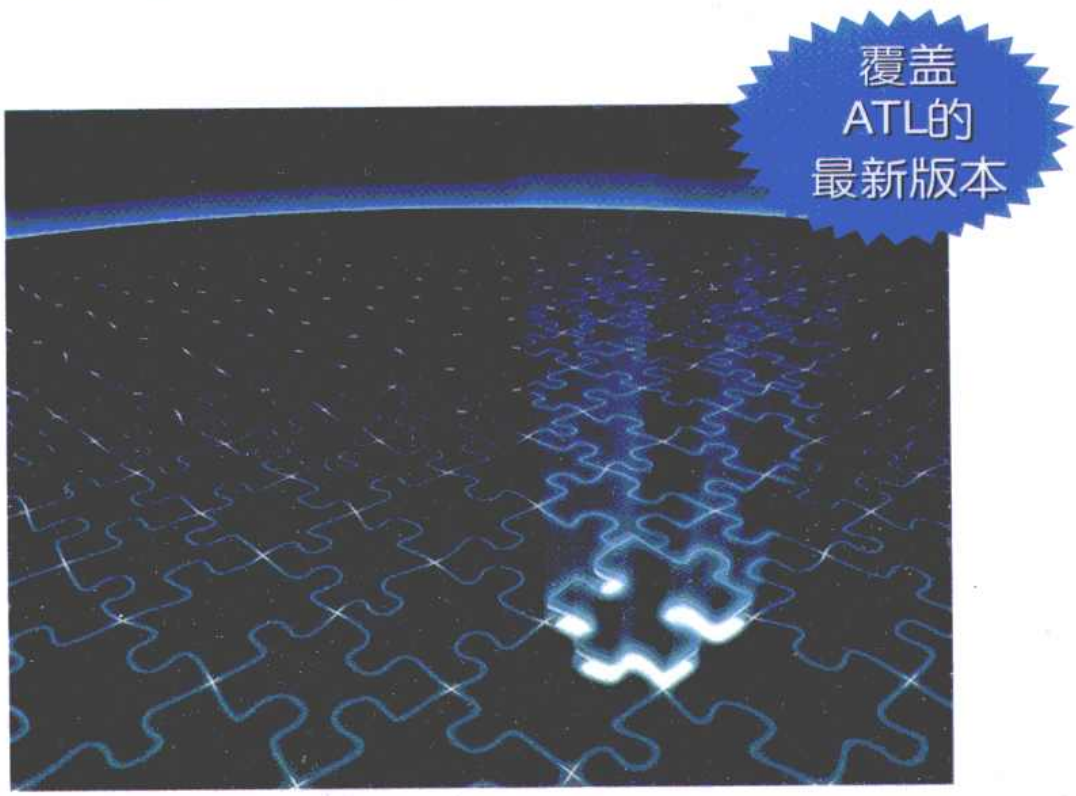
ET PUBLISHING 今日電子

美国IDG“高级开发工具”丛书

ATL开发指南

（第二版）

ATL Developer's Guide, 2nd Edition



[美] Tom Armstrong Ron Patton 著
董 梁 丁 杰 李长业 等译
董 梁 审校

- ATL——活动模块库，C++最常用的工具
- 设计Windows网络应用程序
- 掌握基于COM和COM+的程序开发技术
- 把握软件发展的必然趋势

IDG BOOKS WORLDWIDE M&T BOOKS PHEI
電子工業出版社
Publishing House of Electronics Industry
URL:http://www.phei.com.cn

美国 IDG“高级开发工具”丛书

ATL 开发指南

(第二版)

ATL Developer's Guide, 2nd Edition

[美] Tom Armstrong
Ron Patton 著

董 梁 丁 杰 李长业 等译

董 梁 审校

電子工業出版社
Publishing House of Electronics Industry
北京·Beijing

内 容 简 介

本书是介绍使用 ATL 进行软件开发的参考用书。全书分为十三章:第一章引入了 C++ 模板的概念;第二章详细介绍了组件对象模型(COM);第三章介绍了活动模板库(ATL)的概念和使用方法;第四章涉及接口的实现和线程之间的参数调度等细节问题;第五章探讨了代码复用的两种方法——包容和集合;第六章讨论了关于自动化的问题;第七章说明了事件和连接点的管理;第八章围绕 ActiveX 控件展开了详细的论述;第九章介绍了 COM 中的两种实体——枚举器和集合;第十章研究了 COM 里的线程管理;第十一章讨论了 OLE DB 同 ATL 之间的关系;第十二章介绍了两种 GUI 接口——对话框和视窗;最后一章解释了 COM + 的一些基本概念。

由于书中的内容非常庞杂,并且涉及到 Visual C++ 开发的一些内核问题,所以本书对读者有一定的要求。我们希望你在开始阅读之前,已经了解 COM、ActiveX 控件的基本知识,并且具有使用 Visual C++ 开发应用程序的实践经验。本书的主要对象是程序设计或开发人员,同时也可以作为大专院校计算机专业师生和计算机爱好者的参考资料。

ATL Developer's Guide and Edition by Tom Armstrong, with Ron Patton

图书在版编目(CIP)数据

ATL 开发指南/(美)阿姆斯特朗(Armstrong, T.)著;董梁译 .—2 版 .-北京:电子工业出版社,2000.11
(美国 IDG 高级开发工具)
ISBN 7-5053-6298-4
Ⅰ.A… Ⅱ.①阿…②董… Ⅲ.软件开发,ATL Ⅳ.TP311.52
中国版本图书馆 CIP 数据核字(2000) 第 56452 号

丛 书 名:美国 IDG"高级开发工具"丛书
书 名:**ATL 开发指南(第二版)**
著 者:[美]Tom Armstrong Ron Patton
译 者:董 梁 丁 杰 李长业 等
责任编辑:陈晓莉
特约编辑:申 本
印 刷 者:北京东光印刷厂
出版发行:电子工业出版社 URL:http: //www.phei.com.cn
北京市海淀区万寿路 173 信箱 邮编 100036
经 销:各地新华书店经销
开 本:787 × 1092 1 /16 印张: 33.125 字数: 852 千字
版 次:2001 年 5 月第 2 次印刷
书 号:ISBN 7-5053-6298-4 / TP·3402 著作权合同登记号: 图字:01-2000-2007
定 价:56.00 元

前　　言

* *

本书所讲述的是Microsoft推出的一个非常重要的开发工具:活动模板库(Active Template Library,ATL)。ATL是一个基于C++的框架,它主要用在基于Microsoft的组件对象模型(Component Object Model,COM)的软件开发上,使用它可以大大简化组件的开发过程并提高代码的效率。ATL同Microsoft的基础类(Microsoft Foundation Class,MFC)库有异曲同工之处,后者是在Windows平台下最常用的基于C++的框架。MFC的存在已经有六年之久,它已经成为了占主导地位的Windows应用程序框架。然而在很多情况下,ATL在开发基于Windows的软件上已经有后来居上的趋势。

MFC不会在一夜间黯然失色。事实上,完全可以把MFC和ATL结合在一起使用,但是随着时间的推移,现有的MFC特性将逐步融合在ATL里。Microsoft开发新的ATL框架主要是出于一个原因:为了适应一种新的应用程序开发架构——组件对象模型(COM)。

COM是Microsoft研制的一项系统级别的面向对象的技术,Microsoft的产品和工具软件里广泛使用了这一技术。COM提供了软件开发者所需的几项特征:首先是COM同语言的无关性,这一特性使程序开发者可以在Visual Basic、Delphi和几乎其他所有开发环境下重复使用C++模块;其次,COM提供了位置的透明性,这一性质使软件模块可以在分布式网络环境下的任何位置上运行。除此之外,COM还提供了标准的面向对象的特征,如封装、多态和继承等。COM标志着未来程序开发的方向,ATL则是开发者手中最为重要的工具,只有通过ATL我们才能充分利用COM这一工具,并领略未来技术的发展为我们的软件开发带来的新气象。

ATL的主要目的是创建小的、基于COM的软件模块,然后,再把这些模块组装成大的应用程序。当软件开发者将目光投入到这一新的、基于组件的开发模型上时,ATL也将逐渐被大家所接受。Microsoft也允许在非Windows的平台下传递COM,如UNIX、Sun公司的Solaris和DEC公司的VMS。总之,ATL是一种跨平台的COM开发工具。

ATL的发展历史源自于1996年秋季,当时Microsoft发行了ATL的1.0版,那时的ATL还只是作为Visual C++(当时是4.2版)的一个附件,用户可以从网上免费下载它。随后ATL 1.1版很快就投入市场。ATL的2.1版较早期版本增加了很多新功能,它仍然作为Visual C++ 5.0的一部分。ATL的最新版本——3.0版捆绑在Visual C++ 6.0里。同样,它也对ATL所提供的大量功能进行了升级。ATL的未来版本将可以提供当前的MFC所提供的所有功能。

本书的目的是帮助你理解和掌握基于Windows的一项最重要的开发环境:基于COM的应用程序开发环境。在这一新环境里,ATL是一个非常强大的工具。本书也将围绕这些方面的问题展开全面、深刻的介绍。

本书的章节编排

我建议你按照章节顺序阅读本书。如果你已经熟悉了 C++ 的模板和 COM ,也可以略过开始的两章,不过我还是希望你最好还是能通读一下这两章。第一章涉及基于 C++ 模板的程序开发。第二章对组件对象模型(Component Object Model,COM)进行了深入的讨论,并且引入了本书使用的一些示例。

除了第一章、第七章和第十三章之外,其他各章都分成了两个主要部分。第一部分为某一专题提供了一个概念性的、基于代码的讨论,第二部分则介绍如何根据这一技术专题开发一个或多个示例应用程序。采用这样的结构一方面有助于理解各种操作和处理的概念,另一方面是当你需要使用它们时,也可以迅速在书中找到它们的实现细则。

本书的内容

下面简要介绍一下每一章的内容:

第一章　使用 C++ 模板开发应用程序

ATL 在它的实现中广泛使用了模板,它是用来建立可复用的 C++ 类的工具,在这一章里详细介绍了模板的概念。

第二章　组件对象模型(COM)简介

这一章综述了 COM ,并对它的概念和使用进行了深入的解说。另外,本章还引入了一个简单的 C++ COM 客户程序和服务器程序。在后面的章节里,我将使用 ATL 和其他一些不同的 COM 技术再次实现该示例,并对示例程序的功能进行一定的增强。

第三章　活动模板库

ATL 是用来建立基于 COM 的应用程序的。本章介绍了 ATL ,并讨论了 ATL 基于模板的实现及相关的向导程序。本章还包括关于 ATL 的类和术语等方面的内容。

第四章　接口、接口定义语言和调度

从第四章起,我们逐步深入到 COM 的接口机制的内核。特别需要说明的是,在这一章里还将了解关于接口定义语言(Interface Definition Language,IDL)和进程之间参数调度(marshaling)的一些知识。除此之外,还在本章介绍了 COM 的一些细节问题,如错误处理、内存管理和基本的数据类型等。当然,这些问题都是 ATL 环境下所特有的。

第五章 包容与集合

COM 的一个重要特征是它对软件模块复用所提供的支持，而这一支持又是建立在二进制代码级别上的。在这一章里，我们将接触 COM 的二进制复用技术，它们分别是包容(Containment)和集合(Aggregation)。简述完这两种技术的主要特征之后，我们将讨论 ATL 是如何支持开发使用包容和(或)集合的组件。

第六章 自动化

自动化(Automation)是一项基于 COM 的技术，从用户的角度来看，几乎所有的 Windows 程序开发员都不会对它感到陌生。Visual Basic 里广泛使用了自动化技术，而 ActiveX 控件也通过自动化来把它的功能对用户公开。本章详细介绍了自动化技术，讨论了前绑定(early binding)和后绑定(late binding)之间的关系和区别，以及 ATL 处理自动化的方法。

第七章 事件和连接点

这一章包括了开发者使用 COM 开发实际工程的两个重要问题，它们分别是事件(event)和连接点(connection point)。现在，COM 接口的调用是同步进行的。然而，通过使用别的技术，如接口回调支持和连接点支持，COM 组件可以提供伪异步的行为特性。本章着重讨论的就是这个问题和 ATL 的实现。

第八章 ActiveX 控件

ActiveX控件在今后的Microsoft基于组件的程序设计中将扮演一个重要的角色。ActiveX 控件是基于 COM 的组件，它可以实现很多标准的、由 Microsoft 定义的接口。ActiveX 控件实际非常难以给出一个概念上的描述，这主要是因为它的定义在过去的几年里变化太快了。在这一章里，我将解释所谓的完全控件(full control)。完全控件是在一些流行的开发环境(如 Visual Basic)里使用的技术，它可以实现至少 20 种不同的 COM 接口。

第九章 COM 的枚举器和集合

软件开发中经常需要管理一系列相关的条目。COM 为插入和迭代项目列表提供了一个标准的技术，这就是枚举器对象(enumerator object)。一些计算机语言和开发工具，如 Visual Basic，都对这一概念进行了扩展，使它涵盖了组合的概念，而组合也正是以一种标准的方式把枚举器对象对用户公开。这一章就是围绕这些技术展开的。

第十章　COM 的线程管理

COM 里的线程管理是最让人难以理解的一个概念,它甚至让一些人感到恐惧。多线程的调度和管理是一个非常复杂的问题,再加上把 COM 引入到其中,无疑是雪上加霜。这一章首先介绍关于 COM 的线程管理的一些基本概念,然后再解释 ATL 对不同的 COM 线程管理模型所提供的支持。本章以一个支持异步方法调用的多线程 math 组件的示例作为结束。

第十一章　OLE DB 和 ATL

最新版本的 ATL 支持一个非常强大的 Microsoft 技术,它就是 OLE DB。现在,基于 Web 的电子商务应用软件发展得如火如荼,这也使得 OLE DB(具有 ADO 支持)必然将成为日后 Microsoft 程序开发者访问数据的标准。最新版本的 ATL 提供了极多的新类来协助开发人员实现一种通用的数据访问方法,这种方法是针对数据持有者的数据源的,同时也是基于 OLE DB 的。本章的主要内容也就是围绕这些技术细节展开。

第十二章　对话框和窗口

起初,ATL 的框架主要是集中在为 COM 组件提供一个非 GUI 的支持。作为一个程序员,如果你希望在自己的应用程序中有强大的视窗支持,那么 Microsoft 基础类(Microsoft Foundation Class, MFC)库或 Visual Basic 无疑是上佳之选。然而,在新版本的 ATL 里,Microsoft 也添加了类似于 MFC 的 GUI 功能。在这一章里,我将简要地向你介绍 ATL 所提供的视窗支持和对话框支持。

第十三章　COM + 入门

COM 作为一项工作站级别(workstation-level)的组件技术已经问世已久。在 Windows NT 4 里出现了分布式版本的 COM ,称为 DCOM 。该技术和 Microsoft Transaction Server(MTS)结合在一起提供了服务器端的组件服务,并且解决了以前 DCOM 中出现的很多缺陷。此后,COM + 又成功地开发出来,并且集成到了 Windows 2000 里。这一项技术把 COM、DCOM、MTS 更为紧密地结合在了一起,因此它极具商业潜质。本章将带你一起浏览一下这些新问世的技术。

本书的示例程序

由于万维网的广泛普及和它的低廉花费,所以现在的很多科技图书已经不再带有(或需要)一张刻有示例代码的光盘了。你可以从我们的网站上下载所有的示例和其他补充材料。文中的示例程序大部分是使用C ++ 和ATL开发,它们组合在一起构成了一个庞大的 Visual C ++ 工作区(ATL_Examples.dsw)。还有个别示例程序是使用 Visual Basic 开发的,它

们也组成了一个 Visual Basic 的工作区(VB_Projects.vbg)。你只要访问下面的 URL 就可以了解各个方面的细节问题:

http://www.widgetware.com

意见和出错报告

我欢迎读者提出建设性的意见、建议和 bug 报告,你可以使用下面的网址与我联系。你也可以浏览我的 Web 站点,在该站点里包含了示例程序、常见问题解答(FAQ)、到其他 COM(+)/OLE/ActiveX 站点的快速连接、讨论以及其他关于 COM(+)、ATL 和 ActiveX 技术的资讯。

我的 Web 站点是:tom@WidgetWare.com。

第　一　章
使用 C++模板开发应用程序

* *

本章内容:

- ◆ C++模板简介
- ◆ 通过一个通用的堆栈类示例来说明模板
- ◆ C++模板成员函数的语法结构
- ◆ 在类中指定模板参数
- ◆ C++模板的主要目标:实现代码的重复使用
- ◆ 如何在 ATL 框架中使用模板

* *

正如其名称所暗示的,ATL(Active Template Library,活动模板库)使用 C++模板作为自己的实现的基本特性。在本章里,我们将首先快速浏览一下 C++的模板,并讨论如何使用模板开发应用程序。这种借助模板开发程序的方式会与我们以前常用的方法有一定的不同。

1.1　模板简介

模板是 C++语言中颇为新颖的一个概念。它们提供了一种通用的方法来开发可重用的代码,即可以创建参数化的 C++类型。模板分为两种类型:函数模板(Function Template)和类模板(Class Template)。函数模板的用法同 C++预处理器的用法有一定的类似之处,它们都提供编译代码过程中的文本替换功能,但前者可以对类型进行一定的保护。类模板使你可以编写通用的、类型安全的类。

1.1.1　函数模板

为了更好地说明函数模板(Function Template),我们先分析一下 C++中使用预处理器实现一个通用的 MAX(求最大值)函数。如果不使用模板,一个类型安全(type-safe)的 MAX 函数可以采用以下的方法实现:

```
long MAX( long a, long b )
```

```
{
    if ( a > b )
        return a;
    else
        return b;
}
double MAX( double a, double b )
{
    if ( a > b )
        return a;
    else
        return b;
}
```

对于传递给 MAX 函数的每一种类型,你都必须提供一个显式的实现(即在函数中明确指明参数的类型)。如果你使用了函数模板,就可以使用如下的代码完成同样的功能:

```
template < class Type >
Type MAX( Type a, Type b )
{
    if ( a > b )
        return a;
    else
        return b;
}
```

如果你在此之前还没有接触过模板,那么上面的代码段可能会看起来有点奇怪。关键字 template 指明该函数是一个模板。在大于号和小于号之间的字符指定了参数化的类型,在示例中由关键字 class 表示。Type 参数将在编译时被模板用户指定的类型所替代。下面就是在调用时的用法:

```
int main( int argc, char * argv[] )
{
    int    iMax = MAX< int > ( 10, 12 );
    long   lMax = MAX< long > ( 10, 12 );
    double dMax = MAX< double > ( 10.0, 12.0 );
    return 0;
}
```

这里所使用的语法结构可能也有一点晦涩难解,但是如果我们仔细分析一下后就将了解其中的含义。当我们调用该函数的时候,需要把类型作为参数提供给函数。模板是一种

编译时(compile-time)结构,并且会根据所指定的不同类型进行扩展——这正如早期的 C 预处理器宏的作用一样。例如,上面的代码将在调用函数中进行如下的扩展:

```
int main( int argc, char * argv[] )
{
    int MAXint( int a, int b )
    {
        if ( a > b )
            return a;
        else
            return b;
    }
    int iMax = MAXint( 10, 12 );
    ...
}
```

上面的代码只是针对一种函数类型的扩展,但是想必你已经了解了其中的内涵。函数模板既提供了 C 预处理器的灵活性,同时也提供了在编写类型通用函数时的类型保护功能,从而可以为各种类型编写一个通用的函数。你只需要编写一次函数,在调用时指定不同的函数类型就可以多次重复使用它。类模板在类级别(class level)上也具有相同的灵活性。

1.1.2　类模板

类模板(Class Template)与函数模板类似,它们都使用户可以在类实现的时候指定其类型。你可以开发一个通用类型使它在编译时可以控制用户定义的不同类型,从而令类的实现具有充分的类型安全特征。在基于模板开发的程序里最困难的一个方面是如何指明语法结构。

1.2　基于模板的堆栈类

为了解释基于模板编程的基本特征,我在这里将建立一个完成堆栈功能的示例类。这种类可能你以前已经见得多了,但是本文的例子更加短小精悍,你可以把精力都放在模板的使用上。

首先,堆栈提供了一个 LIFO(先入后出,后入先出)结构,栈里面存储的是相同类型的元素。在硬件里,堆栈被大量地使用,在软件里堆栈也非常有用。我们在这里想要做的是建立一个堆栈类,使它可以存储任何类型的数据。如果不使用 C++模板,这几乎是不可能实现的——至少无法采用直接的方式实现。如果使用了模板,我们就可以编写一段适合各种数据类型的代码。下面就是整型数堆栈的代码:

```
class StackInt
```

```
};
public:
    StackInt()
    {
        m_sPos = 0;
    }
    StackInt() {}

    void        Push( int iValue );
    int         Pop();

    bool IsEmpty()
    {
        return( m_sPos == 0 );
    }

    bool IsFull()
    {
        return( m_sPos == 100 );
    }

    bool HasElements()
    {
        return( m_sPos != 0 );
    }

private:
    int     m_data[100];
    short   m_sPos;
};

void StackInt::Push( int iValue )
{
    m_data[ sPos++ ] = iValue;
}

int StackInt::Pop()
{
    return m_data[ --sPos ];
}
```

这是一个有效、简练的堆栈类的实现。它具有典型的 Push(进栈)和 Pop(出栈)方法,以

及其他的一些功能，例如使用户可以判断堆栈的状态(栈是否为空、栈是否已满等)。该堆栈类的用法如下所示：

```
int MAIN( int argc, char * argv[] )
{
    StackInt stack;
    stack.Push( 100 );
    stack.Push( 200 );
    stack.Push( 300 );
    while( stack.HasElements() )
    {
        cout < < stack.Pop() < < endl;
    }
    return 0;
}
```

该类非常容易使用。然而，它里面还是存在一些问题。首先，程序里没有什么错误检测能力，当然这可能是出于令代码短小和易于跟踪的角度来考虑的。其次，该堆栈只能支持整型数。第三，堆栈的大小已经显式地指定为100。如果用户要使用大的堆栈，它将无法满足需要。现在，我们使用一个参数化的构造函数，代码如下所示：

```
class StackInt
{
public:
    StackInt()
    {
        m_sSize = 100;
        m_data = new int[ m_sSize ];
        m_sPos = 0;
    }
    StackInt( short sSize )
    {
        m_sSize = sSize;
        m_data = new int[ m_sSize ];
        m_sPos = 0;
    }
    StackInt() {}
    void    Push( int iValue );
    int     Pop();
    bool    IsEmpty()
    {
```

```
        return( m_sPos == 0 );
    };
    bool HasElements()
    {
        return( m_sPos != 0 );
    }
    bool IsFull()
    {
        return( m_sPos == m_sSize );
    }
private:
    short m_sSize;
    int* m_data;
    short m_sPos;
};

void StackInt::Push( int iValue )
{
    m_data[ m_sPos++ ] = iValue;
}

int StackInt::Pop()
{
    return m_data[ --m_sPos ];
}
```

其中,参数化的构造函数使该堆栈类变得更为通用,用户可以自行定义堆栈的大小。然而,另外一个问题仍然存在,该堆栈只能支持整型数。如果用户想要对双精度数、字符串、其他类,如 MFC(Microsoft Foundation Classes,Microsoft 基础类)的 CWnd 类或用户自定义的类 CBankAccount 等进行堆栈操作时,该如何实现呢? 这时就需要你完全改写类的代码了。这一工作实际并不费劲,只要执行一些剪切和粘贴操作即可。

例如,如果要建立一个适合于双精度数值的堆栈类,可以采用如下的编码:

```
class StackDouble
{
public:
    StackDouble()
    {
        m_sSize = 100;
        m_data = new double[ m_sSize ];
        m_sPos = 0;
```

```
    }
    StackDouble( short sSize )
    {
        m_sSize = sSize;
        m_data = new double[ m_sSize ];
        m_sPos = 0;
    }
    ~StackDouble() {}

    void Push( double dValue );
    double Pop();

    bool IsEmpty()
    {
        return( m_sPos == 0 );
    }
    bool HasElements()
    {
        return( m_sPos != 0 );
    }
    bool IsFull()
    {
        return( m_sPos == m_sSize );
    }

private:
    short m_sSize;
    double * m_data;
    short m_sPos;
};
```

现在，你就实现了一个对双精度数字进行堆栈的类。更改的代码已经注明(在程序段中以黑体标出)。正如你所看到的，我只更改了三项：类的名称、Push 方法里的参数类型和数组中实际存储的类型。应该讲这样的更改并不困难。那么编译器是否可以为我们提供类型方面的帮助呢？答案是肯定的，这恰恰就是 C++ 模板所能提供的功能。模板在类的级别上为用户提供了一个编译时(compile-time)的替换机制。

通过在编译的时候进行类型(和常量)替换，你可以借助模板建立通用型的类，使它不必只能操作某一指定的数据类型，而可以令它在类实现的时候根据用户提供的数据类型进行调整。编译时，模板将根据用户指定的类型进行“扩展”，生成一个全新的、并且类型安全的类。下面我们还是来看一下堆栈类示例，不过这一次，我使用了 C++ 的模板：

```
template < class T >
class Stack
{
public:
    Stack()
    {
        m_sPos = 0;
    }
    ~Stack() {}

    void Push( T value );
    T    Pop();

    bool IsEmpty()
    {
        return( m_sPos = = 0 );
    }
    bool HasElements()
    {
        return( m_sPos ! = 0 );
    }
    bool IsFull()
    {
        return( m_sPos = = 100 );
    }

private:
    T m_data[100];
    short m_sPos;
};
```

类模板的语法结构和函数模板的语法结构非常相似。使用 template 关键字是为了声明该类是一个模板类。接着,我给出一个或多个模板参数。每一个参数都可以是template < class T > 里 class 关键字所指定的类型,也可以是一个根据已有的某一有效类型声明的常量(constant)。关于参数化常量的问题我们将在下一个例子中讨论。

代码在编译时,类声明语句中的语法将使用你所指定的类型来替换这里的"类型"(在该例子里为 T)。在上面的例子里,你需要替换数组的类型、Push 和 Pop 方法的参数类型以及返回值类型。

1.3 成员函数的实现

在类声明语句之外实现成员函数(member function,亦称为方法——method)的语法有一