# 程序设计

## 去重

def distinct: Repr = {

val b = newBuilder

val seen = mutable.HashSet[A]()

for (x <- this) {

if (!seen(x)) {

b += x

seen += x

}

}

b.result

}

这个程序感觉很有灵感，他是由一个变化的set作为if的守卫条件进行循环的，并且这个set是根据需要守卫的变量变化的。

这也说明编程语言也是一种语言（我认为，用的时候使用的大部分是大脑前额），也有一些语法上的魅力，写一段简洁流畅的代码也是一种审美。

## 多用组合，少用继承

这个也是思想的一种，防止添加功能时整个框架的代码都跟着变。

# 代码程序设计

## Flag = !flag用于奇偶遍历

**以zip形式遍历两个集合**或者**依据情况一次这一次那的操作时**，下列框架不错：

Var flag = true

While(…){

If(flag)

…

Else

…

**Flag = !flag**

}

我将之用于geoHash，代码质量提高了。

## 控制流语句的使用

### 整体的感想

是计算机出身的程序员在控制流使用方面很擅长，数学出身的对高级函数和矩阵运算很擅长。为了弥补不足从今天起开始学习一些常用的控制流语句架构。

### While先加还是后加的问题

While(p < 200){

Arr += p

P +=1

}

While(p < 200){

P +=1

Arr += p

}

### While中i的步长不固定的问题

以线性缺失值补全为例

def fillLinear(values: BV[Double]): BDV[Double] = {

val result = values.copy.toArray

var i = 1

while (i < result.length - 1) {

val rangeStart = i

while (i < result.length - 1 && result(i).isNaN) {

i += 1

}

val before = result(rangeStart - 1)

val after = result(i)

if (i != rangeStart && !before.isNaN && !after.isNaN) {

val increment = (after - before) / (i - (rangeStart - 1))

for (j <- rangeStart until i) {

result(j) = result(j - 1) + increment

}

}

i += 1

}

new BDV[Double](result)

}

不过这种缺失值补全的时间复杂度过高（大约是O(n\*m)），还可以变为O（n）

### Array.distinct源码中对控制语句的使用

def distinct: Repr = {

val b = newBuilder

val seen = mutable.HashSet[A]()

for (x <- this) {

if (!seen(x)) {

b += x

seen += x

}

}

b.result

}

**使用了两个集合A，B，其中A参与了if守卫控制条件**