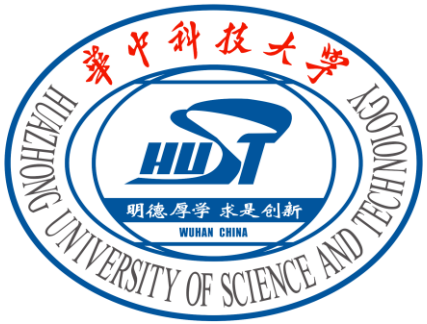
基于Java的面向对象程序设计

陈维亚

weiya_chen@hust.edu.cn

华中科技大学软件学院

第21讲：Java集合框架



目录

1. 简介

2. 集合接口

3. 集合实现类

4. 集合算法

1. 集合框架

❑ 集合

描述一个集合，数组难道不够吗？

```
int[] n = new int[10];
String[] s = {“how”, “are”, “you”};
```

- 长度固定，使用不灵活
- 经常出现访问越界的问题ArrayIndexOutOfBounds
- 没有现成的接口，得自己实现各种算法
- 缺少对元素的访问控制

我爱玩玩具公司WAW有如下需求：

a) 记录所有员工的姓名信息，每个月从员工列表中随机选出一个人，送给他（她）一个公司的玩具作为礼物。

b) WAW决定用员工的名字（去除姓氏）来命名公司的新产品，每个名字只使用一次，请创建一个无重复的名字列表。

c) 由于员工人数太多，WAW决定只使用出现频率最高的名字来命名产品，请创建一个集合，里面存有每个名字出现的次数。

d) WAW为员工准备了若干张福利电影票，请创建一个员工领票的等待队列。

1. 集合框架

❑ 集合

集合 Collection – 也称容器 Container，用于将一组元素合并为一个统一的单元。

集合用于**存储**、**提取**、**操作**和**传递**集成化的数据

天然适合用于代表成组出现的对象，比如一副扑克牌，文件收纳袋，通讯录等等。

在JDK 1.0中，代表集合的类只有Vector，Stack，Enumeration，HashTable，Properties和BitSet类。

虽然这些历史集合类都非常有用，但是它们缺少一个核心的，统一的主题。

JDK 1.2开始加入了多个**接口**，如Collection、Set、List和Map接口。

JDK 1.5开始引入**泛型**，增强了集合框架的通用性，形成了稳定的集合框架。

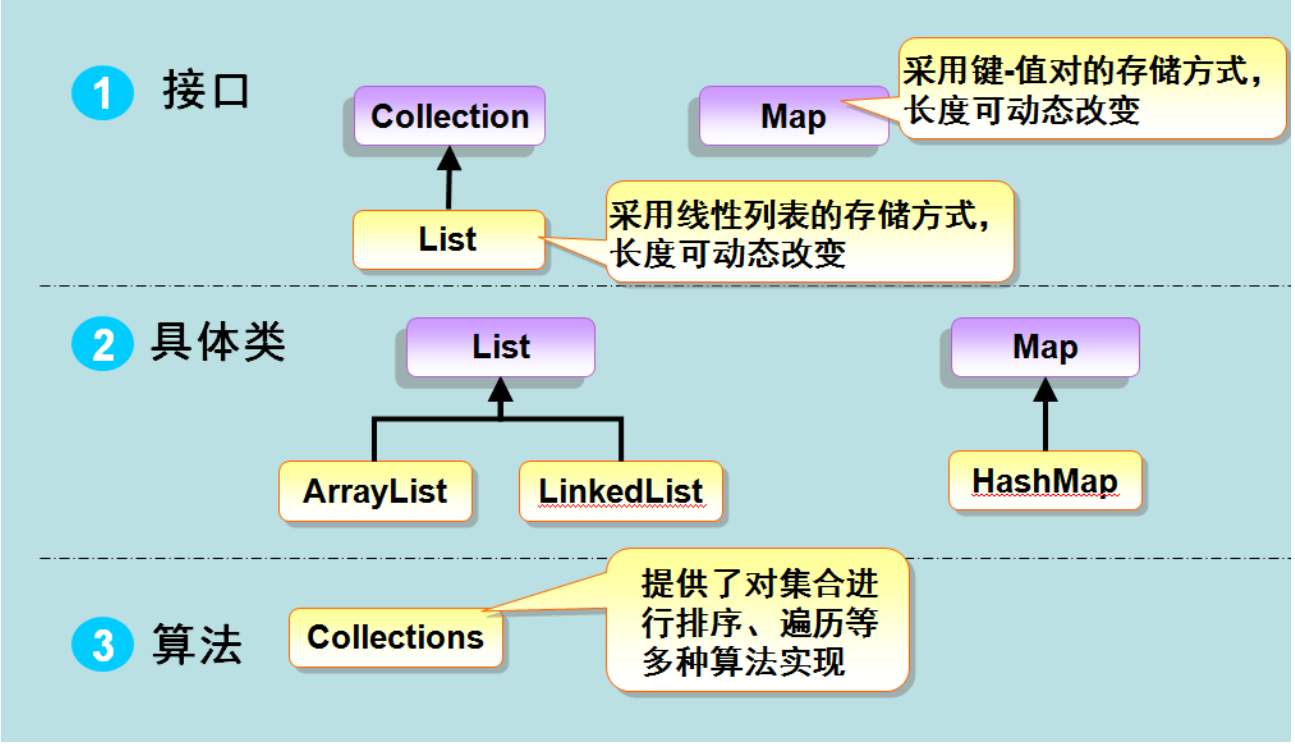
1. 集合框架

❑ Java Collection Framework 定义

接口：代表着集合的抽象数据类型。多级的接口构成了集合框架的标准调用接口。

实现（类）：是集合接口的具体实现，可重复使用的数据结构。

算法：用来执行一些有用的计算，这些算法的设计体现了多态。



1. 集合框架

❑ 好处

（1）使用核心集合类降低开发成本，一般无需自创集合类。

（2）使用经过严格测试的集合框架类可提高代码质量。

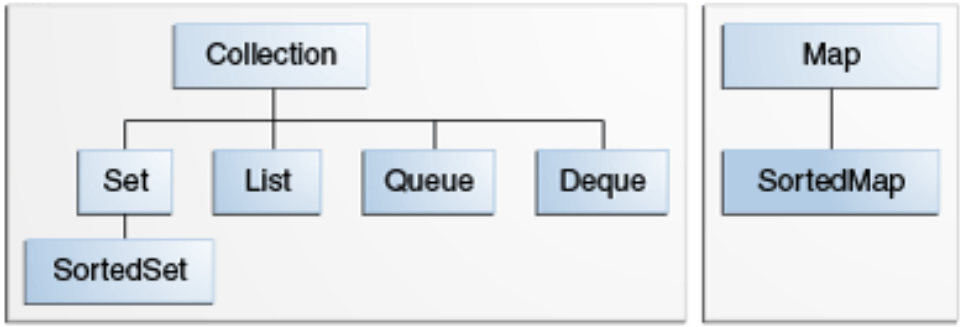
（3）增强独立开发API之间的互操作性。

（4）降低了学习和开发新的API的难度。

（5）有利于软件的重用。

2. 集合接口

❑ 概览



Java不提供直接实现Collection的类，
只提供实现其子接口(如List和set)的类。

```
public interface Collection<E>
```

普通操作

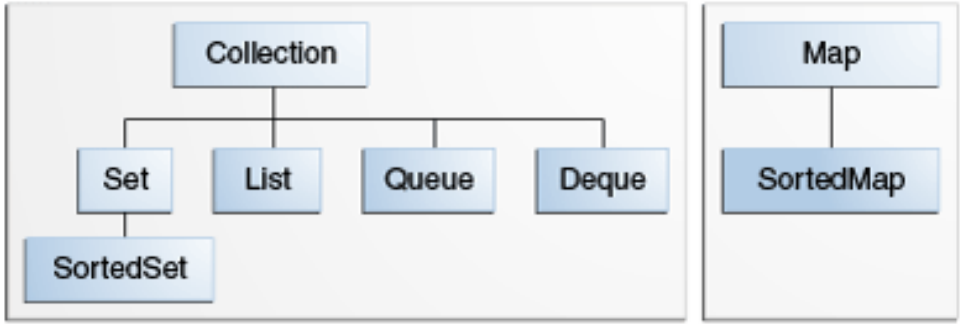
int size()
boolean isEmpty()
boolean contains(Object element)
boolean add(E element)
boolean remove(Object element)
Iterator<E> iterator()

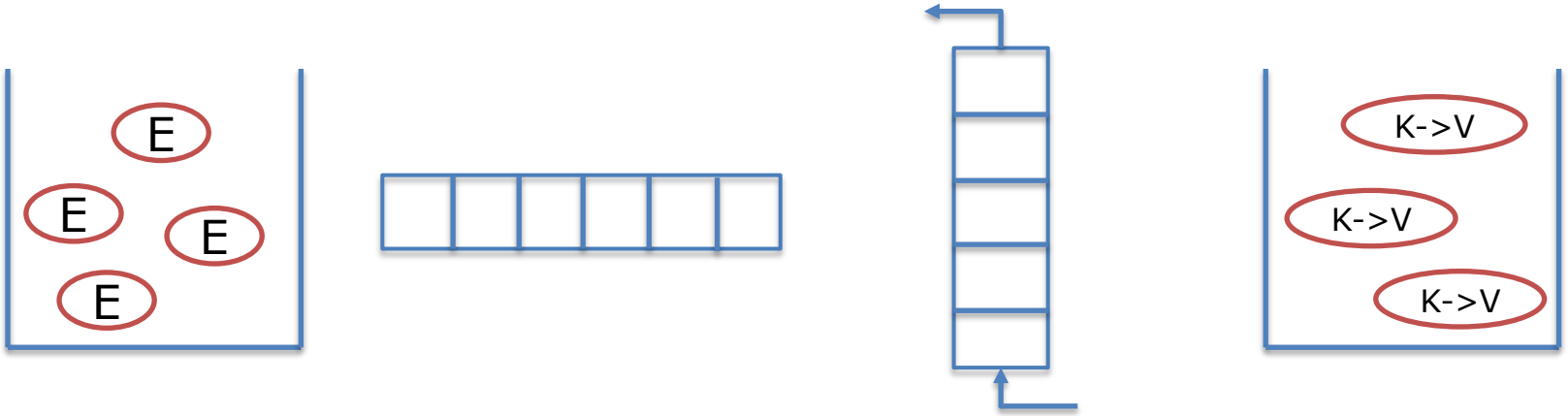
集合级操作

boolean containsAll(Collection<?> c)
boolean addAll(Collection<? extends E> c)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
void clear()

2. 集合接口

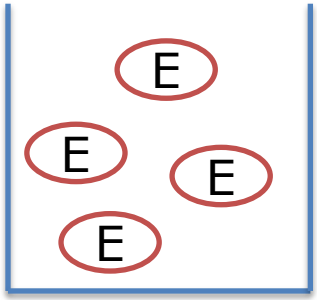
❑ 概览


Collection
Set
List
Queue
Deque
SortedSet
Map
SortedMap


E
E
E
E
K->V
K->V
K->V
Set
List
Queue
Map

2. 集合接口

❑ Set



Set

- HashSet
- TreeSet

Set和Collection功能基本一致

用于存放无重复的元素，类似数学意义上的集合

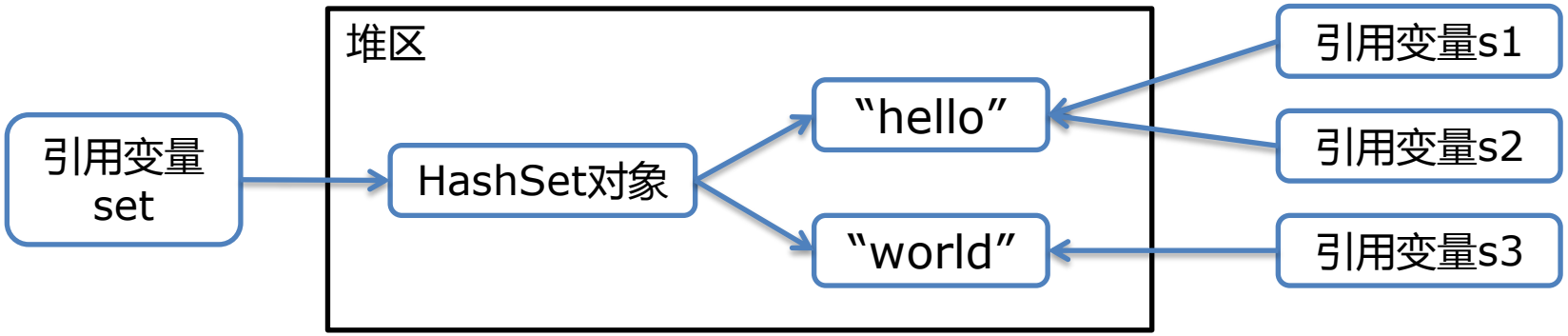
不保证存放顺序

```
public class FindDups {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        for (String a : args)
            s.add(a);
        System.out.println(s.size() + "
distinct words: " + s);
    }
}
```

Set实现

Set中保存的是对象的引用

```
Set<String> set = new HashSet<String>();
String s1 = new String("hello");
String s2 = s1;
String s3 = new String("world");

set.add(s1);
set.add(s2);
set.add(s3);
System.out.println(set.size());
```



2. 集合接口

List

List

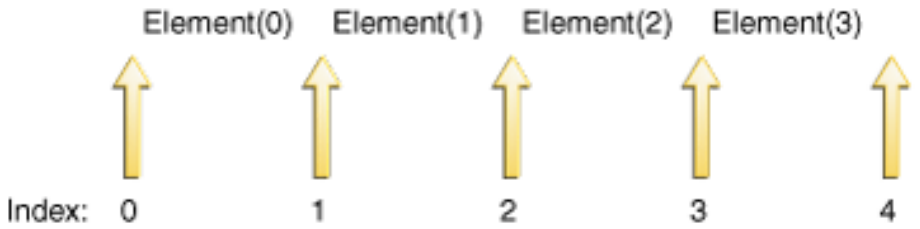
List是一个有序的Collection

List为一个长度可变的数组，即动态数组

可包含重复元素

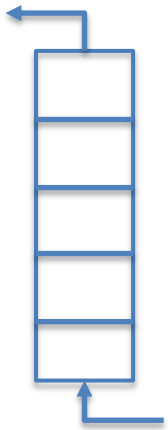
增加了按位置访问、查找和遍历功能

- ArrayList
- LinkedList



2. 集合接口

❑ Queue



Queue

Queue是一个特殊的Collection

元素的存取满足FIFO (first-in-first-out)

Queue可以限定长度

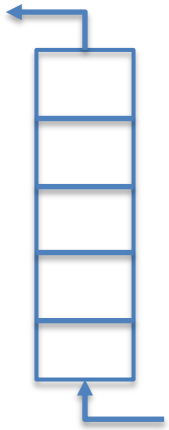
```
public interface Queue<E> extends Collection<E>
```

方法类型	抛异常	返回特殊值
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

- LinkedList

2. 集合接口

❑ Queue



Queue

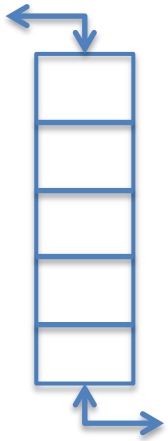
```
public class Countdown {
    public static void main(String[] args) throws
InterruptedException {
        int time = Integer.parseInt(args[0]);
        Queue<Integer> queue = new LinkedList<Integer>();

        for (int i = time; i >= 0; i--)
            queue.add(i);

        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
}
```

2. 集合接口

❑ Deque

双向队列Deque是Queue的子接口

可以从队列的两端加入和删除元素



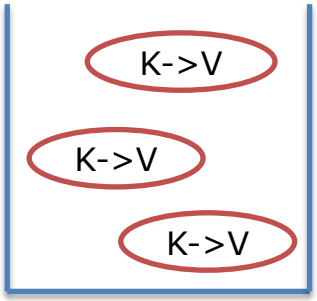
Deque

- LinkedList
- ArrayDeque

方法类型	第一个元素	最后一个元素
Insert	addFirst(e) offerFirst(e)	addLast(e) offerLast(e)
Remove	removeFirst() pollFirst()	removeLast() pollLast()
Examine	getFirst() peekFirst()	getLast() peekLast()

2. 集合接口

❑ Map



映射Map是把键对象和值对象进行映射的集合

按照无重复的键对象去检索值对象

Map.Entry对象代表Map中的一对键与值

Map

```
Map<String,String> map = new HashMap<String,String>();
map.put("1","Mon.");
map.put("1","Monday");
map.put("one","Monday");

Set<Map.Entry<String,String>> set = map.entrySet();
for(Map.Entry entry : set)
    System.out.println(entry.getKey()+":"+entry.getValue());
```

练习 1

1) What interface represents a collection that does not allow duplicate elements?
2) What interface forms the root of the collections hierarchy?
3) What interface represents an ordered collection that may contain duplicate elements?
4) What interface represents a collection that holds elements prior to processing?
5) What interface represents a type that maps keys to values?
6) What interface represents a double-ended queue?

练习 2

我爱玩玩具公司WAW有如下需求：

a) 记录所有员工的姓名信息，每个月从员工列表中随机选出一个人，送给他（她）一个公司的玩具作为礼物。

b) WAW决定用员工的名字（去除姓氏）来命名公司的新产品，每个名字只使用一次，请创建一个无重复的名字列表。

c) 由于员工人数太多，WAW决定只使用出现频率最高的名字来命名产品，请创建一个集合，里面存有每个名字出现的次数。

d) WAW为员工准备了若干张福利电影票，请创建一个员工领票的等待队列。

3. 集合实现

❑ 通用实现类

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

A. Iterator
迭代器

B. Hash集合
的元素存取

C. List
性能比较

D. 有序集合的
元素排序

3. 集合实现

❑ A. 集合的遍历 - Set, List, Queue

I. for循环

II. 增强for循环

III. Iterator接口

hasNext()
next()
remove(Object o)

从集合中删除一个对象的引用

```
List<String> list=new ArrayList<String>();
list.add("Hello");
list.add("World");
list.add("again");

for (String str : list) {
    System.out.println(str);
}

String[] strArray=new String[list.size()];
list.toArray(strArray);
for(int i=0;i<strArray.length;i++) {
    System.out.println(strArray[i]);
}

Iterator<String> it=list.iterator();
while(it.hasNext()) {
    System.out.println(it.next());
}
```

3. 集合实现

❑ A. 集合的遍历 - Map

```
Map<String, String> map = new HashMap<String, String>();
map.put("1", "value1");
map.put("2", "value2");
map.put("3", "value3");

//第一种：普遍使用，二次取值
for (String key : map.keySet()) {
    System.out.println("key= "+ key + " and value= " + map.get(key));
}

//第二种
Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
while (it.hasNext()) {
    Map.Entry<String, String> entry = it.next();
    System.out.println("key= "+entry.getKey()+" and value= "+entry.getValue());
}
```

3. 集合实现

❑ A. 集合的遍历 - Map

```
Map<String, String> map = new HashMap<String, String>();
map.put("1", "value1");
map.put("2", "value2");
map.put("3", "value3");


//第三种
for (Map.Entry<String, String> entry : map.entrySet()) {
    System.out.println("key= "+entry.getKey()+" and value= "+ entry.getValue());
}

//第四种
for (String v : map.values()) {
    System.out.println("value= " + v);
}
```

练习 3

```
ArrayList<String> al = new ArrayList<String>();
System.out.println("Initial size of al: " + al.size());
al.add("Cat");
al.add("Dog");
al.add("Sheep");
al.add(1, "Monkey");
System.out.println("Size of al after additions: " + al.size());
al.remove(0);
System.out.println(al.get(2));
```

```
Initial size of al: 0
Size of al after additions: 4
Sheep
```

3. 集合实现

❑ B. Hash集合的元素存取

Hash集合有很好的存取和查找功能。

向集合中加入对象时，会调用hashCode()方法获得哈希码，进而计算出对象在集合中的存放位置。

```
obj1.equals(obj2);
obj1.hashCode() == obj2.hashCode();
```

Hash集合要求两个对象用equals()方法比较的结果为true时，它们的哈希码也相等。

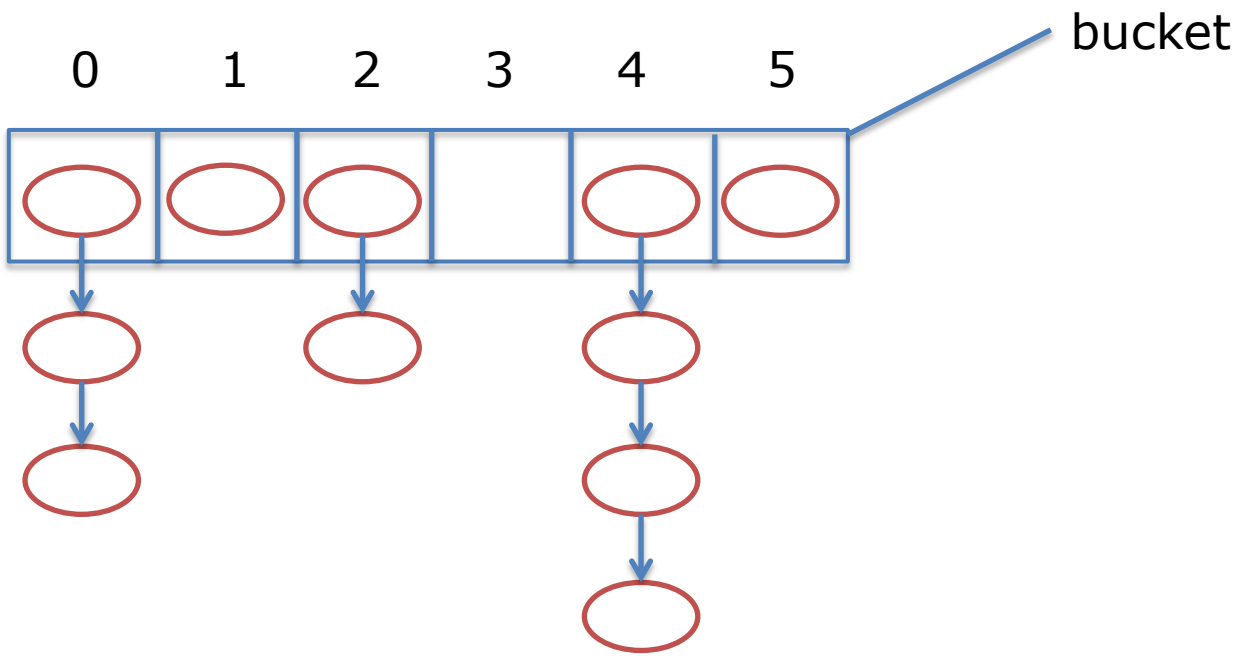
3. 集合实现

```
public class Customer {
    private String name;
    private int age;

    public boolean equals(Object o){
        if(!(o instanceof Customer)) return false;
        Customer other = (Customer)o;
        if(this.name.equals(other.getName())&&
this.age==other.getAge())
            return true;
        else
            return false;
    }
}
```

```
Set<Customer> set = new HashSet<Customer>();
Customer c1 = new Customer("Tom",15);
Customer c2 = new Customer("Tom",15);
set.add(c1);
set.add(c2);
System.out.println(set.size());
```

3. 集合实现

❑ B. Hash集合的负载因子



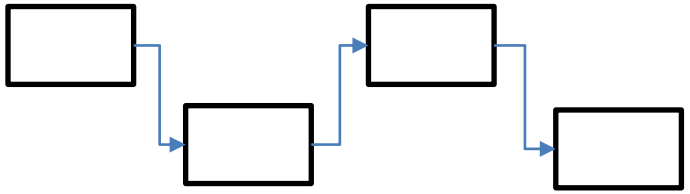
- 容量 capacity：桶的数量
- 初始容量 initial capacity
- 大小 size：元素的数目
- 负载因子 load factor：size/capacity (default=0.75)

3. 集合实现

- C. List性能比较

ArrayList　　Vector

LinkedList



3. 集合实现

❑ C. List性能比较

类型	静态数组	ArrayList	LinkedList	Vector
随机访问（get）	16	23	63	31
遍历（iterate）	31	47	33	48
插入（insert）	/	1610	31	1625
删除（remove）	/	6625	16	6750

各项操作用时比较（单位：ms）

ArrayList是顺序访问的首选

LinkedList可作为栈Stack、队列Queue和双向队列Deque的实现类

3. 集合实现

❑ D. 有序集合

TreeSet 和 TreeMap 分别实现了 SortedSet 和 SortedMap 接口，能够对集合中的元素排序

```
Set<Integer> set = new TreeSet<Integer>();
set.add(8);
set.add(7);
set.add(6);
set.add(9);

for(int i: set)
    System.out.print(i+" ");
```

```
6 7 8 9
```

- 自然排序（实现Comparable的基础类）：如Integer，Double和String等
- 自定义排序（实现Comparable或Comparator的自定义类）

3. 集合实现

❑ Comparable和Comparator接口

Comparable

由需要被排序的类实现
需要实现compareTo()方法

Comparator<T>

由排序器实现
需要实现compare(T x, T y)方法

3. 集合实现

```
public class Customer implements Comparable{
    private String name;
    private int age;

    public int compareTo(Object o){
        Customer other = (Customer)o;
        if(this.name.compareTo(other.getName())>0) return 1;
        if(this.name.compareTo(other.getName())<0) return -1;

        return 0;
    }

    public boolean equals(Object o){…}

    public int hashCode(){…}
}
```

```
Set<Customer> set = new TreeSet<Customer>();
set.add(new Customer("Tom",15));
set.add(new Customer("Mike",20));
for(Customer c: set)
    System.out.println(c.getName());
```

3. 集合实现

```
public class Customer{
    private String name;
    private int age;
}
```

```
public class CustomerComparator implements Comparator<Customer>{
    public int compare(Customer c1, Customer c2){
        if(c1.getName().compareTo(c2.getName())>0) return 1;
        if(c1.getName().compareTo(c2.getName())<0) return -1;

        return 0;
    }
}
```

```
Set<Customer> set = new TreeSet<Customer>(new CustomerComparator());
set.add(new Customer("Tom",15));
set.add(new Customer("Mike",20));

for(Customer c: set)
    System.out.println(c.getName());
```

4. 集合算法

❑ 总览

作为集合框架的一部分，集合实用类 Collection 提供了许多静态方法

大部分适用于List的实现类，少部分适用于所有Collection接口的实现类

- 排序　　Sorting
- 打乱　　Shuffling
- 操作数据　　Routine Data Manipulation
- 查找　　Searching
- 分析　　Composition
- 找极值　　Finding Extreme Values

4. 集合算法

- 排序

```
List<Integer> list = new ArrayList<Integer>();
list.add(4);
list.add(2);
list.add(3);

Collections.sort(list);
System.out.println(list);
```

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

Collections.sort(list, new Comparator<List<String>>() {
    public int compare(List<Integer> o1, List<Integer> o2) {
        return o2.size() - o1.size();
    }});
```

4. 集合算法

- 打乱

```
Collections.shuffle(list);
Collections.shuffle(list, new Random());
```

4. 集合算法

- 操作数据

reverse — reverses the order of the elements in a List.

fill — overwrites every element in a List with the specified value. This operation is useful for reinitializing a List.

copy — takes two arguments, a destination List and a source List, and copies the elements of the source into the destination, overwriting its contents. The destination List must be at least as long as the source. If it is longer, the remaining elements in the destination List are unaffected.

swap — swaps the elements at the specified positions in a List.

addAll — adds all the specified elements to a Collection. The elements to be added may be specified individually or as an array.

4. 集合算法

- 查找

```
int pos = Collections.binarySearch(list, key);
```

- 分析

frequency — counts the number of times the specified element occurs in the specified collection

disjoint — determines whether two Collections are disjoint; that is, whether they contain no elements in common

- 找极值

min()

max()

4. 集合算法

❑ 集合与数组互换

数组转换为集合

```
Integer[] array = {1,2,3,4,5};
List<Integer> list = Arrays.asList(array);
Set<Integer> hashset = new HashSet<Integer>(list);
```

集合转换为数组

```
List<Integer> list = new ArrayList<Integer>();
list.add(1);
list.add(2);

Object[] array1 = list.toArray();
Integer[] array2 = list.toArray(new Integer[0]);
```

4. 集合算法

❑ 集合的批量操作

Collection接口中定义了：

boolean retainAll(Collection<?> c)

boolean removeAll(Collection<?> c)

boolean addAll(Collection<? extends E> c)

boolean containsAll(Collection<?> c)

练习 4

```
final static Integer[] DATA1 = {11,22,33,44,55,66};
final static Integer[] DATA2 = {11,22,77,88};

static Set<Integer> getOriginalSet(Integer[] data){
    Set<Integer> set = new HashSet<Integer>(Arrays.asList(data));
    return set;
}

static void print(Collection<Integer> col){
    for(Integer i: col)
        System.out.print(i+" ");
}

public static void main(String[] args){
    Set<Integer> set1 = getOriginalSet(DATA1);
    Set<Integer> set2 = getOriginalSet(DATA2);
    set1.retainAll(set2);
    print(set1);  // ?

    set1 = getOriginalSet(DATA1);
    set2 = getOriginalSet(DATA2);
    set1.removeAll(set2);
    print(set1);  // ?
}
```

补充

❑ 枚举类型

JDK 5 提供了抽象的 java.lang.Enum 枚举类

```
public abstract class Enum<E extends Enum<E>>
```

用户自定义的枚举类只需继承 Enum 类

```
public class Gender extends Enum {
    public static final Gender FEMALE;
    public static final Gender MALE;
    …
}
```

```
public enum Gender{FEMALE, MALE}
```

补充

❑ 枚举类型

```
public enum Gender{FEMALE, MALE}
```

```
// 遍历Gender类的所有常量
for(Gender g: Gender.values())
    System.out.println(g.ordinal()+" "+g.name());

// 根据 g 的值选择
Gender g = Gender.FEMALE;
switch(g){
    case FEMALE:
        System.out.println("girl");
        break;
    case MALE:
        System.out.println("boy");
        break;
    default:
        System.out.println("unknown");
}
```

补充

❑ 枚举类型

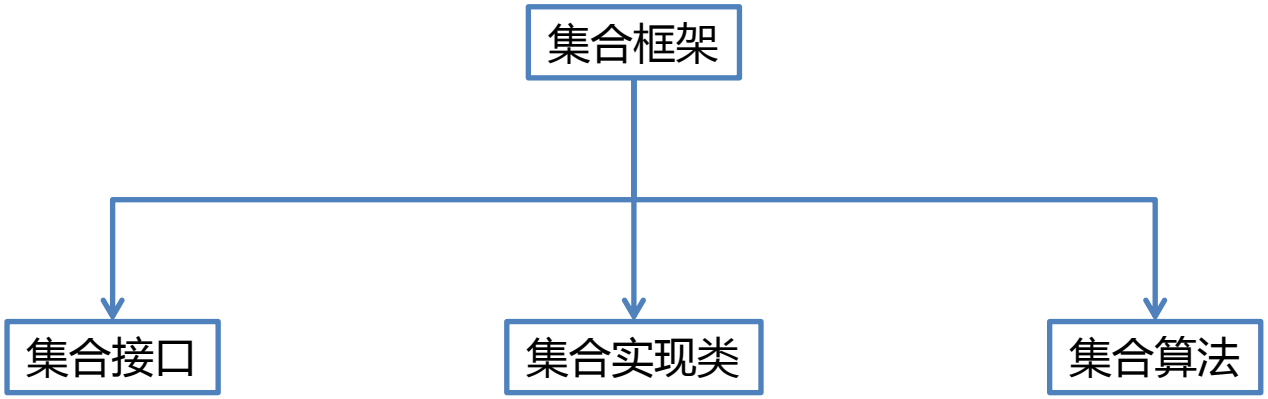
Java API 中有两个 Enum 类的适配器：

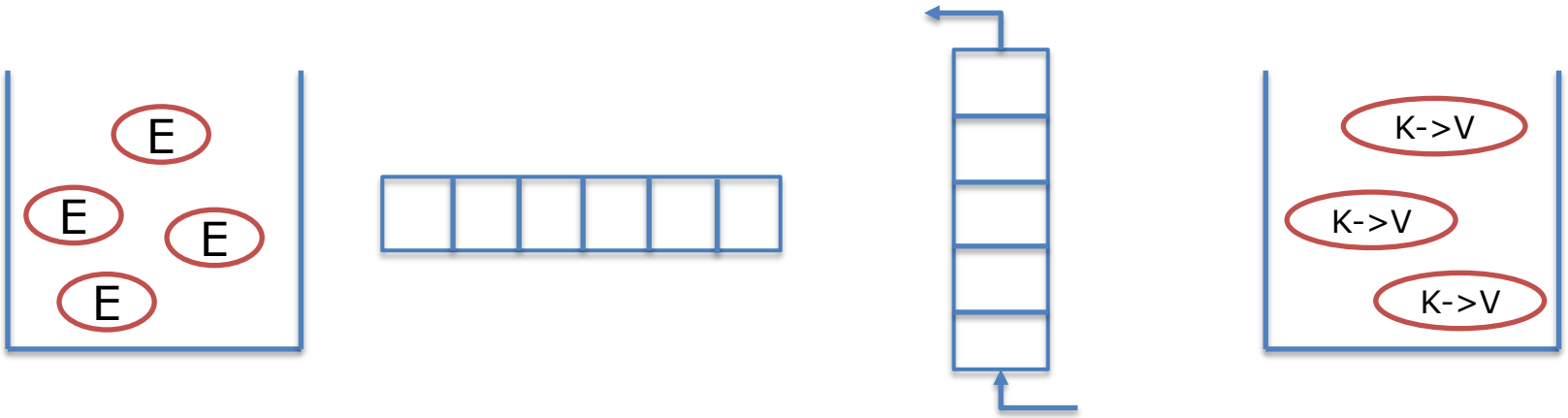
java.util.EnumSet

java.util.EnumMap

```
enum Color{RED,GREEN,BLUE};

// Enum 转为 EnumSet
EnumSet<Color> colorSet = EnumSet.allOf(Color.class);

// Enum 转为 EnumMap
EnumMap<Color,String> colorMap = new EnumMap<Color,String>(Color.class);
colorMap.put(Color.RED,"红色");
colorMap.put(Color.GREEN,"绿色");
colorMap.put(Color.BLUE,"蓝色");
```

总结





总结

❑ 集合框架使用建议

如何选择集合

（1）根据需要选择正确的集合类型。比如，如果指定了大小，我们会选用Array而非ArrayList。如果我们不想重复，我们应该使用Set。

（2）HashSet 和 HashMap 具有较好的性能，是 Set 和 Map 的首选实现类，只有需要排序时才考虑 TreeSet 和 TreeMap。ArrayList 和 LinkedList 各有优缺点，应视情况选择。

如何使用集合

（3）如果我们能够估计到存储元素的数量，需指定初始容量。

（4）多用接口声明集合变量，它允许我们轻易地替换具体实现类。

（5）使用泛型，避免在运行时出现ClassCastException。

（6）使用JDK提供的不可变类作为Map的key，可以避免自己实现hashCode()和equals()。

（7）尽可能使用Collections实用类提供的方法，而非编写自己的实现。

下节预告

异常处理