

C++ STL 开发技术导引

叶至军 编著

人民邮电出版社
北 京

图书在版编目 (CIP) 数据

C++ STL 开发技术导引 / 叶至军编著. —北京: 人民邮电出版社, 2007.7
ISBN 978-7-115-16148-2

. C... . 叶... . C 语言—程序设计 . TP312

中国版本图书馆 CIP 数据核字 (2007) 第 057471 号

内 容 提 要

C++ STL 标准模板库在数据结构和算法的实践领域发挥着重要的作用。本书共分 5 篇 26 章, 以 “C++ 编程技术→C++ STL 泛化技术基础→C++ STL 容器技术→C++ STL 算法技术→C++ STL 迭代器技术” 为线索具体展开, 通过大量的源码分析和应用实例, 详细介绍了 C++ STL 的技术原理和使用方法。

通过本书的学习, 读者不仅可以轻松掌握 C++ STL, 还可以从它的一流源代码中受益匪浅。本书可用作高等院校计算机及相关专业的教学参考书。也适合各层次的 C++ 开发人员和爱好者为锤炼自身的 C++ 基本功阅读使用。

C++ STL 开发技术导引

- ◆ 编 著 叶至军
责任编辑 汤 倩
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京密云春雷印刷厂印刷
新华书店总店北京发行所经销
- ◆ 开本: 787×1092 1/16
印张: 28.25
字数: 691 千字 2007 年 7 月第 1 版
印数: 1—4 000 册 2007 年 7 月北京第 1 次印刷

ISBN 978- 7-115-16148-2/TP

定价: 55.00 元 (附光盘)

读者服务热线: (010)67132692 印装质量热线: (010)67129223

前言

搜索网上一些比较著名的开源软件项目，可以发现几乎都是使用 C++ 进行开发的。从它们高品质的源码可以看出，C++ STL 标准模板库使用得非常广泛。颇负盛名的 3D 图形渲染引擎 OGRE，就很好地应用了 C++ STL 库提供的数据结构和算法。

为什么写本书

对于大多数人来说，他们最关心的是如何使用 C++ STL 标准模板库，而不是如何实现一个数据结构和算法的通用库，毕竟这会涉及到编译平台的一些语言细节问题。现在 C++ STL 已被纳入 C++ 的编程环境中，成为 C++ 标准库的一个重要组成部分。用户不必再像以往那样经常翻阅资料，只需寥寥数行代码的调用，即可实现常用的数据结构和算法。

本书主要内容

全书共分 5 篇 26 章，所采用的 SGI STL 库代码，可到 sgi 的网站（www.sgi.com）下载。

第一篇为“预备知识”，介绍 C++ 编程技术，包括实现类型泛化的模板技术和实现输入输出迭代器所需要的 IO 流技术。

第二篇为“C++ STL 泛化技术基础”，介绍 C++ STL 的基本设计轮廓，包括它的发展历程、编译配置、结构体系，以及容器、算法、迭代器、内存分配器、函数对象和各种 STL 概念的综合分析。

第三篇为“C++ STL 容器技术”，具体介绍各种容器的原理和应用，包括 vector 向量容器、deque 双端队列容器、list 双向链表容器、slist 单向链表容器、bit_vector 位向量容器、set 集合容器、multiset 多重集合容器、map 映照容器、multimap 多重映照容器、hash_set 哈希集合容器、hash_map 哈希映照容器、string 基本字符序列容器、stack 堆栈容器、queue 队列容器和 priority_queue 优先队列容器。

第四篇为“C++ STL 算法技术”，介绍七十多种高效算法，包括非变易算法、变易算法、排序算法和数值算法。

第五篇为“C++ STL 迭代器技术”，介绍输入输出流的迭代器和在已有迭代器之上构造出来的迭代器，包括输入流迭代器、输出流迭代器、向前插入迭代器、向后插入迭代器、插入迭代器、反向迭代器、反向双向迭代器和原始存储迭代器。

为便于阅读，书中删去了源代码的命名标志符中的下划线“ ”。附录中则给出了 SGI STL 的版权信息。

本书主要由叶至军编写，参与编写的人员还有欧阳国、孙德俊、郭景春、苏骞、叶至力、李德扬和叶青峰等。另外，感谢焦焦、叶敦介、符译尹、吴嘉颖、郭浩宇和符传学等人对本书的顺利完稿给予的大力支持和帮助。在编写过程中，我们力求精益求精，但书中难免存在疏漏和不足之处，敬请批评指正。如果读者使用本书时遇到困难或问题，请发 E-mail 到 tangqian@ptpress.cn 与我们联系。

编 者

光盘使用说明

为了方便读者学习，本书附带一张光盘，提供全书所有例程的工程文件。下面对光盘内容和使用方法进行简要的介绍。

一、光盘的运行环境

硬件环境：一般的 PC 机，对 CPU 和内存不需特别要求。

软件环境：操作系统为 Windows2000/XP，集成开发平台为 Visual C++ 6.0 或以上。前 9 章例程直接采用 VC++ 内置的 P.J.Plauger STL 进行编译，余下各章使用 STLport 进行编译，具体的下载和配置说明请参见第 9 章的内容。

二、光盘目录

如图 1 所示，C++ STL Source 是光盘的根目录，内有 01 ~ 26 的子目录。其下是各章的例程目录，如图 2 所示，为第 7 章的各个例程目录。

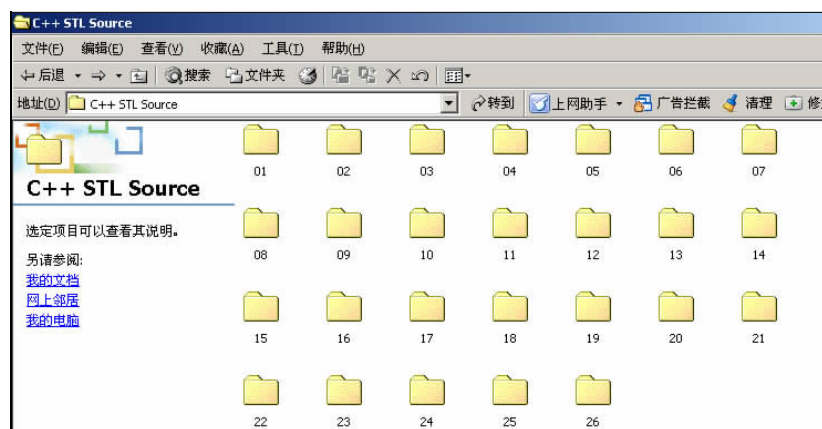


图 1 光盘的根目录

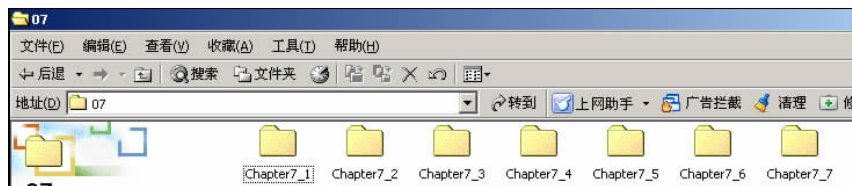


图 2 例程的工程目录

三、光盘的使用方法

将光盘的内容拷贝到硬盘，去除文件的只读属性。打开各例程的 Visual C++工程文件，就可进行程序的运行调试。

目 录

第一篇 预 备 知 识

第 1 章 C++ 编程技术	2
1.1 类和对象	2
1.2 类的继承	5
1.3 函数重载	5
1.4 访问控制	7
1.5 操作符重载	8
1.6 显式类型转换	9
1.7 异常处理	13
1.8 名字空间	17
1.9 友员函数	20
1.10 内联函数	21
1.11 静态成员	22
1.12 本章小结	23
第 2 章 C++ 模板技术	25
2.1 函数模板	25
2.2 类模板	27
2.3 模板完全特化	28
2.4 函数模板重载	30
2.5 类模板继承	30
2.6 本章小结	31
第 3 章 C++ I/O 流技术	32
3.1 I/O 流类	32
3.2 标准输入输出	34

3.3 文件输入输出	36
3.4 流的格式控制	41
3.5 本章小结	45

第二篇 C++ STL 泛化技术基础

第4章 C++ STL 泛型库概述	48
4.1 C++ STL 的发展历程	48
4.2 C++ STL 的各种实现版本	49
4.2.1 HP STL	49
4.2.2 SGI STL	50
4.2.3 STLport	50
4.2.4 P.J.Plauger STL	50
4.2.5 Rouge Wave STL	50
4.3 C++ STL 的 Visual C++ 编译	50
4.4 C++ STL 的体系结构	52
4.4.1 容器 (Container)	52
4.4.2 迭代器 (Iterator)	53
4.4.3 算法 (Algorithm)	53
4.4.4 函数对象 (Function Object)	54
4.4.5 适配器 (Adapter)	55
4.4.6 内存分配器 (Allocator)	56
4.4.7 概念 (Concept) 和模型 (Model)	56
4.5 C++ STL 存在的一些问题	57
4.6 本章小结	57
第5章 C++ STL 泛化技术分析	58
5.1 算法和迭代器	58
5.1.1 算法	58
5.1.2 迭代器	61
5.1.3 函数对象	65
5.1.4 适配器	68
5.2 内存分配器和容器	74
5.2.1 内存分配器	75
5.2.2 容器	77
5.3 概念	82
5.3.1 基础性概念	82
5.3.2 容器概念	84
5.3.3 迭代器概念	86
5.3.4 函数对象概念	88
5.4 本章小结	89

第三篇 C++ STL 容器技术

第6章 vector 向量容器	92
-----------------------	----

6.1	vector 技术原理	92
6.2	vector 应用基础	94
6.3	本章小结	101
第 7 章	deque 双端队列容器	102
7.1	deque 技术原理	102
7.2	deque 应用基础	108
7.3	本章小结	115
第 8 章	list 双向链表容器	116
8.1	list 技术原理	116
8.2	list 应用基础	124
8.3	本章小结	131
第 9 章	slist 单向链表容器	132
9.1	slist 技术原理	132
9.2	slist 应用基础	140
9.3	本章小结	148
第 10 章	bit_vector 位向量容器	149
10.1	bit_vector 技术原理	149
10.2	bit_vector 应用基础	156
10.3	本章小结	161
第 11 章	set 集合容器	162
11.1	set 技术原理	162
11.2	set 应用基础	181
11.3	本章小结	186
第 12 章	multiset 多重集合容器	187
12.1	multiset 技术原理	187
12.2	multiset 应用基础	190
12.3	本章小结	196
第 13 章	map 映照容器	197
13.1	map 技术原理	197
13.2	map 应用基础	200
13.3	本章小结	206
第 14 章	multimap 多重映照容器	207
14.1	multimap 技术原理	207
14.2	multimap 应用基础	210
14.3	本章小结	216

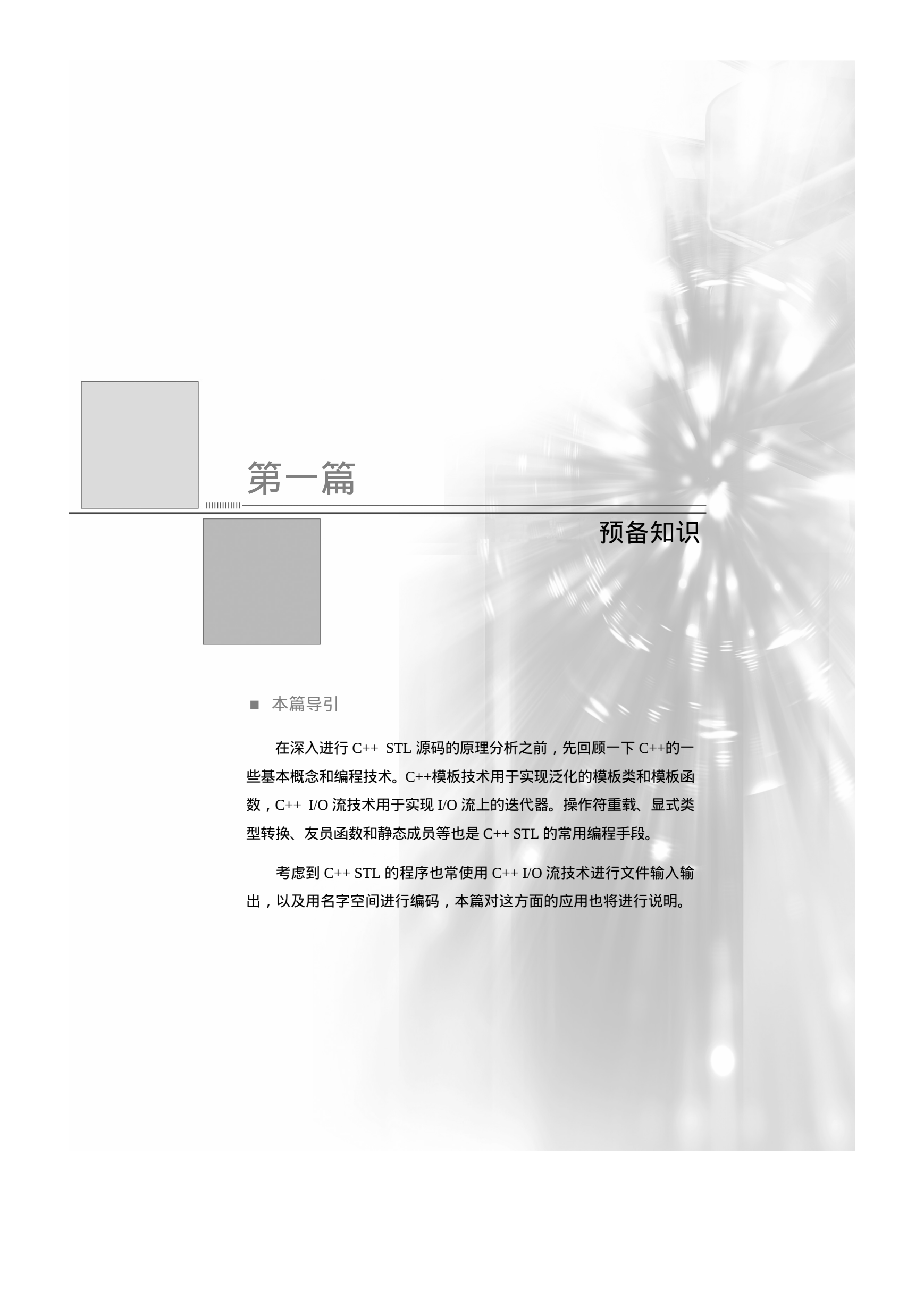
第 15 章	hash_set 哈希集合容器	217
15.1	hash_set 技术原理	217
15.2	hash_set 应用基础	230
15.3	本章小结	234
第 16 章	hash_map 哈希映照容器	235
16.1	hash_map 技术原理	235
16.2	hash_map 应用基础	237
16.3	本章小结	242
第 17 章	string 基本字符序列容器	243
17.1	string 技术原理	243
17.2	string 应用基础	258
17.3	本章小结	264
第 18 章	stack 堆栈容器	265
18.1	stack 技术原理	265
18.2	stack 应用基础	266
18.3	本章小结	269
第 19 章	queue 队列容器	270
19.1	queue 技术原理	270
19.2	queue 应用基础	271
19.3	本章小结	274
第 20 章	priority_queue 优先队列容器	275
20.1	priority_queue 技术原理	275
20.2	priority_queue 应用基础	278
20.3	本章小结	281

第四篇 C++ STL 算法技术

第 21 章	非变易算法	284
21.1	逐个容器元素 for_each	284
21.2	查找容器元素 find	285
21.3	条件查找容器元素 find_if	286
21.4	邻近查找容器元素 adjacent_find	287
21.5	范围查找容器元素 find_first_of	289
21.6	统计等于某值的容器元素个数 count	290
21.7	条件统计容器元素个数 count_if	291
21.8	元素不匹配查找 mismatch	293
21.9	元素相等判断 equal	295
21.10	子序列搜索 search	296
21.11	重复元素子序列搜索 search_n	299
21.12	最后一个子序列搜索 find_end	301

21.13	本章小结	303
第 22 章	变易算法	304
22.1	元素复制 copy	304
22.2	反向复制 copy_backward	305
22.3	元素交换 swap	306
22.4	迭代器交换 iter_swap	307
22.5	区间元素交换 swap_ranges	308
22.6	元素变换 transform	309
22.7	替换	310
22.8	条件替换 replace_if	311
22.9	替换和复制 replace_copy	312
22.10	条件替换和复制 replace_copy_if	313
22.11	填充 fill	314
22.12	n 次填充 fill_n	315
22.13	随机生成元素 generate	316
22.14	随机生成 n 个元素 generate_n	317
22.15	移除复制 remove_copy	318
22.16	条件移除复制 remove_copy_if	319
22.17	移除 remove	320
22.18	条件移除 remove_if	321
22.19	不连续重复元素复制 unique_copy	322
22.20	剔除连续重复元素 unique	324
22.21	元素反向 reverse	325
22.22	反向复制 reverse_copy	326
22.23	旋转 rotate	327
22.24	旋转复制 rotate_copy	329
22.25	随机抖动 random_shuffle	330
22.26	随机采样 random_sample	331
22.27	容器分割 partition	333
22.28	容器稳定分割 stable_partition	335
22.29	本章小结	338
第 23 章	排序算法	339
23.1	元素入堆 push_heap	339
23.2	创建堆 make_heap	343
23.3	元素出堆 pop_heap	348
23.4	堆排序 sort_heap	351
23.5	是否为堆 is_heap	352
23.6	局部排序 partial_sort	354
23.7	局部排序复制 partial_sort_copy	356
23.8	排序 sort	359
23.9	归并 merge	366
23.10	内部归并 inplace_merge	368
23.11	稳定排序 stable_sort	376

23.12	是否排序 <code>is_sorted</code>	383
23.13	第 n 个元素 <code>nth_element</code>	384
23.14	下确界 <code>lower_bound</code>	386
23.15	上确界 <code>upper_bound</code>	388
23.16	等价区间 <code>equal_range</code>	390
23.17	折半搜索 <code>binary_search</code>	392
23.18	子集合 <code>includes</code>	393
23.19	集合求并 <code>set_union</code>	394
23.20	集合求交 <code>set_intersection</code>	396
23.21	集合求差 <code>set_difference</code>	398
23.22	集合求异 <code>set_symmetric_difference</code>	399
23.23	最小值 <code>min</code>	401
23.24	最大值 <code>max</code>	402
23.25	最小元素 <code>min_element</code>	403
23.26	最大元素 <code>max_element</code>	404
23.27	字典比较 <code>lexicographical_compare</code>	405
23.28	下一排列组合 <code>next_permutation</code>	406
23.29	上一排列组合 <code>prev_permutation</code>	409
23.30	本章小结	411
第 24 章	数值算法	412
24.1	递增赋值 <code>iota</code>	412
24.2	元素求和 <code>accumulate</code>	413
24.3	两序列元素内积 <code>inner_product</code>	414
24.4	部分元素求和 <code>partial_sum</code>	415
24.5	相邻元素求差 <code>adjacent_difference</code>	417
24.6	n 次方计算 <code>power</code>	419
24.7	本章小结	421
第五篇 C++ STL 迭代器技术		
第 25 章	输入输出流迭代器	424
25.1	输入流迭代器	424
25.2	输出流迭代器	426
25.3	本章小结	427
第 26 章	插入/反向/存储迭代器	428
26.1	向前插入迭代器	428
26.2	向后插入迭代器	429
26.3	插入迭代器	431
26.4	反向迭代器	432
26.5	反向双向迭代器	434
26.6	原始存储迭代器	435
26.7	本章小结	437
附录	STL 版权说明	438



第一篇

预备知识

■ 本篇导引

在深入进行 C++ STL 源码的原理分析之前，先回顾一下 C++ 的一些基本概念和编程技术。C++ 模板技术用于实现泛化的模板类和模板函数，C++ I/O 流技术用于实现 I/O 流上的迭代器。操作符重载、显式类型转换、友员函数和静态成员等也是 C++ STL 的常用编程手段。

考虑到 C++ STL 的程序也常使用 C++ I/O 流技术进行文件输入输出，以及用名字空间进行编码，本篇对这方面的应用也将进行说明。

第 1 章

C++编程技术

C++语言由 Bjarne Stroustrup 开发，兼容于 C 语言，以面向对象方式，提供更直观的程序设计方法、更强大的语言开发能力、更严谨的安全管理和更有效的程序代码组织机制。

本章主要介绍 C++ STL 源码中常用的一些 C++编程技术，包括类、继承、操作符重载、访问控制、显式类型转换操作、异常处理、名字空间、友员函数、内联函数和静态成员等内容。

1.1 类和对象

C++是 C 的一个超集，最大特点是提供面向对象的 C++标准库，支持面向对象编程。面向对象的程序设计 (Object-Oriented Programming)，采用一种更自然、更直观、更艺术化的方式看待和编写程序。程序的设计过程如同在现实世界中有秩序地应用各种物质材料，去实施一项既定任务，例如打开空调机的电源，用遥控器设定温度高低，空调机就会把房间温度降低到预定值。

举例来说，我们可以深入到执行码的生成去理解它的意义。在这样的编程观之下，程序设计就是编程者与编译器对话的过程，让编译器生成二进制的指令和数据——分配 3 个 int 型存储空间 a、b 和 c，把整数 3、8 和 10 分别存入，作为函数 f、g 和 h 的参数进行调用。

```
int a=3;
int b=8;
int c=10;
f(a, b);
g();
h(c);
```

以上观点可算是把程序设计看作是一门科学对待，其实也可以从艺术的角度去理解它——做出 3 个 int 类型的物质材料 a、b 和 c，然后再把整数 3、8 和 10 放到这 3 种材料里面，再



让精灵 f 做个动作（需补给材料 a 和 b），让精灵 g 做个动作，最后让精灵 h 做个动作（需补给材料 c）。这样整个的程序编写，都是程序员自己在布置和运用各种材料（包括可动作的精灵），然后按照一定的语法把做事过程写出来，比起对话编程模式，少了许多拐弯抹角之处，思考问题更直接、更符合现实的思维方式。

从感觉上来说，上面的存储材料和函数精灵还是有一定的区别。在面向对象语言中，两者在形式上可统一起来对待，因为在一个类型中，不仅可定义充当材料的变量，也可定义充当精灵的函数，从而神秘的函数精灵被划入到具体的材料中，成为该材料的一个能动性。

假设在类型 someType 中定义了变量 a、b、c 和函数 f、g、h，someObject 是 someType 类型的一个材料，someObject.f(a, b) 语句就是用材料 someObject 的 f 能动特性做一下事情（需供给材料 a 和 b），例如让汽车的喇叭响一下，又或者让汽车的方向盘左转一下，以免碰着右边路上的行人。

```
double d;    //做出材料 d
d=13.6;      //把 13.6 放入材料 d
char e;      //做出材料 e
e='y';       //把 'y' 放入材料 e
int i;       //做出材料 i
i=60;        //把 60 放入材料 i
someType someObject(); //做出材料 someObject
someObject.a=3;      //把 3 放入构成材料 a 中
someObject.b=8;      //把 8 放入构成材料 b 中
someObject.c=10;     //把 10 放入构成材料 c 中
someObject.f(a, b);  //用材料 someObject 的 f 做一下
someObject.g();      //用材料 someObject 的 g 做一下
someObject.h(c);     //用材料 someObject 的 h 做一下
```

从上面代码的注释可以看到，面向对象的编程过程，犹如在现实物质世界中运用各种可见、可触摸的材料，完成一件工作任务。把这个处理过程书写出来，就可获得程序代码，当然必须遵循一定规则才行，正如其他文字作品一样，需要按照相应的语法写出来。

编程中可使用的材料，也就是面向对象语言所谓的对象，是通过一个类型或者一个类，用类似传统的“类型 变量”形式构建出来。面向对象语言不仅提供了大量内建类型和类，还可用已有的类型或类，去定义更多的类型和类。

下面是 C++ 定义一个类的基本语法，其中关键字 class 说明是一个类的定义，ClassName 是类名，public、private 和 protected 是访问控制符。

```
class ClassName{
public:
    公有数据和函数；
protected:
    保护数据和函数；
private:
    私有数据和函数；
};
```

在类的定义中，一般都要求提供一个 public 的构造函数和析构函数。构造函数要求与类同名，可带参数，但不能有函数返回值，一个类中可定义多个重载的构造函数。析构函数的名字由波浪符号“~”开头，名字的其余部分与类名相同，不能带任何参数，也不能有返回值，一个类只能定义一个析构函数。缺省情形下，由编译器提供一个空的构造函数和析构函数。

创建对象有如下的一般形式，构造函数利用各个参数值来初始化类中定义的变量。用

delete 语句删除对象时，会调用析构函数进行资源释放处理。

```
ClassName object(参数值 1, 参数值 2, ..., 参数值 n); //创建对象
```

一般将类的定义放在.h 文件中，而具体的函数实现代码则放在.cpp 文件（c plus plus），如下面 CCompute 类所示，头文件 Comput.h 存放类的定义，实现文件 Compute.cpp 用来编写函数代码，各函数需用类名前缀“CCompute::”引出，表明是该类的成员函数。

```
//Compute.h 文件
class CCompute{
protected:
    int i;
    int j;
public:
    CCompute(int a, int b){
        i=a;
        j=b;
    }
    ~CCompute(){}
    int sum();
    int minus();
};
```

```
//Compute.cpp 文件
#include "Compute.h"
int CCompute::sum(){
    return i+j;
}

int CCompute::minus(){
    return i-j;
}
```

下面在 main.cpp 文件中，编写一个程序调用 CCompute 类，计算两数之和与两数之差，其中创建对象的“CCompute compute Obj(3, 8);”语句会调用构造函数，将对象内部的 i 和 j 初始化为 3 和 8。

```
//main.cpp 文件
#include "Compute.h"
#include <stdio.h>
int main(void){ //C++可省去 void 不写
    CCompute computeObj(3, 8);
    printf("3 和 8 之和等于%d\n", computeObj.sum());
    printf("3 和 8 之差等于%d\n", computeObj.minus());
    return 0;
}
```

也可以使用 new 操作符创建对象，返回对象的地址，然后用指针去调用对象的函数，如下面代码所示。

```
CCompute *pComputeObj=new CCompute(3, 8);
printf("3 和 8 之和等于%d\n", pComputeObj->sum());
printf("3 和 8 之差等于%d\n", pComputeObj->minus());
delete pComputeObj; //delete 与 new 配对，删除对象（调用析构函数释放资源）
```

由此可见，类与传统 C 模块的编写基本类似，可把类看作是对 C 模块的一个封装，把一



系列功能接近的函数写入一个类中，以便于代码的管理和维护。

最后需要说明的是，C++扩充了C结构体的定义，在C++结构体中，可像类一样定义数据和函数，而且可说明它们的访问属性，因此，struct也可用作定义类的关键字。同样地，C++也扩充了联合体union的定义，联合体中各个数据域依然占据同一块空间，但可在联合体中定义成员函数。

1.2 类的继承

C++提供继承的机制，使一个类（继承类）可在另一个类（基类）的基础上，增设变量和函数，从而使继承类不仅拥有基类的变量和函数，也拥有新定义的变量和函数，实现了基类的一个扩充。类的继承的一般形式如下。

```
class DirevedClassName : public BaseClass1Name1, protected BaseClassName2,
                        private BaseClassName3{
    // 新定义的变量;
    // 新定义的函数;
};
```

例如，上面 CCompute 类需要添加一个乘法函数，可用继承方式进行添加，实现一个继承于 CCompute 的继承类 CCompute2。

```
//CCompute2 类的头文件 Compute2.h
#include "Compute.h"
class CCompute2:public CCompute{
public:
    CCompute2(int a, int b):CCompute(a,b){} //调用基类的构造函数
    int mul(); //添加的乘法函数
};
//CCompute2 类的实现文件 Compute2.cpp
#include "Compute2.h"
int CCompute2::mul(){
    return i * j;
}
```

这样，CCompute2 类不仅具有基类 CComopute 的变量 i、j 和函数 sum、minus，还具有新增加的 mul 乘法函数。以下是使用 CCompute2 类的主程序，计算整数 6 和 5 的乘积。

```
#include "Compute2.h"
#include <stdio.h>
int main(void){
    CCompute2 compute2obj(6, 5);
    printf("6 和 5 的乘积等于%d", compute2obj.mul());
    return 0;
}
```

1.3 函数重载

C++允许对函数或类的成员函数进行重载。重载的函数都具有相同的函数名，但要求参数的类型或个数必须不同，以使编译器确切知道所调用的函数是哪一个重载函数。