集合（Collections）

@M了个J
李明杰

https://github.com/CoderMJLee
http://cnblogs.com/mjios

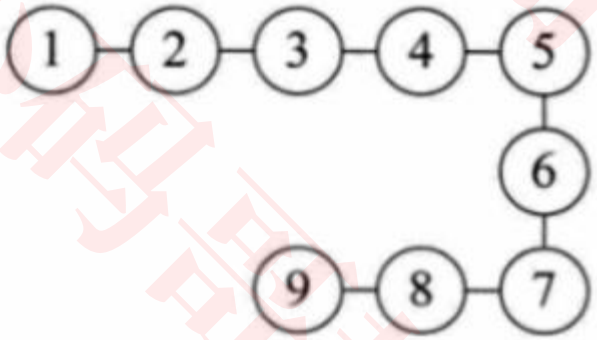
码拉松

实力IT教育 www.520it.com

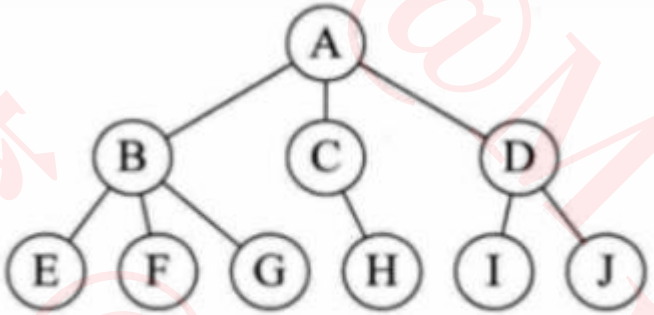
集合（Collections）

- java.util 包中有个集合框架（Collections Framework），提供了一大堆常用的数据结构
  - ArrayList、LinkedList、Queue、Stack、HashSet、HashMap 等

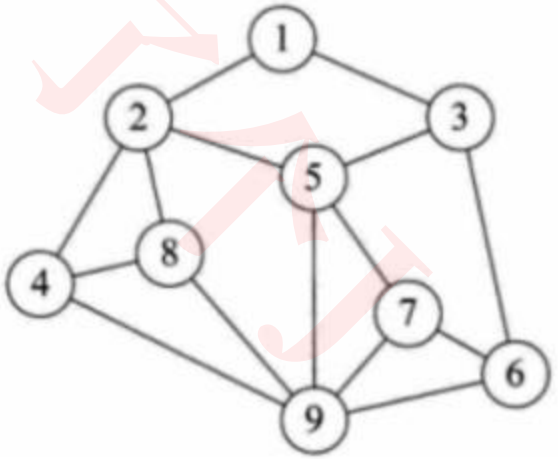
- 数据结构是计算机存储、组织数据的方式



线性结构



树形结构



图形结构

- 在实际应用中，根据使用场景来选择最合适的数据结构

小码哥教育 SEEMYGO

数据结构与算法课程推荐

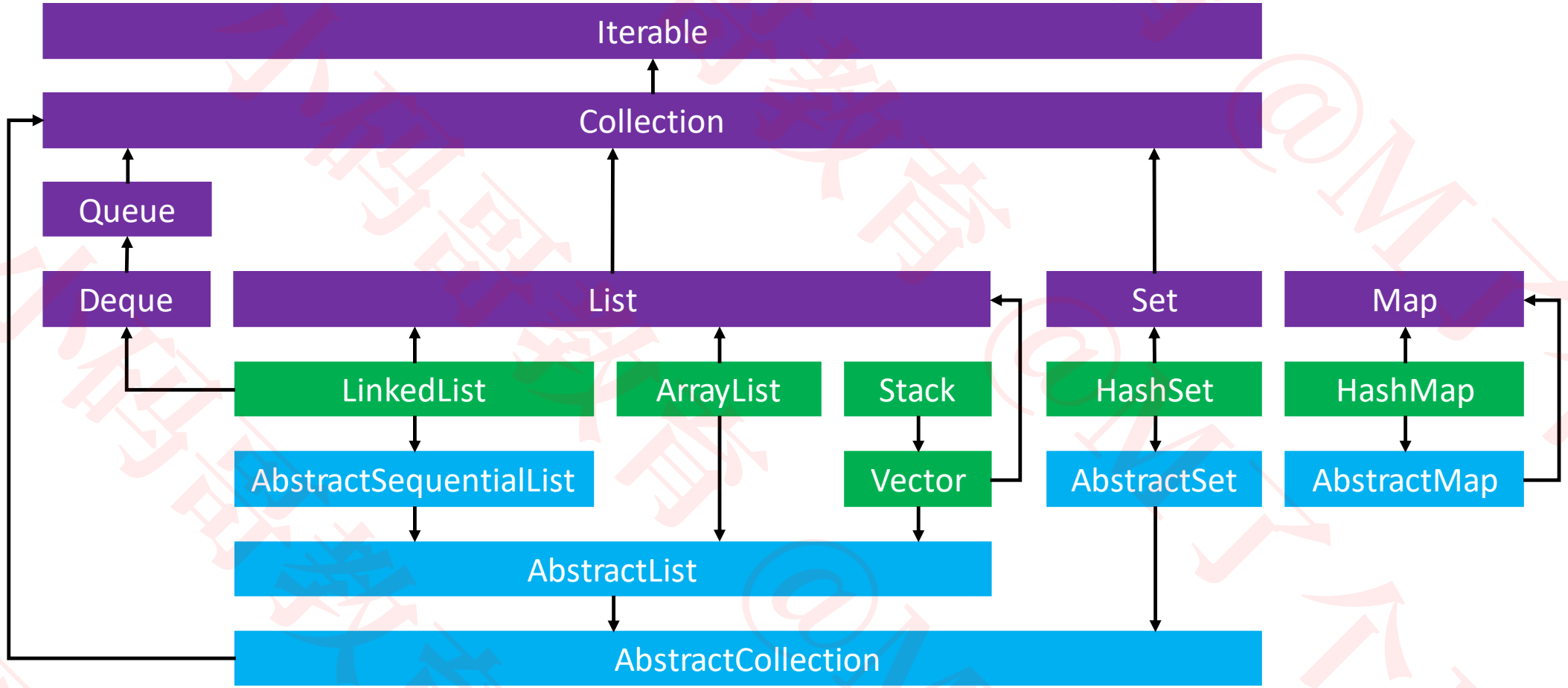
- 《恋上数据结构与算法》合集
  - https://ke.qq.com/course/package/22853

- 《每周一到算法题》
  - https://ke.qq.com/course/436549

- 个人 B 站的免费算法教程
  - https://www.bilibili.com/video/av78037011
  - https://www.bilibili.com/video/av77758248

小码哥教育 SEEMYGO

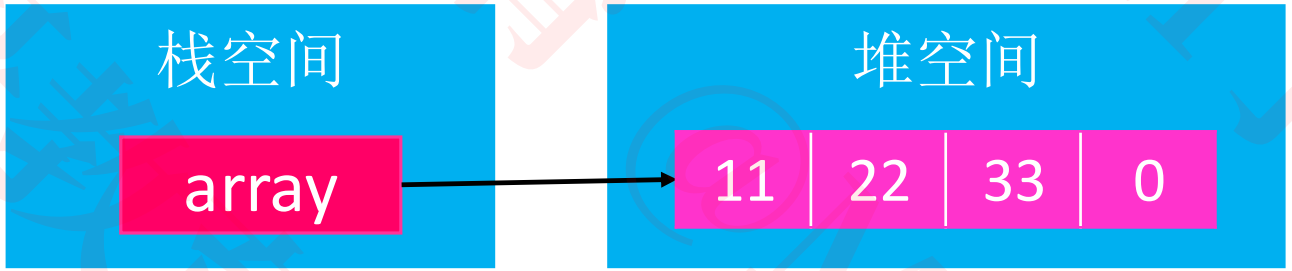
集合框架预览



- 思考题：为何 Map 如此独立？跟 Collection、Iterable 毫无继承关系？

数组的局限性

```
int[] array = new int[4];
array[0] = 11;
array[1] = 22;
array[2] = 33;
```



- 数组的局限性
  - 无法动态扩容
  - 操作元素的过程不够面向对象
  - ……

- ArrayList 是 Java 中的动态数组
  - 一个可以动态扩容的数组
  - 封装了各种实用的数组操作

小码哥教育 SEEMYGO

ArrayList 的常用方法

```
public int size()
public boolean isEmpty()
public boolean contains(Object o)
public int indexOf(Object o)
public int lastIndexOf(Object o)
public E get(int index)
public E set(int index, E element)
public boolean add(E e)
public void add(int index, E element)
public E remove(int index)
public boolean remove(Object o)
public void clear()
```

```
public Object[] toArray()
public <T> T[] toArray(T[] a)
public void trimToSize()
public void ensureCapacity(int minCapacity)
```

```
public boolean addAll(Collection<? extends E> c)
public boolean addAll(int index, Collection<? extends E> c)
public boolean removeAll(Collection<?> c)
public boolean retainAll(Collection<?> c)
public void forEach(Consumer<? super E> action)
public void sort(Comparator<? super E> c)
```

小码哥教育 SEEMYGO

ArrayList 的基本使用

```
ArrayList list = new ArrayList();
list.add(11);
list.add(false);
list.add(null); // 可以添加null值
list.add(3.14);
list.add(0, "Jack");
list.add('8');
// 3
System.out.println(list.indexOf(null));
// 6
System.out.println(list.size());
// [Jack, 11, false, null, 3.14, 8]
System.out.println(list);
```

小码哥教育 SEEMYGO

ArrayList – retainAll

```java
List<Integer> list1 = new ArrayList<>();
list1.add(11);
list1.add(22);
list1.add(33);
list1.add(44);

List<Integer> list2 = new ArrayList<>();
list2.add(22);
list2.add(44);

// 从 list1 中删掉除 list2 中元素以外的所有元素
list1.retainAll(list2);

// [22, 44]
System.out.println(list1);
```

小码哥教育 SEEMYGO

ArrayList – toArray

```
List<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);
```

```
Object[] array1 = list.toArray();
// [Ljava.lang.Object;@15db9742
System.out.println(array1);
// [11, 22, 33]
System.out.println(Arrays.toString(array1));
```

```
Integer[] array2 = list.toArray(new Integer[0]);
// [Ljava.lang.Integer;@6d06d69c
System.out.println(array2);
// [11, 22, 33]
System.out.println(Arrays.toString(array2));
```

小码哥教育 SEEMYGO

ArrayList 的遍历

```
int size = list.size();
for (int i = 0; i < size; i++) {
    System.out.println(list.get(i));
}
```

```
Iterator it = list.iterator();
while (it.hasNext()) {
    Object obj = it.next();
    System.out.println(obj);
}
```

```
for (Object obj : list) {
    System.out.println(obj);
}
```

```
list.forEach((obj) -> {
    System.out.println(obj);
});
```

for-each 格式

```
// for-each 格式
for (元素类型 变量名 : 数组\Iterable) {

}
```

- 实现了 Iterable 接口的对象，都可以使用 for-each 格式遍历元素
  - 比如 List、Set 等

- Iterable 在使用 foreach 格式遍历元素时，本质是使用了 Iterator 对象

小码哥教育 SEEMYGO

自定义 Iterable、Iterator

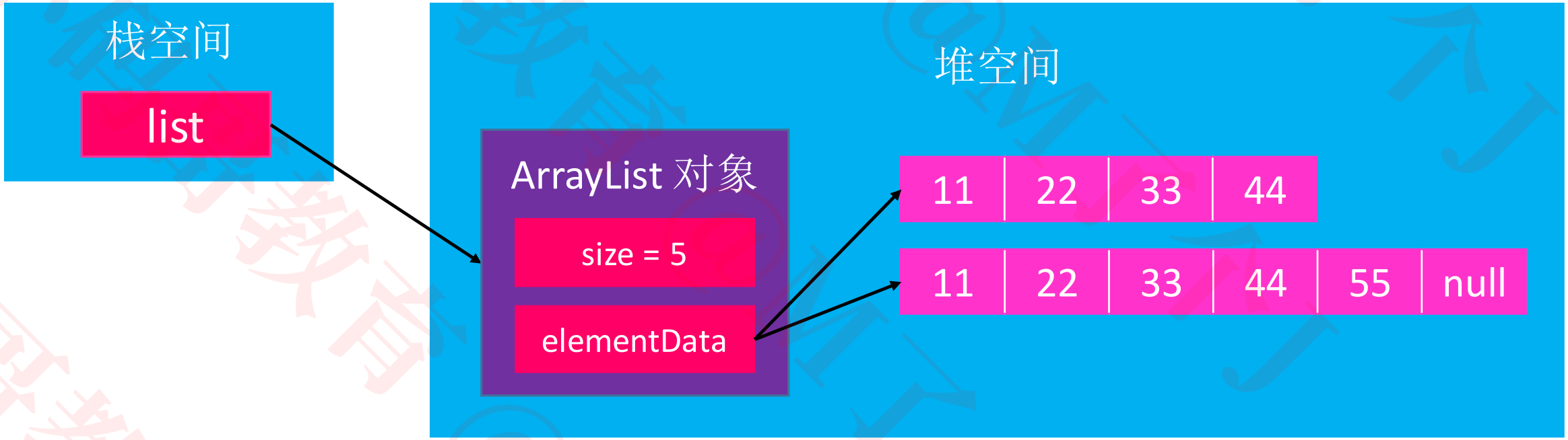
```
public class ClassRoom implements Iterable<String> {
    private String[] students;
    public ClassRoom(String... students) {
        this.students = students;
    }
    @Override
    public Iterator<String> iterator() {
        return new ClassRoomIterator();
    }

    private class ClassRoomIterator implements Iterator<String> {
        private int index;
        @Override
        public boolean hasNext() {
            return index < students.length;
        }
        @Override
        public String next() {
            return students[index++];
        }
    }
}
```

```
ClassRoom room = new ClassRoom("Jack", "Rose");
for (String string : room) {
    System.out.println(string);
} // Jack Rose
```

ArrayList 的扩容原理

```
ArrayList<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);
list.add(44);
list.add(55);
```

- ArrayList 的最小容量是 10
- 每次扩容时，新容量是旧容量的 1.5 倍



遍历的注意点

```
List<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);
list.add(44);
```

```
int size = list.size();
for (int i = 0; i < size; i++) {
    // java.lang.IndexOutOfBoundsException
    list.remove(i);
}
```

```
for (int i = 0; i < list.size(); i++) {
    list.remove(i);
}
// [22, 44]
System.out.println(list);
```

- 需求：通过遍历的方式，挨个删除所有的元素

小码哥教育 SEEMYGO

遍历的注意点

```
// java.util.ConcurrentModificationException
for (Integer e : list) {
    list.remove(e);
}
```

```
// java.util.ConcurrentModificationException
list.forEach((e) -> {
    list.remove(e);
});
```

```
Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    it.next();
    it.remove();
}
// 0
System.out.println(list.size());
```

- 如果希望在遍历元素的同时删除元素
  - 请使用 Iterator 进行遍历
  - 然后使用 Iterator 的 remove 方法删除元素

- 使用迭代器、forEach 遍历集合元素时，若使用了集合自带的方法修改集合的长度（比如 add、remove 等方法）
  - 那么会抛出 java.util.ConcurrentModificationException 异常

小码哥教育 SEEMYGO

ListIterator

- ListIterator 继承自 Iterator，在 Iterator 的基础上增加了一些功能

```
boolean hasNext();
E next();
boolean hasPrevious();
E previous();
int nextIndex();
int previousIndex();
void remove();
void set(E e);
void add(E e);
```

```
List<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);

ListIterator<Integer> it = list.listIterator();
while (it.hasNext()) {
    System.out.println(it.next());
} // 11 22 33

while (it.hasPrevious()) {
    System.out.println(it.previous());
} // 33 22 11
```

ListIterator – 示例

```
List<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);

ListIterator<Integer> it = list.listIterator();
while (it.hasNext()) {
    it.set(it.next() + 55);
}

System.out.println(list); // [66, 77, 88]
```

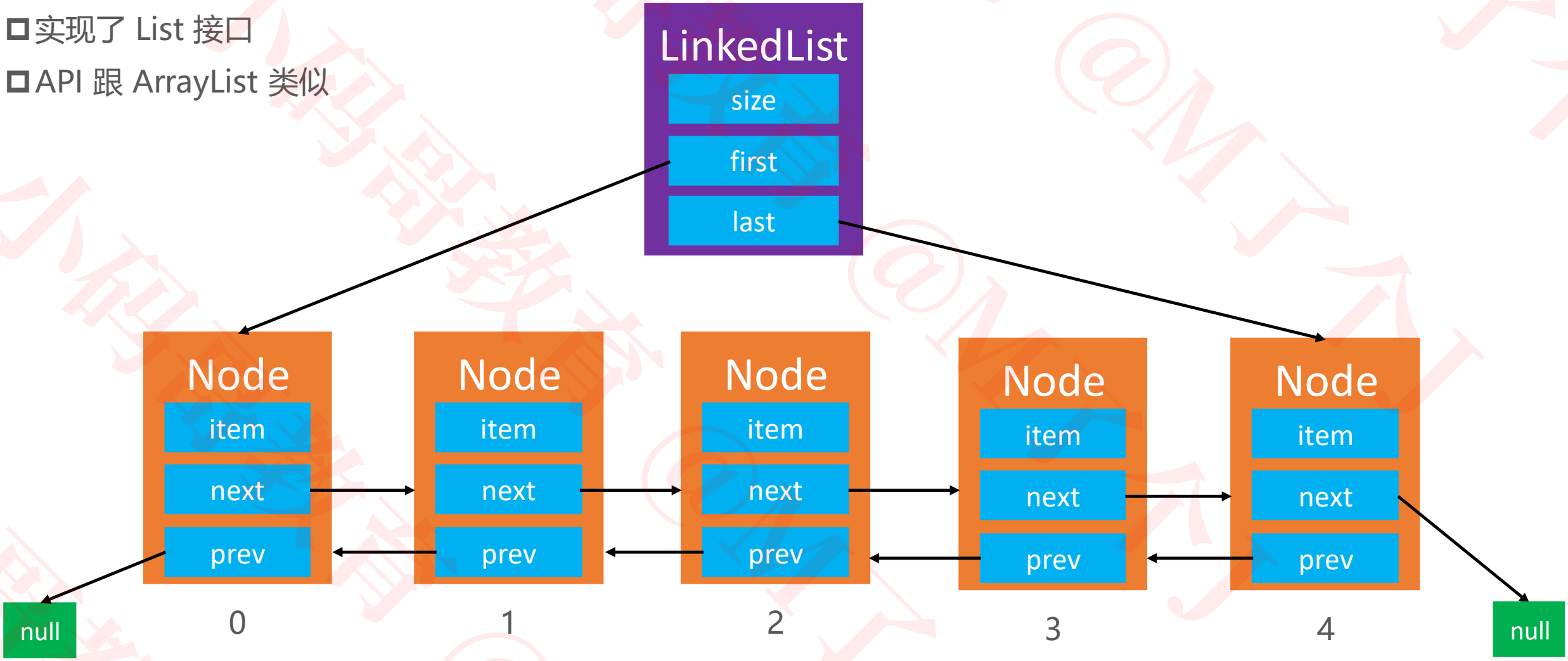
小码哥教育 SEEMYGO

ListIterator – 示例

```
List<Integer> list = new ArrayList<>();
list.add(11);
list.add(22);
list.add(33);

ListIterator<Integer> it = list.listIterator();
while (it.hasNext()) {
    it.add(66);
    System.out.println(it.next());
    it.add(77);
} // 11 22 33

// [66, 11, 77, 66, 22, 77, 66, 33, 77]
System.out.println(list);
```

LinkedList

- LinkedList 是一个双向链表
  - 实现了 List 接口
  - API 跟 ArrayList 类似



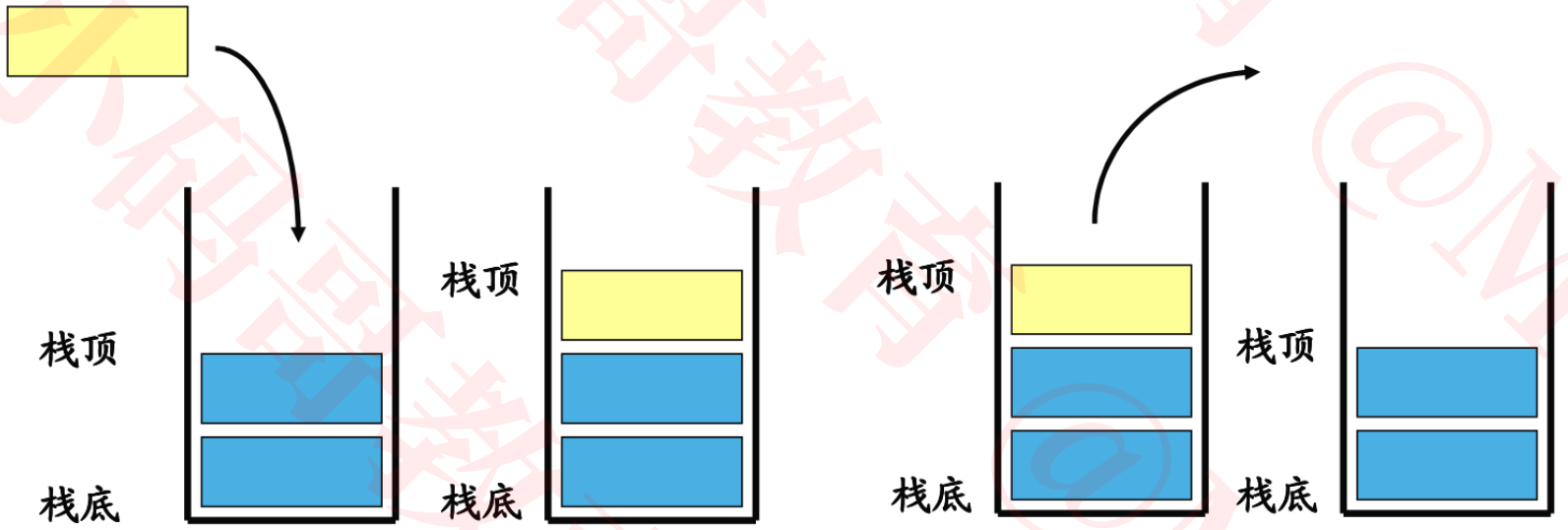
LinkedList vs ArrayList

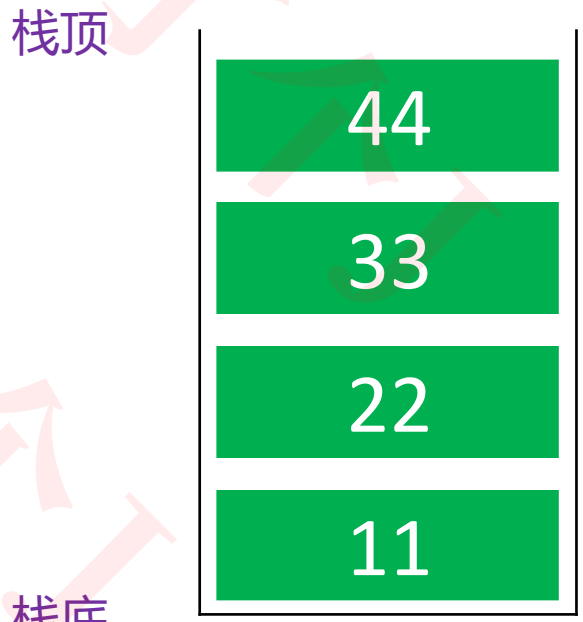
- ArrayList：开辟、销毁内存空间的次数相对较少，但可能造成内存空间浪费（可以通过缩容解决）
- LinkedList：开辟、销毁内存空间的次数相对较多，但不会造成内存空间的浪费

- 如果频繁在尾部进行添加、删除操作， ArrayList、LinkedList 均可选择
- 如果频繁在头部进行添加、删除操作，建议选择使用 LinkedList
- 如果有频繁的（在任意位置）添加、删除操作，建议选择使用 LinkedList
- 如果有频繁的查询操作（随机访问操作），建议选择使用 ArrayList

Stack

- Stack，译为“栈”，只能在一端进行操作
- 往栈中添加元素的操作，一般叫做 push，入栈
- 从栈中移除元素的操作，一般叫做 pop，出栈（只能移除栈顶元素，也叫做：弹出栈顶元素）
- 后进先出的原则，Last In First Out，LIFO





- 注意：这里说的“栈”与内存中的“栈空间”是两个不同的概念

小码哥教育 SEEMYGO

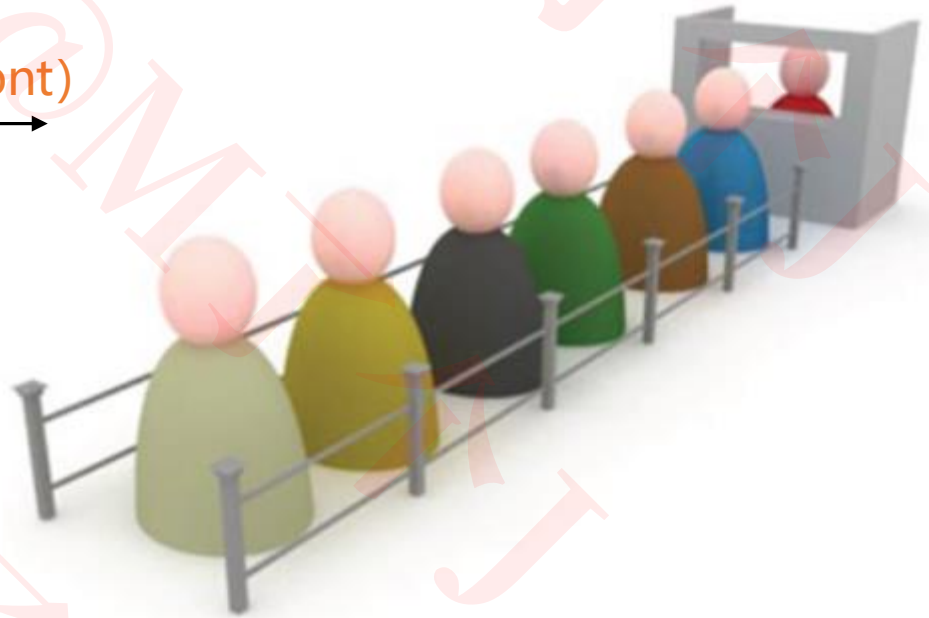
Stack – 常用方法

```
public E push(E item)
public E pop()
public E peek() // 返回栈顶元素
public int size()
public boolean empty() // 查看栈是否为空
// 返回元素的索引（从 1 开始递增，栈顶为 1）
public int search(Object o)
```

```
Stack<Integer> stack = new Stack<>();
stack.push(11);
stack.push(22);
stack.push(33);
// 33
System.out.println(stack.peek());
// 3
System.out.println(stack.search(11));
while (!stack.empty()) {
    System.out.println(stack.pop());
} // 33 22 11
```

Queue

- Queue，译为“队列” ，只能在头尾两端进行操作
  - 队尾（rear）：只能从队尾添加元素，一般叫做 入队
  - 队头（front）：只能从队头移除元素，一般叫做 出队
  - 先进先出的原则，First In First Out，FIFO



Queue – 常用方法

```
boolean add(E e); // 入队
boolean offer(E e); // 入队
E remove(); // 出队
E poll(); // 出队
E element(); // 返回队头
E peek(); // 返回队头
boolean isEmpty();
int size();
```

- java.util.Queue 是一个接口，它的常用实现是 LinkedList

```
Queue<Integer> queue = new LinkedList<>();
queue.add(11);
queue.add(22);
queue.add(33);
// 11
System.out.println(queue.element());
while (!queue.isEmpty()) {
    System.out.println(queue.remove());
} // 11 22 33
```

小码哥教育 SEEMYGO

HashSet

```
Set<String> set = new HashSet<>();
set.add("Jack");
set.add("Rose");
set.add("Jim");
set.add("Jack");
set.add("Kate");
// 4
System.out.println(set.size());
// [Kate, Rose, Jack, Jim]
System.out.println(set);
set.remove("Rose");
// [Kate, Jack, Jim]
System.out.println(set);
```

```
Set<String> set = new HashSet<>();
set.add("Jack");
set.add("Rose");
set.add("Jim");

for (String str : set) {
    System.out.println(str);
} // Rose Jack Jim

Iterator<String> it = set.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
} // Rose Jack Jim

set.forEach((str) -> {
    System.out.println(str);
}); // Rose Jack Jim
```

小码哥教育 SEEMYGO

LinkedHashSet

- LinkedHashSet 在 HashSet 的基础上，记录了元素的添加顺序

```
Set<String> set = new LinkedHashSet<>();
set.add("Jack");
set.add("Rose");
set.add("Jim");
set.add("Jack");

// [Jack, Rose, Jim]
System.out.println(set);

for (String str : set) {
    System.out.println(str);
} // Jack Rose Jim

set.remove("Rose");
// [Jack, Jim]
System.out.println(set);
```

小码哥教育 SEEMYGO

TreeSet

- TreeSet 要求元素必须具备可比较性，默认按照从小到大的顺序遍历元素

```
Set<String> set = new TreeSet<>();
set.add("Jack");
set.add("Rose");
set.add("Jim");
set.add("Kate");
set.add("Rose");
set.add("Larry");

// [Jack, Jim, Kate, Larry, Rose]
System.out.println(set);

for (String str : set) {
    System.out.println(str);
} // Jack Jim Kate Larry Rose
```

小码哥教育 SEEMYGO

TreeSet – 自定义比较方式

```
Set<Integer> set = new TreeSet<>((i1, i2) -> i2 - i1);
set.add(33);
set.add(11);
set.add(55);
set.add(22);
set.add(44);

// [55, 44, 33, 22, 11]
System.out.println(set);

for (Integer i : set) {
    System.out.println(i);
} // 55 44 33 22 11
```

小码哥教育 SEEMYGO

HashMap

- HashMap 存储的是键值对（key-value），Map 译为“映射”，有些编程语言中叫做“字典”

```
Map<String, Integer> map = new HashMap<>();
map.put("Jack", 11);
map.put("Rose", 22);
map.put("Jim", 22);
map.put("Jack", 33);
map.put("Kate", 11);

// 4
System.out.println(map.size());
// 33
System.out.println(map.get("Jack"));
// {Kate=11, Rose=22, Jack=33, Jim=22}
System.out.println(map);

map.remove("Rose");
// {Kate=11, Jack=33, Jim=22}
System.out.println(map);
```

小码哥教育 SEEMYGO

HashMap – 遍历

```
Map<String, Integer> map = new HashMap<>();
map.put("Jack", 11);
map.put("Rose", 22);
map.put("Jim", 33);
```

```
Set<String> keys = map.keySet();
for (String key : keys) {
    // 先遍历key，又通过key去找到value（整体效率低）
    System.out.println(key + "=" + map.get(key));
} // Rose=22 Jack=11 Jim=33
```

```
// 只能遍历value
Collection<Integer> values = map.values();
for (Integer value : values) {
    System.out.println(value);
} // 22 11 33
```

小码哥教育 SEEMYGO

HashMap – 遍历

```
// 推荐
Set<Entry<String, Integer>> entries = map.entrySet();
for (Entry<String, Integer> entry : entries) {
    System.out.println(entry.getKey() + "=" + entry.getValue());
} // Rose=22 Jack=11 Jim=33

// 推荐
map.forEach((key, value) -> {
    System.out.println(key + "=" + value);
}); // Rose=22 Jack=11 Jim=33
```

小码哥教育 SEEMYGO

LinkedHashMap

- LinkedHashMap 在 HashMap 的基础上，记录了元素的添加顺序

```
Map<String, Integer> map = new LinkedHashMap<>();
map.put("Jack", 11);
map.put("Rose", 22);
map.put("Jim", 33);
map.put("Kate", 44);

// {Jack=11, Rose=22, Jim=33, Kate=44}
System.out.println(map);

map.forEach((k, v) -> {
    System.out.println(k + "=" + v);
}); // Jack=11 Rose=22 Jim=33 Kate=44

map.remove("Rose");
// {Jack=11, Jim=33, Kate=44}
System.out.println(map);
```

小码哥教育 SEEMYGO

TreeMap

- TreeMap 要求 key 必须具备可比较性，默认按照从小到大的顺序遍历 key

```
Map<String, Integer> map = new TreeMap<>();
map.put("Jack", 11);
map.put("Rose", 22);
map.put("Jim", 33);
map.put("Kate", 44);
map.put("Larry", 55);

// {Jack=11, Jim=33, Kate=44, Larry=55, Rose=22}
System.out.println(map);

map.forEach((k, v) -> {
    System.out.println(k + "=" + v);
}); // Jack=11 Jim=33 Kate=44 Larry=55 Rose=22
```

小码哥教育 SEEMYGO

List vs Set vs Map

- List 的特点
  - 可以存储重复的元素
  - 可以通过索引访问元素
- Set 的特点
  - 不可以存储重复的元素
  - 不可以通过索引访问元素
- Map 的特点
  - 不可以存储重复的 key，可以存储重复的 value
  - 不可以通过索引访问 key-value
- Set 的底层是基于 Map 实现的
  - HashSet 底层用了 HashMap
  - LinkedHashSet 底层用了 LinkedHashMap
  - TreeSet 底层用了TreeMap

小码哥教育 SEEMYGO

java.util.Collections

- java.util.Collections 是一个常用的集合工具类，提供了很多实用的静态方法

```
void sort(List<T> list)
void sort(List<T> list, Comparator<? super T> c)
T max(Collection<? extends T> coll)
T max(Collection<? extends T> coll, Comparator<? super T> comp)
T min(Collection<? extends T> coll)
T min(Collection<? extends T> coll, Comparator<? super T> comp)
boolean addAll(Collection<? super T> c, T... elements)
void reverse(List<?> list)
Comparator<T> reverseOrder()
void shuffle(List<?> list) // 随机打乱list中的元素顺序
void swap(List<?> list, int i, int j)
void fill(List<? super T> list, T obj) // 用obj填满list
void copy(List<? super T> dest, List<? extends T> src)
boolean replaceAll(List<T> list, T oldVal, T newVal)
```