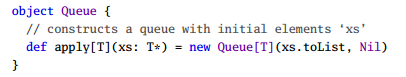


b.或者增加参数也没问题



因为a和b都是自己类本身调用的构造函数,是被允许的

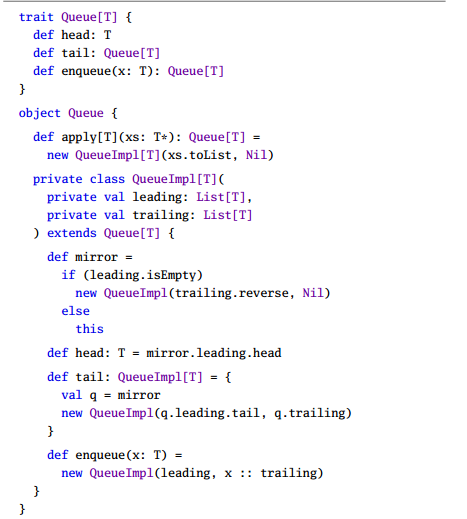
c.伴随对象



因此客户端可以这么使用该对象

三. An alternative: private classes 一个替代方案 私有class

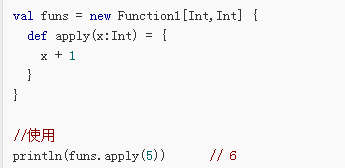
1.提供public的trait,还有一个私有的实现类,也是一个方案



1. Variance annotations 就是讲述三种变化的关系

网上资料

1.function1---23的实现方式



表示两个参数类型组成Function1,表示一个参数是Int,经过运算后返回值还是Int





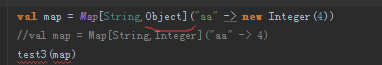
2.map



可以看到key是一个不变的对象,value是一个协变对象

因此





是有问题的,因为map是[String,Object],而需要的是[String,Integer],因此肯定是不合适的





这样是可以的,原因就是Map定义的是协变value,是子类都可以被传递进来

3.总结

a.不变对象、协变对象、逆变对象都是指代的是泛型后的对象之间的关系

b. 不变对象是说无论T是否有关系,泛型后的对象都没有关系,因此不能互相替代。

c. 协变对象,如果两个T之间是有关系的,则泛型后的对象也是有关系的,关系和T之间关系相同,因此可以子类代替父类

d. 逆变对象, 如果两个T之间是有关系的,则泛型后的对象也是有关系的,关系和T之间关系是逆关系,因此可以父类代替子类

1. 如何看待以下几个定义

covariant 协变的,灵活的

nonvariant ,rigid 非协变的,刚性的

contravariant 逆变的

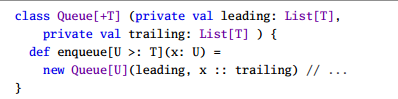
如果一个类型支持协变或逆变，则称这个类型为**variance(翻译为可变的或变型)**，否则称为**invariant(不可变的)**

在Java里，泛型类型都是invariant，比如 List<String> 并不是 List<Object> 的子类型

第三四六七节后续详细看,暂时没太理解

第五节 Lower bounds 添加T的父类

一、重新定义Queue,变成协变,即Queue队列是有父子关系的



1. 定义一个新的队列, enqueue方法有参数类型U,该U是T的父类,

因此enqueue方法使用U返回值就变成U了,即返回的是Queue[U]对象。

2.例如水果作为父类,有两个子类—苹果和橘子。

如果让一个Queue[Apple]队列中enqueue(橘子对象),结果变成Queue[Fruit]

理论上橘子是不能添加到苹果队列的,但是因为橘子也是水果的子类,因此可以理解成添加的是水果.因此就变成水果队列。

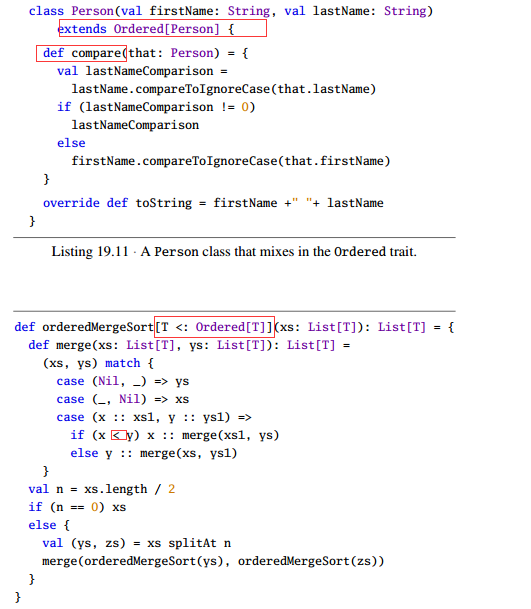
二、Lower bounds和covariance(协变)可以一起搭配使用

可以看到上面的例子就是组合使用。

可以看到Queue[水果] 是 Queue[苹果]的父类

第8节 Upper bounds

一、例子



二、说明

1.定义person的时候,他继承Ordered,即person是可以比较的对象

2.排序方法传入的是一个List,要求List中的T是ordered的子类.即T使可以排序的。

因此后续比较久可以使用> <方式比较。

三、总结

1.和是定义一个类接受泛型T的类型的约束,即使用了这种约束后,可以导致泛型对象不会接受任意对象

2.而协变+T, 逆变-T是针对List[T]和List[+T]、List[-T]结果集之间的关系，

常常用于函数调用.比如有一个函数,参数是List[String],如果我们持有的对象是List[Object]是不能传递到该函数的。

如果List定义得是-T,则因为Object是String的父类,因此List[Object]就是List[String]的子类,因此是可以被传递到函数中的。