# 1.Set集合

## 1.1Set集合概述和特点【应用】

- 不可以存储重复元素
- 没有索引,不能使用普通for循环遍历

## 1.2Set集合的使用【应用】

存储字符串并遍历

```
public class MySet1 {
    public static void main(String[] args) {
        //创建集合对象
        Set<String> set = new TreeSet<>();
        //添加元素
        set.add("ccc");
        set.add("aaa");
        set.add("aaa");
        set.add("bbb");

//        for (int i = 0; i < set.size(); i++) {
//            //Set集合是没有索引的，所以不能使用通过索引获取元素的方法
//        }

        //遍历集合
        Iterator<String> it = set.iterator();
        while (it.hasNext()){
            String s = it.next();
            System.out.println(s);
        }
        System.out.println("----------------------------------");
        for (String s : set) {
            System.out.println(s);
        }
    }
}
```

# 2.TreeSet集合

## 2.1TreeSet集合概述和特点【应用】

- 不可以存储重复元素
- 没有索引
- 可以将元素按照规则进行排序
    - TreeSet()：根据其元素的自然排序进行排序
    - TreeSet(Comparator comparator)： 根据指定的比较器进行排序

## 2.2TreeSet集合基本使用【应用】

存储Integer类型的整数并遍历

```
public class TreeSetDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Integer> ts = new TreeSet<Integer>();

        //添加元素
        ts.add(10);
        ts.add(40);
        ts.add(30);
        ts.add(50);
        ts.add(20);

        ts.add(30);

        //遍历集合
        for(Integer i : ts) {
            System.out.println(i);
        }
    }
}
```

## 2.3自然排序Comparable的使用【应用】

- 案例需求
  - 存储学生对象并遍历，创建TreeSet集合使用无参构造方法
  - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序
- 实现步骤
  1. 使用空参构造创建TreeSet集合
     - 用TreeSet集合存储自定义对象，无参构造方法使用的是自然排序对元素进行排序的
  2. 自定义的Student类实现Comparable接口
     - 自然排序，就是让元素所属的类实现Comparable接口，重写compareTo(T o)方法
  3. 重写接口中的compareTo方法
     - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写
- 代码实现

  学生类

```
public class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student() {
    }
```

```
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Student o) {
        //按照对象的年龄进行排序
        //主要判断条件：按照年龄从小到大排序
        int result = this.age - o.age;
        //次要判断条件：年龄相同时，按照姓名的字母顺序排序
        result = result == 0 ? this.name.compareTo(o.getName()) : result;
        return result;
    }
}
```

测试类

```
public class MyTreeSet2 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Student> ts = new TreeSet<>();
        //创建学生对象
        Student s1 = new Student("zhangsan",28);
        Student s2 = new Student("lisi",27);
        Student s3 = new Student("wangwu",29);
        Student s4 = new Student("zhaoliu",28);
        Student s5 = new Student("qianqi",30);
        //把学生添加到集合
```

```
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        //遍历集合
        for (Student student : ts) {
            System.out.println(student);
        }
    }
}
```

## 2.4比较器排序Comparator的使用【应用】

- 案例需求
  - 存储老师对象并遍历，创建TreeSet集合使用带参构造方法
  - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序
- 实现步骤
  - 用TreeSet集合存储自定义对象，带参构造方法使用的是比较器排序对元素进行排序的
  - 比较器排序，就是让集合构造方法接收Comparator的实现类对象，重写compare(T o1,T o2)方法
  - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写
- 代码实现

  老师类

```
public class Teacher {
    private String name;
    private int age;

    public Teacher() {
    }

    public Teacher(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
```

```
    }

    @Override
    public String toString() {
        return "Teacher{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

测试类

```
public class MyTreeSet4 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Teacher> ts = new TreeSet<>(new Comparator<Teacher>() {
            @Override
            public int compare(Teacher o1, Teacher o2) {
                //o1表示现在要存入的那个元素
                //o2表示已经存入到集合中的元素

                //主要条件
                int result = o1.getAge() - o2.getAge();
                //次要条件
                result = result == 0 ? o1.getName().compareTo(o2.getName()) : result;
                return result;
            }
        });
         //创建老师对象
        Teacher t1 = new Teacher("zhangsan",23);
        Teacher t2 = new Teacher("lisi",22);
        Teacher t3 = new Teacher("wangwu",24);
        Teacher t4 = new Teacher("zhaoliu",24);
         //把老师添加到集合
        ts.add(t1);
        ts.add(t2);
        ts.add(t3);
        ts.add(t4);
         //遍历集合
        for (Teacher teacher : ts) {
            System.out.println(teacher);
        }
    }
}
```

## 2.4两种比较方式总结【理解】

- 两种比较方式小结
  - 自然排序: 自定义类实现Comparable接口,重写compareTo方法,根据返回值进行排序
  - 比较器排序: 创建TreeSet对象的时候传递Comparator的实现类对象,重写compare方法,根据返回值进行排序

- 在使用的时候,默认使用自然排序,当自然排序不满足现在的需求时,必须使用比较器排序

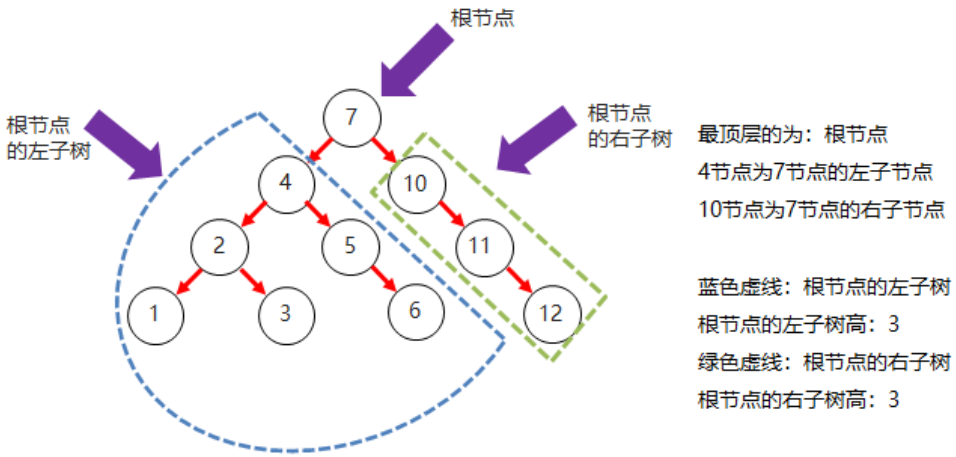
- 两种方式中关于返回值的规则
  - 如果返回值为负数，表示当前存入的元素是较小值，存左边
  - 如果返回值为0，表示当前存入的元素跟集合中元素重复了，不存
  - 如果返回值为正数，表示当前存入的元素是较大值，存右边

# 3.数据结构

## 3.1二叉树【理解】

- 二叉树的特点
  - 二叉树中,任意一个节点的度要小于等于2
    - 节点: 在树结构中,每一个元素称之为节点
    - 度: 每一个节点的子节点数量称之为度

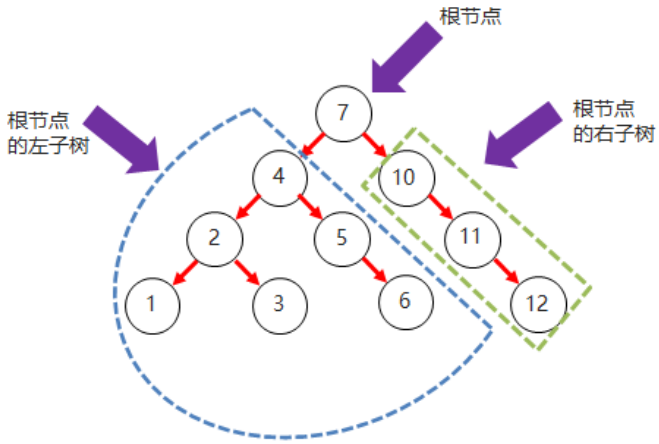
- 二叉树结构图




## 3.2二叉查找树【理解】

- 二叉查找树的特点
  - 二叉查找树,又称二叉排序树或者二叉搜索树
  - 每一个节点上最多有两个子节点
  - 左子树上所有节点的值都小于根节点的值
  - 右子树上所有节点的值都大于根节点的值

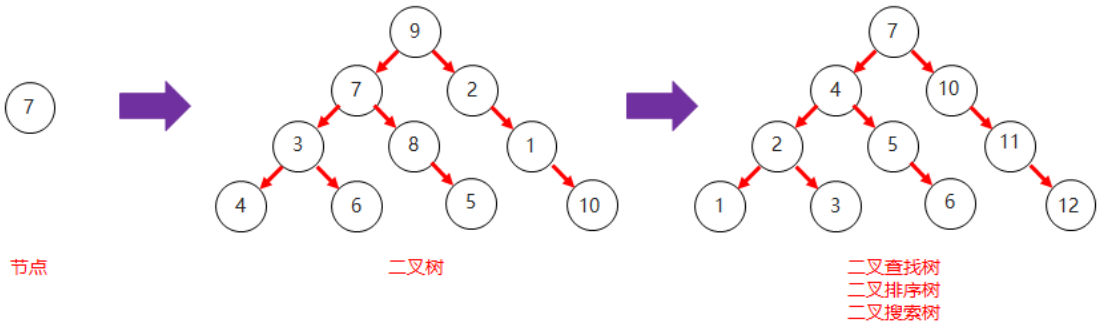
- 二叉查找树结构图



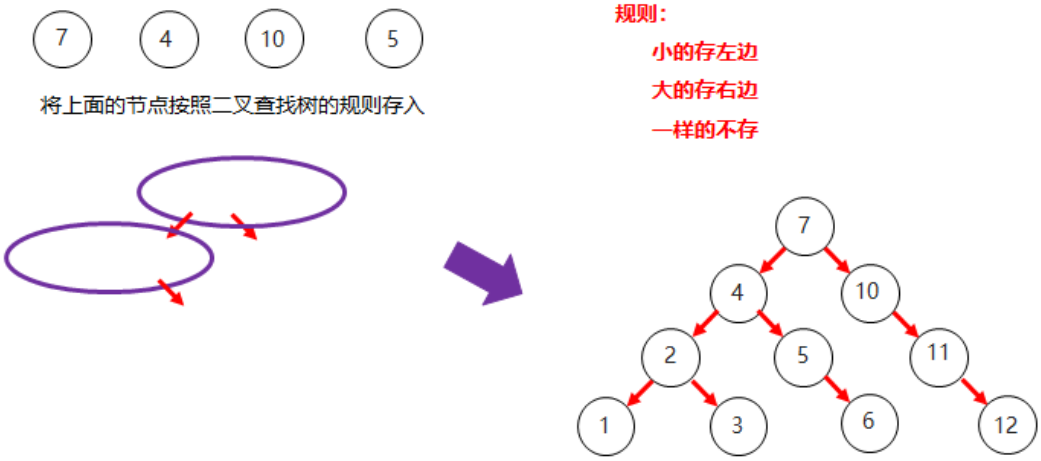

整个树高：4
最顶层的为：根节点
蓝色虚线：根节点的左子树（左孩子）
根节点的左子树高：3
绿色虚线：根节点的右子树（右孩子）
根节点的右子树高：3

这颗树中，根节点左边的都比根节点小
根节点右边的都比根节点大，所以是二叉查找树。

- 二叉查找树和二叉树对比结构图

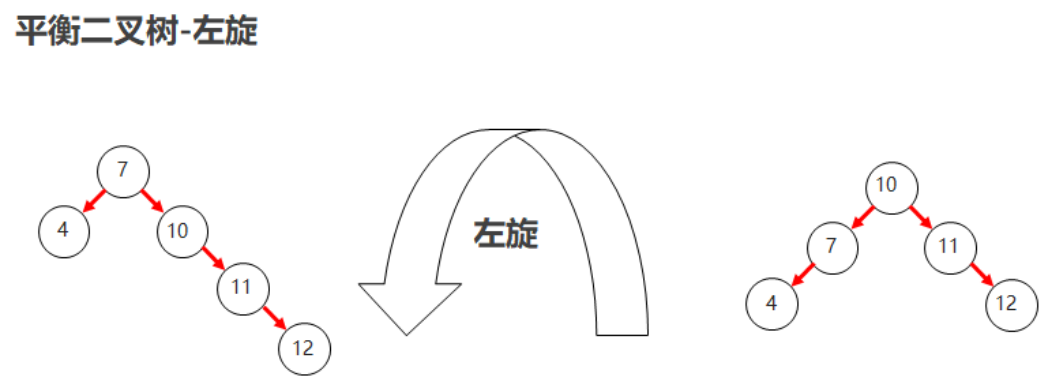



- 二叉查找树添加节点规则
  - 小的存左边
  - 大的存右边
  - 一样的不存



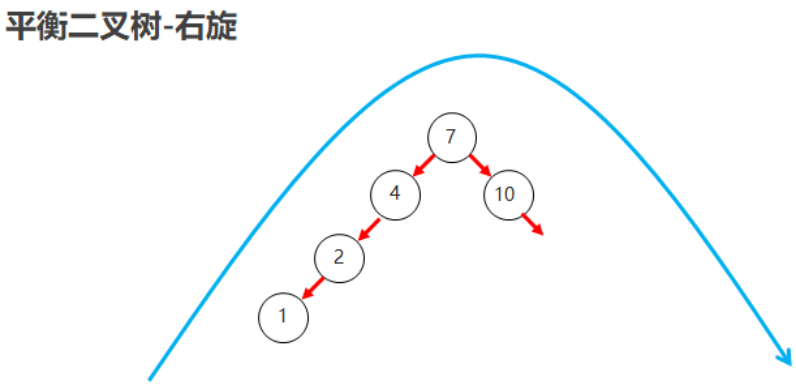

## 3.3平衡二叉树【理解】

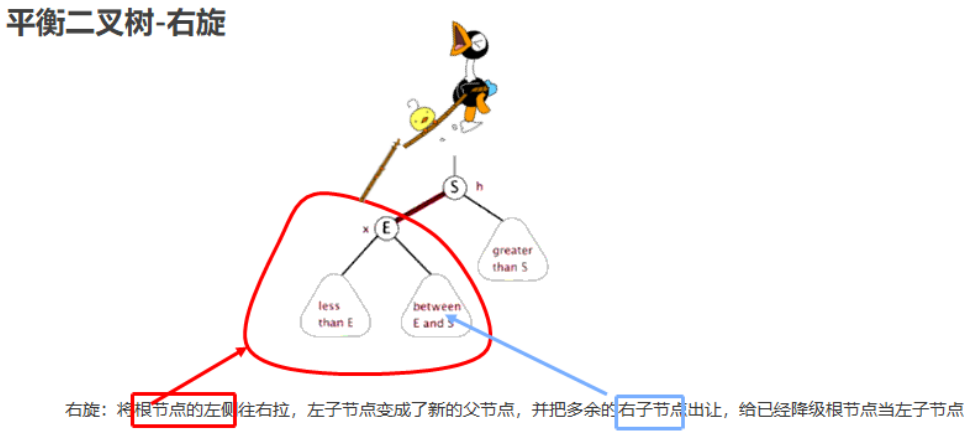
- 平衡二叉树的特点
  - 二叉树左右两个子树的高度差不超过1
  - 任意节点的左右两个子树都是一颗平衡二叉树
- 平衡二叉树旋转
  - 旋转触发时机
    - 当添加一个节点之后,该树不再是一颗平衡二叉树
  - 左旋
    - 就是将根节点的右侧往左拉,原先的右子节点变成新的父节点,并把多余的左子节点出让,给已经降级的根节点当右子节点



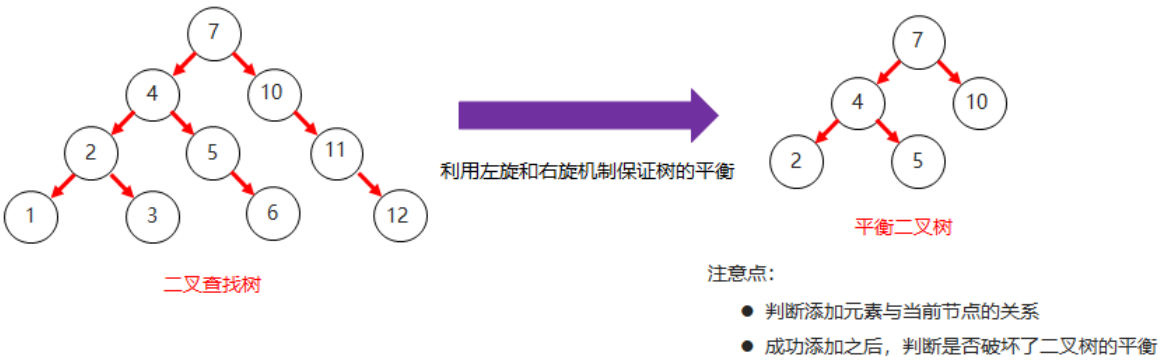


  - 右旋
    - 就是将根节点的左侧往右拉,左子节点变成了新的父节点,并把多余的右子节点出让,给已经降级根节点当左子节点

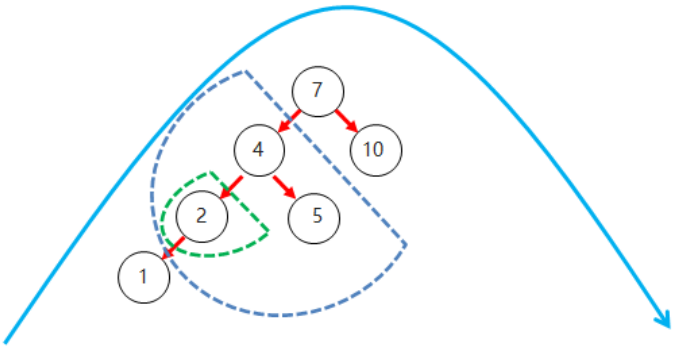


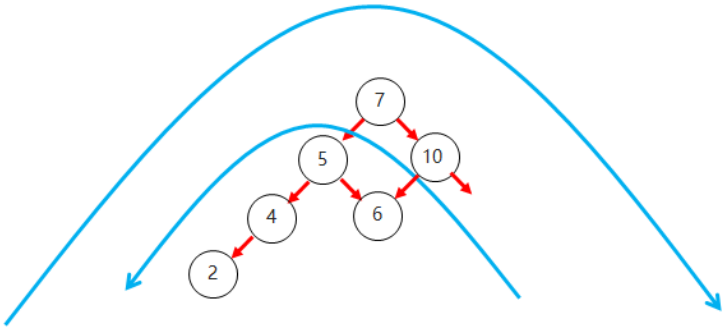

- 平衡二叉树和二叉查找树对比结构图



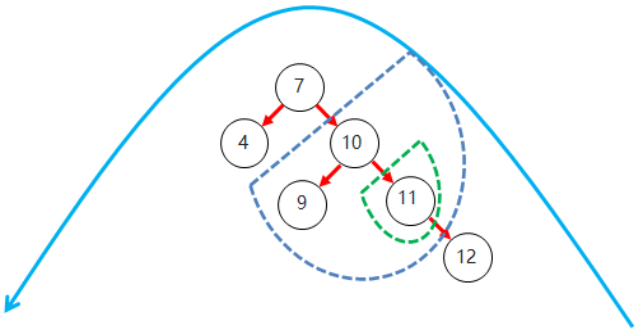

- 平衡二叉树旋转的四种情况
  - 左左
    - 左左: 当根节点左子树的左子树有节点插入,导致二叉树不平衡
    - 如何旋转: 直接对整体进行右旋即可



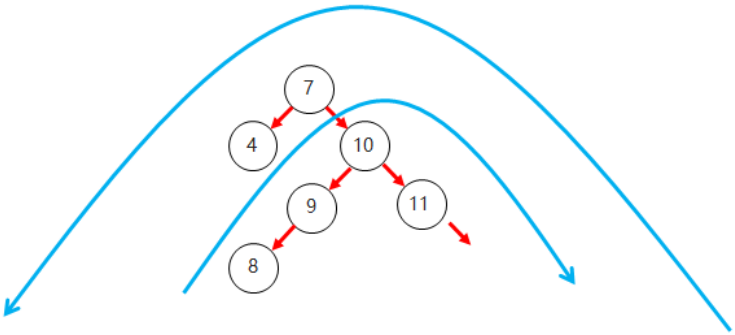

- 左右
  - 左右: 当根节点左子树的右子树有节点插入,导致二叉树不平衡
  - 如何旋转: 先在左子树对应的节点位置进行左旋,在对整体进行右旋




- 右右
  - 右右: 当根节点右子树的右子树有节点插入,导致二叉树不平衡
  - 如何旋转: 直接对整体进行左旋即可

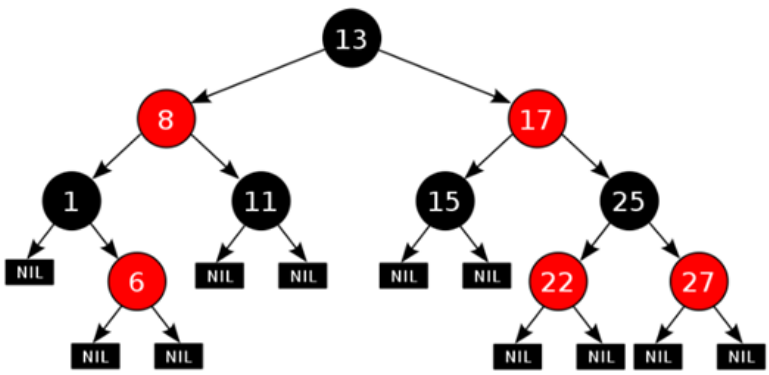



- 右左
  - 右左:当根节点右子树的左子树有节点插入,导致二叉树不平衡
  - 如何旋转: 先在右子树对应的节点位置进行右旋,在对整体进行左旋

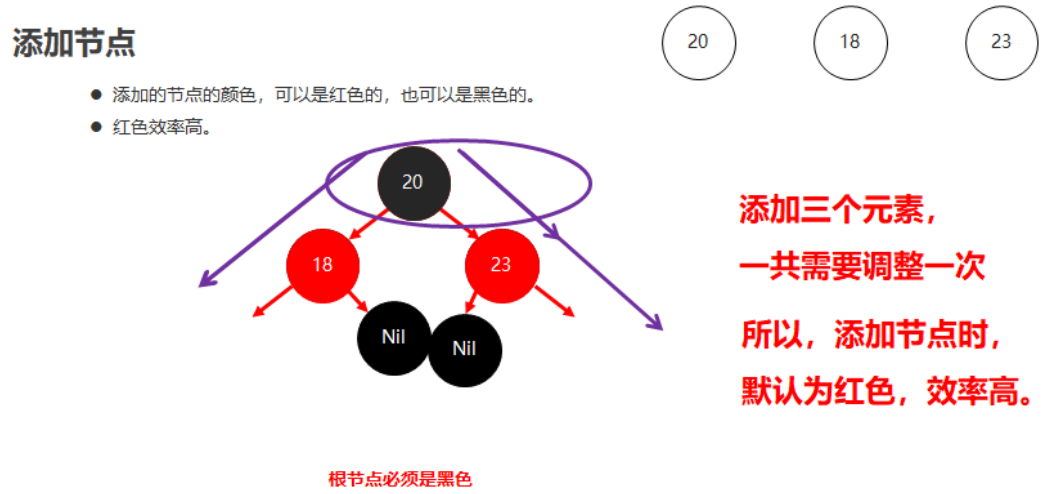



## 3.4红黑树【理解】

- 红黑树的特点
  - 平衡二叉B树
  - 每一个节点可以是红或者黑
  - 红黑树不是高度平衡的,它的平衡是通过"自己的红黑规则"进行实现的
- 红黑树的红黑规则有哪些
  1. 每一个节点或是红色的,或者是黑色的
  2. 根节点必须是黑色
  3. 如果一个节点没有子节点或者父节点,则该节点相应的指针属性值为Nil,这些Nil视为叶节点,每个叶节点(Nil)是黑色的
  4. 如果某一个节点是红色,那么它的子节点必须是黑色(不能出现两个红色节点相连 的情况)
  5. 对每一个节点,从该节点到其所有后代叶节点的简单路径上,均包含相同数目的黑色节点




- 红黑树添加节点的默认颜色
  - 添加节点时,默认为红色,效率高




- 红黑树添加节点后如何保持红黑规则
  - 根节点位置
    - 直接变为黑色
  - 非根节点位置
    - 父节点为黑色
      - 不需要任何操作,默认红色即可
    - 父节点为红色
      - 叔叔节点为红色
        1. 将"父节点"设为黑色,将"叔叔节点"设为黑色
        2. 将"祖父节点"设为红色
        3. 如果"祖父节点"为根节点,则将根节点再次变成黑色
      - 叔叔节点为黑色
        1. 将"父节点"设为黑色
        2. 将"祖父节点"设为红色
        3. 以"祖父节点"为支点进行旋转

## 3.5成绩排序案例【应用】

- 案例需求
  - 用TreeSet集合存储多个学生信息(姓名,语文成绩,数学成绩,英语成绩),并遍历该集合
  - 要求: 按照总分从高到低出现
- 代码实现

  学生类

```
public class Student implements Comparable<Student> {
    private String name;
    private int chinese;
    private int math;
    private int english;


    public Student() {
```

```
    }

    public Student(String name, int chinese, int math, int english) {
        this.name = name;
        this.chinese = chinese;
        this.math = math;
        this.english = english;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getChinese() {
        return chinese;
    }

    public void setChinese(int chinese) {
        this.chinese = chinese;
    }

    public int getMath() {
        return math;
    }

    public void setMath(int math) {
        this.math = math;
    }

    public int getEnglish() {
        return english;
    }

    public void setEnglish(int english) {
        this.english = english;
    }

    public int getSum() {
        return this.chinese + this.math + this.english;
    }

    @Override
    public int compareTo(Student o) {
        // 主要条件：按照总分进行排序
        int result = o.getSum() - this.getSum();
        // 次要条件：如果总分一样,就按照语文成绩排序
        result = result == 0 ? o.getChinese() - this.getChinese() : result;
        // 如果语文成绩也一样,就按照数学成绩排序

        result = result == 0 ? o.getMath() - this.getMath() : result;
```

```
        // 如果总分一样,各科成绩也都一样,就按照姓名排序
        result = result == 0 ? o.getName().compareTo(this.getName()) : result;
        return result;
    }
}
```

测试类

```
public class TreeSetDemo {
    public static void main(String[] args) {
        //创建TreeSet集合对象，通过比较器排序进行排序
        TreeSet<Student> ts = new TreeSet<Student>();
        //创建学生对象
        Student s1 = new Student("jack", 98, 100, 95);
        Student s2 = new Student("rose", 95, 95, 95);
        Student s3 = new Student("sam", 100, 93, 98);
        //把学生对象添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);

        //遍历集合
        for (Student s : ts) {
            System.out.println(s.getName() + "," + s.getChinese() + "," + s.getMath() + ","
+ s.getEnglish() + "," + s.getSum());
        }
    }
}
```

# 4.HashSet集合

## 4.1HashSet集合概述和特点【应用】

- 底层数据结构是哈希表
- 存取无序
- 不可以存储重复元素
- 没有索引,不能使用普通for循环遍历

## 4.2HashSet集合的基本应用【应用】

存储字符串并遍历

```
public class HashSetDemo {
    public static void main(String[] args) {
        //创建集合对象
        HashSet<String> set = new HashSet<String>();

        //添加元素
        set.add("hello");
        set.add("world");
        set.add("java");
```

```
        //不包含重复元素的集合
        set.add("world");

        //遍历
        for(String s : set) {
            System.out.println(s);
        }
    }
}
```

## 4.3哈希值【理解】

- 哈希值简介

  是JDK根据对象的地址或者字符串或者数字算出来的int类型的数值

- 如何获取哈希值

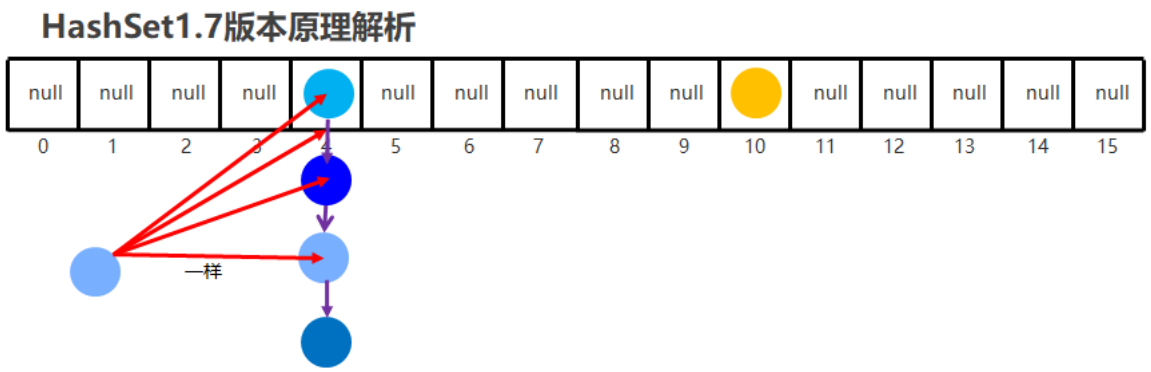
  Object类中的public int hashCode()：返回对象的哈希码值

- 哈希值的特点
  - 同一个对象多次调用hashCode()方法返回的哈希值是相同的
  - 默认情况下，不同对象的哈希值是不同的。而重写hashCode()方法，可以实现让不同对象的哈希值相同

## 4.4哈希表结构【理解】

- JDK1.8以前

  数组 + 链表

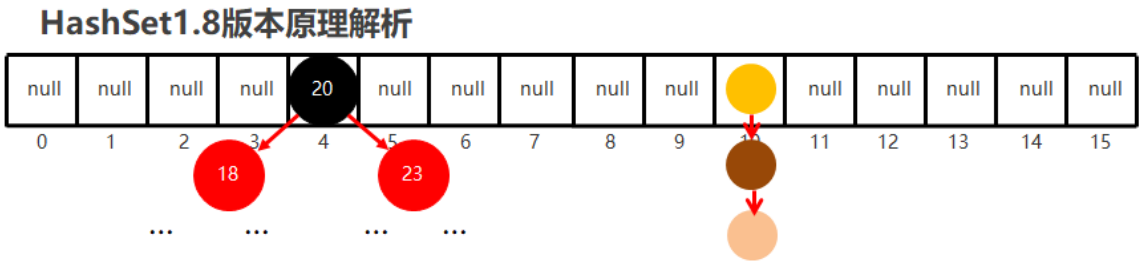



1. 创建一个默认长度16，默认加载因为0.75的数组，数组名table
2. 根据元素的哈希值跟数组的长度计算出应存入的位置
3. 判断当前位置是否为null，如果是null直接存入
4. 如果位置不为null，表示有元素，则调用equals方法比较属性值
5. 如果一样，则不存，如果不一样，则存入数组，老元素挂在新元素下面

- JDK1.8以后
  - 节点个数少于等于8个

    数组 + 链表
  - 节点个数多于8个

    数组 + 红黑树




1. 创建一个默认长度16，默认加载因为0.75的数组，数组名table
2. 根据元素的哈希值跟数组的长度计算出应存入的位置
3. 判断当前位置是否为null，如果是null直接存入
4. 如果位置不为null，表示有元素，
   则调用equals方法比较属性值
5. 如果一样，则不存，如果不一样，则存入数组，老元素挂在新元素下面

## 4.5HashSet集合存储学生对象并遍历【应用】

- 案例需求
  - 创建一个存储学生对象的集合，存储多个学生对象，使用程序实现在控制台遍历该集合
  - 要求：学生对象的成员变量值相同，我们就认为是同一个对象
- 代码实现

  学生类

```
public class Student {
    private String name;
    private int age;

    public Student() {
    }

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Student student = (Student) o;

        if (age != student.age) return false;
        return name != null ? name.equals(student.name) : student.name == null;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

测试类

```
public class HashSetDemo02 {
    public static void main(String[] args) {
        //创建HashSet集合对象
        HashSet<Student> hs = new HashSet<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        Student s4 = new Student("王祖贤", 33);

        //把学生添加到集合
        hs.add(s1);
        hs.add(s2);
        hs.add(s3);
        hs.add(s4);

        //遍历集合(增强for)
        for (Student s : hs) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

- 总结

  HashSet集合存储自定义类型元素,要想实现元素的唯一,要求必须重写hashCode方法和equals方法