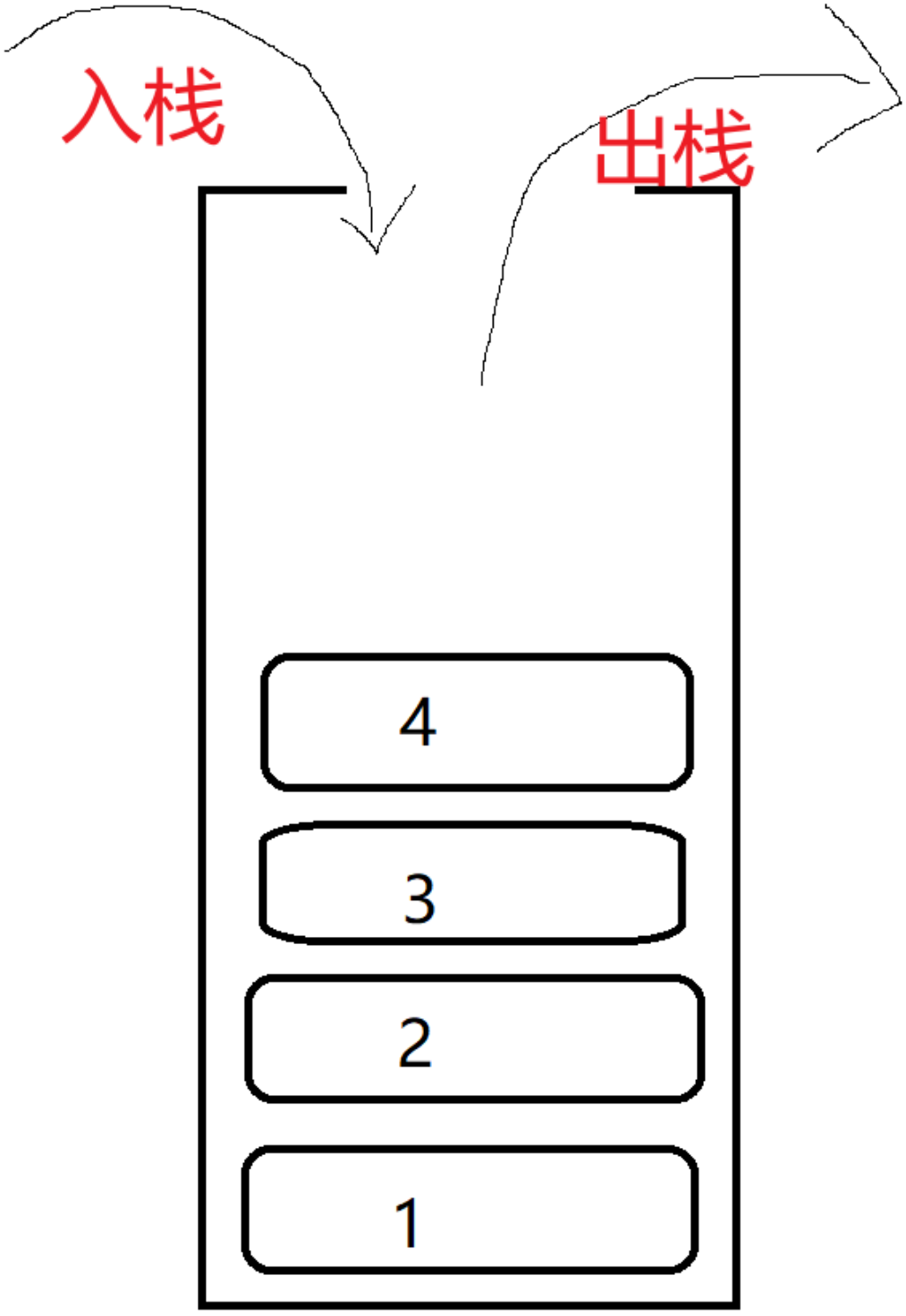
数据结构

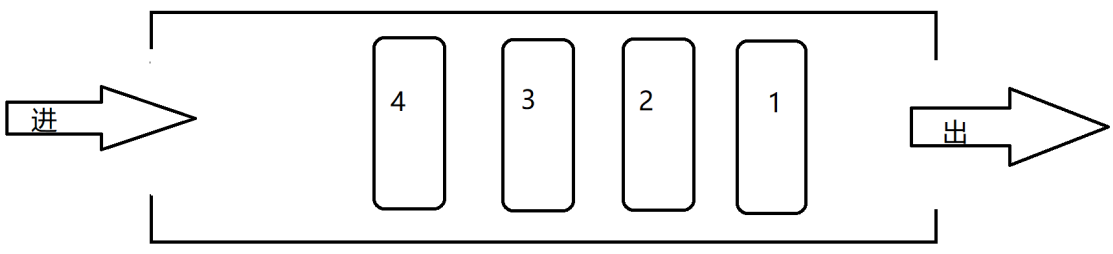
数据结构是计算机科学中的一个重要概念，用于组织和存储数据以便有效地进行访问、操作和管理。它涉及了如何在计算机内存中组织数据，以便于在不同操作中进行查找、插入、删除等操作

数据结构可以看作是一种数据的组织方式，不同的数据结构适用于不同的应用场景，根据操作的需求和效率要求，选择合适的数据结构可以提高算法的执行效率。

常见的数据结构

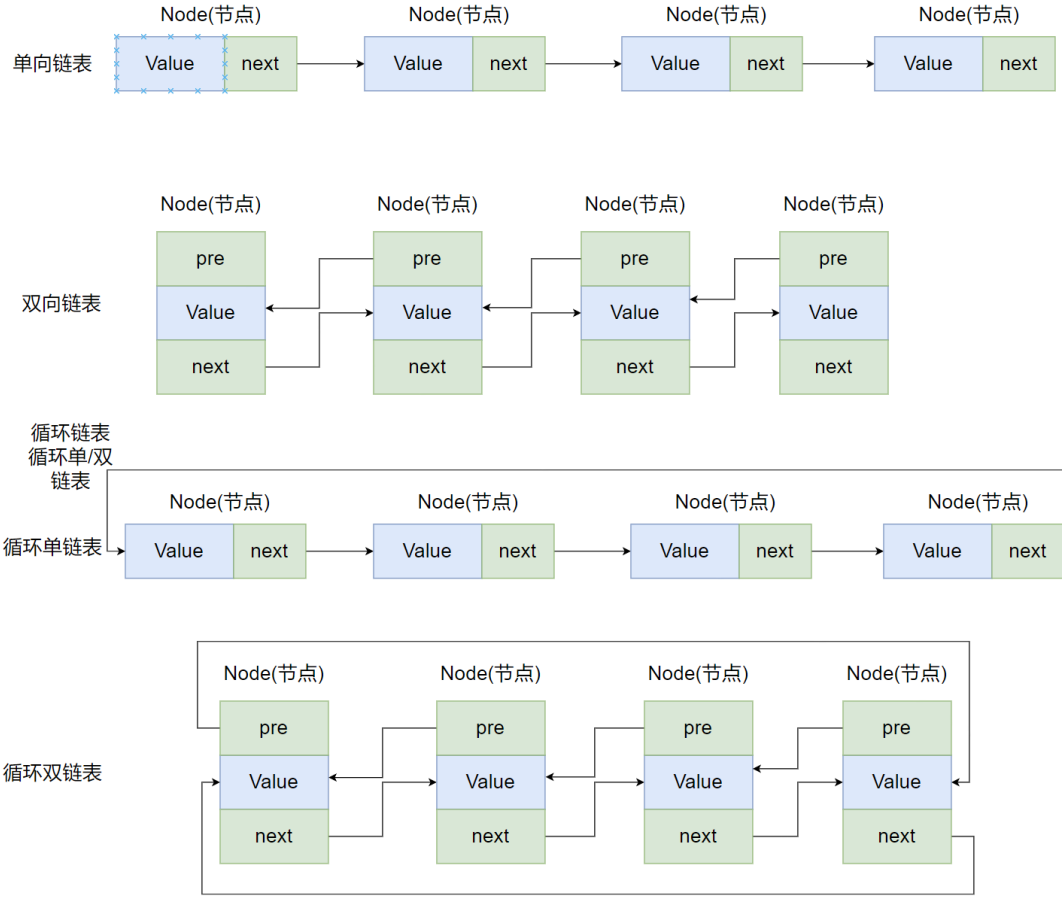
- 数组（Array） 将相同类型的数据元素按顺序存储在连续的内存单元中，可以通过索引快速访问元素，定长。
- 哈希表（Hash Table）：通过散列函数将键映射到值，实现高效的数据查找和插入
- 栈（Stack） 一种具有后进先出（LIFO）特性的数据结构，常用于处理函数调用、表达式求值等。
    - java虚拟机栈
    - 就像生活中的 弹夹（装子弹）



- 队列（Queue）：一种具有先进先出（FIFO）特性的数据结构，常用于任务调度、广度优先搜索等。
  - 

  - 像生活中安检机

- 链表（Linked List）：通过节点与节点之间的引用（指针）链接来存储数据，分为单向链表和双向链表、循环链表，对于插入和删除操作较为高效。

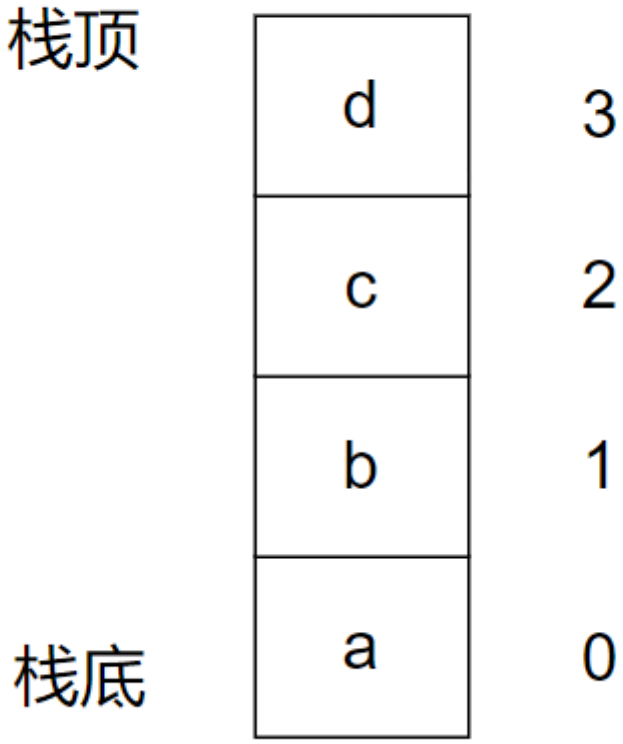


- 树（Tree）：一种层次结构的数据结构，包括二叉树、平衡树、二叉搜索树等，常用于搜索和排序操作。
- 图（Graph）：由节点和边构成的数据结构，用于表示各种复杂的关系和连接。

栈

如何创建一个类实现栈的功能？

- 底层存元素使用数组
- 添加元素始终添加到数组的最后一个
- 获取元素时永远从最后一个开始取元素
- 扩容问题




```java
public abstract class Stack {
    /**
     * 将元素压入栈顶
     * 入栈
     * @param element 要压入的元素
     */
    abstract void push(Object element);

    /**
     * 弹出栈顶元素并返回
     * 把栈顶元素删除，并返回
     * 出栈
     * @return 弹出的栈顶元素，如果栈为空返回 null
     */
    abstract Object pop();

    /**
     * 返回栈顶元素，但不弹出
     * @return 栈顶元素
     */
    abstract Object peek();

    /**
     * 检查栈是否为空
     * @return 如果栈为空则返回true，否则返回false
     */
```

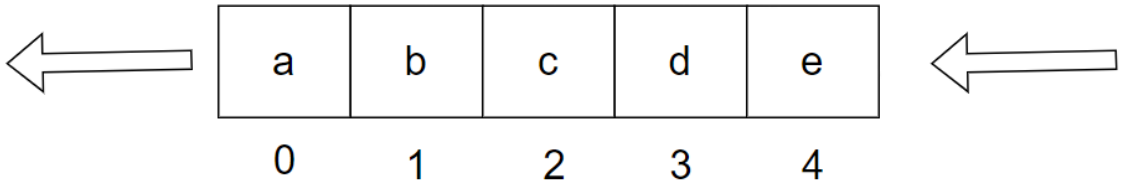
```
    abstract boolean isEmpty();

    /**
     * 返回栈中的元素个数
     * @return 栈中元素的个数
     */
    abstract int size();
}
```

> 通过继承抽象类 Stack 实现一个栈。并重写 toString/equals/hashCode 方法

队列

如何实现？

- 使用数组存元素
- 存元素还是放到数组的最后
- 取元素从第一个开始取




```
public abstract class Queue {
    /**
     * 将元素插入队尾
     * @param element 要插入的元素
     */
    void enqueue(Object element);

    /**
     * 移除并返回队首元素
     * 删除第一个元素，并返回
     * @return 队首元素，如果队列为空时，返回 null
     */
    abstract Object dequeue();

    /**
     * 返回队首元素，但不移除
     * @return 队首元素
     */
    abstract Object peek();

    /**
```

```
     * 检查队列是否为空
     * @return 如果队列为空则返回true，否则返回false
     */
    abstract boolean isEmpty();

    /**
     * 返回队列中的元素个数
     * @return 队列中元素的个数
     */
    abstract int size();
}
```

> 通过继承抽象类 Queue 实现一个队列。并重写 toString/equals/hashCode 方法

链表

敬请期待!