java.util.Set 继承于 Collection接口

特点：1.存入元素不重复

2.元素无序，无索引方法，不能用普通for遍历（但可用迭代器或增强for）

方法与Collection中一致

Set集合不允许元素重复的原理：

java.util.HashSet 哈希表结构，实现了Set接口

特点：1.存入元素不重复

2.元素无序，无索引方法，不能用普通for遍历（但可用迭代器和增强for）

3.无序集，存储元素和取出元素的顺序可能不一致

4.底层是一个哈希表结构（查询速度非常快）

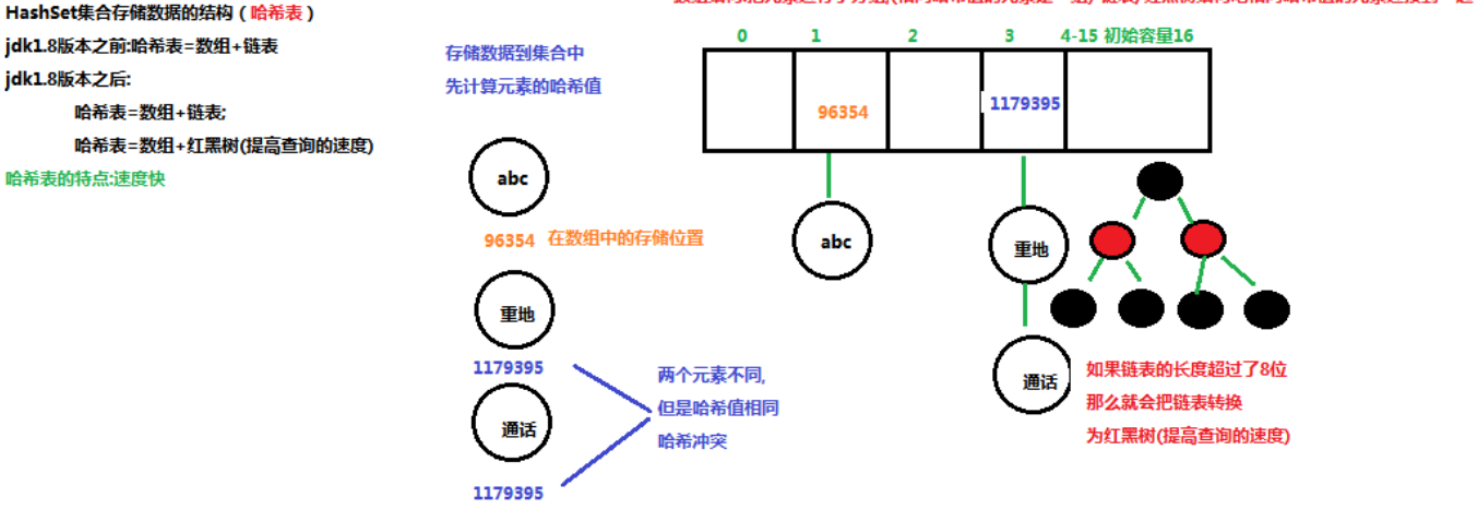
哈希值：十进制整数，由系统随机给出（对象的地址值，是模拟得出的逻辑地址，不是数据实际存储的物理地址）

Object类中有一个方法，能获取对象的哈希值：

public native int hashCode()：返回对象的哈希值。

native 代表方法调用的是本地操作系统的方法

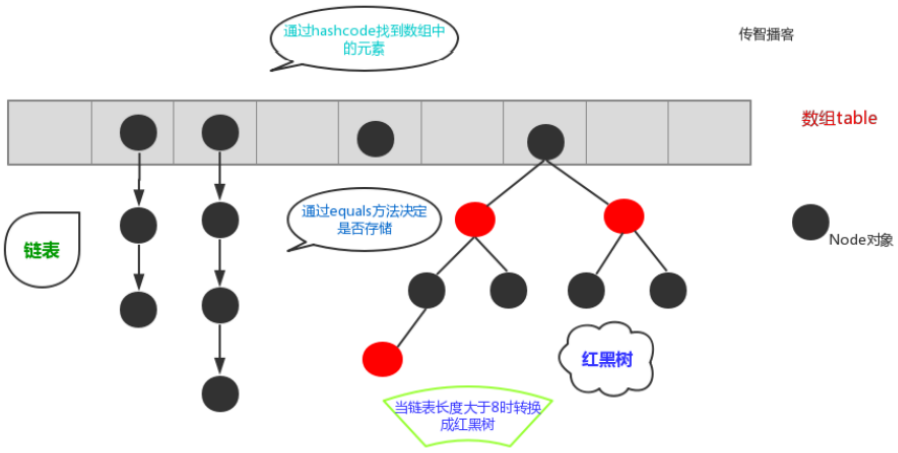
String类的哈希值：String类重写Object类的hashCode方法

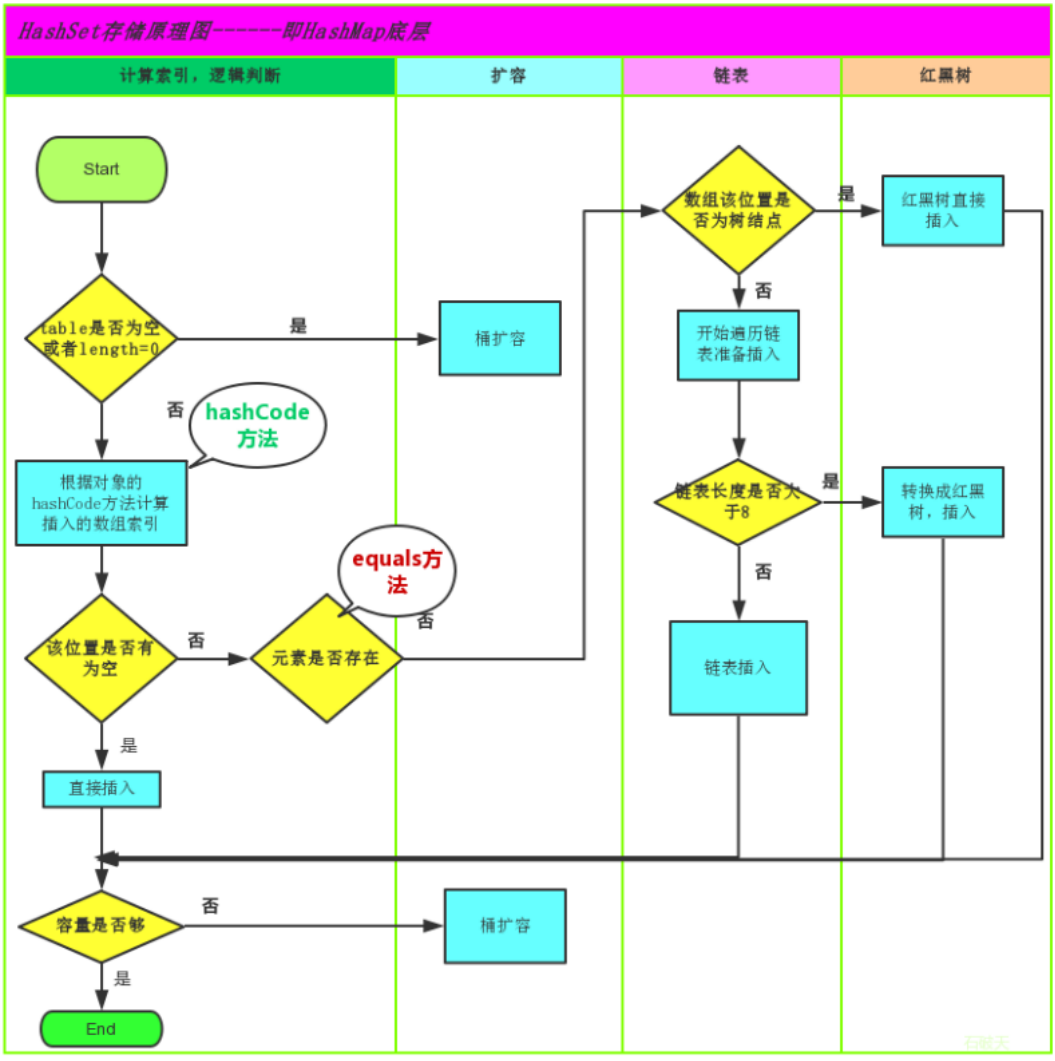


HashSet集合存储数据的结构(哈希表HashMap):速度快

在**JDK1.8**之前，哈希表底层采用数组+链表实现，即使用链表处理冲突，同一hash值的链表都存储在一个链表里。但是当位于一个桶中的元素较多，即hash值相等的元素较多时，通过key值依次查找的效率较低。而JDK1.8中，哈希表存储采用数组+链表+红黑树实现，当链表长度超过阈值（8）时，将链表转换为红黑树，这样大大减少了查找时间。

简单的来说，哈希表是由数组+链表+红黑树（JDK1.8增加了红黑树部分）实现的，如下图所示。





总而言之，**JDK1.8**引入红黑树大程度优化了HashMap的性能，那么对于我们来讲保证HashSet集合元素的唯一，其实就是根据对象的hashCode和equals方法来决定的。如果我们往集合中存放自定义的对象，那么保证其唯一，就必须复写hashCode和equals方法建立属于当前对象的比较方式。

用HashSet存放自定义类型元素时，需要重写对象的hashCode和equals方法，建立比较方式，才能保证HashSet集合中的对象唯一。

创建自定义Student类

```java
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;
        return age == student.age &&
               Objects.equals(name, student.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```java
public class HashSetDemo2 {
    public static void main(String[] args) {
        //创建集合对象   该集合中存储 Student类型对象
        HashSet<Student> stuSet = new HashSet<Student>();
        //存储
        Student stu = new Student("于谦", 43);
        stuSet.add(stu);
        stuSet.add(new Student("郭德纲", 44));
        stuSet.add(new Student("于谦", 43));
        stuSet.add(new Student("郭麒麟", 23));
        stuSet.add(stu);

        for (Student stu2 : stuSet) {
            System.out.println(stu2);
        }
    }
}
```

执行结果：

```
Student [name=郭德纲, age=44]
Student [name=于谦, age=43]
Student [name=郭麒麟, age=23]
```

java.util.LinkedHashSet 链表和哈希表组合的一个数据存储结构，继承了HashSet集合

特点：底层是一个哈希表+链表(记录元素存储顺序，保证元素有序)

```java
/*
    java.util.LinkedHashSet集合 extends HashSet集合
    LinkedHashSet集合特点:
        底层是一个哈希表(数组+链表/红黑树)+链表:多了一条链表(记录元素的存储顺序),保证元素有序
 */
public class Demo04LinkedHashSet {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("www");
        set.add("abc");
        set.add("abc");
        set.add("itcast");
        System.out.println(set);//[abc, www, itcast] 无序,不允许重复

        LinkedHashSet<String> linked = new LinkedHashSet<>();
        linked.add("www");
        linked.add("abc");
        linked.add("abc");
        linked.add("itcast");
        System.out.println(linked);//[www, abc, itcast] 有序,不允许重复
    }
```

可变参数：

使用前提：当方法的参数列表数据类型已经确定，但参数的个数不确定，可在定义方法时使用

修饰符 返回值类型 方法名(参数类型 ... 形参名){

//...

}

(实际上与 修饰符 返回值类型 方法名(参数类型[] 形参名){

//...

}

完全等价，但后者用时必须传递数组，前者传数据即可)

原理：底层是一个数组，根据传参个数不同，会创建不同长度的数组来储存

传参个数可以是 0、1、2、…… 多个

注意：1、一个方法的参数列表只能有一个可变参数

2、若方法的参数有多个，则可变参数必须在参数列表的末尾