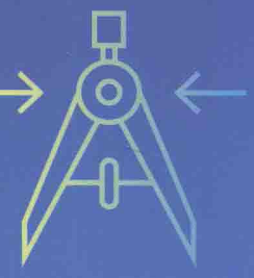
直击招聘

199道面试真题

直击招聘

程序员面试笔试算法设计深度解析



◎ 李春葆 李筱驰 编著

定位准确 面向企业应聘人才，面向编程技术提高者。

答疑解惑 解析相关课程中难点、疑点和热点，涉及目前各大网站上热门讨论的话题。

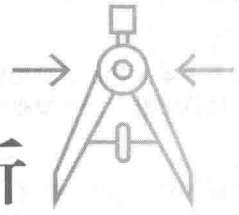
实战性强 收集近些年笔试和面试题目，涵盖常见考点。

清華大學出版社

直击招聘

直击招聘

程序员面试笔试
算法设计深度解析



◎ 李春葆 李筱驰 编著

清华大学出版社
北京

内 容 简 介

本书以常见的算法策略为中心，以 STL 为工具，汇总国内外众多著名 IT 企业近几年的算法面试笔试真题并予以透彻解析，从中提炼和归纳出求解问题的基本方法，并提供了一定数量的自测题以便于读者自我检验。

全书逻辑清晰、通俗易懂，适合参加 IT 企业校园招聘和面试笔试环节的同学复习使用，也适合算法设计编程爱好者与在校学生阅读和提高。

图书在版编目（CIP）数据

直击招聘：程序员面试笔试算法设计深度解析 / 李春葆，李筱驰编著. —北京：清华大学出版社，2018
（直击招聘）
ISBN 978-7-302-48796-8

Ⅰ. ①直… Ⅱ. ①李… ②李… Ⅲ. ①电子计算机-算法设计-资格考试-自学参考资料
Ⅳ. ①TP301.6

中国版本图书馆 CIP 数据核字（2017）第 272571 号

责任编辑：魏江江　王冰飞
封面设计：杨　兮
责任校对：时翠兰
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×240mm　　**印　张**：24.5　　**字　　数**：535 千字
版　　次：2018 年 7 月第 1 版　　**印　　次**：2018 年 7 月第 1 次印刷
印　　数：1～2000
定　　价：79.80 元

产品编号：077559-01

出版说明

国内外许多著名的IT企业都采用“企业网申→测评→笔试→面试→发放录用意向书”的招聘流程，如阿里巴巴校园招聘网站“https://campus.alibaba.com/process.htm”发布了详细的招聘信息，申报各种技术类岗位（如研发工程师、算法工程师、前端开发工程师和测试开发工程师等）的同学必须经过笔试和面试环节。

尽管不同IT公司的笔试和面试方式存在差异，但考查点基本相同，除了语言表达能力、自我控制能力、人际关系处理能力和生活情趣等非工作技能外，从专业角度看，无非就是如下几点：

❶ 基本知识点掌握情况。面试者至少需要熟练掌握一门计算机语言，常用的有C/C++、Java等，这是作为程序员的基本功。所谓“熟练掌握”，就是不仅限于基本语法，还要理解语言的实现与运行时情况，如C语言中的指针、变量存储类别和函数执行过程等。

❷ 基本程序设计情况。考查面试者是否具有编程能力，包含编写代码、调试代码和测试代码的整个过程，能够做到“让代码说话”，不能只会夸夸其谈，却不会编程。

❸ 算法设计能力。考查面试者求解问题的算法设计和实现能力，采用的算法策略是否合适，算法是否涵盖所有的测试用例。

❹ 计算逻辑思维能力。考查面试者是否具有一定的分析问题和解决问题的能力，用计算机求解问题时思维是否缜密，能否抓住问题的本质，采用的思路是否得当，条理是否清晰。

本丛书是面向参加笔试和面试的同学编写的，不仅对接大学计算机课程体系，而且结合IT企业招聘人才的基本行规，涵盖的内容如下：

- 程序员面试笔试C语言深度解析；
- 程序员面试笔试C++语言深度解析；
- 程序员面试笔试数据结构深度解析；
- 程序员面试笔试算法设计深度解析。

本丛书以 C/C++语言为工具、数据结构为基础、算法设计为目标，其中《直击招聘——程序员面试笔试 C 语言深度解析》和《直击招聘——程序员面试笔试 C++语言深度解析》侧重 C/C++语言的核心概念和深层次用法，《直击招聘——程序员面试笔试数据结构深度解析》侧重以常用数据结构为核心的算法设计，《直击招聘——程序员面试笔试算法设计深度解析》侧重通用的算法设计。本丛书具有如下特点。

❶ 定位准确。面向企业应聘人才，面向编程技术提高者。

❷ 答疑解惑。解析相关课程中的难点、疑点和热点，包括目前各大网站上热门讨论的话题。

❸ 实战性强。收集近些年的笔试和面试题目，涵盖常见考点。

前　言

算法是计算机科学的核心主题之一，有一个好的算法设计功底是从事 IT 工作的坚实基础，也是 IT 领域研究工作中必需的技能。传统的数据结构课程中的算法设计是围绕数据结构展开的，而算法设计课程主要讨论通用的算法设计策略，在面试笔试的编程中具有更好的实用性。

常用的通用算法设计策略有暴力法、回溯法、分枝限界法、分治法、动态规划和贪心法等，本书以此为主线，汇总国内外众多著名 IT 企业近几年的算法面试笔试真题并予以解析，透彻剖析了难点和疑点，突出了算法设计的思想和分析的基本原则，为面试者提供了清晰的、整体的思路和方法。

比起算法设计和分析课程，本书更侧重算法设计思路，而不是算法复杂度分析和算法正确性证明的赘述。另外，算法的实现采用 C++标准模板库（STL），STL 不仅功能强大、简洁和实用，提供的数据结构（如栈、队列、优先队列等）和算法（如排序等）效率高，而且使面试者有很多的时间专注算法策略的实现，而不用将大量时间浪费在程序细节上。面试题编号后面的星号个数表示本题的难度，其中一星难度最低，五星难度最高。

书中程序除了个别外其他都在 Visual C++ 6.0 环境下调试通过（采用较低版本编译器的目的是让代码具有更好的兼容性）。在本书的编写过程中参考了众多网站和博客，无法一一列出，编者在此表示衷心感谢。

限于编者水平，书中难免存在遗漏，恳请读者批评指正，编者的联系邮箱是 licb1964@126.com。

编　者

2018 年 3 月

目 录

·第1章·
算法设计库——STL

常见考点

- STL 的组成。
- STL 中各个容器的特点和应用。
- 迭代器类型、迭代器适配器及其应用。
- 各个通用算法的使用。
- 灵活应用 STL 的各种数据结构和算法解决一些较复杂的应用问题。

1.1 STL 要点归纳

1.1.1 STL 概述

在算法设计中经常用到某些数据结构，为了提高设计效率，应尽可能使用现成的数据结构来实现。C++的 STL（Standard Template Library，标准模板类库）就是一个非常好的算法设计库，其具有如下特点：

- STL 提供了常用的数据结构模板类和丰富的操作函数。
- STL 的所有组件都是用模板方式定义的，全面支持泛型，强调的是让数据的结构和算法独立于其类型。
- STL 所追求的是在尽量小的框架内实现最大的弹性。

STL 从广义上讲分为 3 类，即 container（容器）、algorithm（算法）和 iterator（迭代器），容器用于存放数据对象（元素），算法用于操作容器中的数据对象。尽管各种容器的内部结构各异，但其外部常常给人的感觉都是相似的，即将对容器数据的操作设计成通用算法，也就是将算法和容器分离开来。算法和容器之间的中介就是迭代器。容器、算法和迭代器称为 STL 的三大件，它们之间的关系如图 1.1 所示。

说明：在 VC++ 6.0 中使用 STL 时编译程序经常会出现大量的有关 STL 的警告信息，可以使用#pragma warning(disable:4786)预编译指令屏蔽这些警告信息。

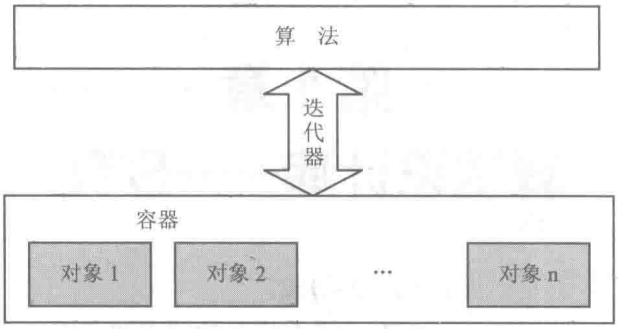


图 1.1 容器、算法和迭代器之间的关系

1.1.2 STL 中的常用容器

STL 容器是 STL 的核心部分，包含各种数据类型的数据结构，如链表、栈和队列等。STL 容器主要分为顺序容器、关联容器、适配器容器和哈希容器（其中，哈希容器不是 STL 标准容器）。表 1.1 列出了 STL 容器的类别、对应的数据结构和相应头文件。

表 1.1 数据结构和相应头文件的对应关系

容器类别	数 据 结 构	说 明	实现头文件
顺序容器	vector（向量）	连续存储元素	vector
	list（链表）	由结点组成的双向链表，每个结点包含一个元素	list
	deque（双端队列）	连续存储的指向不同元素的指针所组成的数组	deque
关联容器	set（集合）/multiset（多重集合）	由结点组成的红黑树，每个结点都包含一个元素。在 set 中没有相同关键字的元素，而 multiset 中允许存在两个相同关键字的元素	set
	map（映射）/multimap（多重映射）	由（关键字,值）对组成的集合。在 map 中没有相同关键字的元素，而 multimap 中允许存在两个相同关键字的元素	map
适配器容器	stack（栈）	后进先出的序列	stack
	queue（队列）	先进先出的序列	queue
	priority_queue（优先队列）	元素的出队次序由指定的某种谓词决定的一种队列	queue
哈希容器	hash_map（hash_multimap, hash_set, hash_multiset）	哈希表，VC++ 6.0 不支持	hash_map
	unordered_map	无序哈希表，替代 hash_map，仅在 VC++ 11 或者更高版本中支持	unordered_map

C++中引入命名空间的概念，在不同命名空间中可以存在相同名字的标识符。程序员可能在自己的程序中定义了 sort()函数，而 STL 中也有这样的算法，为了避免二者混淆和

冲突，STL 的 sort()以及其标识符都封装在命名空间 std 中。STL 的 sort()算法编译为 std::sort()，从而避免了名字冲突。为此，在使用 STL 时必须将下面的语句插入到源代码文件开头：

```
using namespace std;
```

这样直接把程序代码定位到 std 命名空间中。

1. vector（向量容器）

vector 相当于动态数组，它存储具有相同数据类型的一组元素，可以从尾部快速插入与删除元素，快速随机访问元素，但是在序列中间插入、删除元素较慢，因为需要移动插入或删除处后面的所有元素。

如果初始分配的空间不够，当超过空间大小时会重新分配更大的空间（通常按两倍大小扩展），此时需要进行大量的元素复制，从而增加了性能开销。

定义 vector 向量的几种方式如下：

```
vector<int> v1;                    //定义元素为 int 的向量 v1
vector<int> v2(10);                //指定向量 v2 的初始大小为 10 个 int 元素
vector<double> v3(10,1.23);        //指定 v3 的 10 个初始元素的初值为 1.23
vector<int> v4(a,a+5);             //用数组 a[0..4]共 5 个元素初始化 v4
```

vector 主要的成员函数如下。

- max_size()：vector 容器中能保存的最大元素个数。
- size()：当前 vector 容器中的实际元素个数。
- []：返回指定下标的元素，表明 vector 的元素可以随机访问。
- reserve(n)：为 vector 容器预分配 n 个元素的存储空间。
- capacity()：返回当前 vector 容器在重新进行内存分配以前所能容纳的元素个数。
- resize(n)：调整 vector 容器的容器大小，使其能容纳 n 个元素。
- empty()：判断 vector 是否为空。
- at()：返回指定位置的元素。
- front()：获取 vector 的第 1 个元素。
- back()：获取 vector 的最后一个元素。
- assign()：给 vector 中的元素赋值。
- insert()：它需要两个参数，一个是位置迭代器，一个是待插入的元素。该函数将元素插入到迭代器指定元素之前。
- push_back()：在 vector 的尾部添加一个元素。
- pop_back()：删除 vector 的尾部元素。
- erase()：删除某个 iterator 或者 iterator 区间指定的元素。

- clear()：删除 vector 中的所有元素。
- begin()：用于正向迭代，返回 vector 中第 1 个元素的位置。
- end()：用于正向迭代，返回 vector 中最后一个元素后面的一个位置。
- rbegin()：用于反向迭代，返回 vector 中最后一个元素的位置。
- rend()：用于反向迭代，返回 vector 中第 1 个元素前面的一个位置。

注意：resize()与 reserve()的说明如下。

❶ reserve()是为容器预留空间，但并不真正创建元素对象，在创建对象之前不能引用容器内的元素，因此当加入新的元素时需要用 push_back()/insert()函数。

❷ resize()是改变容器的大小，并且创建对象，因此在调用这个函数之后就可以引用容器内的对象了，故加入新的元素时用 operator[]操作符或者用迭代器来引用元素对象。

❸ reserve()函数需要一个参数指出要预留的容器的大小；resize()函数可以有两个参数，第 1 个参数是容器的新的大小，第 2 个参数是要加入容器中的新元素，如果这个参数被省略，那么就调用元素对象的默认构造函数。

❹ 采用 reserve()成员函数为 vector 预先分配需要的内存空间可以提高执行效率。

例如以下程序说明 vector 容器的应用：

```
#include <stdio.h>
#include <vector>
#include <algorithm>
using namespace std;
bool cmp(int &x,int &y)                    //指定排序比较运算符
{   return x>y; }
int main()
{   int a[10]={2,5,8,6,3,7,1,9,4,10};
    vector <int> vec(a,a+10);              //用 a 的 10 个元素初始化 vec
    printf("size:%d\n",vec.size());
    printf("正向输出 vec:");
    vector<int>::iterator it,it1;
    for(it=vec.begin();it!=vec.end();++it)
        printf("%d ",*it);                 //通过正向迭代器输出
    printf("\n");
    int x=6;
    it1=find(vec.begin(),vec.end(),x);
    if(it1!=vec.end())
        printf("查找到元素%d\n",*it1);
    else
        printf("没有找到元素%d\n",x);
```

```
    printf("递减排序\n");
    sort(vec.begin(),vec.end(),cmp);    //调用 sort 算法排序
    vector<int>::reverse_iterator rit;
    printf("反向输出 vec:");
    for(rit=vec.rbegin();rit!=vec.rend();++rit)
        printf("%d ",*rit);             //通过反向迭代器输出
    printf("\n");
    return 0;
}
```

上述程序的执行结果如图 1.2 所示。

图 1.2　程序的执行结果

⋙ 2. deque（双端队列容器）

它是一个双端队列类模板。双端队列容器可以从前面或后面快速插入与删除元素，并可以快速随机访问元素，但在中间插入和删除元素较慢。

deque 容器不像 vector 把所有的元素保存在一块连续的内存块，而是采用多个连续的存储块，即由一段一段的定量连续空间构成，随时可以增加一段新的空间并连接起来，因此 deque 没有 capacity 的概念。deque 通过一个中央管理器来管理这些空间并隐藏 deque 的这些底层细节，让用户感觉在使用一个整体连续的对象，并且提供了随机存取的接口。

deque 的空间的重新分配比 vector 快，在重新分配空间后原有的元素不需要全部复制。实际上，deque 是对 vector 和 list 优/缺点的结合，它是处于两者之间的一种容器。

定义 deque 双端队列对象的几种方式如下：

```
deque<int> dq1;                          //定义元素为 int 的双端队列 dq1
deque<int> dq2(10);                      //指定双端队列 dq2 的初始大小为 10 个 int 元素
deque<double> dq3(10,1.23);              //指定 dq3 的 10 个初始元素的初值为 1.23
deque<int> dq4(dq2.begin(),dq2.end()); //用 dq2 的所有元素初始化 dq4
```

deque 主要的成员函数如下。

- size()：返回双端队列中元素的个数。
- []：返回指定下标的元素。

- empty()：判断双端队列是否为空队列。
- at()：返回指定位置的元素。
- front()：获取队头元素。
- back()：获取队尾元素。
- push_front()：在队头插入元素。
- push_back()：在队尾插入元素。
- pop_front()：删除队头的一个元素。
- pop_back()：删除队尾的一个元素。
- erase()：删除指定位置或者区间的元素。
- clear()：删除所有的元素。
- begin()：用于正向迭代，返回 deque 中第 1 个元素的位置。
- end()：用于正向迭代，返回 deque 中最后一个元素后面的一个位置。
- rbegin()：用于反向迭代，返回 deque 中最后一个元素的位置。
- rend()：用于反向迭代，返回 deque 中第 1 个元素前面的一个位置。

例如有以下程序：

```
#include <stdio.h>
#include <deque>
using namespace std;
void Dispdq(deque<int> &dq)     //输出双端队列中的元素
{   deque<int>::iterator iter; //定义正向迭代器
    for(iter=dq.begin();iter!=dq.end();iter++)
        printf("%d ",*iter);
    printf("\n");
}
void main()
{   deque<int> dq;              //建立一个双端队列 dq
    dq.push_front(1);           //队头插入 1
    dq.push_back(2);            //队尾插入 2
    dq.push_front(3);           //队头插入 3
    dq.push_back(4);            //队尾插入 4
    Dispdq(dq);                 //从队头开始输出队列元素
    dq.pop_front();             //删除队头元素
    dq.pop_back();              //删除队尾元素
    Dispdq(dq);                 //从队头开始输出队列元素
}
```

上述程序的执行结果如图 1.3 所示。

图 1.3　程序的执行结果

3. list（链表容器）

它是一个双链表类模板，可以从任何地方快速插入与删除。它的每个元素间用指针链接。与 vector 相比，list 不支持随机访问（没有提供 operator[]运算符和 at()成员函数），为了访问链表容器中特定的元素，必须从第 1 个位置（表头）开始，随着指针从一个元素到下一个元素，直到找到满足条件的元素。

list 中插入元素的速度比 vector 快。由于 list 对每个元素单独分配空间，所以不存在空间不够、重新分配空间的情况。

定义 list 链表对象的几种方式如下：

```
list<int> l1;                    //定义元素为 int 的链表 l1
list<int> l2(10);                //指定链表 l2 的初始大小为 10 个 int 元素
list<double> l3(10,1.23);        //指定 l3 的 10 个初始元素的初值为 1.23
list<int> l4(a,a+5);             //用数组 a[0..4]共 5 个元素初始化 l4
```

list 的主要成员函数如下。

- size()：返回 list 中实际元素的个数。
- empty()：判断 list 是否为空。
- push_back()：在 list 表的尾部插入元素。
- pop_back()：删除 list 表尾元素。
- push_front()：在 list 表的头部插入元素。
- pop_front()：删除 list 表头元素。
- insert(pos,num)：在 pos 位置插入元素 num。
- insert(pos,n,num)：在 pos 位置插入 n 个元素 num。
- insert(pos,beg,end)：在 pos 位置插入区间为[beg,end)的元素。
- remove ()：删除 list 中所有指定值的元素。
- remove_if(cmp)：删除条件满足的元素。
- erase()：从 list 容器删除一个或几个元素。
- clear()：删除 list 中所有的元素。
- unique()：删除相邻的重复元素。
- reverse()：反转链表。

- sort()：对 list 容器中的元素排序。
- c1.merge(c2)：合并两个有序的链表并使之有序，重新放到 c1 中，且释放 c2。
- c1.merge(c2,comp)：合并两个有序的链表并使之按照自定义规则排序之后重新放到 c1 中，且释放 c2。
- c1.splice(c1.beg,c2)：将 c2 连接在 c1 的 beg 位置，并释放 c2。
- begin()：用于正向迭代，返回 list 中第 1 个元素的位置。
- end()：用于正向迭代，返回 list 中最后一个元素后面的一个位置。
- rbegin()：用于反向迭代，返回 list 中最后一个元素的位置。
- rend()：用于反向迭代，返回 list 中第 1 个元素前面的一个位置。

例如，以下程序通过 list.remove_if()函数删除链表中的部分元素，然后调用 unique()函数删除链表中相邻的重复元素：

```
#include <iostream>
#include <list>
using namespace std;
bool cmp(int n)                                    //自定义谓词
{   return n<3;  }
void Disp(list<int> &mylist)                       //输出 mylist 链表
{   list<int>::iterator it;
    for(it=mylist.begin();it!=mylist.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main()
{   int a[]={1,2,3,3,3,4,5,4,4,5};
    int n=sizeof(a)/sizeof(a[0]);
    list<int> mylist(a,a+n);
    mylist.remove_if(cmp);
    cout<<"remove_if:";Disp(mylist);    //输出:3 3 3 4 5 4 4 5
    mylist.unique();
    cout<<"unique: ";Disp(mylist);      //输出:3 4 5 4 5
    return 0;
}
```

说明：上述程序在 Dev C++ 5.11 中调试通过。

由于 STL 算法中的排序算法 sort()只能用于顺序容器，不能用于 list 容器，所以 list 提供了自己的 sort 成员函数。在使用 list.sort()时，默认情况下使用底层元素类型的小于运算