Java集合框架

主讲老师：申雪萍

主要内容

- 集合框架的价值和意义
- 集合框架设计时需要满足的几个目标
- Java 集合框架概述
- Collection 接口
- Set
- List
- Map
- 集合类对比
- Iterator 接口
- 泛型集合

编程中的问题：

- 假定一个班容纳20名学员，如何存储？
- 如何存储每天的新闻信息？
  - 每天的新闻总数不确定，太少浪费空间，太多空间不足
- 如何存储计算机专业课程的代码与课程信息？
- 如果访问多，增删少用什么数据结构？
- 如何增删多，访问少用什么数据结构？
- 多线程环境下，哪些是线程安全，哪些不是？如何选择？

如果并不知道程序运行时会需要多少对象，或者需要更复杂方式存储对象——可以考虑使用Java集合框架

问题

- 集合框架的价值和意义？
- 集合框架设计时需要满足的目标?

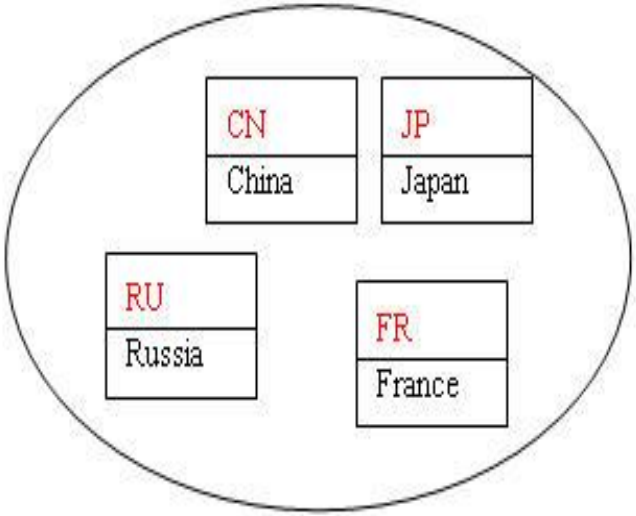
集合框架的价值和意义

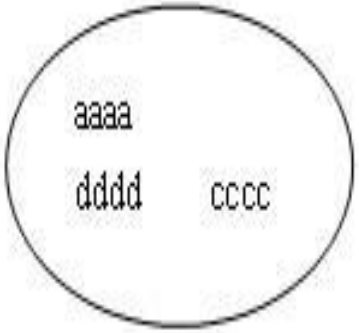
Java集合框架为我们提供了一套性能优良、使用方便的接口和类，我们不必再重新发明轮子，只需学会如何使用它们，就可处理实际应用中问题

- 集合框架对编程有什么好处呢？
  - 提高程序设计效率。
  - 提高程序速度和质量。集合框架通过提供对有用的数据结构和算法的高性能和高质量的实现，使你的程序速度和质量得到提高。
  - 集合框架鼓励软件的复用。对于遵照标准集合框架接口的新的数据结构是可复用的。

数学中集合框架包含的内容（1）

- Collection 接口存储一组不唯一，无序的对象
- List 接口存储一组不唯一，有序（插入顺序）的对象
- Set 接口存储一组唯一，无序的对象
- Map接口存储一组键值对象，提供key到value的映射





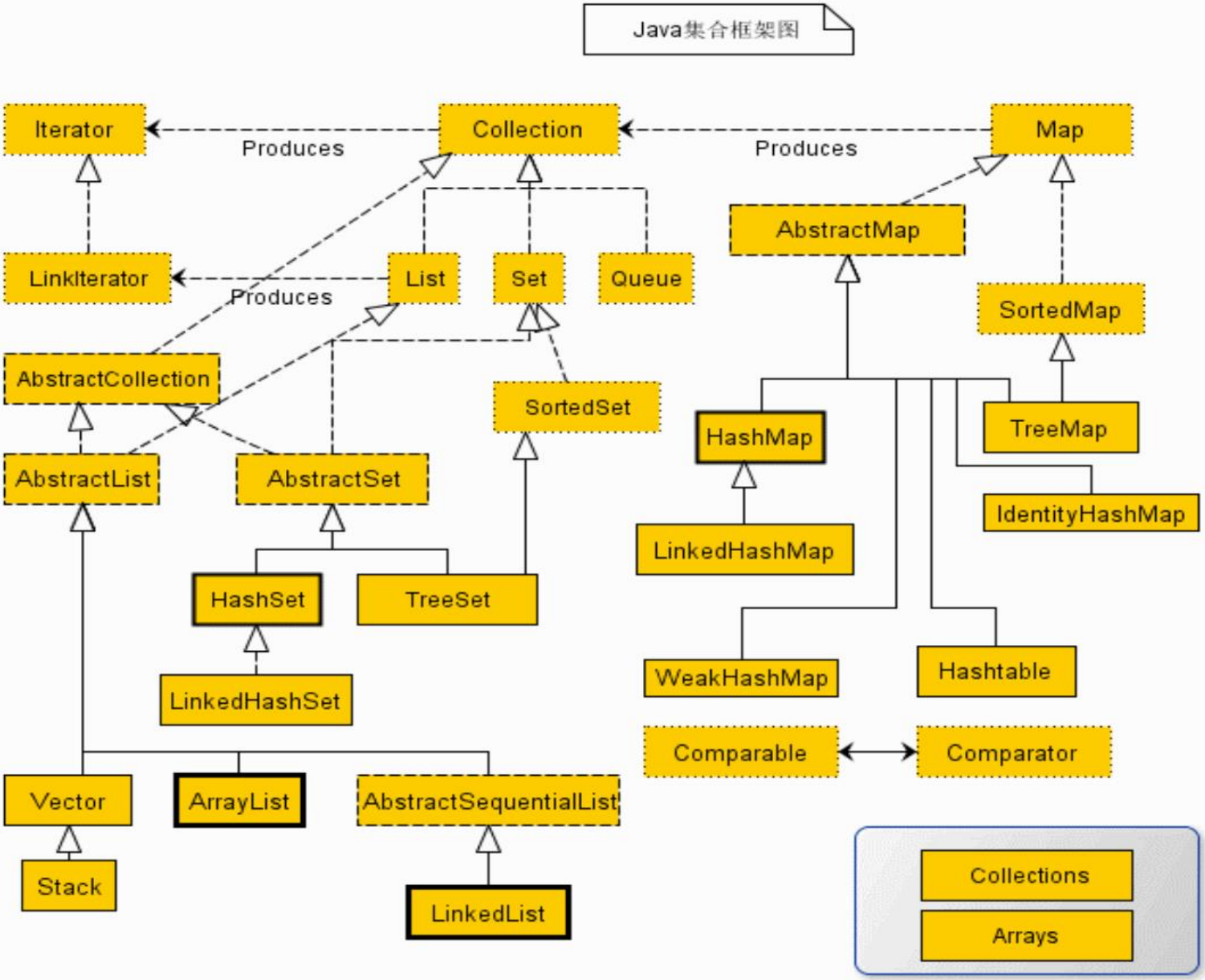
0	1	2	3	4	5	
aaaa	dddd	cccc	aaaa	eeee	dddd	

Java集合框架设计时需要满足的几个目标

- 该框架允许不同类型的集合，以类似的方式工作，具有高度的互操作性。
- 提供丰富的算法。
- 该框架必须是高性能的。基本集合（动态数组，链表，树，哈希表）的实现也必须是高效的。
- 对一个集合的扩展和适应必须是简单的。

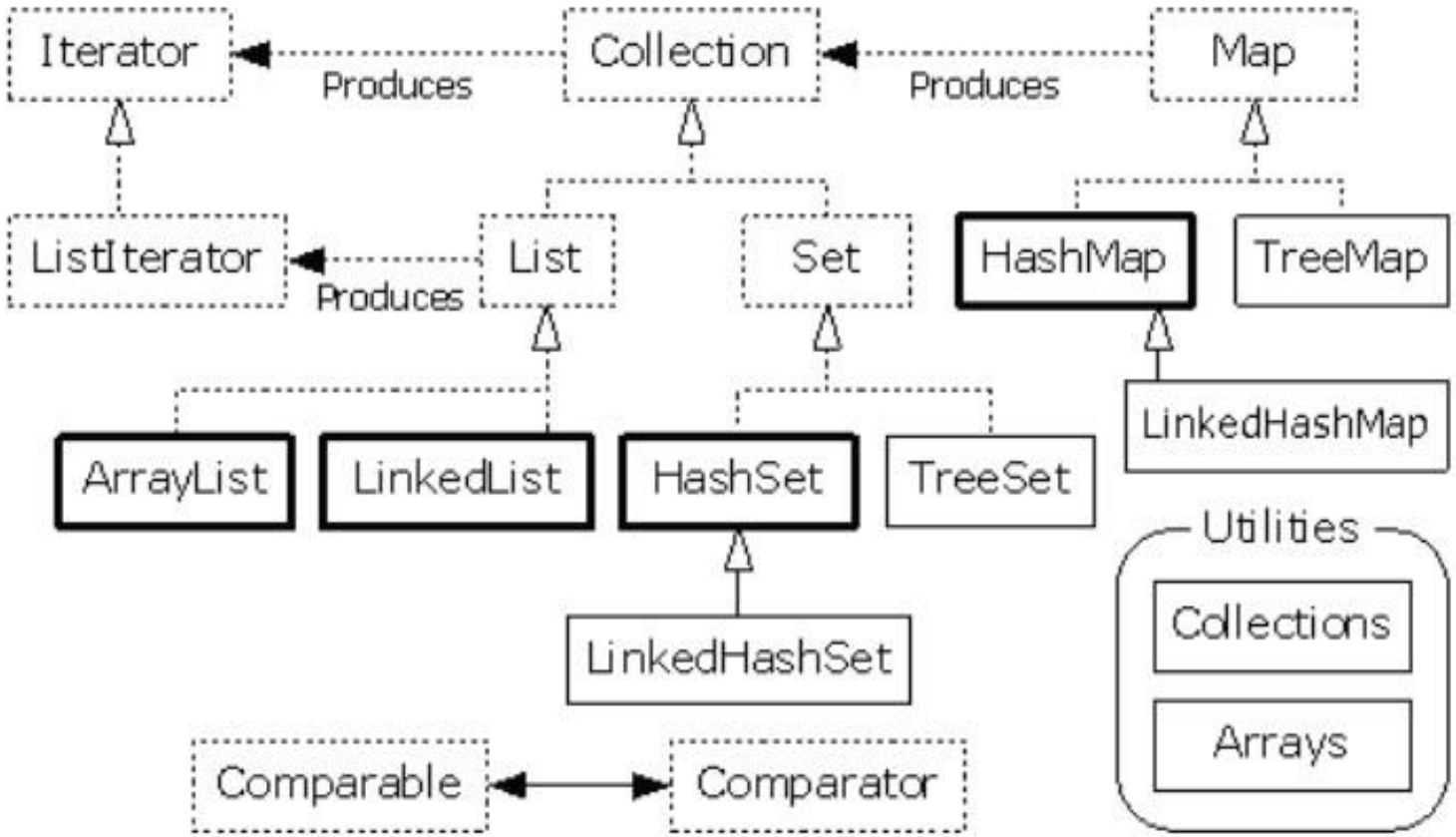
Java集合框架

- Java 集合框架中主要封装的是典型的数据结构和算法，如动态数组、双向链表、队列、栈、Set、Map 等
  - Java 集合就像一种容器，可以把**多个对象的引用**放入容器中。
  - Java 集合类可以用于存储数量不等的多个对象，还可用于保存具有映射关系的关联数组。


Java集合框架图
Iterator
Produces
Collection
Produces
Map
LinkIterator
Produces
List
Set
Queue
AbstractMap
SortedMap
AbstractCollection
SortedSet
HashMap
TreeMap
AbstractList
AbstractSet
IdentityHashMap
LinkedHashMap
HashSet
TreeSet
WeakHashMap
Hashtable
LinkedHashSet
Comparable
Comparator
Vector
ArrayList
AbstractSequentialList
Stack
LinkedList
Collections
Arrays

简化图

简化图：



说明（1）：

1. 所有集合类都位于 java.util 包下。Java的集合类主要由两个接口派生而出：Collection 和 Map，Collection 和 Map 是 Java 集合框架的根接口，这两个接口又包含了一些子接口或实现类。
2. 集合接口：短虚线表示，表示不同集合类型，是集合框架的基础。
3. 抽象类：长虚线表示，对集合接口的部分实现。可扩展为自定义集合类。
4. 实现类：实线表示，对接口的具体实现。

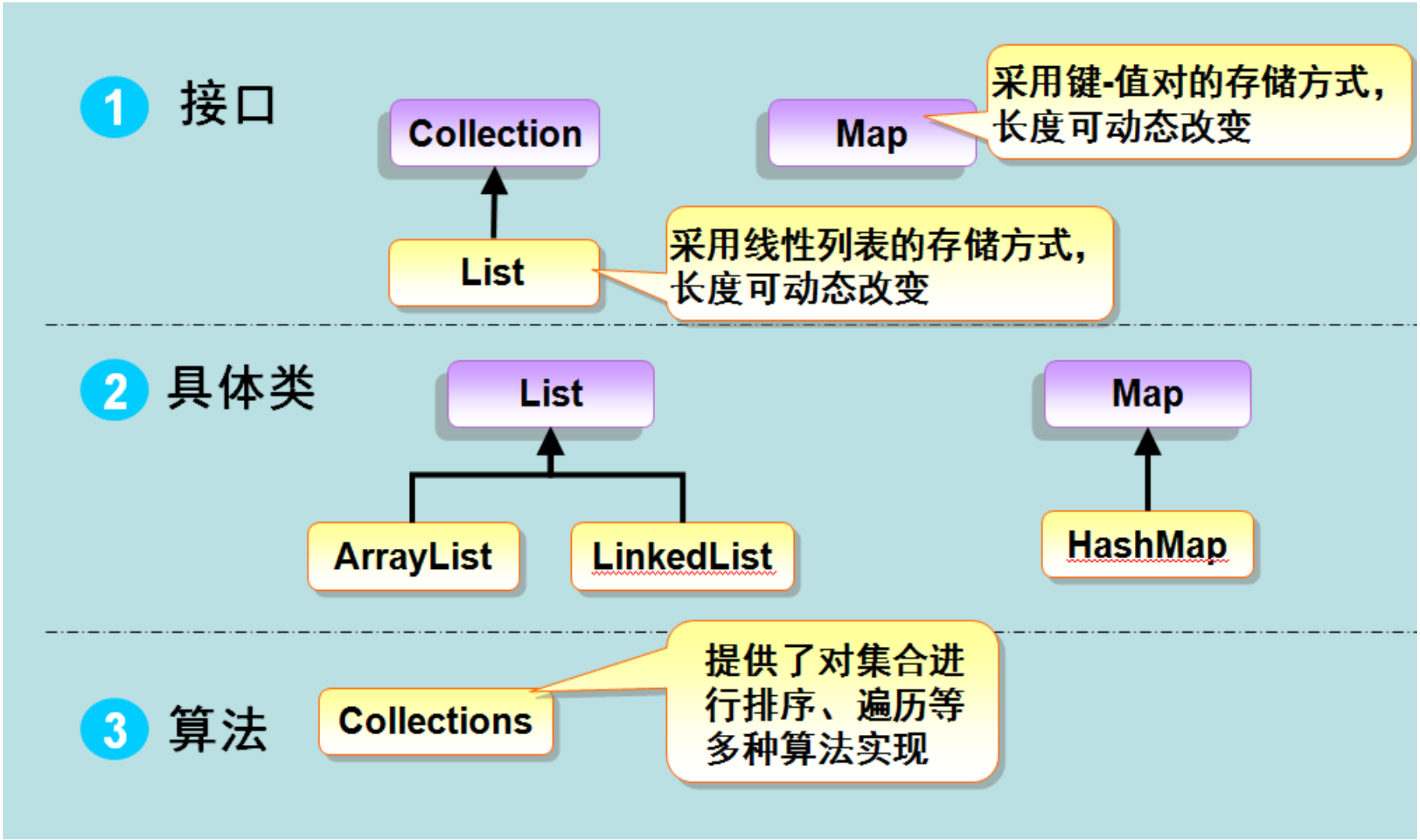
说明（2）：

5. Collection 接口是一组允许重复的对象。
6. Set 接口继承 Collection，集合元素不重复。
7. List 接口继承 Collection，允许重复，维护元素插入顺序。
8. Map接口是键一值对象，与Collection接口没有什么关系。
9. Set、List 和 Map 可以看做集合的三大类：
   - List 集合是有序集合，集合中的元素可以重复，访问集合中的元素可以根据元素的索引来访问。
   - Set 集合是无序集合，集合中的元素不可以重复，访问集合中的元素只能根据元素本身来访问（也是集合里元素不允许重复的原因）。
   - Map 集合中保存 Key-value 对形式的元素，访问时只能根据每项元素的 key 来访问其 value。

将集合框架挖掘处理，可以分为以下几个部分

1. 数据结构：List列表、Queue队列、Deque双端队列、Set集合、Map映射
2. 比较器：Comparator比较器和Comparable排序接口
3. 算法：Collections常用算法类、Arrays静态数组的排序、查找算法
4. 迭代器：Iterator通用迭代器、ListIterator针对 List 特化的迭代器

Java集合框架包含的内容：接口、具体类和算法



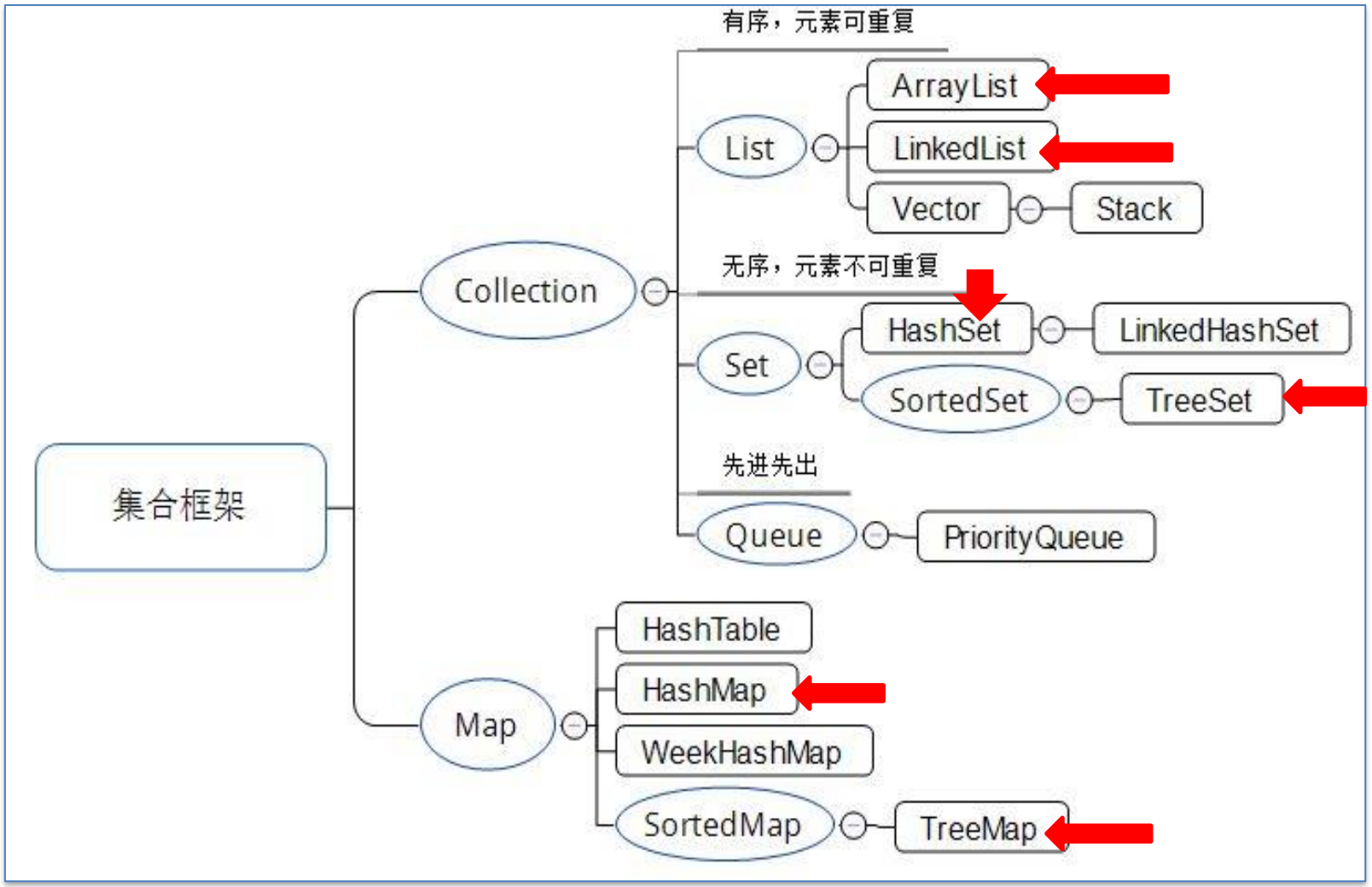
Java集合框架包含的内容：接口、具体类和算法

- 集合框架：是为表示和操作集合而规定的一种统一的、标准的体系结构。
- 集合框架包含三大块内容：对外的接口、接口的实现和对集合运算的算法。
  - 接口：即表示集合的抽象数据类型。接口提供了让我们对集合中所表示的内容进行单独操作的可能。
  - 实现：也就是集合框架中接口的具体实现。实际它们就是那些可复用的数据结构。
  - 算法：在一个实现了某个集合框架中的接口的对象上，完成某种有用的计算的方法，例如查找、排序等。

集合框架中的接口(四个主要接口)

- Collection：集合层次中的根接口，JDK没有提供这个接口直接的实现类。
- Set：不能包含重复的元素。SortedSet是一个按照升序排列元素的Set。
- List：是一个有序的集合，可以包含重复的元素。提供了按索引访问的方式。
- Map：包含了key-value对。Map不能包含重复的key。SortedMap是一个按照升序排列key的Map

Java 集合框架——数据结构



集合类特点

- **只容纳对象**。这一点和数组不同，数组可以容纳基本数据类型数据和对象。
  - 如果集合类中想使用基本数据类型，又想利用集合类的灵活性，可以把基本数据类型数据封装成该数据类型的对象，然后放入集合中处理。
  - 集合类容纳的对象都是Object类的实例，一旦把一个对象置入集合类中，它的类信息将丢失，这样设计的目的是为了集合类的通用性。
  - 因为Object类是所有类的祖先，所以可以在这些集合中存放任何类的对象而不受限制，但是切记在使用集合成员之前必须对它重新造型。

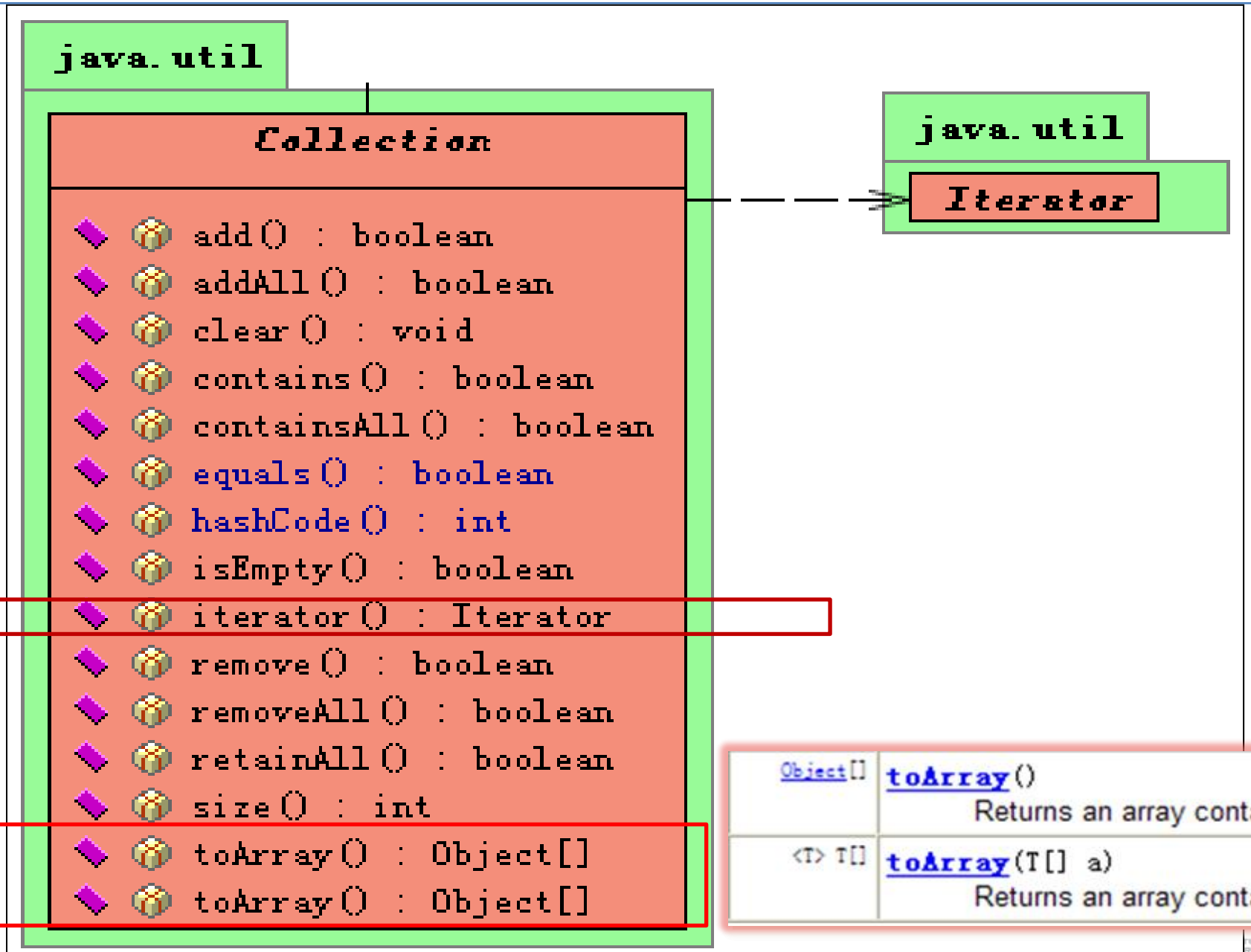
主要内容

- 集合框架的价值和意义
- 集合框架设计时需要满足的几个目标
- Java 集合框架概述
- Collection 接口
- Set
- List
- Map
- 集合类对比
- Iterator 接口
- 泛型集合

Collection接口

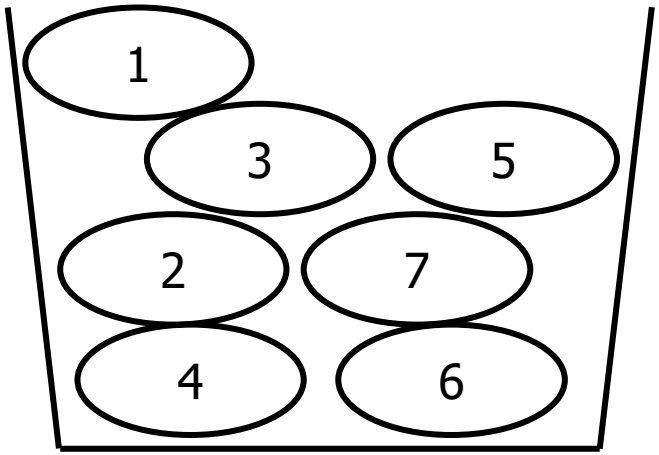
- Collection 接口是 List、Set 和 Queue 接口的父接口，该接口里定义的方法既可用于操作 Set 集合，也可用于操作 List

Collection接口

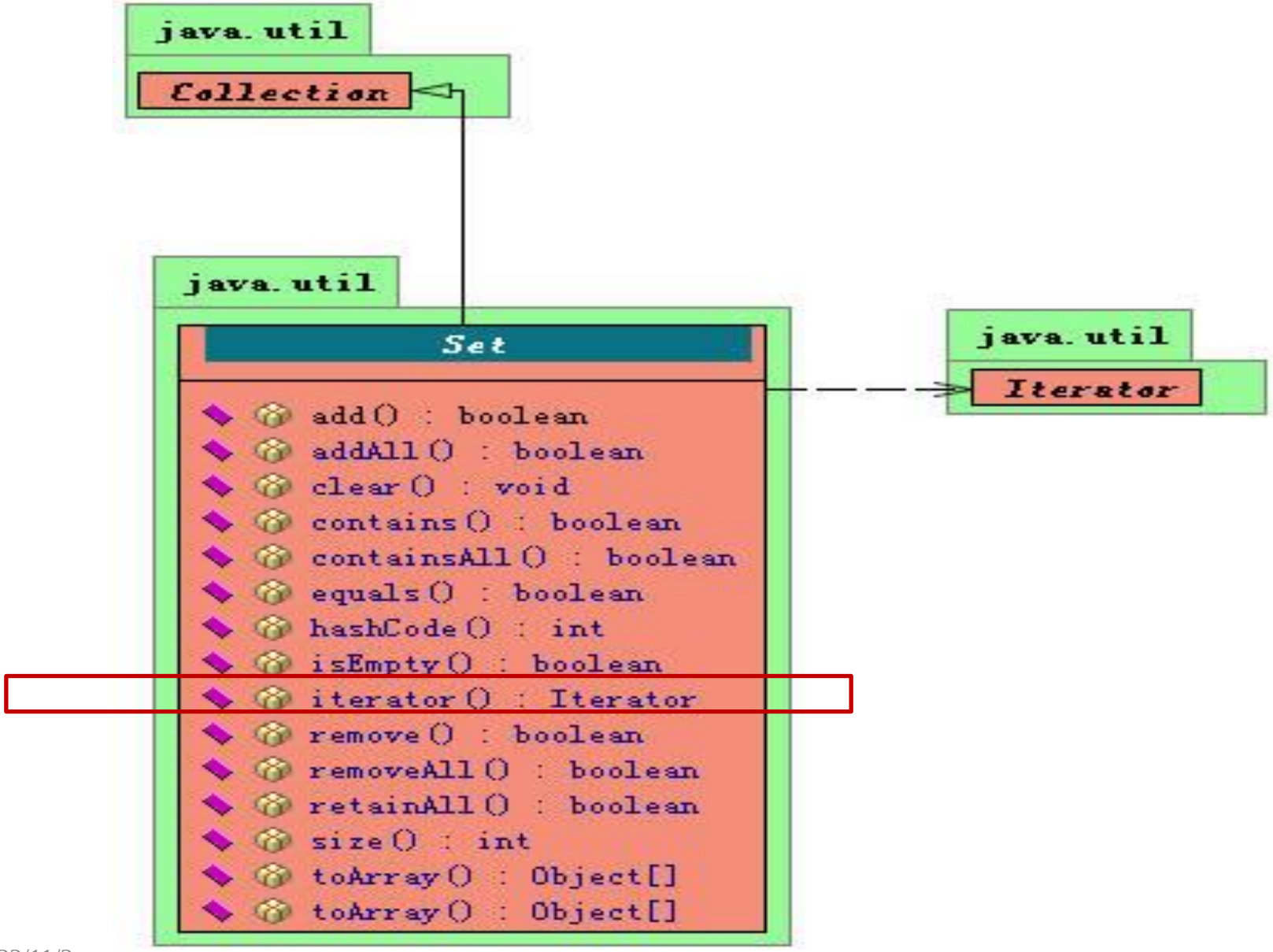


Set

- Set中的元素必须唯一。
- 添加到Set中的元素必须定义equals方法，以提供算法来判断欲添加进来的对象是否与已经存在的某对象相等，从而建立对象的唯一性。
- 实现Set接口的类有HashSet，TreeSet



Set

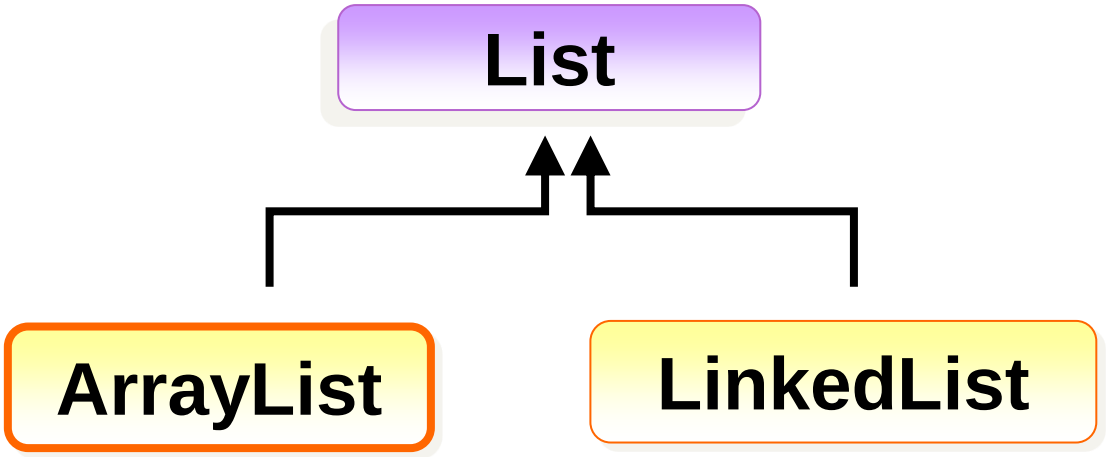

java.util
Collection
java.util
Set
add() : boolean
addAll() : boolean
clear() : void
contains() : boolean
containsAll() : boolean
equals() : boolean
hashCode() : int
isEmpty() : boolean
iterator() : Iterator
remove() : boolean
removeAll() : boolean
retainAll() : boolean
size() : int
toArray() : Object[]
toArray() : Object[]
java.util
Iterator

Set举例

```
public class SetDemo {
    public static void main(String[] argv) {
        HashSet<String> h = new HashSet<String>();
        //也可以Set h=new HashSet()
        h.add("One");
        h.add("Two");
        h.add("One"); // DUPLICATE
        h.add("Three");
        Iterator<String> it = h.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

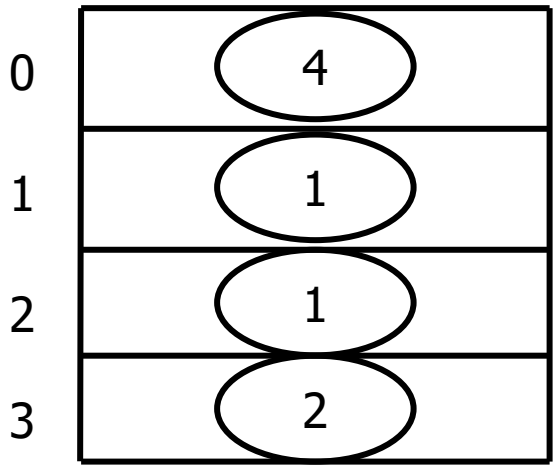
```
Three
One
Two
```

List接口实现类



List

- List的明显特征是它的元素都有一个确定的顺序。
- 实现它的类有ArrayList和LinkedList。
  - ArrayList内存中是顺序存储的。
  - LinkedList内存中是以链表方式存储的



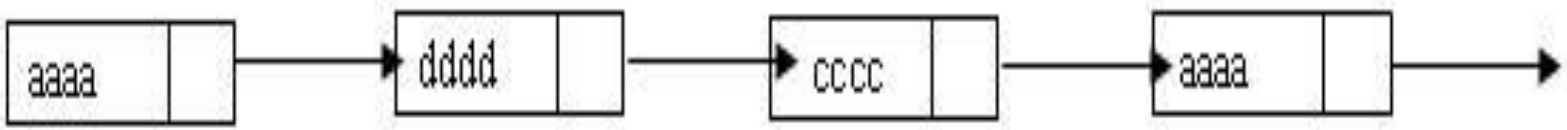
List (列表)

List接口实现类

- **ArrayList实现了长度可变的数组**，在内存中分配连续的空间。遍历元素和随机访问元素的效率比较高

0	1	2	3	4	5	
aaaa	dddd	cccc	aaaa	eeee	dddd	

- **LinkedList采用链表存储方式**。插入、删除元素时效率比较高



ArrayList集合类

- 存储多条狗狗信息，获取狗狗总数，逐条打印出各条狗狗信息
- 通过List接口的实现类ArrayList实现该需求
  - 元素个数不确定
  - 要求获得元素的实际个数
  - 按照存储顺序获取并打印元素信息

示例代码：

```
public class Dog {
    private String name;
    private String strain;
    public Dog() {
        super();
    }
    public Dog(String name, String type) {
        this.name = name;
        this.strain = type;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getStrain() {
        return strain;
    }
    public void setStrain(String type) {
        this.strain = type;
    }}
```

```java
import java.util.*;
public class Test1 {
    public static void main(String[] args) {
        Dog ououDog = new Dog("欧欧", "雪娜瑞");
        Dog yayaDog = new Dog("亚亚", "拉布拉多");
        Dog meimeiDog = new Dog("美美", "雪娜瑞");
        Dog feifeiDog = new Dog("菲菲", "拉布拉多");

        List<Dog> dogs = new ArrayList<Dog>(); // 泛型
        dogs.add(ououDog);
        dogs.add(yayaDog);
        dogs.add(meimeiDog);
        dogs.add(2, feifeiDog); // 添加feifeiDog到指定位置

        System.out.println("共计有" + dogs.size() + "条狗狗。");

        System.out.println("分别是：");
        for (int i = 0; i < dogs.size(); i++) {
            Dog dog = (Dog) dogs.get(i);
            System.out.println(dog.getName() + "\t" + dog.getStrain());
        }}}
```

```java
import java.util.*;
public class ArrayListDemo {
    public static void main(String[] argv) {
        ArrayList<Object> al = new ArrayList<Object>();
        // Add lots of elements to the ArrayList...
        al.add(new Integer(11));
        al.add(new Integer(12));
        al.add(new Integer(13));
        al.add(new String("hello"));
        // First print them out using a for loop.
        System.out.println("Retrieving by index:");
        for (int i = 0; i < al.size(); i++) {
            System.out.println("Element " + i + " = "
                                        + al.get(i));
        }
        for (Object obj : al) { // 增强型的for循环
            System.out.println(obj);
        }
    }
}
```

```java
public class LinkedListDemo {
    public static void main(String[] argv) {
        LinkedList<Object> l = new LinkedList<Object>();
        l.add(new Object());
        l.add("Hello");
        l.add("zhangsan");
        ListIterator<Object> li = l.listIterator();
        while (li.hasNext())
            System.out.println(li.next());
        if (l.indexOf("Hello") < 0)
            System.err.println("Lookup does not work");
        else
            System.err.println("Lookup works");
    }
}
```

ArrayList集合类

- 扩充以下几部分功能
  - 删除指定位置的狗狗，如第一个狗狗
  - 删除指定的狗狗，如删除feifeiDog对象
  - 判断集合中是否包含指定狗狗
- List接口提供相应方法remove()、contains()，直接使用即可

```java
public class Test2 {
    public static void main(String[] args) {
        Dog ououDog = new Dog("欧欧", "雪娜瑞");
        Dog yayaDog = new Dog("亚亚", "拉布拉多");
        Dog meimeiDog = new Dog("美美", "雪娜瑞");
        Dog feifeiDog = new Dog("菲菲", "拉布拉多");

        List<Dog> dogs = new ArrayList<Dog>(); // 泛型
        dogs.add(ououDog);
        dogs.add(yayaDog);
        dogs.add(meimeiDog);
        dogs.add(2, feifeiDog); // 添加feifeiDog到指定位置

        System.out.println("共计有" + dogs.size() + "条狗狗。");
        System.out.println("分别是：");
        for (int i = 0; i < dogs.size(); i++) {
            Dog dog = (Dog) dogs.get(i);
            System.out.println(dog.getName() + "\t" + dog.getStrain());
        }
```

```
        dogs.remove(0);
        dogs.remove(feifeiDog);

        System.out.println("\n删除之后还有" + dogs.size() + "条狗狗。");

        if (dogs.contains(meimeiDog))
            System.out.println("\n集合中包含美美的信息");
        else
            System.out.println("\n集合中不包含美美的信息");
    }
}
```

ArrayList接口常用方法

方法名	说　明
boolean add(Object o)	在列表的末尾顺序添加元素，起始索引位置从0开始
void add(int index,Object o)	在指定的索引位置添加元素。索引位置必须介于0和列表中元素个数之间
int size()	返回列表中的元素个数
Object get(int index)	返回指定索引位置处的元素。取出的元素是Object类型，使用前需要进行强制类型转换
boolean contains(Object o)	判断列表中是否存在指定元素
boolean remove(Object o)	从列表中删除元素
Object remove(int index)	从列表中删除指定位置元素，起始索引位置从0开始

LinkedList集合类

- 在集合任何位置（头部、中间、尾部）添加、获取、删除狗狗对象
- 插入、删除操作频繁时，可使用LinkedList来提高效率
- LinkedList还额外提供对头部和尾部元素进行添加和删除操作的方法

示例代码

```
public class Test3 {
    public static void main(String[] args) {
        Dog ououDog = new Dog("欧欧", "雪娜瑞");
        Dog yayaDog = new Dog("亚亚", "拉布拉多");
        Dog meimeiDog = new Dog("美美", "雪娜瑞");
        Dog feifeiDog = new Dog("菲菲", "拉布拉多");

        LinkedList<Object> dogs = new LinkedList<Object>();
//      LinkedList<Dog> dogs = new LinkedList<Dog>();
        dogs.add(ououDog);
        dogs.add(yayaDog);
        dogs.addFirst(meimeiDog);  //// 添加meimeiDog到指定位置
        dogs.addLast(feifeiDog); // 添加feifeiDog到指定位置

        System.out.println("共计有" + dogs.size() + "条狗狗。");

        System.out.println("分别是：");
        for (int i = 0; i < dogs.size(); i++) {
            Dog dog = (Dog) dogs.get(i);
            System.out.println(dog.getName() + "\t" + dog.getStrain());
        }
```

```
Dog dogFirst= (Dog)dogs.getFirst();
System.out.println("第一条狗狗昵称是"+dogFirst.getName() );

Dog dogLast= (Dog)dogs.getLast();
System.out.println("最后一条狗狗昵称是"+dogLast.getName());

dogs.removeFirst();
dogs.removeLast();
System.out.println("共计有" + dogs.size() + "条狗狗。");

System.out.println("分别是：");
for (int i = 0; i < dogs.size(); i++) {
    Dog dog = (Dog) dogs.get(i);
    System.out.println(dog.getName() + "\t" + dog.getStrain());
}
```

LinkedList的特殊方法

方法名	说　明
void　addFirst(Object o)	在列表的首部添加元素
void　addLast(Object o)	在列表的末尾添加元素
Object　getFirst()	返回列表中的第一个元素
Object　getLast()	返回列表中的最后一个元素
Object　removeFirst()	删除并返回列表中的第一个元素
Object　removeLast()	删除并返回列表中的最后一个元素

Map接口

- 建立国家英文简称和中文全名间的键值映射，并通过key对value进行操作，应该如何实现数据的存储和操作呢？
- Map接口专门处理键值映射数据的存储，可以根据键实现对值的操作
- **最常用的实现类是HashMap**

```
public class Test4 {
    public static void main(String[] args) {
        Map<String,String> countries =
                                new HashMap<String,String>();
        countries.put("CN", "中华人民共和国");
        countries.put("RU", "俄罗斯联邦");
        countries.put("FR", "法兰西共和国");
        countries.put("US", "美利坚合众国");
        String country = (String) countries.get("CN");
        System.out.println("CN对应的国家是: " + country);
        System.out.println("Map中共有"+countries.size()+"组数据");
        countries.remove("FR");
        System.out.println("Map中包含FR的key吗？" +
                countries.containsKey("FR"));
        System.out.println( countries.keySet() ) ;
        System.out.println( countries.values() );
        System.out.println( countries );
    }
}
```

输出结果

```
CN对应的国家是：中华人民共和国
Map中共有4组数据
Map中包含FR的key吗？false
[US, RU, CN]
[美利坚合众国, 俄罗斯联邦, 中华人民共和国]
{US=美利坚合众国, RU=俄罗斯联邦, CN=中华人民共和国}
```

Map接口常用方法

方法名	说　明
Object put(Object key, Object val)	以“键-值对”的方式进行存储
Object get (Object key)	根据键返回相关联的值，如果不存在指定的键，返回null
Object remove (Object key)	删除由指定的键映射的“键-值对”
int size()	返回元素个数
Set keySet ()	返回键的集合
Collection values ()	返回值的集合
Boolean containsKey (Object key)	如果存在由指定的键映射的“键-值对”，返回true

集合类对比

- **Vector和ArrayList的异同**
  - 实现原理相同，功能相同，很多情况下可以互用
  - 两者的主要区别如下
    - **Vector线程安全，ArrayList重速度轻安全，线程非安全**
    - 长度需增长时，Vector默认增长一倍，ArrayList增长50%

集合类对比

- Hashtable和HashMap的异同
  - 实现原理相同，功能相同，在很多情况下可以互用
  - 两者的主要区别如下
    - Hashtable继承Dictionary类，HashMap实现Map接口
    - Hashtable线程安全，HashMap线程非安全
    - Hashtable不允许null值，HashMap允许null值

主要内容

- Java 集合概述
- Collection 接口
- Set
- List
- Map
- 集合类对比
- Iterator 接口
- 泛型集合

迭代器Iterator

- 问答题：
  - 如何遍历List集合呢？

Answer:

- 方法1：通过for循环和get()方法配合实现遍历
- 方法2：增强型for循环
- 方法3：通过迭代器Iterator实现遍历

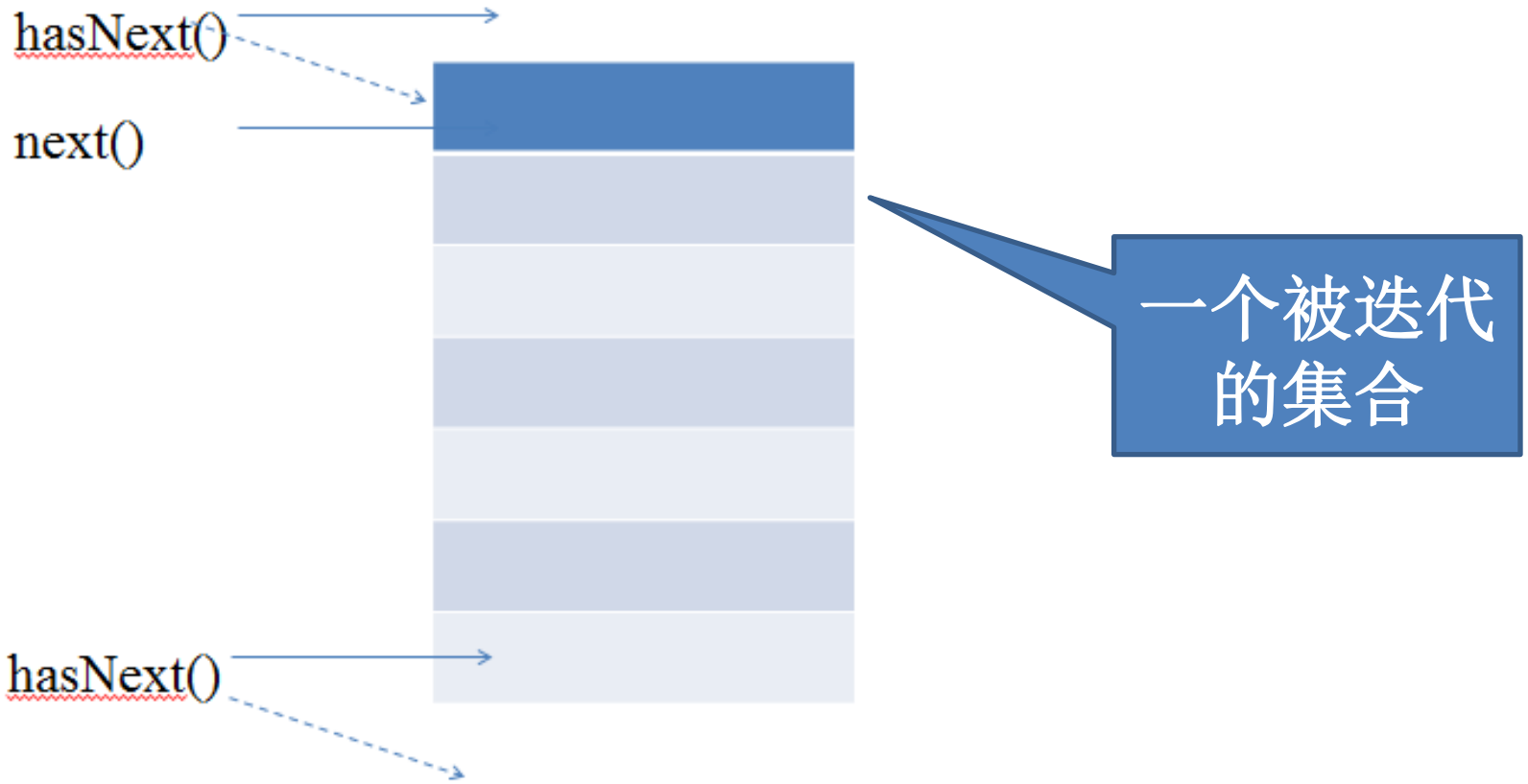
方法3：通过迭代器Iterator实现遍历

- 所有集合接口和类都没有提供相应遍历方法，而是由Iterator实现集合遍历
- Collection 接口的iterate()方法返回一个Iterator，然后通过Iterator接口的两个方法可实现遍历
  - **boolean hasNext(): 判断是否存在另一个可访问的元素**
  - **Object next(): 返回要访问的下一个元素**

使用 Iterator 接口遍历集合元素

- Iterator 接口主要用于遍历 Collection 集合中的元素，Iterator 对象也被称为迭代器
- Iterator 接口隐藏了各种 Collection 实现类的底层细节，向应用程序提供了遍历 Collection 集合元素的统一编程接口
- **Iterator 仅用于遍历集合，Iterator 本身并不提供承装对象的能力。如果需要创建 Iterator 对象，则必须有一个被迭代的集合。**

Iterator 本身并不提供承装对象的能力



使用 Iterator 接口遍历集合元素

Method Summary

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

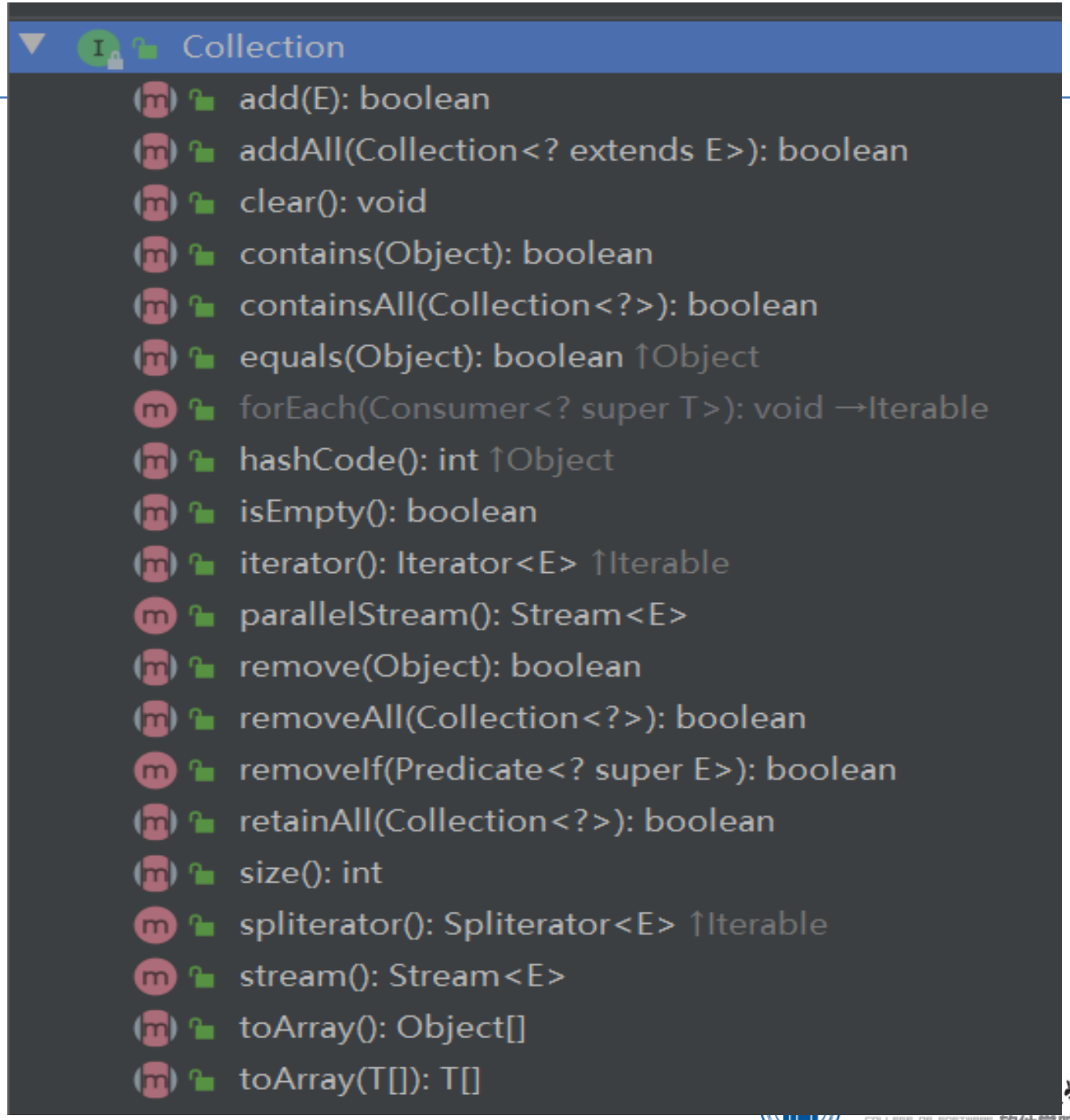
Java中Collection和Collections的区别

- Collection是一个顶层接口，绝大多数常用的容器类都实现了Collection接口，里面定义的容器类的基本方法。

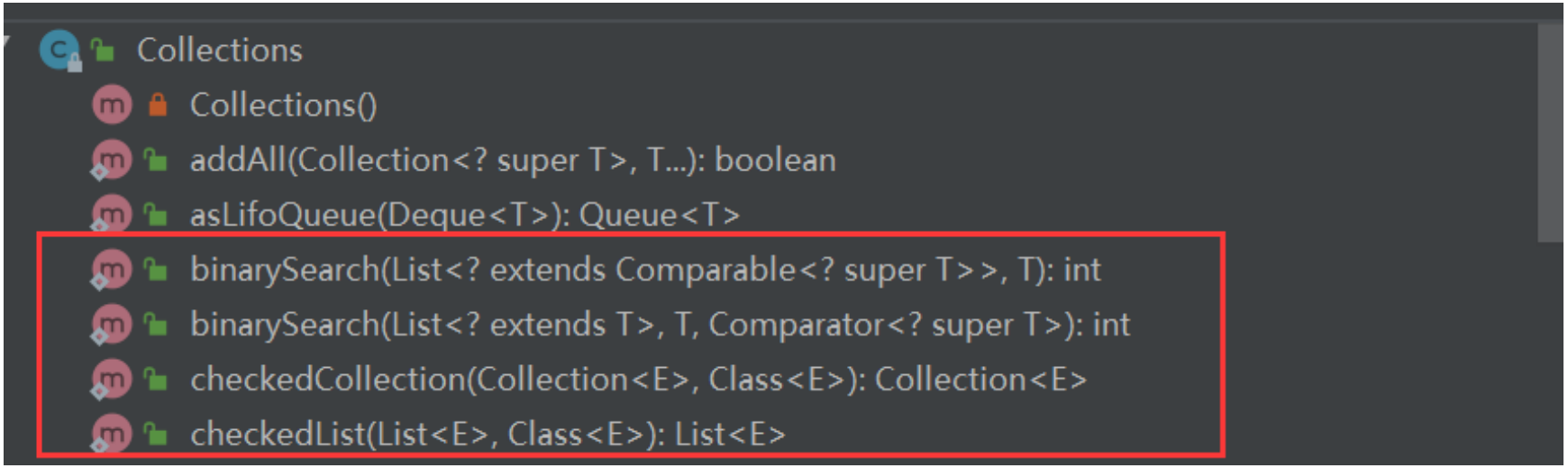
- Collections是一个工具类，此类不能实例化，服务于Java的Collection框架。他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作

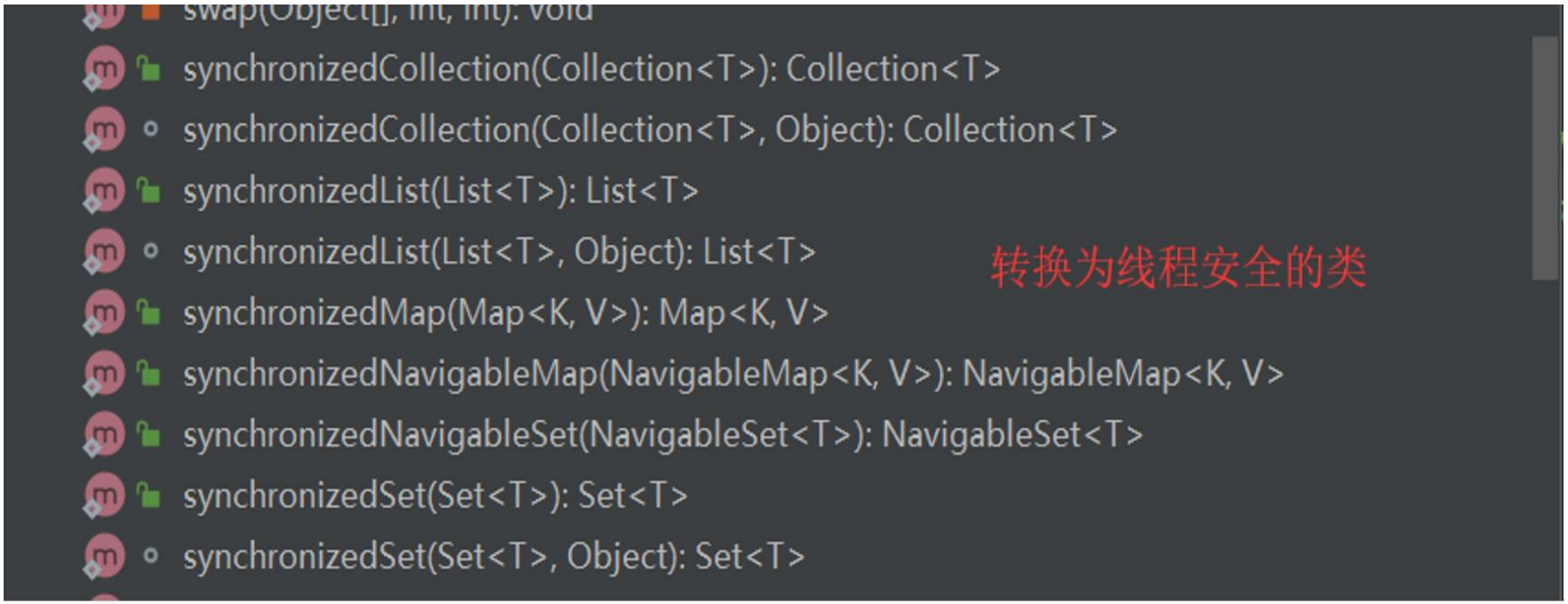
Collection接口

Collection接口的意义：是为各种具体的集合提供了最大化的统一操作方式



Collections是一个工具类，服务于Java的Collection框架





转换为线程安全的类

工具类Collections

- 工具类collections用于操作集合类，如List,Set,常用方法有；
  - **1） 排序（Sort）**
  - **2） 混排（Shuffling）**
  - **3） 反转（Reverse）**
  - **4） 替换所有的元素（Fill）**
  - **5） 拷贝（Copy）**
  - **6） 返回Collections中最小元素（min） 最大元素（max）**
  - **8） lastIndexOfSubList**
  - **9） IndexOfSubList**

工具类Collections

- **10） Rotate**
- 11)static int binarySearch(List list,Object key) 使用二分搜索查找key对象的索引值，因为使用的二分查找，所以前提是必须有序。
- 12)static Object max(Collection coll) 根据元素自然顺序，返回集合中的最大元素
- 13)static Object max(Collection coll,Compare comp) 根据Comparator指定的顺序，返回给定集合中的最大元素
- 14)static Object min(Collection coll) 根据元素自然顺序，返回集合中的最小元素
- 15)static Object min(Collection coll,Compare comp) 根据Comparator指定的顺序，返回给定集合中的最小元素
- 16）static void fill(List list,Object obj) 使用指定元素替换指定集合中的所有元素
- 17）static int frequency(Collection c,Object o) 返回指定元素在集合中出现在次数
- 18）static int indexOfSubList(List source, List target) 返回子List对象在父List对象中第一次出现的位置索引； 如果父List中没有出现这样的子List，则返回-1
- 19）static int lastIndexOfSubList(List source,List target) 返回子List对象在父List对象中最后一次出现的位置索引，如果父List中没有出现这样的子List，刚返回-1
- 20）static boolean replaceAll(List list,Object oldVal,Object newVal) 使用一个新值newVal替换List对象所有旧值oldVal
- 21）synchronizedXXX(new XXX) Collections类为集合类们提供的同步控制方法

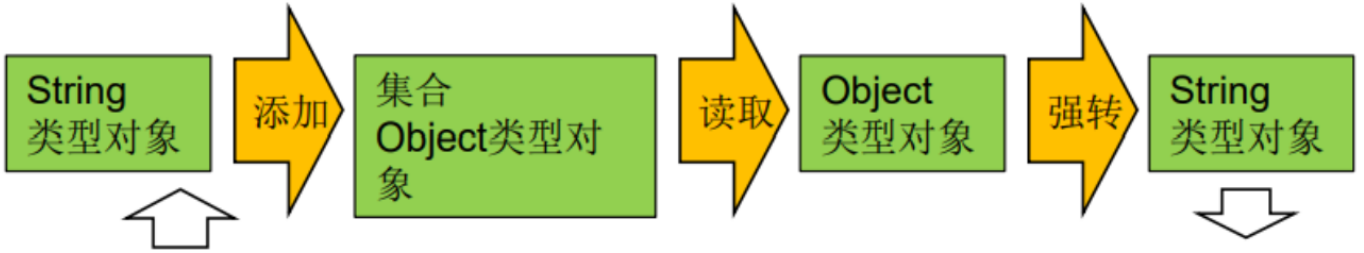
泛型集合的必要性

- 把任何类型对象通过add(Object obj) 放入集合中，都会向上转型为Object类型；
- 通过get(int index) 取出集合中元素时必须进行强制类型转换，繁琐而且容易出现异常；
- 使用Map的put(Object key, Object value)和get(Object key)存取对象时存在同样问题
- 使用Iterator的next()方法获取元素时存在同样问题

那么为什么要有泛型呢，直接Object不是也可以存储数据吗？

1. 解决元素存储的安全性问题，好比商品、药品标签，不会弄错。
2. 解决获取数据元素时，需要类型强制转换的问题，好比不用每回拿商品、药品都要辨别。

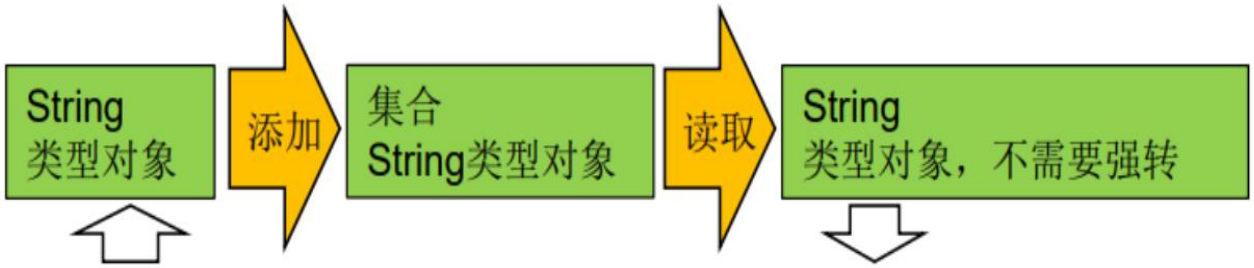
在集合中没有泛型时



任何类型都可以添加到集合中：**类型不安全**

读取出来的对象需要强转：**繁琐**
可能有ClassCastException

在集合中有泛型时



只有指定类型才可以添加到集合中：**类型安全**　读取出来的对象不需要强转：**便捷**

Java泛型可以保证如果程序在编译时没有发出警告，运行时就不会产生ClassCastException异常。同时，代码更加简洁、健壮。

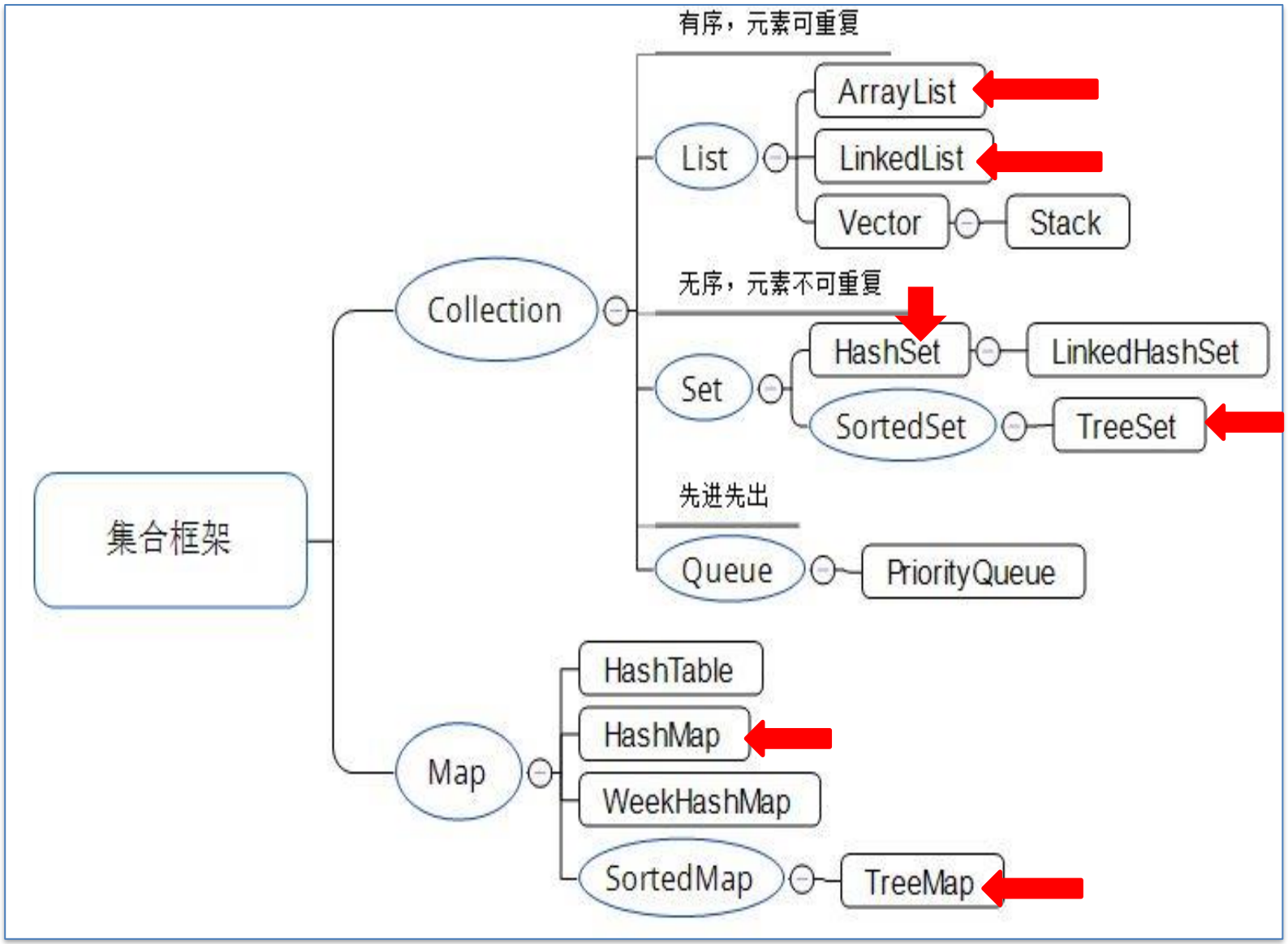
Java 泛型

- jdk 5.0新增的特性
- 在集合中使用泛型： 、
  - ① 集合接口或集合类在jdk5.0时都修改为带泛型的结构。
  - ② 在实例化集合类时，可以指明具体的泛型类型
  - ③指明完以后，在集合类或接口中凡是定义类或接口时，内部结构（比如：方法、构造器、属性等）使用到类的泛型的位置，都指定为实例化的泛型类型。 比如：add(E e) —>实例化以后：add(Integer e)
  - ④注意点：泛型的类型必须是类，不能是基本数据类型。需要用到基本数据类型的位置，拿包装类替换
  - ⑤如果实例化时，没有指明泛型的类型。默认类型为java.lang.Object类型。

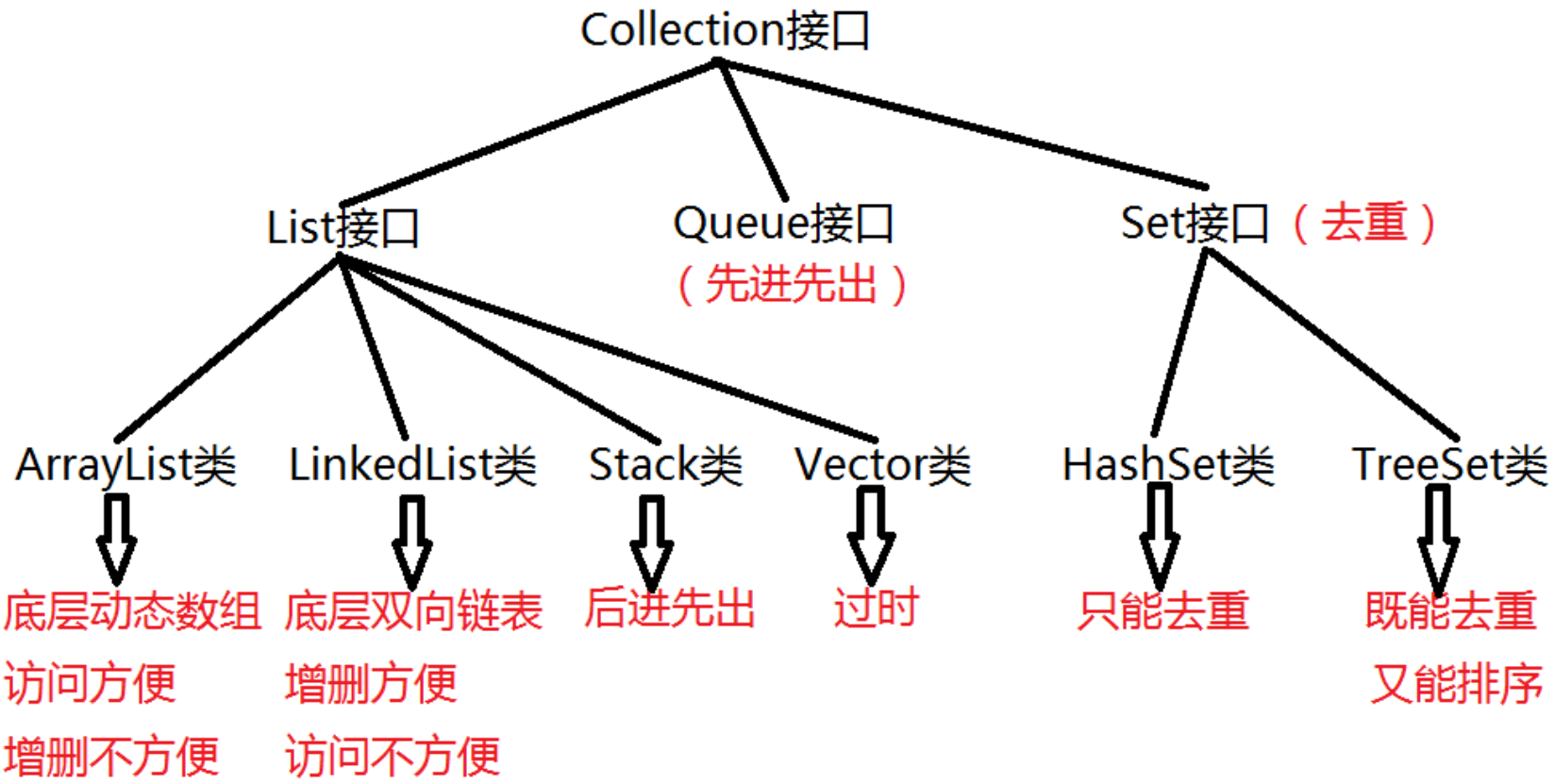
案例：

- GenericTest.java

小结（1）



小结（2）



本节课结束时，需要回答的问题

- Collection 、List 、Set 、Map接口的联系和区别有哪些？
- ArrayList和LinkedList有什么异同之处？
- 有哪些遍历集合的方法？
- 为什么要引入泛型集合？

谢谢！