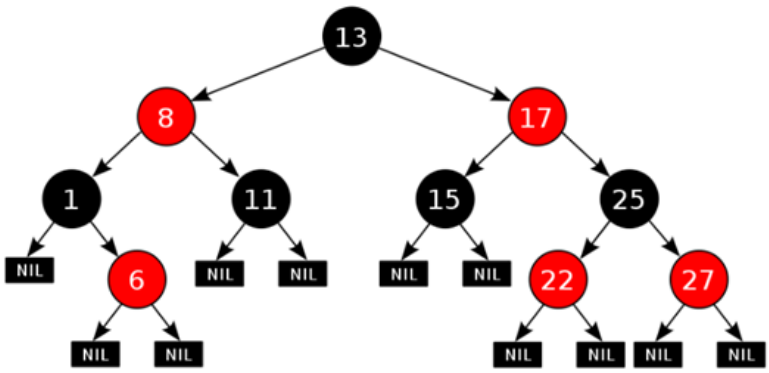
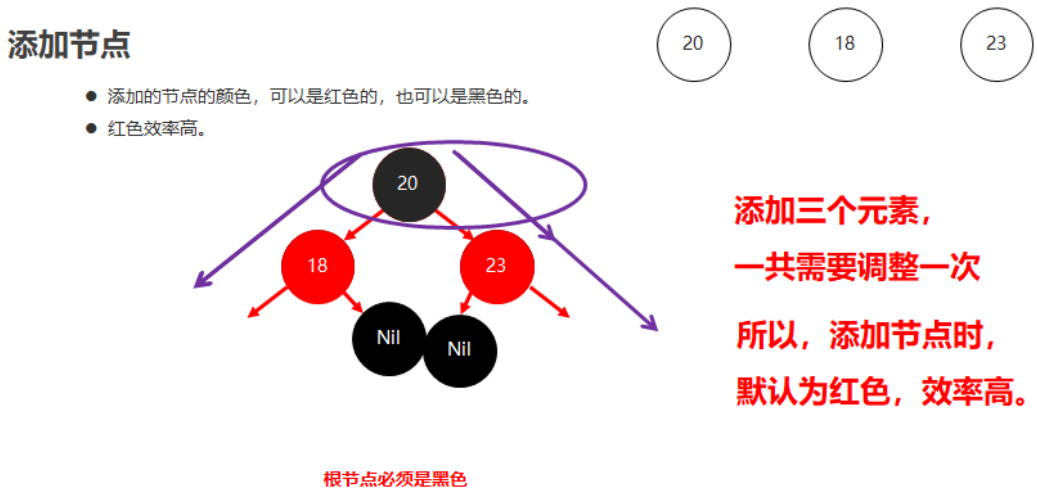
1.红黑树

1.1概述【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

1.2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());
        }
    }
}
```

2.HashSet集合

2.1HashSet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

2.2HashSet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);
        }
    }
}
```

2.3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

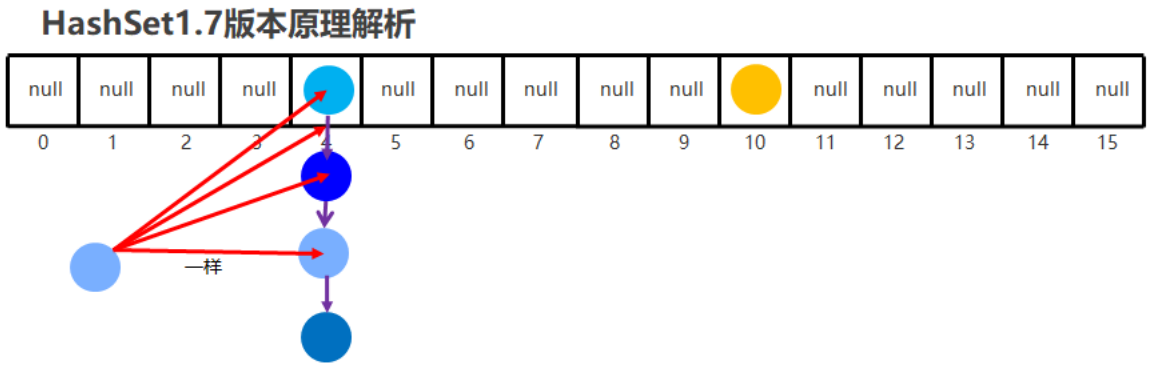
  Object类中的public int hashCode()：返回对象的哈希码值

- 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

2.4哈希表结构【理解】

- JDK1.8以前

  数组 + 链表

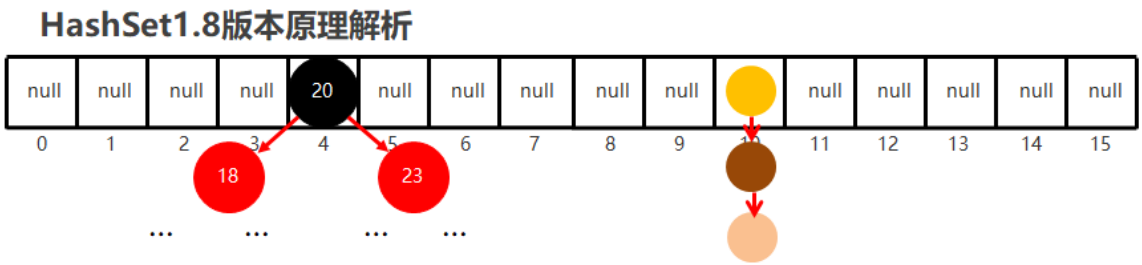


1. 创建一个默认长度16，默认加载因为0.75的数组，数组名table
2. 根据元素的哈希值跟数组的长度计算出应存入的位置
3. 判断当前位置是否为null，如果是null直接存入
4. 如果位置不为null，表示有元素，则调用equals方法比较属性值
5. 如果一样，则不存，如果不一样，则存入数组，老元素挂在新元素下面

- 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

2.5HashSet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

3.Map集合

3.1Map集合概述和特点【理解】

- Map集合概述

```
interface Map<K,V>  K：键的类型；V：值的类型
```

- Map集合的特点
  - 双列集合,一个键对应一个值
  - 键不可以重复,值可以重复
- Map集合的基本使用

```
public class MapDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key, V value) 将指定的值与该映射中的指定键相关联
        map.put("itheima001","林青霞");

        map.put("itheima002","张曼玉");
```

```
        map.put("itheima003","王祖贤");
        map.put("itheima003","柳岩");

        //输出集合对象
        System.out.println(map);
    }
}
```

3.2Map集合的基本功能【应用】

- 方法介绍

方法名	说明
V put(K key,V value)	添加元素
V remove(Object key)	根据键删除键值对元素
void clear()	移除所有的键值对元素
boolean containsKey(Object key)	判断集合是否包含指定的键
boolean containsValue(Object value)	判断集合是否包含指定的值
boolean isEmpty()	判断集合是否为空
int size()	集合的长度，也就是集合中键值对的个数

- 示例代码

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key,V value): 添加元素
        map.put("张无忌","赵敏");
        map.put("郭靖","黄蓉");
        map.put("杨过","小龙女");

        //V remove(Object key): 根据键删除键值对元素
//        System.out.println(map.remove("郭靖"));
//        System.out.println(map.remove("郭襄"));

        //void clear(): 移除所有的键值对元素
//        map.clear();

        //boolean containsKey(Object key): 判断集合是否包含指定的键
//        System.out.println(map.containsKey("郭靖"));
//        System.out.println(map.containsKey("郭襄"));

        //boolean isEmpty(): 判断集合是否为空
//        System.out.println(map.isEmpty());
```

```
        //int size(): 集合的长度，也就是集合中键值对的个数
        System.out.println(map.size());

        //输出集合对象
        System.out.println(map);
    }
}
```

3.3Map集合的获取功能【应用】

- 方法介绍

方法名	说明
V get(Object key)	根据键获取值
Set keySet()	获取所有键的集合
Collection values()	获取所有值的集合
Set> entrySet()	获取所有键值对对象的集合

- 示例代码

```
public class MapDemo03 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //V get(Object key):根据键获取值
//        System.out.println(map.get("张无忌"));
//        System.out.println(map.get("张三丰"));

        //Set<K> keySet():获取所有键的集合
//        Set<String> keySet = map.keySet();
//        for(String key : keySet) {
//            System.out.println(key);
//        }

        //Collection<V> values():获取所有值的集合
        Collection<String> values = map.values();
        for(String value : values) {
            System.out.println(value);
        }
    }

}
```

3.4Map集合的遍历(方式1)【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 把所有的丈夫给集中起来
    - 遍历丈夫的集合，获取到每一个丈夫
    - 根据丈夫去找对应的妻子
- 步骤分析
  - 获取所有键的集合。用keySet()方法实现
  - 遍历键的集合，获取到每一个键。用增强for实现
  - 根据键去找值。用get(Object key)方法实现
- 代码实现

```
public class MapDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //获取所有键的集合。用keySet()方法实现
        Set<String> keySet = map.keySet();
        //遍历键的集合，获取到每一个键。用增强for实现
        for (String key : keySet) {
            //根据键去找值。用get(Object key)方法实现
            String value = map.get(key);
            System.out.println(key + "," + value);
        }
    }
}
```

3.5Map集合的遍历(方式2)【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 获取所有结婚证的集合
    - 遍历结婚证的集合，得到每一个结婚证
    - 根据结婚证获取丈夫和妻子
- 步骤分析
  - 获取所有键值对对象的集合
    - Set> entrySet()：获取所有键值对对象的集合
  - 遍历键值对对象的集合，得到每一个键值对对象

        - 用增强for实现，得到每一个Map.Entry
    - 根据键值对对象获取键和值
        - 用getKey()得到键
        - 用getValue()得到值
- 代码实现

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //获取所有键值对对象的集合
        Set<Map.Entry<String, String>> entrySet = map.entrySet();
        //遍历键值对对象的集合，得到每一个键值对对象
        for (Map.Entry<String, String> me : entrySet) {
            //根据键值对对象获取键和值
            String key = me.getKey();
            String value = me.getValue();
            System.out.println(key + "," + value);
        }
    }
}
```

4.HashMap集合

4.1HashMap集合概述和特点【理解】

- HashMap底层是哈希表结构的
- 依赖hashCode方法和equals方法保证键的唯一
- 如果键要存储的是自定义对象，需要重写hashCode和equals方法

4.2HashMap集合应用案例【应用】

- 案例需求
    - 创建一个HashMap集合，键是学生对象(Student)，值是居住地 (String)。存储多个元素，并遍历。
    - 要求保证键的唯一性：如果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 HashMapDemo {
    public static void main(String[] args) {
        //创建HashMap集合对象
        HashMap<Student, String> hm = new HashMap<Student, String>();

        //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
        hm.put(s1, "西安");
        hm.put(s2, "武汉");
        hm.put(s3, "郑州");
        hm.put(s4, "北京");

        //遍历集合
        Set<Student> keySet = hm.keySet();
        for (Student key : keySet) {
            String value = hm.get(key);
            System.out.println(key.getName() + "," + key.getAge() + "," + value);
        }
    }
}
```

5.TreeMap集合

5.1TreeMap集合概述和特点【理解】

- TreeMap底层是红黑树结构
- 依赖自然排序或者比较器排序,对键进行排序
- 如果键存储的是自定义对象,需要实现Comparable接口或者在创建TreeMap对象时候给出比较器排序规则

5.2TreeMap集合应用案例【应用】

- 案例需求
  - 创建一个TreeMap集合,键是学生对象(Student),值是籍贯(String),学生属性姓名和年龄,按照年龄进行排序并遍历
  - 要求按照学生的年龄进行排序,如果年龄相同则按照姓名进行排序
- 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) {
        //按照年龄进行排序
        int result = o.getAge() - this.getAge();
        //次要条件，按照姓名排序。
        result = result == 0 ? o.getName().compareTo(this.getName()) : result;
        return result;
    }
}
```

测试类

```
public class Test1 {
    public static void main(String[] args) {
        // 创建TreeMap集合对象
        TreeMap<Student,String> tm = new TreeMap<>();

         // 创建学生对象
        Student s1 = new Student("xiaohei",23);
        Student s2 = new Student("dapang",22);
        Student s3 = new Student("xiaomei",22);

         // 将学生对象添加到TreeMap集合中
        tm.put(s1,"江苏");
        tm.put(s2,"北京");
        tm.put(s3,"天津");

         // 遍历TreeMap集合,打印每个学生的信息
        tm.forEach(
                (Student key, String value)->{
                    System.out.println(key + "---" + value);
                }
```

```
        );
    }
}
```