

## <<深度探索C++对象模型>>

### 图书基本信息

书名：<<深度探索C++对象模型>>

13位ISBN编号：9787121149528

10位ISBN编号：7121149524

出版时间：2012-1

出版时间：电子工业出版社

作者：斯坦利·B.李普曼

页数：356

译者：侯捷

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## <<深度探索C++对象模型>>

### 内容概要

作者Lippman参与设计了全世界第一套C++编译程序cfront，这本书就是一位伟大的C++编译程序设计者向你阐述他如何处理各种explicit（明确出现于C++程序代码中）和implicit（隐藏于程序代码背后）的C++语意。

本书专注于C++面向对象程序设计的底层机制，包括结构式语意、临时性对象的生成、封装、继承，以及虚拟——虚拟函数和虚拟继承。

这本书让你知道：一旦你能够了解底层实现模型，你的程序代码将获得多么大的效率。

Lippman澄清了那些关于C++额外负荷与复杂度的各种错误信息和迷思，但也指出其中某些成本和利益交换确实存在。

他阐述了各式各样的实现模型，指出它们的进化之道及其本质因素。

书中涵盖了C++对象模型的语意暗示，并指出这个模型是如何影响你的程序的。

作者Lippman参与设计了全世界第一套C++编译程序cfront，这本书就是一位伟大的C++编译程序设计者向你阐述他如何处理各种explicit（明确出现于C++程序代码中）和implicit（隐藏于程序代码背后）的C++语意。

本书专注于C++面向对象程序设计的底层机制，包括结构式语意、临时性对象的生成、封装、继承，以及虚拟——虚拟函数和虚拟继承。

这本书让你知道：一旦你能够了解底层实现模型，你的程序代码将获得多么大的效率。

Lippman澄清了那些关于C++额外负荷与复杂度的各种错误信息和迷思，但也指出其中某些成本和利益交换确实存在。

他阐述了各式各样的实现模型，指出它们的进化之道及其本质因素。

书中涵盖了C++对象模型的语意暗示，并指出这个模型是如何影响你的程序的。

对于C++底层机制感兴趣的读者，这必然是一本让你大呼过瘾的绝妙好书。

<<深度探索C++对象模型>>

作者简介

作者:(美)Lippman

## &lt;&lt;深度探索C++对象模型&gt;&gt;

## 书籍目录

## 目 录

本立道生 (侯捷 译序) III

目录 VII

前言 (Stanley B. Lippman) XIII

第0章 导读 (译者的话) XXV

第1章 关于对象 (Object Lessons)

加上封装后的布局成本 (Layout Costs for Adding Encapsulation)

1.1 C++对象模式 (The C++ Object Model)

简单对象模型 (A Simple Object Model)

表格驱动对象模型 (A Table-driven Object Model)

C++对象模型 (The C++ Object Model)

对象模型如何影响程序 (How the Object Model Effects Programs)

1.2 关键词所带来的差异 (A Keyword Distinction)

关键词的困扰

策略性正确的struct (The Politically Correct Struct)

1.3 对象的差异 (An Object Distinction)

指针的类型 (The Type of a Pointer)

加上多态之后 (Adding Polymorphism)

第2章 构造函数语意学 (The Semantics of Constructors)

2.1 Default Constructor的构造操作

“带有Default Constructor”的Member Class Object

“带有Default Constructor”的Base Class

“带有一个Virtual Function”的Class

“带有一个Virtual Base Class”的Class

总结

2.2 Copy Constructor的构造操作

Default Memberwise Initialization

Bitwise Copy Semantics (位逐次拷贝)

不要Bitwise Copy Semantics!

重新设定Virtual Table的指针

处理Virtual Base Class Subobject

2.3 程序转化语意学 (Program Transformation Semantics)

显式的初始化操作 (Explicit Initialization)

参数的初始化 (Argument Initialization)

返回值的初始化 (Return Value Initialization)

在使用者层面做优化 (Optimization at the User Level)

在编译器层面做优化 (Optimization at the Compiler Level)

Copy Constructor: 要还是不要?

摘要

2.4 成员们的初始化队伍 (Member Initialization List)

第3章 Data语意学 (The Semantics of Data)

3.1 Data Member的绑定 (The Binding of a Data Member)

3.2 Data Member的布局 (Data Member Layout)

## &lt;&lt;深度探索C++对象模型&gt;&gt;

## 3.3 Data Member的存取

Static Data Members

Nonstatic Data Members

## 3.4 “继承”与Data Member

只要继承不要多态 ( Inheritance without Polymorphism )

加上多态 ( Adding Polymorphism )

多重继承 ( Multiple Inheritance )

虚拟继承 ( Virtual Inheritance )

## 3.5 对象成员的效率 ( Object Member Efficiency )

## 3.6 指向Data Members的指针 ( Pointer to Data Members )

“指向Members的指针”的效率问题

## 第4章 Function语意学 ( The Semantics of Function )

## 4.1 Member的各种调用方式

Nonstatic Member Functions ( 非静态成员函数 )

Virtual Member Functions ( 虚拟成员函数 )

Static Member Functions ( 静态成员函数 )

## 4.2 Virtual Member Functions ( 虚拟成员函数 )

多重继承下的Virtual Functions

虚拟继承下的Virtual Functions

## 4.3 函数的效能

## 4.4 指向Member Function的指针 ( Pointer-to-Member Functions )

支持“指向Virtual Member Functions”的指针

在多重继承之下, 指向Member Functions的指针

“指向Member Functions之指针”的效率

## 4.5 Inline Functions

形式参数 ( Formal Arguments )

局部变量 ( Local Variables )

## 第5章 构造、析构、拷贝语意学 ( Semantics of Construction, Destruction, and Copy )

纯虚函数的存在 ( Presence of a Pure Virtual Function )

虚拟规格的存在 ( Presence of a Virtual Specification )

虚拟规格中const的存在

重新考虑class的声明

## 5.1 “无继承”情况下的对象构造

抽象数据类型 ( Abstract Data Type )

为继承做准备

## 5.2 继承体系下的对象构造

虚拟继承 ( Virtual Inheritance )

vptr初始化语意学 ( The Semantics of the vptr Initialization )

## 5.3 对象复制语意学 ( Object Copy Semantics )

## 5.4 对象的效能 ( Object Efficiency )

## 5.5 析构语意学 ( Semantics of Destruction )

## 第6章 执行期语意学 ( Runtime Semantics )

## 6.1 对象的构造和析构 ( Object Construction and Destruction )

全局对象 ( Global Objects )

局部静态对象 ( Local Static Objects )

对象数组 ( Array of Objects )

## <<深度探索C++对象模型>>

Default Constructors和数组

6.2 new和delete运算符

针对数组的new语意

Placement Operator new的语意

6.3 临时性对象 ( Temporary Objects )

临时性对象的迷思 ( 神话、传说 )

第7章 站在对象模型的尖端 ( On the Cusp of the Object Model )

7.1 Template

Template的“实例化”行为 ( Template Instantiation )

Template的错误报告 ( Error Reporting within a Template )

Template中的名称决议法 ( Name Resolution within a Template )

Member Function的实例化行为 ( Member Function Instantiation )

7.2 异常处理 ( Exception Handling )

Exception Handling快速检阅

对Exception Handling的支持

7.3 执行期类型识别 ( Runtime Type Identification , RTTI )

Type-Safe Downcast ( 保证安全的向下转换操作 )

Type-Safe Dynamic Cast ( 保证安全的动态转换 )

References并不是Pointers

Typeid运算符

7.4 效率有了，弹性呢？

动态共享函数库 ( Dynamic Shared Libraries )

共享内存 ( Shared Memory )

## &lt;&lt;深度探索C++对象模型&gt;&gt;

## 章节摘录

版权页：插图：Member Function的实例化行为（Member Function Instantiation）对于template的支持，最困难的莫过于template function的实例化（instantiation）。

目前的编译器提供了两个策略：一个是编译时期策略，程序代码必须在program text file中备妥可用；另一个是链接时期策略，有一些meta.compilation工具可以导引编译器的实例化行为（instantiation）。

下面是编译器设计者必须回答的三个主要问题：1.编译器如何找出函数的定义？

答案之一是包含template program text file，就好像它是一个header文件一样。

Borland编译器就遵循这个策略。

另一种方法是要求一个文件命名规则，例如，我们可以要求，在Point.h文件中发现的函数声明，其template program text一定要放置于文件Point.C或Point.cpp中，依此类推。

cfront就遵循这个策略。

Edison DesignGroup编译器对这两种策略都支持。

2.编译器如何能够只实例化程序中用到的member functions？

解决办法之一就是，根本忽略这项要求，把一个已经实例化的class的所有member functions都产生出来。

Borland就是这么做的——虽然它也提供#pragmamas让你压制（或实例化）特定实例。

另一种策略就是模拟链接操作，检测看看哪一个函数真正需要，然后只为它（们）产生实例。

cfront就是这么做的。

Edison DesignGroup编译器对这两种策略都支持。

3.编译器如何阻止member definitions在多个.o文件中都被实例化呢？

解决办法之一就是产生多个实例，然后从链接器中提供支持，只留下其中一个实例，其余都忽略。

另一个办法就是由使用者来导引“模拟链接阶段”的实例化策略，决定哪些实例（instances）才是所需求的。

目前，不论是编译时期还是链接时期的实例化（instantiation）策略，均存在以下弱点：当template实例被产生出来时，有时候会大量增加编译时间。

很显然，这将是template functions第一次实例化时的必要条件。

然而当那些函数被非必要地再次实例化，或是当“决定那些函数是否需要再实例化”所花的代价太大时，编译器的表现令人失望！

C++支持template的原始意图可以想见是一个由使用者导引的自动实例化机制（use-directed automatic instantiation mechanism），既不需要使用者的介入，也不需要相同文件有多次的实例化行为。

但是这已被证明是非常难以达成的任务，比任何人此刻所能想象的还要难（请参考[S7ROUP94]）。

ptlink，随着cfront 3.0版所附的原始实例化工具，提供了一个由使用者驱动的自动实例化机制（use-driven automatic instantiation mechanism），但它实在太复杂了，即使是久经世故的人也没法一下子了解。

## <<深度探索C++对象模型>>

### 编辑推荐

《深度探索C++对象模型》重点探索了“面向对象程序所支持的C++对象模型”下的程序行为，对于“面向对象性质的基础实现技术”以及“各种性质背后的隐含利益交换”提供了一个清楚的认识，检验了由程序变形所带来的效率冲击，提供了丰富的程序范例、图片，以及面向对象观念和底层对象模型之间的效率测量。



<<深度探索C++对象模型>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>