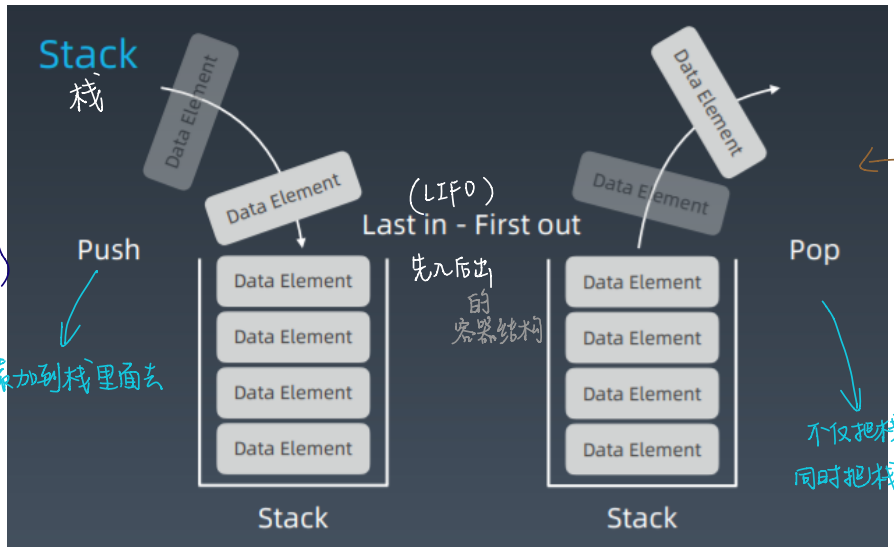
栈：先进来的元素，就窗在一起了，不能从下面出任何元素。

先入后出

添加 O(1)
删除 O(1)
查询 O(n)
(因为它是无序的)



先入后出的容器结构

与ArrayList区别
线程安全的

把一个元素加到栈里面去

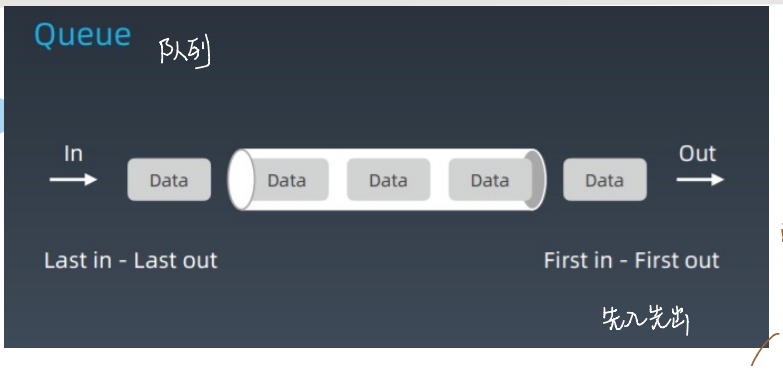
不仅把栈顶元素给返回，同时把栈顶元素去掉

```
示例代码 - Stack

Stack<Integer> stack = new Stack<>();
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
System.out.println(stack);
System.out.println(stack.search(4));

stack.pop();
stack.pop();
Integer topElement = stack.peek();
System.out.println(topElement);
System.out.println(" 3的位置 " + stack.search(3));
```

队列

先入先出

添加 O(1)
删除 O(1)
查询 O(n)



在Java里又是一个接口

Interface

底层可以实现的class很多

两组接口

add 添加 offer()
remove 删除 poll()

有异常现象会抛出异常的

返回特殊值

示例代码 - Queue

```
Queue<String> queue = new LinkedList<String>();
queue.offer("one");
queue.offer("two");
queue.offer("three");
queue.offer("four");
System.out.println(queue);

String polledElement = queue.poll();
System.out.println(polledElement);
System.out.println(queue);

String peekedElement = queue.peek();
System.out.println(peekedElement);
System.out.println(queue);

while(queue.size() > 0) {
  System.out.println(queue.poll());
}
```

API = offer 添加

poll 删除

假如到 queue 为空，则返回 null

双端队列（实战中常用）：

可理解为 Queue & Stack 的结合体

它可以往最前面添加进去，也可以往最前面 pop 出来

也可以往尾端添加元素，或者是取元素出去

（头和尾都可以进行元素的出和入）

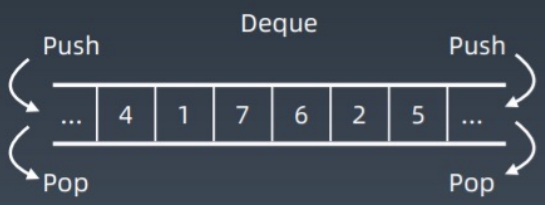
插入 O(1)

删除 O(1)

查询 O(n)

Deque: Double-End Queue

双端队列



在 Java 是一个接口 Interface

add 添加 offer

remove 删除 poll

抛异常 返回特殊值

示例代码 - Deque

```
Deque<String> deque = new LinkedList<String>();

deque.push("a");
deque.push("b");
deque.push("c");
System.out.println(deque);

String str = deque.peek();
System.out.println(str);
System.out.println(deque);

while (deque.size() > 0) {
  System.out.println(deque.pop());
}
System.out.println(deque);
```

背后的实现

example：数组的双端队列

并发的用 Linked List 实现的双端队列

Blocking（单线程）

并发式编程

non-blocking

concurrent

blocking

优先队列 (Priority Queue)

1、插入 O(1)

2、取出 O(logN) — 按照元素的优先级取出

相较于 stack

取出变慢.

But：顺序 (Like：UP 光行)

(保持了一定的有序性)

3、底层具体实现的数据结构较为多样和复杂：可以用 heap，BST (Binary Search Tree)，treap

(heap 堆也是多种实现的)

class

实现了接口

or

定义的一种抽象的数据结构

取优先级最高

这个函数里面放进去元素的话，必须实现一个 comparator (比较器) 方法

Stack、Queue、Deque 的工程实现

Google 中搜 "Stack Java 版本"

常见 API

查找源代码：Google 中搜 "Source Stack Java"

Queue

or

"Java Source code download"

视频质量好，good 好看

23 min 左右

复杂度分析

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)