# 数据结构

## 一、学习目标

**了解** 树形结构

**了解** 二叉树结构及特点

**了解** 二叉查找树及特点

**了解** 平衡二叉树及特点

**理解** 红黑树结构及特点

**了解** 哈希值特点

**掌握** hashCode方法

**理解** 哈希表结构之数组+链表

**理解** 哈希表结构之数组+红黑树

**了解** HashMap的扩容机制

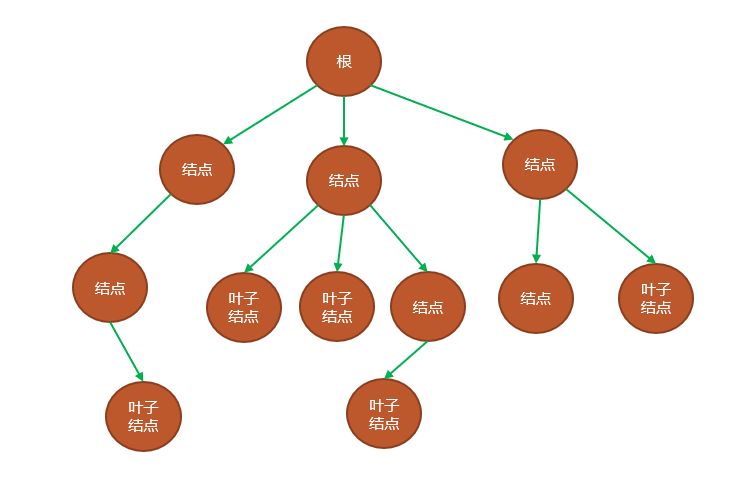
## 二、学习目标

### 1、数据结构

### 1.1 树形结构概述

* 树形结构指的是数据元素之间存在着“一对多”的树形关系的数据结构，是一类重要的非线性数据结构。
* 在树形结构中，树根结点没有前驱结点，其余每个结点有且只有一个前驱结点。叶子结点没有后续结点，其余每个结点的后续节点数可以是一个也可以是多个。
* 树形结构可表示层次关系。
* 树形结构在其他许多方面也有应用。可表示从属关系、并列关系。

树形结构图：



#### 1.2 二叉树

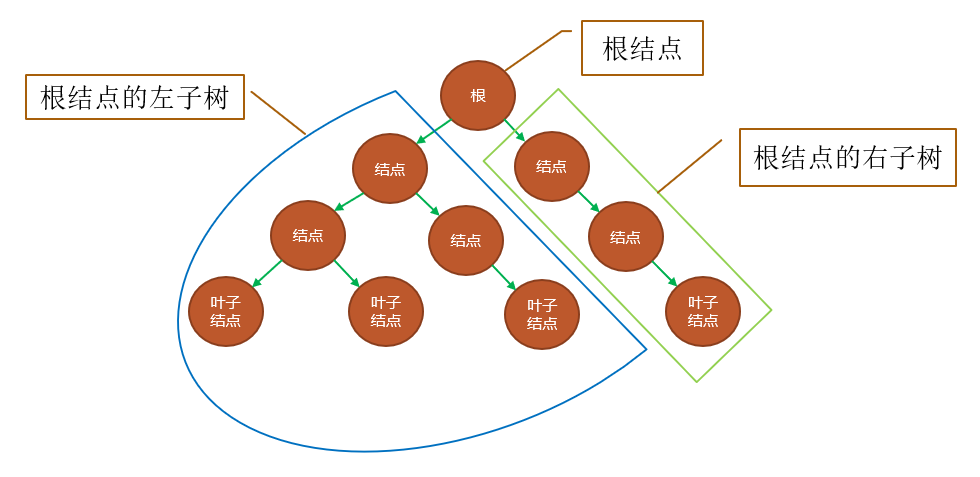
##### 1.2.1 二叉树概述

二叉树是树形结构的一个重要类型。许多实际问题抽象出来的数据结构往往是二叉树形式，即使是一般的树也能简单地转换为二叉树，而且二叉树的存储结构及其算法都较为简单，因此二叉树显得特别重要。

##### 1.2.1 二叉树特点

* 二叉树中,任意一个结点的度要小于等于2，即二叉树每个结点最多只能有两颗子树，且有左右之分
* 结点: 在树结构中,每一个元素称之为结点
* 度: 每一个结点的子结点数量称之为度

##### 1.2.2 二叉树结构图



#### 1.3 二叉查找树

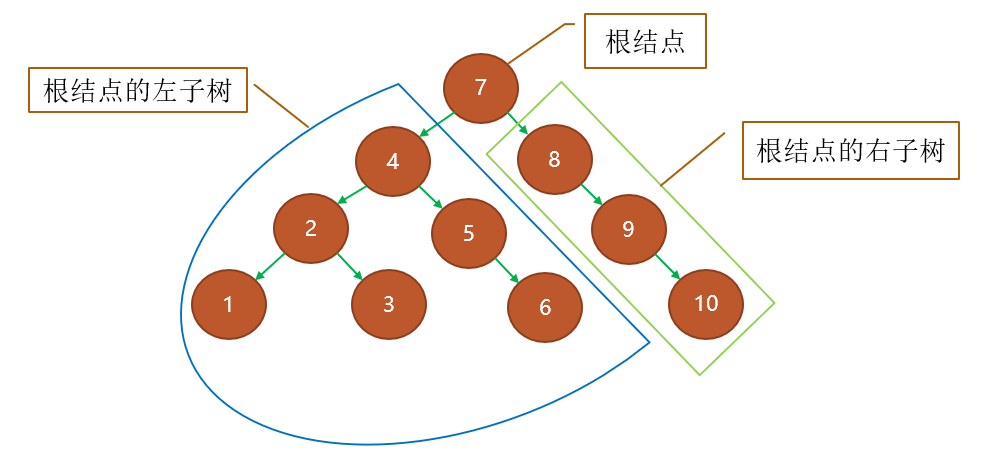
##### 1.3.1 二叉查找树概述

二叉查找树又称为二叉排序数或二叉搜索树，是具有一定顺序的二叉树

##### 1.3.2 二叉查找树特点

* 具有二叉树的所有特点
* 左子树上所有结点的值都小于根结点的值
* 右子树上所有结点的值都大于根结点的值

##### 1.3.3 二叉查找树结构图



#### 1.4 平衡二叉树

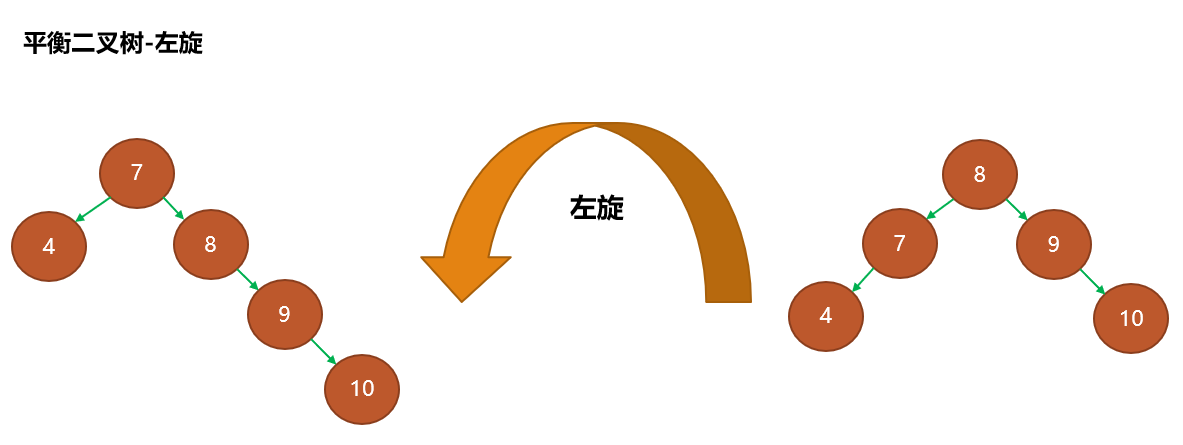
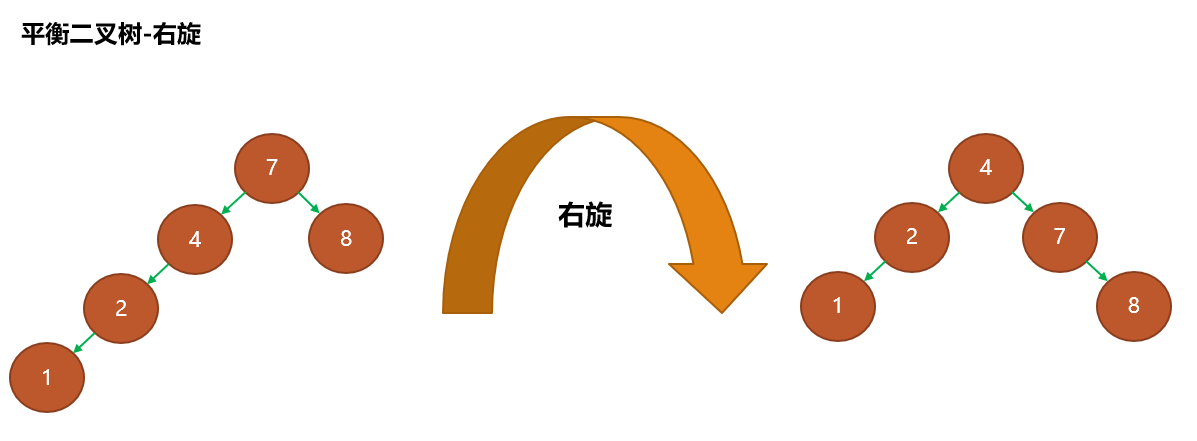
##### 1.4.1 平衡二叉树概述

平衡二叉树指的是任意结点的左右子树的高度差都小于等于1的二叉查找树

##### 1.4.2 平衡二叉树特点

* 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

##### 1.4.3 平衡二叉树的旋转机制

* 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，就会触发旋转机制
* 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左子节点
* 
* 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右子节点
* 

#### 1.5 红黑树

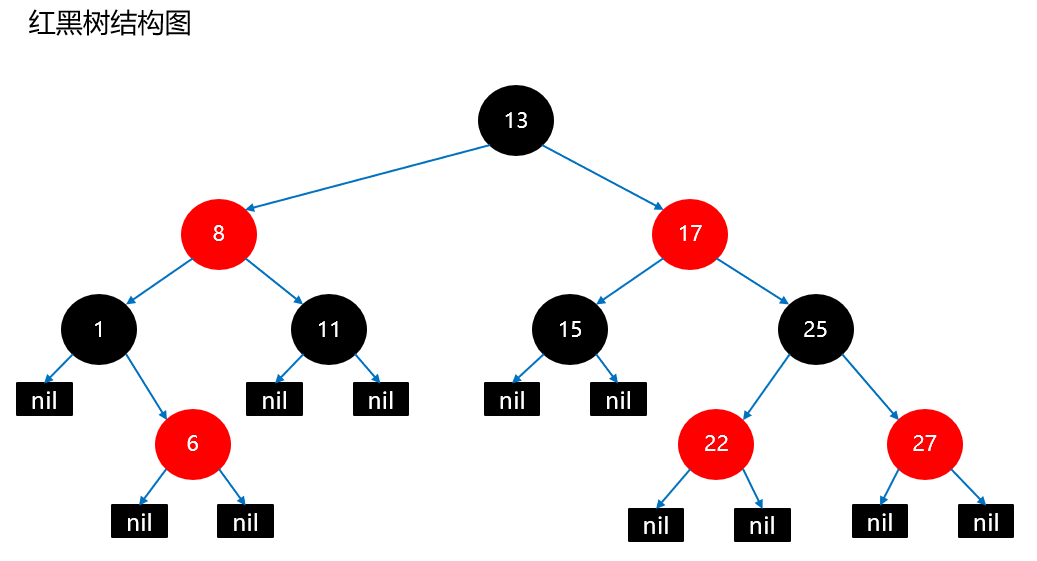
##### 1.5.1 红黑树概述

红黑树是一种自平衡二叉查找树，又叫平衡二叉B树，都是在进行插入和删除操作时通过特定操作保持二叉查找树的平衡，从而获得较高的查找性能。

##### 1.5.2 红黑树特点

1. 每一个结点或是红色的,或者是黑色的
2. 根结点必须是黑色
3. 所有叶子结点都是黑色（叶子是NIL结点）
4. 每个红色结点的两个子结点都是黑色的(不能出现两个红色节点相连的情况)
5. 从任一结点到其每个叶子结点的所有路径都包含相同数目的黑色结点

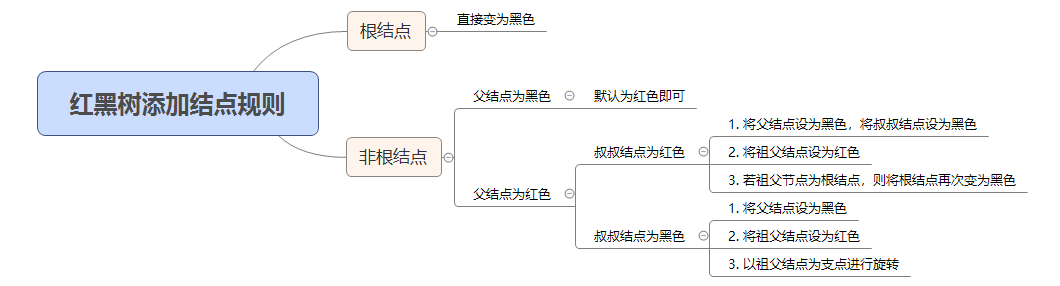
##### 1.5.3 红黑树结构图



##### 1.5.4 红黑树添加结点规则

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



#### 1.6 小结

二叉树的特点

* 二叉树上的每个结点都最多只有两个子结点，并且有左右之分

二叉查找树特点

* 具有一定顺序的二叉树

平衡二叉树特点

* 任意一个结点的左右子树高度差不超过1的二叉查找树

红黑树特点

* 具有红黑规则的平衡二叉树

### 2、哈希值

#### 2.1 哈希值概述

* 哈希值是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
* 获取哈希值：Object类中的public int hashCode()：返回对象的哈希码值
* 哈希表是基于数组的，编程实现相对容易

#### 2.2 哈希值特点

* 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
* 哈希表优点：提供快速的插入和查找的操作
* 哈希表缺点：一旦创建就不好扩展

#### 2.3 hashCode方法

hashCode方法是java中用于获取对象的哈希值的方法。不同的对象哈希值是不同的

代码示例

Student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;  
 }  
}  
  
测试类：  
public class TestHashCode {  
 public static void main(String[] args) {  
 //创建对象，在内存中的地址是不一样的，是不同的对象  
 Student s1 = new Student("张三",18);  
 Student s2 = new Student("张三",18);  
 // 所以hashCode值是不同的  
 System.out.println(s1.hashCode()); //1060830840  
 System.out.println(s2.hashCode()); //2137211482  
 }  
}

通常情况下，对象的属性值一样，会默认为同一个对象，我们可以通过重写hashCode方法实现让不同对象的哈希值相同

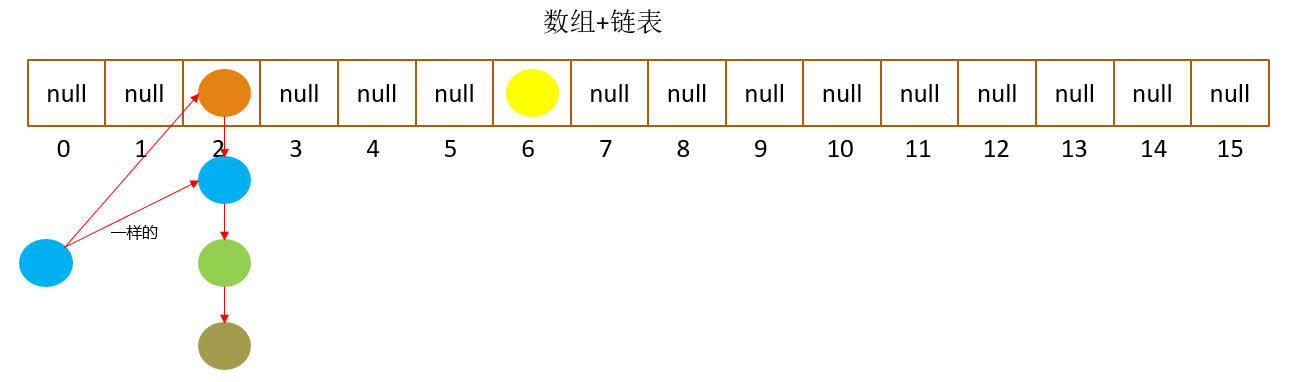
代码示例

Student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 int hashCode() {  
 int result = name != null ? name.hashCode() : 0;  
 result = 31 \* result + age;  
 return result;  
 }  
}  
  
测试类：  
public class TestHashCode {  
 public static void main(String[] args) {  
 //创建对象，在内存中的地址是不一样的，是不同的对象  
 Student s1 = new Student("张三",18);  
 Student s2 = new Student("张三",18);  
 // 重写hashCode方法之后， 获取hashCode值一样  
 System.out.println(s1.hashCode()); //24021577  
 System.out.println(s2.hashCode()); //24021577  
 }  
}

#### 2.4 HashSet的哈希表结构

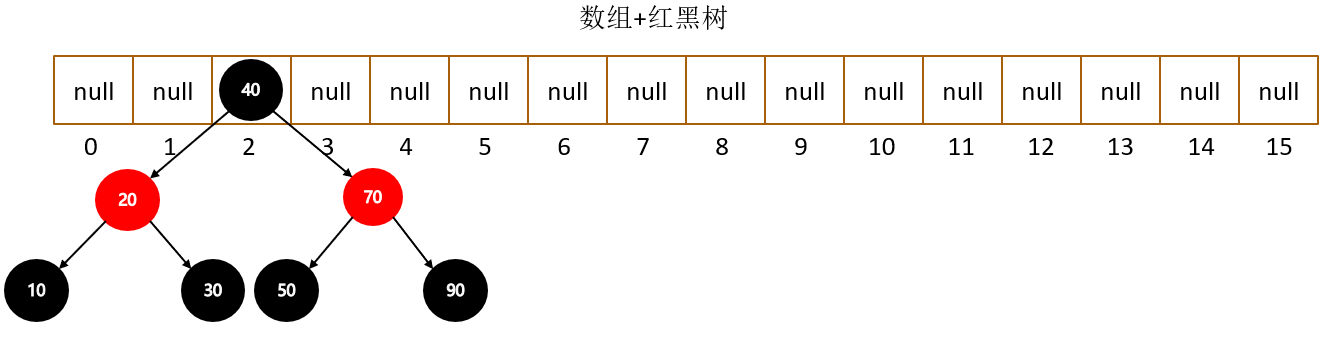
##### 2.4.1 HashSet哈希表结构之数组+链表(注意:JDK1.8之前使用这样的存储结构)

* 当创建HashSet集合时，系统会创建一个默认长度为16，负载因子为0.75的数组
* 当HashSet添加元素时，系统跟根据元素的哈希值与数组的长度计算元素存放在数组的索引位置
* 判断当前索引位置是否为null，若为null则元素作为链表数据存放
* 若不为null，则当前位置有元素，则比较元素的值
* 若元素的值一样，则不存，若不一样，则存入数据，老元素挂载新元素的下面



##### 2.4.2 HashSet哈希表结构之数组+红黑树(注意:JDK1.8之后使用这样的存储结构)

* 当创建HashSet集合时，系统会创建一个默认长度为16，加载因子为0.75的数组
* 当HashSet添加元素时，系统跟根据元素的哈希值与数组的长度计算元素存放在数组的索引位置
* 判断当前索引位置是否为null，若为null则元素作为红黑树结点数据存放
* 若不为null，则当前位置有元素，则比较元素的值
* 若元素的值一样，则不存，若不一样，则存入数据，按照红黑树规则有序存放



##### 2.4.3 HashMap底层数据结构

HashMap底层是数组+链表+红黑树结构，当数组某一个索引位置元素个数小8时，采用的是数组+链表结构。当数组某一个索引位置元素个数大于等于8时，会转换成数组+红黑树结构

##### 2.4.4 HashMap的扩容机制

HashMap的底层是数组+链表+红黑树，数组默认初始长度是16，加载因子为0.75。虽然我们在数组中的元素存储的是红黑树，可以提高我们查找的效率。但是当数据存储足够多时，我们就需要对HashMap进行扩容，能承载更大的数据量。

HashMap的扩容时机：当数组的索引位置不为空的元素个数大于等于数组长度\*加载因子时触发扩容机制，扩容后新数组的长度为原数组长度的2倍

例：数组长度为16，加载因子为0.75，当数组索引位置不为空的元素大于12个时触发扩容机制，扩容后新数组长度为32

#### 2.5 小结

哈希值特点

* 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
* 哈希表优点：提供快速的插入和查找的操作
* 哈希表缺点：一旦创建就不好扩展