## 算法

### LRU

我们可以参考<https://blog.csdn.net/crazyhuntsman/article/details/81981014>

写的不错。

### 排序

<https://www.cnblogs.com/onepixel/articles/7674659.html>

## 设计模式

### Factory

### Builder

### Singleton

### Clone

-- 浅克隆， 深克隆

一个类实现了Cloneable接口，就可以实现克隆了。

但是对于引用属性，还需要将这个引用属性也实现Cloneable接口，才能实现深克隆。

### Adapter

我的理解： 一个controller的需求是实验一个功能C，而这个功能可以利用serviceM的A和B函数完成，我们可以在service中写一个C，把A和B穿起来，但是如果在该类中需要多种像C这样的情况，那么都写到一个service中就显得有些多余了，因此，可以把函数C这类的函数全部提取到一个类中，命名为adapter，让controller调用即可。

### Façade

我的理解：

一个controller的需求是实现一个功能C，但是这个功能需要多个不同的serviceM，serviceN，serviceL来组合完成，那么我们就可以写一个类，把这些函数串起来实现功能C，提供给controller调用，简化controller的代码量，使它可以专注于参数校验等逻辑中，这个类，我们叫façade。

### 组合模式

### 装饰着模式

Wrapper就是利用的这个模式，实现我们想要的对某个函数的小处理。

情景可以参考response的wrapper使用情景。

可能你会问，那么我直接修改response不就可以了吗？呵呵，那如果我既需要原先的response，又需要wapper的内容呢？知道了吧。

## 用接口而不是用实现类编程的好处

## 数据结构

参考文章：<https://www.cnblogs.com/skywang12345/p/3245399.html>

**红黑树的特性**:  
**（1）每个节点或者是黑色，或者是红色。**  
**（2）根节点是黑色。**  
**（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]**  
**（4）如果一个节点是红色的，则它的子节点必须是黑色的。**  
**（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。**

**注意**：  
(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。  
(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

**1. 左旋**



对x进行左旋，意味着"将x变成一个左节点"。

左旋的伪代码《算法导论》：参考上面的示意图和下面的伪代码，理解“红黑树T的节点x进行左旋”是如何进行的。

LEFT-ROTATE(T, x)

01 y ← right[x] // 前提：这里假设x的右孩子为y。下面开始正式操作

02 right[x] ← left[y] // 将 “y的左孩子” 设为 “x的右孩子”，即 将β设为x的右孩子

03 p[left[y]] ← x // 将 “x” 设为 “y的左孩子的父亲”，即 将β的父亲设为x

04 p[y] ← p[x] // 将 “x的父亲” 设为 “y的父亲”

05 if p[x] = nil[T]

06 then root[T] ← y // 情况1：如果 “x的父亲” 是空节点，则将y设为根节点

07 else if x = left[p[x]]

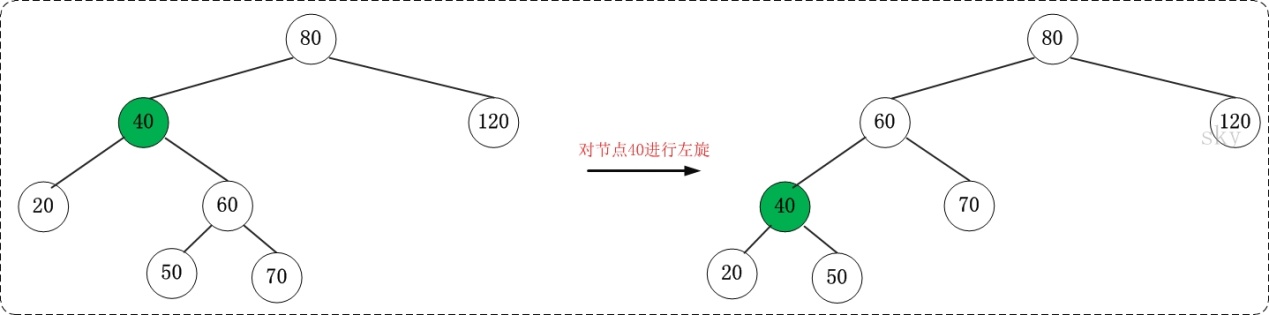
08 then left[p[x]] ← y // 情况2：如果 x是它父节点的左孩子，则将y设为“x的父节点的左孩子”

09 else right[p[x]] ← y // 情况3：(x是它父节点的右孩子) 将y设为“x的父节点的右孩子”

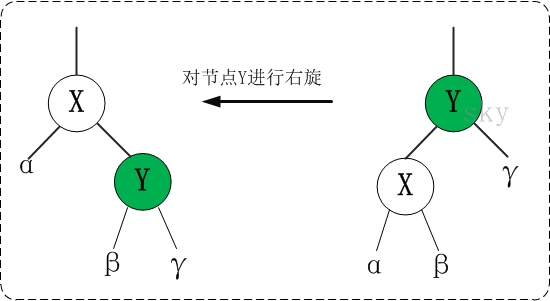
10 left[y] ← x // 将 “x” 设为 “y的左孩子”

11 p[x] ← y // 将 “x的父节点” 设为 “y”

理解左旋之后，看看下面一个更鲜明的例子。你可以先不看右边的结果，自己尝试一下。



**2. 右旋**



对x进行左旋，意味着"将x变成一个左节点"。

右旋的伪代码《算法导论》：参考上面的示意图和下面的伪代码，理解“红黑树T的节点y进行右旋”是如何进行的。

RIGHT-ROTATE(T, y)

01 x ← left[y] // 前提：这里假设y的左孩子为x。下面开始正式操作

02 left[y] ← right[x] // 将 “x的右孩子” 设为 “y的左孩子”，即 将β设为y的左孩子

03 p[right[x]] ← y // 将 “y” 设为 “x的右孩子的父亲”，即 将β的父亲设为y

04 p[x] ← p[y] // 将 “y的父亲” 设为 “x的父亲”

05 if p[y] = nil[T]

06 then root[T] ← x // 情况1：如果 “y的父亲” 是空节点，则将x设为根节点

07 else if y = right[p[y]]

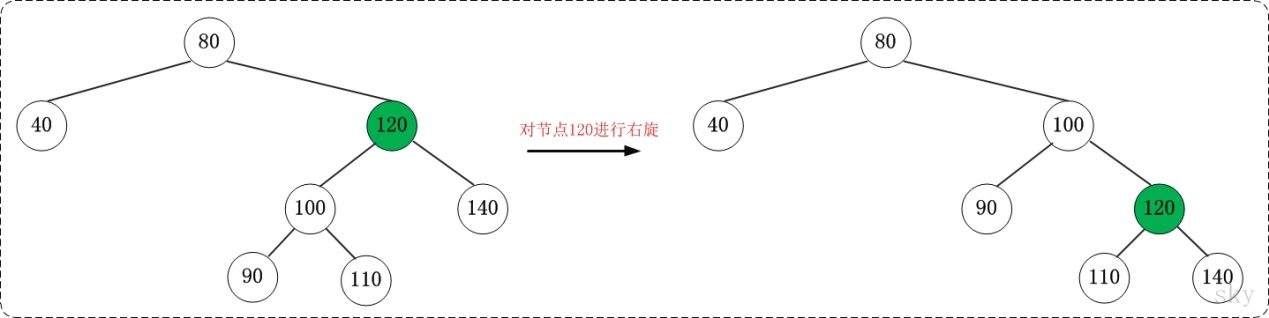
08 then right[p[y]] ← x // 情况2：如果 y是它父节点的右孩子，则将x设为“y的父节点的左孩子”

09 else left[p[y]] ← x // 情况3：(y是它父节点的左孩子) 将x设为“y的父节点的左孩子”

10 right[x] ← y // 将 “y” 设为 “x的右孩子”

11 p[y] ← x // 将 “y的父节点” 设为 “x”

理解右旋之后，看看下面一个更鲜明的例子。你可以先不看右边的结果，自己尝试一下。



**旋转总结**：

(01) 左旋 和 右旋 是相对的两个概念，原理类似。理解一个也就理解了另一个。

(02) 下面谈谈如何区分 左旋 和 右旋。  
在实际应用中，若没有彻底理解 左旋 和 右旋，可能会将它们混淆。下面谈谈我对如何区分 左旋 和 右旋 的理解。