# 基础

## 背包、队列和栈

**许多基础数据类型**都和对象的集合有关。

本节的第一个目标是说明对集合中的对象的表示方式将直接影响各种操作的效果。

本节的第二个目标是介绍**泛型**和**迭代**。

本节的第三个目标是介绍并说明链式数据结构的重要性，特别是经典数据结构**链表**。

### API

表格 .1泛型可迭代的基础数据类型的API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 背包 | | | | |
| public class Bag<Item> implements Iterable<Item> | | | | |
|  | Bag() | | 创建一个空背包 | |
| void | add(Item item) | | 添加一个元素 | |
| boolean | isEmpty() | | 背包是否为空 | |
| int | size() | | 背包中的元素数量 | |
| 先进先出（FIFO）队列 | | | | |
| public class Queue<Item> implements Iterable<Item> | | | | |
|  | | Queue() | | 创建一个空队列 |
| void | | enqueue(Item item) | | 添加一个元素 |
| Item | | dequeue() | | 删除最早添加的元素 |
| boolean | | isEmpty() | | 队列是否为空 |
| int | | size() | | 队列中元素的数量 |
| 下压（后进先出，LIFO）栈 | | | | |
| public class Stack<Item> implements Iterable<Item> | | | | |
|  | | Stack() | | 创建一个空栈 |
| void | | push(Item item) | | 添加一个元素 |
| Item | | pop() | | 删除最近添加的元素 |
| boolean | | isEmpty() | | 栈是否为空 |
| int | | size() | | 队列中元素的数量 |

#### 泛型

#### 自动装箱

#### 可迭代的集合类型

#### 背包

背包是一种你不支持从中删除元素的集合数据类型——它的目的就是帮助用例收集元素并迭代遍历所有收集的元素（用例也可以检查背包是否为空或者获取背包中元素的数量）。迭代的顺序不确定且与用例无关。

#### 先进先出队列

先进先出队列（或简称队列）是一种基于先进先出（FIFO）策略的集合类型。

#### 下压栈

下压栈（或简称栈）是一种基于后进先出（LIFO）策略的集合类型。

#### 算数表达式求职

表达式有括号、运算符和操作数（数字）组成。我们根据以下4种情况从左到右逐个将这些实体送入栈处理：

* 将操作数压入操作数栈；
* 将运算符压入运算符栈；
* 忽略左括号；
* 在遇到右括号时，弹出一个运算符，弹出所需数量的操作数，并将运算符和操作数的运算结果压入操作数栈。

**Dijkstra双栈算数表达式求值算法。**

### ****集合类数据类型的实现****

#### 定容栈

* 数组中的元素顺序和它们被插入的顺序相同；
* 当N为0时栈为空；
* 栈的顶部位于a[N-1]（如果栈非空）。

#### 泛型

创建泛型数组在Java中是不允许的。需要使用类型转化：

a=(Item[]) new Object[cap];

#### 调整数组大小

在push()中，检查数组是否太小。具体来说，通过检查站大小N和数组大小a.length是否相等来检查数组是否能够容纳新的元素。如果没有多余的空间，我们会将数组的长度**加倍**。

在pop()中，首先删除栈顶的元素，然后如果数组太大我们就将它的长度**减半**。正确的检测条件是栈大小是否小于数组大小的四分之一。在数组长度被减半之后，它的状态约为半满。

在这种实现中，栈永远不会溢出，使用率也永远不会低于四分之一（除非栈为空，那种情况下数组的大小为1）。

#### 对象游离

保存一个不需要的对象的引用称为游离。

#### 迭代

在任意可迭代的集合数据类型中都必须要实现的东西：

* 集合数据类型必须包含一个iterator()方法并返回一个Iterator对象；
* Iterator类必须包含连个方法：hasNext()方法（返回一个boolean值）和next()（返回集合中的一个泛型元素）。

使用foreach时先调用hasNext()，再调用next()方法。

请注意：**嵌套类可以访问包含它的类的实例变量。**

**基于数组的可调节大小栈算法十分重要，因为它几乎（但还没有）达到了任意集合类数据类型的实现的最佳性能：**

* 每项操作的用时都与集合大小无关；
* 空间需求总是不超过集合大小乘以一个常数。

### 链表

**定义**。**链表**是一种递归的数据结构，它或者为空（null），或者是指向一个结点（node）的引用，该结点含有一个泛型的元素和一个指向另一条链表的引用。

#### 结点记录

没有定义任何方法且会在代码中直接引用实例变量的类也被称为**记录**。

#### 构造链表

链表表示的一类元素。

#### 在表头插入结点

先将first保存在oldfirst中，然后将一个新结点赋予first，并将它得item域设置为插入的值，检查链表是否为空，如果为空，将维护链表大小的n加1结束（相当于默认给first的next域赋值null，如果维护last连接，将first赋予last）；否则将first的next域设为oldfirst，最后将维护链表大小的n加1。



#### 在表头删除结点

这个操作很简单：只需要将first指向first.next即可。

如果维护last链接，在删除first后，检查链表是否为空，如果为空，将last设为null。



#### 在表尾插入结点

我们需要一个指向链表最后一个结点的链接，因为该结点的链接必须被修改并指向一个含有新元素的新结点。我们不能再链表代码中草率的决定维护一个额外的链接，因为每个修改链接的操作都需要添加检查是否要修改该变量（以及做出相应修改）的代码。

例如：我们刚刚讨论过的删除链表首结点的代码就可能改变指向链表的尾结点的引用，因为当链表中只有一个结点时，它既是首结点又是尾结点！

先将last保存在oldlast中，然后将一个新结点赋予last，并将它得item域设置为插入的值，next域设为null，检查链表是否为空，如果为空，将维护链表大小的n加1结束（如果维护first连接，将last赋予first）；否则将oldlast的next域设为last，最后将维护链表大小的n加1。



#### 在其他位置的插入和删除操作

* 删除制定的结点；
* 在指定的结点前插入一个新结点。

这些功能没那么容易实现，例如，我们怎样才能删除链表的尾结点呢？last链接帮不上忙，因为我们需要将链接表尾结点的前一个结点中的链接（它指向的正式last）值改为null。在缺少其他信息的情况下，唯一的解决办法就是遍历整条链表并找出指向last的结点。这种解决方案并不是我们想要的，因为它所需要的时间和链表的长度成正比。实现任意插入和删除操作的标准解决方案是使用**双向链表**，其中每个结点都含有两个链接，分别指向不同的方向。

#### 遍历

for(Node x=first; x!=null; x=x.next)

{

处理x；

}

#### 栈的实现

链表的使用达到了我们的最优设计目标：

* 它可以处理任意类型的数据；
* 所需的空间总是和集合的大小成正比；
* 操作所需的时间总是和集合的大小无关。

#### 队列的实现

链表是数组的一种重要的替代方式。

#### 背包的实现

### 综述

数组和链表都非常基础，常常被称为顺序存储和链式存储。

表格 .2 基础数据结构

|  |  |  |
| --- | --- | --- |
| 数据结构 | 优点 | 缺点 |
| 数组 | 通过索引可以直接访问任意元素 | 在初始化时就需要知道元素的数量 |
| 链表 | 使用的空间大小和元素数量成正比 | 需要通过引用访问任意元素 |

表格 .3

|  |  |  |  |
| --- | --- | --- | --- |
| 数据结构 | 章节 | 抽象数据类型 | 数据表示 |
| 父链接树 | 1.5 | UnionFind | 整形数组 |
| 二分查找树 | 3.2、3.3 | BST | 含有两个链接的结点 |
| 字符串 | 5.1 | String | 数组、偏移量和长度 |
| 二叉堆 | 2.4 | PQ | 对象数组 |
| 散列表（拉链法） | 3.4 | SeparateChainHashSet | 链表数组 |
| 散列表（线性探测法） | 3.4 | LinearProbingHashSet | 两个对象数组 |
| 图的邻接链表 | 4.1、4.2 | Graph | Bag对象的数组 |
| 单词查找树 | 5.2 | TrieST | 含有链接数组的结点 |
| 三向单词查找树 | 2、5.3 | TST | 含有三个链接的结点 |

## 算法分析

### 科学方法

科学家用来理解自然世界的方法对于研究计算机程序的运行时间同样有效：

* 细致地**观察**真实世界的特点，通常还要有精确的测量；
* 根据观察结果提出**假设**模型；
* 根据模型**预测**未来的事件；
* 继续观察并**核实**预测的准确性；
* 如此反复直到确认预测和观察一致。

科学方法的一条关键原则是我们所涉及的实验必须是可重现的，这样他人也可以自己验证假设的真实性。所有的假设必须是可证伪的，这样我们才能确认某个假设是错误的（并需要修正）。

### 观察

我们对大多数程序的第一个定量观察就是计算性任务的困难程度可以用**问题的规模来衡量。一般来说，问题的规模可以是输入的大小或是某个命令行参数的值。根据直觉**