第6章 集合

老师：李沁洳

/6.1

集合概述

6.1 集合概述

Collection

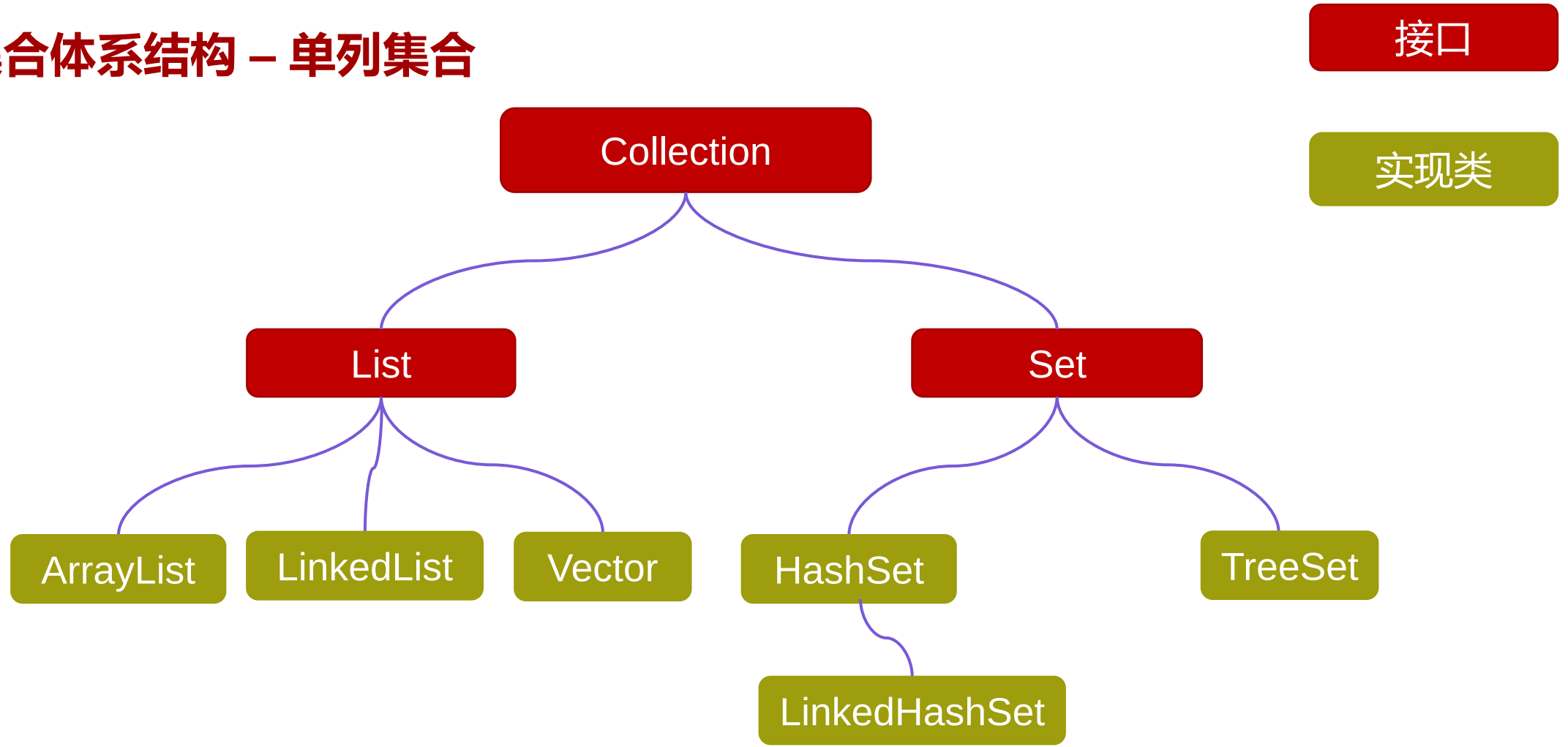
Map

单列集合

双列集合

6.1 集合概述

集合体系结构 – 单列集合



6.1 集合概述

集合体系结构 – 单列集合

List系列集合：添加的元素是有序，可重复，有索引

Set系列集合：添加的元素是无序，不重复，无索引

/6.2

Collection接口

6.2 Collection接口

Collection

Collection是单列集合的祖宗接口，它的功能是全部单列集合都可以继承使用的。

方法名称	说明
public boolean add(E e)	把给定的对象添加到当前集合中
public void clear()	清空集合中所有的元素
public boolean remove(E e)	把给定的对象在当前集合中删除
public boolean contains(Object obj)	判断当前集合中是否包含给定的对象
public boolean isEmpty()	判断当前集合是否为空
public int size()	返回集合中元素的个数/集合的长度

6.2 Collection接口

Collection的遍历方式

迭代器遍历

增强for遍历

Lambda表达式遍历

6.2 迭代器

迭代器遍历

迭代器在Java中的类是Iterator,迭代器是集合专用的遍历模式。

Collection集合获取迭代器

方法名称	说明
Iterator<E> iterator()	返回迭代器对象，默认指向当前集合的0索引

Iterator中的常用方法

方法名称	说明
boolean hasNext()	判断当前位置是否有元素，有元素返回true,没有元素返回false
E next()	获取当前位置的元素，并将迭代器对象移向下一个位置

6.2 迭代器

迭代器不依赖索引


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

创建指针

判断是否有元素

获取元素

移动指针

6.2 迭代器

细节注意点：

1. 报错NoSuchElementException
2. 迭代器遍历完毕，指针不会复位
3. 循环中只能用一次next方法
4. 迭代器遍历时，不能用集合的方法进行增加或者删除。

6.2 foreach循环

增强for遍历

- 增强for的底层就是迭代器，为了简化迭代器的代码书写的
- 它是JDK5之后出现的，其内部原理就是Iterator迭代器
- 所有的单列集合和数组才能用增强for进行遍历。

格式

```
for(元素的数据类型 变量名: 数组或者集合){

}
```

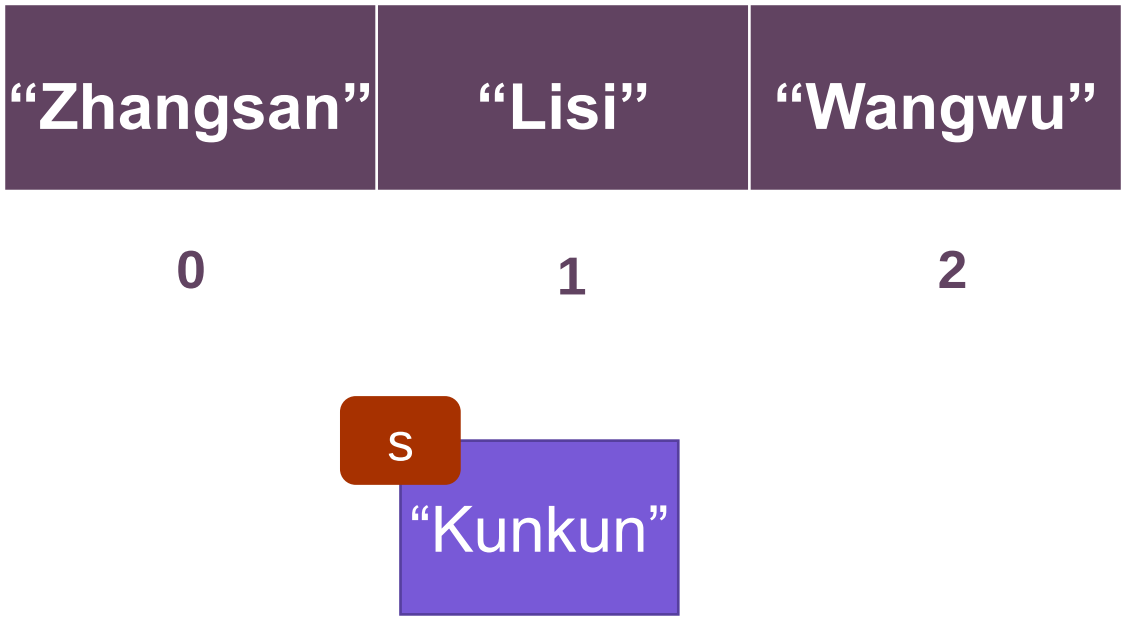
```
for(String s: list){
System.out.println(s);
}
```

6.2 foreach循环

细节

- 修改增强for中的变量，不会改变集合中原本的数据。

```
for (String s : arrList) {
    s = "Kunkun";
    System.out.println(s);
}
```



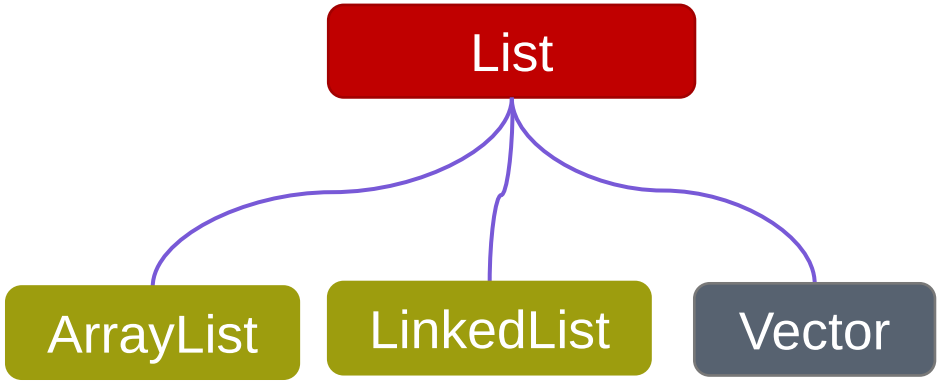
/6.3

List接口

6.3.1 List接口

List集合的特点

- 有序：存和取的元素顺序一致。
- 有索引：可以通过索引操作元素
- 可重复：存储的元素可以重复



List集合的特有方法

- Collection的方法List都继承了
- List集合因为有索引，所以多了很多索引的操作方法。

6.3.1 List接口

List集合的特有方法

- Collection的方法List都继承了
- List集合因为有索引，所以多了很多索引的操作方法。

方法名称	说明
void add(int index, E element)	在此集合中的指定位置插入指定的元素
E remove(int index)	删除指定索引处的元素，返回被删除的元素
E set(int index, E element)	修改指定索引处的元素，返回被修改的元素
E get(int index)	返回指定索引处的元素

6.3.1 List接口

List集合的遍历方式

迭代器遍历

在遍历的过程中需要删除元素，请使用迭代器

列表迭代器遍历

在遍历的过程中需要添加元素，请使用列表迭代器

foreach循环

Lambda表达式遍历

仅仅想遍历，那么使用foreach或Lambda表达式

普通for循环（因为List集合存在索引）

如果遍历的时候想操作索引，可以用普通for

各种数据结构的特点和作用

- 栈：后进先出，先进后出。
- 队列：先进先出，后进后出。
- 数组：内存连续区域，查询快，增删慢。
- 链表：元素式游离的，查询慢，首尾操作极快。

6.3.2 ArrayList集合

ArrayList集合的创建：

```
ArrayList arrList = new ArrayList();
```

没有指定集合中存储什么类型的元素，会产生安全隐患

```
ArrayList<String> arrList = new ArrayList<>();
```

6.3.3 LinkedList集合

- 底层数据结构是双链表，查询慢，增删快，但是如果操作的是首尾元素，速度也是极快的。
- LinkedList本身多了很多直接操作首尾元素的特有API。

双向链表

6.3.3 LinkedList集合

底层数据结构是双链表，查询慢，首尾操作的速度是极快的，所以多了很多首尾操作的特有API。

特有方法	说明
public void addFirst(E e)	在该列表开头插入指定的元素
public void addLast(E e)	将指定的元素追加到此列表的末尾
public E getFirst()	返回此列表中的第一个元素
public E getLast()	返回此列表中的最后一个元素
public E removeFirst()	从此列表中删除并返回第一个元素
public E removeLast()	从此列表中删除并返回最后一个元素

6.3.3 LinkedList集合

LinkedList集合的创建：

```
LinkedList lilist = new LinkedList();
```

没有指定集合中存储什么类型的元素，会产生安全隐患

```
LinkedList<String> list = new LinkedList<>();
```

/6.4

Set接口

6.4.1 Set集合

Set系列集合

- 无序：存取顺序不一致
- 不重复：可以去除重复
- 无索引：没有带索引的方法，所以不能使用普通for循环遍历，也不能通过索引来获取元素

Set集合的实现类

- HashSet: 无序，不重复，无索引
- LinkedHashSet: 有序，不重复，无索引
- TreeSet: 可排序，不重复，无索引

Set接口中的方法基本上与Collection的API一致

6.4.2 HashSet集合

HashSet底层原理

- HashSet集合底层采取哈希表存储数据
- 哈希表是一种对于增删改查数据性能都较好的结构

哈希表组成

- JDK8之前：数组和链表
- JDK8开始：数组+链表+红黑树

6.4.2 HashSet集合

哈希值

哈希值：对象的整数表现形式

int index = (数组长度-1) & 哈希值

6.4.2 HashSet集合

哈希值

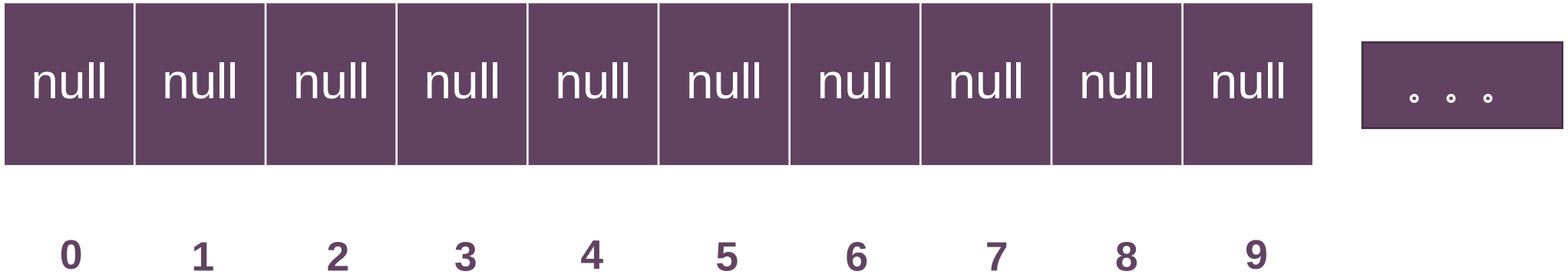
- 根据hashCode方法算出来的int类型的整数
- 该方法定义在Object类中，所有对象都可以调用，默认使用地址值进行计算
- 一般情况下，会重写hashCode方法，利用对象内部的属性值计算哈希值

对象的哈希值特点

- 如果没有重写hashCode方法，不同对象计算出的哈希值是不同的
- 如果已经重写hashCode方法，不同的对象只要属性值相同，计算出来的哈希值就是一样的。
- 在小部分情况下，不同的属性或者不同的地址值计算出来的哈希值也有可能一样。（哈希碰撞）

6.4.2 HashSet集合

HashSet JDK8以后底层原理



1. 创建一个默认长度16，默认加载因子为0.75的数组，数组名table

```
HashSet<String> hm = new HashSet<>();
```

2. 根据元素的哈希值跟数组的长度计算出应存入的位置，也就是**hashCode**方法
3. 判断当前位置是否为null，如果是null直接存入
4. 如果位置不为null，表示有元素，则调用**equals**方法比较属性值
5. 一样：不存　　不一样：存入数组，形成链表， JDK8以后新元素直接挂靠在老元素下面

6.4.2 HashSet集合

HashSet JDK8以后底层原理

如果集合中存储的是自定义对象，必须要重写hashCode和equals方法

注意：如果不重写hashCode和equals，那么在底层都是用地址值进行比较的

6.4.2 HashSet集合

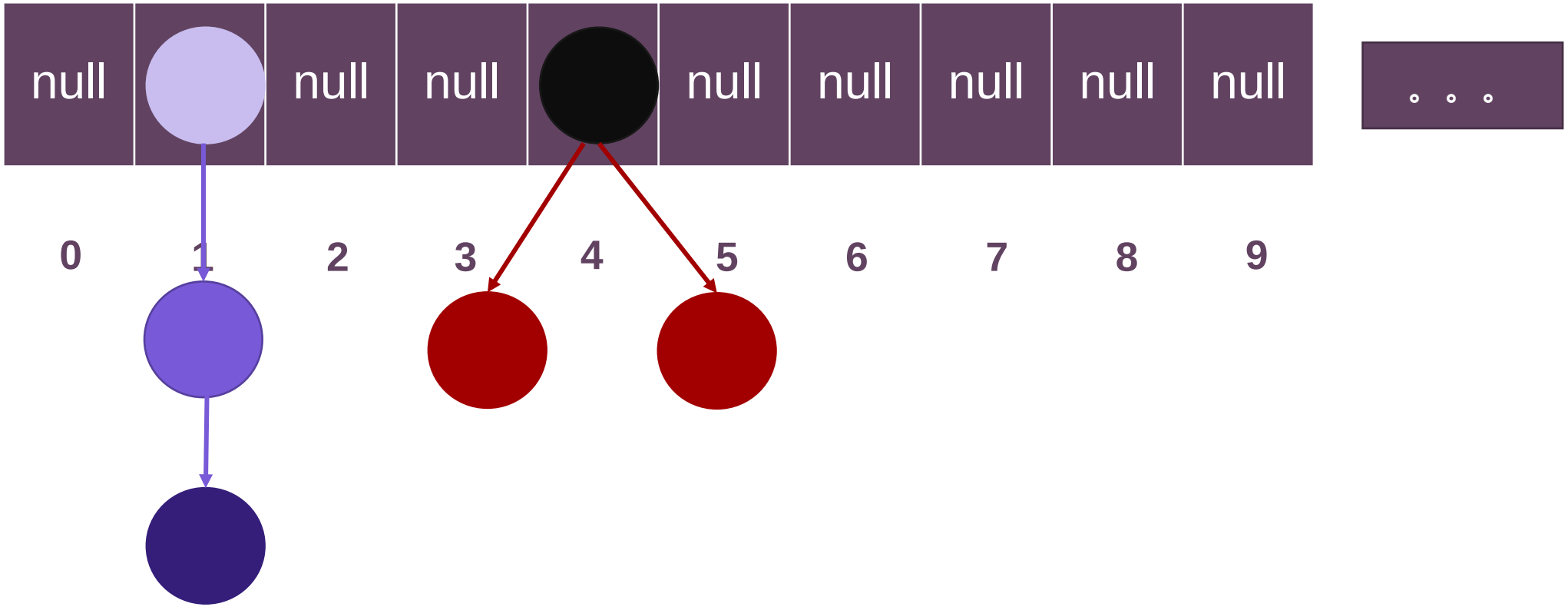
HashSet 的三个问题

问题1：HashSet为什么存和取得顺序不一样?

问题2：HashSet为什么没有索引?

问题3：HashSet是利用什么机制保证数据去重得?

6.4.2 HashSet集合



6.4.2 HashSet集合

练习：利用HashSet 集合去除重复元素

需求：创建一个存储学生对象得集合，存储多个学生对象。

使用程序实现在控制台遍历该集合。

要求：学生对象得成员变量值相同，我们就认为是同一个对象。

6.4.2 LinkedHashSet集合

LinkedHashSet底层原理

- 有序，不重复，无索引。
- 这里的有序指的是保证存储和取出的元素顺序一致。
- 原理：底层数据结构是哈希表，只是每个元素又额外的多了一个双链表的机制记录存储的顺序。

6.4.2 LinkedHashSet集合

1. HashSet

- 无序，不重复，无索引，存取顺序不一致
- 底层从JDK8开始：数组+链表+红黑树
- 使用hashCode和equals去重

2. LinkedHashSet集合的特点和是怎么样的?

- 有序，不重复，无索引
- 底层基于哈希表，使用双链表记录添加顺序

3. 在以后如果要数据去重，我们使用哪个?

- 默认使用HashSet
- 如果要求去重且存取有序，才使用LinkedHashSet

6.4.3 TreeSet集合

TreeSet的特点

- 不重复，无索引，可排序
- 可排序：按照元素的默认规则（有小到大）排序。
- TreeSet集合底层是基于红黑树的数据结构实现排序的，增删改查性能都较好。

6.4.3 TreeSet集合

练习：TreeSet对象排序练习题

需求：存储整数并进行排序

6.4.3 TreeSet集合

TreeSet 集合默认的规则

- 对于数值类型：Integer， Double，默认按照从小到达的顺序进行排序
- 对于字符，字符串类型：按照字符在ASCII码表中的数字升序进行排序。
- 如果说TreeSet里面存的是自定义对象?

6.4.3 TreeSet集合

练习：TreeSet对象排序练习题

需求：创建TreeSet集合，并添加3个学生对象

学生对象属性：

姓名，年龄。

要求按照学生的年龄进行排序

同年龄按照姓名字母排列（暂不考虑中文）

同姓名，同年龄认为是同一个人

6.4.3 TreeSet集合

```java
@Override
public int compareTo(Student o) {
    //指定排序的规则
    //只看年龄，我想要按照年龄的升序进行排列
    int result = this.getAge() - o.getAge();
    return result;
}
```

this: 表示当前要添加的元素

o:表示已经在红黑树中存在的元素

返回值:

负数：就认为当前要添加的元素是小的，存左边

正数：要添加的元素是大的，存右边

0：认为要添加的元素已经存在，舍弃

6.4.3 TreeSet集合

TreeSet 的两种比较方式

方式一：

默认排序/自然排序： Javabean类实现Comparable接口指定比较规则

方式二：

比较器排序： 创建TreeSet对象时候，传递比较器Comparator指定规则

使用规则：默认使用第一种，如果第一种不能满足当前需求，就使用第二种

6.4.3 TreeSet集合

练习：TreeSet对象排序练习题

需求：请自行选择比较器排序和自然排序两种方式

要求：存入四个字符串，“c”，“ab”，“df”，“qwer”

按照长度排序，如果一样长则按照首字母排序

6.4.3 TreeSet集合

1. TreeSet集合的特点是怎么样的?

- 可排序，不重复，无索引
- 底层基于红黑树实现排序，增删改查性能好

2. TreeSet集合自定义排序规则有几种方式

- 方式一：Javabean类实现Comparable接口，指定比较规则
- 方式二：创建集合时，自定义Comparator比较器对象，指定比较规则

单列集合

1. 如果想要集合中的元素可重复

- 用ArrayList集合，基于数组的（用的最多）

2. 如果想要集合中的元素可重复，而且当前的增删操作明显多于查询

- 用LinkedList集合，基于链表的。

3. 如果想对集合中的元素去重

- 用HashSet集合，基于哈希表的（用的最多）

4. 如果相对集合中的元素去重，而且保证存取顺序

- 用LinkedHashSet集合，基于哈希表和双链表，效率低于HashSet。

5. 如果想对集合中的元素进行排序

- 用TreeSet集合，基于红黑树。

/6.5

Map接口

6.5.1 Map接口

双列集合

键和值之间是一一对应的关系

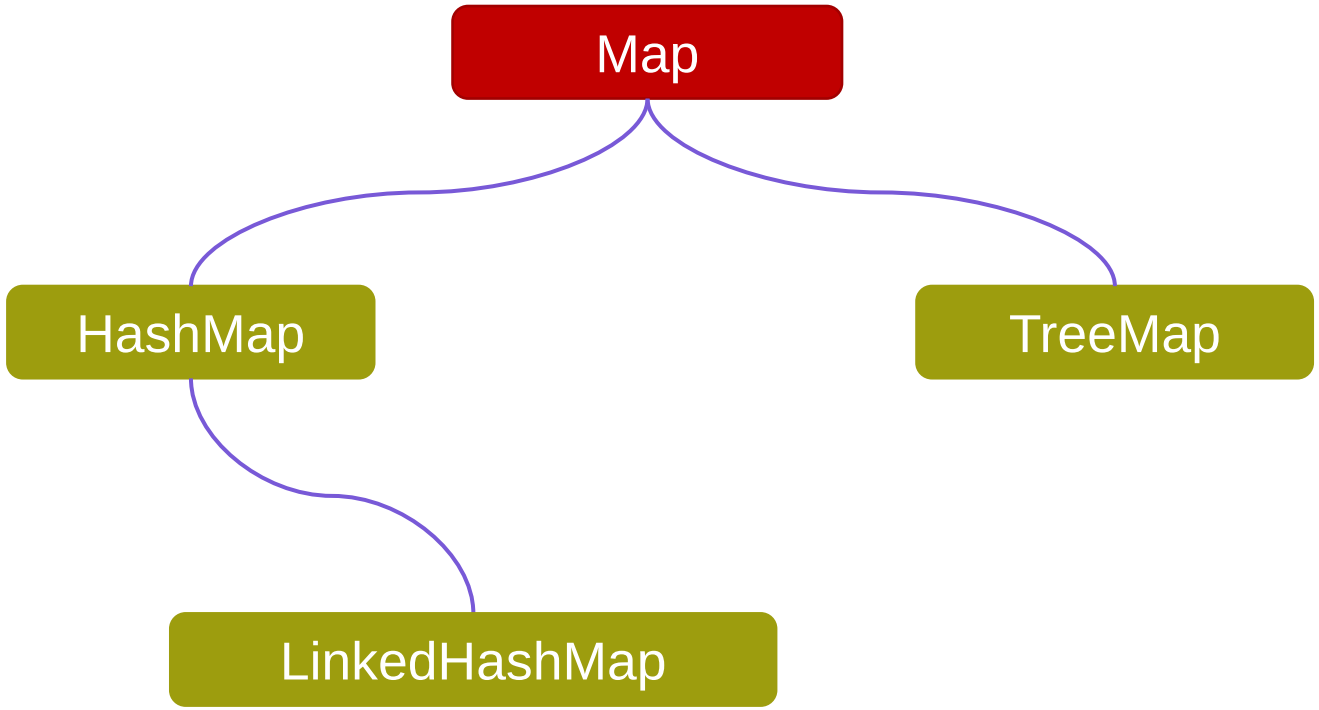
- 键值对
- 键值对对象
- entry

6.5.1 Map接口

双列集合的特点

1. 双列集合一次需要存一对数据，分别为键和值。
2. 键不能重复，值可以重复。
3. 键和值一一对应的，每一个键只能找到自己对应的值。
4. 键 + 值这个整体，我们称之为“键值对”或者“键值对对象”，在Java中叫做“Entry对象”

6.5.1 Map接口



6.5.1 Map接口

Map的常见API

Map是双列集合的顶层接口，它的功能是全部双列集合都可以继承使用的。

方法名称	说明
V put(K key, V value)	添加元素
V remove(Object key)	根据键删除键值对元素
void clear()	移除所有的键值对元素
boolean containsKey(Object key)	判断集合是否包含指定的键
boolean containsValue(Object value)	判断集合是否包含指定的值
boolean isEmpty()	判断集合是否为空
Int size()	集合的长度，也就是集合中键值对的个数

6.5.1 Map接口

Map的遍历方式

1. 键找值

2. 键值对

3. Lambda表达式

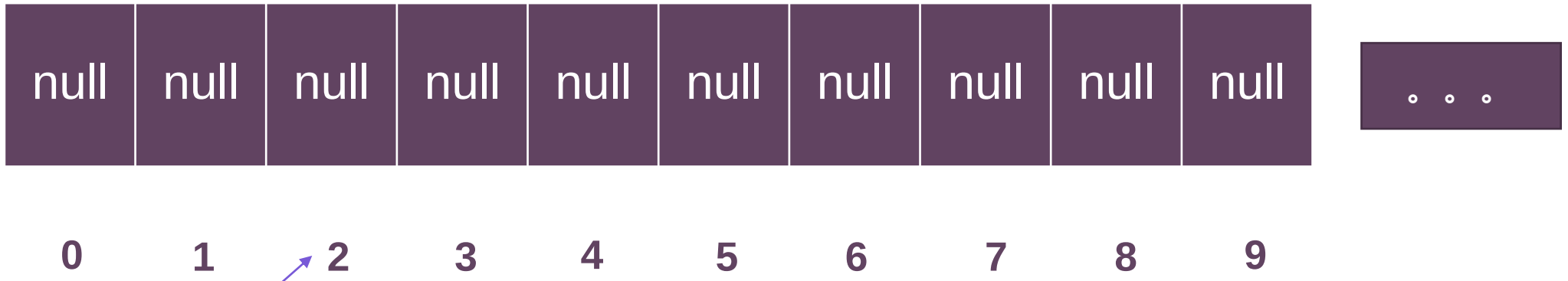
6.5.2 HashMap集合

HashMap的特点

1. HashMap是Map里面的一个实现类。
2. 没有额外需要学习的特有方法，直接使用Map里面的方法就可以了。
3. 特点都是由键决定的：无序，不重复，无索引
4. HashMap跟HashSet底层原理是一模一样的，都是哈希表结构。

6.5.2 HashMap集合

HashMap的底层原理



6.5.2 HashMap集合

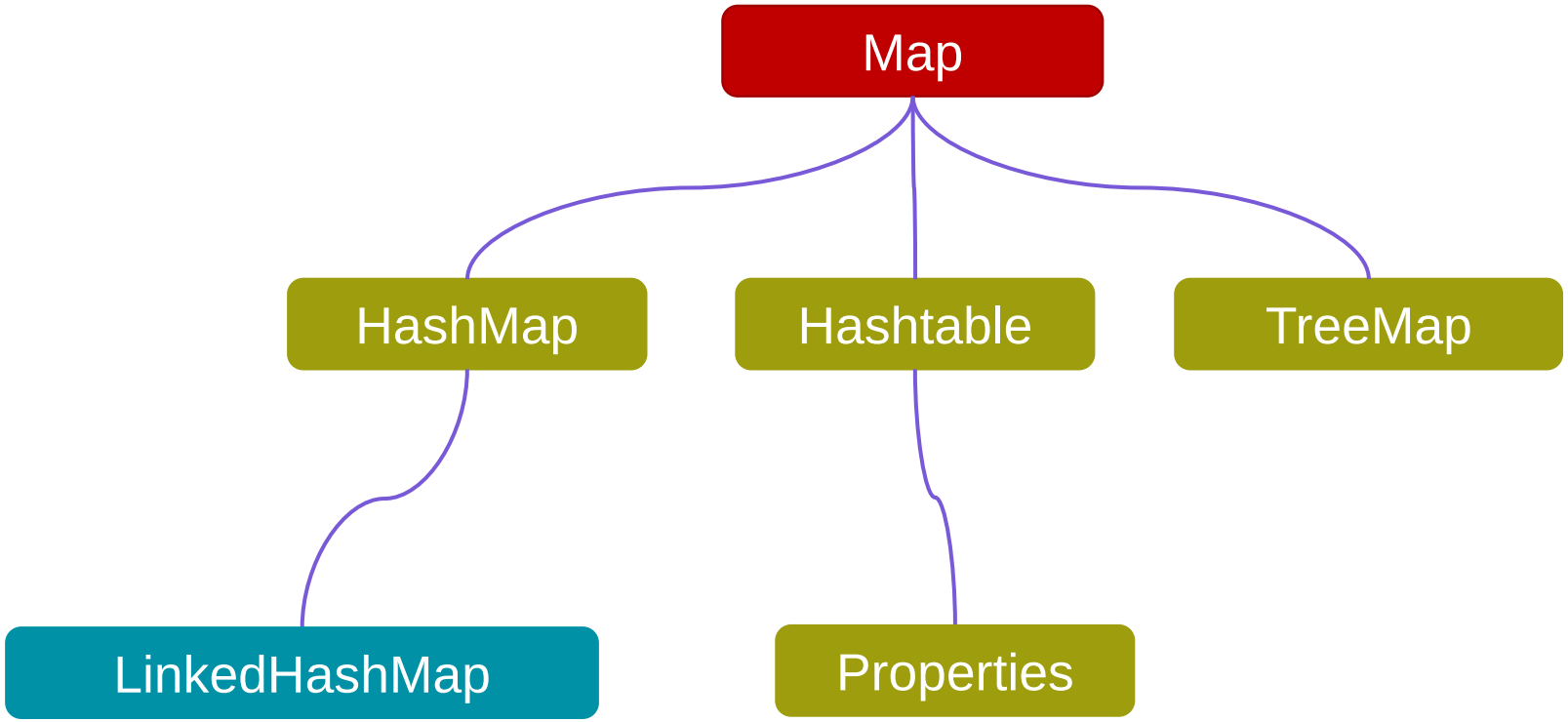
1. HashMap底层是哈希表结构

2. 依赖hashCode方法和equals方法保证**键的唯一**

3. 如果键存储的是自定义对象，需要重写hashCode和equals方法。

   如果值存储自定义对象，不需要重写hashCode和equals方法。

6.5.2 HashMap集合



6.5.2 HashMap集合

LinkedHashMap

- 由键决定：有序，不重复，无索引。
- 这里的有序指的是保证存储和取出的元素顺序一致
- 原理：底层数据结构是哈希表，只是每个键值对元素又额外的多了一个双链表的机制记录存储的顺序。

6.5.3 TreeMap集合

TreeMap

- TreeMap跟TreeSet底层原理一样，都是红黑树结构的。
- 由键决定特性：不重复，无索引，可排序。
- 可排序：对键进行排序。
- 注意：默认按照键的从小到大进行排序，也可以自己规定键的排序规则

代码书写两种排序规则

- 实现Comparable接口，指定比较规则。
- 创建集合时传递Comparator比较器对象，指定比较规则。

6.5.4 Properties集合

- Properties继承于Hashtable, 但Hashtable现在基本不用了。
- 表示一个持久的属性集。
- Properties类被许多Java类使用。

例如：在获取环境变量时它就作为System.getProperties()方法的返回值。

- 有两个专门的字符串存取方法

    1. getProperty()
    2. setProperty()

/6.6

泛型

6.6.1 泛型概述

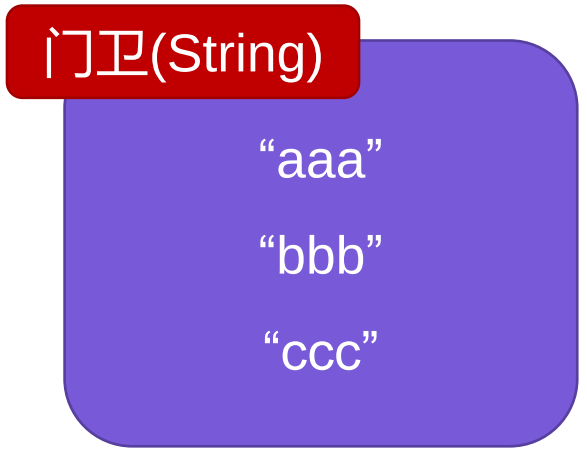
泛型深入

泛型：是JDK5中引入的特性，可以在编译阶段约束操作的数据类型，并进行检查。

泛型的格式：<数据类型>

注意：泛型只能支持引用数据类型。

```
ArrayList<String> list = new ArrayList<>();
list.add("aaa");
list.add("bbb");
list.add("ccc");
```



6.6.1 泛型

泛型的好处

- 统一数据类型。
- 把运行期间的问题提前到了编译期间，避免了强制类型转换可能出现的异常，因为在编译阶段类型就能确定下来。

6.6.1 泛型

泛型的好处

- 泛型中不能写基本数据类型
- 指定泛型的基本类型后，传递数据，可以传入该数据类型或者其子类类型
- 如果不写泛型，类型默认是Object.

6.6.1 泛型

泛型的好处

类后面	泛型类
方法上面	泛型方法
接口后面	泛型接口

6.6.2 泛型类和泛型对象

泛型类

使用场景：当一个类中，某个变量的数据类型不确定时，就可以定义带有泛型的类

格式：

```
修饰符 class 类名<类型>{


}
```

举例：

```
public class ArrayList<E>{
        创建该类对象时，E就确定类型
}
```

此处E可以理解为变量，但是不是用来记录数据的，而是记录数据的类型，可以写成：T, E, K, V等

6.6.3 泛型方法

泛型方法

方法中形参类型不确定时，可以使用类名后面定义的泛型<E>

```
public class MyArrayList <E> {
    public boolean add(E e){
        obj[size] = e;
        size++;
        return true;
    }
}
```

```
public class MyArrayList {
    public <E> boolean add(E e){
        obj[size] = e;
        size++;
        return true;
    }
}
```

6.6.3 泛型方法

泛型方法

方法中形参类型不确定时

方案1：使用类名后面定义的泛型　　所有方法都能用

方案2：在方法申明上定义自己的泛型　　只有本方法能用

6.6.3 泛型方法

泛型方法

格式：

```
修饰符 <类型> 返回值类型 方法名（类型 变量名）{


}
```

举例：

```
Public <T> void show(T t){


}
```

此处T可以理解为变量，但是不是用来记录数据的，而是记录数据的类型，可以写成：T, E, K, V等

6.6.4 泛型接口

泛型接口

格式：

```
修饰符 interface 接口名<类型>{


}
```

举例：

```
public interface List<E> {


}
```

重点：如何使用一个带泛型的接口

方法1：实现类给出具体类型

方法2：实现类延续泛型，创建对象时再确定

6.6.5 泛型的继承和通配符

泛型的继承和通配符

- 泛型不具备继承性，但是数据具备继承性

应用场景：

1. 如果我们在定义类，方法，接口的时候，如果类型不确定，就可以定义泛型类，泛型方法，泛型接口。
2. 如果类型不确定，但是能知道以后只能传递某个继承体系中的，就可以使用泛型的通配符。

关键点：可以限定类型的范围。

/6.7

Lambda表达式

6.7 Lambda表达式

函数式编程

函数式编程（Functional programming）是一种思想特点。

面向对象：先找对象，让对象做事情。

函数式编程思想，忽略面向对象的复杂语法，强调做什么，而不是谁去做。

而我们要学习的Lambda表达式就是函数式思想的体现。

6.7 Lambda表达式

Lambda表达式的标准格式

Lambda表达式是JDK8开始后的一种新语法形式。

```
() -> {

}
```

- （）对应着方法的形参
- -> 固定格式
- {} 对应着方法的方法体

6.7 Lambda表达式

Lambda表达式的标准格式

Lambda表达式是JDK8开始后的一种新语法形式。

```
Arrays.sort(arr, new Comparator<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o2-o1;
    }
});
```

```
Arrays.sort(arr, (Integer o1, Integer o2) -> {
    return o2 - o1;
});
```

注意点：

- Lambda表达式可以用来简化匿名内部类的书写
- Lambda表达式只能简化函数式接口的匿名内部类的书写
- 函数式接口：

  有且仅有一个抽象方法的接口叫做函数式接口，接口上方可以加@FunctionalInterface注解

6.7 Lambda表达式遍历

Lambda表达式遍历

得益于JDK 8开始的新技术Lambda表达式，提供了一种更简单，更直接的遍历集合的方式。

方法名称	说明
default void forEach(Consumer<? Super T> action):	结合lambda遍历集合

底层原理：

其实也会自己遍历集合，一次得到每一个元素，把得到的每一个元素，传递给下面的accept方法

6.7 Lambda表达式遍历

Lambda表达式的省略写法

省略核心：可推导，可省略

- Lambda参数类型可以省略不写。
- 如果只有一个参数，参数类型可以省略，同时（）也可以省略。
- 如果Lambda表达式的方法体只有一行，大括号，分号，return可以省略不写，需要同时省略。

6.7 Lambda表达式

1. Lambda表达式的基本作用

简化函数式接口的匿名内部类的写法。

2. Lambda表达式有什么使用前提?

必须是接口的匿名内部类，接口中只能由一个抽象方法

3. Lambda的好处?

Lambda是一个匿名函数，我们可以把Lambda表达式理解为一段可以传递的代码，它可以写出更简洁，更灵活的代码，作为一种更紧凑的代码风格，使Java语言表达能力得到了提升。