

集合



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

集合和数组类似，都是容器。 为什么用集合？

search.51job.com

☐ 全选 已选条件: java(全文) ☆ 收藏 批量申请

1-1.5万/月 广州 | 3-4年经验 | 大专 | 招5人 申请职位

☐ Java开发工程师 (微服务方向) 02-22发布 上海金指软件技术有限公司

1-1.8万/月 上海-浦东新区 | 3-4年经验 | 大专 | 招6人 民营企业 | 50-150人

五险一金 通讯补贴 绩效奖金 员工旅游 年终奖 专业培训 餐饮补贴 ... 互联网/电子商务

☐ Java开发工程师 02-22发布 用友汽车信息科技(上海)股份...

1.2-1.8万/月 北京-朝阳区 | 2年经验 | 本科 | 招3人 民营企业 | 1000-5000人

五险一金 交通补贴 餐饮补贴 出国机会 年终奖 定期体检 弹性工作 计算机软件

☐ JAVA开发工程师 02-22发布 杭州天迈网络科技有限公司

0.8-1.2万/月 杭州-江干区 | 2年经验 | 本科 | 招5人 民营企业 | 50-150人

五险一金 弹性工作 补充公积金 餐饮补贴 年终奖 员工旅游 互联网/电子商务

☐ JAVA 数据库开发工程师 02-22发布 成都虚谷伟业科技有限公司

6:34 5G

购物车 编辑 14

全部 (35) 降价 (8) 常买 (0) 分类 ~

Java基础案例教程 黑马程序员 9787115439376 选服务 ¥31.32 - 1 +

Java自学宝典 黑马程序员 9787302475415 选服务 ¥74.24 - 1 +

Java Web 程序开发入门 传智播客 高教产品研发部著 9787302387... 选服务 ¥25.81 1 +

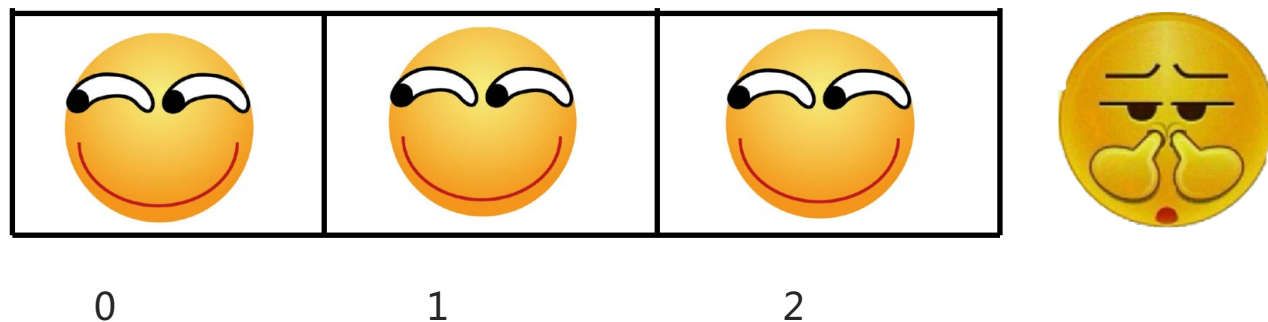
Java EE企业级应用开发教程 黑马程序员 9787115461025 选服务 ¥28.88 - 1 +

JD 京东自营 优惠券

全选 合计:¥0.00 去结算 (0)

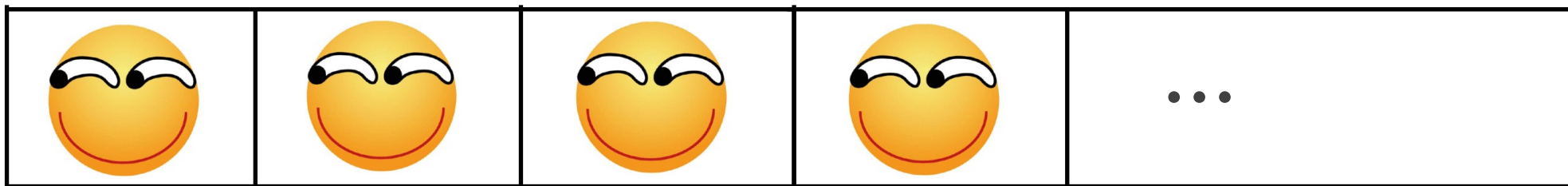
首页 分类 发现 购物车 35 我的

数组的特点



- 数组定义完成并启动后，**类型确定、长度固定**。
- 不适合元素的个数和类型不确定的业务场景，更不适合做需要增删数据操作。
- 数组的功能也比较的单一，处理数据的能力并不是很强大。

集合的特点



- 集合的大小不固定，启动后可以动态变化，类型也可以选择不固定。集合更像气球。
- 集合非常适合元素个数不能确定，且需要做元素的增删操作的场景。
- 同时，集合提供的种类特别的丰富，功能也是非常强大的，开发中集合用的更多。



总结

1、数组和集合的元素存储的个数问题。

- 数组定义后类型确定，长度固定
- 集合类型可以不固定，大小是可变的。

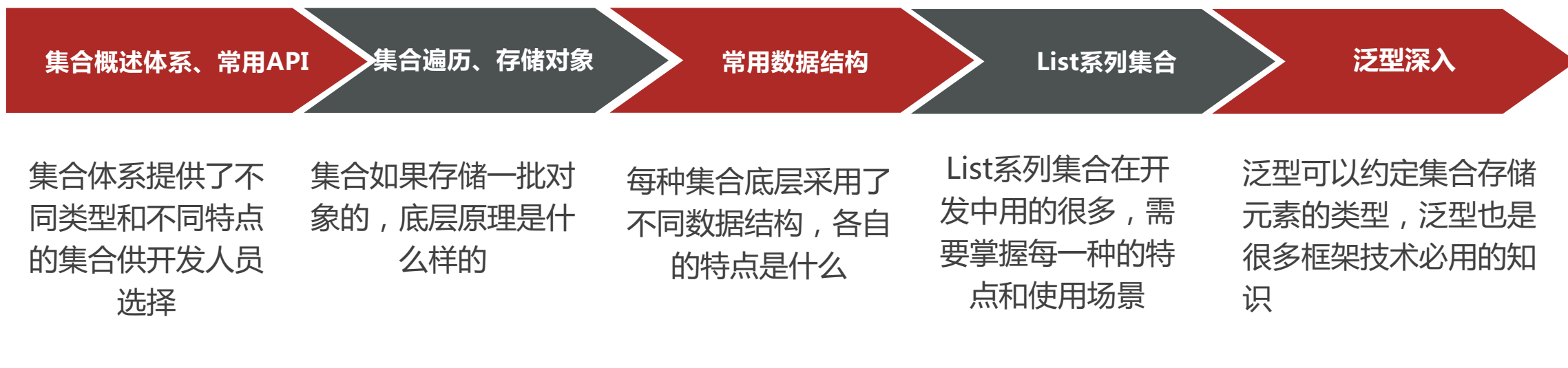
2、数组和集合存储元素的类型问题。

- 数组可以存储基本类型和引用类型的数据。
- 集合只能存储引用数据类型的数据。

3、数组和集合适合的场景

- 数组适合做数据个数和类型确定的场景。
- 集合适合做数据个数不确定，且要做增删元素的场景，集合种类更多，功能更强大。

关于集合同学们需要学会什么



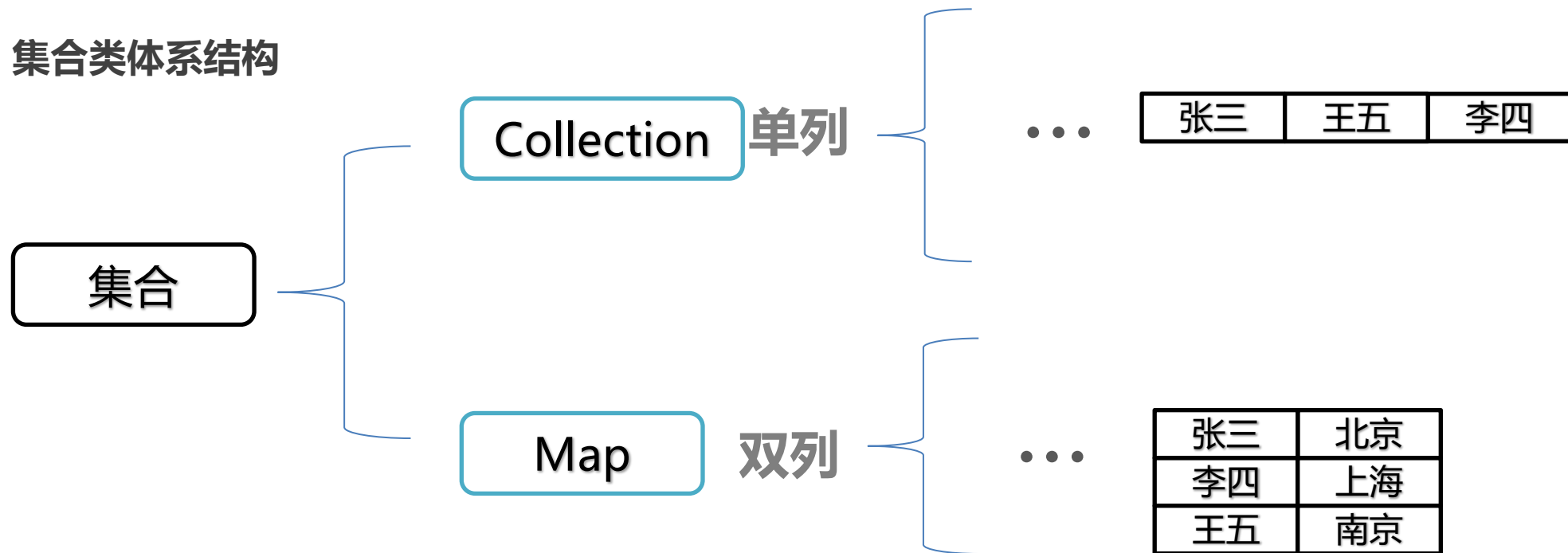


目录

Contents

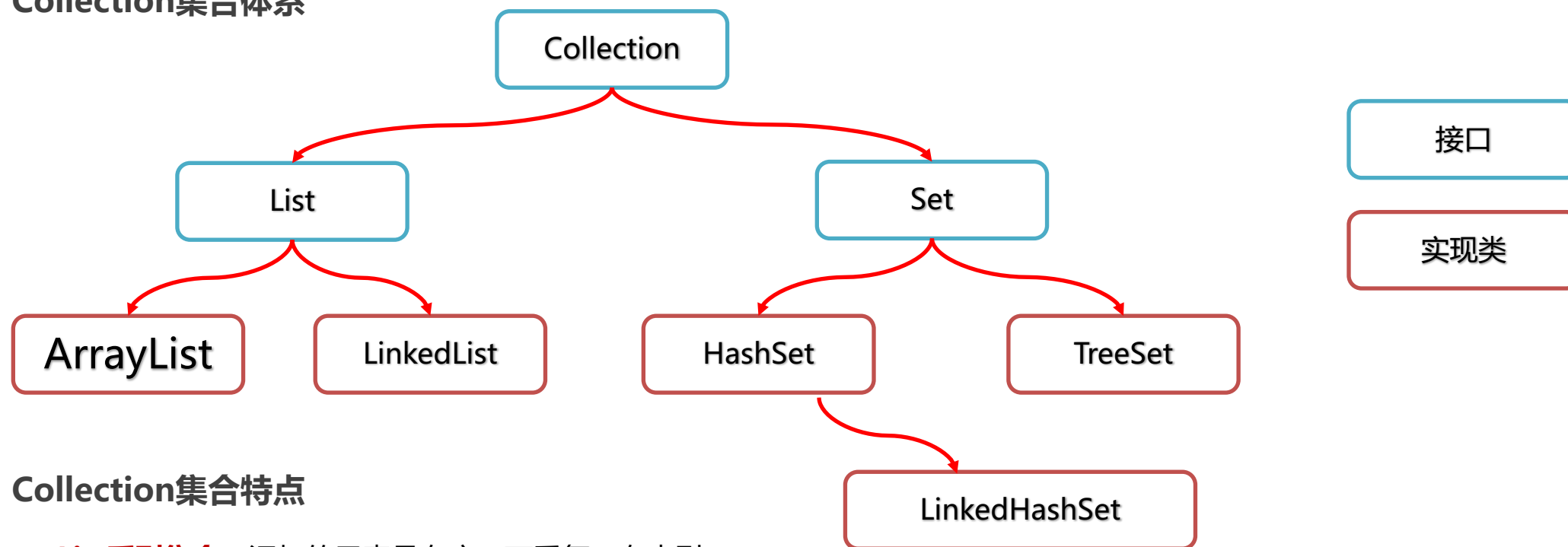
- **集合的体系特点**
- **Collection的常用方法**
- **集合的遍历方式**
- **集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

集合类体系结构



- Collection单列集合，每个元素（数据）只包含一个值。
- Map双列集合，每个元素包含两个值（键值对）。
- **注意：前期先掌握Collection集合体系的使用。**

Collection集合体系



Collection集合特点

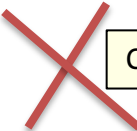
- **List系列集合**：添加的元素是有序、可重复、有索引。
 - ◆ ArrayList、LinkedList：有序、可重复、有索引。
- **Set系列集合**：添加的元素是无序、不重复、无索引。
 - ◆ HashSet: 无序、不重复、无索引；LinkedHashSet: **有序**、不重复、无索引。
 - ◆ TreeSet：**按照大小默认升序排序**、不重复、无索引。

泛型

- 集合都是泛型的形式，可以在编译阶段约束集合只能操作某种数据类型

```
Collection<String> lists = new ArrayList<String>();  
Collection<String> lists = new ArrayList<>(); // JDK 1.7开始后面的泛型类型声明可以省略不写
```

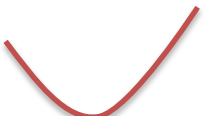
注意：集合和泛型都只能支持引用数据类型，不支持基本数据类型，所以集合中存储的元素都认为是对象。



```
Collection<int> lists = new ArrayList<>();
```

如果集合中要存储基本类型的数据怎么办？

```
// 存储基本类型使用包装类  
Collection<Integer> lists = new ArrayList<>();  
Collection<Double> lists = new ArrayList<>();
```





总结

1、集合的代表是？

- **Collection接口。**

2、Collection集合分了哪2大常用的集合体系？

- **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。**

3、如何约定集合存储数据的类型，需要注意什么？

- **集合支持泛型。**
- **集合和泛型不支持基本类型，只支持引用数据类型。**



目录

Contents

- 集合的体系特点
- **Collection集合常用API**
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入

Collection集合

- 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。

Collection API如下:

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。
public Object[] toArray()	把集合中的元素，存储到数组中



目录

Contents

- **Collection集合常用API**
- **Collection集合的遍历方式**
 - ◆ 方式一：迭代器
 - ◆ 方式二：foreach/增强for循环
 - ◆ 方式三：lambda表达式
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

迭代器遍历概述

- 遍历就是一个一个的把容器中的元素访问一遍。
- 迭代器在Java中的代表是**Iterator**，迭代器是集合的专用的遍历方式。

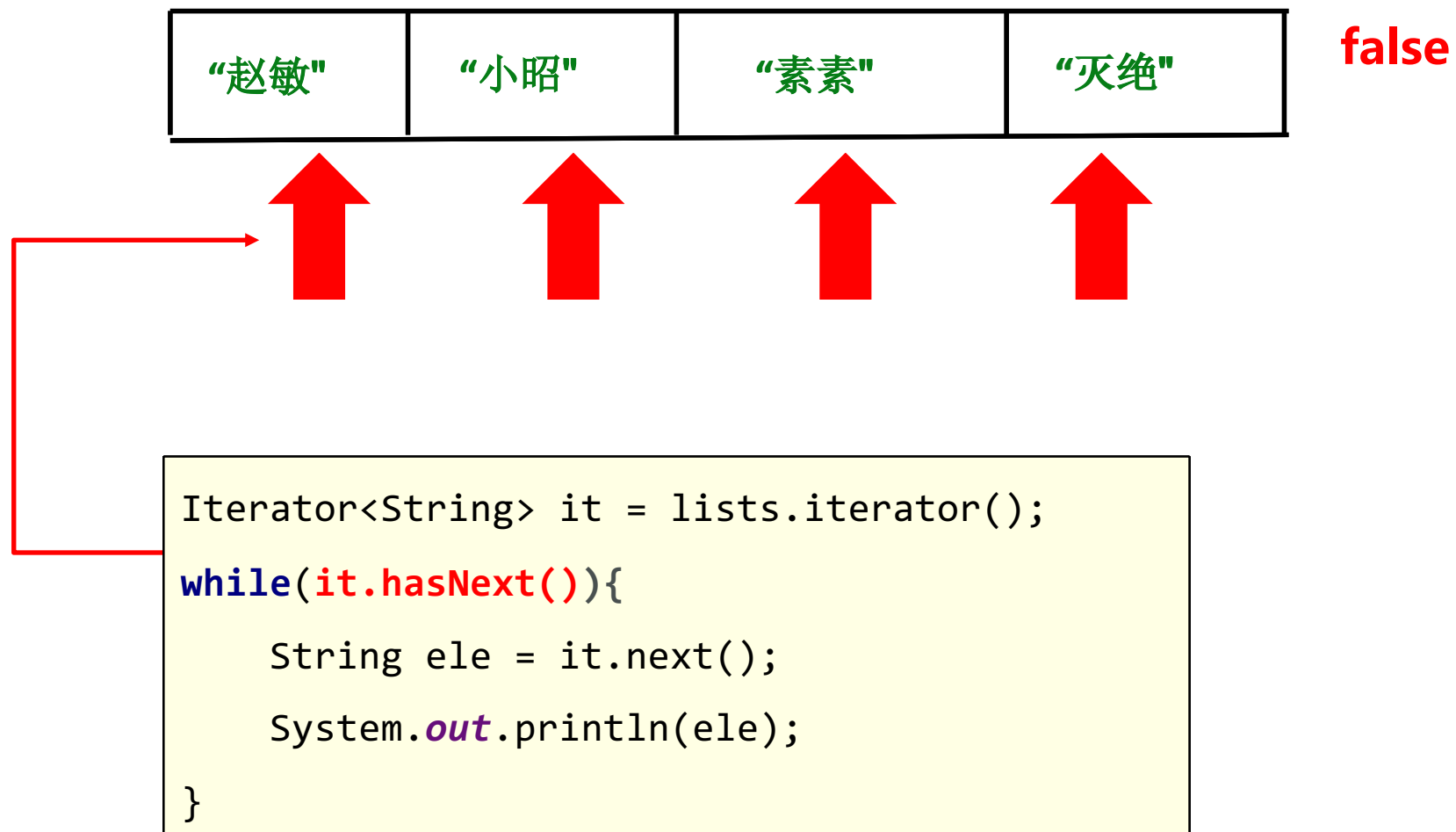
Collection集合获取迭代器

方法名称	说明
<code>Iterator<E> iterator()</code>	返回集合中的迭代器对象，该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
<code>boolean hasNext()</code>	询问当前位置是否有元素存在，存在返回true，不存在返回false
<code>E next()</code>	获取当前位置的元素，并同时迭代器对象移向下一个位置，注意防止取出越界。

迭代器执行流程





总结

1、迭代器的默认位置在哪里。

- `Iterator<E> iterator()`：得到迭代器对象，默认指向当前集合的索引 0

2、迭代器如果取元素越界会出现什么问题。

- 会出现 `NoSuchElementException` 异常。



目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
 - ◆ 方式一：迭代器
 - ◆ 方式二：foreach/增强for循环
 - ◆ 方式三：lambda表达式
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

增强for循环

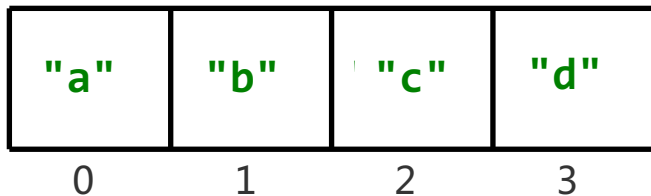
```
for(元素数据类型 变量名 : 数组或者Collection集合) {  
    //在此处使用变量即可，该变量就是元素  
}
```

- 增强for循环：既可以遍历集合也可以遍历数组。

举例说明

```
Collection<String> list = new ArrayList<>();  
...  
for(String ele : list) {  
    System.out.println(ele);  
}
```

增强for修改变量



"q"



修改第三方变量的值不会影响到集合中的元素

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



总结

1、增强for可以遍历哪些容器？

- 既可以遍历集合也可以遍历数组。

2、增强for的关键是记住它的遍历格式

```
for(元素数据类型 变量名 : 数组或者Collection集合) {  
    //在此处使用变量即可，该变量就是元素  
}
```



目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
 - ◆ 方式一：迭代器
 - ◆ 方式二：foreach/增强for循环
 - ◆ 方式三：lambda表达式
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

Lambda表达式遍历集合

- 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。

Collection结合Lambda遍历的API

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

```
Collection<String> lists = new ArrayList<>();  
...  
lists.forEach(new Consumer<String>() {  
    @Override  
    public void accept(String s) {  
        System.out.println(s);  
    }  
});
```



```
lists.forEach(s -> {  
    System.out.println(s);  
});  
// lists.forEach(s -> System.out.println(s));
```



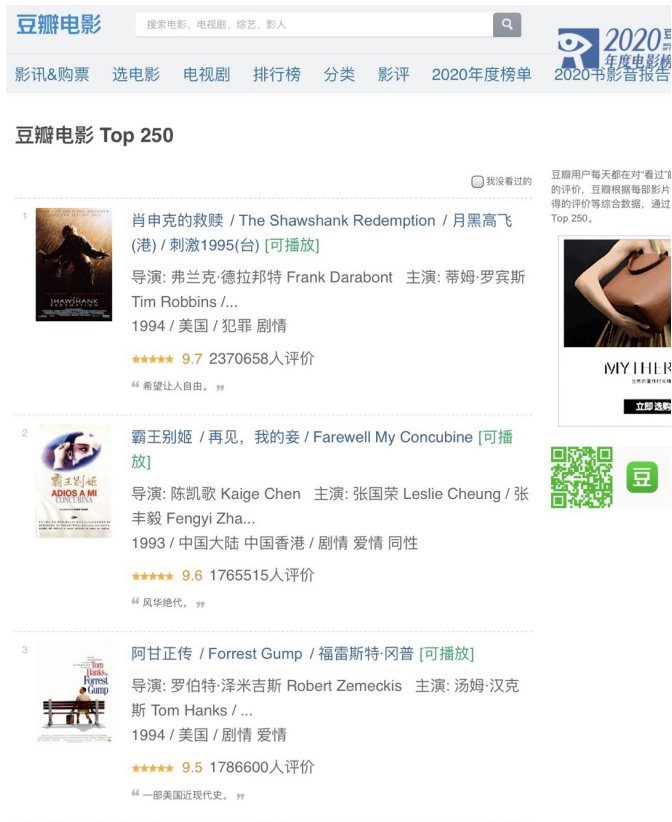
目录

Contents

- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- **Collection集合存储自定义类型的对象**
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入

案例

影片信息在程序中的表示



需求

- 某影院系统需要在后台存储上述三部电影，然后依次展示出来。

分析

- ①：定义一个电影类，定义一个集合存储电影对象。
- ②：创建3个电影对象，封装相关数据，把3个对象存入到集合中去。
- ③：遍历集合中的3个对象，输出相关信息。

```
public class Movie {  
    private String name;  
    private double score;  
    private String acotr;  
  
    public Movie(String name, double score, String acotr) {  
        this.name = name;  
        this.score = score;  
        this.acotr = acotr;  
    }  
    // ... getter + setter  
}
```

```
public class SystemDemo {  
    public static void main(String[] args) {  
        Collection <Movie> movies = new ArrayList<>();  
        movies.add(new Movie("《肖生克的救赎》", 9.7, "罗宾斯"));  
        movies.add(new Movie("《霸王别姬》", 9.6, "张国荣、张丰毅"));  
        movies.add(new Movie("《阿甘正传》", 9.5, "汤姆.汉克斯"));  
        System.out.println(movies);  
  
        for (Movie movie : movies) {  
            System.out.println("片名: " + movie.getName());  
            System.out.println("评分: " + movie.getScore());  
            System.out.println("主演: " + movie.getAcotr());  
        }  
    }  
}
```

栈内存

方法：main

...

...

movices

堆内存

《阿甘正传》", 9.5 , 汤姆.汉克斯

1aab2c

《霸王别姬》", 9.6 , 张国荣、张丰毅

2aab2c

《肖生克的救赎》", 9.7 , 罗宾斯

3aab2c

23afc5

movie

结论。



总结

1、集合中存储的是元素的什么信息?

- 集合中存储的是元素对象的地址。



目录

Contents

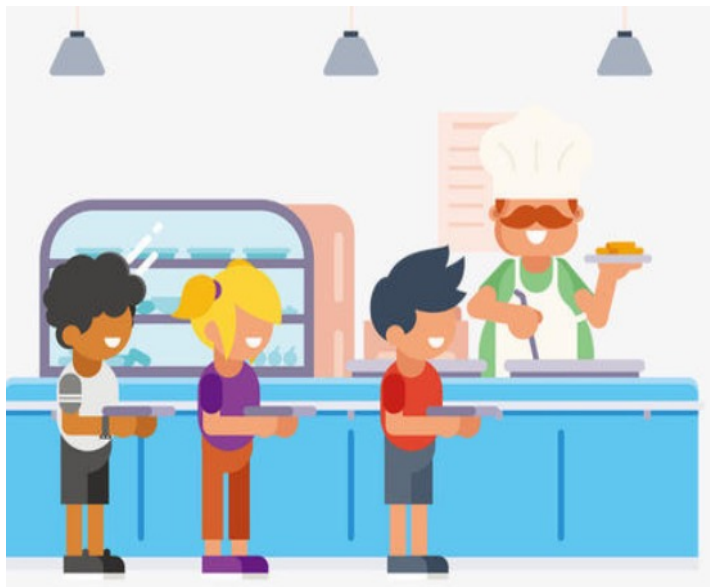
- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ **数据结构概述、栈、队列**
 - ◆ 数组
 - ◆ 链表
 - ◆ 二叉树、 二叉查找树
 - ◆ 平衡二叉树
 - ◆ 红黑树
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

数据结构概述

- 数据结构是计算机底层存储、组织数据的方式。是指数据相互之间是以什么方式排列在一起的。
- 通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率

常见的数据结构

- 栈
- 队列
- 数组
- 链表
- 二叉树
- 二叉查找树
- 平衡二叉树
- 红黑树
- ...



栈数据结构的执行特点

- 后进先出，先进后出



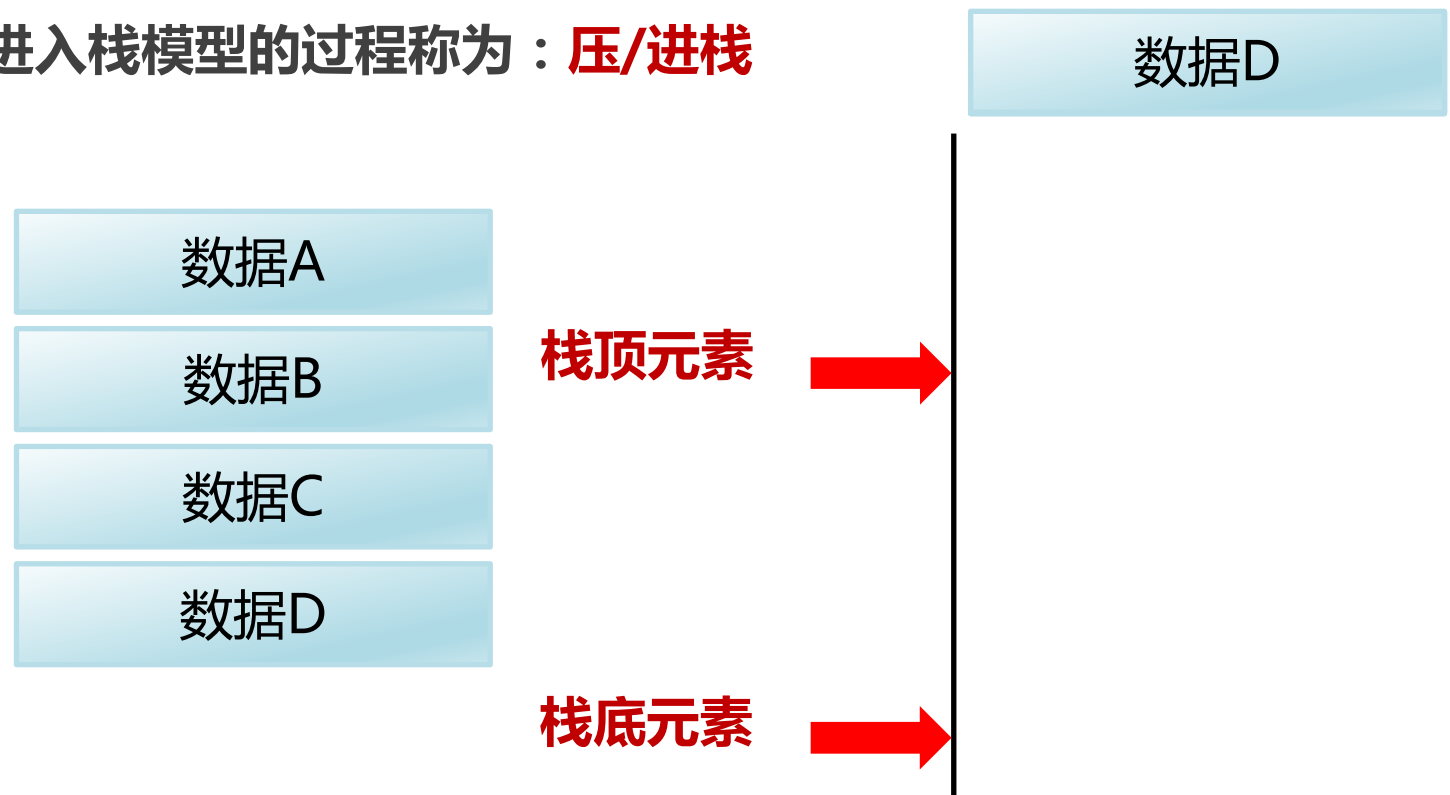
一端开口 **栈顶**

一端封闭 **栈底**

栈数据结构的执行特点

- 后进先出，先进后出

数据进入栈模型的过程称为：**压/进栈**



栈数据结构的执行特点

- 后进先出，先进后出

数据进入栈模型的过程称为：**压/进栈**

数据离开栈模型的过程称为：**弹/出栈**

栈顶元素



数据D

数据C

数据B

栈底元素



数据A

栈数据结构的执行特点

- 后进先出，先进后出



栈顶元素



数据D

数据C

数据B

栈底元素



数据A

常见数据结构之队列

- 先进先出，后进后出



一端开口 后端

一端开头 前端

常见数据结构之队列

- 先进先出，后进后出

数据从**后端**进入队列模型的过程称为：**入队列**



常见数据结构之队列

- 先进先出，后进后出

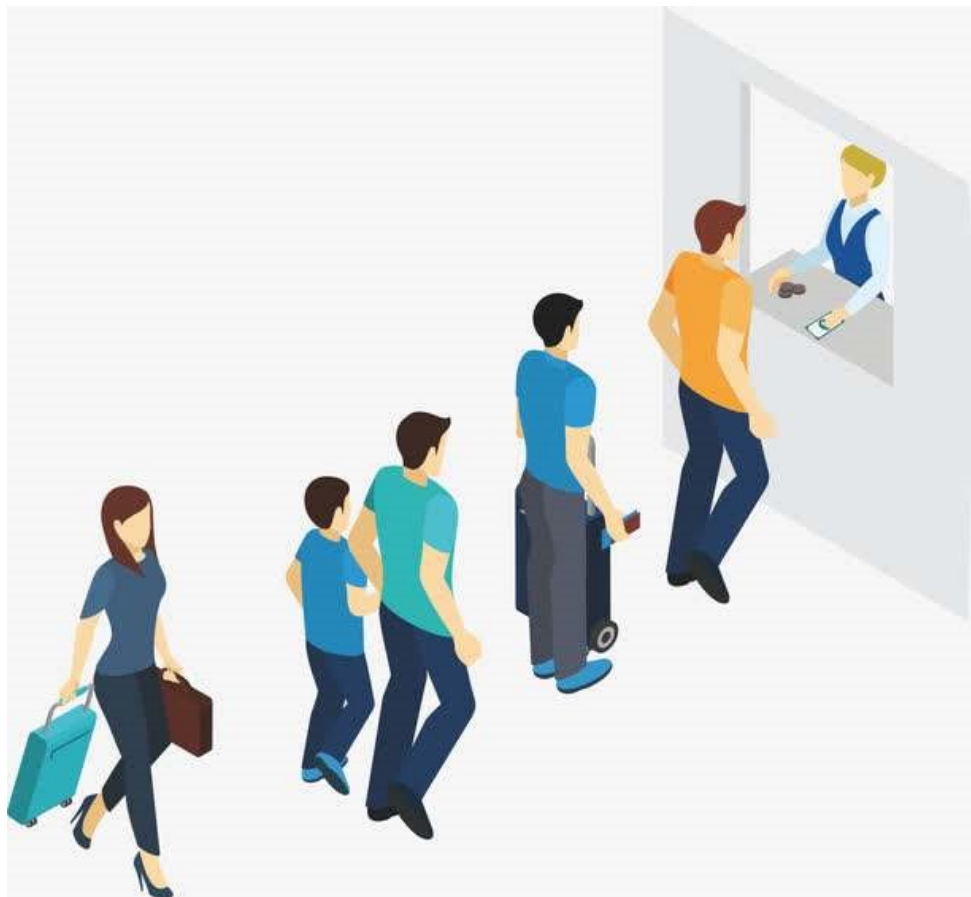
数据从**后端**进入队列模型的过程称为：**入队列**

数据从**前端**离开队列模型的过程称为：**出队列**



常见数据结构之队列

- 先进先出，后进后出



入队列方向



后端

数据D

数据C

数据B

数据A

前端

出队列方向



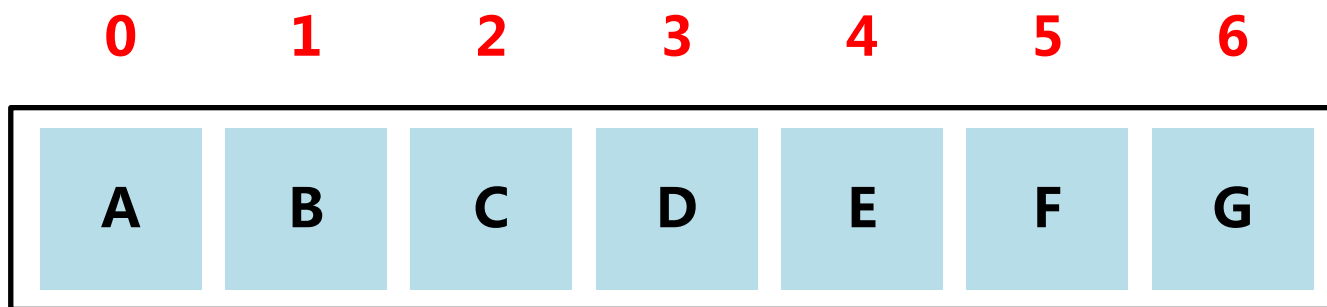


目录

Contents

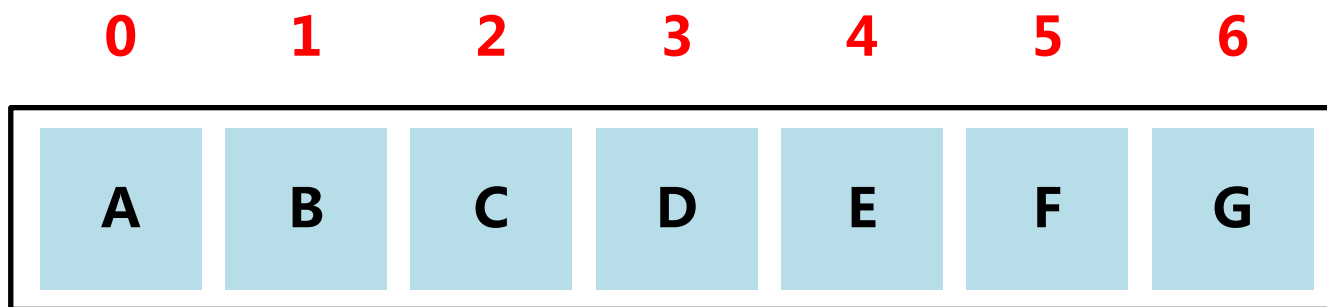
- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ 数据结构概述、栈、队列
 - ◆ **数组**
 - ◆ 链表
 - ◆ 二叉树、 二叉查找树
 - ◆ 平衡二叉树
 - ◆ 红黑树
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

常见数据结构之数组



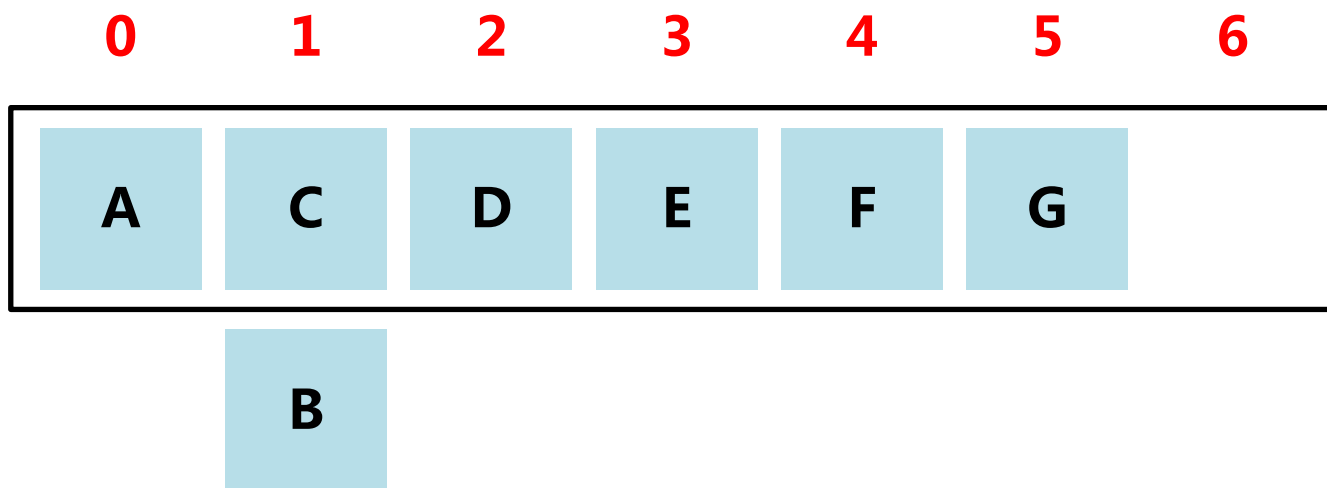
- **查询速度快**：查询数据通过地址值和索引定位，查询任意数据耗时相同。（元素在内存中是连续存储的）

常见数据结构之数组



- **查询速度快**：查询数据通过地址值和索引定位，查询任意数据耗时相同。（元素在内存中是连续存储的）
- **删除效率低**：要将原始数据删除，同时后面每个数据前移。

常见数据结构之数组



数组是一种**查询快，增删慢**的模型

- **查询速度快**：查询数据通过地址值和索引定位，查询任意数据耗时相同。（元素在内存中是连续存储的）
- **删除效率低**：要将原始数据删除，同时后面每个数据前移。
- **添加效率极低**：添加位置后的每个数据后移，再添加元素。



目录

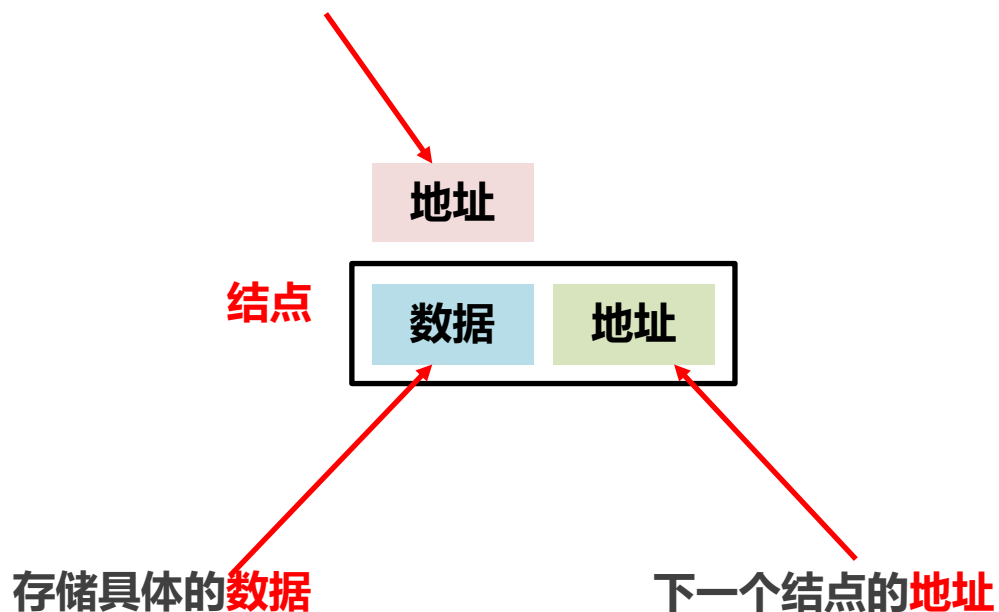
Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ 数据结构概述、栈、队列
 - ◆ 数组
 - ◆ **链表**
 - ◆ 二叉树、 二叉查找树
 - ◆ 平衡二叉树
 - ◆ 红黑树
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

链表的特点

- 链表中的元素是在内存中不连续存储的，每个元素节点包含数据值和下一个元素的地址。

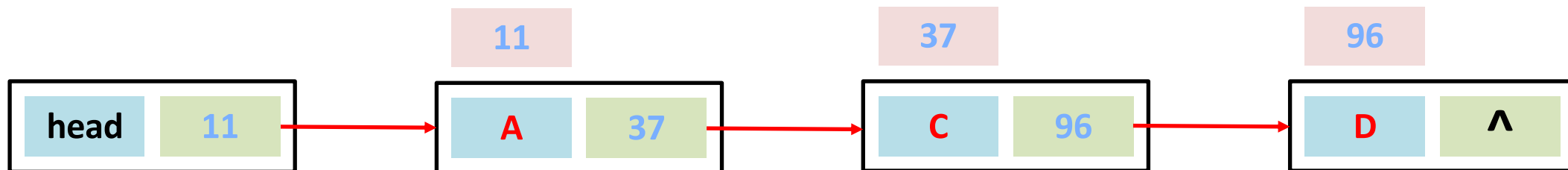
结点的**存储位置**（地址）



链表的特点

- 链表中的元素是游离存储的，每个元素节点包含数据值和下一个元素的地址。
- **链表查询慢。无论查询哪个数据都要从头开始找**

添加一个链表：



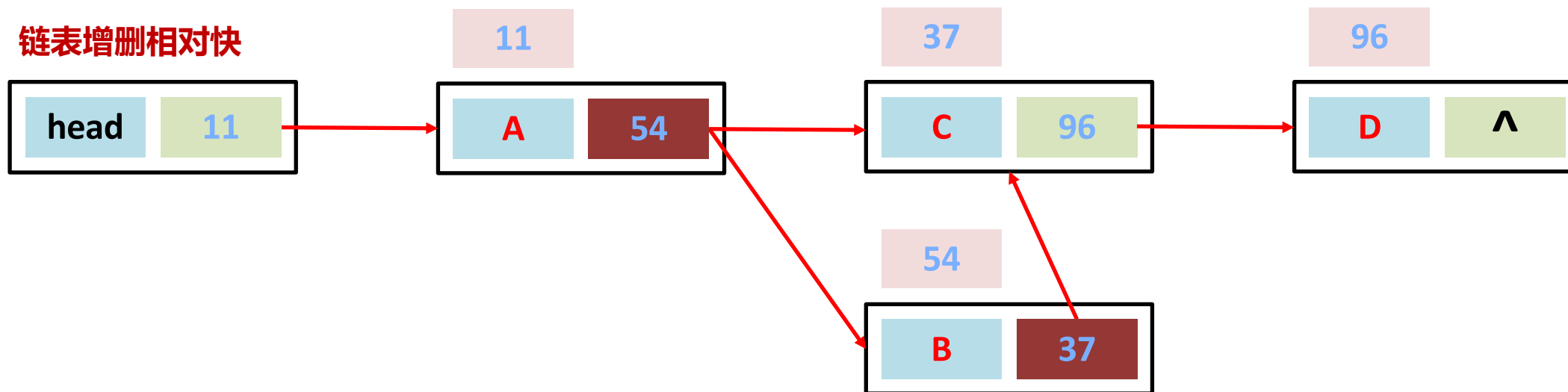
添加一个数据**A**

再添加一个数据**C**

再添加一个数据**D**

链表的特点

- 链表中的元素是游离存储的，每个元素节点包含数据值和下一个元素的地址。
- 链表查询慢。无论查询哪个数据都要从头开始找
- 链表增删相对快

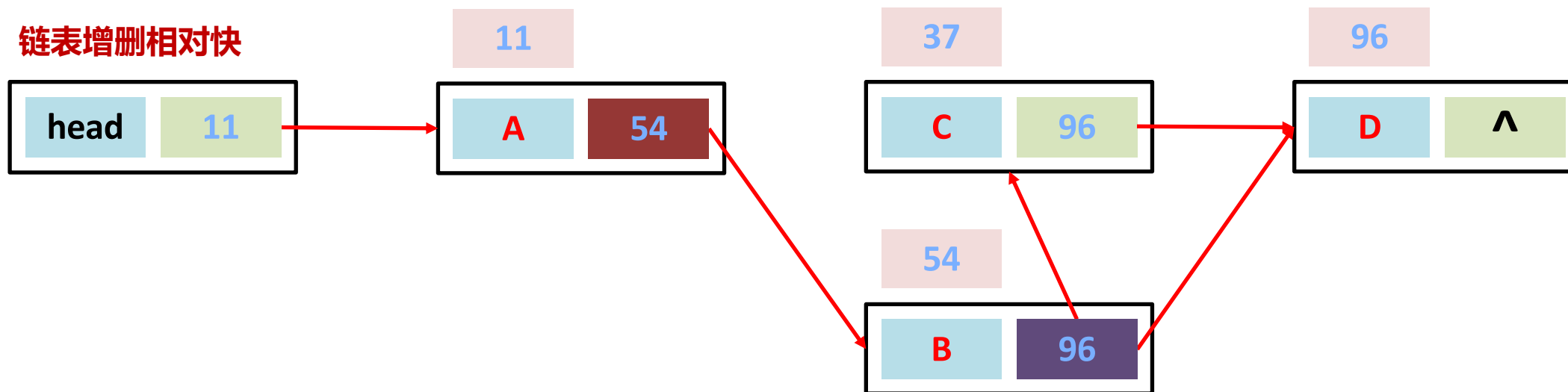


在数据AC之间添加一个数据B

- ① 数据B对应的下一个数据地址指向数据C
- ② 数据A对应的下一个数据地址指向数据B

链表的特点

- 链表中的元素是游离存储的，每个元素节点包含数据值和下一个元素的地址。
- 链表查询慢。无论查询哪个数据都要从头开始找
- 链表增删相对快



删除数据BD之间的数据C

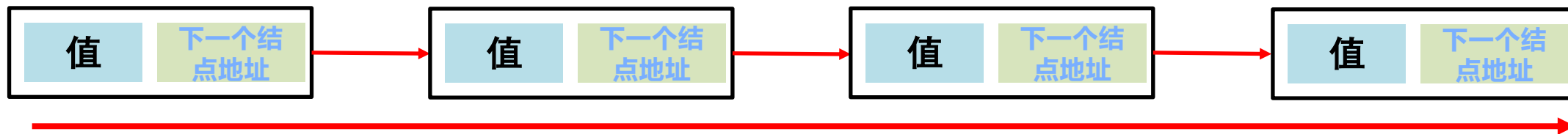
- ① 数据B对应的下一个数据地址指向数据D
- ② 数据C删除

链表是一种**增删快**的模型(对比数组)

链表是一种**查询慢**的模型(对比数组)

链表的种类

单向链表



双向链表





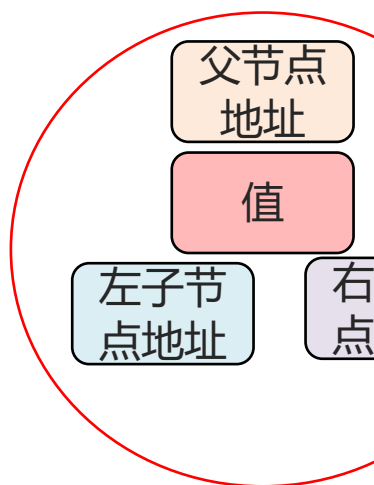
目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ 数据结构概述、栈、队列
 - ◆ 数组
 - ◆ 链表
 - ◆ **二叉树、 二叉查找树**
 - ◆ 平衡二叉树
 - ◆ 红黑树
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

二叉树概述

B节点

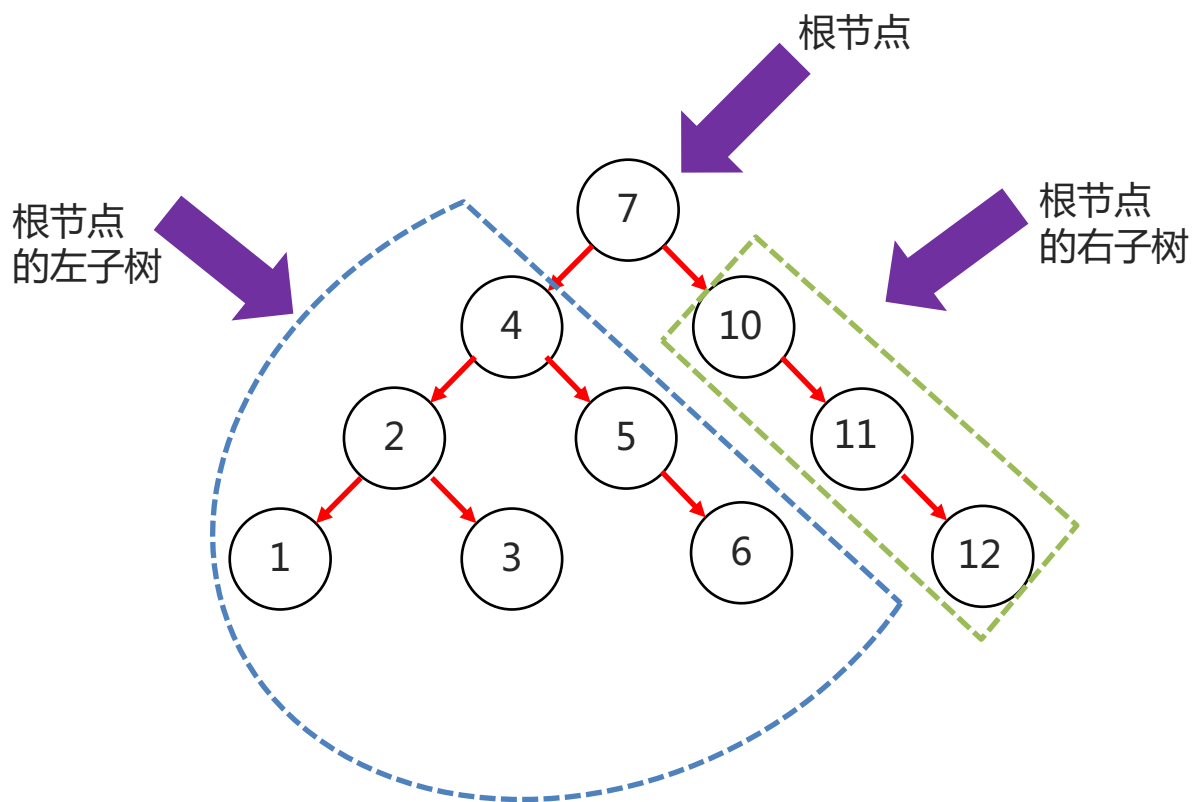


A节点是BC的父节点
BC是A的子结点

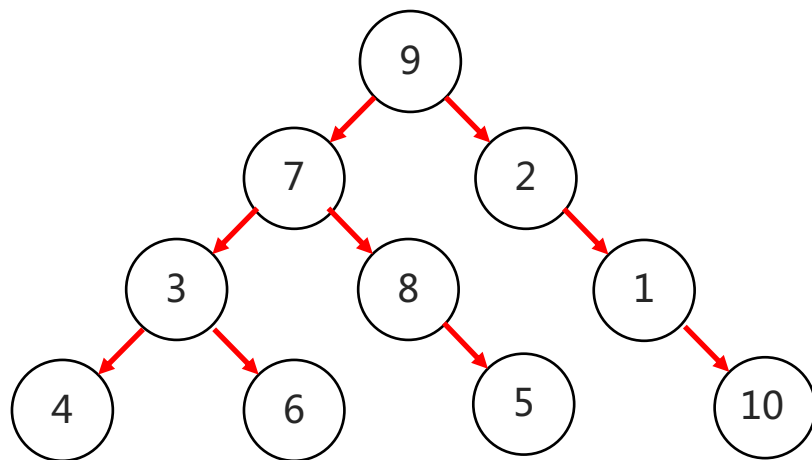
C节点



二叉树的特点



- **只能有一个根节点**，每个节点最多支持2个直接子节点。
- **节点的度**：节点拥有的子树的个数，二叉树的度不大于2
叶子节点 度为0的节点，也称之为终端结点。
- **高度**：叶子结点的高度为1，叶子结点的父节点高度为2，
以此类推，根节点的高度最高。
- **层**：根节点在第一层，以此类推
- **兄弟节点**：拥有共同父节点的节点互称为兄弟节点

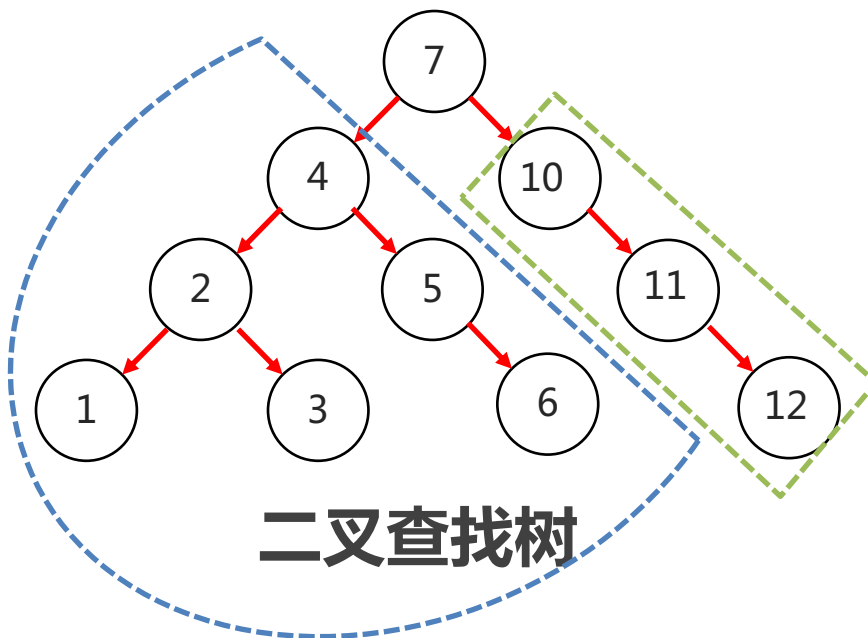


普通的二叉树

二叉查找树又称二叉排序树或者二叉搜索树。

特点：

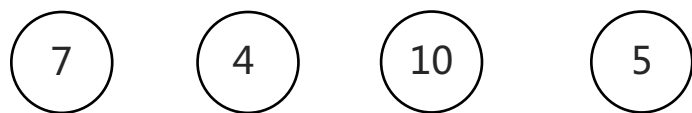
- 1，每一个节点上最多有两个子节点
- 2，左子树上所有节点的值都小于根节点的值
- 3，右子树上所有节点的值都大于根节点的值



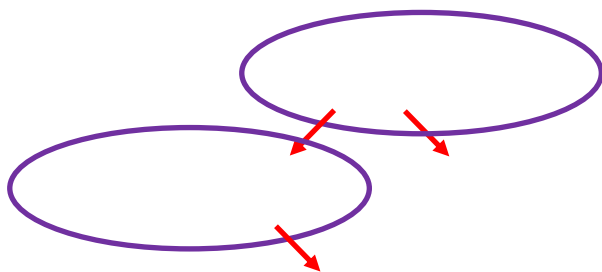
二叉查找树

目的：提高检索数据的性能。

二叉树查找树添节点



将上面的节点按照二叉查找树的规则存入

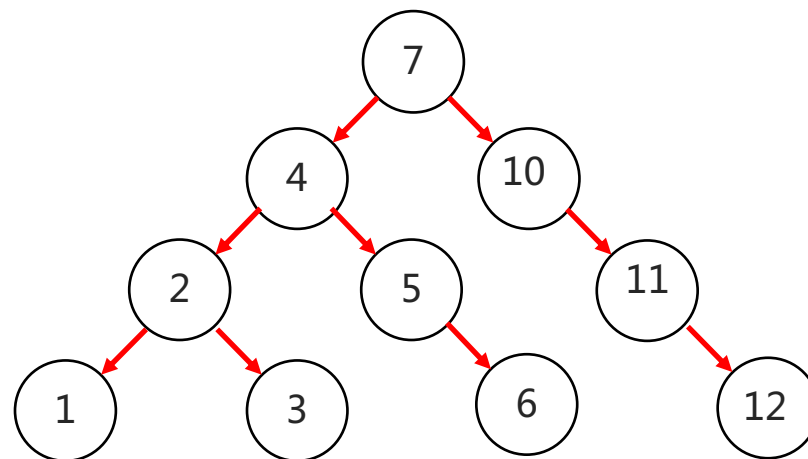


规则：

小的存左边

大的存右边

一样的不存





目录

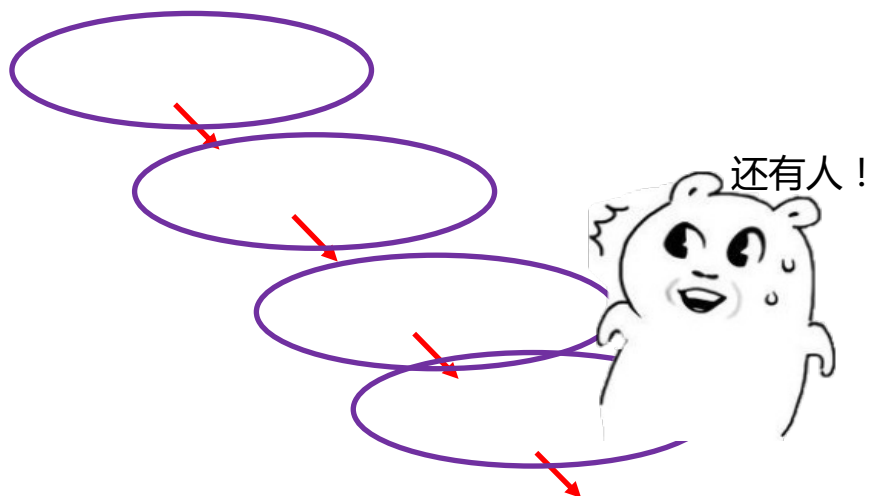
Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ 数据结构概述、栈、队列
 - ◆ 数组
 - ◆ 链表
 - ◆ 二叉树、 二叉查找树
 - ◆ 平衡二叉树
 - ◆ 红黑树
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

二叉树查找存在的问题：



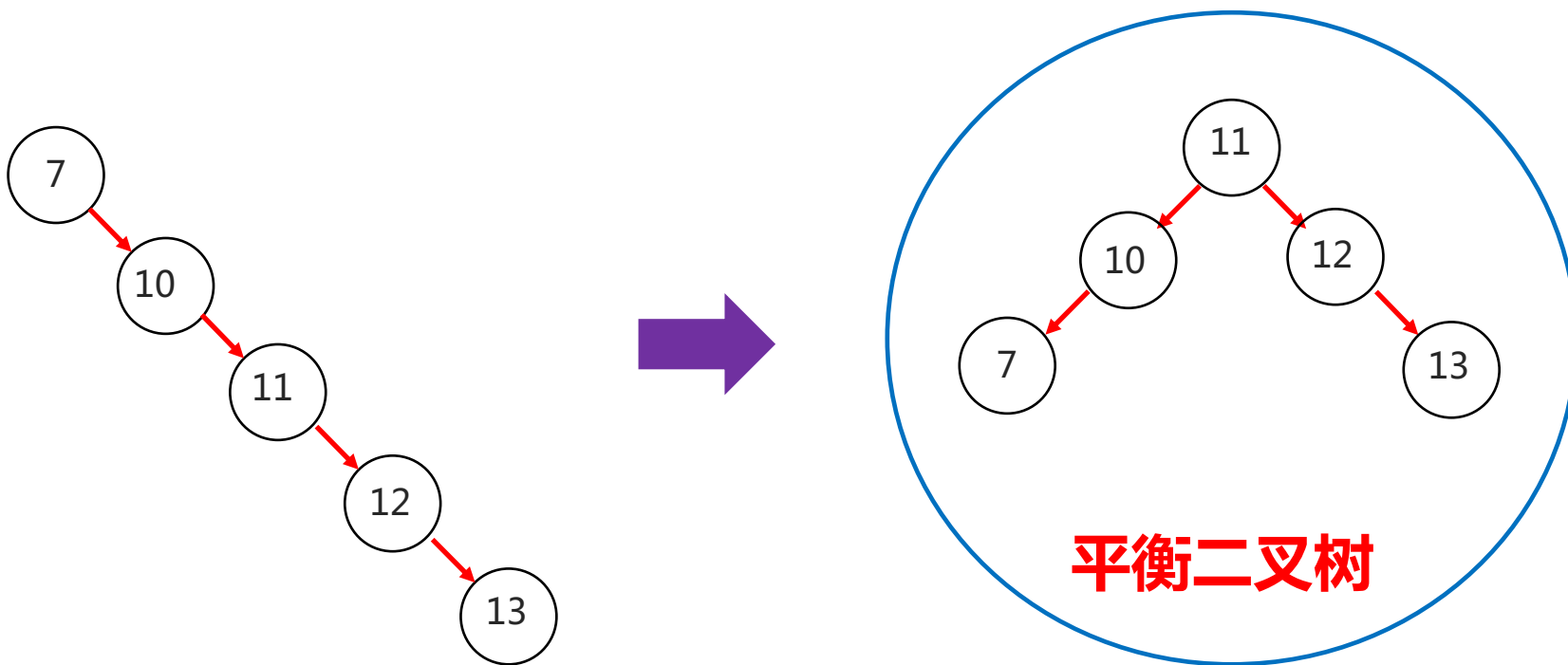
将上面的节点按照二叉查找树的规则存入



问题：出现瘸子现象，导致查询的性能与单链表一样，查询速度变慢！

数据结构-平衡二叉树

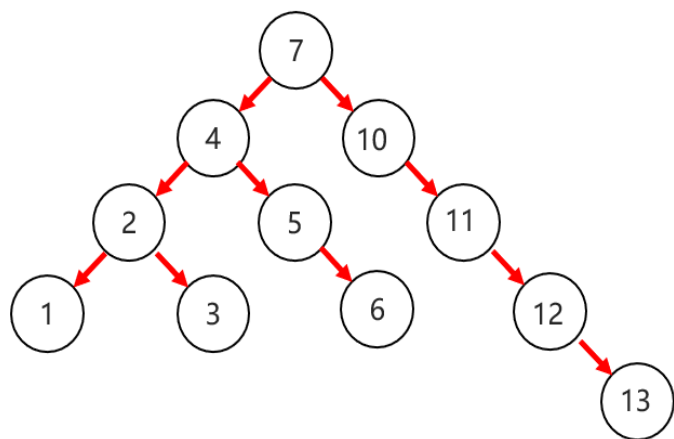
- 平衡二叉树是在满足查找二叉树的大小规则下，让树尽可能矮小，以此提高查数据的性能。



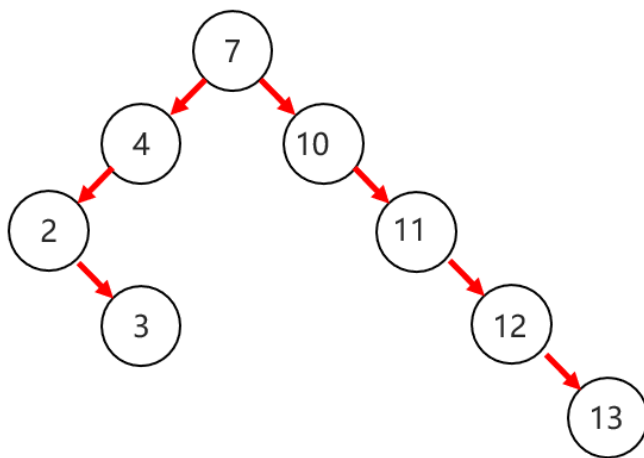
平衡二叉树的要求

- 任意节点的左右两个子树的高度差不超过1，任意节点的左右两个子树都是一颗平衡二叉树

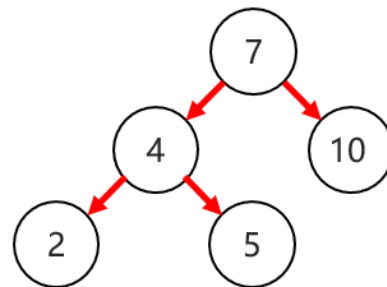
识别如下是否是平衡二叉树



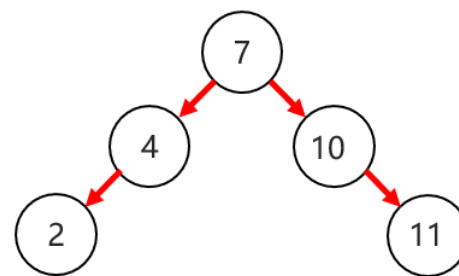
不是的



不是的



是的



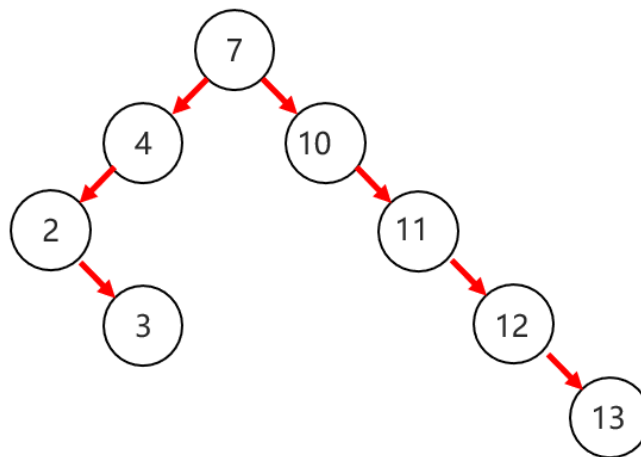
是的

平衡二叉树在添加元素后可能导致不平衡

- 基本策略是进行**左旋**，或者**右旋**保证平衡。

平衡二叉树-旋转的四种情况

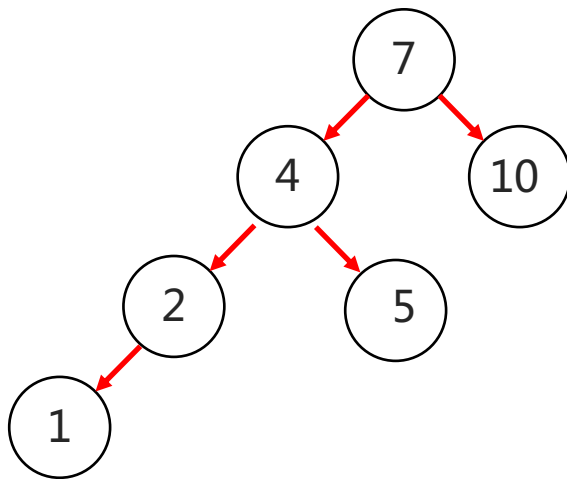
- 左左
- 左右
- 右右
- 右左



平衡二叉树-左左

- 左左

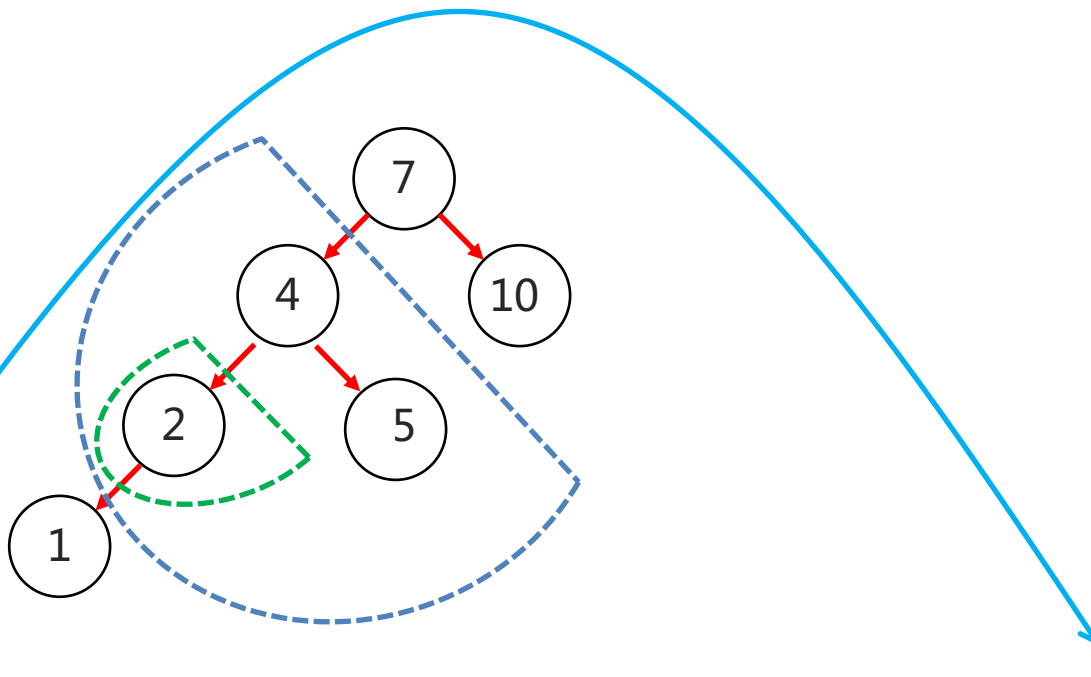
当根节点左子树的左子树有节点插入，导致二叉树不平衡



平衡二叉树-左左

- 左左

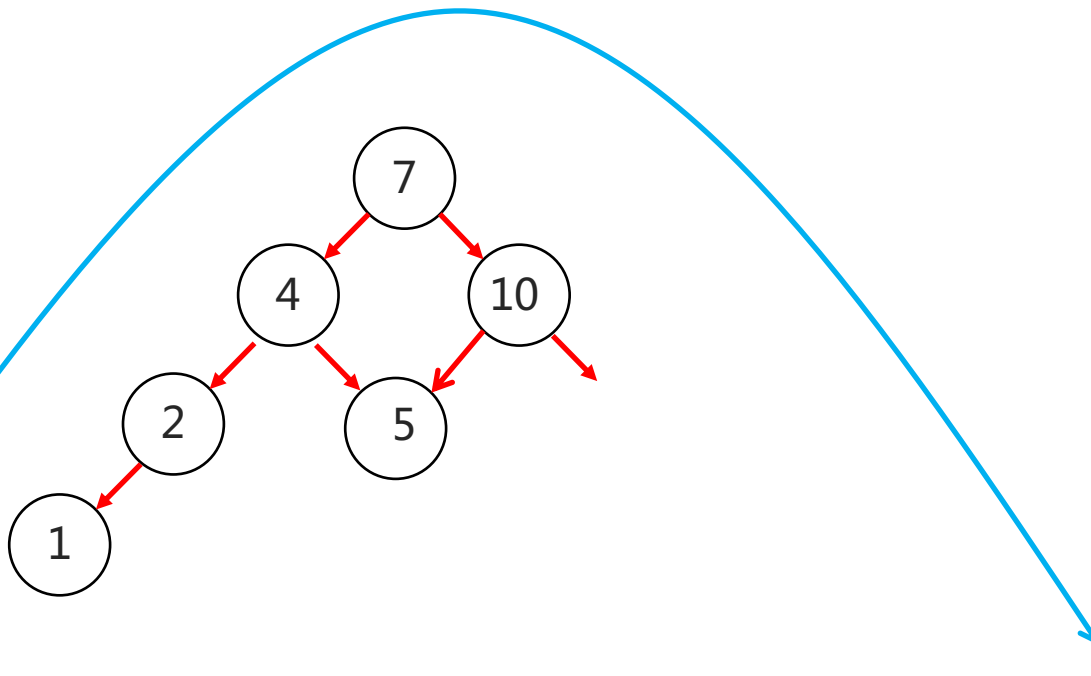
当根节点左子树的左子树有节点插入，导致二叉树不平衡



平衡二叉树-左左

- 左左

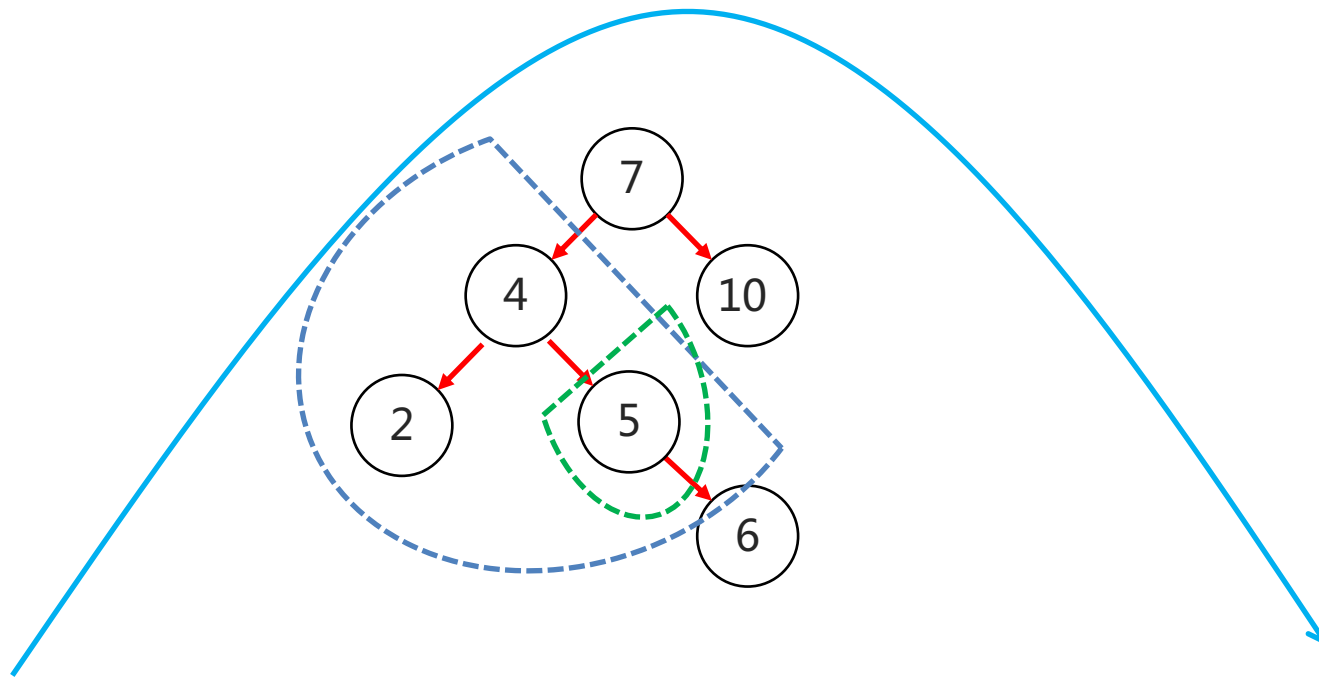
当根节点左子树的左子树有节点插入，导致二叉树不平衡



平衡二叉树-左右

- 左右

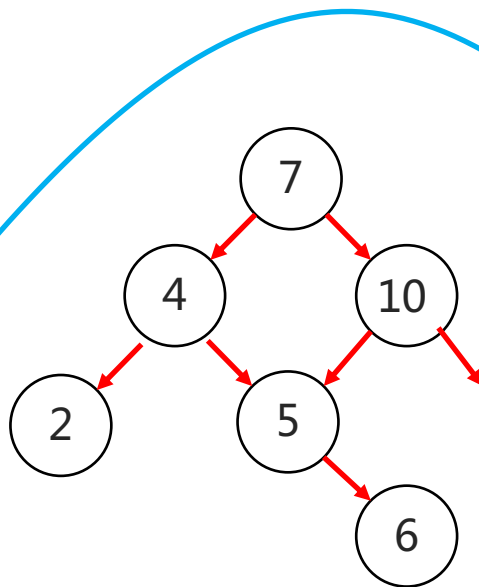
当根节点左子树的右子树有节点插入，导致二叉树不平衡



平衡二叉树-左右

- 左右

当根节点左子树的右子树有节点插入，导致二叉树不平衡

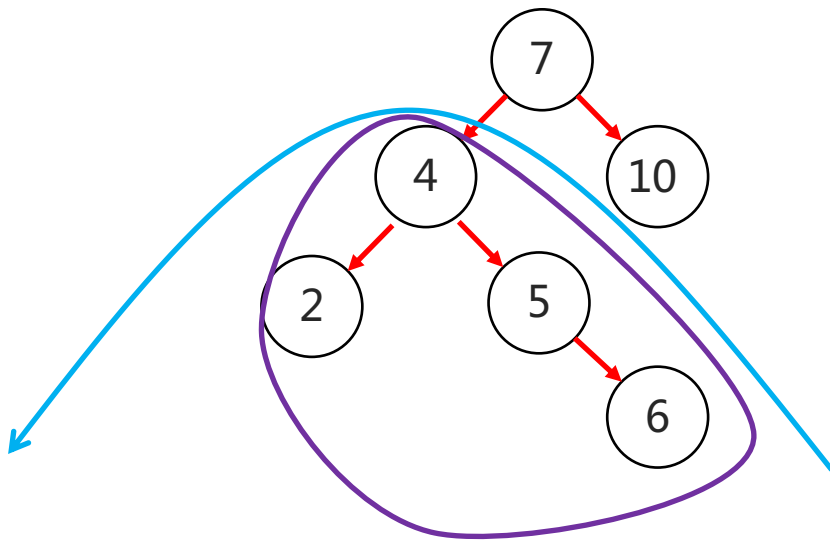


仅仅做一个右旋还是不行

平衡二叉树-左右

- 左右

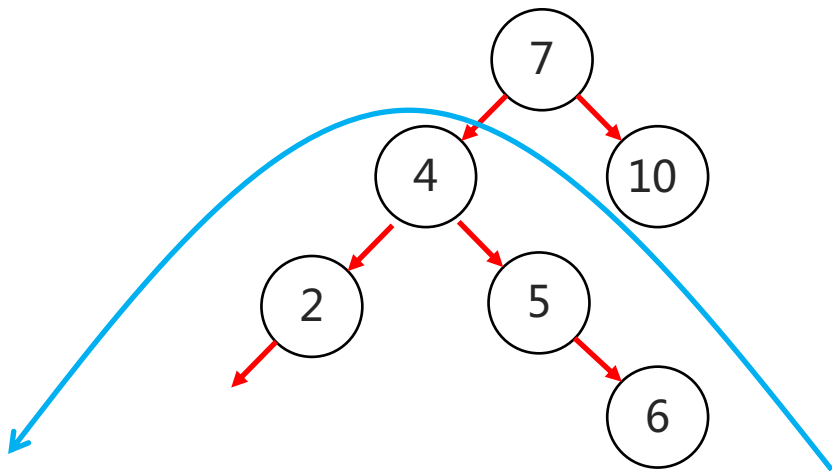
当根节点左子树的右子树有节点插入，导致二叉树不平衡



平衡二叉树-左右

- 左右

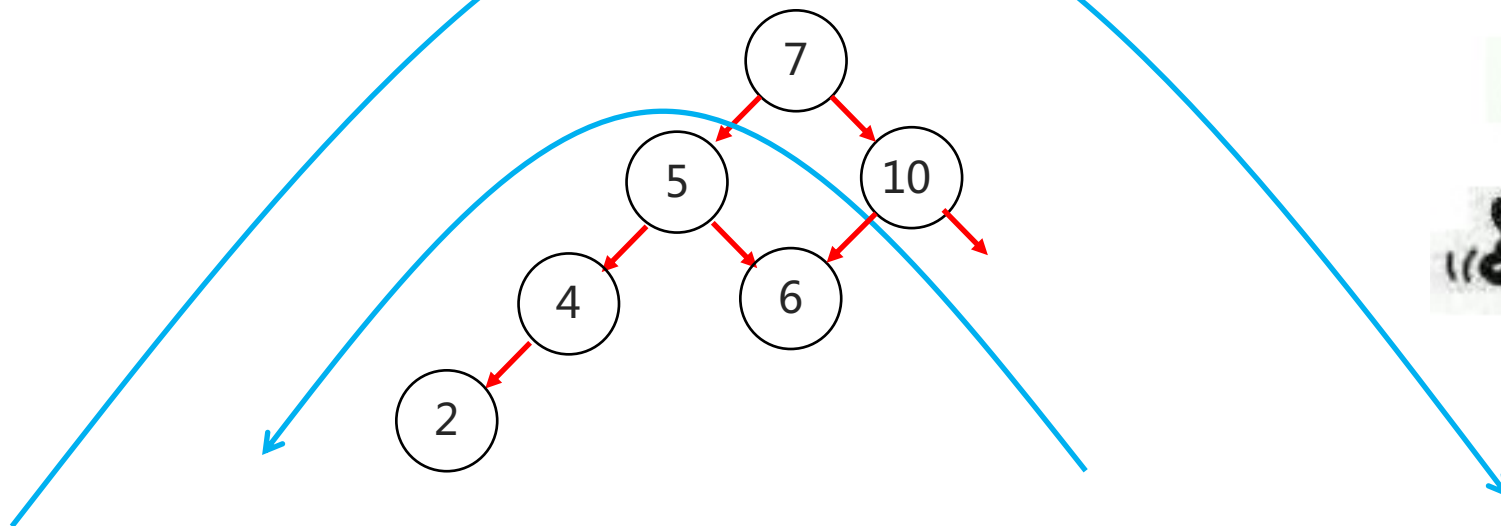
当根节点左子树的右子树有节点插入，导致二叉树不平衡



平衡二叉树-左右

- 左右

当根节点左子树的右子树有节点插入，导致二叉树不平衡



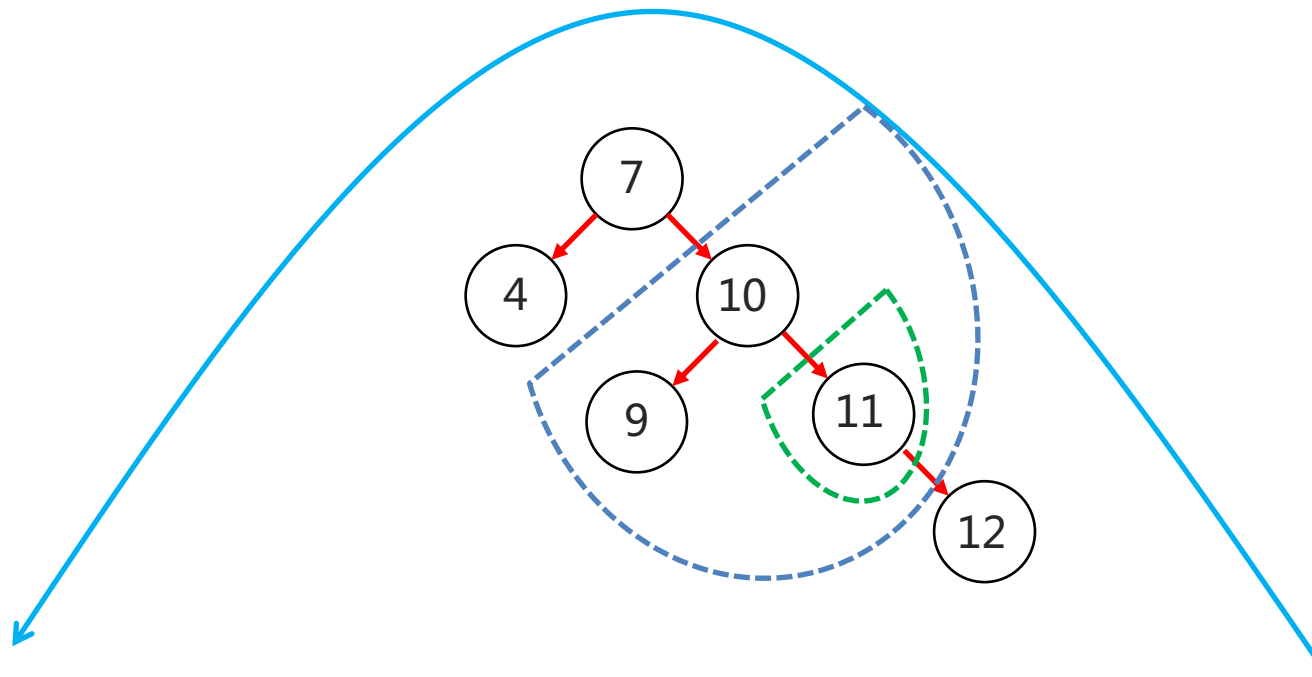
完美!



平衡二叉树-右右

- 右右

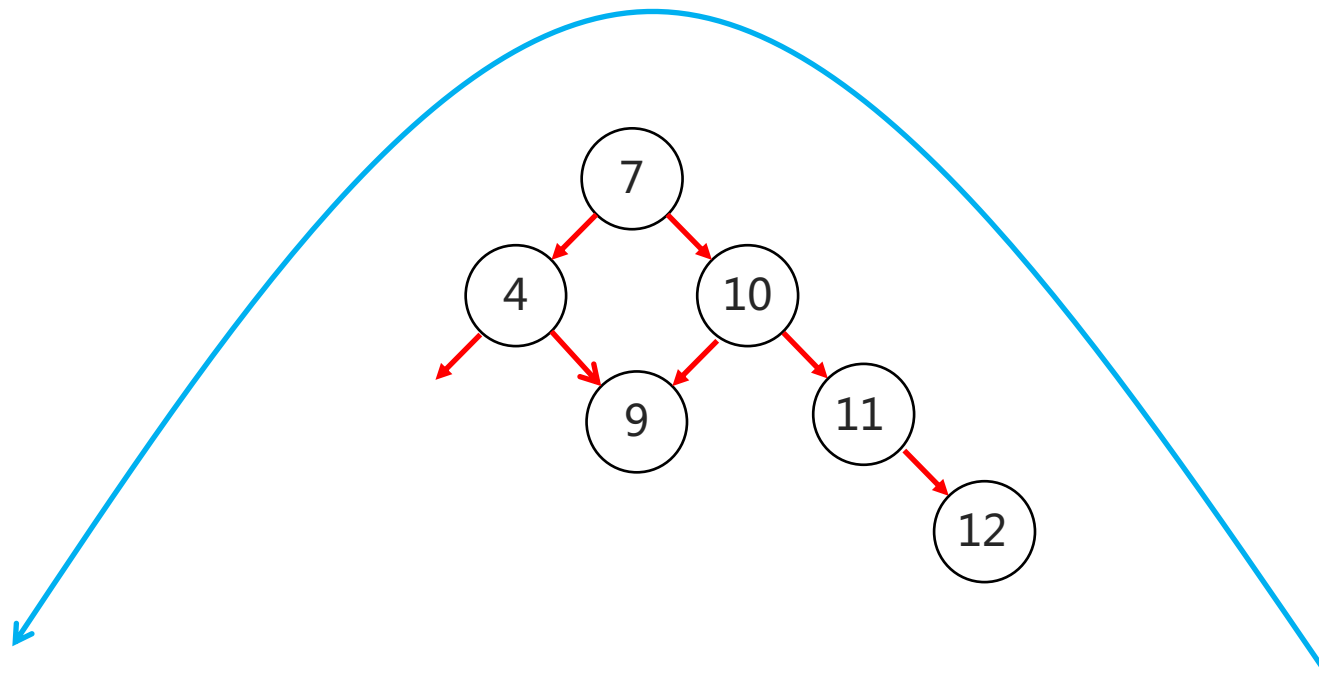
当根节点右子树的右子树有节点插入，导致二叉树不平衡



平衡二叉树-右右

- 右右

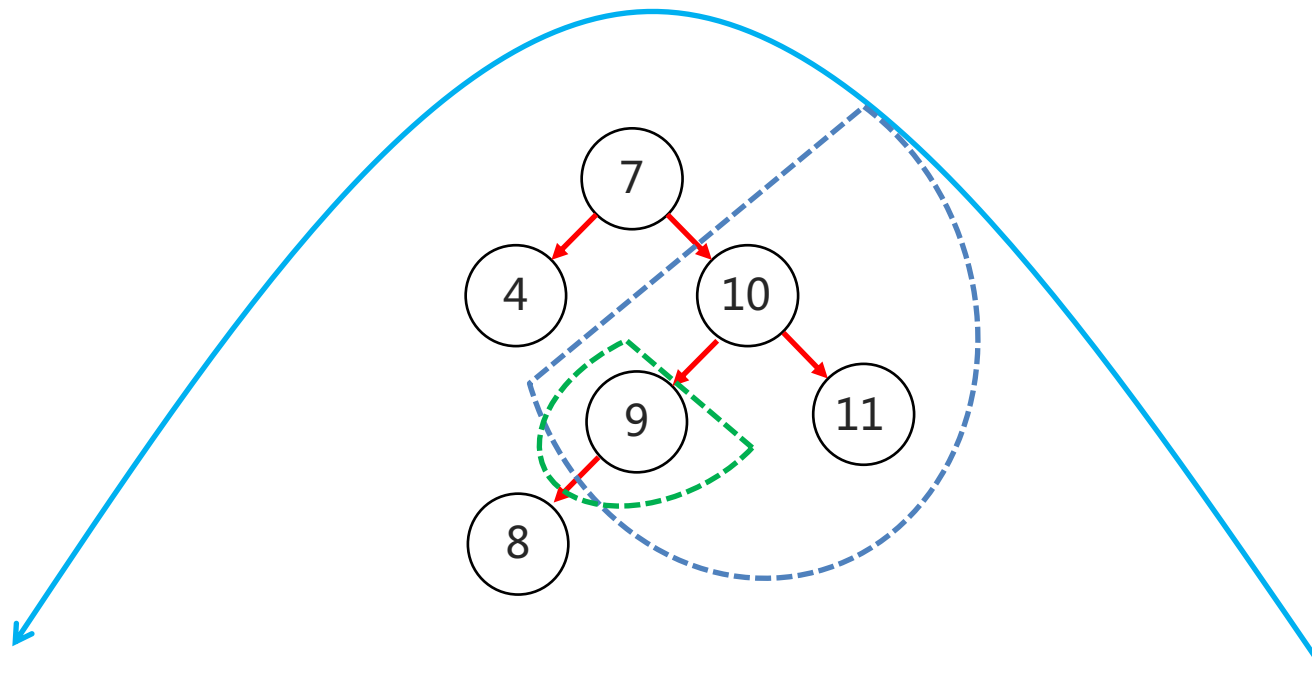
当根节点右子树的右子树有节点插入，导致二叉树不平衡



平衡二叉树-右左

- 右左

当根节点右子树的左子树有节点插入，导致二叉树不平衡

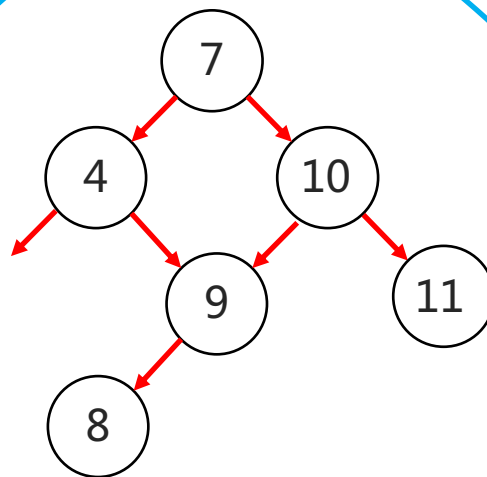


平衡二叉树-右左

- 右左

当根节点右子树的左子树有节点插入，导致二叉树不平衡

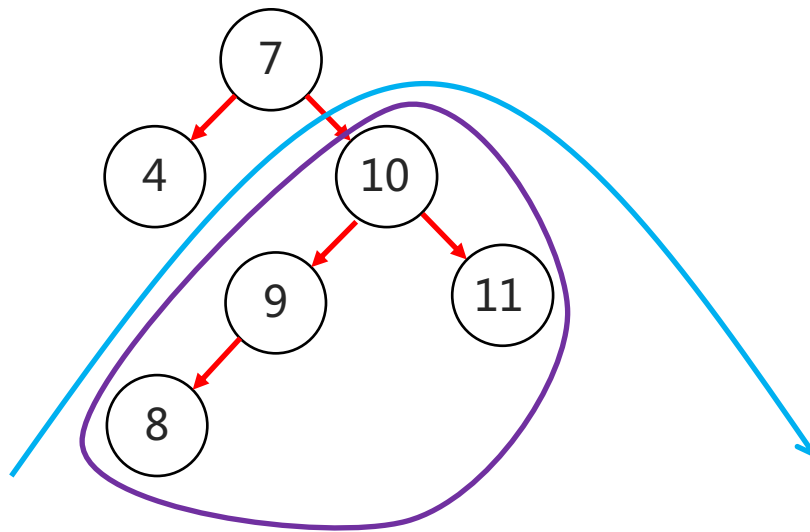
仅仅做一个左旋还是不行



平衡二叉树-右左

- 右左

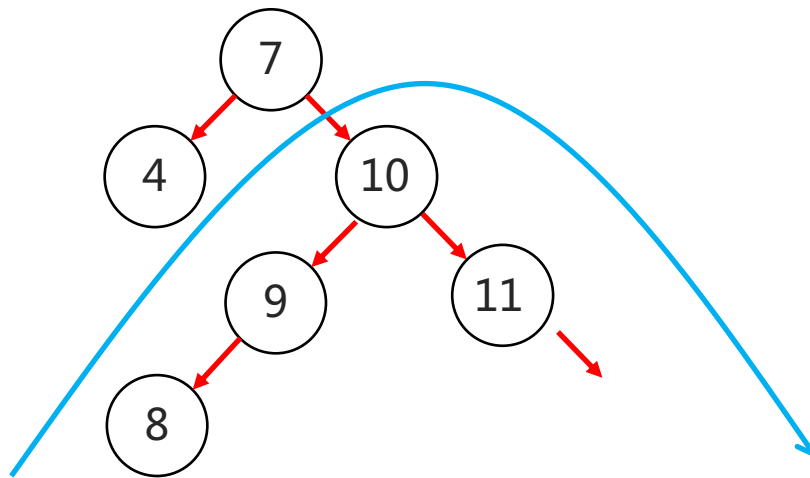
当根节点右子树的左子树有节点插入，导致二叉树不平衡



平衡二叉树-右左

- 右左

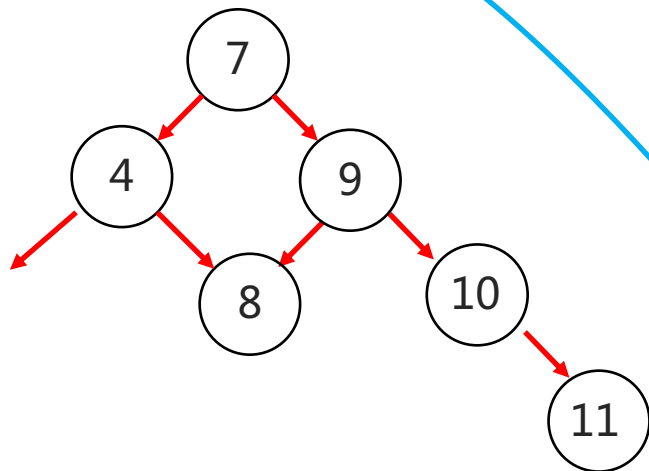
当根节点右子树的左子树有节点插入，导致二叉树不平衡



平衡二叉树-右左

- 右左

当根节点右子树的左子树有节点插入，导致二叉树不平衡



完美!





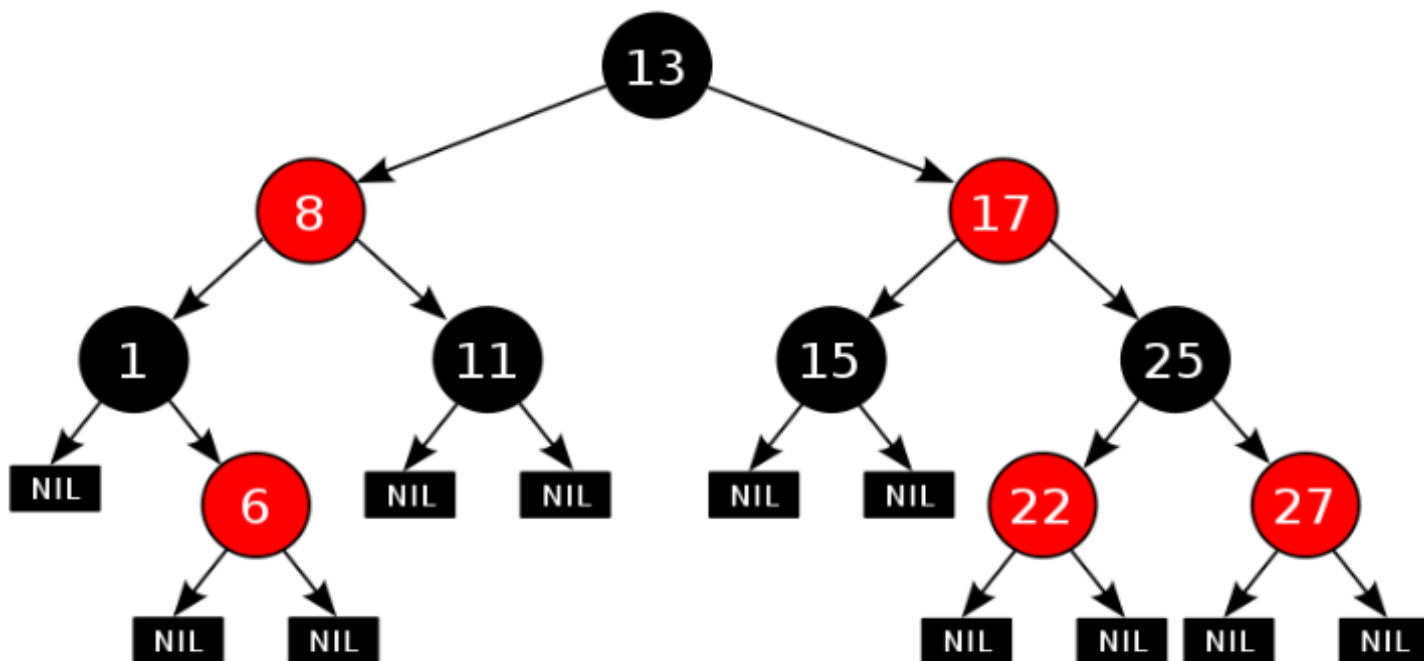
目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
 - ◆ 数据结构概述、栈、队列
 - ◆ 数组
 - ◆ 链表
 - ◆ 二叉树、 二叉查找树
 - ◆ 平衡二叉树
 - ◆ **红黑树**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

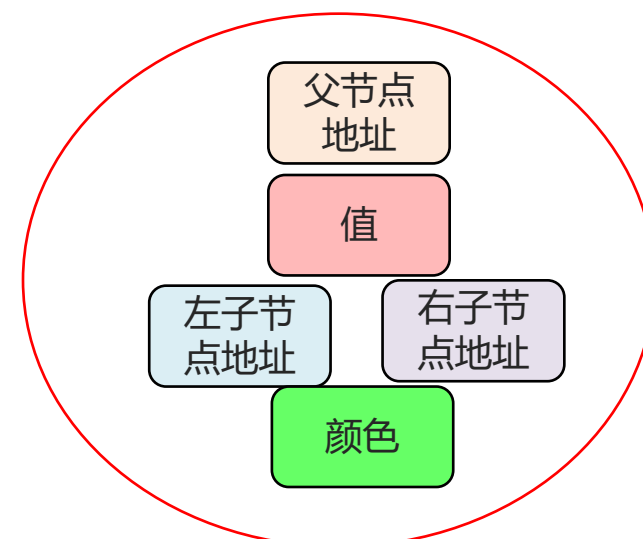
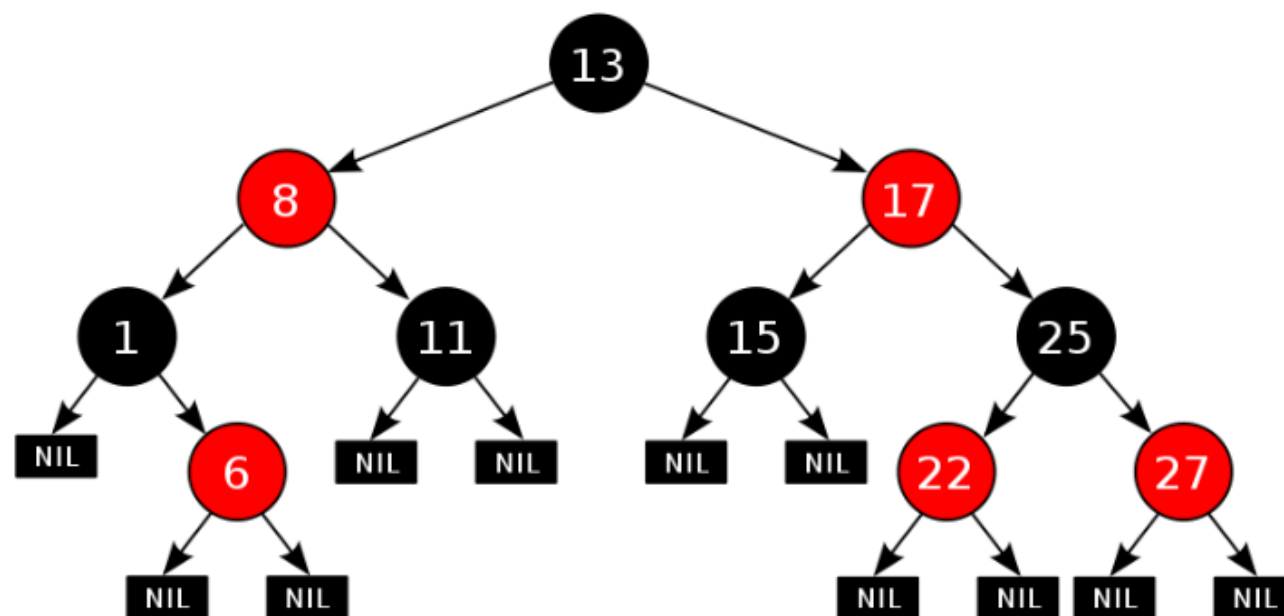
红黑树概述

- 红黑树是一种自平衡的二叉查找树，是计算机科学中用到的一种数据结构。
- 1972年出现，当时被称之为平衡二叉B树。1978年被修改为如今的"红黑树"。
- 每一个节点可以是红或者黑；红黑树不是通过高度平衡的，它的平衡是通过“红黑规则”进行实现的。



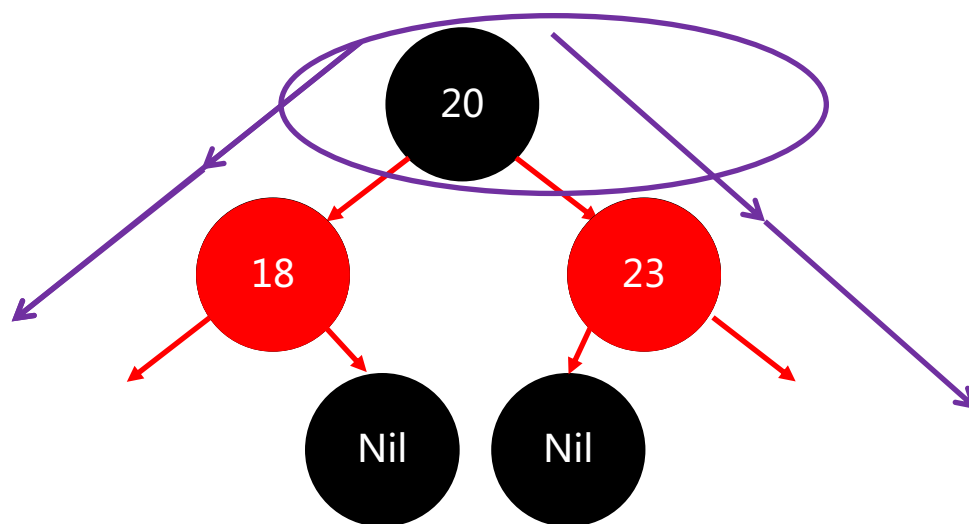
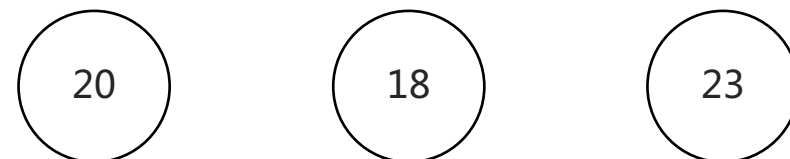
红黑规则

- 每一个节点或是红色的，或者是黑色的，**根节点必须是黑色。**
- 如果某一个节点是红色，那么它的子节点必须是黑色(**不能出现两个红色节点相连的情况**)。
- **对每一个节点，从该节点到其所有后代叶节点的简单路径上，均包含相同数目的黑色节点。**



添加节点

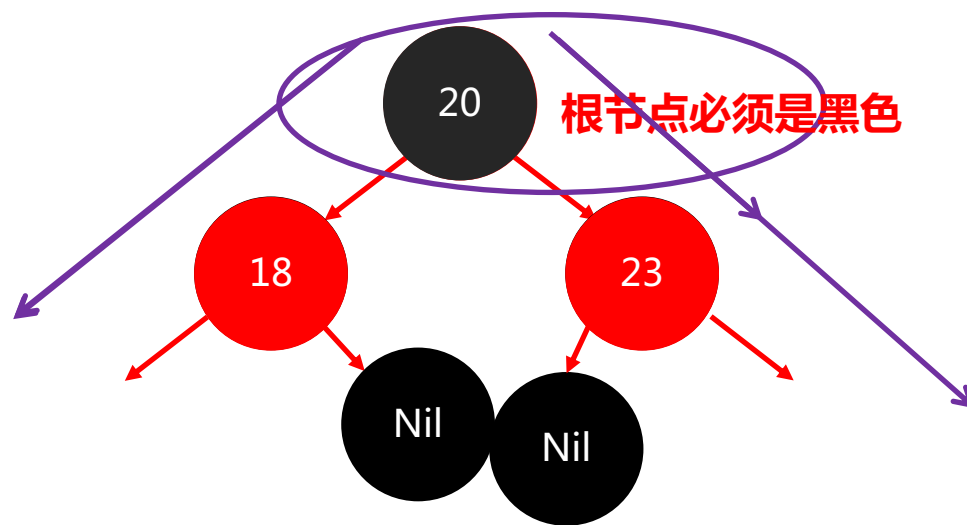
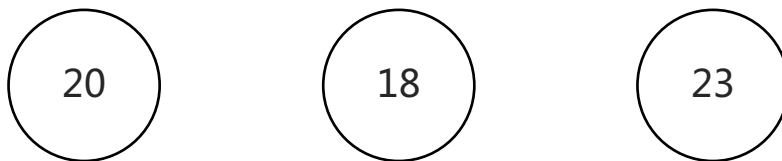
- 添加的节点的颜色，可以是红色的，也可以是黑色的。
- 默认用红色效率高。



添加三个元素，
一共需要调整两次

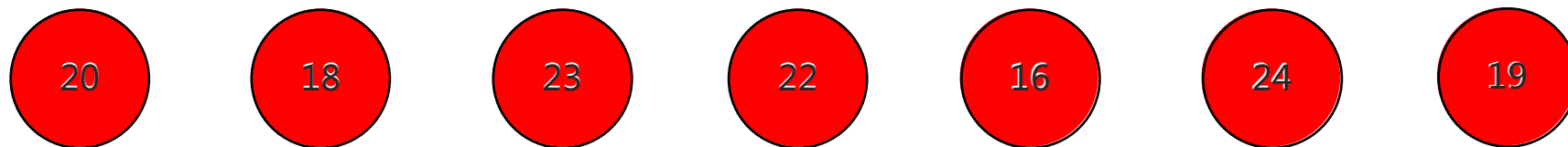
添加节点

- 添加的节点的颜色，可以是红色的，也可以是黑色的。
- 默认用红色效率高。

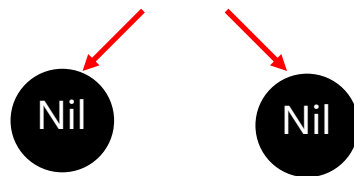


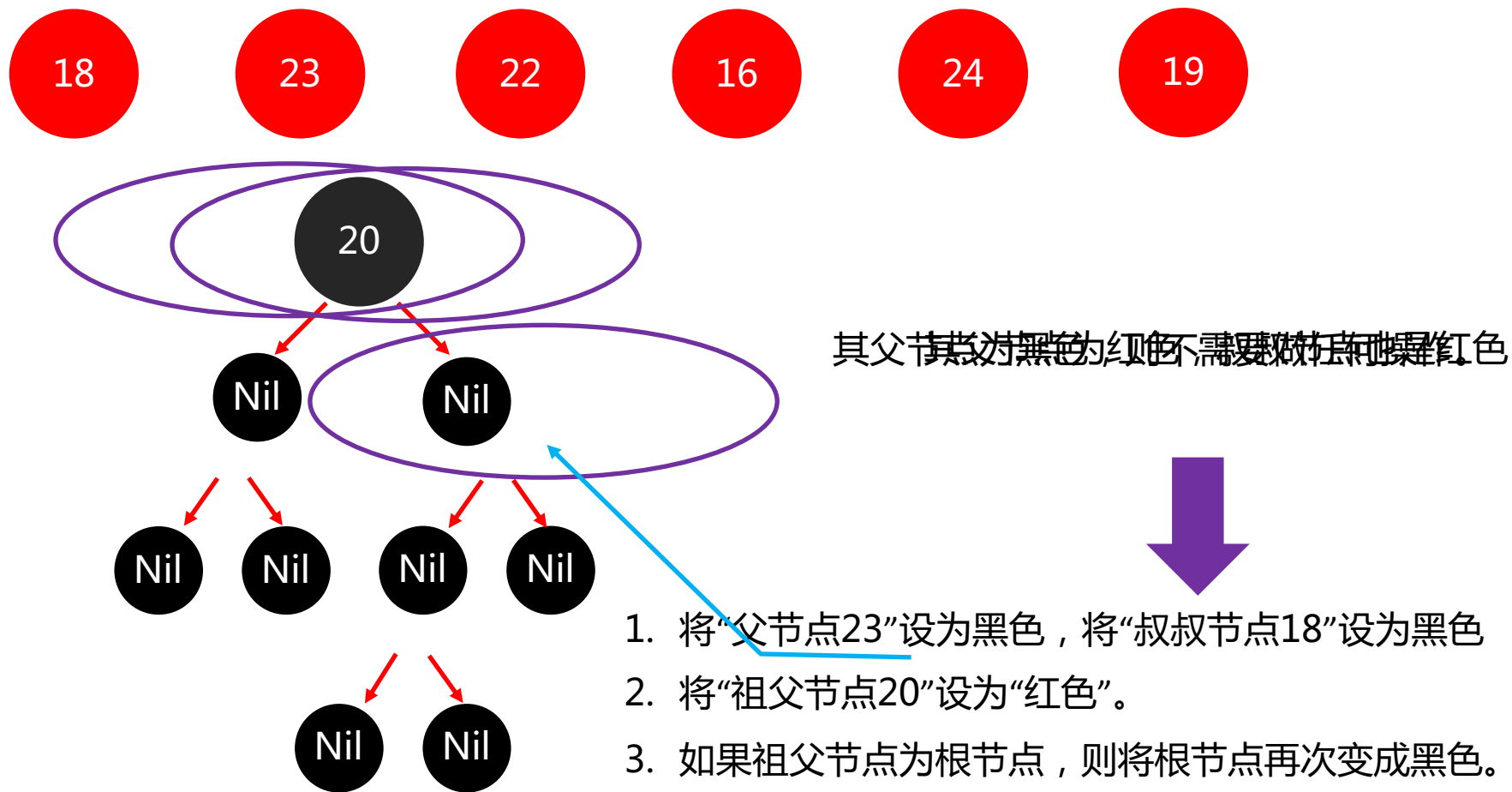
添加三个元素，
一共需要调整一次

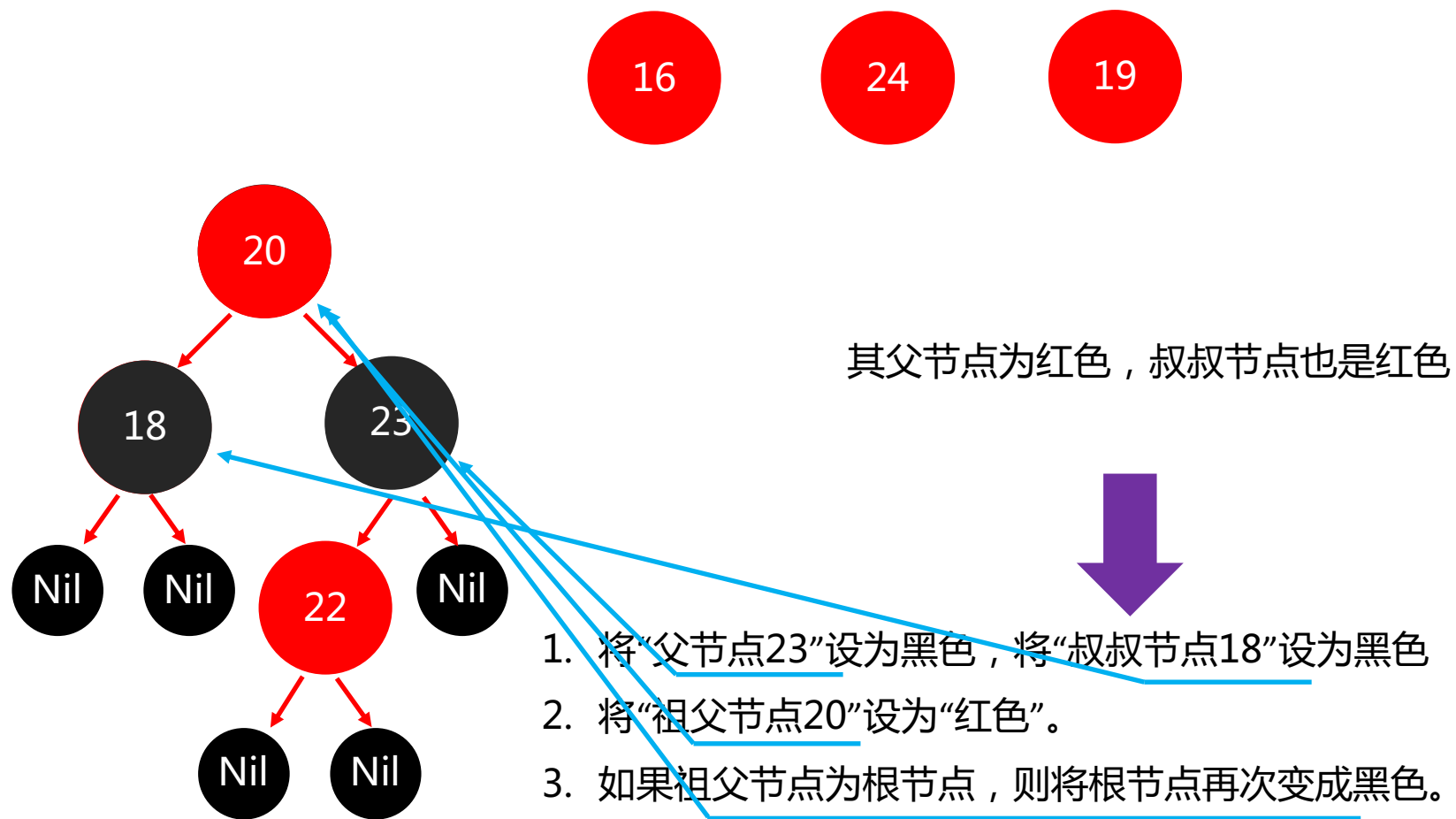
所以，添加节点时，
默认为红色，效率高。

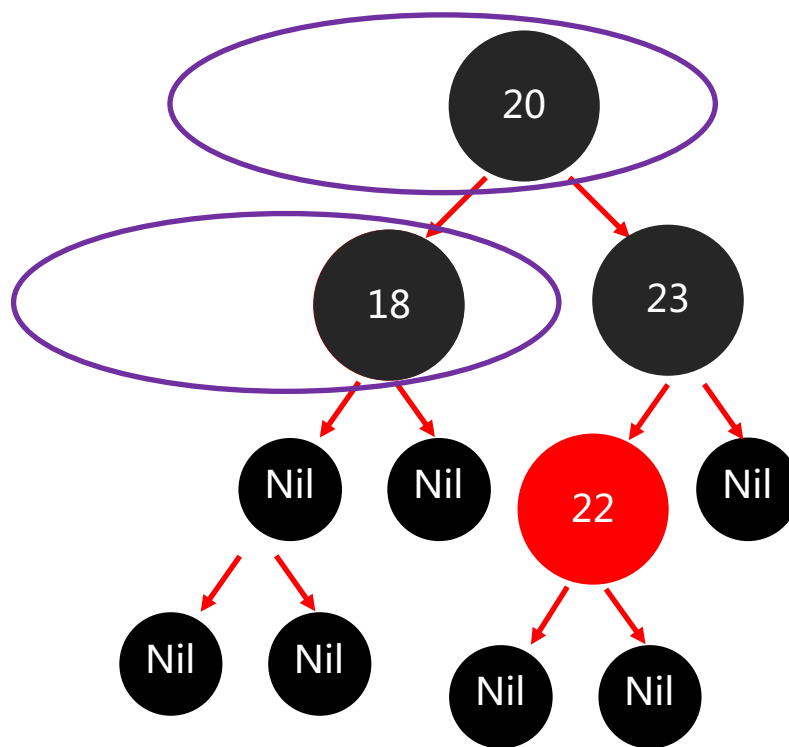


当添加的节点为根节点时，直接变成黑色就可以了



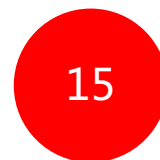
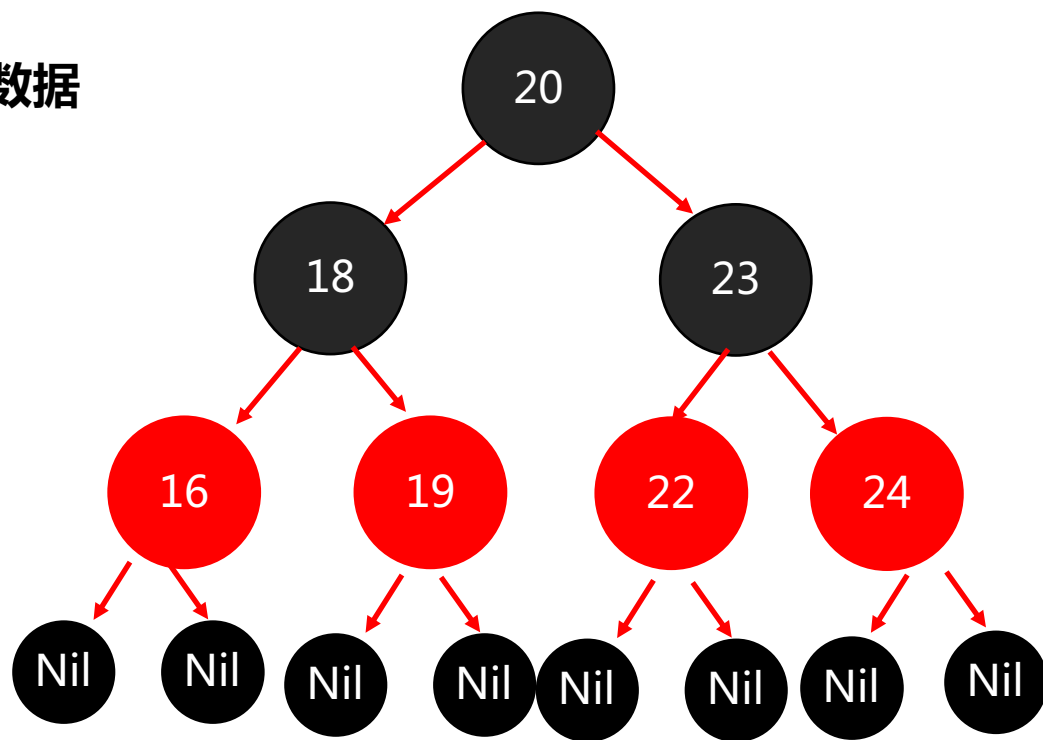


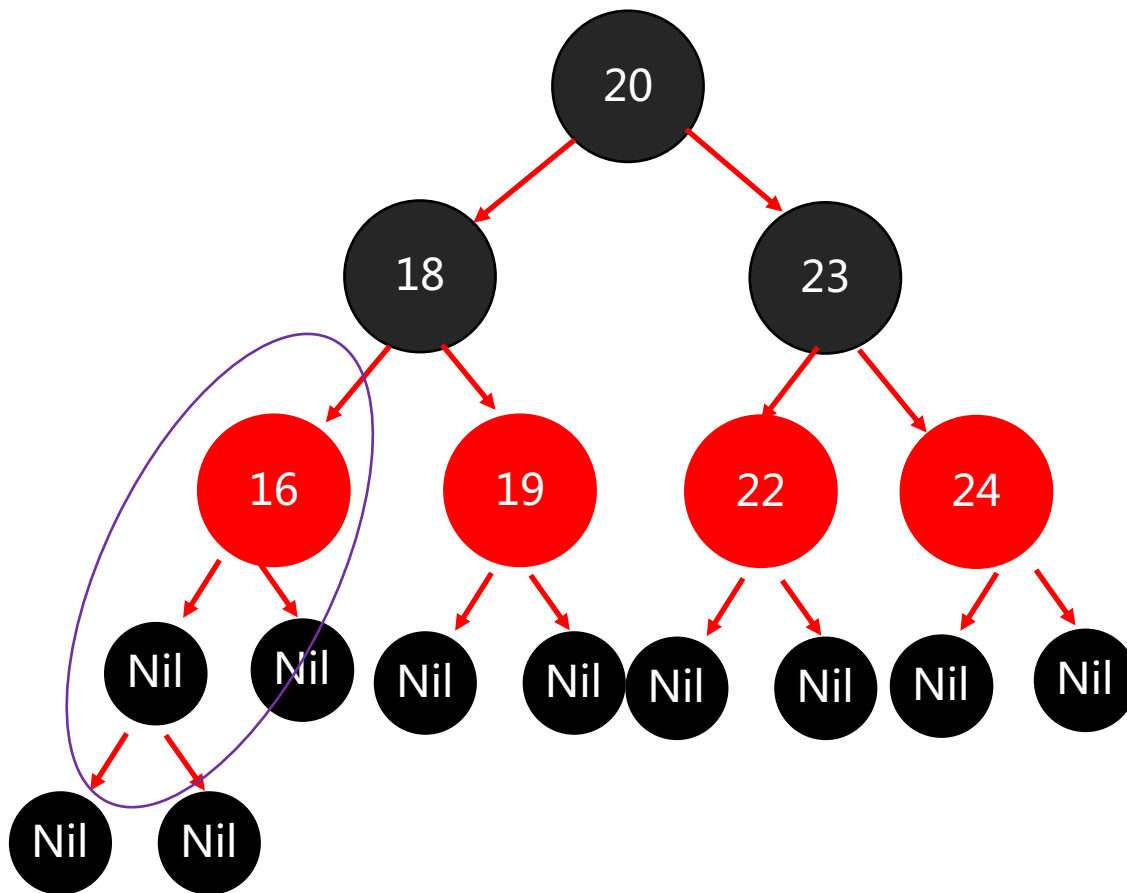




其父节点为黑色，则不需要做任何操作。

继续添加数据

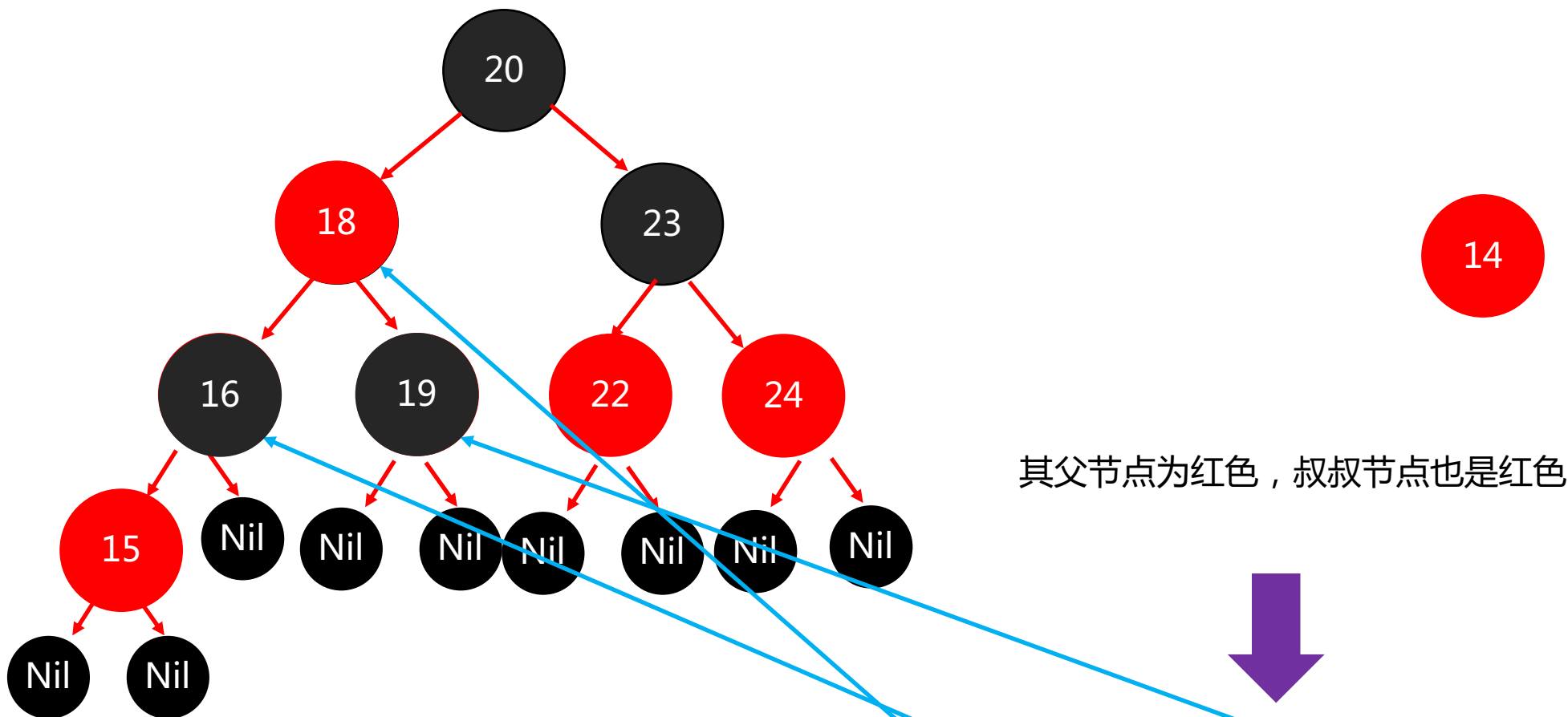




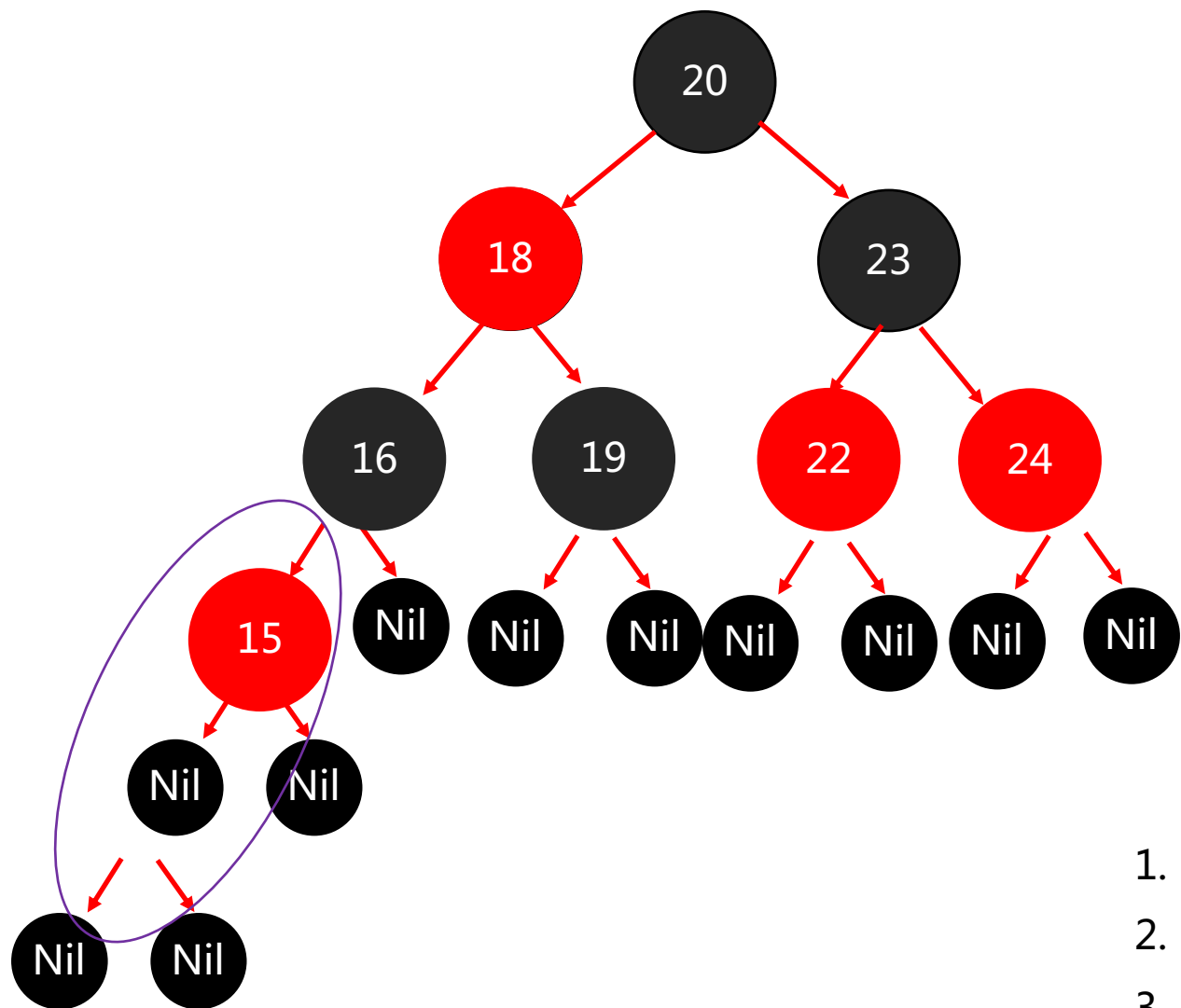
其父节点为红色，叔叔节点也是红色



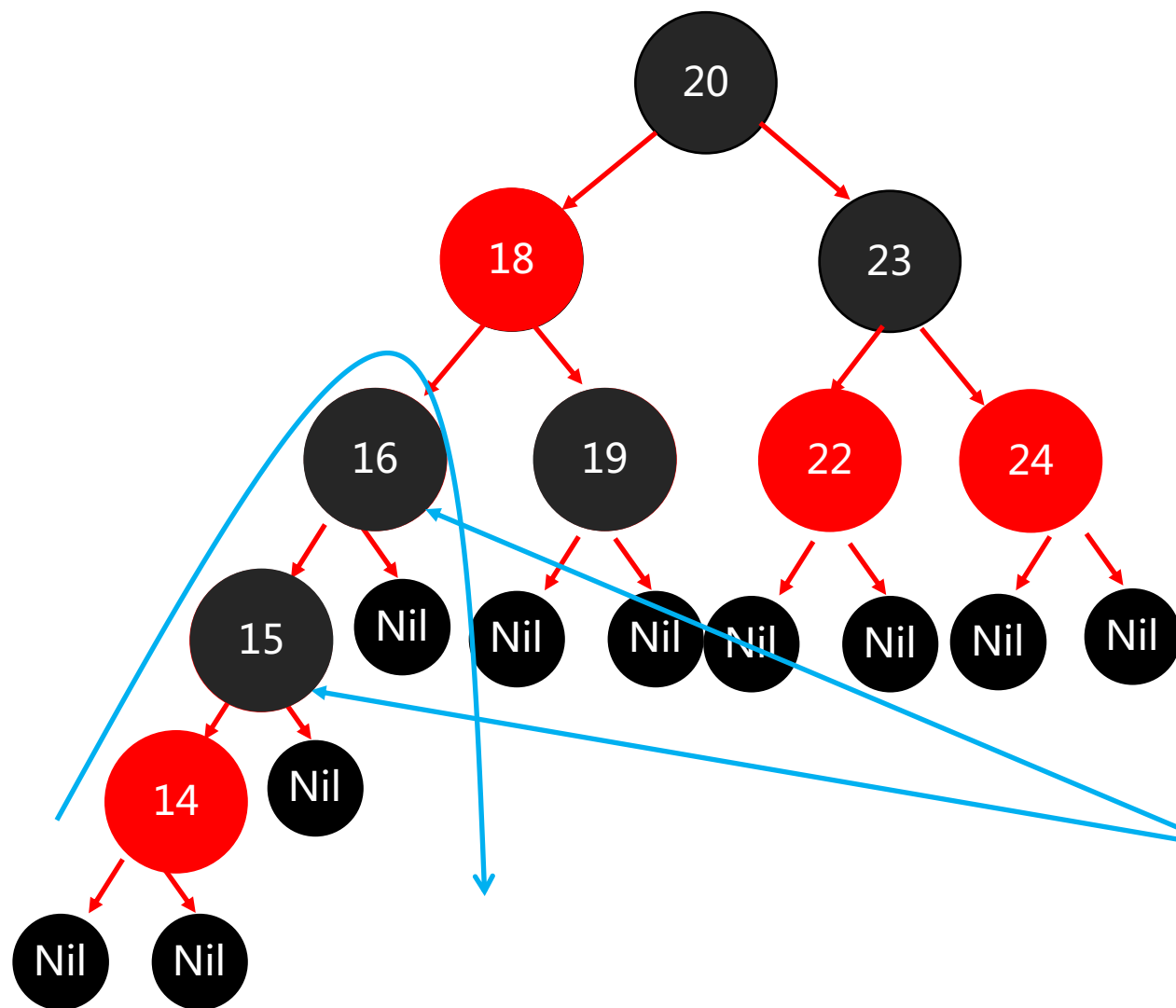
1. 将“父节点16”设为黑色，将“叔叔节点19”设为黑色
2. 将“祖父节点18”设为“红色”。
3. 如果祖父节点为根节点，则将根节点再次变成黑色。



1. 将“父节点16”设为黑色，将“叔叔节点19”设为黑色
2. 将“祖父节点18”设为“红色”。
3. 如果祖父节点为根节点，则将根节点再次变成黑色。

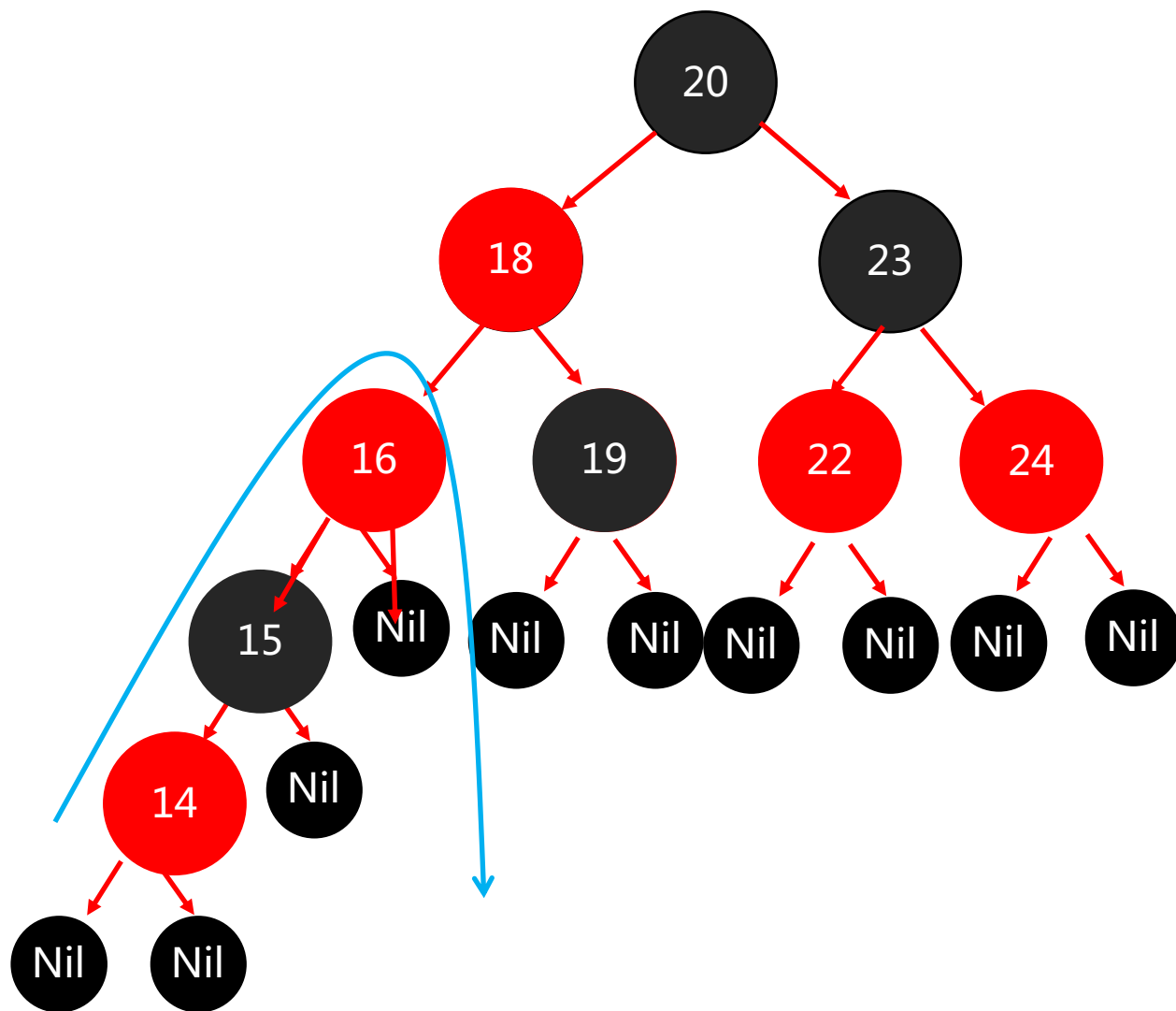


1. 将“父节点16”设为黑色，将“叔叔节点19”设为黑色
2. 将“祖父节点18”设为“红色”。
3. 如果祖父节点为根节点，则将根节点再次变成黑色。



其父节点为红色，叔叔节点也是黑色

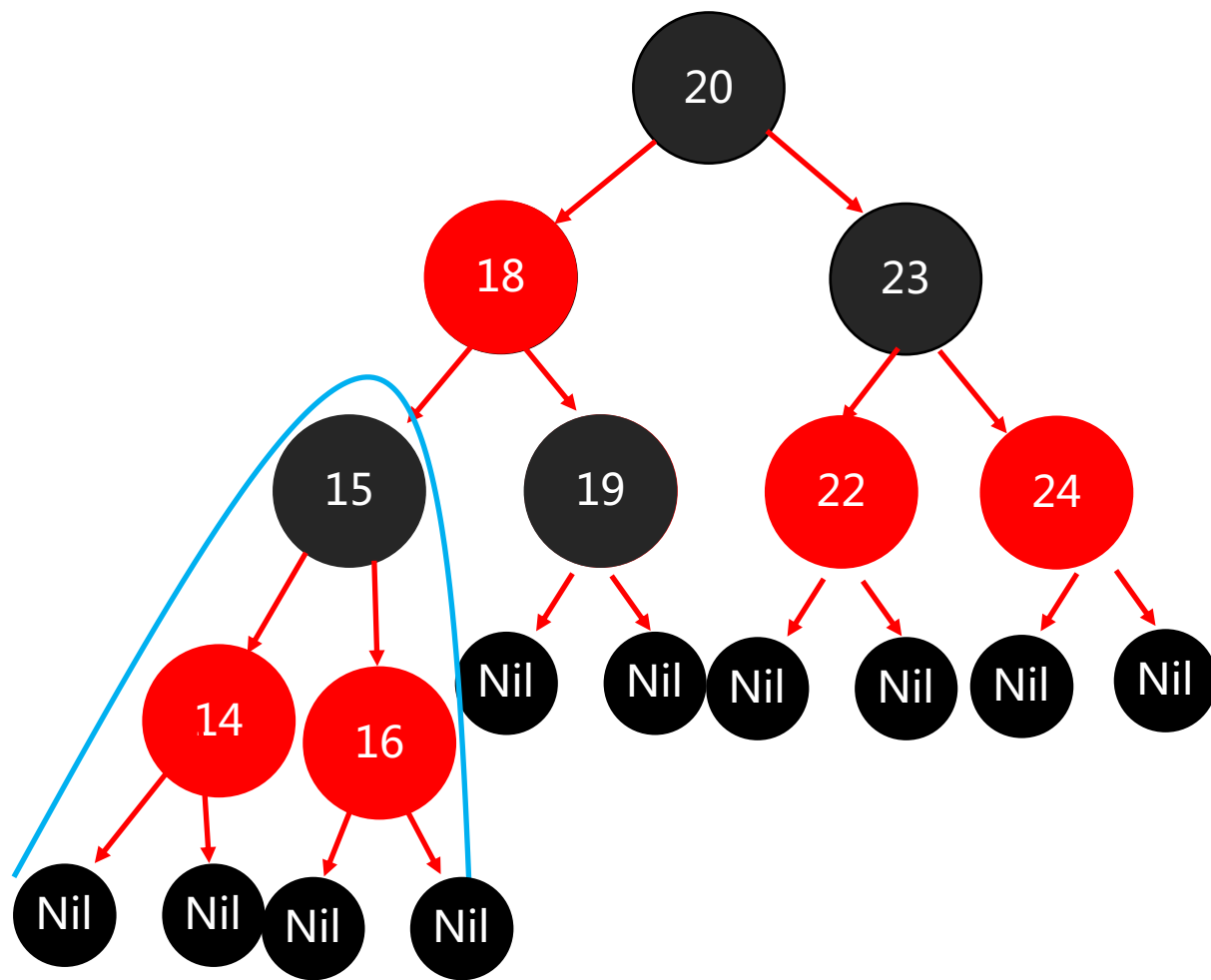
1. 将“父节点15”设为“黑色”
2. 将“祖父节点16”设为“红色”。
3. 以祖父节点为支点进行旋转



其父节点为红色，叔叔节点也是黑色



1. 将“父节点15”设为“黑色”
2. 将“祖父节点16”设为“红色”。
3. 以祖父节点为支点进行旋转



其父节点为红色，叔叔节点也是黑色



1. 将“父节点15”设为“黑色”
2. 将“祖父节点16”设为“红色”。
3. 以祖父节点为支点进行旋转

红黑树小结

- 红黑树**不是高度平衡的**，它的平衡是通过"红黑规则"进行实现的

规则如下：

- 每一个节点或是红色的，或者是黑色的，**根节点必须是黑色**
- 如果一个节点没有子节点或者父节点，则该节点相应的指针属性值为Nil，这些Nil视为叶节点，每个叶节点(Nil)是黑色的；
- 如果某一个节点是红色，那么它的子节点必须是黑色(**不能出现两个红色节点相连的情况**)
- **对每一个节点，从该节点到其所有后代叶节点的简单路径上，均包含相同数目的黑色节点。**

红黑树增删改查的性能都很好



总结

各种数据结构的特点和作用是什么样的

- 队列：先进先出，后进后出。
- 栈：后进先出，先进后出。
- 数组：内存连续区域，查询快，增删慢。
- 链表：元素是游离的，查询慢，首尾操作极快。
- 二叉树：永远只有一个根节点, 每个结点不超过2个子节点的树。
- 查找二叉树：小的左边，大的右边，但是可能树很高，查询性能变差。
- 平衡查找二叉树：让树的高度差不大于1，增删改查都提高了。
- 红黑树（就是基于红黑规则实现了自平衡的排序二叉树）

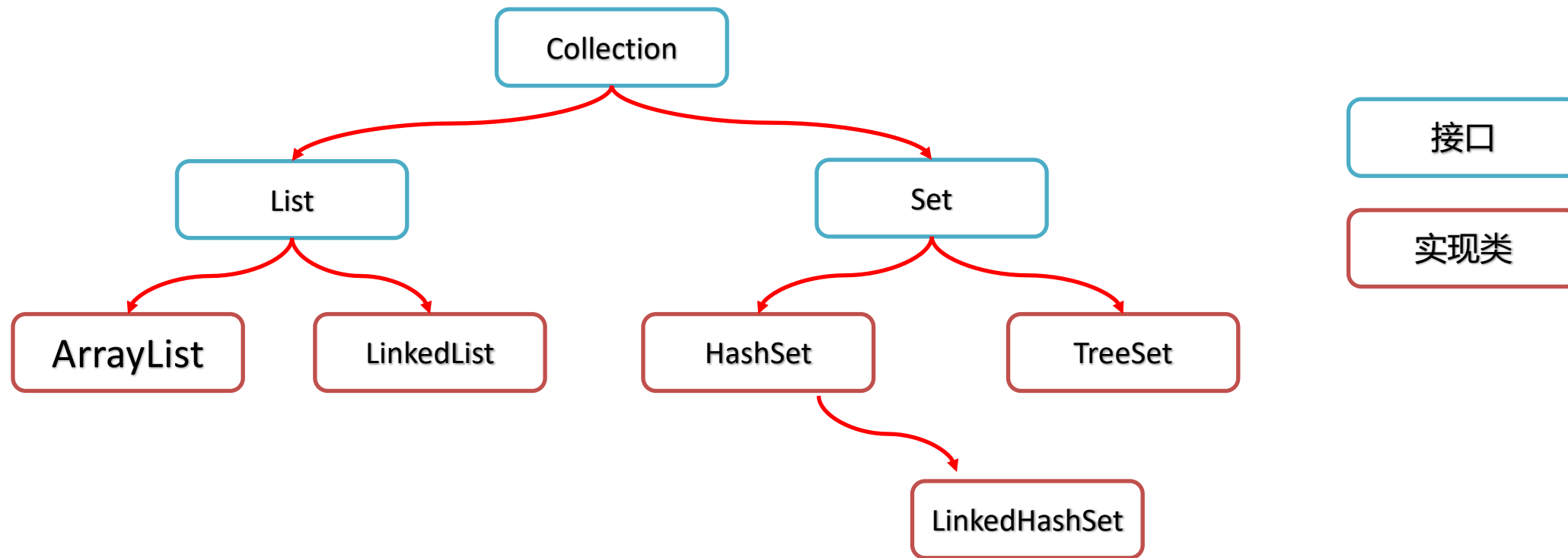


目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
 - ◆ **List集合特点、特有API**
 - ◆ List集合的遍历方式小结
 - ◆ ArrayList集合的底层原理
 - ◆ LinkedList集合的底层原理
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**

Collection集合体系



List系列集合特点

- ArrayList、LinkedList：有序，可重复，有索引。
- 有序：存储和取出的元素顺序一致
- 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集合特有方法

- List集合因为支持索引，所以多了很多索引操作的独特api，其他Collection的功能List也都继承了。

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



总结

1、List系列集合特点

- ArrayList、LinkedList：有序，可重复，有索引。

2、List的实现类的底层原理

- ArrayList底层是基于数组实现的，根据查询元素快，增删相对慢。
- LinkedList底层基于双链表实现的，查询元素慢，增删首尾元素是非常快的。



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
 - ◆ List集合特点、特有API
 - ◆ List集合的遍历方式小结
 - ◆ ArrayList集合的底层原理
 - ◆ LinkedList集合的底层原理
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入

List集合的遍历方式有几种？

- ① 迭代器
- ② 增强for循环
- ③ Lambda表达式
- ④ for循环（因为List集合存在索引）

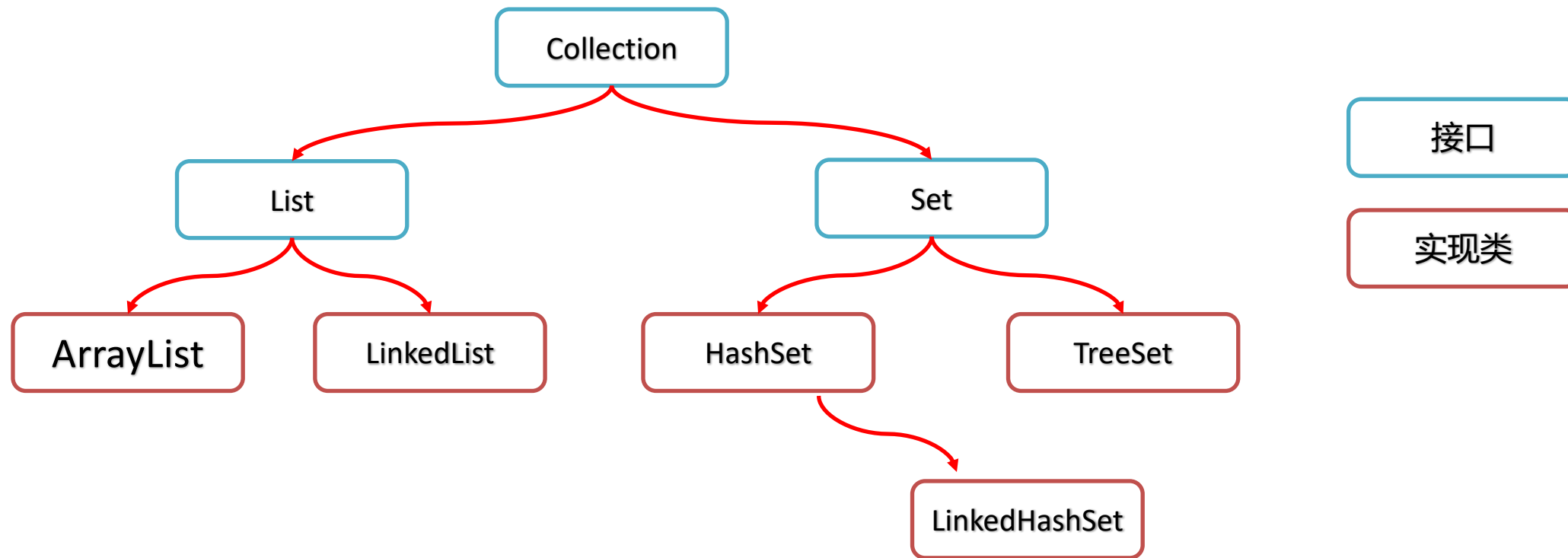


目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
 - ◆ List集合特点、特有API
 - ◆ List集合的遍历方式小结
 - ◆ ArrayList集合的底层原理
 - ◆ LinkedList集合特点
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入

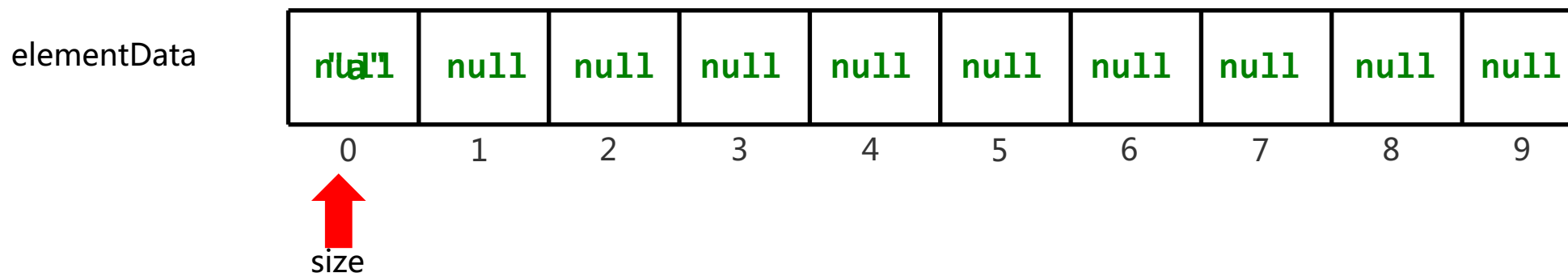
Collection集合体系



ArrayList集合底层原理

- ArrayList底层是基于数组实现的：根据索引定位元素快，增删需要做元素的移位操作。
- 第一次创建集合并添加第一个元素的时候，在底层创建一个默认长度为10的数组。

```
List<String> list = new ArrayList<>();  
list.add("a");
```



ArrayList集合底层原理

- ArrayList底层是基于数组实现的：根据索引定位元素快，增删需要做元素的移位操作。
- 第一次创建集合并添加第一个元素的时候，在底层创建一个默认长度为10的数组。

```
List<String> list = new ArrayList<>();  
list.add("a");
```

elementData

"a"	"b"	"c"	"d"	null	null	null	null	null	null
0	1	2	3	4	5	6	7	8	9

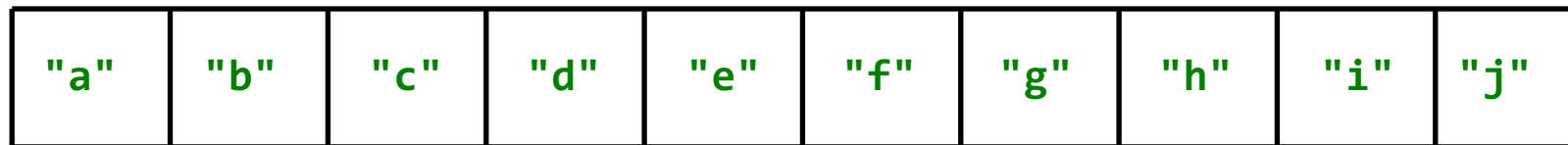


```
list.add("b");  
list.add("c");  
list.add("d");
```

思考：为何ArrayList查询快、增删元素相对较慢？

List集合存储的元素要超过容量怎么办？

elementData



0

1

2

3

4

5

6

7

8

9



size



0

1

2

3

4

5

6

7

8

9

...

null



size



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
 - ◆ List集合特点、特有API
 - ◆ List集合的遍历方式小结
 - ◆ ArrayList集合的底层原理
 - ◆ **LinkedList集合的底层原理**
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入

LinkedList的特点

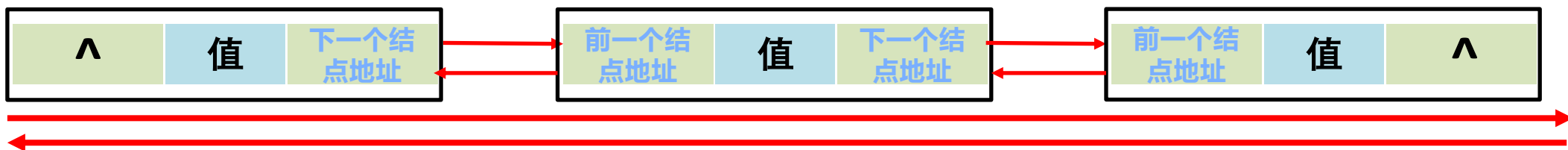
- 底层数据结构是双链表，查询慢，首尾操作的速度是极快的，所以多了很多首尾操作的特有API。

LinkedList集合的特有功能

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

链表的种类

双向
链表





目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- **补充知识：集合的并发修改异常问题**
- 补充知识：泛型深入



从集合中的一批元素中找出某些数据并删除，如何操作？是否存在问题呢？

问题引出

- 当我们从集合中找出某个元素并删除的时候可能出现一种并发修改异常问题。

哪些遍历存在问题？

- 迭代器遍历集合且直接用集合删除元素的时候可能出现。
- 增强for循环遍历集合且直接用集合删除元素的时候可能出现。

哪种遍历且删除元素不出问题

- 迭代器遍历集合但是用迭代器自己的删除方法操作可以解决。
- 使用for循环遍历并删除元素不会存在这个问题。



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入
 - ◆ 泛型的概述和优势
 - ◆ 自定义泛型类
 - ◆ 自定义泛型方法
 - ◆ 自定义泛型接口
 - ◆ 泛型通配符、上下限

泛型概述

- 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。
- 集合体系的全部接口和实现类都是支持泛型的使用的。

泛型的好处：

- 统一数据类型。
- 把运行时时期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，因为编译阶段类型就能确定下来。

泛型可以在很多地方进行定义:

类后面	→	泛型类
方法申明上	→	泛型方法
接口后面	→	泛型接口



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入
 - ◆ 泛型的概述和优势
 - ◆ 自定义泛型类
 - ◆ 自定义泛型方法
 - ◆ 自定义泛型接口
 - ◆ 泛型通配符、上下限

泛型类的概述

- 定义类时同时定义了泛型的类就是泛型类。
- 泛型类的格式：修饰符 class 类名<泛型变量>{ }

范例：public class MyArrayList<T> { }

- 此处泛型变量**T**可以随便写为任意标识，常见的如**E、T、K、V**等。
- **作用**：编译阶段可以指定数据类型，类似于集合的作用。

课程案例导学

- 模拟ArrayList集合自定义一个集合MyArrayList集合,完成添加和删除功能的泛型设计即可。

泛型类的原理：

- 把出现泛型变量的地方全部替换成传输的真实数据类型。



总结

1、泛型类的核心思想：

- 把出现泛型变量的地方全部替换成传输的真实数据类型

2、泛型类的作用

- 编译阶段约定操作的数据的类型，类似于集合的作用。



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入
 - ◆ 泛型的概述和优势
 - ◆ 自定义泛型类
 - ◆ 自定义泛型方法
 - ◆ 自定义泛型接口
 - ◆ 泛型通配符、上下限

泛型方法的概述

- 定义方法时同时定义了泛型的方法就是泛型方法。
- 泛型方法的格式：修饰符 <泛型变量> 方法返回值 方法名称(形参列表){}

范例： `public <T> void show(T t) { }`

- **作用**：方法中可以使用泛型接收一切实际类型的参数，方法更具备通用性。

课程案例导学

- 给你任何一个类型的数组，都能返回它的内容。也就是实现Arrays.toString(数组)的功能！

泛型方法的原理：

- 把出现泛型变量的地方全部替换成传输的真实数据类型。



总结

1、泛型方法的核心思想：

- 把出现泛型变量的地方全部替换成传输的真实数据类型

2、泛型方法的作用

- 方法中可以使用泛型接收一切实际类型的参数，方法更具备通用性



目录

Contents

- 集合概述
- Collection集合的体系特点
- Collection集合常用API
- Collection集合的遍历方式
- Collection集合存储自定义类型的对象
- 常见数据结构
- List系列集合
- 补充知识：集合的并发修改异常问题
- 补充知识：泛型深入
 - ◆ 泛型的概述和优势
 - ◆ 自定义泛型类
 - ◆ 自定义泛型方法
 - ◆ 自定义泛型接口
 - ◆ 泛型通配符、上下限

泛型接口的概述

- 使用了泛型定义的接口就是泛型接口。
- 泛型接口的格式：修饰符 interface 接口名称<泛型变量>{ }

范例：public interface Data<E>{ }

- **作用**：泛型接口可以让实现类选择当前功能需要操作的数据类型

课程案例导学

- 教务系统，提供一个接口可约束一定要完成数据（学生，老师）的增删改查操作

泛型接口的原理：

- 实现类可以在实现接口的时候传入自己操作的数据类型，这样重写的方法都将是针对于该类型的操作。



总结

1、泛型接口的作用

- 泛型接口可以约束实现类，实现类可以在实现接口的时候传入自己操作的数据类型这样重写的方法都将是针对于该类型的操作。



目录

Contents

- **Collection集合的体系特点**
- **Collection集合常用API**
- **Collection集合的遍历方式**
- **Collection集合存储自定义类型的对象**
- **常见数据结构**
- **List系列集合**
- **补充知识：集合的并发修改异常问题**
- **补充知识：泛型深入**
 - ◆ 泛型的概述和优势
 - ◆ 自定义泛型类
 - ◆ 自定义泛型方法
 - ◆ 自定义泛型接口
 - ◆ 泛型通配符、上下限

泛型通配符：案例导学

- 开发一个极品飞车的游戏，所有的汽车都能一起参加比赛。

通配符:?

- ? 可以在“使用泛型”的时候代表一切类型。
- E T K V 是在定义泛型的时候使用的。

注意：

- 虽然BMW和BENZ都继承了Car但是ArrayList<BMW>和ArrayList<BENZ>与ArrayList<Car>没有关系的！！

泛型的上下限：

- ? **extends Car**: ?必须是Car或者其子类 泛型上限
- ? **super Car** : ?必须是Car或者其父类 泛型下限





传智教育旗下高端IT教育品牌