栈和队列

栈

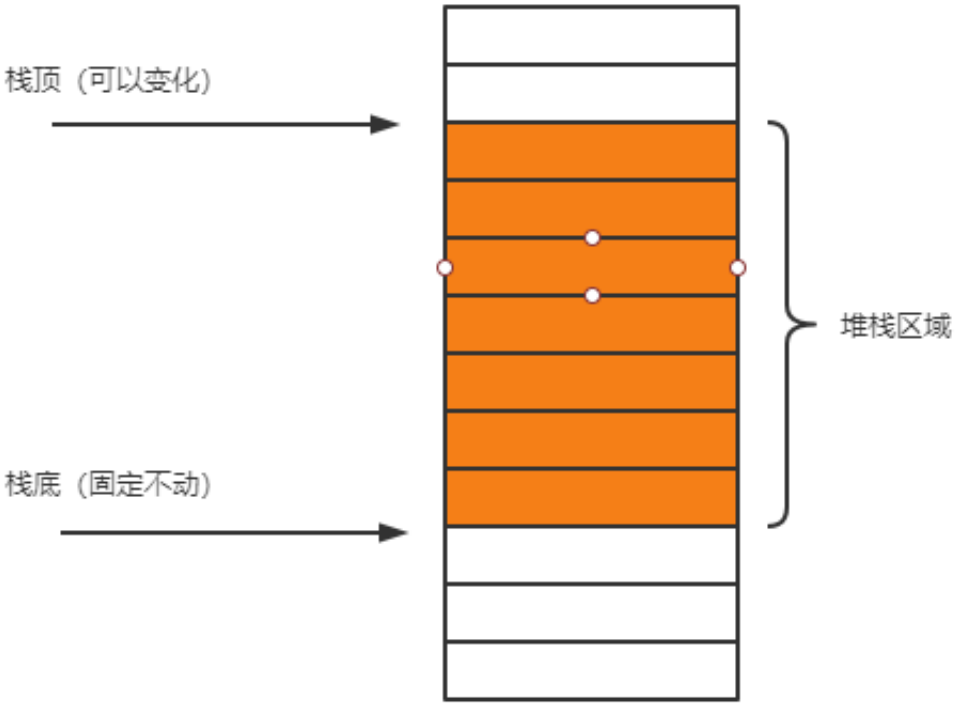
物理中的堆栈

既然讲到栈，那我们不如先来看一看，堆栈到底是个啥。这个就要从我们的打工人CPU开始说起。

在CPU当中，有一个非常核心的模块，叫做ALU（算术逻辑单元），也就是执行各种计算和逻辑运算操作的一个部件，是我们CPU的执行单元。比如1 + 1 = 2。如果说，有多个运算参与，比如 111 + 222 + 333，这个时候，他会先计算 111 + 222 ，然后得到一个临时结果，再将我们的临时结果和剩下的数字相加。这个时候，我们就需要将临时结果找个地方存一下，这个地方，我们叫寄存器，他们的名字就是AX，BX，CX，DX等等。临时结果就保存在寄存器中。为了实现更复杂的计算，能不能做特别多的寄存器呀？答案是不能，如果做特别多的寄存器，只会增加我们的CPU设计上的成本和复杂性。这个时候，就需要从外面找帮手。这个帮手需要什么条件呢？那就是速度要快。然后，计算机的设计者就将目光放在了内存条上面。

接下来，就要在内存条中划出一片专门的区域，用来临时存储数据。既然是专用的，那就需要有个名字。叫做栈。栈其实只是一个乳名，实际上这个区域叫做堆栈。要注意，内存里面还有一个区域，叫做堆。和栈的特性很不一样。所以，栈的本质就是内存中的一个区域。他的特殊之处就在于，CPU从中存取数据的方式。就好像弹夹装子弹，就是先入后出，后入先出。

而CPU中，很多对于数据的操作都要遵循这个规律。在内存中，有一个个的存储单元，在存储单元中，就有一片区域，就是堆栈。



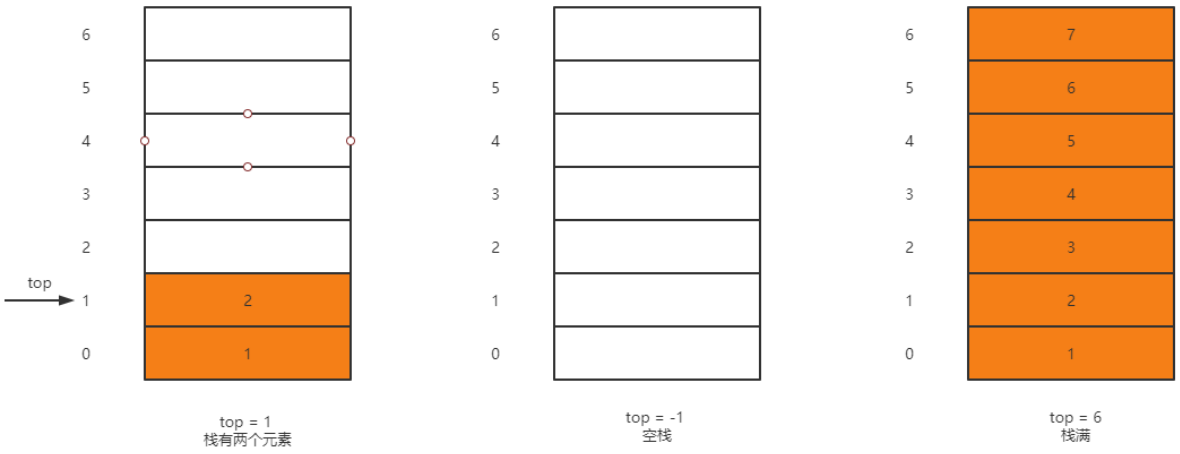
对于堆栈而言，如果栈顶和栈底重合，那么表示栈里面没有数据。如果需要往栈里面存放数据，那么栈顶指针就往上挪一挪，然后将要存储的数据存放在栈顶的位置。这个动作就叫做压栈\入栈。当需要从栈中取出数据的时候，就先将数据保存到寄存器当中，然后栈顶往下挪，这个动作就叫做弹栈\出栈。但是要注意，出栈了以后，数据还是在堆栈中，只是这个数据就被当成了垃圾。

再进一步理解，就是我们的程序。要知道，我们的程序是以机器码的形式躺平在内存当中，每一句机器码都有自己的位置，也就是地址。CPU执行程序的过程，就是把每一句机器码拿出来挨个分析，然后做相应操作。比如函数调用的地址存取，就用到了堆栈。

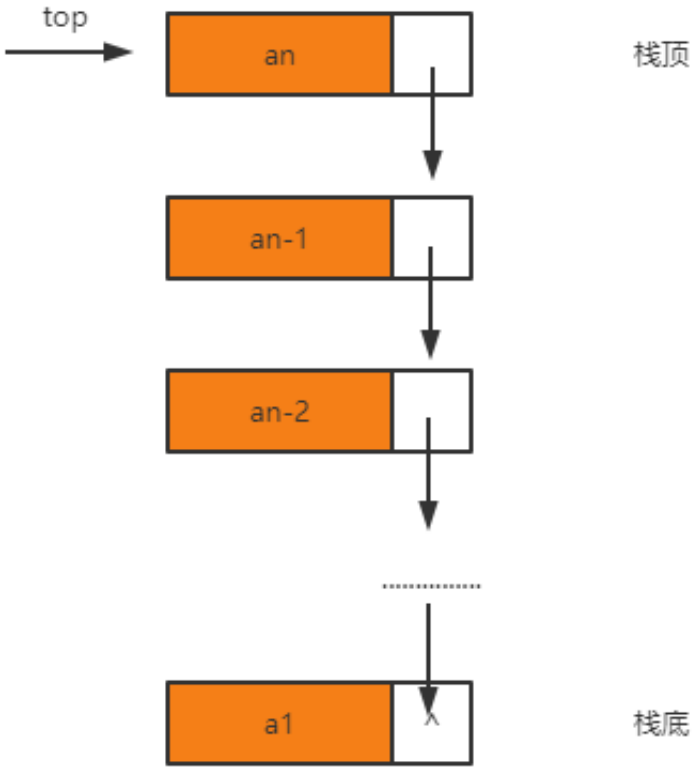
数据结构中的栈

数据结构中的栈，跟物理中的栈一样么？当然是不一样。数据结构中的栈，我们就将其称之为栈结构的一个抽象数据类型。也就是模仿了栈结构的特点，而做出的一系列动作。

前面我们讲了，对于计算机而言，有两种存储结构。就是顺序存储结构、链式存储结构。同样的，我们可以用这两种结构来实现栈的操作。



顺序存储结构



链式存储结构

队列

刚才我们介绍了一种先进后出的结构，栈。接下来，我们介绍一种正好跟栈相反顺序的结构，队列。

队列是一种特殊的线性表，特殊之处就在于它只允许在表的前端进行删除操作，在表的后端进行插入操作。和栈一样，队列也是一种操作受到限制的线性表。进行插入操作的端称之为队尾，进行删除操作的端称之为队头。队列中没有队列的时候，称之为空队列。队列的数据元素，又叫做队列元素。在队列中插入一个队列元素称之为入队，在队列中删除一个队列元素，称之为出队。因为队列只允许在一端插入，在另一端删除，所以只有最早进入的队列元素才可以从队列中删除，故队列又称为先进先出线性表。

队列的应用

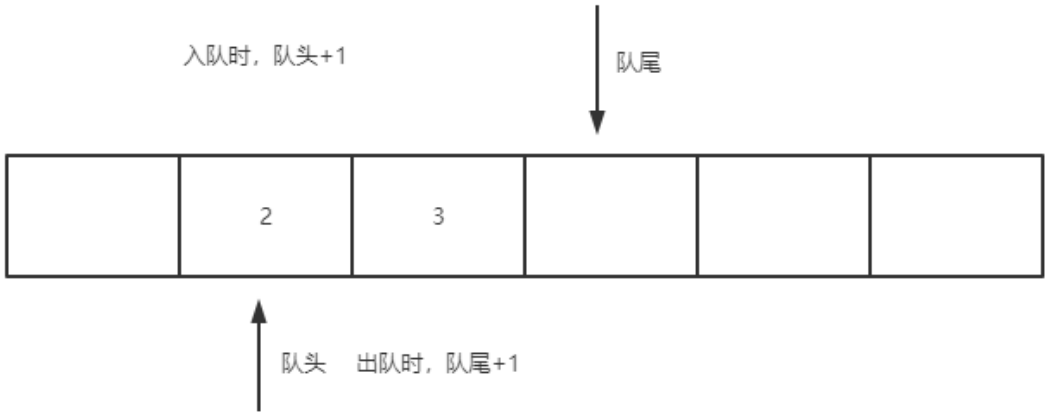
解决主机与外部设备速度不匹配

多用户引起的资源竞争问题

单向队列

顺序队列

当我用一片连续的存储空间来存储队列中的数据元素的时候，这样的队列就称之为顺序队列。类似于顺序栈。用一维数组来存放顺序队列中的数据元素。队头设置在最近一个离开队列元素所占的位置。队尾设置在最近一个进行入队列的元素位置。那么队头和队尾随着插入和删除的变化而变化。当队列为空时，front = rear；队列中的元素个数可以由队头 - 队尾求得。



但是，这个时候，会有一个问题。当我们的队尾指针指向size - 1 时，代表长度已满。但是根据队列的规则，就实际情况来说，他还有空闲的空间。那么这个时候，我们就将其称之为假溢出。

为了解决假溢出的问题，就是将我们的顺序队列看成是首位相接的循环结构。首尾指示器不变，这种队列就叫做，循环顺序队列。

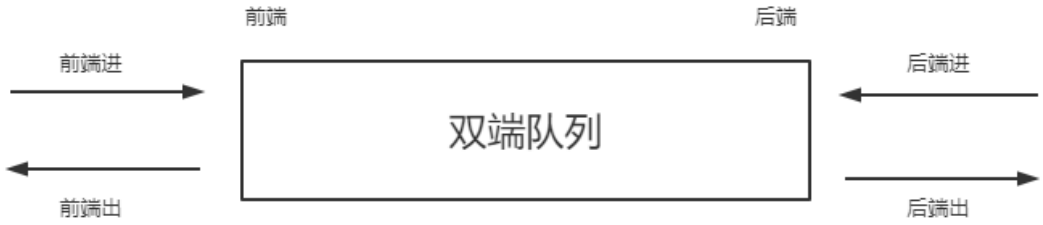
也就是说，当我们的队尾元素达到数组的上限时，如果还有数据元素入队并且数组的第0个空间是空闲时，队尾指示器就指向数组的0端，所以。整个队尾指示器+1的操作就可以修改为：rear = （rear + 1）%maxSize；队头指示器同样是如此。当队头的操作达到数组的上限的时候，如果还有数组元素出队，这个时候，队头指示器就要指向数组的0端。所以，队头指示器+1的操作就是 front = （front + 1）%maxSize。

这样的话，就又有一个问题，队满和队空的时候，都有rear = front。为了解决这个问题，一般的方法就是，少使用一个空间。所以，我们判断队空的条件就变成了 rear = front。判断队满的条件是（rear + 1）%maxSize = front。与此同时，循环队列中数据元素的个数是（rear - front + maxSize）% maxSize。

链队列

双端队列

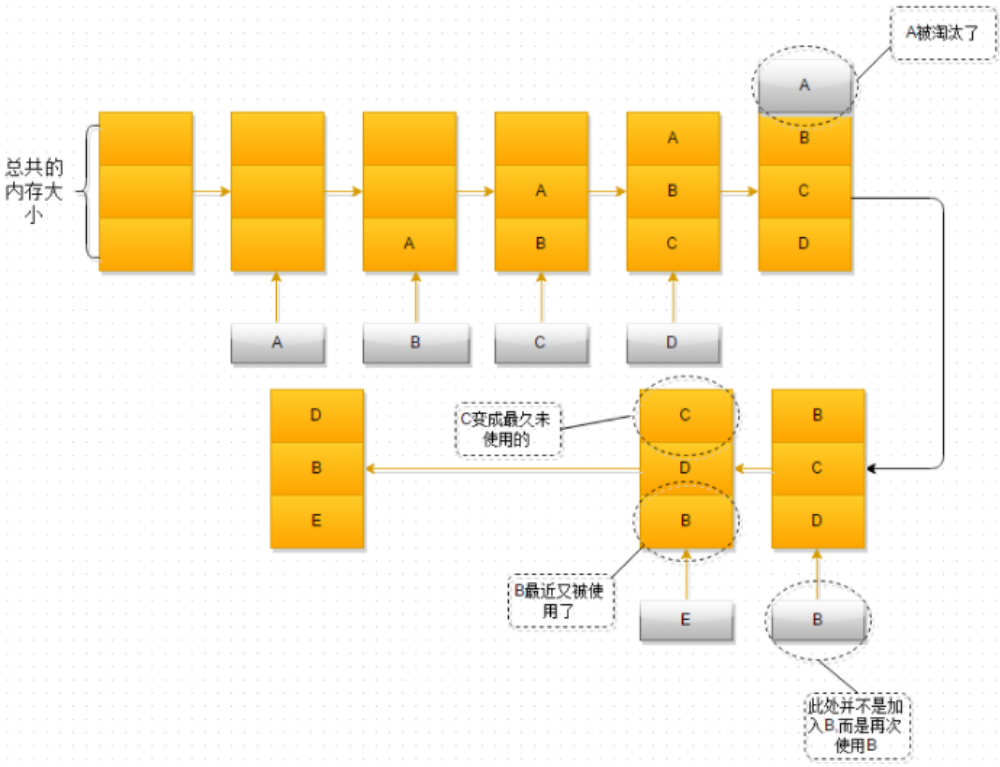
双端队列如图示



所以说，对于双端队列来说，就是两端都是结尾的队列。队列的每一端都可以插入数据项和移除数据项。相对于普通队列。双端队列的入队和出队操作在两端都可以进进行。

这种数据结构的特性，使得他更加的实用和方便。当你只允许使用一端出队、入队操作的时候，他等价于一个栈。当限制一端只能出队，另一端只能入队，他就等价于一个普通队列。

LRU（Least Recently Used）缓存淘汰



1、新数据插入到链表头部

2、每当缓存命中（即缓存数据被访问），则将数据移到链表头部

3、当链表满的时候，将链表尾部的数据丢弃

这里我们是根据时间来来判断的，就是最近最久未使用的。如果根据使用次数来判断，做缓存的命中，那就叫做LFU（Least Freequently used）。目前Redis应该就是用的LFU。

所以说，这些框架的底层，并不难。这也是为什么大厂就是揪着数据结构与算法不放。

延迟队列

我们学了，队列是一种先进先出的数据结构。普通队列中，队列的元素是有顺序的，先进入队列的会被优先取出来消费。

延迟队列相比于普通队列最大的区别就是在属性上面，普通的队列是先进先出，按入队顺序处理。延时队列中的元素在入队时会制定一个延迟时间，表示其希望能够在经过该指定时间后处理。从某种意义来说，他不像是一个队列。更像是一个以时间为权重的堆。

我在开发业务需求时遇到的使用场景是这样的，用户可以在小程序中订阅不同的微信或者 QQ 的模板消息，产品同学可以在小程序的管理端新建消息推送计划，当到达指定的时间节点的时候给所有订阅模板消息的用户进行消息推送。

如果仅仅是服务单一的小程序，那也许起个定时任务，或者甚至人工的定时去执行能够最便捷最快速的去完成这项需求，但我们希望能够抽象出一个消息订阅的模块服务出来给所有业务使用，这时候就需要一种通用的系统的解决方案，这时候便需要使用到延迟队列了。

除了上述我所遇到的这样的典型的需求以外，延迟队列的应用场景其实也非常的广泛，比如说以下的场景：

新建的订单，如果用户在 15 分钟内未支付，则自动取消。

公司的会议预定系统，在会议预定成功后，会在会议开始前半小时通知所有预定该会议的用户。

安全工单超过 24 小时未处理，则自动拉企业微信群提醒相关责任人。

用户下单外卖以后，距离超时时间还有 10 分钟时提醒外卖小哥即将超时。

对于数据量比较少并且时效性要求不那么高的场景，一种比较简单的方式是轮询数据库，比如每秒轮询一下数据库中所有数据，处理所有到期的数据，比如如果我是公司内部的会议预定系统的开发者，我可能就会采用这种方案，因为整个系统的数据量必然不会很大并且会议开始前提前 30 分钟提醒与提前 29 分钟提醒的差别并不大。

但是如果需要处理的数据量比较大实时性要求比较高，比如淘宝每天的所有新建订单 15 分钟内未支付的自动超时，数量级高达百万甚至千万，这时候如果你还敢轮询数据库怕是要被你老板打死，不被老板打死估计也要被运维同学打死。

这种场景下，就需要使用到我们今天的主角 —— 延迟队列了。

阻塞队列

阻塞队列是这样的一种数据结构，它是一个队列（类似于一个List），可以存放0到N个元素。我们可以对这个队列执行插入或弹出元素操作，弹出元素操作就是获取队列中的第一个元素，并且将其从队列中移除；而插入操作就是将元素添加到队列的末尾。当队列中没有元素时，对这个队列的弹出操作将会被阻塞，直到有元素被插入时才会被唤醒；当队列已满时，对这个队列的插入操作就会被阻塞，直到有元素被弹出后才会被唤醒。

在线程池中，往往就会用阻塞队列来保存那些暂时没有空闲线程可以直接执行的任务，等到线程空闲之后再从阻塞队列中弹出任务来执行。一旦队列为空，那么线程就会被阻塞，直到有新任务被插入为止。