附　录

附录A　C++面试宝典

阅读这本书肯定能让你的C++生涯有一个爆发式的跃进，不过老板们在花钱雇用你之前，肯定希望你能证明自己。不同公司的面试方法不同，但是对于技术面试这一关，很多方面还是有章可循的。力求全面的面试人员可能想对你的编码技能、调试技能、设计和风格方面的技能，以及问题解决技能统统都测试一番。可能会问到的问题有很多。在这个附录里，你将了解到可能会遇到的一些不同类型的问题，在此还提供了一些最佳策略可以帮助你获得一份高薪的C++编程工作。

这个附录还是按照本书章节的顺序来组织，我们会讨论在面试情况下，针对各章的内容哪些方面会被问到。每一节都会讨论面试人员可能会设计哪些问题来测试你有关的技能，并指出可以采用哪些最佳方法来对付这些问题。

A.1　第1章：C++快速入门

技术面试人员经常会准备一些基本的C++问题，如果有人只是听说过C++而已，就在简历中写上“有C++经验”想要滥竽充数，这些问题就能验明他们的正身，而无法蒙混过关。这种问题可能在“电话场景”中就会问到，在叫你参加现场面试之前先由一个开发人员或招聘人员打电话给你。也可能是通过电子邮件或当面询问。在回答这些问题时，要记住，面试人员只是想知道你确实学过并用过C++。一般不需要为获得高分过于深入到每个细节中。

A.1.1　要点

- main() 及其参数
- 头文件语法，而且要知道标准库头文件名中没有“.h”
- 命名空间的基本使用
- 语言基本知识，如循环语法、三元操作符和变量
- 栈和堆之间的差别
- 动态分配的数组
- const的使用

A.1.2　常见问题

基本C++问题一般会采用术语测试的形式提出。面试人员可能让你定义一些C++术语，如const或

static。你只要回答出课本上的答案就可以让面试人员满意了，不过，如果你再给出一个使用示例或提供额外的一些细节，可能会得到加分。例如，除了说明 const 的一个作用是可以指定引用参数不能被修改，还可以指出在向函数或方法传递一个对象时，const 引用要比复制对象效率更高。

基本 C++问题还可能以另一种形式出现，也许让你当着面试人员的面编写一个小程序。他会给你一个“热身”问题，如“用 C++写 *Hello, World* 程序”。如果拿到这样一个看来很简单的问题，你要保证能充分展现自己的技能来获得所有加分，如可以使用命名空间，使用流而不是 printf()，而且要知道应该包含哪些标准头文件。

A.2 第 2 章：设计专业的 C++程序

面试人员可能想确定你除了知道 C++语言外，对如何应用这种语言也很精通。在此并不是明确地问你一个设计问题，不过好的面试人员会把许多技术“暗藏”在其他一些问题中，后面就会看到。

A.2.1 要点

- 设计是主观的，要准备好在面试中为你的设计决定“辩护”。
- 在面试之前，回忆一下以前所做设计的一些细节，以备面试时可能会要求你提供一个例子。
- 准备好可能要定义抽象（abstraction）这个术语，并提供一个例子。
- 准备好可能会让你指出代码重用有哪些好处。
- 准备好要以可视化的方式画出一个设计，包括类层次体系。

A.2.2 常见问题

面试人员一般很难提问设计问题，在面试情况下无论编写多大的程序，也不足以展示实际的设计技能。设计问题可能会以一种比较模糊的方式提出“告诉我设计一个好的程序有哪几步?”或者“请解释抽象的原则”。还可能更隐含一些，在讨论你先前的工作时，面试人员可能会说“你能解释一下那个项目的设计吗?”。

A.3 第 3 章：基于对象的设计

面向对象设计问题可以把 C 程序员从 C++程序员中筛出来，C 程序员只知道引用是什么，而 C++程序员确实会使用 C++语言的面向对象特性。面试人员不会想当然地相信你，即使你使用面向对象语言已经很多年了，他们还是希望看到真正的证据来表明你确实理解面向对象方法。

A.3.1 要点

- 过程式和面向对象模式的区别
- 类和对象的区别
- 用组件、属性和行为来表示类
- Is-a 和 has-a 关系
- 多重继承时的折衷

A.3.2 常见问题

面向对象设计问题一般有两种问法。可能让你定义一个面向对象概念，或者可能让你画出一个面向对象层次体系。前者相当简单。要记住提供例子可以给你加分。

如果让你画出一个面向对象层次体系，面试人员通常会提供一个简单的应用，如扑克牌游戏，你应

该能为这个游戏设计一个类层次体系。面试人员经常会问有关游戏的问题，因为这是大多数人熟悉的应用。与数据库实现之类的问题相比，这些游戏问题还可以稍微活跃一下气氛。当然，取决于所问到的游戏或应用，你生成的层次体系可能有很大差别。以下是要考虑的一些要点：

- 面试人员想了解你的思考过程。要把想法充分表现出来，进行头脑风暴，与面试人员一同讨论，不要怕和面试人员的观点不一致。
- 面试人员可能认为你对所问的应用已经很熟悉。如果你从来没有听说过 21 点扑克牌游戏，可以指出来，让面试人员把问题说清楚一些，或者换个问题。
- 除非面试人员要求你在描述层次体系时使用一种特定的格式，否则，建议你的类图采用继承树的形式，并为每个类提供一个大致的方法和数据成员列表。
- 你可能要为你的设计“辩护”，或者把新增加的需求考虑在内，对设计进行修改。

要搞清楚面试人员到底是什么目的，是想看你的设计中是否存在问题，还是就是要故意与你有不同观点，来看你的说服力如何。

A.4　第 4 章：基于库和模式的设计

你的准老板想知道你能不能使用并不是你写的代码。如果你在简历中罗列了一些特定的库，就应该准备好回答有关的问题。如果没有，对库的重要性有一个一般的理解就足够了。

A.4.1　要点

- 从头开始构建和重用既有代码之间的折衷
- 大 O 记法的基础知识（或者，至少要记住 $O(n \log n)$ 要优于 $O(n^2)$）（译者注：这种说法只适用于问题规模大的情况）
- C++标准库中包含的功能
- 设计模式的高层定义

A.4.2　常见问题

如果面试人员问有关一个特定库的内容，他（她）可能会强调库的高层方面而不是技术细节。例如，本书作者之一就经常问应聘者这样一个问题，从库设计的角度看，STL 的优点和缺点是什么。最好的应聘者会指出 STL 的广度和标准化是它的强大之处，但它的学习曲线很陡，这是 STL 的主要缺点。

你可能会被问到一个设计问题，但是如果这个问题有关于一个库，可能听上去并不像是一个设计问题。例如，面试人员可能会问你如何创建一个应用从网上下载 MP3 音乐，并在本地计算机上播放。这个问题并没有明确与库相关，但是这正是关键所在：这个问题实际上问的是过程。应该先说明你将如何收集需求，如何建立初始原型。由于这个问题提到了两个特定的技术，面试人员可能想了解你会如何处理这些技术。在此库就要登场了。如果告诉面试人员你要编写自己的 Web 类和播放 MP3 的代码，并不一定导致你面试失败，但是也许会让你衡量一下从头开始建立这样一些工具需要多少时间，代价如何，这就会把你难住。更好的回答是指出你会调查现有的库，看看有没有完成 Web 和 MP3 功能的库可以满足项目要求。还可以提到一些你了解的技术，如 Linux 中可以采用 libcurl 完成 Web 获取，或者在 Windows 中可以使用 Windows Media 库来完成音乐回放。

A.5　第 5 章：重用设计

面试人员很少会问到设计可重用代码的问题。疏忽了这一点会带来不好的后果，如果只会写一次性代码的程序员通过了面试，得到机会参与团队开发，他们往往会对整个编程组织带来危害。有时，你可

能发现某个公司特别强调代码重用，在面试时会问到这方面的问题。如果真的问到了重用方面的问题，这就表明这是一家不错的公司，值得你为它效力！

A.5.1 要点

- 抽象原则
- 创建子系统和类层次体系
- 好的接口设计的一般原则
- 什么时候使用模板，什么时候使用继承

A.5.2 常见问题

有关重用的问题大多都有关于你先前开发过的项目。例如，如果你曾经为某家公司工作过，该公司同时推出两套视频编辑应用，分别面向普通消费者和专业人员，面试人员可能就会问这两个应用之间如何共享代码。即使没有明确地问你有关代码重用的问题，你也能“隐蔽”地谈到。在介绍你以前的工作时，可以告诉面试人员你写的模块是否用于其他项目中。即使是回答相当直接的编码问题，也一定要考虑并提到所涉及的接口。

A.6 第6章：充分利用软件工程方法

如果你顺利地通过了一家公司的整个面试，但是面试人员从头到尾都没有问到任何过程问题，你就要有所怀疑了，这可能说明这家公司根本没有过程，或者他们不关心过程。还有一种可能，也许他们不希望你被他们的庞大过程吓跑。大多数情况下，你都有机会问公司一些问题。建议你考虑把工程过程作为标准问题之一。

A.6.1 要点

- 传统的生命期模型
- 形式化模型（如统一开发过程）的权衡考虑
- 极限编程的主要原则
- 你用过的其他过程

A.6.2 常见问题

最常问到的问题就是要求你介绍一下以前的老板所用的过程。在回答这个问题时，应该指出哪些地方成功了，哪些地方失败了，不过不要对任何特定的方法本身妄加断言。你批评的方法没准正是你的面试人员所用的方法。如果你看不上极限编程，现在先不要表露出来。

本书作者曾经花大量的时间翻看应聘人员的简历，有一个趋势很明显，这就是如今每一个人都把极限编程列为他们的一项技能。尽管在这方面没有多少确凿的数据可以证明，但是在编程环境中严格遵循极限编程往往不太可能。我们了解的情况是，许多公司已经开始涉入极限编程，而且采纳了极限编程的一些原则，但是并没有正式地真正实施。

如果面试人员问你极限编程的问题，他或她可能不希望你只是背书上的定义，面试人员知道，你可能看过一本极限编程书的目录。应该挑出极限编程中你觉得有意义的一些思想。向面试人员解释这些思想，同时要加上自己的理解。尽量和面试人员展开讨论，根据面试人员言辞里的蛛丝马迹，让讨论朝着他（她）感兴趣的方向走。

> 你与面试人员讨论极限编程之类高级概念的时间越多，他纠缠一些细节问题（如模板语法）的时间就越少。当然你肯定不能完全躲过技术问题，不过可以借此尽量减少！

A.7 第7章：好的编码风格

有机会参与专业编程的人都应该见过，有的同事的代码编写得相当简明。如果一个人编写的代码一塌糊涂，肯定没有人愿意同他共事。所以面试人员有时想明确应聘人员在编码风格方面的技能。

A.7.1 要点

- 风格问题，即使面试问题并不是明确地与风格相关！
- 写得好的代码不需要太多注释。
- 注释可以传达一些元信息。
- 分解的原则
- 重构的原则
- 命名技术

A.7.2 常见问题

风格问题会以几种不同的形式提出。本书作者之一就曾经遇到过这样一个问题，要求他在白板上写出一个相当复杂的算法的代码。他写出第一个变量名时，面试人员就让他停下来，告诉他已经通过了。问题并不在于算法，真正意图是想了解他能怎样对变量命名。更常见的情况是，可能会让你提交一份以前写的代码，或者只是让你说说对风格的观点。

如果你的准老板让你提交代码时，可要注意。把你为以前老板编写的代码提交出去可能并不合法。而且你必须找出一段合适的代码，既能展现你的技能，又不需要太多背景知识。例如，如果要面试一份数据库管理员的职位，肯定不会把有关高速图像渲染的硕士论文提交给面试公司。

如果面试公司让你写一个特定的程序，这是一个极好的机会，可以充分展现你从这本书里学到了什么。其他应聘人员可能不会为他们的程序提供单元测试，或者加上充分的注释。即使准老板没有指定程序，你也应该考虑专门编写一个小程序提交给公司。不用选择你以前编写的一些代码，可以从头开始编写与所应聘工作相关的代码，而且要强调好的风格。

A.8 第8、9章：类和对象

关于类和对象，可能问到的问题门类相当繁多。有些面试人员比较看注语法，可能会给你看（或者让你编写）一些复杂的代码。另外一些可能不太关心实现，而对你的设计技能更感兴趣。

A.8.1 要点

- 基本类定义语法
- 方法和数据成员的访问限定符
- this 指针的使用
- 对象创建和撤销
- 编译器在哪些情况下会为你生成构造函数
- 初始化列表

- 复制构造函数和赋值操作符
- mutable 关键字
- 方法重载和默认参数
- 友元类

A.8.2 常见问题

像“关键字 mutable 有什么含义?”之类的问题经常会在电话中问到。招聘人员可能备有一组 C++术语，根据回答的正确与否来确定应聘人员能否进入下一轮面试。也许无法知道所有可能问到的术语，不过要记住，其他应聘人员也将遇到同样的问题，这也是招聘人员对你们做出评判的标准之一。

面试人员和授课老师都经常会问一种查找 bug 的问题。可能会给你一段不太复杂的代码，要求指出这段代码中存在的问题。面试人员会想方设法地分析应聘人员的能力，而这种查找 bug 的问题就是可能用到的一种方法。一般来讲，应当仔细阅读每一行代码，把你的想法说出来，充分地进行头脑风暴。bug 可能有以下几类：

- 语法错误。这种错误比较少见，面试人员知道你可以利用编译器找出编译时 bug。
- 内存错误。这包括内存泄漏和双重删除等问题。
- “不能做”的问题。这种错误包括理论上讲正确但是结果不合理的一些问题。
- 风格错误。即使面试人员不会说这是一个 bug，但还是应该指出不好的注释或变量名。

比如要找出以下程序的 bug，这里就表现出上述几个方面的问题。

```
class Buggy
{
    Buggy(int param);
    ~Buggy();

    double fjord(double inVal);
    int fjord(double inVal);

protected:
    void turtle(int i = 7, int j);
    int param;
    double* graphicDimension;
};

Buggy::Buggy(int param)
{
    param = param;
    graphicDimension = new double;
}

Buggy::~Buggy()
{
}

double Buggy::fjord(double inVal)
{
    return inVal * param;
}

int Buggy::fjord(double inVal)
{
```

```
    return (int)fjord(inVal);
}

void Buggy::turtle(int i, int j)
{
    cout << "i is " << i << ", j is " << j << endl;
}
```

请仔细查看代码，然后参考以下正确的版本，找出答案。

```
#include <iostream> // Streams are used in the implementation.

class Buggy
{
public: // These should probably be public or else the class is pretty useless.
    Buggy(int inParam);  // Parameter naming
    ~Buggy();

    Buggy(const Buggy& src);              // Provide copy ctor and operator=
    Buggy& operator=(const Buggy& rhs);   // when the class has dynamically
                                          // allocated memory.

    double fjord(double inVal); // int version won't compile
                                // (overloaded methods differ only
                                // in return type). It's also useless
                                // because it just returns the argument
                                // it's given.

protected:
    void turtle(int i, int j);  // Only last arguments can have defaults.
    int mParam;                 // Data member naming
    double* graphicDimension;
};

Buggy::Buggy(int inParam) : mParam(inParam) // Avoid name ambiguity.
{
    graphicDimension = new double;
}

Buggy::~Buggy()
{
    delete graphicDimension;  // Avoid memory leak.
}

Buggy::Buggy(const Buggy& src)
{
    graphicDimension = new double;
    *graphicDimension = *(src.graphicDimension);
}

Buggy& Buggy::operator=(const Buggy& rhs)
{
    if (this == &rhs) {
        return (*this);
    }
```

```
    delete graphicDimension;
    graphicDimension = new double;
    *graphicDimension = *(rhs.graphicDimension);
    return (*this);
}

double Buggy::fjord(double inVal)
{
    return inVal * mParam;    // Changed data member name
}
void Buggy::turtle(int i, int j)
{
    std::cout << "i is " << i << ", j is " << j << std::endl; // Namespaces
}
```

A.9 第 10 章：探索继承技术

有关继承的问题一般与类问题形式相同。面试人员可能还要求你实现一个类层次体系来表明你确实用过 C++，而且用得不少，能够派生子类，而不仅仅只是书面了解。

A.9.1 要点

- 派生一个类的语法
- 从子类的角度看，私有（private）和保护（protected）的区别
- 方法覆盖和虚方法
- 串链构造函数
- 向上和向下类型强制转换的输入和输出
- 多态原则
- 纯虚方法和抽象基类
- 多重继承
- 运行时类型识别

A.9.2 常见问题

继承问题中的许多陷阱都与细节有关。在编写一个基类时，不要忘记给方法加上关键字 virtual。如果把所有方法都加上 virtual，要准备好应该能解释这种做法。要能解释 virtual 的含义及其原理。类似地，在子类定义中，不要忘记在父类名的前面加上 public 关键字（例如 class Foo : public Bar）。一般不会在面试时让你完成多重继承。

更有难度的继承问题与超类和子类的关系有关。一定要明白不同访问级别如何工作，以及私有和保护之间的区别。要提醒自己一种称为切割的现象，即某些类型的强制转换会导致一个类丢失其子类信息。

A.10 第 11 章：利用模板编写通用代码

作为 C++中最神秘的一部分，面试人员会充分利用模板把 C++新手从高手中区分出来。如果你没有记住某些高级模板语法，尽管大多数面试人员都会原谅你，但是你起码要展示出你了解基本的模板语法。

A.10.1 要点

- 如何编写一个基本的模板类
- 模板的两个主要缺点，语法难看和代码膨胀
- 如何使用模板类

A.10.2 常见问题

许多面试问题都会从一个简单的问题入手，然后逐步增加复杂度。通常，面试人员准备的问题复杂性可以无限延伸，他只是想知道你能够应付到哪一级。比方说，面试人员开始时可能要求你创建一个类，为固定数目的 int 提供顺序访问。接下来，可能要求这个类能够扩展以适应一个任意大小的数组。然后，可能需要有任意的数据类型（而不只是 int），在这里模板可以上场了。在此之后，面试人员可以让问题朝着不同的方向发展，比如让你使用操作符重载来提供数组型的语法，或者继续沿着模板的道路让你提供一个默认类型。

一般不会明确地问模板问题，模板更有可能在解决其他编码问题时用到。你应该好好复习有关的基础知识，以备出现这种问题。不过，大多数面试人员都知道模板语法很难，在面试时要求写出复杂的模板代码有些过于苛刻。

A.11 第 12 章：理解 C++疑难问题

许多面试人员可能想强调一些更难解的情况，这样一来，有经验的 C++程序员就能充分发挥，表现出他们连 C++这些不寻常的领域也能够征服。有时面试人员想问一些有意思的问题，但是可能有困难，最后只好问他们能想到的最难解的问题。

A.11.1 要点

- 引用在声明时必须与变量绑定，而且这种绑定不能改变。
- const 的多种使用
- static 的多种使用
- C++中不同类型的强制转换

A.11.2 常见问题

让应聘人员定义 const 和 static，这是一个经典的 C++面试问题。这两个关键字能够作为标尺，面试人员可以用它来衡量应聘者的水平。比如，一个一般的应聘者可能会谈到 static 方法和 static 数据成员。一个好的应聘者会给出 static 方法和 static 数据成员的好例子。最棒的应聘者还知道 static 链接和函数中的 static 变量。

这一章介绍的边缘情况也常常出现在查找 bug 之类的问题中。要当心引用的误用。例如，假设有一个包含引用作为数据成员的类。

```
class Gwenyth
{
    public:
        int& mCaversham;
};
```

由于 mCaversham 是一个引用，在类构造时它就必须与一个变量绑定。为此，需要使用一个初始化

列表。类可以取所引用的变量作为构造函数的一个参数。

```
class Gwenyth
{
    public:
        Gwenyth(int i);
        int& mCaversham;
};

Gwenyth::Gwenyth(int i) : mCaversham(i)
{
}
```

A.12 第 13 章：有效的内存管理

低级程序员或有 C 背景的 C++程序员可能会问到与内存有关的问题。目的是想确定你是否过于重视 C++的面向对象方面，而忽略了底层实现细节。你可以把内存管理问题作为一个机会，来证明你确实知道底层的具体工作。

A.12.1 要点

- 画出栈和堆将有助于你理解底层的具体工作。
- 使用 new 和 delete 而不是 malloc()和 free()。
- 对数组使用 new[]和 delete[]。
- 如果有一个对象指针数组，还需要为各个单个指针分配内存和释放内存，数组分配语法并没有考虑指针的情况。
- 或者，你总能这么说“当然，在实际中，我可以具体运行来找出问题。”

A.12.2 常见问题

在查找 bug 的问题中通常包含内存问题，如双重删除、new/new[] 不匹配和内存泄漏等。在查看大量使用指针和数组的代码时，可以在分析每行代码时画出并更新内存的状态。即使你能马上看到答案，这样做也能让面试人员知道，你确实能够画出内存的状态。

要看应聘人员是否理解内存，还有一个好办法，就是让他回答指针和数组有什么区别。在这个时候，你心里对这二者之间的差别可能没有一个清晰的界定，这会让你有所迟疑。如果是这样，请再略读第 13 章有关的讨论。

A.13 第 14 章：揭开 C++I/O 的神秘面纱

如果你应聘一份编写 GUI 应用的工作，可能不会问你太多有关 I/O 流的问题，因为 GUI 应用会使用其他的机制来完成 I/O。不过，流可能会在其他问题中出现，而且，作为 C++的一个标准部分，只要面试人员关心，肯定就会问到这方面的问题。

A.13.1 要点

- 流的定义
- 使用流的基本输入和输出
- 管理器的概念

- 流的类型（控制台、文件、字符串等等）
- 错误处理技术
- 国际化的重要性

A.13.2　常见问题

I/O 可以在任何问题中出现。比如说，面试人员可能让你读入一个包含测验分数的文件，再把读入的分数放在一个 vector 中。这个问题可以测试你的基本 C++ 技能以及对基本 STL 和基本 I/O 的理解。即使 I/O 只是所处理问题的一个小部分，也要仔细检查是否存在错误。如果没有很好地检查 I/O 错误，本来很好的程序也会让面试人员有些微辞。

你的面试人员可能不会专门问有关国际化的问题，不过在面试时如果使用 wchar_t 而不是 char，就能表现出你考虑到了国际化情况。如果确实要让你说说对国际化有什么经验，一定要提到从一开始就考虑国际化的重要性，还要表明你了解 C++ 的本地化工具。

A.14　第 15 章：处理错误

管理层有时不太会让刚刚毕业的研究生或新手承担一份重要的（高薪）工作，因为一般认为他们写不出成品质量的代码。你可以在面试过程中展示你的错误处理技能，从而向面试人员证明你不是泛泛之辈。

A.14.1　要点

- 将异常捕获为 const 引用。
- 对于成品代码，异常体系更可取，而不只是几个通用异常而已。
- C++ 中的抛出列表与 Java 中的抛出列表有所不同。
- 在抛出异常时智能指针有助于避免内存泄漏。

A.14.2　常见问题

你可能会拿到一个与异常直接有关的问题，除非会问得更为特定，如让你描述栈展开是如何进行的。不过，面试人员可能会特别留意你如何报告和处理错误。

当然，并不是所有程序员都理解或赞同异常。有些程序员出于性能原因甚至反对使用异常。如果面试人员让你完成某个工作时不要用异常，你就必须“还原”为传统的 NULL 检查和错误码。这是一个好机会，可以让你展示自己了解 nothrow new!

A.15　第 16 章：重载 C++ 操作符

在面试时，有可能不只是让你完成一个简单的操作符重载，而是要完成某个更难的工作，尽管这种情况比较少见。有些面试人员可能会准备一个高级问题，他们并不指望每个人都能正确地做出回答。操作符重载的复杂性使之完全可以作为这样的高级问题，因为很少有程序员在不检查的情况下一次就把语法写对。这说明，面试前应该好好看看这个方面。

A.15.1　要点

- 重载流操作符，因为这是最常见的重载操作符，而且概念上是特有的。
- 函数对象是什么，如何创建函数对象。
- 在方法操作符和全局友元函数之间如何选择。

- 有些操作符可以用其他操作符表示（也就是说，operator<= 可以写作对 operator >的结果取反）。

A.15.2 常见问题

要接受一个现实：操作符重载问题（除了简单的操作符重载）确实很苛刻。问这种问题的人都知道这一点，所以如果你回答正确，肯定会对你印象加深。要想预计到具体会问你什么问题，这往往不太可能，但是操作符是有限的。只要你看过每个可重载操作符的重载例子，你就会有很好的表现。

除了让你实现一个重载操作符，还可能让你回答有关重载操作符的高级问题。查找 bug 的问题可能包含有一个重载操作符，但是这个操作符重载为完成某种不合适的行为，即并不是这个操作符从概念上讲本应完成的行为。除了语法外，还要牢记操作符重载的用例和理论。

A.16 第 17 章：编写高效的 C++程序

效率问题在面试中很常见，因为许多组织都遭遇到代码的可扩展性问题，特别需要在性能方面精通的程序员。

A.16.1 要点

- 语言级效率很重要，但是这方面的改进是有限的。从最后看来，设计级选择更为重要。
- 引用参数效率更高，因为引用参数可以避免复制。
- 对象池有助于避免创建和撤销对象的开销。
- 测评非常重要，可以确定哪些操作确实占用了大部分的运行时间。

A.16.2 常见问题

通常，面试人员会使用他们自己的产品作为例子来提问效率问题。有时面试人员会描述一个原来的设计，并指出他遇到的一些与性能有关的症状，要求应聘人员提出一个新的设计来解决这个问题。遗憾的是，面试人员可能已经解决了这个问题，而你的解决方案与他的解决方案完全相同的可能性微乎其微，这正是此类问题的一个不合适的地方。因为可能性很小，所以更要仔细检查你的设计。也许你提出的不是面试人员的解决方案，但是你的答案也许也是正确的，没准比他的新设计更好。

还可能有其他类型的效率问题，比如让你调整某些 C++代码来改善性能，或者描述某个算法。例如，面试人员可能给你看一段代码，其中包含太多复制或低效的循环。

A.17 第 18 章：开发跨平台和跨语言的应用

程序员提交的简历中都很少只列一种语言或一种技术，而且大型应用也很少只利用一种语言或技术。即使你只是应聘一份有关 C++的工作，面试人员也可能会问到有关其他语言的问题，特别是其他语言与 C++的关系。

A.17.1 要点

- 平台在哪些方面有区别（如体系结构、大小等等）
- 编程和写脚本之间的界线
- C++和其他语言的交互

A.17.2 常见问题

最常见的跨语言问题是对两种不同语言进行比较。你不要对某种语言发表溢美或贬低之辞，即使你确实不喜欢 Java。面试人员只是想知道你能够了解到不同语言之间的权衡，并能做出选择。

跨平台问题常常在讨论以前工作的时候问到。如果你的简历中称你曾经在一个定制的硬件平台上编写过 C++应用，就要准备好谈一谈你用的编译器，并指出使用那个平台时有哪些难题。

A.18 第 19 章：熟练地测试

你的准老板会对高超的测试能力大加赞赏。因为你的简历可能无法展示测试技能，除非你有 QA 经验，否则面试时就会遇到有关测试的问题。

A.18.1 要点

- 黑盒测试和白盒测试的区别
- 单元测试的概念，以及编写代码时同时编写测试
- 更高级测试技术
- 你以前工作的测试和 QA 环境：哪些可取，哪些不可取？

A.18.2 常见问题

面试人员可能要求你在面试时编写一些测试，不过在面试时写出的测试程序往往不太可能有足够的深度，不能算是有趣的测试。更有可能让你回答高级测试问题。要准备好介绍你的上一个工作中测试是如何完成的，还要指出你觉得其中哪些可取，哪些不可取。等你回答完之后，也可以问问面试人员的看法，这是一个很好的问题。在理想情况下，你们将就此展开对测试的讨论，还能让你对将来的工作环境有一个更好的了解。

A.19 第 20 章：征服调试

工程组织所需要的应聘人员是这样的，他们不仅能调试自己的代码，还能调试他们以前从未见过的代码。技术面试人员通常想衡量你的调试能力究竟如何。

A.19.1 要点

- 调试不是在 bug 出现时才开始，你应当提前在代码中做准备，以便出现 bug 时能够从容应对。
- 日志和调试工具是最好的帮手。
- bug 的症状可能表现得与实际的根本原因毫不相干。
- 内存图表有助于完成调试，特别是在面试过程中更需要画出内存图表。

A.19.2 常见问题

在面试时，你可能会被一个很复杂的调试问题难住。要记住，过程是最重要的，面试人员也可能知道这一点。即使你在面试过程中没有找出 bug，但是要让面试人员知道，你想通过哪些步骤找出问题所在。如果面试人员给你一个函数，指出它在运行时出问题了，如果应聘人员正确地说出了查找 bug 的一系列步骤，即使他没有马上找出 bug，面试人员也会给他同样的加分。

A.20 第 21、22、23 章：标准模板库

可以看到，STL 可能很难使用。面试人员一般不会指望你背出 STL 类的细节，除非你声称自己是一个 STL 专家。如果你知道你所面试的工作会大量使用 STL，就需要在面试前一天写一些 STL 代码来恢复记忆。否则，复习 STL 的高级设计就应该足够了。

A.20.1 要点

- 不同类型的容器及其与迭代器的关系
- vector 的基本用法，这可能是最常用的 STL 类
- 关联容器的用法，如 map
- STL 算法和一些内置算法的用途
- 如何扩展 STL（往往不太需要细节）
- 你自己对 STL 的看法

A.20.2 常见问题

如果面试人员特别热衷于问详细的 STL 问题，能够问到什么确实没有止境。如果你对语法不太肯定，可以在面试时表明“当然在实际中，我会在《C++高级编程》里查，不过肯定它会这样工作……”。至少这会让面试人员感觉到，只要你基本思想正确，细节上有出入是可以原谅的。

有关 STL 的高级问题通常用来衡量你用过多少 STL，而不会让你回忆所有细节。例如，一般的用户可能很熟悉关联和非关联容器。比较高级的用户能够定义一个迭代器，并说明迭代器如何用于容器。其他高级问题可能会问你有关 STL 算法的经验，以及你是否对 STL 做过定制实现。

A.21 第 24 章：探讨分布式对象

因为分布式应用非常常见，可能会让你设计一个分布式系统，或者回答有关某种特定分布式技术的问题。

A.21.1 要点

- 使用分布式计算的原因
- 分布式和网络式计算的区别
- 串行化和 RPC 的概念
- 如果你声称通晓 CORBA 或 XML，就必须有所准备

A.21.2 常见问题

许多技术简历中都充斥着大量的缩写词。如果在你的简历中列了一项诸如 XML 的技术，你的准老板并不能知道你的水平如何。除非你具体指出“基本 XML 技能”或“XML 专家”，针对你对自己的评价，可能会问你一些问题来确定你的评价是否属实。具体地，对 XML 来说，可能会让你定义诸如模式（schema）等术语，或者要求给出一个应用于给定 XML 文档的模式。

由于 XML 如此流行，本书作者之一就曾经在面试时为应聘者提供了一个简单的 XML 文档。要求应聘人员指出所有属性、所有元素，以及所有文本节点。这让应聘者有些为难，不过这能有效地证明这个人是否用过 XML，或者只是知道它是一种类 HTML 的语法。

A.22 第25章：结合技术和框架

第25章介绍的每种技术都能作为一个很好的面试问题。在此不再重复这一章中的内容，建议你在面试前再回过头来把第25章通读一遍，确保自己理解了每一项技术。

A.23 第26章：应用设计模式

由于设计模式在专业领域越来越流行（许多应聘人员甚至把这列为他们的一项技能），所以你很有可能遇到一位面试人员要求你解释某种模式，给出某个模式的用例，或者实现一个模式。

A.23.1 要点

- 模式的基本思想是作为一个可重用的面向对象设计概念
- 你在本书中读到的模式以及在工作中用到的其他模式
- 要知道成百上千的模式名字往往有冲突，所以你和面试人员可能会使用不同的词表述同一个东西。

A.23.2 常见问题

回答有关设计模式的问题通常就像在公园散步，除非面试人员希望你了解每一种已知模式的细节。幸运的是，大多数欣赏设计模式的程序员都只是想与你讨论，了解你的观点而已。毕竟，从书上或网上查找有关概念而不是自己死记硬背，这本身也是一个好模式！

附录B 参考书目

本附录提供了与C++相关主题有关的一些书和在线资源，其中有的曾作为我们编写本书的参考，还有一些书和资源则建议你自己补充阅读，以便更为深入地了解有关内容或背景。

B.1 C++

B.1.1 C++入门

- Harvey M. Deitel and Paul J. Deitel, *C++ How to Program (Fourth Edition)*, Prentice Hall, 2002, ISBN: 0-130-38474-7

一般就称之为“Deitel”书，阅读这本书不要求有任何编程经验。

- Bruce Eckel, *Thinking in C++, Volume 1: Introduction to Standard C++ (Second Edition)*, Prentice Hall, 2000, ISBN: 0-139-79809-9.

这是一本介绍C++编程的绝好的书，要求读者已经了解C。这本书可以免费从www.bruceeckel.com得到。

- Stanley B. Lippman and Josée Lajoie, *C++ Primer (Third Edition)*, Addison Wesley, 1998, ISBN: 0-201-82470-1.

这本书不要求读者了解C++，但是需要有使用高级面向对象语言的经验。

- Steve Oualline, *Practical C++ Programming (Second Edition)*, O'Reilly, 2003, ISBN: 0-596-00419-2.

这是一本入门性的C++书，不要求有任何编程经验。

- Walter Savitch, *Problem Solving with C++: The Object of Programming (Fourth Edition)*, Addison Wesley Longman, 2002, ISBN: 0-321-11347-0.

这本书不要求有任何编程经验，常作为入门性编程课程的教材。

B.1.2 C++综合

- Marshall Cline, *C++ FAQ LITE*, www.parashift.com/c++-faq-lite.
- Marshall Cline, Greg Lomow, and Mike Giru, *C++ FAQs (Second Edition)*, Addison Wesley, 1998, ISBN: 0-201-30983-1.

这些常见问题是从comp.lang.c++新闻组收集整理的，要想快速查找有关C++的某个知识点，这个资源很有用处。印刷版本比在线版本包含了更多信息，不过在线版本中的内容对于大多数专业C++程序员来说应该已经足够了。

- Stephen C. Dewhurst, *C++ Gotchas*, Addison Wesley, 2003, ISBN: 0-321-12518-5.

提供了C++编程的99项特别提示。

- Bruce Eckel and Chuck Allison, *Thinking in C++, Volume 2: Practical Programming (Second Edition)*, Prentice Hall, 2003, ISBN: 0-130-35313-2.

Eckel 书的第二卷，这本书涵盖了一些更高级的C++主题。同样地，这本书也可以免费从 www.bruceeckel.com. 在线获得。

- Ray Lischner, *C++ in a Nutshell*, O'Reilly, 2003, ISBN: 0-596-00298-X.

这是一本C++参考书，涵盖了从基础知识到更高级内容等所有方方面面。

- Scott Meyers, *Effective C++ (Second Edition): 50 Specific Ways to Improve Your Programs and Designs*, Addison Wesley, 1998, ISBN: 0-201-92488-9.
- Scott Meyers, *More Effective C++: 35 New Ways to Improve Your Programs and Designs*, Addison Wesley, 1996, ISBN: 0-201-63371-X.

这两本书对于通常被误用或误解的一些C++特性提供了很好的提示和技巧。

- Stephen Prata, *C++ Primer Plus*, Sams Publishing, 2001, ISBN: 0-672-32223-4.

这是目前内容最为详尽的C++书之一。

- Bjarne Stroustrup, *The C++ Programming Language (Special Third Edition)*, Addison Wesley, 2000, ISBN: 0-201-70073-5.

这是C++书的“圣经”，由C++设计者本人编写，每一个C++程序员都应该拥有这本书，不过对C++新手来说，这本书的某些地方可能不太好懂。

- *The C++ Standard: Incorporating Technical Corrigendum No. 1*, John Wiley & Sons, 2003, ISBN:0-470-84674-7.

这本书基本就是一个800页的标准。它没有解释如何使用C++，只是指出了有哪些形式规则。除非你确实想了解C++的每一个细节，否则我们并不推荐这本书。

- http://groups.google.com 上的新闻组，包括 comp.lang.c++.moderated 和 comp.std.c++。

这些新闻组包含了大量有用的信息，不过在这里也会出现许多不当言辞和错误信息。

- *The C++ Resources Network*: www.cplusplus.com/.

这个网页没有听上去那么有用。在作者写本书时，其中的C++参考部分尚在建设当中。

B.1.3 I/O流

- Cameron Hughes and Tracey Hughes, *Mastering the Standard C++ Classes: An Essential Reference*, Wiley, 1999, ISBN: 0-471-328-936.

这是一本介绍如何编写定制 istream 和 ostream 类的好书。

- Cameron Hughes and Tracey Hughes, *Stream Manipulators and Iterators in C++*, Professional Technical Reference, Prentice Hall, http://phptr.com/articles/article.asp?p=171014&seqNum=2.

这篇很棒的文章是由《*Mastering the Standard C++ Classes*》的作者撰写的，它澄清了用C++定义定制 stream 管理器的疑难问题。

- Philip Romanik and Amy Muntz, *Applied C++: Practical Techniques for Building Better Software*, Addison Wesley, 2003, ISBN: 0-321-10894-9.

这本书除了别具匠心地综合了软件开发建议和C++具体内容之外，还很好地解释了C++中为什么提供本地化和 Unicode 支持。

- Joel Spolsky, *The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)*, www.joelonsoftware.com/articles/

Unicode. html.

读了 Joel 的关于国际化重要性的文章后，你可能还想看他在 *Joel on Software* 上发表的其他文章。

- Unicode, Inc., *Where is my Character?*, www. unicode. org/standard/where.

这是查找 Unicode 字符、图、表的最佳资源。

B. 1. 4 C++标准库

- Nicolai M. Josuttis, *The C++ Standard Library: A Tutorial and Reference*, Addison Wesley, 1999, ISBN: 0-201-37926-0.

这本书涵盖了整个标准库，包括 I/O streams 和 strings 以及容器和算法。这是一个很好的参考文献。

- Scott Meyers, *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*, Addison Wesley, 2001, 0-201-74962-9.

Meyers 本着“Effective C++”的一贯精神编写了这本书。这本书提供了使用 STL 的有针对性的提示，但不是一本纯粹的参考书或教程。

- David R. Musser, Gillmer J. Derge, and Atul Saini, *STL Tutorial and Reference Guide (Second Edition)*, Addison Wesley, 2001, ISBN: 0-201-37923-6.

这本书与 Josuttis 的书很类似，不过只介绍了标准库中的 STL 部分。

B. 1. 5 C++模板

- Herb Sutter, *Sutter's Mill: Befriending Templates*, C/C++ User's Journal, www. cuj. com/documents/s=8244/cujcexp2101sutter/sutter. htm.

一般认为，应当让函数模板与类携手工作，而这篇文章是我们见过的有关这一点的最好解释。

- David Vandevoorde and Nicolai M. Josuttis, *C++ Templates: The Complete Guide*, Addison Wesley, 2002, ISBN: 0-201-73484-2.

在这本书中，你能找到想知道的有关 C++模板的任何内容（也包括不想知道的内容）。这本书要求对 C++的各个方面有一定的认识。

B. 2 C

- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language (Second Edition)*, Prentice Hall, 1998, ISBN: 0-13-110362-8.

这本书也称为“K and R”，这是有关 C 语言的一本绝好的参考书，不过作为入门不太合适。

- Peter Prinz, Tony Crawford (Translator), Ulla Kirch-Prinz, *C Pocket Reference*, O'Reilly, 2002, ISBN: 0-596-00436-2.

这是 C 中所有内容的一个简明参考。

- Eric S. Roberts, *The Art and Science of C: A Library Based Introduction to Computer Science*, Addison Wesley, 1994, ISBN: 0-201-54322-2.
- Eric S. Roberts, *Programming Abstractions in C: A Second Course in Computer Science*, Addison Wesley, 1997, ISBN: 0-201-54541-1.

这两本书很好地介绍了如何采用好的风格编写 C 程序，通常用作入门性编程课程的教材。

- Peter Van Der Linden, *Expert C Programming: Deep C Secrets*, Pearson Education, 1994, ISBN: 0-131-77429-8.

深入探究 C 语言，包括其演进以及其内部工作原理。

B.3 集成C++和其他语言

- Ian F. Darwin, *Java Cookbook*, O'Reilly, 2001, ISBN: 0-596-00170-3.

这本书循序渐进地介绍了如何使用JNI将Java与其他语言集成，其中也包括C++。

B.4 算法和数据结构

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms* (*Second Edition*), The MIT Press, 2001, ISBN: 0-262-03293-7.

这本书是最流行的入门性算法书之一，涵盖了所有常用的数据结构和算法。本书作者就是在上研究生时从这本书第一版开始学习算法和数据结构的。

- Donald E. Knuth, *The Art of Computer Programming Volume 1: Fundamental Algorithms* (*Third Edition*), Addison Wesley, 1997, ISBN: 0-201-89683-4.
- Donald E. Knuth, *The Art of Computer Programming Volume 2: Seminumerical Algorithms* (*Third Edition*), Addison Wesley, 1997, ISBN: 0-201-89684-2.
- Donald E. Knuth, *The Art of Computer Programming Volume 3: Sorting and Searching* (*Third Edition*), Addison Wesley, 1998, ISBN: 0-201-89685-0.

如果你力求严谨，那么再没有比Knuth的这三卷系列更好的算法和数据结构书了。如果没有在研究生阶段学习过数学或计算机科学理论，可能无法真正掌握这三本书。

- Kyle Loudon, *Mastering Algorithms with C*, O'Reilly, 1999, ISBN: 1-565-92453-3.

这是一本比较容易掌握的数据结构和算法参考书。

B.5 开源软件

- The Open Source Initiative (www.opensource.org)。
- GNU操作系统— Free Software Foundation (www.gnu.org)。

这是两个主要开源活动的相关网页，解释了其原则，并提供了如何得到开源软件，以及如何参与开源软件开发的有关信息。

- sourceforge.net (www.sourceforge.net)。

这个网站提供了许多开源项目，查找有用的开源软件的一个很好的资源。

B.6 软件工程方法论

- Barry W. Boehm, TRW Defense Systems Group, *A Spiral Model of Software Development and Enhancement*, IEEE Computer, 21 (5): 61-72, 1988.

这篇里程碑性的论文描述了软件开发随时间的状态变化，并提出螺旋模型。

- Kent Beck, *Extreme Programming Explained: Embrace Change*, Pearson Education, 1999, ISBN:0-201-61641-6.

有一系列书促进了极限编程成为软件开发的一种新方法，而这正是其中的一本。

- Robert T. Futrell, Donald F. Shafer, and Linda Isabell Shafer, *Quality Software Project Management*, Pearson Education, 2003, ISBN: 0-130-91297-2.

如果你负责软件开发过程的管理，可以拿这本书作为指南。

- Robert L. Glass, *Facts and Fallacies of Software Engineering*, Pearson Education, 2002, ISBN: 0-321-11742-5.

这本书讨论了软件开发过程的各个方面，并在此过程中指出了一些不被注意的“真理”。

- Philippe Kruchten, *Rational Unified Process: An Introduction (Second Edition)*. Addison Wesley, 2000, ISBN: 0-201-70710-1.

提供了 RUP 的一个概述，包括任务和过程。

- Edward Yourdon, *Death March (Second Edition)*, Prentice Hall, 2003, ISBN: 0-131-43635-X.

这本很有意思的书讨论了软件开发的政治性和现实性。

- Rational Unified Process from IBM, www3.software.ibm.com/ibmdl/pub/software/rational/web/demos/viewlets/rup/runtime/index.html

IBM 的网站包含有关 RUP 的大量信息，包括以上 URL 中提供的交互式表示。

B.7 编程风格

- Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999, ISBN: 0-201-48567-2.

这本经典的书针对识别和改进不好的代码提出了实践做法。

- James Foxall, *Practical Standards for Microsoft Visual Basic .NET*, Microsoft Press, 2002, ISBN:0-7356-1356-7.

这本书展示了 Microsoft Windows 编码风格的原则，但采用 Visual Basic 来介绍。

- Diomidis Spinellis, *Code Reading: The Open Source Perspective*, Addison Wesley, 2003, ISBN: 0-201-79940-5.

这本独一无二的书把编程风格反过来，要求读者学会正确地读代码来成为一个好的程序员。

- Dimitri van Heesch, *Doxygen*, www.stack.nl/~dimitri/doxygen/index.html.

这是一个高可配置的程序，可以从源代码和注释生成文档。

B.8 计算机体系结构

- David A. Patterson and John L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface (Second Edition)*, Morgan Kaufman, 1997, ISBN: 1-558-60428-6.
- John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach (Third Edition)*, Morgan Kaufman, 2002, ISBN: 1-558-60596-7.

这两本书提供了大多数软件工程师需要知道的有关计算机体系结构的所有信息。

B.9 效率

- Dov Bulka and David Mayhew, *Efficient C++: Performance Programming Techniques*, Addison Wesley, 1999, ISBN: 0-201-37950-3.

这是为数不多的专门介绍高效 C++编程的几本书之一。在此涵盖了语言级和设计级效率。

- GNU gprof, www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html.

提供了有关 gprof 测评工具的信息。

- Rational Software from IBM, www-306.ibm.com/software/rational.

Rational Quantify 是一个非常棒的（但不是免费的）测评工具。

B.10 测试

- Elfriede Dustin, Effective Software Testing: 50 Specific Ways to Improve Your Testing, Addison

Wesley, 2002, ISBN: 0-201-79429-2.

尽管这本书所面向的是质量保证专业人员，不过所有软件工程人员都能从软件测试过程的讨论获益。

B.11 调试

- Gnu DeBugger (GDB)，位于 www.gnu.org/software/gdb/gdb.html.

GDB 是一个很不错的符号调试工具。

- IBM 的 Rational Software，www 306.ibm.com/software/rational.

Rational Purify 是一个很好的（但不是免费的）的内存错误调试工具。

- Valgrind, at http://valgrind.kde.org.

这是面向 Linux 的一个开源内存调试工具。

B.12 分布式对象

- Jim Farley, *Java Distributed Computing*, O'Reilly, 1998, ISBN: 1-56592-206-9.

这本书提供了以 Java 为中心的分布式计算技术。

- Ron Hipschman, *How SETI@home Works*,
  setiathome.ssl.berkeley.edu/about_seti/about_seti_at_home_1.html.

介绍了 SETI@home 项目有趣的背景，这个项目使用分布式计算来分析来自外太空的数据。

- Sassafras Software, *General KeyServer Questions*, http://www.sassafras.com/faq/general.html

提供了 KeyServer 的有关信息，这是一个使用分布式计算来控制软件许可的应用。

B.12.1 CORBA

- 对象管理组织（Object Management Group）的 CORBA 网站位于 http://www.corba.org

CORBA 是对象管理组织（Object Management Group，OMG）的一个“产品”。这个网站包含了基本的背景信息，并且提供了所涉及具体标准的链接。

- Michi Henning and Steve Vinoski, Advanced CORBA Programming with C++, Addison Wesley, 1999, ISBN: 0-201-379270-9.

对于基于 Java 的 CORBA 有许多参考书，而介绍基于 C++的 CORBA 的书并不多。这本书主要强调 C++，尽管书名中有“高级”一词，但是 CORBA 初学者也可以轻松阅读。

B.12.2 XML 和 SOAP

- Ethan Cerami, *Web Services Essentials*, O'Reilly, 2002, ISBN: 0-596-00224-6.

这本书解释了新兴的 Web 服务概念，并用 Java 提供了使用 SOAP 完成分布式计算的例子。

- Erik T. Ray, *Learning XML (Second Edition)*, O'Reilly, 2003, ISBN: 0-596-00420-6.

这是事实上的 XML 标准参考，其中讨论了 XML 模式、XPath 和 XHTML 等相关技术。

- James Snell, Doug Tidwell, Pavel Kulchenko, *Programming Web Services with SOAP*, O'Reilly, 2001, ISBN: 0-596-00095-2.

这本书讨论了 SOAP 和相关技术，如 UDDI 和 WSD。分别用 Java、Perl、C#和 Visual Basic 提供了例子。

- Eric van der Vlist, *XML Schema*, O'Reilly, 2002, ISBN: 0-596-00252-1.

这本书解决了 XML 模式的一些难题，并讨论了 XML 语言的一些细微之处。

- Altova Software xmlspy, www.xmlspy.com.

提供了有关 Altova Software 的 xmlspy 软件包的信息。

B.13 设计模式

- Andrei Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison Wesley, 2001, ISBN: 0 201-70431-5.

为C++编程提供了一种充分利用可重用代码和模式的方法。

- Cunningham and Cunningham, *The Portland Pattern Repository*, www.c2.com/cgi/wiki? WelcomeVisitors.

你可以花上一整天浏览这个普及型网站来了解设计模式。

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN: 0 201-63361-2.

这也称为 GoF (Gang of Four) 书（因为有四位作者），这本书堪称是设计模式领域的经典之作。