08|栈：如何实现浏览器的前进和后退功能？

浏览器的前进、后退功能，我想你肯定很熟悉吧？

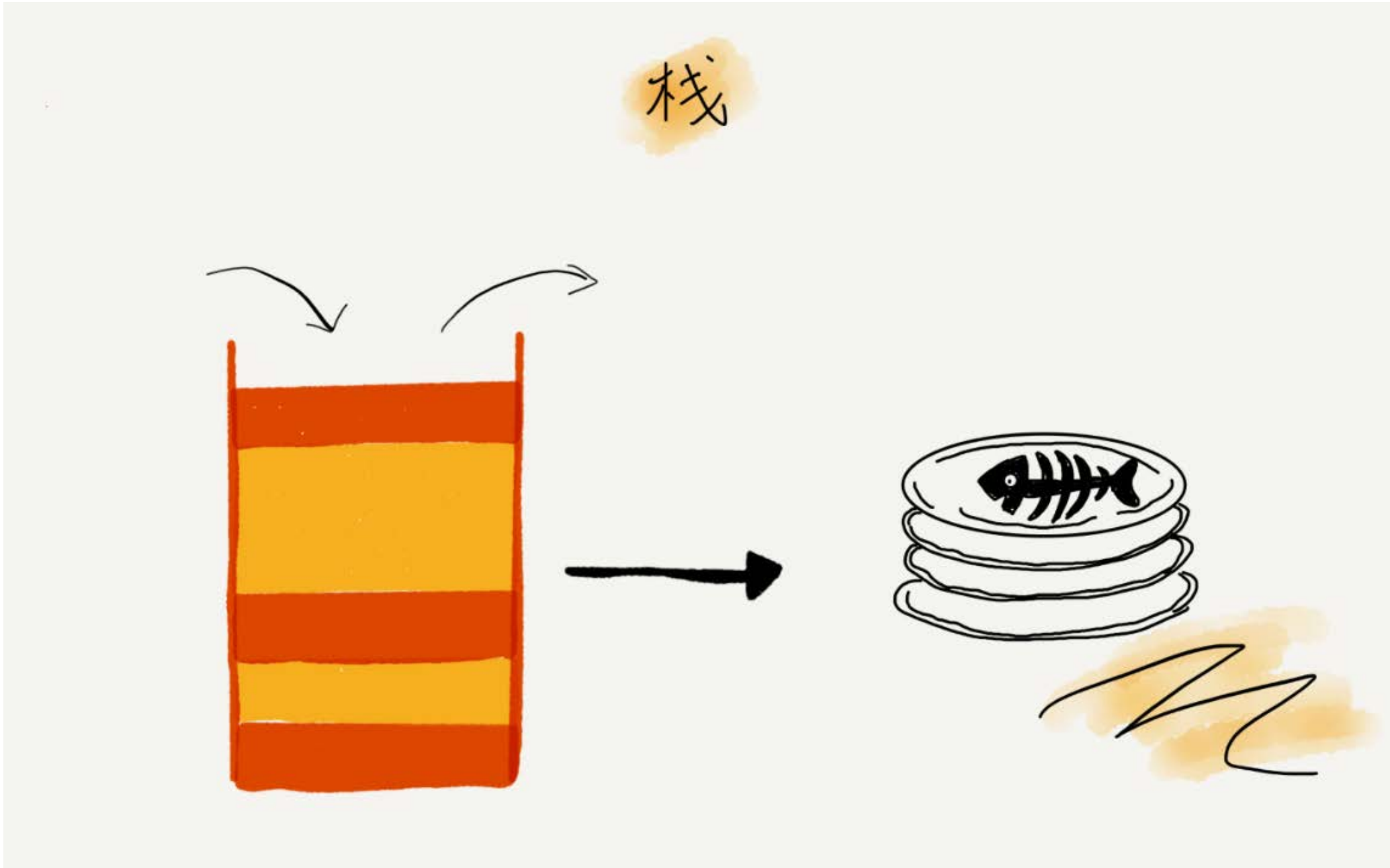
当你依次访问完一串页面a-b-c之后，点击浏览器的后退按钮，就可以查看之前浏览过的页面b和a。当你后退到页面a，点击前进按钮，就可以重新查看页面b和c。但是，如果你后退到页面b后，点击了新的页面d，那就无法再通过前进、后退功能查看页面c了。

假设你是**Chrome**浏览器的开发工程师，你会如何实现这个功能呢？

这就要用到我们今天要讲的“栈”这种数据结构。带着这个问题，我们来学习今天的内容。

如何理解“栈”？

关于“栈”，我有一个非常贴切的例子，就是一摞叠在一起的盘子。我们平时放盘子的时候，都是从下往上一个一个放；取的时候，我们也是从上往下一个一个地依次取，不能从中间任意抽出。后进者先出，先进者后出，这就是典型的**“栈”**结构。



从栈的操作特性上来看，栈是一种“操作受限”的线性表，只允许在一端插入和删除数据。

我第一次接触这种数据结构的时候，就对它存在的意义产生了很大的疑惑。因为我觉得，相比数组和链表，栈带给我的只有限制，并没有任何优势。那我直接使用数组或者链表不就好了吗？为什么还要用这个“操作受限”的“栈”呢？

事实上，从功能上来说，数组或链表确实可以替代栈，但你要知道，特定的数据结构是对特定场景的抽象，而且，数组或链表暴露了太多的操作接口，操作上的确灵活自由，但使用时就比较不可控，自然也就更容易出错。

当某个数据集合只涉及在一端插入和删除数据，并且满足后进先出、先进后出的特性，我们就应该首选“栈”这种数据结构。

如何实现一个“栈”？

从刚才栈的定义里，我们可以看出，栈主要包含两个操作，入栈和出栈，也就是在栈顶插入一个数据和从栈顶删除一个数据。理解了栈的定义之后，我们来看一看如何用代码实现一个栈。

实际上，栈既可以用数组来实现，也可以用链表来实现。用数组实现的栈，我们叫作顺序栈，用链表实现的栈，我们叫作链式栈。

我这里实现一个基于数组的顺序栈。基于链表实现的链式栈的代码，你可以自己试着写一下。我会将我写好的代码放到Github上，你可以去看一下自己写的是否正确。

我这段代码是用Java来实现的，但是不涉及任何高级语法，并且我还用中文做了详细的注释，所以你应该是可以看懂的。

```
// 基于数组实现的顺序栈
public class ArrayStack {
  private String[] items;  // 数组
  private int count;       // 栈中元素个数
  private int n;           //栈的大小

  // 初始化数组，申请一个大小为n的数组空间
  public ArrayStack(int n) {
    this.items = new String[n];
    this.n = n;
    this.count = 0;
  }

  // 入栈操作
  public boolean push(String item) {
    // 数组空间不够了，直接返回false，入栈失败。
    if (count == n) return false;
    // 将item放到下标为count的位置，并且count加一
    items[count] = item;
    ++count;
    return true;
  }

  // 出栈操作
  public String pop() {
    // 栈为空，则直接返回null
    if (count == 0) return null;
    // 返回下标为count-1的数组元素，并且栈中元素个数count减一
    String tmp = items[count-1];
    --count;
    return tmp;
  }
}
```

了解了定义和基本操作，那它的操作的时间、空间复杂度是多少呢？

不管是顺序栈还是链式栈，我们存储数据只需要一个大小为n的数组就够了。在入栈和出栈过程中，只需要一两个临时变量存储空间，所以空间复杂度是O(1)。

注意，这里存储数据需要一个大小为n的数组，并不是说空间复杂度就是O(n)。因为，这n个空间是必须的，无法省掉。所以我们说空间复杂度的时候，是指除了原本的数据存储空间外，算法运行还需要额外的存储空间。

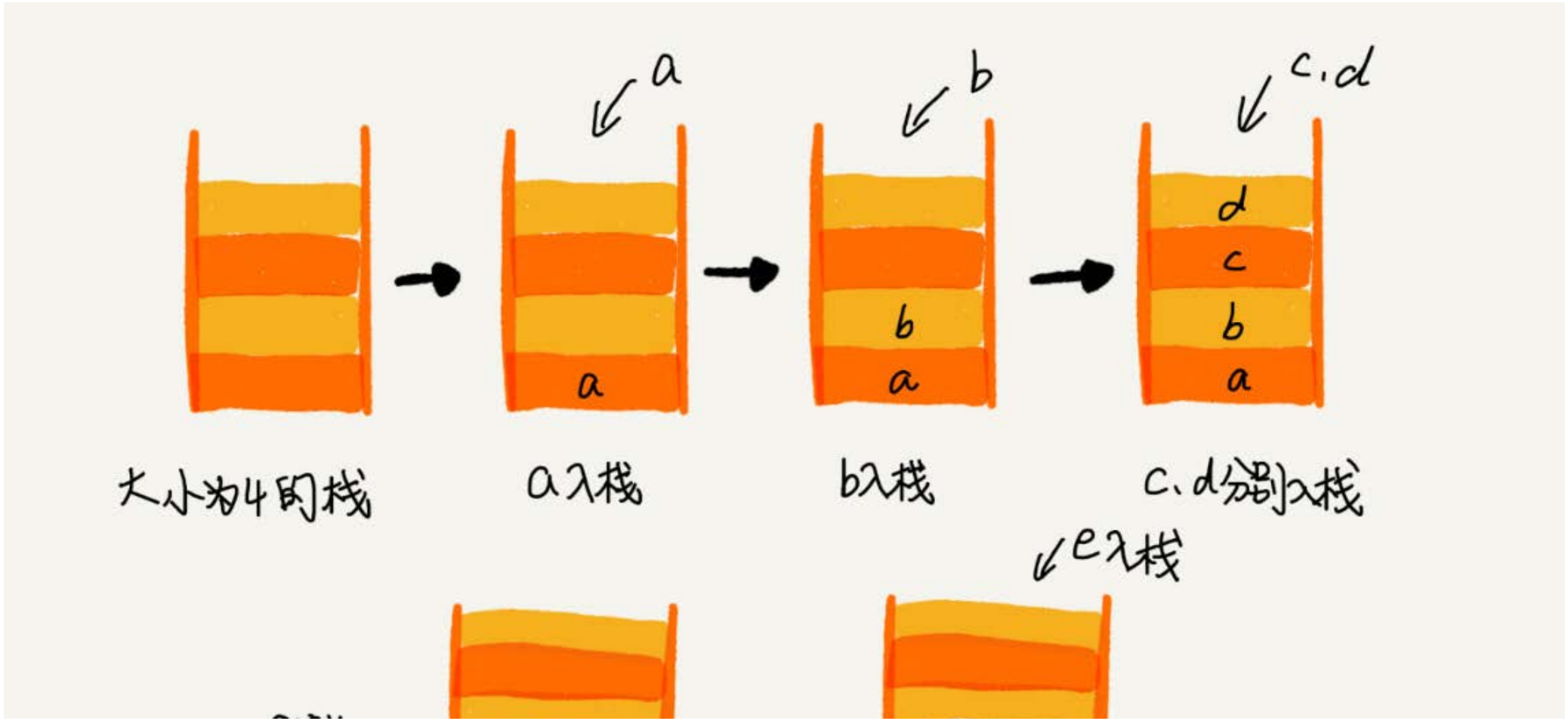
空间复杂度分析是不是很简单？时间复杂度也不难。不管是顺序栈还是链式栈，入栈、出栈只涉及栈顶个别数据的操作，所以时间复杂度都是O(1)。

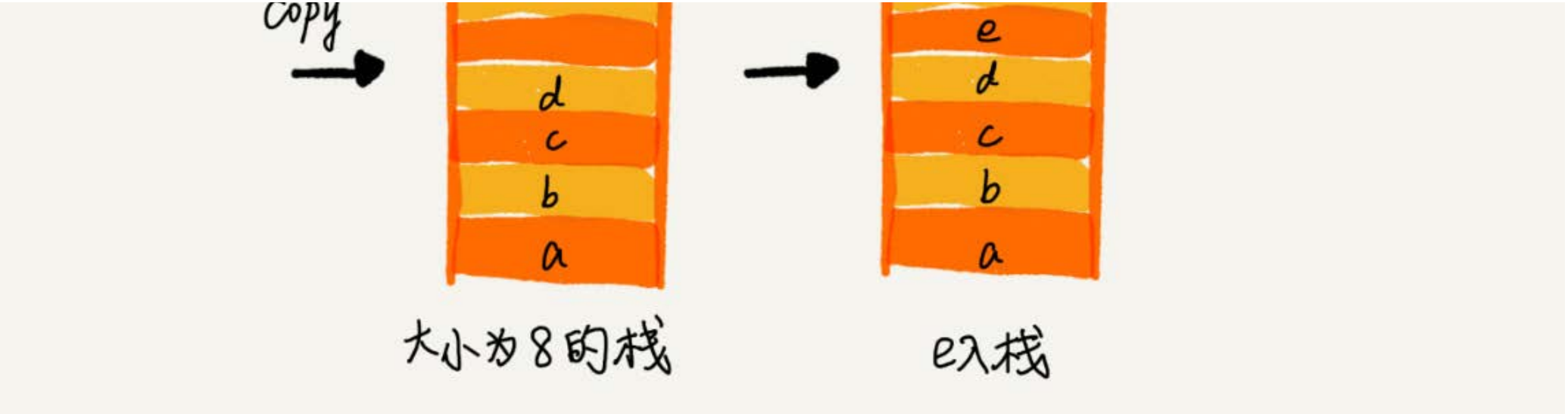
支持动态扩容的顺序栈

刚才那个基于数组实现的栈，是一个固定大小的栈，也就是说，在初始化栈时需要事先指定栈的大小。当栈满之后，就无法再往栈里添加数据了。尽管链式栈的大小不受限，但要存储next指针，内存消耗相对较多。那我们如何基于数组实现一个可以支持动态扩容的栈呢？

你还记得，我们在数组那一节，是如何来实现一个支持动态扩容的数组的吗？当数组空间不够时，我们就重新申请一块更大的内存，将原来数组中数据统统拷贝过去。这样就实现了一个支持动态扩容的数组。

所以，如果要实现一个支持动态扩容的栈，我们只需要底层依赖一个支持动态扩容的数组就可以了。当栈满了之后，我们就申请一个更大的数组，将原来的数据搬移到新数组中。我画了一张图，你可以对照着理解一下。





实际上，支持动态扩容的顺序栈，我们平时开发中并不常用到。我讲这一块的目的，主要还是希望带你练习一下前面讲的复杂度分析方法。所以这一小节的重点是复杂度分析。

你不用死记硬背入栈、出栈的时间复杂度，你需要掌握的是分析方法。能够自己分析才算是真正掌握了。现在我就带你分析一下支持动态扩容的顺序栈的入栈、出栈操作的时间复杂度。

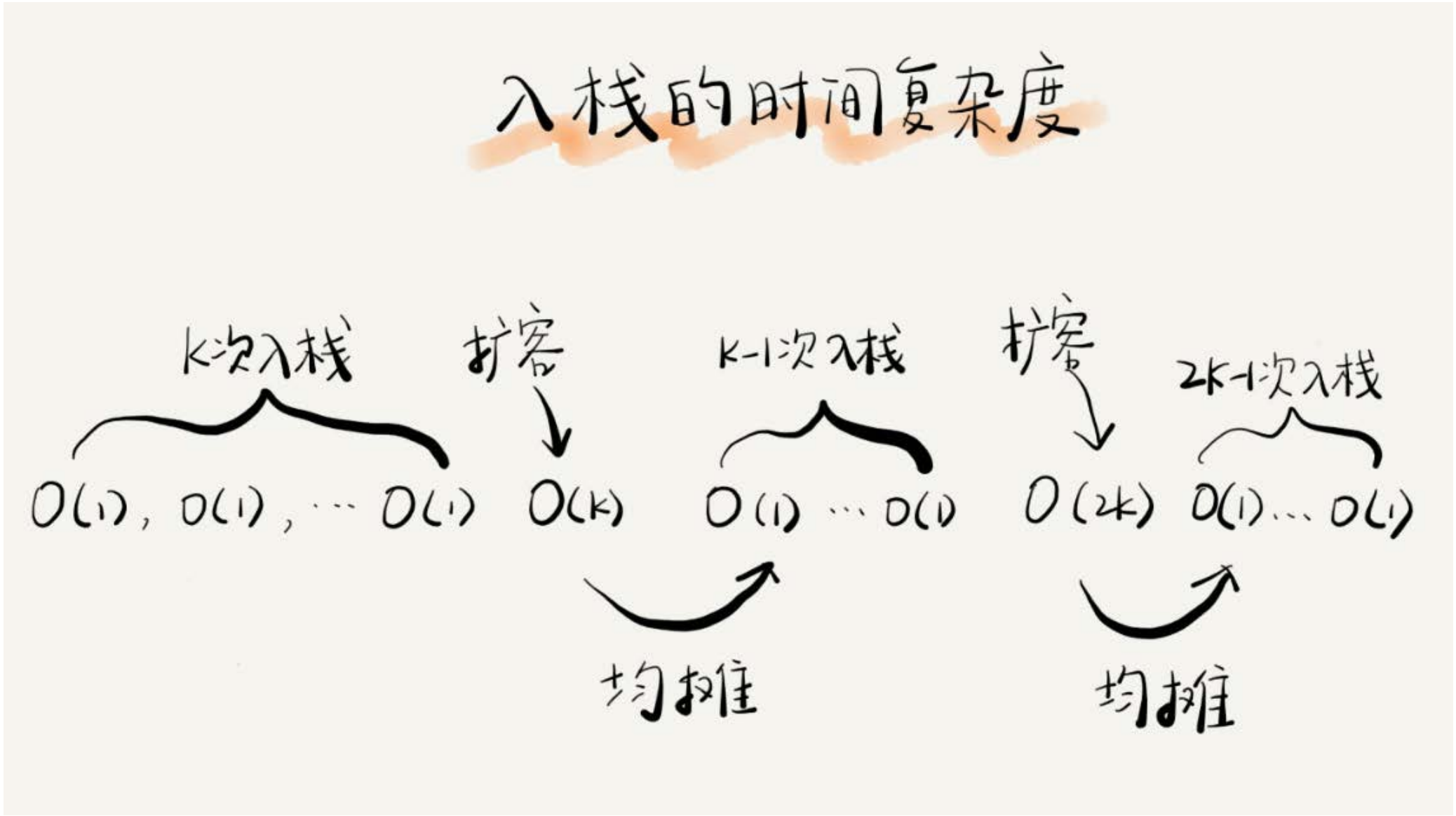
对于出栈操作来说，我们不会涉及内存的重新申请和数据的搬移，所以出栈的时间复杂度仍然是O(1)。但是，对于入栈操作来说，情况就不一样了。当栈中有空闲空间时，入栈操作的时间复杂度为O(1)。但当空间不够时，就需要重新申请内存和数据搬移，所以时间复杂度就变成了O(n)。

也就是说，对于入栈操作来说，最好情况时间复杂度是O(1)，最坏情况时间复杂度是O(n)。那平均情况下的时间复杂度又是多少呢？还记得我们在复杂度分析那一节中讲的摊还分析法吗？这个入栈操作的平均情况下的时间复杂度可以用摊还分析法来分析。我们也正好借此来实战一下摊还分析法。

为了分析的方便，我们需要事先做一些假设和定义：

- 栈空间不够时，我们重新申请一个是原来大小两倍的数组；
- 为了简化分析，假设只有入栈操作没有出栈操作；
- 定义不涉及内存搬移的入栈操作为simple-push操作，时间复杂度为O(1)。

如果当前栈大小为K，并且已满，当再有新的数据要入栈时，就需要重新申请2倍大小的内存，并且做K个数据的搬移操作，然后再入栈。但是，接下来的K-1次入栈操作，我们都不需要再重新申请内存和搬移数据，所以这K-1次入栈操作都只需要一个simple-push操作就可以完成。为了让你更加直观地理解这个过程，我画了一张图。



你应该可以看出来，这K次入栈操作，总共涉及了K个数据的搬移，以及K次simple-push操作。将K个数据搬移均摊到K次入栈操作，那每个入栈操作只需要一个数据搬移和一个simple-push操作。以此类推，入栈操作的均摊时间复杂度就为O(1)。

通过这个例子的实战分析，也印证了前面讲到的，均摊时间复杂度一般都等于最好情况时间复杂度。因为在大部分情况下，入栈操作的时间复杂度O都是O(1)，只有在个别时刻才会退化为O(n)，所以把耗时多的入栈操作的时间均摊到其他入栈操作上，平均情况下的耗时就接近O(1)。

栈在函数调用中的应用

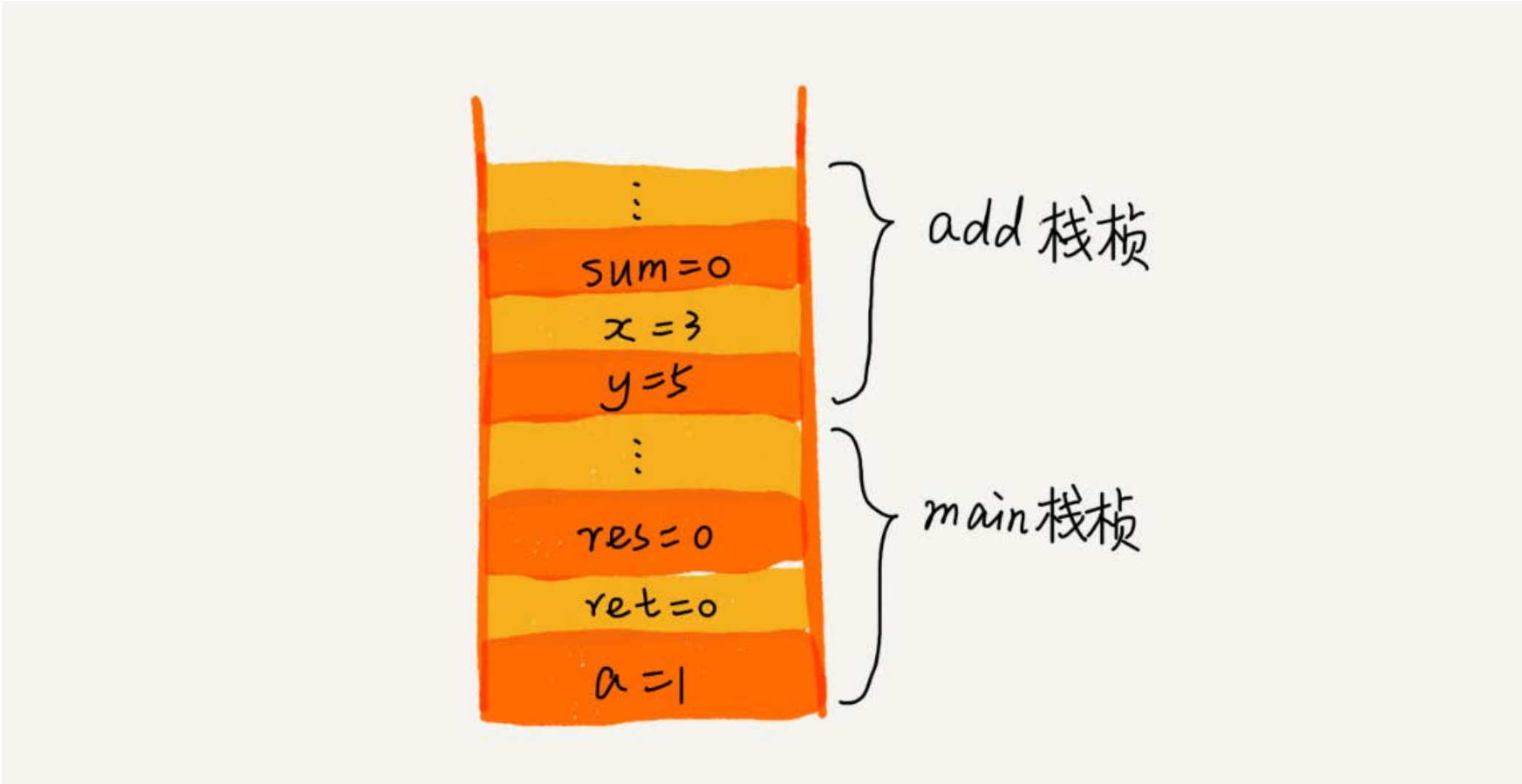
前面我讲的都比较偏理论，我们现在来看下，栈在软件工程中的实际应用。栈作为一个比较基础的数据结构，应用场景还是蛮多的。其中，比较经典的一个应用场景就是函数调用栈。

我们知道，操作系统给每个线程分配了一块独立的内存空间，这块内存被组织成“栈”这种结构，用来存储函数调用时的临时变量。每进入一个函数，就会将临时变量作为一个栈帧入栈，当被调用函数执行完成，返回之后，将这个函数对应的栈帧出栈。为了让你更好地理解，我们一块来看下这段代码的执行过程。

```
int main() {
   int a = 1;
   int ret = 0;
   int res = 0;
   ret = add(3, 5);
   res = a + ret;
   printf("%d", res);
   reuturn 0;
}

int add(int x, int y) {
   int sum = 0;
   sum = x + y;
   return sum;
}
```

从代码中我们可以看出，main()函数调用了add()函数，获取计算结果，并且与临时变量a相加，最后打印res的值。为了让你清晰地看到这个过程对应的函数栈里出栈、入栈的操作，我画了一张图。图中显示的是，在执行到add()函数时，函数调用栈的情况。



栈在表达式求值中的应用

我们再来看栈的另一个常见的应用场景，编译器如何利用栈来实现表达式求值。

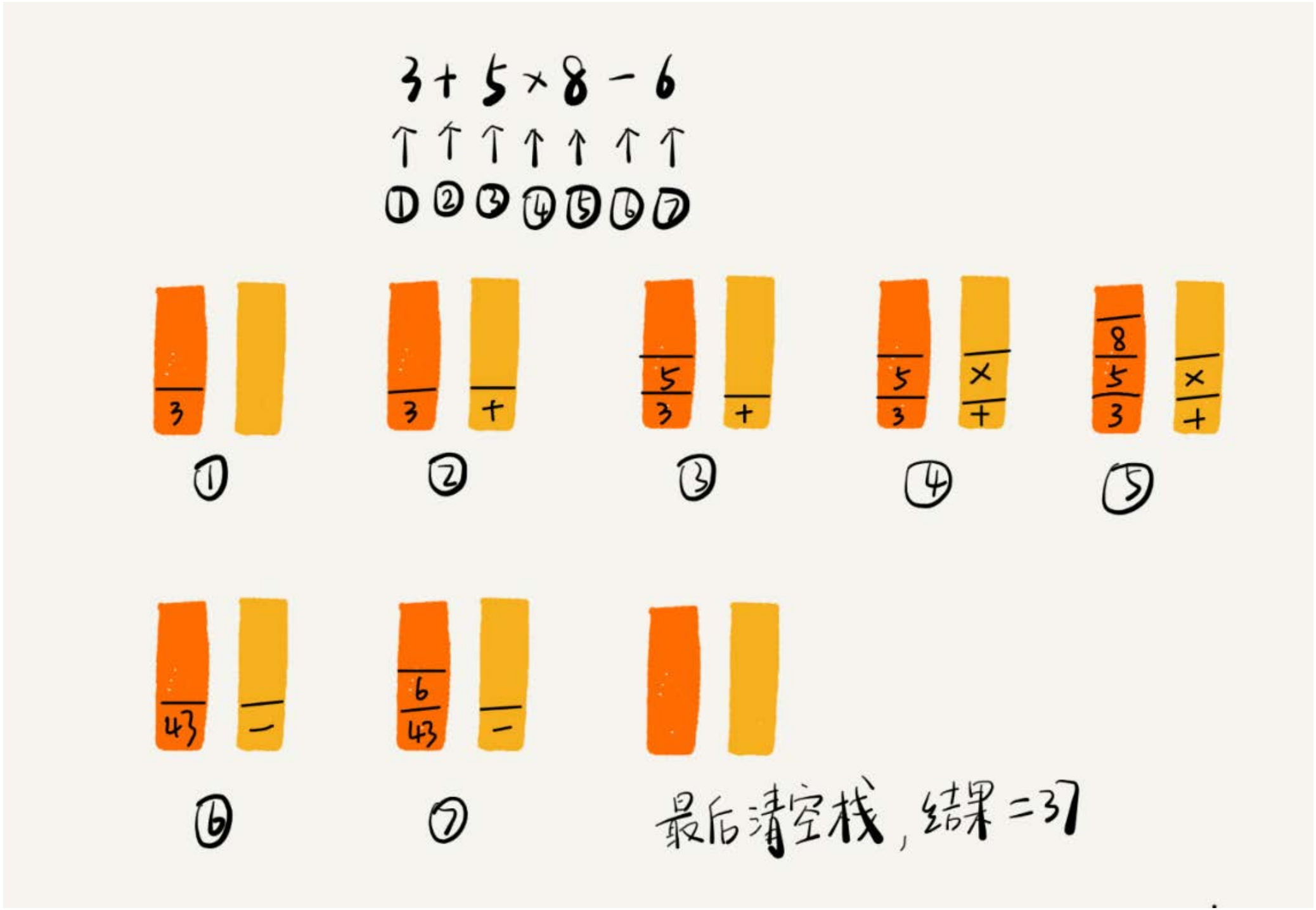
为了方便解释，我将算术表达式简化为只包含加减乘除四则运算，比如：34+13*9+44-12/3。对于这个四则运算，我们人脑可以很快求解出答案，但是对于计算机来说，理解这个表达式本身就是个挺难的事儿。如果换作你，让你来实现这样一个表达式求值的功能，你会怎么做呢？

实际上，编译器就是通过两个栈来实现的。其中一个保存操作数的栈，另一个是保存运算符的栈。我们从左向右遍历表达式，当遇到数字，我们就直接压入操作数栈；当遇到运算符，就与运算符栈的栈顶元素进行比较。

如果比运算符栈顶元素的优先级高，就将当前运算符压入栈；如果比运算符栈顶元素的优先级低或者相同，从运算符栈中取栈顶运算符，从操作数栈的栈顶取2个

操作数，然后进行计算，再把计算完的结果压入操作数栈，继续比较。

我将3+5*8-6这个表达式的计算过程画成了一张图，你可以结合图来理解我刚讲的计算过程。


3+5×8-6
①
②
③
④
⑤
⑥
⑦
最后清空栈，结果=37

这样用两个栈来解决的思路是不是非常巧妙？你有没有想到呢？

栈在括号匹配中的应用

除了用栈来实现表达式求值，我们还可以借助栈来检查表达式中的括号是否匹配。

我们同样简化一下背景。我们假设表达式中只包含三种括号，圆括号()、方括号[]和花括号{}，并且它们可以任意嵌套。比如，{[{}]}或[{()}([])]等都为合法格式，而{[]()]或[({)]为不合法的格式。那我现在给你一个包含三种括号的表达式字符串，如何检查它是否合法呢？

这里也可以用栈来解决。我们用栈来保存未匹配的左括号，从左到右依次扫描字符串。当扫描到左括号时，则将其压入栈中；当扫描到右括号时，从栈顶取出一个左括号。如果能够匹配，比如“(”跟“)”匹配，“[”跟“]”匹配，“{”跟“}”匹配，则继续扫描剩下的字符串。如果扫描的过程中，遇到不能配对的右括号，或者栈中没有数据，则说明为非法格式。

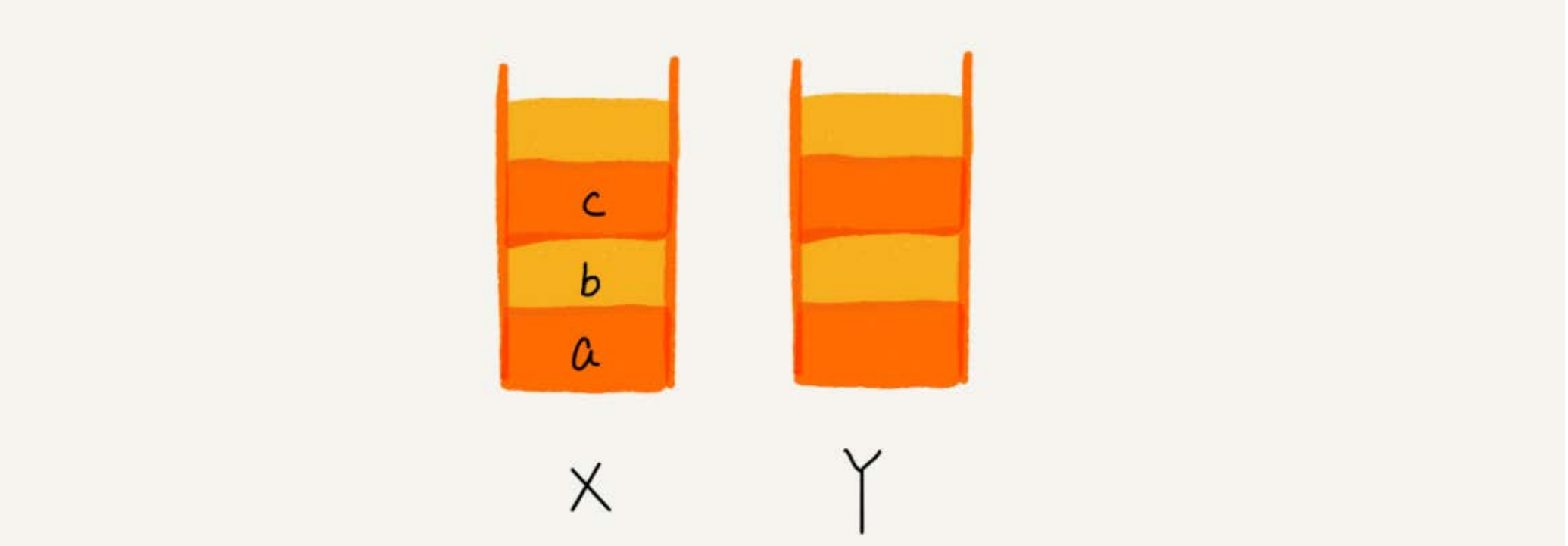
当所有的括号都扫描完成之后，如果栈为空，则说明字符串为合法格式；否则，说明有未匹配的左括号，为非法格式。

解答开篇

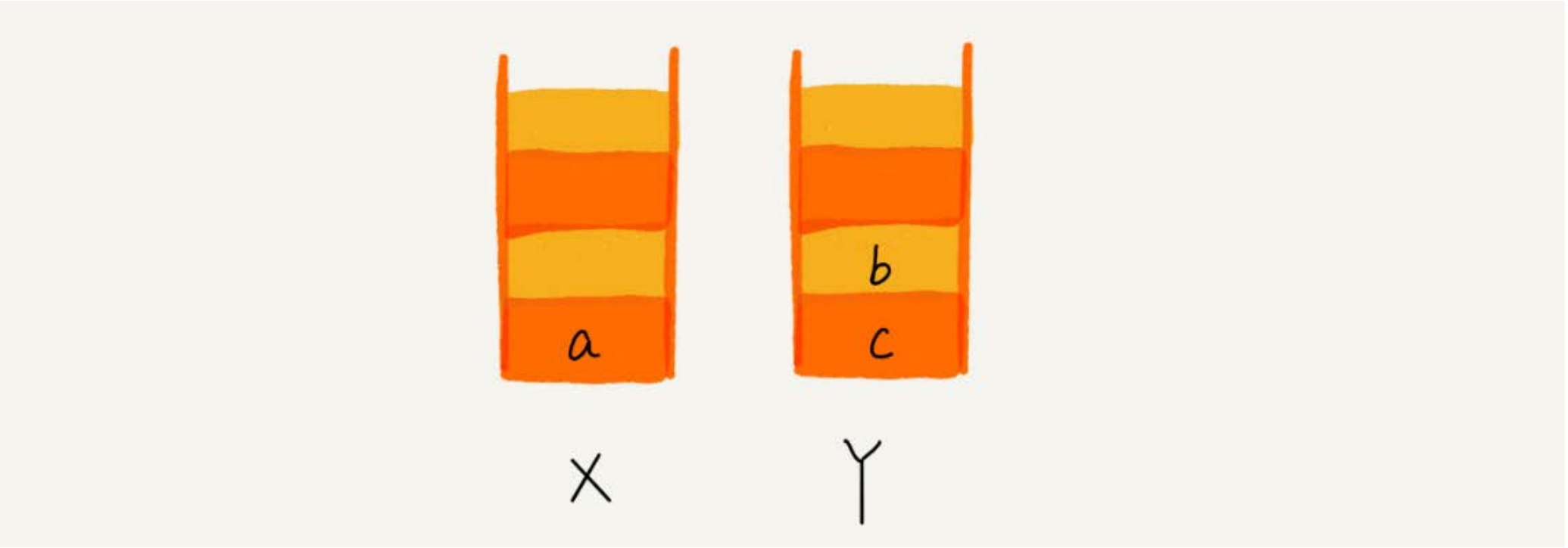
好了，我想现在你已经完全理解了栈的概念。我们再回来看看开篇的思考题，如何实现浏览器的前进、后退功能？其实，用两个栈就可以非常完美地解决这个问题。

我们使用两个栈，X和Y，我们把首次浏览的页面依次压入栈X，当点击后退按钮时，再依次从栈X中出栈，并将出栈的数据依次放入栈Y。当我们点击前进按钮时，我们依次从栈Y中取出数据，放入栈X中。当栈X中没有数据时，那就说明没有页面可以继续后退浏览了。当栈Y中没有数据，那就说明没有页面可以点击前进按钮浏览了。

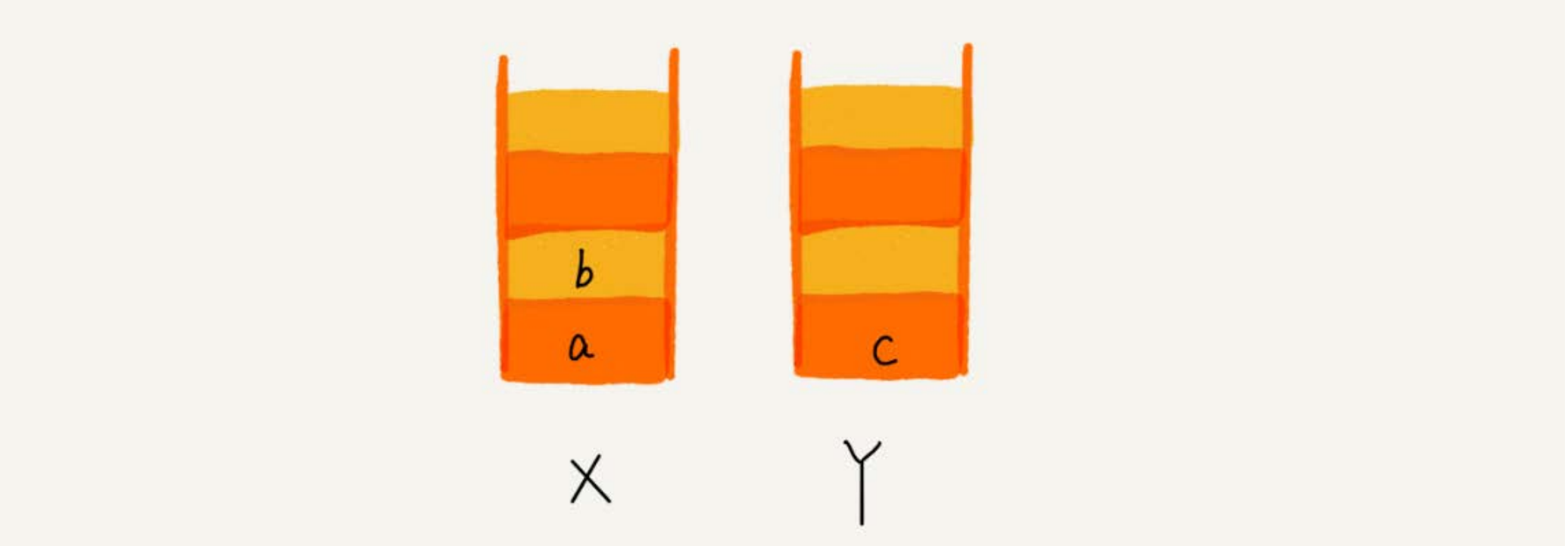
比如你顺序查看了a，b，c三个页面，我们就依次把a，b，c压入栈，这个时候，两个栈的数据就是这个样子：



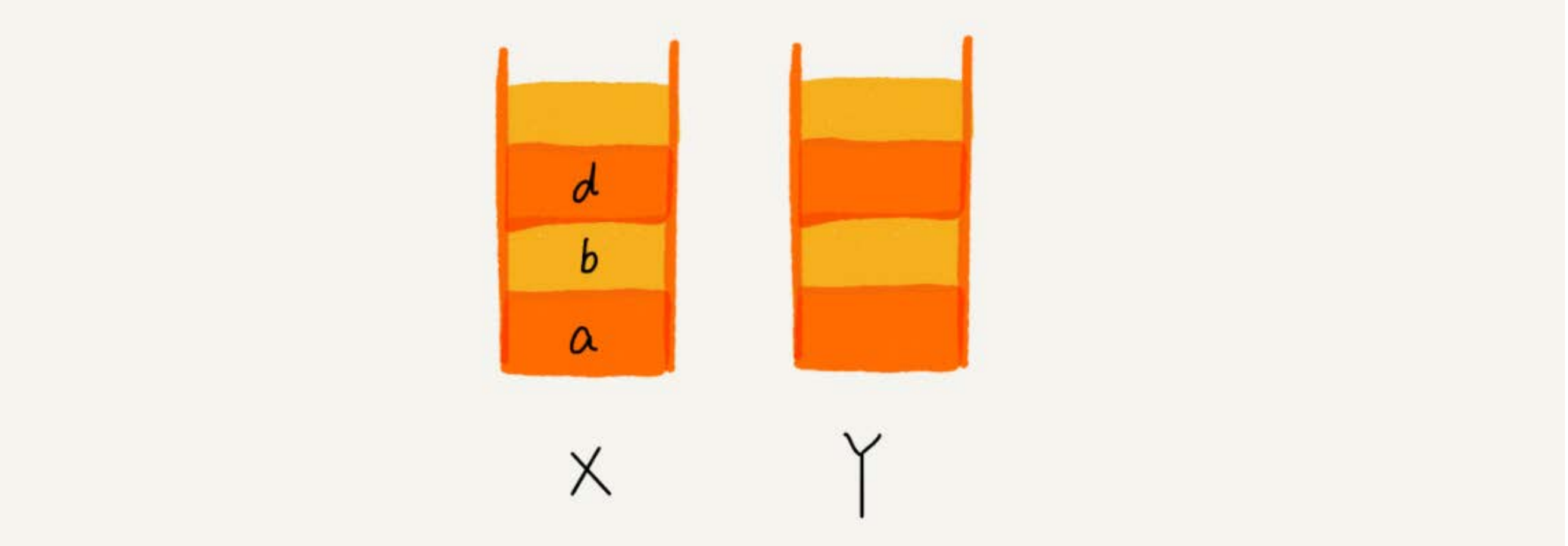
当你通过浏览器的后退按钮，从页面c后退到页面a之后，我们就依次把c和b从栈X中弹出，并且依次放入到栈Y。这个时候，两个栈的数据就是这个样子：



这个时候你又想看页面b，于是你又点击前进按钮回到b页面，我们就把b再从栈Y中出栈，放入栈X中。此时两个栈的数据是这个样子：



这个时候，你通过页面b又跳转到新的页面d了，页面c就无法再通过前进、后退按钮重复查看了，所以需要清空栈Y。此时两个栈的数据这个样子：



内容小结

我们来回顾一下今天讲的内容。栈是一种操作受限的数据结构，只支持入栈和出栈操作。后进先出是它最大的特点。栈既可以通过数组实现，也可以通过链表来实现。不管基于数组还是链表，入栈、出栈的时间复杂度都为O(1)。除此之外，我们还讲了一种支持动态扩容的顺序栈，你需要重点掌握它的均摊时间复杂度分析方法。

课后思考

1. 我们在讲栈的应用时，讲到用函数调用栈来保存临时变量，为什么函数调用要用“栈”来保存临时变量呢？用其他数据结构不行吗？
2. 我们都知道，JVM内存管理中有个“堆栈”的概念。栈内存用来存储局部变量和方法调用，堆内存用来存储Java中的对象。那JVM里面的“栈”跟我们这里说的“栈”是不是一回事呢？如果不是，那它为什么又叫作“栈”呢？

欢迎留言和我分享，我会第一时间给你反馈。

我已将本节内容相关的详细代码更新到GitHub，戳此即可查看。

精选留言：

- 阿杜S考特 2018-10-08 01:03:33
  内存中的堆栈和数据结构堆栈不是一个概念，可以说内存中的堆栈是真实存在的物理区，数据结构中的堆栈是抽象的数据存储结构。
  内存空间在逻辑上分为三部分：代码区、静态数据区和动态数据区，动态数据区又分为栈区和堆区。
  代码区：存储方法体的二进制代码。高级调度（作业调度）、中级调度（内存调度）、低级调度（进程调度）控制代码区执行代码的切换。

静态数据区：存储全局变量、静态变量、常量，常量包括final修饰的常量和String常量。系统自动分配和回收。
栈区：存储运行方法的形参、局部变量、返回值。由系统自动分配和回收。
堆区：new一个对象的引用或地址存储在栈区，指向该对象存储在堆区中的真实数据。

[60赞]

- gg 2018-11-01 03:24:40
  为什么函数调用要用“栈”来保存临时变量呢？用其他数据结构不行吗？

  其实，我们不一定非要用栈来保存临时变量，只不过如果这个函数调用符合后进先出的特性，用栈这种数据结构来实现，是最顺理成章的选择。

  从调用函数进入被调用函数，对于数据来说，变化的是什么呢？是作用域。所以根本上，只要能保证每进入一个新的函数，都是一个新的作用域就可以。而要实现这个，用栈就非常方便。在进入被调用函数的时候，分配一段栈空间给这个函数的变量，在函数结束的时候，将栈顶复位，正好回到调用函数的作用域内。 [21赞]

- 阿杜S考特 2018-10-08 01:03:35
  内存中的堆栈和数据结构堆栈不是一个概念，可以说内存中的堆栈是真实存在的物理区，数据结构中的堆栈是抽象的数据存储结构。
  内存空间在逻辑上分为三部分：代码区、静态数据区和动态数据区，动态数据区又分为栈区和堆区。
  代码区：存储方法体的二进制代码。高级调度（作业调度）、中级调度（内存调度）、低级调度（进程调度）控制代码区执行代码的切换。
  静态数据区：存储全局变量、静态变量、常量，常量包括final修饰的常量和String常量。系统自动分配和回收。
  栈区：存储运行方法的形参、局部变量、返回值。由系统自动分配和回收。
  堆区：new一个对象的引用或地址存储在栈区，指向该对象存储在堆区中的真实数据。

  [147赞]

- 观弈道人 2018-10-08 14:15:08
  感觉在留言区做笔记没多大意义，留言区还是提问问题或回答问题笔记合适，长篇累牍的笔记给谁看啊，占空间~~ [112赞]

  作者回复2018-10-09 01:58:11
  还是有很多同学看的 个人喜好吧 不必强求

- 姜威 2018-10-08 00:31:48
  一、什么是栈？
  1.后进者先出，先进者后出，这就是典型的“栈”结构。
  2.从栈的操作特性来看，是一种“操作受限”的线性表，只允许在端插入和删除数据。

二、为什么需要栈？

1.栈是一种操作受限的数据结构，其操作特性用数组和链表均可实现。

2.但，任何数据结构都是对特定应用场景的抽象，数组和链表虽然使用起来更加灵活，但却暴露了几乎所有的操作，难免会引发错误操作的风险。

3.所以，当某个数据集合只涉及在某端插入和删除数据，且满足后进者先出，先进者后出的操作特性时，我们应该首选栈这种数据结构。

三、如何实现栈？

1.栈的API

```
public class Stack<Item> {
//压栈
public void push(Item item){}
//弹栈
public Item pop(){}
//是否为空
public boolean isEmpty(){}
//栈中数据的数量
public int size(){}
//返回栈中最近添加的元素而不删除它
public Item peek(){}
}
```

2.数组实现（自动扩容）

时间复杂度分析：根据均摊复杂度的定义，可以得数组实现（自动扩容）符合大多数情况是O(1)级别复杂度，个别情况是O(n)级别复杂度，比如自动扩容时，会进行完整数据的拷贝。

空间复杂度分析：在入栈和出栈的过程中，只需要一两个临时变量存储空间，所以O(1)级别。我们说空间复杂度的时候，是指除了原本的数据存储空间外，算法运行还需要额外的存储空间。

实现代码：（见另一条留言）

3.链表实现

时间复杂度分析：压栈和弹栈的时间复杂度均为O(1)级别，因为只需更改单个节点的索引即可。

空间复杂度分析：在入栈和出栈的过程中，只需要一两个临时变量存储空间，所以O(1)级别。我们说空间复杂度的时候，是指除了原本的数据存储空间外，算法运行还需要额外的存储空间。

实现代码：（见另一条留言）

四、栈的应用

1.栈在函数调用中的应用

操作系统给每个线程分配了一块独立的内存空间，这块内存被组织成“栈”这种结构，用来存储函数调用时的临时变量。每进入一个函数，就会将其中的临时变量作为栈帧入栈，当被调用函数执行完成，返回之后，将这个函数对应的栈帧出栈。

2.栈在表达式求值中的应用（比如：34+13*9+44-12/3）

利用两个栈，其中一个用来保存操作数，另一个用来保存运算符。我们从左向右遍历表达式，当遇到数字，我们就直接压入操作数栈；当遇到运算符，就与运算符栈的栈顶元素进行比较，若比运算符栈顶元素优先级高，就将当前运算符压入栈，若比运算符栈顶元素的优先级低或者相同，从运算符栈中取出栈顶运算符，从操作数栈顶取出2个操作数，然后进行计算，把计算完的结果压入操作数栈，继续比较。

3.栈在括号匹配中的应用（比如：{}{[()]()}）

用栈保存为匹配的左括号，从左到右一次扫描字符串，当扫描到左括号时，则将其压入栈中；当扫描到右括号时，从栈顶取出一个左括号，如果能匹配上，则继续扫描剩下的字符串。如果扫描过程中，遇到不能配对的右括号，或者栈中没有数据，则说明为非法格式。

当所有的括号都扫描完成之后，如果栈为空，则说明字符串为合法格式；否则，说明未匹配的左括号为非法格式。

4.如何实现浏览器的前进后退功能？

我们使用两个栈X和Y，我们把首次浏览的页面依次压如栈X，当点击后退按钮时，再依次从栈X中出栈，并将出栈的数据一次放入Y栈。当点击前进按钮时，我们依次从栈Y中取出数据，放入栈X中。当栈X中没有数据时，说明没有页面可以继续后退浏览了。当Y栈没有数据，那就说明没有页面可以点击前进浏览了。

五、思考

1. 我们在讲栈的应用时，讲到用函数调用栈来保存临时变量，为什么函数调用要用“栈”来保存临时变量呢？用其他数据结构不行吗？

答：因为函数调用的执行顺序符合后进者先出，先进者后出的特点。比如函数中的局部变量的生命周期的长短是先定义的生命周期长，后定义的生命周期短；还有函数中调用函数也是这样，先开始执行的函数只有等到内部调用的其他函数执行完毕，该函数才能执行结束。

正是由于函数调用的这些特点，根据数据结构是特定应用场景的抽象的原则，我们优先考虑栈结构。

2.我们都知道，JVM 内存管理中有个“堆栈”的概念。栈内存用来存储局部变量和方法调用，堆内存用来存储 Java 中的对象。那 JVM 里面的“栈”跟我们这里说的“栈”是不是一回事呢？如果不是，那它为什么又叫作“栈”呢？

答：JVM里面的栈和我们这里说的是一回事，被称为方法栈。和前面函数调用的作用是一致的，用来存储方法中的局部变量。 [101赞]

- 他在她城断了弦 2018-10-15 02:07:11

  leetcode上关于栈的题目大家可以先做20,155,232,844,224,682,496. [89赞]

- 小洋洋 2018-10-08 00:03:28

  函数调用之所以用栈，是因为函数调用中经常嵌套，栗子：A调用B，B又调用C，那么就需要先把C执行完，结果赋值给B中的临时变量，B的执行结果再赋值给A的临时变量，嵌套越深的函数越需要被先执行，这样刚好符合栈的特点，因此每次遇到函数调用，只需要压栈，最后依次从栈顶弹出依次执行即可，这个过程很像文稿中的3+5*8-6//小白之拙见，欢迎拍砖*^o^* [60赞]

- 鲫鱼 2018-10-10 03:26:21

  对我来说理解有些困难，所以姜威的笔记给了我很大的帮助的。给了我更好完善笔记的构架，以及用不同方式解释加深理解和记忆。真的有的人不喜欢看不看就好，划掉不过两秒的事情。 [38赞]

- gg 2018-11-01 03:24:59

  为什么函数调用要用“栈”来保存临时变量呢？用其他数据结构不行吗？

  其实，我们不一定非要用栈来保存临时变量，只不过如果这个函数调用符合后进先出的特性，用栈这种数据结构来实现，是最顺理成章的选择。

  从调用函数进入被调用函数，对于数据来说，变化的是什么呢？是作用域。所以根本上，只要能保证每进入一个新的函数，都是一个新的作用域就可以。而要实现这个，用栈就非常方便。在进入被调用函数的时候，分配一段栈空间给这个函数的变量，在函数结束的时候，将栈顶复位，正好回到调用函数的作用域内。 [34赞]

  作者回复2018-11-02 02:09:48

  答案 大家可以参考下

- 姜威 2018-10-08 00:35:10

  实现代码：（栈的数组实现）

```
public class StackOfArray<Item> implements Iterable<Item>{
//存储数据的数组
Item[] a = (Item[])new Object[1];
//记录元素个数N
int N = 0;
//构造器
public StackOfArray(){}
//添加元素
public void push(Item item){
//自动扩容
if (N == a.length ) resize(2*a.length );
a[N++] = item;
}
//删除元素
public Item pop(){
Item item = a[--N];
a[N] = null;
if (N > 0 && N == a.length / 4) resize(a.length / 2);
return item;
}
```

```
//是否为空
public boolean isEmpty(){
return N == 0;
}
//元素个数
public int size(){
return N;
}
//改变数组容量
private void resize(int length) {
Item[] temp = (Item[])new Object[length];
for (int i = 0; i < N; i++) {
temp[i] = a[i];
}
a = temp;
}
//返回栈中最近添加的元素而不删除它
public Item peek(){
return a[N-1];
}
@Override
public Iterator<Item> iterator() {
return new ArrayIterator();
}
//内部类
class ArrayIterator implements Iterator{
//控制迭代数量
int i = N;
@Override
public boolean hasNext() {
return i > 0;
}
@Override
```

```
public Item next() {
return a[--i];
}
}
}
```

实现代码：（栈的链表实现）

```
public class StackOfLinked<Item> implements Iterable<Item> {
//定义一个内部类，就可以直接使用类型参数
private class Node{
Item item;
Node next;
}
private Node first;
private int N;
//构造器
public StackOfLinked(){}
//添加
public void push(Item item){
Node oldfirst = first;
first = new Node();
first.item = item;
first.next = oldfirst;
N++;
}
//删除
public Item pop(){
Item item = first.item;
first = first.next;
N--;
return item;
}
//是否为空
```

```
public boolean isEmpty(){
return N == 0;
}
//元素数量
public int size(){
return N;
}
//返回栈中最近添加的元素而不删除它
public Item peek(){
return first.item;
}
@Override
public Iterator<Item> iterator() {
return new LinkedIterator();
}
//内部类：迭代器
class LinkedIterator implements Iterator{
int i = N;
Node t = first;
@Override
public boolean hasNext() {
return i > 0;
}
@Override
public Item next() {
Item item = (Item) t.item;
t = t.next;
i--;
return item;
}
}
} [25赞]
```