

集合

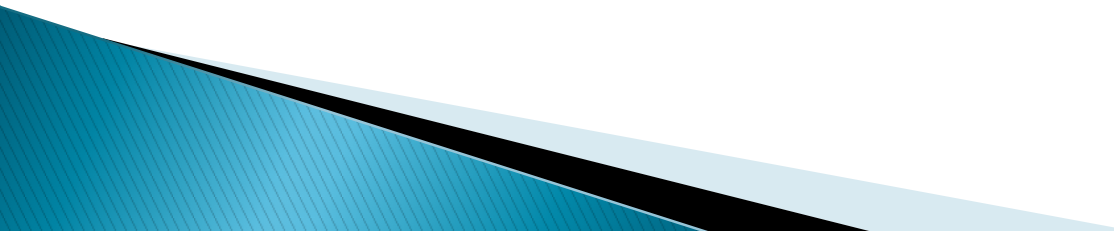
本章知识点

- ▶ Java集合框架
- ▶ List
 - ArrayList
 - LinkedList
- ▶ Set
 - HashSet
 - TreeSet
- ▶ Map
 - HashMap
 - Properties

Java中的集合

- ▶ Java为程序设计者封装了很多数据结构，它们位于java.util包下，称作Java的**集合类**，包括常用的各种数据结构：List表、Set集合、Map映射等。
- ▶ 从Java SE 5.0开始，这些集合类使用泛型进行了改写，集合中对象的数据类型可以被记住，使用者不必再担心对象存储至集合后就失去数据类型的信息。

复习

- ▶ 当需要在程序中记录单个数据的时候，则声明变量记录。
 - ▶ 当需要在程序中记录多个类型相同的数据时，则声明数组记录。
 - ▶ 当需要在程序中记录多个类型不同的数据时，则构造对象记录一类。
 - ▶ 当需要在程序中记录多个类型不同的对象的时候，则使用集合来处理。
- 

8.1 Java中的集合框架

数组集合的比较

▶ 数组的特点

1. 数组本质上就是一段连续的内存空间，用来记录多个类型相同的数据。

2. 数组一旦声明完毕，则内存空间固定不变。

3. 在执行插入和删除操作不方便，可能会移动大量元素导致效率太低

4. 支持下标访问，可以实现随机访问。

5. 数组中的基本元素可以是基本数据类型，也可以是引用数据类型。

数组集合的比较

▶ 集合的特点

1. 内存空间可以不连续，数据类型可以不同。
2. 内存空间可以动态的调整。
3. 集合的插入删除操作可以不移动大量元素。
4. 部分支持下标访问，部分不支持。
5. 集合中的元素必须是引用数据类型。

8.1.1 集合框架的常用部分

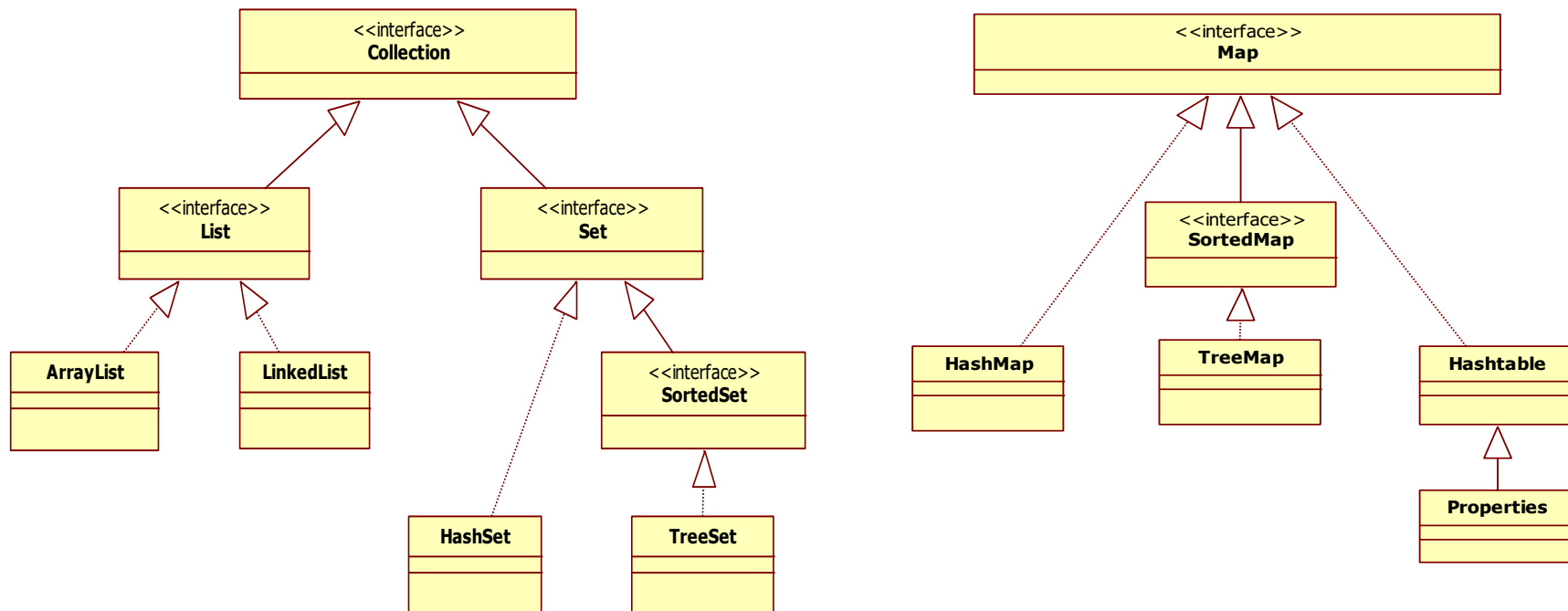
- ▶ 什么是集合

是在计算机中用于存储一种或多种引用数据类型，并且长度可变的容器。

容器就是用来存储和组织其他对象的对象。

8.1.1 集合框架的常用部分

- Java集合框架的根是两个接口：Collection和Map。



Collection集合操作元素的基本单位是：单个元素。

Map集合操作元素的基本单位是：单对元素。

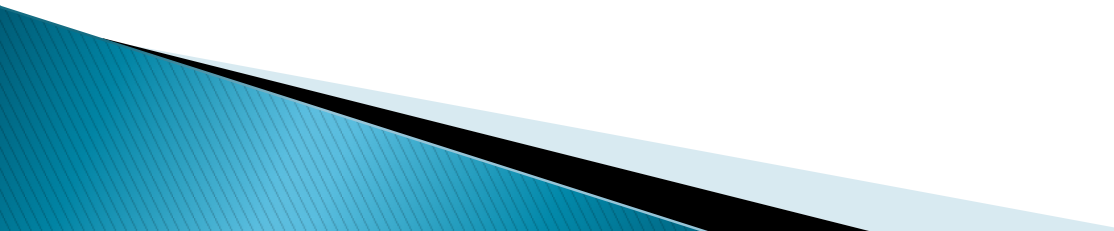
```
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List list=new List;
    }
}
```



List集合---接口

- ▶ List继承自Collection接口，
特点：有序元素可以重复的集合。
List的实现类： ArrayList, LinkedList

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        //List list=new List;
        List c1=new ArrayList();
    }
}
```



集合中的常用方法

1. `void add(int index, Object element)`: 在指定位置`index`上插入元素`element`。
2. `boolean addAll(int index, Collection c)`: 指定位置`index`上插入集合`c`中的所有元素，如果`List`对象发生变化返回`true`。

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1);
    }
}
```

True

集合中可以存放不同类型

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1);
        System.out.println(c1);
    }
}
```

```
true
[1]
```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1);
        System.out.println(c1);
        b1=c1.add(new String("two"));
        System.out.println(b1);
        System.out.println(c1);
```

```
true
[1]
true
[1, two]
```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
public static void main(String[] args){
List c1 =new ArrayList();
boolean b1 =c1.add(new Integer(1));
System.out.println(b1);
System.out.println(c1);
b1 =c1.add(new String("two"));
System.out.println(b1);
System.out.println(c1);
b1 =c1.add(new Student(1001,"张三",30));
System.out.println(b1);
System.out.println(c1);
}
```

```
public class Student {
private int id;
private String name;
private int age;
public int getId() {return id;}
public void setId(int id) {this.id = id;}
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;}
public Student() {}
public Student(int id, String name, int age) {
this.id = id;  this.name = name;  this.age = age;}
public String toString() {
return "Student [id=" + id + ", name=" + name + ", age="
+ age + "]; } }
```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
public static void main(String[] args){
List c1 =new ArrayList();
boolean b1 =c1.add(new Integer(1));
System.out.println(b1);
System.out.println(c1);
b1 =c1.add(new String("two"));
System.out.println(b1);
System.out.println(c1);
b1 =c1.add(new Student(1001,"张三",30));
System.out.println(b1);
System.out.println(c1);
}
```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
public static void main(String[] args){
List c1 =new ArrayList();
boolean b1 =c1.add(1);
System.out.println(b1);
System.out.println(c1);
b1 =c1.add(new Student(1001,"张三",30));
System.out.println(b1);
System.out.println(c1);
}
```

```
true
[1]
true
[1, two]
true
[1, two, Student [id=1001, name=张三, age=30]]
```



```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new String("two"));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new Student(1001,"张三",30));
        System.out.println(b1); System.out.println(c1);
        List c2=new ArrayList();
        b1=c2.add(3);//采用自动装箱技术 int=>Integer
        System.out.println(b1);System.out.println(c2);
        b1=c2.add("four");
        System.out.println(b1); System.out.println(c2);} }
```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args) {
        List c1 = new ArrayList();
        boolean b1 = c1.contains(1);
        System.out.println(b1);
        b1 = c1.add(new Student(1001, "张三", 30));
        System.out.println(b1);
        b1 = c1.add(new Student(1002, "李四", 25));
        System.out.println(b1);
        b1 = c1.add(new Student(1001, "张三", 30));
        System.out.println(b1);
        System.out.println(c1);
        List c2 = new ArrayList();
        b1 = c2.add(3); //采用自动装箱技术 int=>Integer
        System.out.println(b1); System.out.println(c2);
        b1 = c2.add("four");
        System.out.println(b1); System.out.println(c2);} }
```

集合中的常用方法

1. `void add(int index, Object element)`: 在指定位置`index`上插入元素`element`, 如果`List`对象发生变化返回`true`。
2. `boolean addAll(int index, Collection c)`: 指定位置`index`上插入集合`c`中的所有元素, 如果`List`对象发生变化返回`true`。
3. `size()`;返回集合的元素数。

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new String("two"));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new Student(1001,"张三",30));
        System.out.println(b1); System.out.println(c1);
        List c2=new ArrayList();
        b1=c2.add(3);//采用自动装箱技术 int=>Integer
        System.out.println(b1);System.out.println(c2);
        b1=c2.add("four");
        System.out.println(b1); System.out.println(c2);
        System.out.println(c1.size());System.out.println(c2.size());} }
```

```

import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args) {
        List c1 = new ArrayList();
        boolean b1 = c1.contains(1);
        System.out.println(b1);
        b1 = c1.add(new Student(1, two));
        System.out.println(b1);
        b1 = c1.add(new Student(3, four));
        System.out.println(b1);
        b1 = c1.add(new Student(3, four));
        System.out.println(b1);
        System.out.println(c1);
        List c2 = new ArrayList();
        b1 = c2.add(3); //采用自动装箱技术 int=>Integer
        System.out.println(b1); System.out.println(c2);
        b1 = c2.add("four");
        System.out.println(b1); System.out.println(c2);
        System.out.println(c1.size()); System.out.println(c2.size());
    }
}

```

```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        boolean b1=c1.add(new Integer(1));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new String("two"));
        System.out.println(b1); System.out.println(c1);
        b1=c1.add(new Student(1001,"张三",30));
        System.out.println(b1); System.out.println(c1);
        List c2=new ArrayList();
        b1=c2.add(3);//采用自动装箱技术 int=>Integer
        System.out.println(b1);System.out.println(c2);
        b1=c2.add("four");
        System.out.println(b1); System.out.println(c2);
        System.out.println(c1.size());System.out.println(c2.size());
        b1=c1.addAll(c2);
        System.out.println(c1);System.out.println(c2);} }
```

```
import java.util.ArrayList;
```

```
import java.u
```

```
public class T
```

```
    public
```

```
        List c1
```

```
        boolea
```

```
        System
```

```
        b1 = c1
```

```
        System
```

```
        b1 = c1
```

```
        System
```

```
        List c2
```

```
    true
```

```
    [1]
```

```
    true
```

```
    [1, two]
```

```
    true
```

```
    [1, two, Student [id=1001, name=张三, age=30]]
```

```
    true
```

```
    [3]
```

```
    true
```

```
    [3, four]
```

```
    3
```

```
    2
```

```
    [1, two, Student [id=1001, name=张三, age=30], 3, four]
```

```
    [3, four]
```

```
    b1 = c2.add(3); //采用自动装箱技术 int=>Integer
```

```
        System.out.println(b1); System.out.println(c2);
```

```
        b1 = c2.add("four");
```

```
        System.out.println(b1); System.out.println(c2);
```

```
        System.out.println(c1.size()); System.out.println(c2.size());
```

```
        b1 = c1.addAll(c2);
```

```
        System.out.println(c1); System.out.println(c2); } }
```

集合中的常用方法

1. `void add(int index, Object element)`: 在指定位置`index`上插入元素`element`, 如果`List`对象发生变化返回`true`。
2. `boolean addAll(int index, Collection c)`: 指定位置`index`上插入集合`c`中的所有元素, 如果`List`对象发生变化返回`true`。
3. `size()`;返回集合的元素数。
4. `Object remove(int index)`: 删除指定位置`index`上的元素。


```
import java.util.ArrayList;
import java.util.List;
public class TestList {
    public static void main(String[] args){
        List c1=new ArrayList();
        ..... //省略
        System.out.println(c1.size());System.out.println(c2.size());
            b1=c1.addAll(c2);
        System.out.println(c1);System.out.println(c2);
b1=c1.remove(new String("two"));
System.out.println(c1);
b1=c1.removeAll(c2);
System.out.println(c1);} }
```

```
import java.util.ArrayList;
import java.util.List;
public class Test {
    public static void main(String[] args) {
        List c1 = new ArrayList<>();
        List c2 = new ArrayList<>();
        c1.add("one");
        c1.add("two");
        c1.add(new Student(1001, "张三", 30));
        c1.add(3);
        c1.add("four");
        c2.add("one");
        c2.add(new Student(1001, "张三", 30));
        System.out.println(c1.size());
        System.out.println(c2.size());
        c1.addAll(c2);
        System.out.println(c1);
        System.out.println(c2);
        c1.remove(new String("two"));
        System.out.println(c1);
        c1.removeAll(c2);
        System.out.println(c1);
    }
}
```

8.2.1 List接口

- ▶ List接口添加的面向位置的方法
 - void **add**(int index, Object element): 在指定位置index上插入元素element。
 - boolean **addAll**(int index, Collection c): 指定位置index上插入集合c中的所有元素, 如果List对象发生变化返回true。
 - Object **get**(int index): 返回指定位置index上的元素。
 - Object **remove**(int index): 删除指定位置index上的元素。
 - Object **set**(int index, Object element): 用元素element取代位置index上的元素, 并且返回旧元素的取值。
 - public int **indexOf**(Object obj): 从列表的头部开始向后搜索元素obj, 返回第一个出现元素obj的位置, 否则返回-1。
 - public int **lastIndexOf**(Object obj): 从列表的尾部开始向前搜索元素obj, 返回第一个出现元素obj的位置, 否则返回-1。

8.2.2 ArrayList

▶ 1. ArrayList的底层

- ArrayList是List的实现类，它封装了一个可以动态再分配的Object[]数组。
- ArrayList中的常用方法包括：
 - (1) ArrayList(): 创建ArrayList 对象，Object[]数组的长度取值为10。
 - (2) ArrayList(int initialCapacity): 创建ArrayList 对象，使用参数initialCapacity设置Object[]数组的长度。
 - (3) ensureCapacity(): 如果要向ArrayList中添加大量元素，可使用此方法对Object[]数组进行指定的扩容。

8.2.2 ArrayList

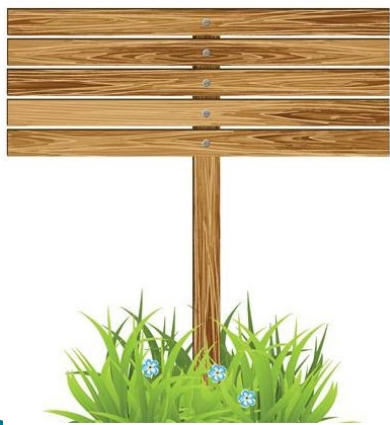
【例8-3】 测试ArrayList初始容量对性能的影响。

- 假设要向ArrayList添加100万个字符串，第一次设置ArrayList的初始容量为10万，通过自动扩容管理ArrayList的存储空间；第二次设置ArrayList的初始容量为100万，不进行扩容处理。比较两次操作的时间。

8.2.2 ArrayList

2. 关于remove()方法

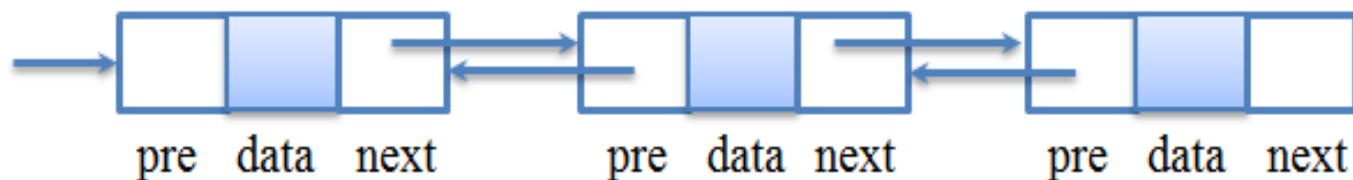
- Collection接口中的remove()方法
 - boolean remove(Object obj)
 - boolean removeAll(Collection c)。
- List中声明了一个按位置索引删除的remove()方法
 - Object remove(int index)。



- 使用迭代器循环迭代的过程中，是不能用这些remove()方法删除集合元素的，因为Collection和List中定义的remove()方法对迭代器的修改，与迭代器本身的管理不能同步，从而会引发ConcurrentModificationException异常。
- **正确做法**：使用迭代器Iterator自身的remove()方法则可以正常地完成删除操作！

8.2.3 LinkedList

- ▶ 链表：采用“按需分配”的原则为每个对象分配独立的存储空间（称为结点），并在每个结点中存放序列中下一个结点的引用。
- ▶ Java中的链表是双链表，每个结点还存放它前面结点的引用。



8.2.3 LinkedList

选择ArrayList还是LinkedList? ?

(1) 如果经常要存取List集合中的元素，那么使用ArrayList采用随机访问的形式（get(index), set(index ,Object element)）性能更好。

(2) 如果要经常执行插入、删除操作，或者改变List集合的大小，则应该使用LinkedList。

8.3 Set及其实现类

- ▶ Set按照无序、不允许重复的方式存放对象
- ▶ 实现类
 - HashSet: 基于散列结构
 - TreeSet: 基于查找树结构

8.3.1 Set接口

- ▶ Set接口继承自Collection 接口，它没有引入新方法，所以Set就是一个Collection，只是行为方式不同。
- ▶ Set不允许集合中存在重复项，如果试图将两个相同的元素加入同一个集合，则添加无效，add()方法返回false。

8.3.1 Set接口

- ▶ Set的实现类依赖添加对象的equals()方法检查对象的唯一性
 - 只要两个对象使用equals()方法比较的结果为true，Set就会拒绝后一个对象的加入（哪怕它们实际上是不同的对象）；
 - 只要两个对象使用equals()方法比较的结果为false，Set就会接纳后一个对象（哪怕它们实际上是相同的对象）。

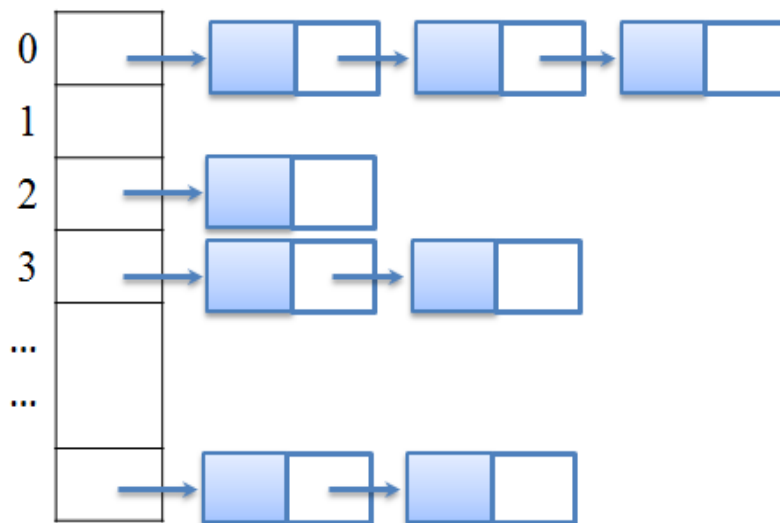


- 使用Set存放对象时，重写该对象所在类的equals()方法、制定正确的比较规则。

8.3.2 HashSet

1. HashSet的底层结构

- HashSet是Set的实现类，它基于一种著名的、可以实现快速查找的散列表（Hash Table）结构。
- 散列表也称哈希表，它采用按照对象的取值计算对象存储地址的策略，实现对象的“定位”存放，相应也提高了查找效率。



8.3.2 HashSet

2. 散列码和hashCode()方法

- 散列码是以某种方法从对象的属性字段产生的整数，Object类中的hashCode()方法完成此任务。



- 要将对象存储在基于散列结构的HashSet，自定义类必须按规则重写Object中的hashCode()方法。
- 所谓的规则就是要**保证hashCode()方法与重写的equals()方法完全兼容**，即如果a.equals(b)为true，那么a和b也必须通过hashCode()方法得到相同的散列码。
- 同时，还要保证计算散列码是快速的。

8.3.2 HashSet

3. 向HashSet中添加元素

- HashSet中不允许存在重复的元素，因此add()方法首先尝试查找要添加的对象，只有在该对象不存在的情况下才执行添加。

(1) 依据自定义类的hashCode()方法计算得到对象obj的散列码，它是一个整数。（需要自定义类重写hashCode()方法）

(2) 将散列码对表长求余，得到对象在散列表中的存储位置p。例如，表长为16时，散列码%16，映射为地址空间0~15。

(3) 如果p位置不发生冲突，则将对象obj插入在p位置的链表中。

8.3.2 HashSet

3. 向HashSet中添加元素

(4) 如果p位置发生冲突，在p位置对应的链表中利用equals()方法查找是否已存在obj对象。（需要自定义类重写equals()方法，规则要与hashCode()方法互相匹配）

- 1) 如果某个equals()比较的结果为true，说明obj对象已存在，将其舍弃。
- 2) 如果与链表中所有对象的equals()比较的结果均为false，说明obj对象尚未存在，obj插入该链表。

8.3.2 HashSet

【例8-5】定义一个Student类（具有name和age两个属性），向HashSet集合中添加几个Student对象，并打印该集合。

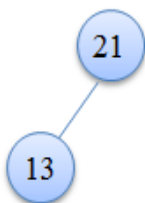
8.3.3 TreeSet

1. TreeSet的底层结构

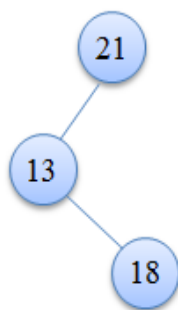
- TreeSet如其名字一样，是一种基于树的集合。TreeSet是Set接口的实现类，秉承了Set不记录对象在集合中出现顺序的特点。但是它最终建立的是一个有序集合，对象可以按照任意顺序插入集合，而对该集合进行迭代时，各个对象将自动以排序后的顺序出现。



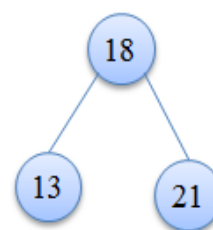
(1)在树中插入结点21



(2)将13插入在21的左侧



(3)将18插入在13的右侧
树失去平衡，进行调整



(4)调整后恢复平衡

8.3.3 TreeSet

2. TreeSet中对象的比较方法

- 有一部分类已实现了java.lang.Comparable接口，如基本类型的包装类、String类等，它们在compareTo()方法中定义好了比较对象的规则。像这样的对象可以直接插入TreeSet集合。
- 使用TreeSet存储**自定义类对象**时，对象所在类要实现Comparable接口，在compareTo()方法中定义对象比较的规则。

8.3.3 TreeSet

4. HashSet和TreeSet的选用

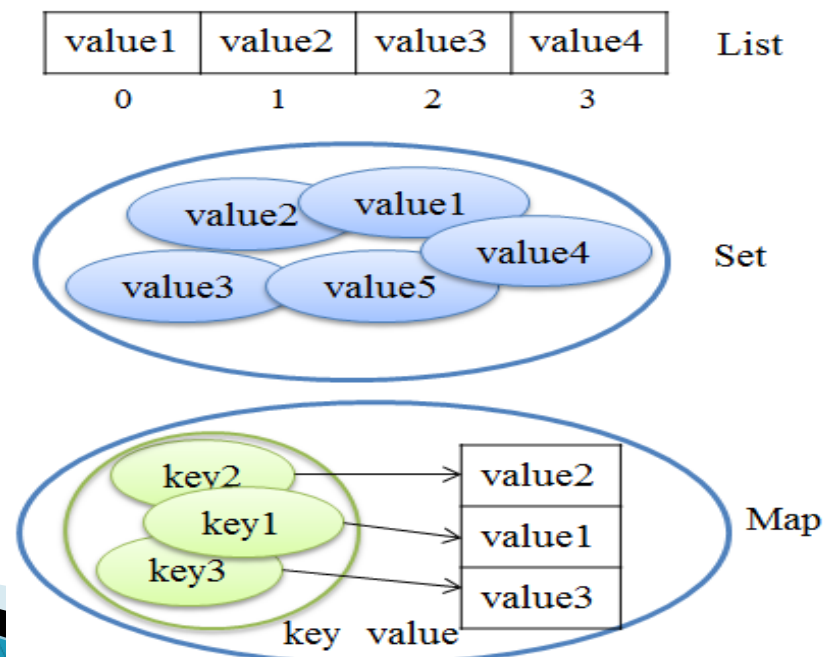
- 原则：取决于集合中存放的对象，如果不需要对对象进行排序，那么就没有理由在排序上花费不必要的开销，使用HashSet即可。
- 散列的规则通常更容易定义，只需要打散排列各个对象就行。而TreeSet要求任何两个对象都必须具有可比性，可是在有的应用中比较的规则会很难定义。

8.3.3 TreeSet

【例8-6】向TreeSet中插入3个字符串，然后输出集合中的所有元素。

8.4 Map及其实现类

- Map用于保存具有映射关系的数据，它们以键值对 $\langle \text{key}, \text{value} \rangle$ 的形式存在，key与value之间存在一对一的关系，多组键值对信息存放于Map集合中。Map集合将键、值分别存放，键的集合用Set存储，不允许重复、无序；值的集合用List存储，与Set对应、可以重复、有序。



8.4.1 Map接口

- ▶ Map接口中的常用方法：查看API文档



事实上，Java是先实现了Map，然后通过包装一个所有value都为null的Map实现了Set，在底层只有Map。

8.4.2 HashMap

【例8-9】假设有一份学生名单，键是学生的id，值是Student对象（包括name和age信息）。建立HashMap对学生信息进行管理。



- HashMap的键在使用时，需要遵守与HashSet一样的规则：如果需要重写equals()方法，那么同时重写hashCode()方法，并保证两个方法判断标准是一致的，因为HashMap的键集就是一个Set。

8.4.2 HashMap

▶ 2. HashMap迭代的方法

- Map的迭代方法较List和Set稍微复杂些，因为它本身是不能迭代的（未实现Iterable接口，不能用迭代器访问）。但从Map出发可以得到三个集合，即键集合、值集合、以及键值对集合，它们都可以被迭代。

8.4.2 HashMap

(1) 按键集合迭代

- 使用keySet()方法可以从Map获取键集，因为键集是Set，所以可以使用迭代器。在迭代过程中，使用get()方法从Map中按key获取到value。

```
Set keys = map.keySet();//(1)取出map中的键集
Iterator it = keys.iterator();
while(it.hasNext()){
    String key = (String)it.next();
    Student stu = (Student)map.get(key); //(2)按键从map中获取对应的值
    System.out.println(key+"："+"("+stu.getName()+","+stu.getAge()+")");
}
```

8.4.2 HashMap

(2) 迭代值集合

- 值集合是一个Collection，它有序、按照键集合中的每个键的排列次序与之一一对应，可以重复，用values()方法即可获取到。但失去了键的对应关系，直接对值集合通常没有什么意义。

(3) 迭代键值对集合

- Map内部定义了一个Entry类，它封装了一个键值对。
- 使用Map的entrySet()方法可以获取键值对集合。

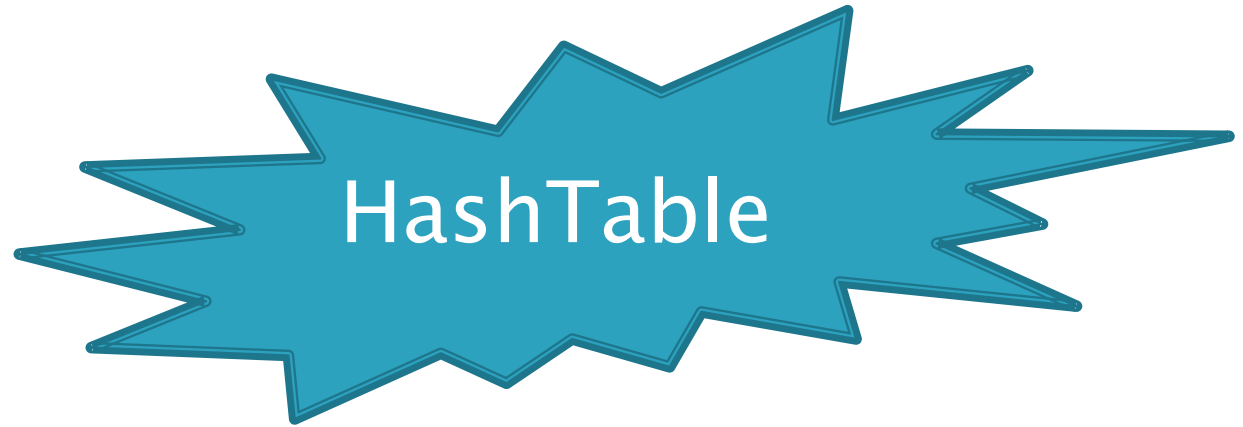
8.4.2 HashMap

(3) 迭代键值对集合

```
Set entrySet = map.entrySet(); //从map获取键值对集合
Iterator it = entrySet.iterator();
while(it.hasNext()){
    Entry entry = (Entry)it.next(); //获取一个键值对对象
    String key = (String)entry.getKey(); //从键值对中获取key
    Student stu = (Student)entry.getValue(); //从键值对中获取value
    System.out.println(key+":"+ "("+stu.getName()+","+stu.getAge()+")");
}
```

8.4.3 Hashtable及其子类Properties

1. Hashtable



- JDK 1.0时代的命名疏忽，延续到现在。
- Hashtable的历史已经非常悠久，它与后来的HashMap几乎相同，它的优势在于它是多线程安全的。
- **但是**，现在即使要保证Map集合的线程安全，也可以不使用Hashtable，后面将介绍的Collections工具类可以将HashMap包装成线程安全的。**所以**，Hashtable终究是要被冷落，终究要退出历史的舞台。

8.4.3 Hashtable及其子类Properties

2. Properties

- Properties是Hashtable的子类，它也实现了Map接口。
- 用途：处理属性文件。

配置文件：“属性名=属性值”

文件ipConfig.properties:

```
server=192.168.0.11  
port=8080
```

8.4.3 Hashtable及其子类Properties

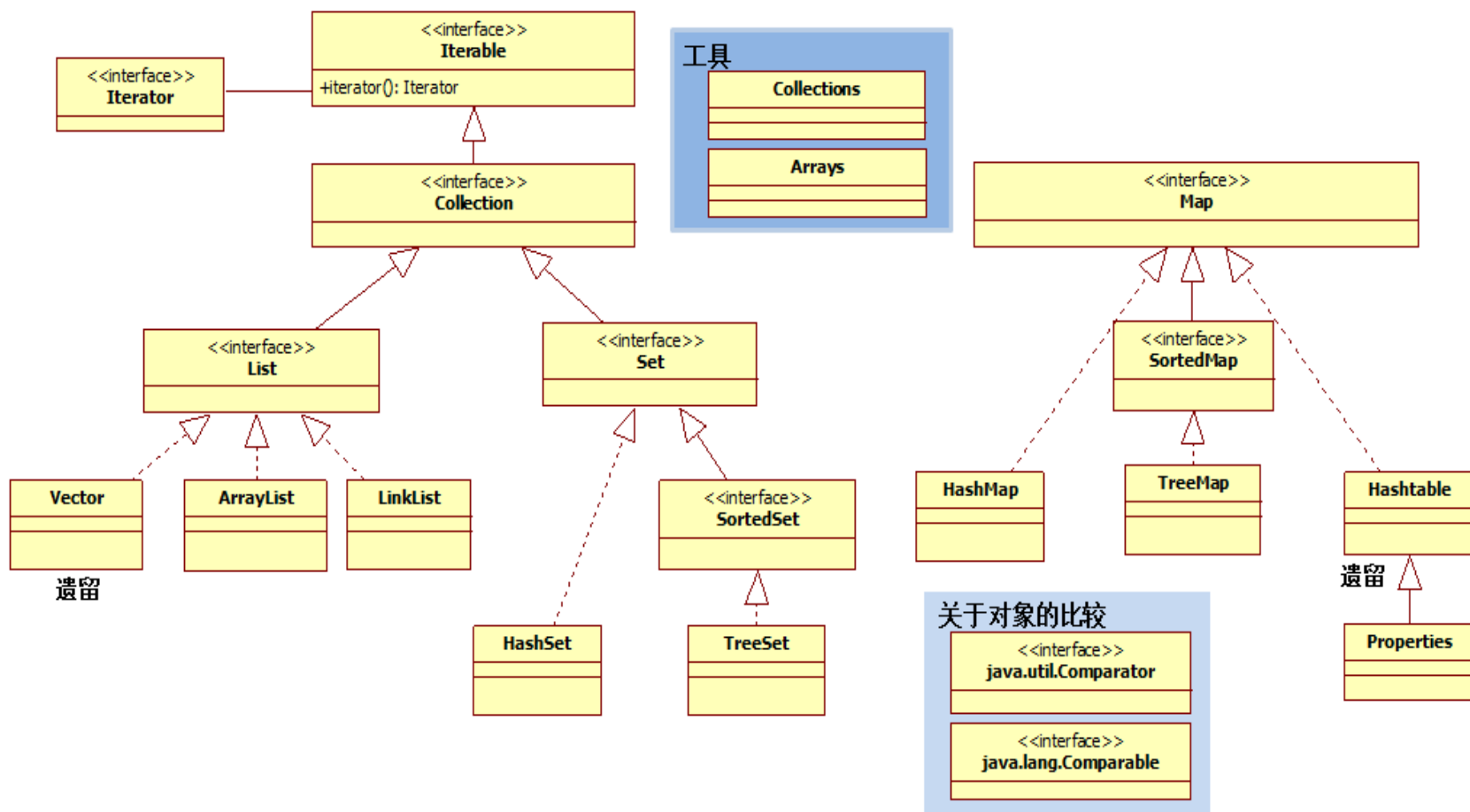
【例8-10】 读取项目中的配置文件"ipConfig.properties"。

```
public static void main(String[] args) throws
FileNotFoundException,IOException{
    Properties pro = new Properties();
    //1.创建一个指向配置文件的输入流
    FileInputStream fis = new FileInputStream("ipConfig.properties");

    //2.读取配置文件
    pro.load(fis);

    //按属性名字获取属性值
    System.out.println("server ip:" + pro.getProperty("server"));
    System.out.println("port:" + pro.getProperty("port"));
}
```

8.5 回首Java集合框架



8.5 回首Java集合框架

接口	实现类	存储方式	优点	缺点	其他
List 有序	ArrayList	数组	按位置索引存取快	插入、删除、查找慢	
	LinkedList	双链表	插入、删除快	按位置存取、查找慢	
Set 无序	HashSet	散列表	插入、删除、查找快		需重写equals()和hashCode()方法
	TreeSet	二叉排序树	插入、删除、查找快 有序排列	速度慢于HashSet	需制定比较规则
Map 映像	HashMap	散列表	插入、删除、查找快	迭代效率较低	键需重写equals()和hashCode()方法
	Properties	散列表	读取属性文件便利		键、值均为String
	TreeMap	二叉排序树	插入、删除、查找快 键值有序排列	速度慢于HashMap 迭代效率低	键需制定比较规则

本章思维导图

