第8章 集合

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

- 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。

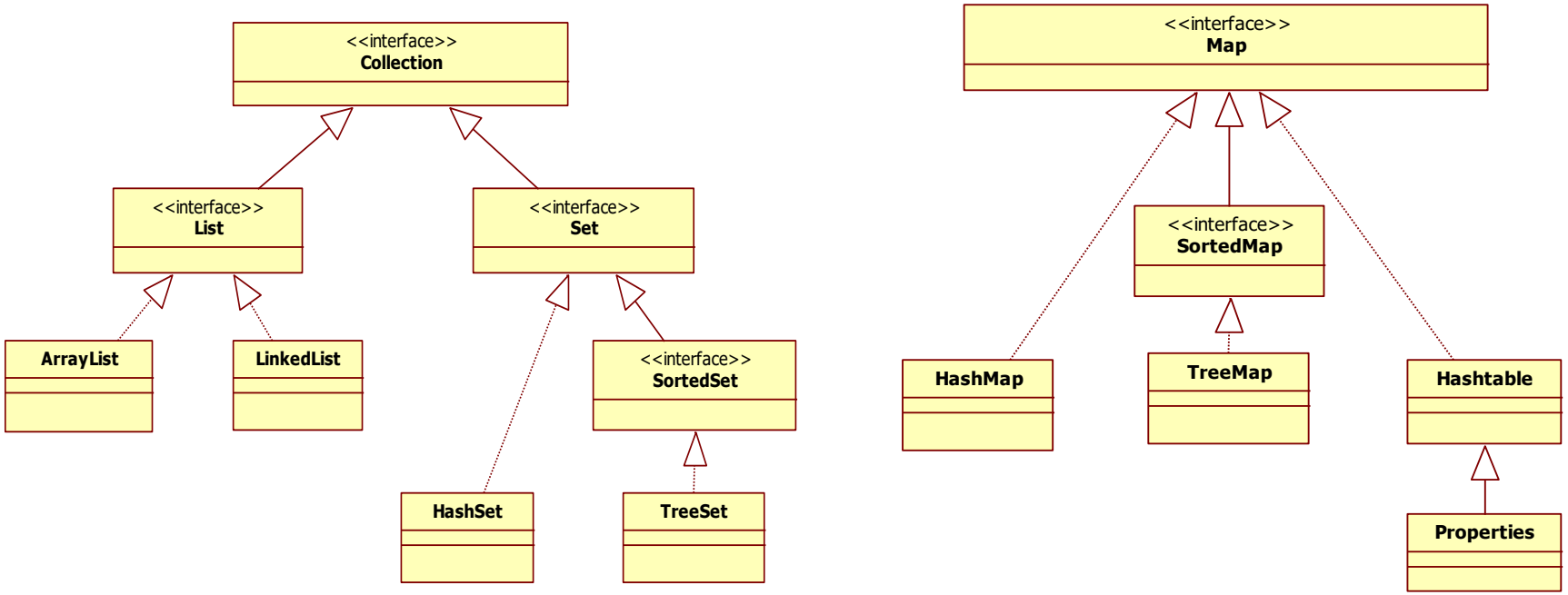
北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

8.1 Java中的集合框架

8.1.1 集合框架的常用部分

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

- Java集合框架的根是两个接口：Collection和Map。



8.1.1 集合框架的常用部分

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 根据Collection和Map的架构，Java中的集合可以分为三大类
  - List：有序集合，它会记录每次添加元素的顺序，元素可以重复；
  - Set：无序集合，它不记录元素添加的顺序，因此元素不可重复；
  - Map：键值对的集合，它的每个元素都由键值key和取值value对应组成，键值和取值分别存储，键值是无序集合Set，取值是有序集合List。

8.1.1 集合框架的常用部分

- 集合与数组有一个明显的不同，就是在Java的集合中只能存储对象，而不能存储基本类型的数据。

8.1.2 迭代器Iterator接口

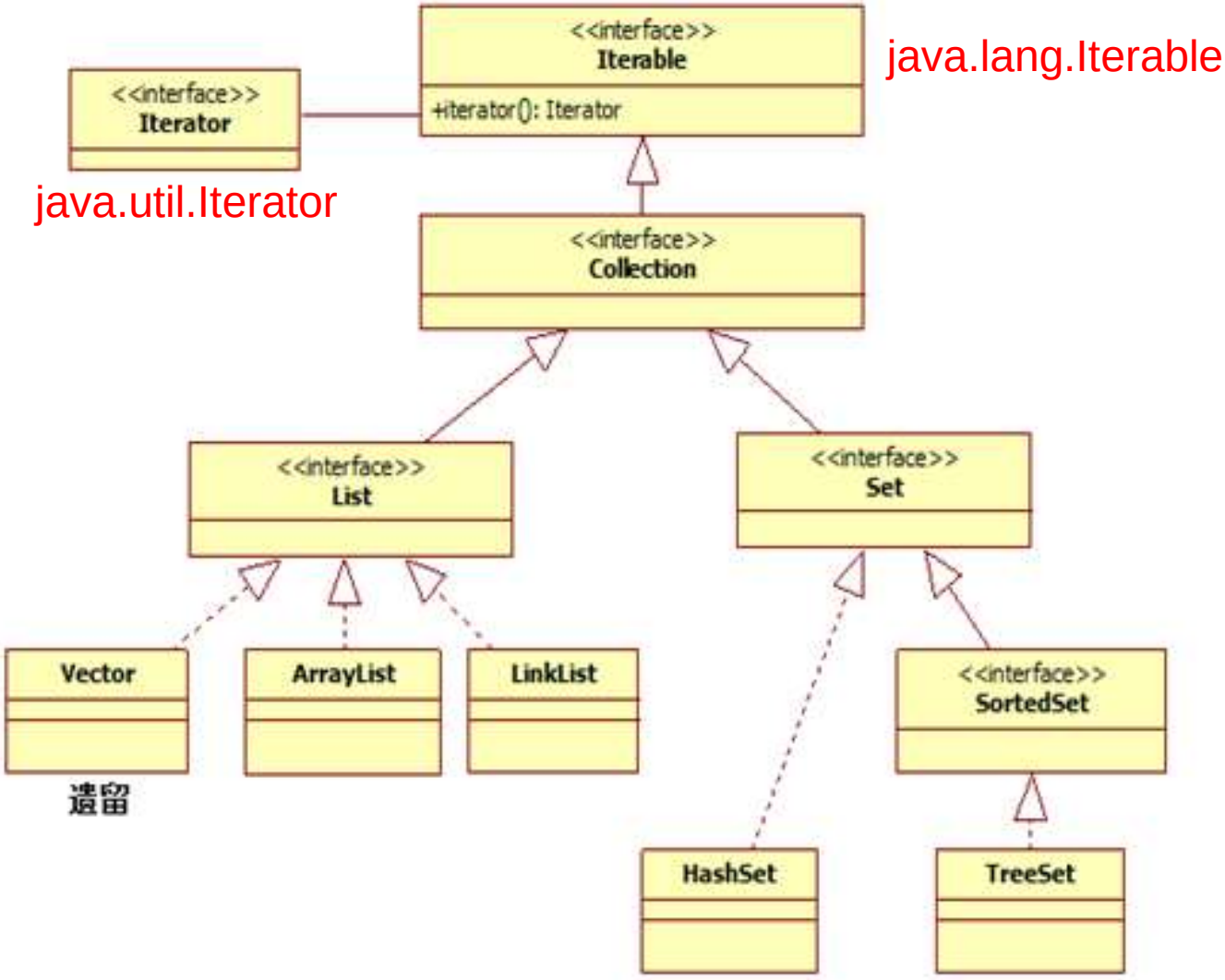
北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Collection接口中的iterator()方法
  - “遍历”：按照某种次序将集合中的元素全部访问到，且每个元素只访问一次。
  - Iterator iterator()方法：用于获取迭代器，从而遍历Collection中的所有元素。

8.1.2 迭代器Iterator接口

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Iterator接口，作为Java集合框架的成员，它的作用不是存放对象，而是遍历集合中的元素。



8.1.2 迭代器Iterator接口

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

- 集合获取迭代器：Iterator iterator()方法。
- Iterator中的方法

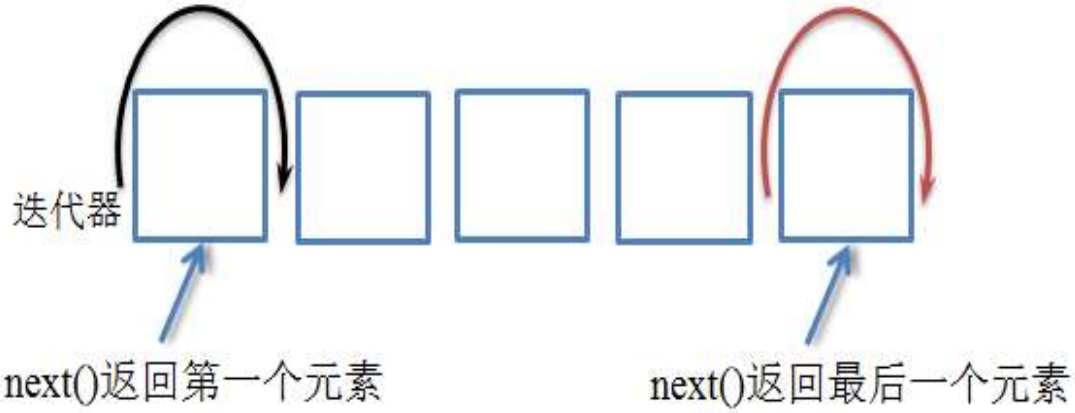
default void	**forEachRemaining**(**Consumer**<? super **E**> action) Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
boolean	**hasNext**() Returns true if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
default void	**remove**() Removes from the underlying collection the last element returned by this iterator (optional operation).

8.1.2 迭代器Iterator接口

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

（1）next()

- Object next()。迭代器会记录迭代位置的变更，它使用next()方法将迭代位置向下一个元素移动，并返回刚刚越过的那个元素。



8.1.2 迭代器Iterator接口

(2) hasNext()

- boolean hasNext()：如果集合中还有未被遍历的元素，返回true，即迭代器还可以继续向后移动的时候返回true。

- 遍历时，在hasNext()方法的控制下，通过反复调用next()方法，逐个访问集合中的各个元素。

8.1.2 迭代器Iterator接口

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```
Collection c = new ArrayList();    //创建一个ArrayList对象
c.add("Java");
c.add("Struts");
c.add("Spring");
```

```
Iterator it = c.iterator();    //(1)iterator()方法获取迭代器
while (it.hasNext()){  //(2)hasNext()方法控制迭代
        //(3)next()方法获取迭代元素
        String element = (String)it.next();
        System.out.println(element);
}
```

传统的for-each循环遍历

```
Collection c = new ArrayList();    //创建一个ArrayList对象
c.add("Java");
c.add("Struts");
c.add("Spring");
```

循环变量 ： 集合对象

```
for(Object element: c){
        String str = (String)element;
        System.out.println(str);
}
```

for-each+lambda表达式遍历

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Java SE 8开始，Iterable接口中新增了forEach方法

Modifier and Type	Method and Description
default void	**forEach**(**Consumer**<? super **T**> action) Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
Iterator<T>	**iterator**() Returns an iterator over elements of type T.

- 参数action：lambda表达式，描述遍历的行为
- lambda表达式能够将行为做为方法的参数

for-each+lambda表达式遍历

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- 对于一个Java变量，我们可以赋给其一个“值”
- int aValue = 2;
- String aString = "Hello World!";
- Boolean aBoolean = aString.startWith("H");
- 如果想要把“一块代码”赋值给Java变量，要怎么做?

aBlockOfCode ← public void doSomeShit(String s){
System.out.println(s);}

- 将右边代码赋值给叫aBlockOfCode的Java变量，利用lambda表达式，可以是实现

aBlockOfCode=public void doSomeShit(String s){
System.out.println(s);}

```
aBlockOfCode = public void doSomeShit(String s){
        System.out.println(s);
}
```

public 是多余的

↓

```
aBlockOfCode = void doSomeShit(String s){
        System.out.println(s);
}
```

函数名字是多余的，因为已经赋值给aBlockOfCode

↓

```
aBlockOfCode = void (String s){
    System.out.println(s);
}
```

返回类型是多余的，编译器可以自己判断返回类型

↓

```
aBlockOfCode = (String s){System.out.println(s);}
```

参数类型也是多余的，编译器可以判断参数类型

↓

```
aBlockOfCode = (s){System.out.println(s);}
```

只有一行可以不要大括号

↓

在参数和函数之间加一个箭头符号->

```
aBlockOfCode = (s)->System.out.println(s);
```

lambda表达式

Q：变量aBlockOfCode的类型

A：所有Lambda的类型都是一个接口，而Lambda表达式本身，就是这个接口的实现。

```
aBlockOfCode=（s）->System.out.println(s);
```

```
MyLambdaInterface aBlockOfCode=
（s）->System.out.println(s);
```

```
@FunctionalInterface
Interface MyLambdaInterface{
  void doSomeShit(String s);
}
```

这种只有一个接口函数需要被实现的接口类型，我们叫“函数式接口”，为了避免后来的程序开发人员在这个接口中增加接口函数导致其有多个接口函数需要被实现，变成“非函数接口”，我们在这个上面加一个声明@FunctionalInterface，这样别人就无法在里面添加新的接口函数。

Lambda表达式的作用：最直观的作用就是使代码变得简洁

对比lambda表达式和传统java对同一个接口的实现：

```
Interface MyLambdaInterface{
        void doSomeShit(Strings);
}
```

```
public class MyInterfaceImpl implements
MyLambdaInterface{
      @Override
       public void doSomeShit(String s){
       System.out.println(s);
 }
  MyLambdaInterface aBlockOfCode=new MyInterfaceImpl();
```

Java8:

```
MyLambdaInterface aBlockOfCode=（s）->System.out.println(s);
```

Java 8中的写法更加优雅简洁。并且，由于Lambda可以直接赋值给一个变量，我们就可以直接把Lambda作为参数传给函数，而传统的Java必须有明确的接口实现的定义，初始化才行：

```
public static void enact(MyLamadaInterface myLambda,String s){
        myLambda.doSomeShit(s);
}
```

java7需要先定义一个class实现接口，再把class instance传给enact（）

```
public class MyInterfaceImpl implements MyLambdaInterface{
        @Override
        public void doSomeShit(String s){
                System.out.println(s);
}
MyLambdaInterface anInterfaceImpl=new MyInterfaceImpl();
enact(anInterfaceImpl,"Hello World!");
```

java8:直接把Lambda表达式传给enact（）

```
enact(s->System.out.printIn(s),"Hello World!");
```

for-each+lambda表达式遍历

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- lambda表达式是java8的特性。它是一个匿名函数，将代码像数据一样可以进行传递。使用它可以写出更简洁、更灵活的代码。即如果你想把“一块代码”赋给一个Java变量，应该怎么做呢

- lambda操作符（->）：
- 将lambda分为两部分：
- 左侧：指定lambda表达式的参数列表
- 右侧：指定lambda体，是抽象方法的实现逻辑，也是表达式要执行的功能

for-each+lambda表达式遍历

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

- lambda表达式是Java SE 8中增加的重要新特性

```
(parameters) -> expression
```

或者：

```
(parameters) -> { statements; }
```

- 1）不需要声明参数的数据类型，编译器可以统一识别参数。
- 2）参数圆括号可选：一个参数无需定义圆括号，但多个参数需要定义圆括号。
- 3）大括号可选：如果主体只包含一个语句，则不需要使用大括号，也不书写分号；主体是多条语句时需要使用大括号，且语句以分号结束。
- 4）返回关键字可选：如果主体只有一个表达式，则编译器会自动将该值返回；如果出现大括号则返回值需要用return指定。

for-each+lambda表达式遍历

```
Collection c = new ArrayList();      //创建一个ArrayList对象

c.add("Java");
c.add("Struts");
c.add("Spring");
```

```
c.forEach( ele -> System.out.println(ele) );
```

集合对象　　循环变量　->

```
c.forEach( (ele) -> {System.out.println(ele);} );
```

- Iterator接口中的方法

（3）remove()

- void remove()：删除集合中上一次调用next()方法时返回的元素。

- 使用remove()删除某位置的元素时，必须先调用next()方法使迭代器跳过该元素。

设集合中已存在若干元素，删除第一个元素的方法：

```
Iterator it = c.iterator();
it.next();      //越过第一个元素
it.remove();  //删除第一个元素
```

- 在泛型出现之前，一旦将一个对象放在Java的集合中，集合就会忘记对象的类型，所有对象都被当做Object处理。当从集合中取出对象后，就需要进行强制类型转换。

- 没有泛型的缺点

1）什么类型的数据都可以扔进同一个集合里。

2）强制类型转换使代码变得臃肿。

3）编译器对强制类型转换采取一律放行的态度，它并不检查这种转换是否真正成立，所以程序执行过程中如果遇到非法的强转，JVM就会抛出ClassCastException异常，代码的可靠性低。

Collection<String> c = new ArrayList<String>();

Iterator<String> it = c.iterator();

- ArrayList类的API文档

boolean	**add**(**E** e) Appends the specified element to the end of this list.
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list.

- 【说明】从Java SE 8开始，等号右侧的“<>”中的泛型可以不写，类型进行自动推断。

```
Collection<String> c = new ArrayList<>();

Iterator<String> it = c.iterator();
while (it.hasNext()){
        String element = it.next();    //不需要强转
        System.out.println(element);
}
```

boolean **addAll(Collection<? extends E> c)**

Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.

“<? extends E>”中“？”表示通配，“<? extends E>”是“上界通配符”，表示该类型的上限是E。

```
6  class Parent{
7  }
8  class Children extends Parent{
9  }
10 public class GenericExtendsDemo {
11
12     public static void main(String[] args) {
13         Collection<Parent> c = new ArrayList<>();
14
15         Collection<Children> c2 = new ArrayList<>();
16         c2.add(new Children());
17         c2.add(new Children());
18
19         c.add(new Parent());  //E类型
20         c.addAll(c2);    //E的子类类型
21         c.add(new String());     //无关类型
22     }
23 }
```

- 线性表结构
  - 顺序表
  - 链表

- 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。

【例8-2】示范List的常规用法。

- 向List集合添加几个字符串，按索引位置进行插入、修改、删除、查找等操作，并打印集合（根据索引位置遍历集合中的元素）。

- **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()：从Java SE 8 开始，ArrayList()初始化时得到一个空数组，在第一次add()运算时将数组扩容为10。

    （2）ArrayList(int initialCapacity)：创建ArrayList 对象，使用参数initialCapacity设置Object[]数组的长度。

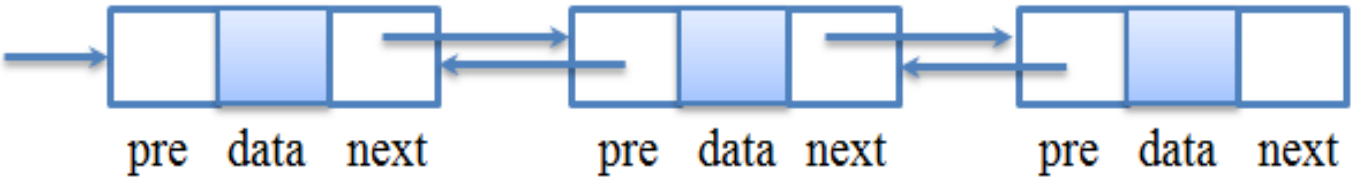
    （3）ensureCapacity():如果要向ArrayList中添加大量元素，可使用此方法对Object[]数组进行指定的扩容。

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！

- 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。



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。

- 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

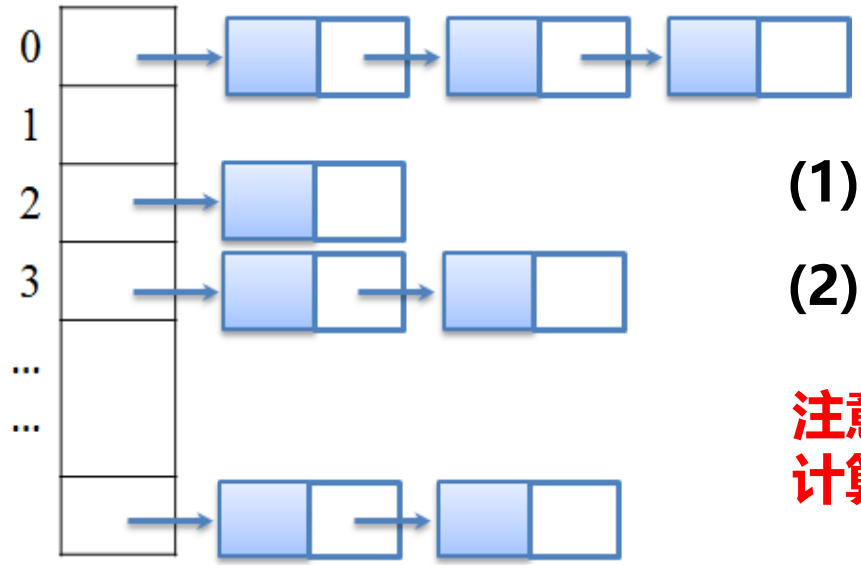
- Set接口继承自Collection 接口，它没有引入新方法，所以Set就是一个Collection，只是行为方式不同。
- Set不允许集合中存在重复项，如果试图将两个相同的元素加入同一个集合，则添加无效，add()方法返回false。

- 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。

HashSet的底层结构

- HashSet是Set的实现类，它基于可以实现快速查找的散列表（Hash Table）结构。
- 散列表采用按照数据的取值计算数据存储地址的策略，实现对象的“定位”存放，相应也提高了查找效率。



(1) 对象定位-- hashCode()方法

(2) 对象查重-- equals()方法

注意：重写两个方法，保证二者的计算依据相同

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。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

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。

（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。

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）

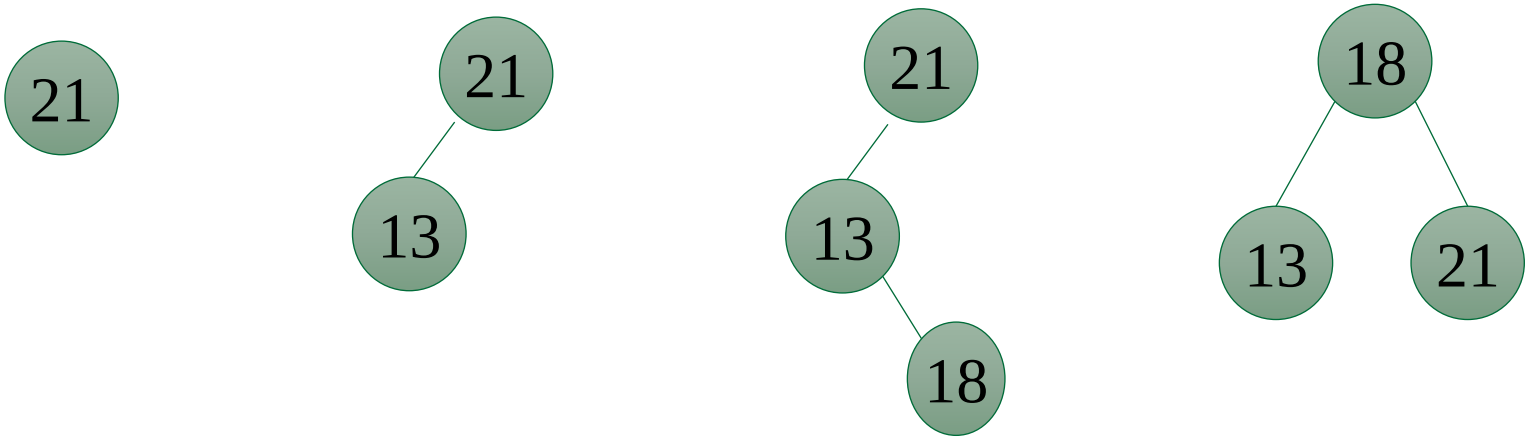
- 1）如果某个equals()比较的结果为true，说明obj对象已存在，将其舍弃。
- 2）如果与链表中所有对象的equals()比较的结果均为false，说明obj对象尚未存在，obj插入该链表。

【例8-4】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。

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。
- TreeSet所基于的数据结构叫做红黑树（Red Black Tree）。红黑树一种会保持左右比较平衡的二叉排序树，所谓“二叉排序树”是这样一种二叉树：它保证每个根结点都比左子树中其他结点值大，都比右子树中其他结点值小，同时左右子树也是二叉排序树。

21,13,18



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。

【例8-5】向TreeSet中插入3个字符串，然后输出集合中的所有元素。

2．TreeSet中对象的比较方法

```
public class Student implements Comparable{
        private String name;
        private int age;
        public int compareTo(Object obj) {
                if(obj instanceof Student){
                        Student stu = (Student)obj;
                        return this.name.compareTo(stu.name);//当前对象-传入对象
                }
                return 0;
        }
}
```

【例8-7】为Student类定义两个比较器ComparatorName和ComparatorNameAge。ComparatorName按name排序，ComparatorNameAge在name相同时继续按age排序。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3．向TreeSet注入比较器

```
public class ComparatorName implements Comparator{  //按name进行排序
        public int compare(Object obj1, Object obj2) {
                if(obj1 instanceof Student && obj2 instanceof Student){
                        Student s1 = (Student)obj1;
                        Student s2 = (Student)obj2;
                        //按name进行比较
                        return s1.getName().compareTo(s2.getName());
                }
                return 0;
        }
}
```

3．向TreeSet注入比较器

```
public class ComparatorNameAge  implements Comparator{ //按name和age进行排序
        public int compare(Object obj1, Object obj2) {
                if(obj1 instanceof Student && obj2 instanceof Student){
                        Student s1 = (Student)obj1;
                        Student s2 = (Student)obj2;
                        if(s1.getName().equals(s2.getName())){
                                return s1.getAge()-s2.getAge();
                        }else{
                                return s1.getName().compareTo(s2.getName());
                        }
                }
                return 0;
        }
}
```

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3．向TreeSet注入比较器

```
Set<Student>  students = new TreeSet<>(new ComparatorName())
```

或者：

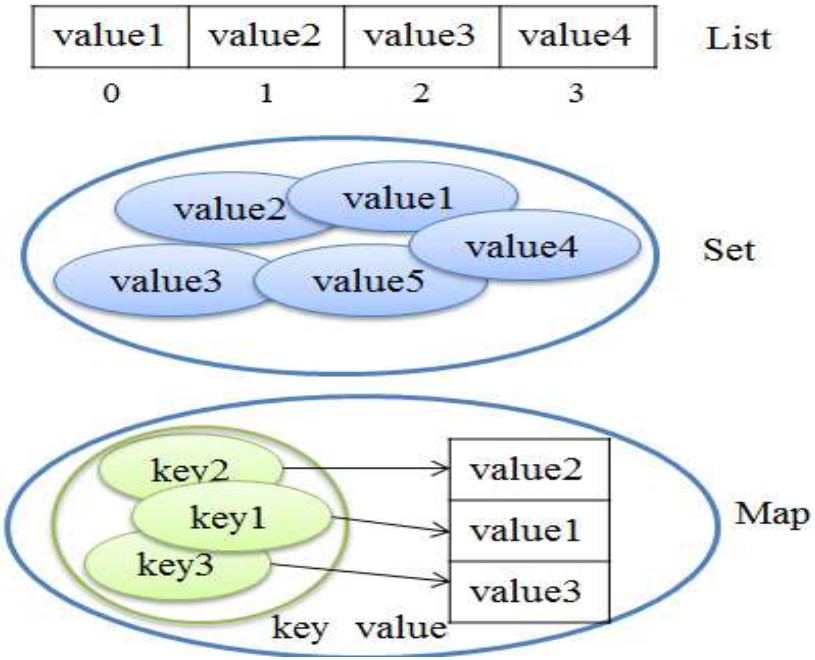
```
Set<Student> students = new TreeSet<>(new ComparatorNameAge());
```

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。

- Map用于保存具有映射关系的数据，它们以键值对<key,value>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。



- 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。

【例8-8】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。

- **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。

（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()+")");
}
```

(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。

- Entry中有3个方法，对每个键值对进行操作：
  - Object getKey()：返回Entry对象的key值。
  - Object getValue()：返回Entry对象的value值。
  - setValue(Object value)：设置Entry对象的value值，并将新值返回。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

（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()+")");
}
```

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。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

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”

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

ipConfig.properties

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/ums
user=root
password=1234
```

database.properties

- Properties类的常用方法：
  （1）load()
  - void load(InputStream inStream)：用于读取属性文件的内容。参数inStream代表了指向配置文件的输入流（关于“流”将会在第12章详细介绍）。load()方法将属性文件中的<key,value>的键值对添加在Properties中（Properties不保证键值对的次序）。

  （2）getProperty ()
  - String getProperty(String key)：获取key所对应的value。在Properties中，key和value都是String类型，不能是其他对象（HashMap中是Object）。

  （3）setProperty()
  - Object setProperty(String key, String value)：设置属性值，类似于Map中的put()方法。

【例8-9】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"));
}
```

- java.util.Collections是Java提供的操作List、Set、Map等集合的工具类
  - 所有的方法都是静态方法
  - 对集合类功能进行补充
  - 对集合进行再包装，将线程不安全的集合包装为线程安全的集合等。

8.6.1 集合功能的增补

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

static <T extends Comparable<? super T>> void	**sort**(List<T> list) Sorts the specified list into ascending order, according to the **natural ordering** of its elements.
static <T> void	**sort**(List<T> list, Comparator<? super T> c) Sorts the specified list according to the order induced by the specified comparator.
static <T> int	**binarySearch**(List<? extends Comparable<? super T>> list, T key) Searches the specified list for the specified object using the binary search algorithm.
static <T> int	**binarySearch**(List<? extends T> list, T key, Comparator<? super T> c) Searches the specified list for the specified object using the binary search algorithm.
static void	**reverse**(List<?> list) Reverses the order of the elements in the specified list.
static void	**shuffle**(List<?> list) Randomly permutes the specified list using a default source of randomness.
static int	**frequency**(Collection<?> c, Object o) Returns the number of elements in the specified collection equal to the specified object.

8.6.2 多线程封装

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

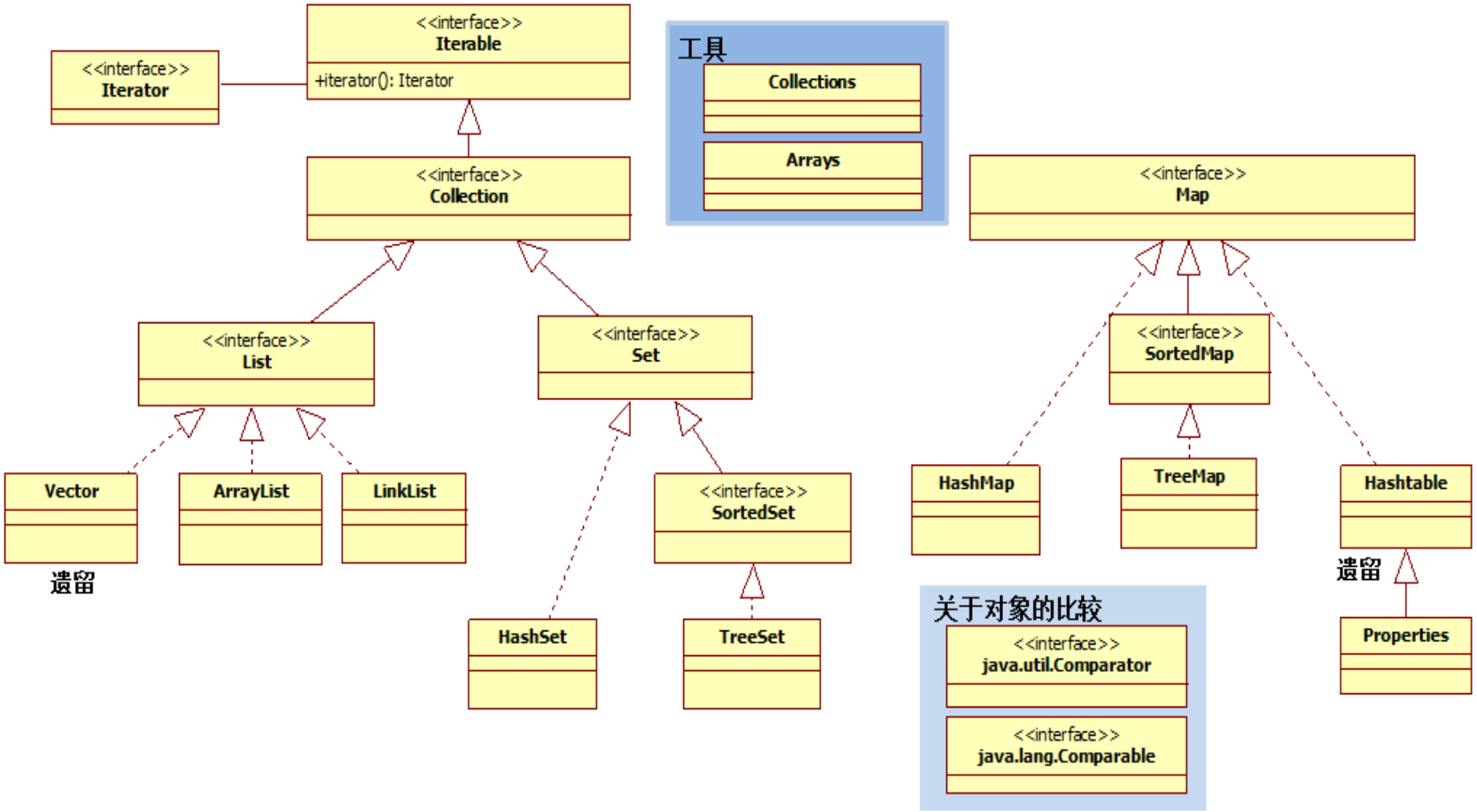
- 在集合框架中，ArrayList、HashSet、TreeSet、HashMap等都是线程不安全的，Collections类提供了多个synchronizedXxx()方法，将指定集合包装为线程同步安全的集合，从而使集合可以工作在并发访问的环境中。

`static <T> Collection<T>`	**`synchronizedCollection(Collection<T> c)`** Returns a synchronized (thread-safe) collection backed by the specified collection.
`static <T> List<T>`	**`synchronizedList(List<T> list)`** Returns a synchronized (thread-safe) list backed by the specified list.
`static <K,V> Map<K,V>`	**`synchronizedMap(Map<K,V> m)`** Returns a synchronized (thread-safe) map backed by the specified map.

```
List<String> list =
     Collections.synchronizedList(new ArrayList<String>());

Map<String,String> map =
     Collections.synchronizedMap(new HashMap<String,String>());
```

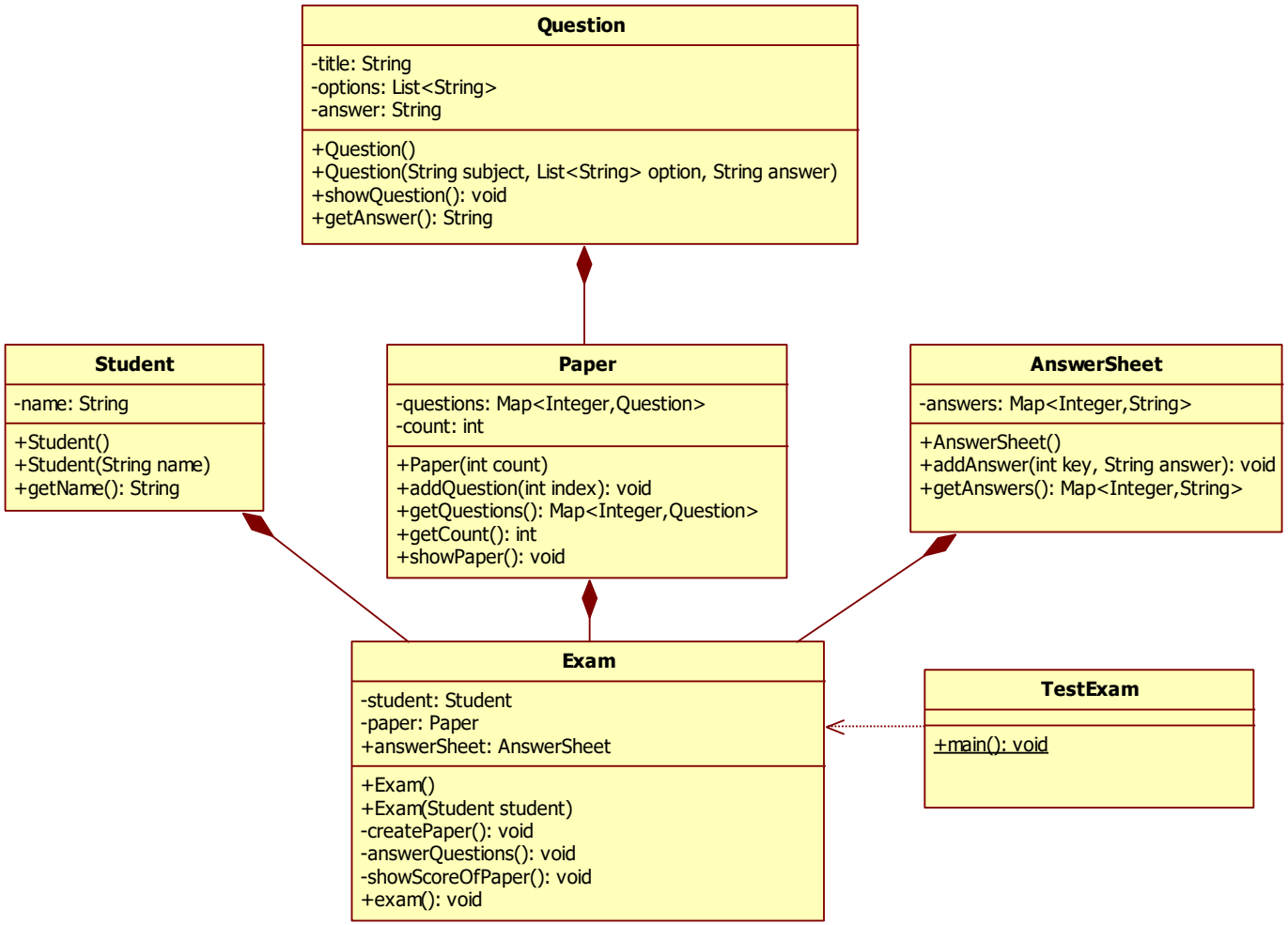
8.7 回首Java集合框架



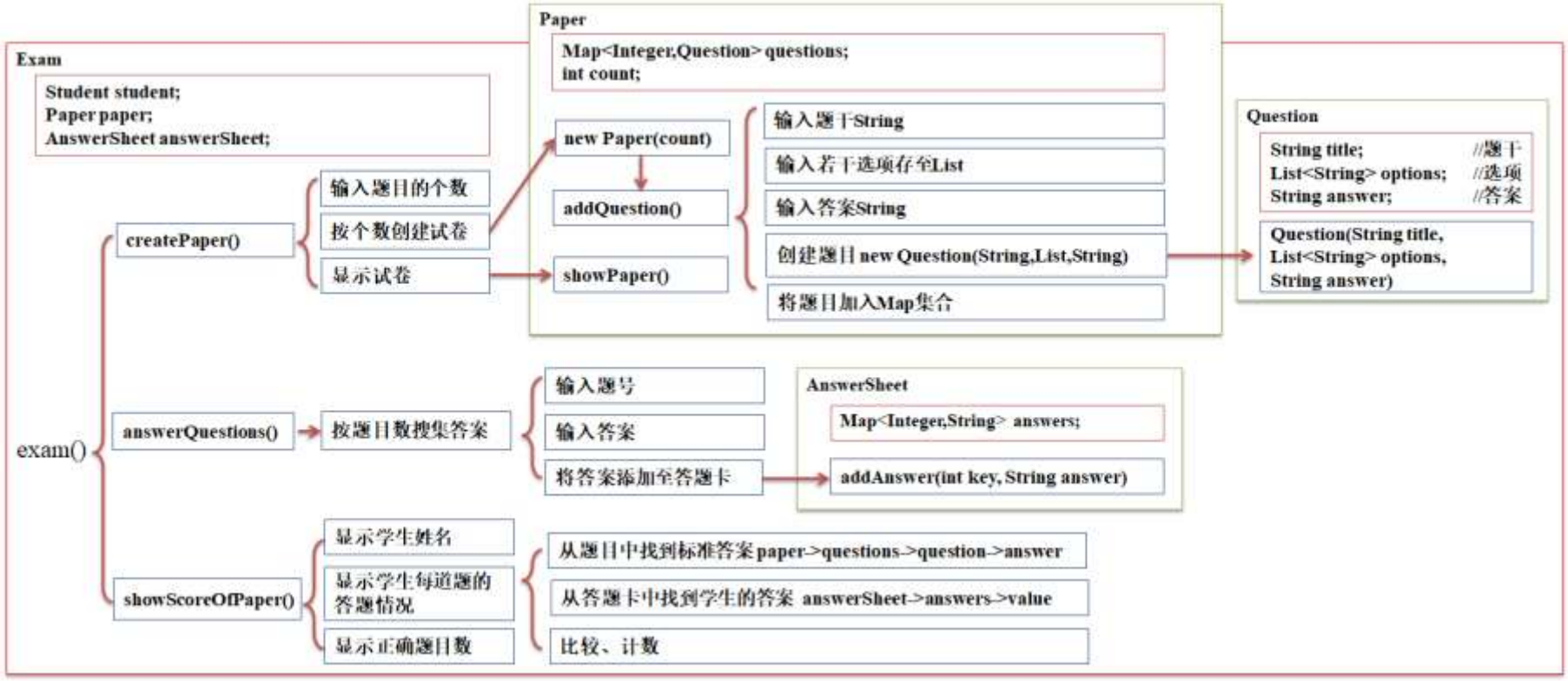
8.7 回首Java集合框架

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

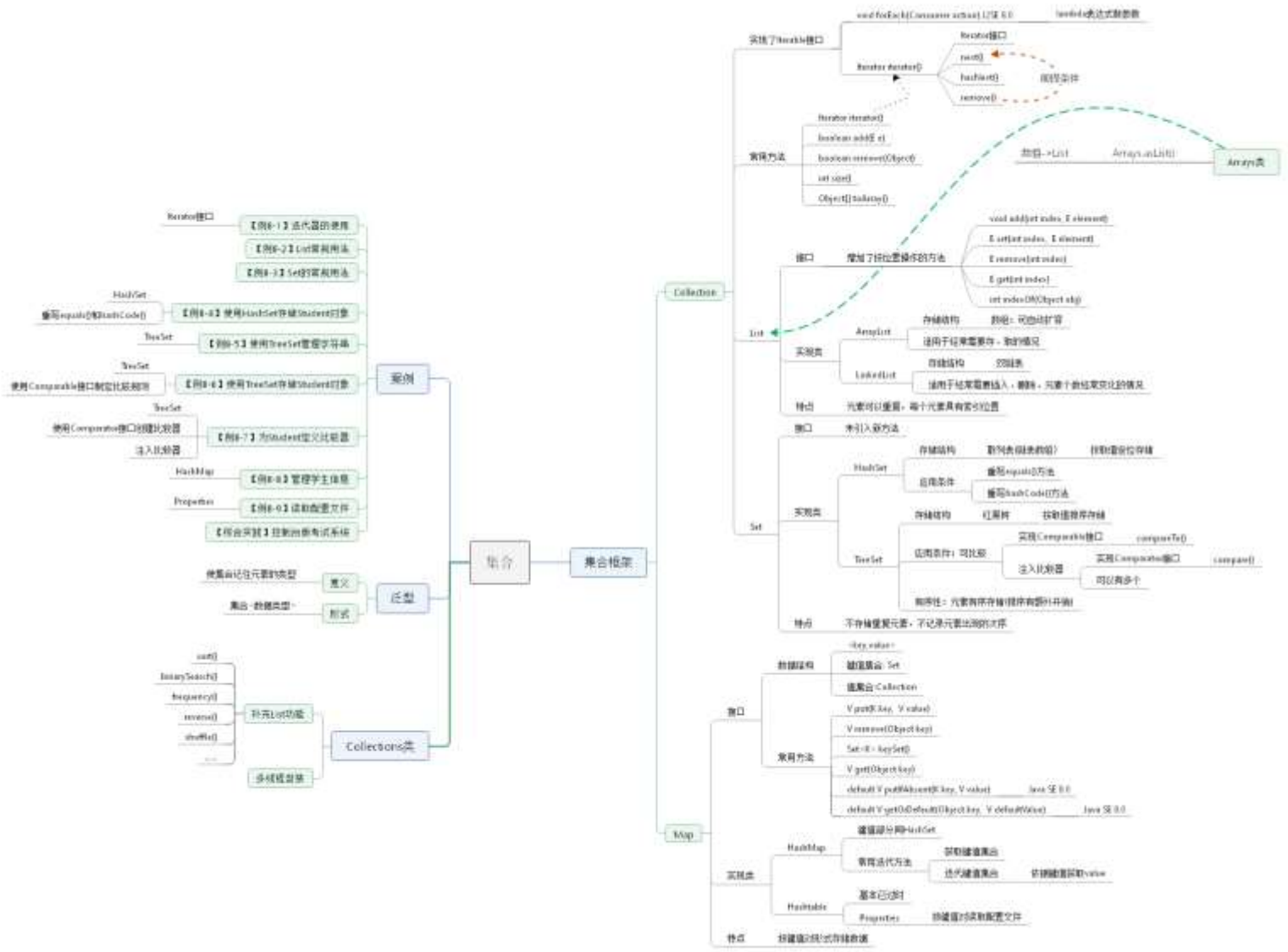
接口	实现类	存储方式	优点	缺点	其他
List 有序	ArrayList	数组	按位置索引存取快	插入、删除、查找慢	
	LinkedList	双链表	插入、删除快	按位置存取、查找慢	
Set 无序	HashSet	散列表	插入、删除、查找快		需重写equals()和hashCode()方法
	TreeSet	二叉排序树	插入、删除、查找快 有序排列	速度慢于HashSet	需制定比较规则
Map 映像	HashMap	散列表	插入、删除、查找快	迭代效率较低	键需重写equals()和hashCode()方法
	Properties	散列表	读取属性文件便利		键、值均为String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


Question
-title: String
-options: List<String>
-answer: String
+Question()
+Question(String subject, List<String> option, String answer)
+showQuestion(): void
+getAnswer(): String
Student
-name: String
+Student()
+Student(String name)
+getName(): String
Paper
-questions: Map<Integer,Question>
-count: int
+Paper(int count)
+addQuestion(int index): void
+getQuestions(): Map<Integer,Question>
+getCount(): int
+showPaper(): void
AnswerSheet
-answers: Map<Integer,String>
+AnswerSheet()
+addAnswer(int key, String answer): void
+getAnswers(): Map<Integer,String>
Exam
-student: Student
-paper: Paper
+answerSheet: AnswerSheet
+Exam()
+Exam(Student student)
-createPaper(): void
-answerQuestions(): void
-showScoreOfPaper(): void
+exam(): void
TestExam
+main(): void

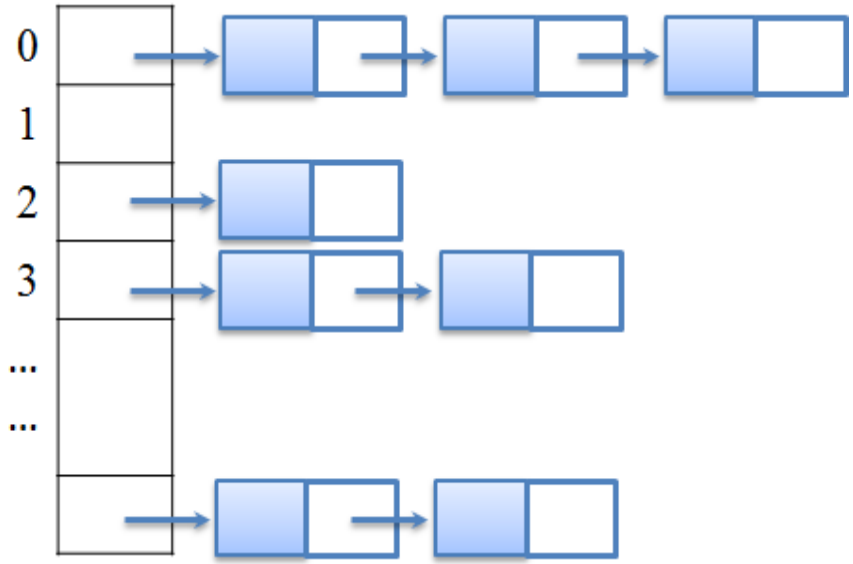
8.8 综合实践---控制台版考试系统



本章思维导图


集合
集合框架
Collection
List
Set
Map
Collections类
泛型
案例
Iterator接口
HashSet
TreeSet
HashMap
Properties
Arrays类
ArrayList
LinkedList
HashSet
TreeSet
HashMap
Hashtable
Properties

向HashSet中添加对象的过程



(1) 对象定位-- hashCode()方法

(2) 对象查重-- equals()方法

（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。

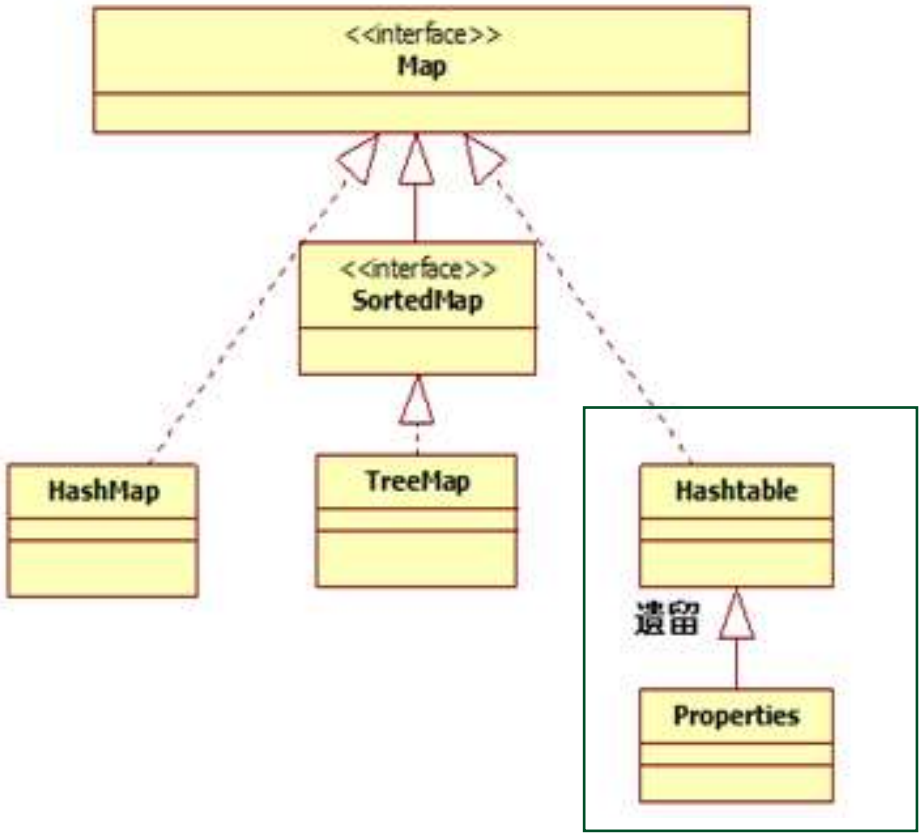
（2）利用散列码得到对象在散列表中的存储位置p。

（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。

（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。

- 如果某个equals()比较的结果为true，说明obj对象已存在，将其舍弃。
- 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。

Hashtable及其子类Properties



- Hashtable源于JDK 1.0时代，它与JDK 1.2中的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，也不再使用Hashtable，利用Collections工具类可以将HashMap包装成线程安全的。
- Properties是Hashtable的子类，它也实现了Map接口。使用它处理属性文件。

【题目2】统计一个字符串中每个单词出现的次数，并按照出现次数从高到低的次序打印统计结果。

Abstract-This paper presents an overview of the field of recommender systems and describes the current generation of recommendation methods that are usually classified into the following three main categories: content-based, collaborative, and hybrid recommendation approaches. This paper also describes various limitations of current recommendation methods and discusses possible extensions that can improve recommendation capabilities and make recommender systems applicable to an even broader range of applications. These extensions include, among others, an improvement of understanding of users and items, incorporation of the contextual information into the recommendation process, support for multcriteria ratings, and a provision of more flexible and less intrusive types of recommendations.

1、分词&统计每个词出现的次数

- 1）利用Arrays.asList()方法将split()分词后的结果从数组转换到List集合中。
- 2）利用Set集合包装List集合，实现List中的单词进行去重。
- 3）对Set进行遍历，使用Collections类中的frequency()方法统计List中每个单词出现的次数，将统计的结果存储在Map<String, Integer>中。

2、对统计的结果进行排序

- 1）将Map中的键值对存储至List<Result>集合中，Result类包含String word，int times两个属性。
- 2）使用Collections类的sort()方法对List集合进行排序，排序规则由Result类实现Comparable接口定义。