

第17章 用C++建立ASP组件

我们已看到用COM服务器组件对于建立一个基于网络的应用程序的重要性，但问题不在于是否建立它们，而在于用什么语言去创建。一种选择是用 C++。

人们对C++有许多不同的看法，保守的 C++程序员坚持用其他语言创建 COM组件，他们认为只有真正的程序员使用 C++。另一方面，VB程序员认为C++是一种很难掌握和使用的语言，没有必要增加编程时间和进行艰难的尝试。Java程序员认为他们比 C++程序员强，因为James Gosling(Java的发明者)吸收了许多语言(包括C++)的优点发明了Java，本章和下一章的目的就是消除对C++的偏见和错误概念。

本章集中介绍用 C++ 建立服务器组件，不讲述 C++ 语言，如果想学 C++，请参阅Ivor.Horton著的《Beginning Visual C++6》，Wrox 出版，书号为ISBN 1-861000-88-X。

本章的主要内容有：

- C++简史。
- 使用C++原因。
- 从VB移植到C++。
- ATL、STL和MFC介绍。
- 建立一个COM组件。
- 错误处理与调试。

更重要的是应该记住，现在，不只是在用 C++建立组件，可以使用 Visual C++中可用的任何工具，使得建立过程更加容易。先从 C++的起源谈起。

17.1 C++语言

在决定是否使用C++语言之前，最好是搞清楚这种语言的实质，让我们看一下 C++的历史和现状。

17.1.1 C++简史

刚开始形成的是C语言，那些想建立更快更有效的代码的程序员非常欣赏 C语言，有一位名叫Bjarne Stroustrup的人却不满足于仅仅是生产快速代码，他想创建面向对象的 C语言编程。他开始对C语言的内核进行必要的修改，使其能满足面向对象模型的要求。C++从此产生。

Bjarne Stroustrup是C++的最初设计者和实现者。它自诞生以来，经过开发和扩充已成一种完全成熟的编程语言。现在 C++已由ANSI、BSI、DIN、其他几个国家标准机构和 ISO定为标准。ISO标准于1997年11月4日经投票正式通过。

C++标准演变了许多年。C++模板是近几年来对此语言的一种扩展，模板是根据类型参数来产生函数和类的机制，有时也称模板为“参数化的类型”。使用模板，可以设计一个对许多类型的数据进行操作的类，而不需要为每个类型的数据建立一个单独的类。标准模板库(Standard Template Library, STL)和微软的活动模板库(Active Template Library, ATL)都基于

这个C++语言扩展。

C++标准可分为两部分，C++语言本身和C++标准库。C++标准库对于Visual C++是相当新的，实际上微软只是在发布 Visual C++ 5.0时去除了一些“bug”。标准库提供了标准的输入/输出、字符串、容器(如矢量、列表和映射等)、非数值运算(如排序、搜索和合并等)和对数值计算的支持。应该说，C/C++包含了相对少的关键字，而且很多最有用的函数都来源于库，C++标准库实现容器和算法的部分就是STL。

STL是数据结构和算法的一个框架，数据结构包括矢量、列表和映射等，算法包括这些数据结构的查找、拷贝和排序等。1994年7月，ANSI/ISO C++标准委员会投票决定接受STL为C++标准库的一部分，这个建议是根据 Alex Stepanov、Meng Lee和David Musser这三人的编程和软件库研究提出的。STL的产生是为了满足通用性的设计目标，而不是为了提高性能。

那么微软对C++标准的态度怎么样？微软运行VC++与Plum-Hall C++，想比较得到的分数在92%和93%之间。为什么不是100%的一个原因是跟踪这个标准并同时建立一个编译器比较困难，微软也考虑了对现有编码兼容性的重要性，有时他们不得不偏离标准以保持这个兼容性。

17.1.2 使用C++的原因

应该有充分的理由使用C++创建服务器组件，而不只是为了给上司一个好印象才使用C++。如果以前没用过C++，你必须要尽力学习。

1. 性能

性能有个两方面，算法速度和机器代码效率。一个算法可以定义为数据通过系统的概念化的路径，它描述一些点，在这些点上，数据能够被操作并可转换产生某个结果。例如，一个算法定义为获取一个字符串，计算字符串中的字符个数，并作为结果返回的过程。算法与语言是独立的，所以在编程之前必须设计算法，编写一个快速程序的第一个步骤是设计良好的算法，能以最少的操作步骤得出问题的答案。第二个步是选择语言，这也影响程序的速度。

从性能的角度考虑，用汇编语言编写程序是最佳的选择，它是计算机能理解的自然语言。但是，几乎没有人用汇编语言编写完整的程序，因为这样做极其乏味。另一个最佳的选择是C语言。然而，由VC++提供的所有工具都产生C++，而不是C。使用VC++的向导可以生成大量的使用代码，而不必人工地编写代码。从编写程序的难易程度和程序的性能综合考虑，C++是最佳的选择。

C++性能良好，因为它被编译为机器代码。对于VB Script和Java等语言，代码在运行时由程序解释，而且每次运行程序时都要将代码转换为机器码，这样做效率比较低，不仅仅是已编译过的C++程序运行得较快，而且微软C++编译器已存在多年。这意味着微软的编译器程序员已经把许多优点集中到编译器上，以致于它能产生非常高效的机器码。因为C++是编译语言，而且非常自然，比VB更接近机器代码，所以由C++编译器产生的代码一定比VB的编译代码效率更高。

2. 错误处理

一个好的程序与一个伟大的程序的区别就是其是否具有良好的错误处理支持。实际上，如果在实现中首先进行错误处理，而不是在最后才进行，那么整个程序的开发和测试过程会更加完美。但是，错误处理只能与语言所支持的内容相一致。

VBScript具有基本的错误处理支持功能。在默认情况下，不能捕获 VBScript中的错误。每次怀疑产生错误时，要调用 On Error Resume Next功能，并检查Error对象。

而C++中的错误处理比较好，这是因为有“异常处理”，本章的后面部分将详细介绍。

3. 最小的依赖性

正如上面所说，C++是一种编译语言，即C++代码在执行之前已转换为机器码。只要此代码不依赖于外部的动态链接库(DLL)，C++就可以在不需要安装额外程序的情况下移动到运行同样操作系统的其他机器和微处理器上，而移动Java程序时需要先安装Java运行期库。

4. 利用现有的代码

由于C和C++已经存在许多年了，现在有许多可利用的代码，你的服务器组件可以使用现有的C/C++代码或库。例如统计库和到老系统的C接口。

5. 最大化COM特征

COM与C++很接近，实际上，Don Box(COM的权威)在他的《Essential COM》一书的第一章写道：“COM就是更好的C++”。他说明了COM规范是如何从C++语言规律中产生出来的。通过理解C++，会对COM有更深入的理解。

某些语言不能利用所有的COM特征，而在C++中，几乎可以使用所有的COM特征。

17.1.3 不使用C++的原因

知道什么时候使用C++是重要的，同样，知道什么时候不使用C++也是重要的。想像一下那些长期维护代码的人，如果他们中没有一些C++程序员支持C++，那么开发者们不得不把眼光转向另外一些他们熟悉的语言。

改变C++组件时，为了看到这些改变的结果，必须重新编译该组件代码，这会花费很长的开发时间。C++不能像ASP页面代码那样，只使用记事本，改变代码的一行，重新装载而得到结果。因此，如果某些工作需要经常变化(如原型)，不要用C++。

在C++中，对一些致命的错误不能获得更多的保护，写一个使组件崩溃的代码是很容易的。这是为了提供快速代码而付出的代价，C++不会停下来去检查代码是否按设计运行能否使程序不崩溃依赖于开发者的技巧。如果在这方面花的时间较少或刚刚学习C++，最好不要使用C++。等到已经意识到C++中所有容易犯的错误，而且在检测组件之前花了许多时间，才可以使用C++，如果想很快、很容易地建立一个组件，而且也不考虑该组件的执行速度，那么使用VB吧！

17.1.4 把ASP技巧转到C++上

学习新东西的最好方法就是利用现有的技巧。对于ASP开发者来说，已经学习了C++所要求的许多技巧，特别是，JScript语法和ActiveX或COM的面向对象编程的概念。

1. JScript

大部分ASP开发者都用JScript在浏览器上使用DHTML。JScript的语法与C非常相似，所以，如果懂得JScript，那么就懂得基本的C语法。当然，只是C++语法的子集。C++有许多额外的语法来支持面向对象编程，这就是我们下一步要做的。

2. 面向对象编程

如果你在VB中使用过类(class)，则对任何COM对象和文档对象模型(Document Object

Model, DOM)都应熟悉,因为已经有了面向对象编程 (OOP)的概念。在前面已经说过, C和C++的区别是C++支持面向对象编程。

17.2 VC++库

大量的程序员都尽可能多地利用现有的代码。程序员经常购买那些包装成库的代码,而且许多成功的公司正是靠生产真正优秀的代码库而发展起来的,例如 Rogue Wave Software(www.roguewave.com)。

当C语言流行时,代码库就是函数库。例如,可以购买一个数学库,该库含有完成微积分和代数运算的函数。通常,可以在程序代码中使用一个包含文件来指定一个函数库,可以静态或动态地链接这个函数库。

静态链接意味着库代码直接集成到程序中。在这种情况下,程序不依赖于其他文件,但文件的大小可能很大。动态链接意味着程序有库的版本信息,其代码存放于一个单独的文件中,这个文件称为动态链接库 (DLL)。只有程序运行期间调用 DLL 中的函数时,才加载 DLL 到内存。DLL 作为一个独立的实体存在于内存中,可以同时被多个程序访问。

出现C++后,函数库转变为类库。两者的区别在于函数库只包含一系列函数,而类库是用面向对象编程的原理设计的,例如,可以为数据结构做一个类库,该库包括一个链接列表的类。如果使用一个函数库代替,那么链接列表会独立于操作它的函数。另一方面,使用类库时,链接列表和操作它的函数存在于同一个文件的同一个类中。然而,正如使用函数库一样,使用类库涉及到包含文件和链接一个静态库。

最近,已经从类库发展到模板库。其原因是 C++ 编译器现在已经能够处理 C++ 模板。模板库提供了一系列优于类库的优点。要使用模板库,只要在程序中加入包含文件即可,不用链接到库,因为所有的库代码已经包含在该包含文件中了。

这听起来不是非常高效,早期确实如此,因为每个包含有模板文件的文件都要得到一个单独的代码拷贝。因此,编译后的程序会很大。但是,为了适应模板,编译器已经优化,使得这种方法非常高效。

VC有三个供开发者使用的库:一个类库 (MFC),二个模板库 (ATL和STL)。

17.2.1 微软基础类库

微软基础类库 (MFC) 是微软为帮助 C++ 开发者而建立的类库,含有支持各种不同功能的许多类,包括实现复杂的数据结构、操作 Windows 和访问数据库等。但是,是否应在服务器组件中使用 MFC? 回答是否定的!

这很明智的,不要在服务器组件中使用 MFC,服务器组件不能访问 Win 3.2,所以不需要 MFC 库,数据库支持最好用 ATL 实现。

ATL 支持 ADO 和 OLE DB,而 MFC 不支持。STL 与 MFC 相比有更好的数据结构支持。即使最常用的 MFC 类,如 CString,在标准 C 库中也有等价的类。偶尔会使用 MFC 类,如 CSocket,但它也可被一种 Win 32 API 代替。

MFC 是用来建立单独 Windows 应用程序的,不是用于轻量的 COM 组件。只有在与现有的使用 MFC 的代码接口时应使用 MFC 类。其他情况使用 ATL 和 STL 类,这样可以使组件更小且运行得更快。

17.2.2 活动模板库

你可以判断出活动模板库(ATL)比MFC新,因为ATL中有“Active”这个词,另一种叫法是“微软模板库”(Microsoft Template Library, MTL)。

ATL用来快速、简易地创建COM组件,而且只占用较少的内存,组件的关键部分是由C++模板完成的。实际上,ATL的设计在模板的使用上沿用了STL。

C++模板提供一种类型安全的方法来实现一般的接口。很幸运,微软提供了一些产生ATL代码的向导,所以在建立简单组件时,不必了解模板,甚至不用知道ActiveX的所有细节。ATL另一个优点是不依靠DLL,就是说在装载应用程序时,不必装载DLL和静态地链接,这两项会增加对内存空间的需求。

ATL使COM的开发更加容易,ATL同样支持通过OLE DB消费者和提供者的数据访问,还有一些特殊的用途,如支持建立微软管理控制台(MMC)插件组件。

与ATL比较,MFC依赖于额外的运行期DLL或静态链接,所以产生了比ATL大而慢的代码。因此,在建立COM组件时,如果没有用户界面,那么选择ATL比MFC好。

17.2.3 标准模板库

标准模板库(STL)对于VC++程序员来说是使用得最不充分的有用工具之一,但它是唯一一个标准的和跨平台的VC++库。

使用STL不多的原因之一是,微软的STL文献难以理解。现在没有为VC++程序员而编写的STL书籍,由Musser和Sauni编写的《STL Tutorial and Reference Guide》是一本关于STL的好书。

STL对C++组件开发很有用,原因是包含了许多数据结构和算法。很多情况下,服务器组件的用途就是处理数据,STL非常合适这一用途。

微软最初实现STL标准是在VC++ 4.2中,但是,这个STL实现做得很差,以致不能使用。一些人通过使用HP或SGI提供的STL实现解决这个问题。既然STL是一种ISO/ANSI标准,它就是跨平台的,只要编译器支持STL所要求的结构就可以。然而,微软更新的STL实现(在VC++ 5.0和6.0中的提供),比以前的STL更稳定、更容易使用。

STL是对ATL的完美补充,ATL没有数据结构和算法。而且,STL和ATL有相似的实现方法,即使用C++模板,所以配合得很好,而且两者都不需要链接额外的动态或静态链接库,这样就会使组件更小,使它们对DLL的依赖更小。

下面让我们对STL的关键概念做些介绍,在建立服务器组件时要用到这些概念。

1. STL遍历器

使用C编程时,一般使用两种方法来遍历数据结构:使用一个下标或递增一个指针,这两种方法达到相同的效果。但是编写一个通用的算法来处理不同的类型是不可能的,用STL遍历器能解决这个问题,下面的例子是最好的说明。

下面的程序展示了两种标准C的遍历方法,最后是STL的遍历方法。该程序的任务是计算一个字符串中“t”的个数。

```
string      mySTLStr;  
const char *myStr;  
int         tCount = 0;
```



```
int            stringLength;

mySTLStr = "test";
myStr = mySTLStr.c_str();
stringLength = strlen( myStr );

/* Iteration using a subscript. */
for ( int i = 0; i < stringLength; i++ )
    if ( myStr[i] == 't' )
        tCount++;

tCount = 0;

/* Iteration using a pointer. */
for ( const char *myPtr = myStr; *myPtr != '\0'; myPtr++ )
    if ( *myPtr == 't' )
        tCount++;

tCount = 0;

/* Iteration using an iterator. */
for(string::iterator iter = mySTLStr.begin(); iter != mySTLStr.end(); iter++ )
    if ( *iter == 't' )
        tCount++;
```

使用遍历器解决了使用指针或下标时出现的问题。指针和下标依赖于数据，这些数据在内存中相邻存放，而遍历器隐藏数据，使数据结构的使用者不必了解数据存储的顺序。隐藏数据结构允许创建通用的算法，这种算法能够与各种数据结构一起工作，例如与容器一起工作。

为了算法的效率，STL定义了五种遍历器的类：输入、输出、前向、双向和随机访问。算法针对特定的遍历方法设计，因此具有最优的性能。

遍历器类似于指针，移动到下一条目和前一条目分别使用操作符“++”和“--”，同样，获得一个条目的值使用操作符“*”。

2. STL映射

一个映射就是一个排好序的容器，这里关心的是能够根据存储在条目中的键以尽可能快的速度检索条目，而不是关心使条目以线性的顺序存储在容器中。

STL排序容器有set、multiset、map和multimap。set用于条目集合，map用于一个条目与另一条目的关联。由于使用双向遍历器，排序容器支持以线性顺序移动数据条目，尽管排序容器的目的是用于键访问。

一个映射容器支持基于单独键的快速数据检索，这个键在映射中是唯一的，映射对于稀疏数据具有节省大量存储空间和减少许多计算时间的潜力，在后面的叙述中你将看到。键和值是以pair类型存储的，这个类型有两个成员：first和second。检索first返回键，检索second返回键的值。映射的遍历器就是pair类型的遍历器。

在本章后面将看到使用映射的例子。

3. STL矢量

一个矢量就是序列容器的一个例子，其他序列容器是deque和list。它们具有相似的功能，基本的区别在于性能。例如，除了在deque的开始处插入和删除的速度比在vector中快之外，一个deque与一个vector没什么区别。不同的问题要用不同的方式来访问数据。矢量能对不同长度的数据提供快速随机的访问，在此序列的末尾插入和删除时较快，而在其他在地方则比较慢。

4. STL算法

STL算法根据对其操作的数据结构的影响方式和相互作用进行分类。STL算法与其操作的STL数据结构的接口是STL遍历器。STL遍历器扮演了一个重要角色，因为它限制了算法怎样访问数据。许多算法接收一个函数作为其接口的一部分，这个函数的参数返回一个布尔值，表示一种情况。用函数参数可以定制算法。

一些STL算法允许函数作为参数传递。多数情况下，这些函数完成一个比较，并且返回布尔值。例如，一个排序算法使用一个函数作为参数来比较两个值，这个函数根据某些准则比较一个值是否比另一个值小。不同的函数有不同的准则，从而导致数据以不同的顺序排列。

更有趣的是，将一个函数对象(而不是函数)作为传递参数。函数对象可以包含状态，这种状态可以启用存在于某处的比较规则。使用的比较规则依赖于对象的状态。在本章的后面有一个排序算法的比较函数的例子，以这种方式实现。

5. STL算法的分类

STL算法可分为四大类。非变异序列算法，操作不改变数据结构内容；变异序列算法则改变数据结构内容；相关排序算法能用于排序、合并和二进制查找等多种算法。最后，STL具有广义的数值算法，用于计算数据结构中的元素。

- 非变异序列算法：不直接改变其操作的数据结构的元素。通常，它们查找数据结构中的元素，检查序列元素的等式，计算序列元素的个数。find和count是非变异序列算法的例子，find算法在数据结构的某个范围中移动，查找第一个等于给定值的遍历器。而count算法计算数据结构中等于给定值的元素的个数。
- 变异序列算法：修改其操作的数据结构中的元素。这些算法在容器中拷贝、替换、转换、删除和循环移动元素。fill算法是这种算法的一个例子，它的功能是把给定值的拷贝填充到一个序列范围中的所有位置。
- 相关排序算法：排序、合并和查找元素。同样，能够对排好序的序列进行设置操作，merge算法是一个例子，它将两个已排序范围内的元素放到一个范围内，并且结果不发生重叠。

广义数值算法的例子是accumulate，这个算法对数据结构内的某一指定范围的值求和。

6. 使用算法的遍历器

前面已经讲过，STL定义了五种遍历器的类：输入、输出、前向、双向和随机访问，算法针对特定的遍历方法设计，以确保优化性能。

例如，find算法可以用于在各种数据结构中查找值，算法并不关心容器是一个矢量，还是一个列表，还是一个数组。唯一的条件是数据结构必须支持InputIterator，InputIterator的设计确保它支持的操作的工作效率。find算法只需要InputIterator支持的操作，因此，find算法能高效地工作。

一些算法要求功能更强的遍历器：如sort需要随机访问遍历器(RandomAccessIterator)。提供随机访问是以降低性能为代价的，并影响其他操作，例如在容器中间插入元素的操作。

现在已经你有充分的理论知识，下面建立一个组件。

17.3 建立一个C++服务器组件

C++是标准化的计算机语言，不属于任何人，而属于一个标准委员会。STL是支持数据结

构和算法的 C++ 扩展。ATL 是微软拥有和维护的模板库，使得 COM 编程更容易。综合这些技术形成了创建 COM 组件的一种有效方法，这些 COM 组件用于 ASP 页面。

下面用所有这些技术创建一个 COM 对象，你将看到 VC++ 6.0 的向导如何提供大量代码，因此，可以把注意力集中在解决问题上，而不是担心具体的编程细节。

17.3.1 问题

表现数据的最普通方法是表，列代表字段的类型，每一行是一条记录，拥有字段的值。在文本文件中，表通常由用逗号分开的值 (comma-separated values，CSV) 组成。

我们要创建的 COM 组件以 CSV 数据作为输入，高效地存储它，并提供访问函数去检索它。这些数据在 COM 组件中以 STL 数据结构表示。在以后部分中，我们会看到怎样用 STL 算法去处理这些数据。另外，在下一章，将介绍怎样在数据库中存储这些数

据。为了便于说明，假设数据在一个稀疏表中。第一行的字段是列标题，接下来的是一条条数据记录，记录的每个字段对齐于列标题。逗号隔离字段，换行符 (\n) 隔离行，空的字段用两个逗号表示，即“，，”。

表 17-1 是一个展开的表的例子。导出时，逗号会隔离每一个字段。

表 17-1 示例表

Name	Group	Instrument
Jim Morrison	The Doors	Vocals
Keith Moon	The Who	Drums
Jimi Hendrix		Guitar & Vocals
Lenny Kravitz		Everything
Robert Plant	Led Zeppelin	Vocals

17.3.2 设计

这个组件的设计目的是使数据的存储空间和访问时间最小。由于数据有可能是稀疏的，即许多字段是空的，这就有可能使数据的存储空间最小化。可以通过数值 (基于零的索引) 访问数据的行，可以通过字段名访问数据的列。例如，要得到表 17-1 中 Keith Moon 的 Instrument，可以调用 GetField (1, "Instrument")。

完成以上工作的工具是 STL 的 vector 和 map 数据结构，这些数据结构是容器，就是说，它们是包含其他对象的一个集合的对象。为了访问集合中的对象，使用 STL 遍历器。

17.3.3 实现

现在你对这个组件的功能已经有了概念，我们将按下面的步骤实现它：

- 创建包含组件的 DLL。
- 创建组件。
- 增加属性。
- 增加方法。

1. 创建 DLL 和一个组件

选择 ATL COM App Wizard，创建一个新的 VC++ 项目，然后命名为 ASPComponents。使

用默认的服务器类型(DLL)，不选中复选框，如图17-1所示。

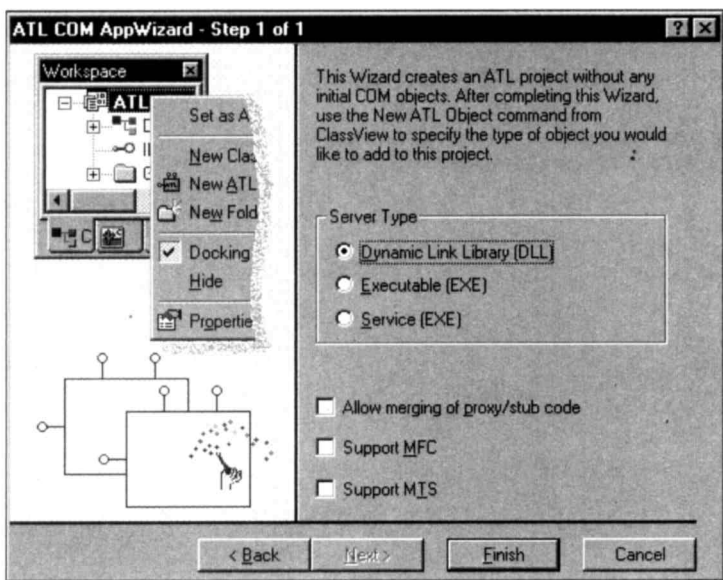


图17-1 ATL COM AppWizard的界面1

当完成后，这个向导会自动产生一个组件的外壳。这时没有组件存在，只有DLL根据COM规范所要求的函数存在，可以在ASPComponents.cpp文件中看到这些函数，但不用关心文件的细节，也不用改变它。当向DLL中添加组件时，向导会修改该文件。

现在将组件放入DLL。在工作区窗口选择ClassView选项卡，右击ASPComponents Classes，选择New ATL Object。然后再选Simple Object，并且在Short Name框中键入Tablestorage，如图17-2所示。

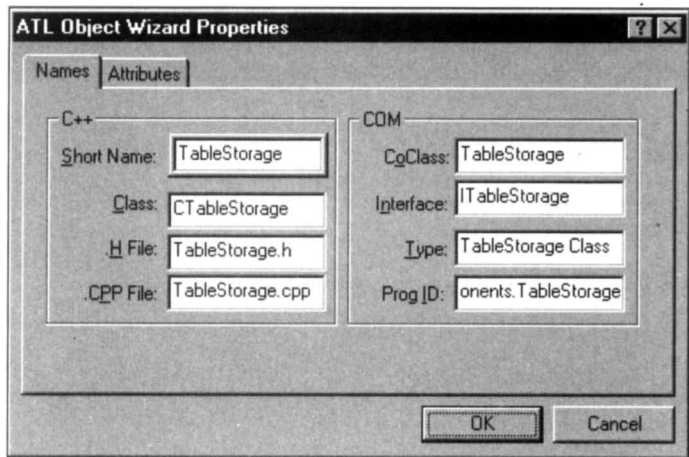


图17-2 ATL COM AppWizard的界面2

除非你打算更深入地研究ATL，否则在Attributes选项卡保持默认状态，不必改变任何选项。对所有这些选项的意义的详细描述已超出本书的范围。Attributes选项卡如图17-3所示。点击OK后，就有了一个用来工作的对象。

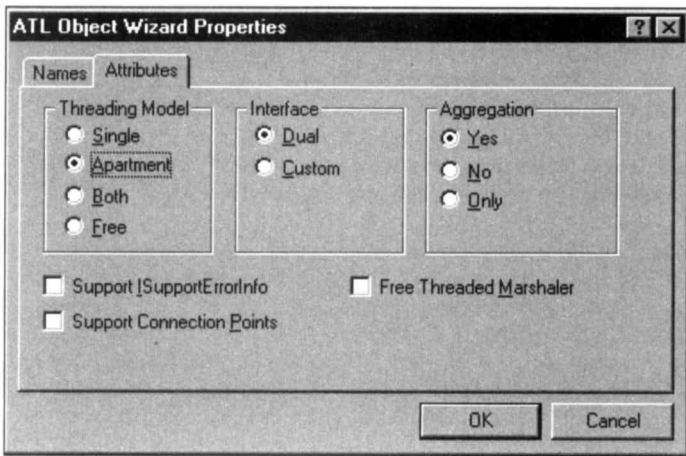


图17-3 设置Attributes选项卡

到目前为止，VC++向导已经完成了所有的工作，如果查看 TableStorage.h，你会看到向导产生的ATL代码。在大多数情况下，不必改变这些代码。实际上，可以只依靠这些代码，不用了解ATL，继续进行编程。然而，需要对默认的设置做一点改变，使得生成的代码能够编译。

为了减小组件的大小，当 AppWizard 创建一个组件时，自动关闭 C++ 的异常处理。因为如果启用这一功能的话，则需要 C 运行期库。STL 使用异常处理，所以需要启用它。在 Project Settings 对话框的 C/C++ 选项卡中，下拉 Category 框选择 C++ Language，确保 Settings for 框设置为 All Configurations，选择 Enable exception handling 复选框，如图 17-4 所示。

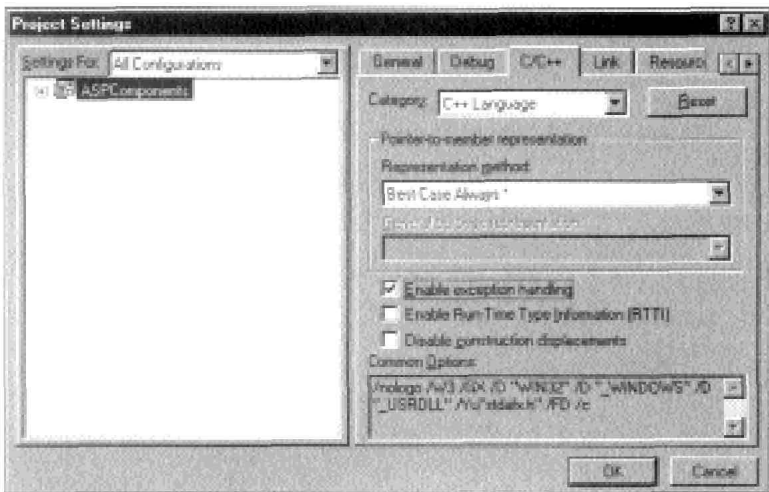


图17-4 启用异常处理

如果用 Release 模式编译可能会得到一个链接错误。当编译一个发行版本的 ATL 项目时，如果异常处理没有关闭，会出现这样的问题。使用异常处理时，需要 C 运行期启动代码；但是在默认方式下，Release 模式的 ATL 项目定义了 _ATL_MIN_CRT 符号，它拒绝启动代码。为了解决这个问题，在 C/C++ 预处理器定义中删除 _ATL_MIN_CRT。详细资料请看微软知识库中的文章 Q165259。

我们增加的第一段代码用于建立 STL数据结构，把下列代码加到 TableStorage.h的开头：

```
#pragma warning(disable:4786)

#include <map>
#include <vector>
#include <string>

using namespace std;

typedef map< wstring, unsigned short > COLUMN_INDEX_MAP;
typedef map< unsigned short, wstring > INDEX_FIELD_MAP;
typedef vector< INDEX_FIELD_MAP > ROW_VECTOR;
```

然后增加下列程序段作为 Ctablestorge的预定义成员：

```
protected:
    COLUMN_INDEX_MAP    m_columnIndexMap;
    ROW_VECTOR          m_rows;
```

这里，声明了两个映射和一个矢量。这个矢量实际是两个映射之一的矢量。使用 C++的 typedef命令定义映射和矢量主要是为了使得代码更易读，对数据类型描述得更好。

COLUMN_INDEX_MAP将一个字符串(列的名称)映射到一个索引中。INDEX_FIELD_MAP表示数据表中的每一行的数据。由于他可能是一个稀疏行，用一个映射实现是高效的，空字段不占用任何空间。INDEX_FIELD_MAP映射COLUMN_INDEX_MAP提供的索引到字段值。最后，ROW_VECTOR包含代表行的每一个映射。

现在已说明了内部数据结构，可将它们用于使用属性的外部世界。

2. 增加属性

我们将增加两个属性：行数属性和列数属性。

在项目工作区选择ClassView选项卡，右击ITableStorage接口。在菜单中选择Add Property，按图17-5填写对话框。选择Get Function复选框，而不选择Put Function复选框，使得它为只读属性。

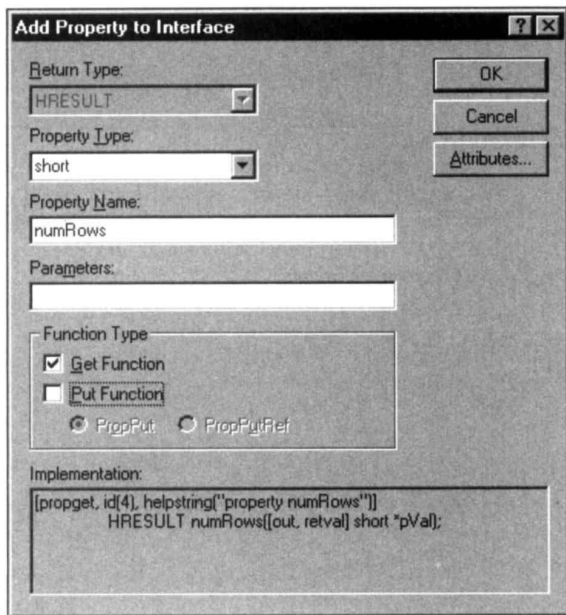


图17-5 增加行数属性

产生返回这一属性的 `get_numRows` 方法。这就是说可有产生属性值的逻辑。在这种情况下，可通过调用行矢量的 `size()` 方法设置属性：

```
STDMETHODIMP CTableStorage::get_numRows(short *pVal)
{
    *pVal = m_rows.size();

    return S_OK;
}
```

用同样的方式增加另一属性，它称为 `numColumns`。添加如下代码：

```
STDMETHODIMP CTableStorage::get_numColumns(short *pVal)
{
    *pVal = m_columnIndexMap.size();

    return S_OK;
}
```

现在有了获得所存储数据的行数和列数的方法，还需让一些数据输入到组件中，这是下一步要做的工作。

3. 增加方法

到目前为止，还无法把任何数据输入到内部数据结构中，也无法读取它们。下面增加四个方法完成下列任务：

- 在数据结构中插入数据。
- 从数据结构中获取一个字段。
- 获取列名称。
- 对列进行排序。

(1) 分析数据

第一个方法将获得一个以逗号隔离的字符串，进行分析，再将数据输入数据结构中。

在项目工作区中选择 `ClassView` 选项卡，右击 `ITableStorage` 接口。在菜单中选择 `Add Method`，按图 17-6 填写对话框。

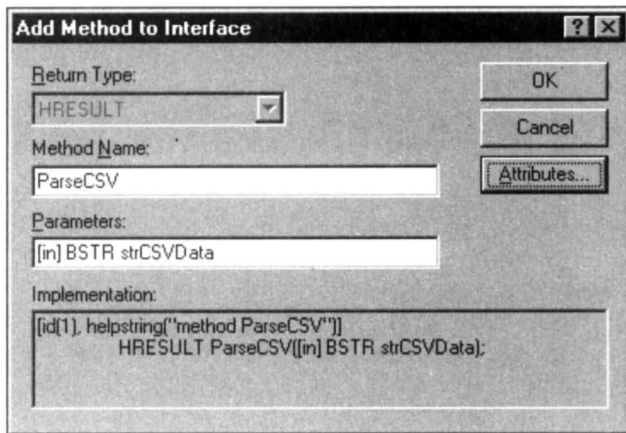


图 17-6 增加 ParseCSV 方法

然后用下列代码填写 ParseCSV 方法的主体部分：

```
STDMETHODIMP CTableStorage::ParseCSV(BSTR strCSVData)
{
    HRESULT                hResult = S_OK;
    wstring                data = strCSVData;
    wstring::size_type     startPos = 0;
    wstring::size_type     endPos = 0;
    wstring                field;
    bool                   headerRow = true;
    INDEX_FIELD_MAP        fieldMap;
    INDEX_FIELD_MAP::size_type fieldIndex = 0;

    // Make sure there is not any data around.
    m_rows.clear();
    m_columnIndexMap.clear();
}
```

CSV数据作为一个字符串参数传递，并清除两个 STL 成员变量，删除以前调用这个方法时输入的数据。

```
endPos = data.find_first_of( L",\n", startPos );

while ( endPos != wstring::npos )
{
    // Obtain a field from the data.
    field = data.substr( startPos, (endPos-startPos) );
}
```

要特别注意第一行，因为它包含列的名称，每一列的名称作为一个键存储在 m_columnIndexMap 中，用映射的当前大小作为索引。当一行处理完后，可以利用列名称映射的索引也就是列的数值索引这一事实。如果被分析的字段有数据，就将其存储在 INDEX_FIELD_MAP 中。

```
if ( headerRow )
{
    m_columnIndexMap[field] = m_columnIndexMap.size();
}
else
{
    if ( field.length() > 0 )
    {
        fieldMap[fieldIndex] = field;
    }
    fieldIndex++;
}
}
```

当达到行的末尾时，映射被存入 ROW_VECTOR 中。

```
// Check for end of the row.
if ( data[endPos] == L'\n' )
{
    if ( headerRow )
    {
        headerRow = false;
    }
    else
    {
        // Start a new row.
        m_rows.push_back( fieldMap );
        fieldIndex = 0;
        fieldMap.clear();
    }
}
```



```

    }
    startPos = (endPos+1);
    endPos = data.find_first_of( L",\\n", startPos );
}

return hResult;
}

```

一旦所有行都处理完后，COM组件中的CSV数据的存储空间已经最小化，并且由于使用映射可以加快访问速度。因为只存储了有实际值的字段，所以存储空间最小。由于映射在内部组织数据，可快速检索，访问速度很快。

(2) 数据访问

现在数据已能高效存储，下一步是能够访问它。

向ITableStorage接口添加一个称为GetField的新方法，按图17-7对对话框进行填写。

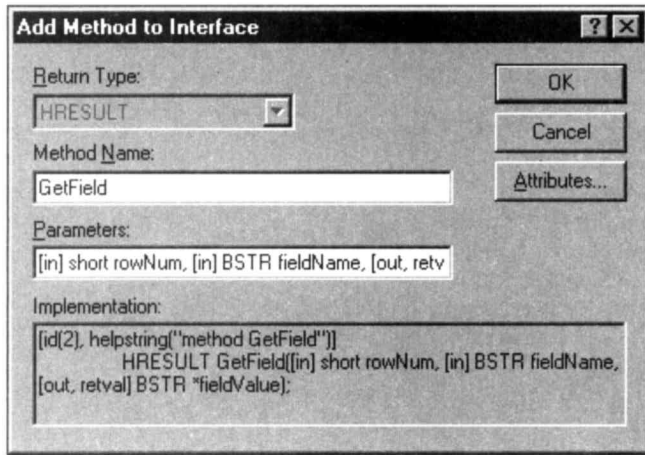


图17-7 增加GetField方法

给定行数和列名称，这个方法将返回字段值（如果它存在）。改变这个方法的主体，如下所示：

```

STDMETHODIMP CTableStorage::GetField(short rowNum, BSTR fieldName, BSTR
                                     *fieldValue)
{
    HRESULT hResult = E_FAIL;
    if ( rowNum < m_rows.size() )
    {
        COLUMN_INDEX_MAP::iterator mapIter =
            m_columnIndexMap.find( fieldName );

        if ( mapIter != m_columnIndexMap.end() )
        {
            int columnNum = (*mapIter).second;

            INDEX_FIELD_MAP::iterator fieldIter =
                m_rows[rowNum].find( columnNum );
            CComBSTR tempBStr( "" );

            if ( fieldIter != m_rows[rowNum].end() )
            {
                tempBStr = (*fieldIter).second.c_str();
                *fieldValue = tempBStr.Detach();
            }
        }
    }
}

```

```

        hResult = S_OK;
    }
}

return hResult;
}

```

用于获得与映射的键相对应的值的映射方法是 `find`。对于 `COLUMN_INDEX_MAP` 映射，从 `find` 中返回 `pair<wstring, unsigned short>` 类型的遍历器。如果没有找到键，遍历器具有 `m_columnIndexMap.end()` 的值。如果找到键，返回的遍历器的 `second` 成员包含对应值：在这种情况下，它是列名称的索引。这个索引作为 `INDEX_FIELD_MAP` 映射的键，给定行即可得到字段值。如果找到这个值，通过 `[out, retval]` 参数返回，也就是 `fieldValue`。

下面创建一个得到列名称的方法。在 `ITableStorage` 中添加一个称为 `GetColumnName` 的新方法，按图 17-8 所示填写对话框。

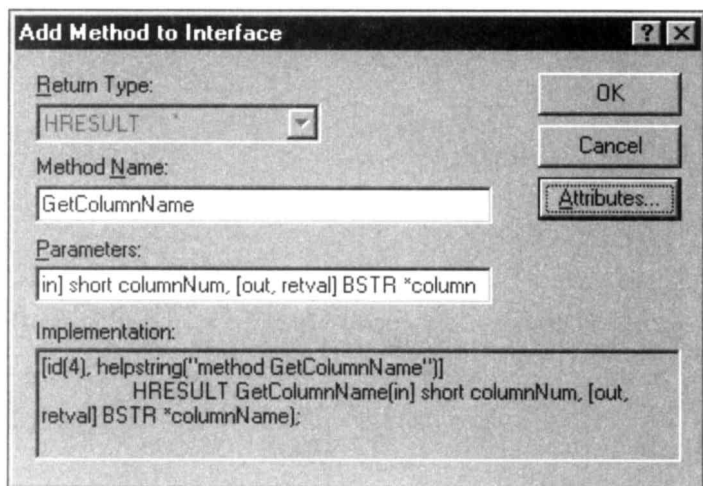


图 17-8 增加 `GetColumnName` 方法

给定一系列的索引，这个方法将返回列的名称，用下列代码改变这个方法的主体：

```

STDMETHODIMP CTableStorage::GetColumnName(short columnNum, BSTR *columnName)
{
    HRESULT hResult = S_OK;
    bool found = false;

    if ( columnNum < m_columnIndexMap.size() )
    {
        COLUMN_INDEX_MAP::iterator mapIter = m_columnIndexMap.begin();

        while ( !found && (mapIter != m_columnIndexMap.end()) )
        {
            if ( (*mapIter).second == columnNum )
            {
                CComBSTR tempBStr( (*mapIter).first.c_str() );
                *columnName = tempBStr.Detach();
                found = true;
            }
            mapIter++;
        }
    }
}

```

```

    if ( !found )
    {
        HRESULT = E_FAIL;
    }

    return HRESULT;
}

```

在这个方法中，必须通过映射进行线性查找，因为程序实际是使用键的值而不是键进行检索，遍历 `m_columnIndexMap` 映射直到找到索引或者搜索到映射的末端（即索引未找到）。注意遍历器的 `second` 成员用于比较，这是因为查找的是键的值而不是键。如果找到列索引，列名称（实际上是键）通过 `[out, retval]` 参数返回，也就是 `columnName`。

(3) 数据排序

下面添加一个对数据有实际影响的方法。使用 STL `sort` 算法对行进行排序。

在 `TableStorage.cpp` 文件的开头添加下列语句：

```
#include <algorithm>
```

然后给 `ITableStorage` 增加一个新的方法 `Sort`，按图 17-9 对对话框进行填写：

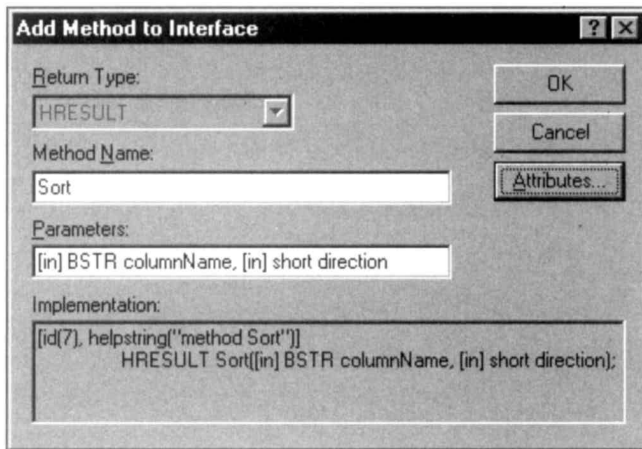


图17-9 增加Sort方法

如果 `direction` 参数为 0，将以降序来排序，否则就以升序来排序。将 `Sort` 方法的代码改为如下：

```

STDMETHODIMP CTableStorage::Sort(BSTR columnName, short direction)
{
    HRESULT    HRESULT = S_OK;

    COLUMN_INDEX_MAP::iterator    mapIter;

    mapIter = m_columnIndexMap.find( columnName );

    if ( mapIter != m_columnIndexMap.end() )
    {
        doCompare    compareFunc( (*mapIter).second, direction );

        sort( m_rows.begin(), m_rows.end(), compareFunc );
    }
    else
    {

```

```

        hResult = E_INVALIDARG;
    }

    return hResult;
}

```

因为CTableStorage数据结构是一种STL数据结构的组合(一种映射矢量),需要提供一种定制的比较函数进行排序。例如,CTableStorage::Sort允许行根据任何列排序,这种排序可以仅使用一个比较函数,或使用一个函数对象。用于排序的列被传递给函数对象doCompare的构造器。

在TableStorage.cpp文件某处增加如下函数对象,例如放在includes和CTableStorage类的实现部分之间。

```

class doCompare : public binary_function< const INDEX_FIELD_MAP &,
                                           const INDEX_FIELD_MAP &, bool >
{
public:
    doCompare( int column, short direction ) : m_column(column),
                                              m_direction(direction) {}
    bool operator()(const INDEX_FIELD_MAP &x, const INDEX_FIELD_MAP &y)
    {
        bool    answer = false;

        INDEX_FIELD_MAP::const_iterator    valueIterX = x.find( m_column );
        INDEX_FIELD_MAP::const_iterator    valueIterY = y.find( m_column );

        if ( ((m_direction > 0) && ( (*valueIterX).second <
                                     (*valueIterY).second )) || ((m_direction == 0) &&
                                                                    ( (*valueIterX).second >
                                                                      (*valueIterY).second )) )
        {
            answer = true;
        }

        return answer;
    }
private:
    int        m_column;
    short      m_direction;
};

```

这是doCompare函数对象的声明和实现部分,在排序中用于比较。确实,这个函数看起来可能复杂一点,详细的语法解释可以查C++手册,不过它的目的却非常简单。比较各行以便按m_direction参数指定的次序排序。以列号作为构造器中的第一个参数,并存储在一个成员变量(m_column)中。这样,当做比较时,函数对象就知道比较的是哪一列。

注意doCompare是由STL binary_function模板衍生的。STL提供binary_function使得创建比较函数非常方便。

17.3.4 测试

这个组件可用于许多地方:比如VB程序中、ASP文件中甚至于C++程序中。下面分析一下在ASP中的使用方式。

```
<H2>TableStorage C++ Component Test Driver</H2>
```

```

<%
    Dim    objTableStorage
    Dim    csvString

```

```

csvString = "Name, Group, Instrument" & vbCrLf
csvString = csvString & "Robert Plant,Led Zeppelin,Vocals" & vbCrLf
csvString = csvString & "Jim Morrison,The Doors,Vocals" & vbCrLf
csvString = csvString & "Lenny Kravitz,,Everything" & vbCrLf
csvString = csvString & "Keith Moon,The Who,Drums" & vbCrLf
csvString = csvString & "Jimi Hendrix,,Guitar & Vocals" & vbCrLf

Set objTableStorage = Server.CreateObject( "ASPComponents.TableStorage" )

objTableStorage.ParseCSV csvString
objTableStorage.Sort "Name", 1

%>

```

首先创建一个字符串 csvString，它代表数据。用 Server.CreateObject 创建一个 TableStorage 对象，用 ParseCSV 方法对字符串进行分析。这时数据在内存中。然后根据名字按升序排序。

获得属性是很容易的，可以这样引用它们：

```

<P>Number for Columns: <%=objTableStorage.numColumns %></P>
<P>Number for Rows: <%=objTableStorage.numRows%></P>

```

可以通过遍历整个表显示含有这些数据的 HTML 表。注意 On Error Resume Next。通常，如果程序试图得到一个字段的值，而该字段没有值，就会产生错误。下一章将用更好的方法来解决这个问题。

```

<TABLE border=1>
  <TR>
    <%
      On Error Resume Next
      For column = 0 to objTableStorage.numColumns - 1
    %>
      <TH><%= objTableStorage.GetColumnName( column ) %></TH>
    <%
      Next %>
  <TR>
    <%
      For row = 0 to objTableStorage.numRows - 1 %>
    %>
      <TD><%= objTableStorage.GetField( row,
        objTableStorage.GetColumnName( column ) ) %></TD>
    <%
      Next %>
    </TR>
  <%
    Next %>
</TABLE>

```

Number for Columns: 3		
Number for Rows: 5		
Name	Group	Instrument
Jim Morrison	The Doors	Vocals
Jimi Hendrix		Guitar & Vocals
Keith Moon	The Who	Drums
Lenny Kravitz		Everything
Robert Plant	Led Zeppelin	Vocals

图17-10 运行componentTest.asp时的界面

注意，componentTest.asp页面的全部代码可以从 Worx网站上下载，本章及下一章的Visual C++项目ASPComponents是本书源代码的一部分。

另外，为了得到一个字段的值，必须指定字段名。一个更灵活的界面应允许使用索引来得到字段的值。运行这个页面，浏览器中应该得到如图 17-10所示的显示。

17.3.5 错误处理

错误处理的两个方面是：

- 确保能捕捉所有的错误。
- 提供错误情况的精确描述。

在程序开发过程中尽量早地使用合适的错误处理工具，能明显减少开发和测试时间，这是因为能更快地发现和纠正错误。可以使用 C++的异常处理来捕捉错误，使用 Error对象向客户端反馈信息。

1. 异常处理

错误处理能力受程序语言限制。一个典型的错误处理过程包含如下几个方面：

- 调用一个函数。
- 检查返回值。
- 如果成功，程序采用一个代码路径，如果失败采用另一个。

这个过程的问题是导致代码嵌套，很难跟踪。另外，没有办法迫使程序员检测返回值。他们可能懒得检查返回值，或是忘了。

我们更希望编程环境能替我们捕捉错误并引导代码按预先确定的错误处理路线运行。在 Visual Basic中可以用On Error Goto <错误处理函数>做到这点。当错误产生时，程序将跳到指定的错误处理函数处。也可以在 Visual Basic中调用Err.Raise来标记一个错误。

C++和 Java拥有相同的错误处理概念，叫作异常处理。异常处理允许把代码的一部分放入try块来保护它。如果错误产生于try块，那么程序的执行跳到catch块，在那儿有错误处理代码。当错误产生于catch程序块时，代码将自动指向try程序块。使用throw关键字发出错误信息。下面是异常处理的简单例子：

```
try
{
    myFunction( char *ptr );
}
catch( string message )
{
    sprintf( stderr, "Error occurred: %s\n", message.c_str() );
}
```

函数与下面的代码类似：

```
Void myFunction( char *ptr )
{
    if ( ptr == NULL )
    {
        throw string( "Invalid pointer passed to myFunction" );
    }
    // continue processing as normal
}
```

在第18章中将使用这种异常处理风格。

一旦捕获了错误，下一步是将错误信息返回给用户。如果所使用的 COM对象的宿主环境是ASP，最好的方法是通过 Error对象报告错误。

2. Error对象

如果未用 On Error Resume Next，当对没有值的字段调用 GetField方法时，将遇到如下错误：

```
error '80004005'
Unspecified error
```

/datashaping/componenttest.asp, line 43

这个提示用处不大，在错误产生时需要组件能提供更多的信息，可以通过 Error对象来做到这一点。

如果创建一个新的 ATL对象，并使用Error对象，可以选择ATL Object Wizard Properties对话框中的Support ISupportErrorInfo复选框，指示向导产生支持Error对象的代码。

手动插入代码也是很容易的。仅需把包含文件 TableStorage.h做如下修改：

```
////////////////////////////////////
// CTableStorage
class ATL_NO_VTABLE CTableStorage :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CTableStorage, &CLSID_TableStorage>,
public ISupportErrorInfo,
public IDispatchImpl<ITableStorage, &IID_ITableStorage,
&LIBID_ASPCOMPONENTSLib>
{
...

BEGIN_COM_MAP(CTableStorage)
    COM_INTERFACE_ENTRY(ITableStorage)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
END_COM_MAP()

// ITableStorage
public:
    STDMETHOD(Sort)(/*[in]*/ BSTR columnName, /*[in]*/ short direction);
    STDMETHOD(get_numColumns)(/*[out, retval]*/ short *pVal);
    STDMETHOD(get_numRows)(/*[out, retval]*/ short *pVal);
    STDMETHOD(GetColumnName)(/*[in]*/ short columnNum, /*[out, retval]*/ BSTR
*columnName);
    STDMETHOD(GetField)(/*[in]*/ short rowNum, /*[in]*/ BSTR fieldName, /*[out,
retval]*/ BSTR *fieldValue);
    STDMETHOD(ParseCSV)(/*[in]*/ BSTR strCSVData);

    // ISupportsErrorInfo
    STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

protected:
    COLUMN_INDEX_MAP    m_columnIndexMap;
    ROW_VECTOR           m_rows;
};

#endif //__TABLESTORAGE_H
```

通过这种修改，我们的类将支持 ISupportErrorInfo COM接口。然而，需要在源文件 TableStorage.cpp插入下面的代码，才可以实现增加的方法。

```
STDMETHODIMP CTableStorage::InterfaceSupportsErrorInfo(REFIID riid)
{
    static const IID* arr[] =
    {
        &IID_ITableStorage
    };

    for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        if (InlineIsEqualGUID(*arr[i],riid))
            return S_OK;
    }
    return S_FALSE;
}
```

我们的类从 CComCoClass派生，有一个 Error方法。也就是说 CTableStorage类能够使用 CComCoClass定义的所有公共方法和属性。现在 CTableStorage类支持错误接口，因而可以使用这个方法。CTableStorage类也提供Error对象的所有参数，包括错误代码、描述和帮助信息。

下面修改一下 GetField方法以便提供更多有用的信息。用下面代码修改方法的结尾。

```
if ( FAILED( hResult ) )
{
    wstring      strMessage;
    wchar_t      strRow[4];

    strMessage = L"The field '";
    strMessage += fieldName;
    strMessage += L"' at row '";
    _itow( rowNum, strRow, 10 );
    strMessage += strRow;
    strMessage += L"' does not have a value.";

    Error( strMessage.c_str() );
}

return hResult;
```

现在将得到如下的错误信息：

ASPComponents.TableStorage.1 error '80004005'

The field ' Group' at row '1' does not have a value.

/datashaping/componenttest.asp, line 43

现在不仅得到了增加的错误信息，而且得到了组件的 ProgId。当组件支持ISupportErrorInfo时，ProgId信息将自动地插入。

另外，我极力推荐程序员在组件开发初期使用错误支持。这将有助于程序员调试组件，并能够帮助组件用户的开发工作。当然，如果不能从错误信息中判断出是什么出了错，下一步就是调试程序了。

17.3.6 调试

在Visual C++中可以直接调试一个正常的可执行程序。可以设置断点，然后在调试模式下

运行程序。如果有DLL，则需要做更多的工作：C++调试器必须附加上支持调试的进程。

一个DLL不在自己的进程运行，它运行在其他进程中。因此，必须给调试器捆绑上能容纳DLL的应用程序进程空间。如果在 Visual Basic中测试，就需要捆绑上 Visual Basic；如果是在ASP上测试，则需要捆绑上 Web服务器进程。另外，在 Visual Basic中，可以运行VB6.EXE，打开一个使用DLL的项目，像正常情况一样设置断点。

还有一个直接设置组件进行调试的方法，只要在组件中需要调试的地方加上下面一行即可：

```
DebugBreak();
```

当容纳DLL程序的进程运行到这一行时，将停下来弹出一个对话框并告诉你已经到了断点，并询问是否调试应用程序。按 Cancel调用调试器(注意，按OK不进入调试器)。如果组件是调试版本而不是发行版本，当按 Cancel时，Visual C++将开始运行并停在插入 DebugBreak()的地方。在这可以设置另外的断点、观察变量和做其他的调试工作。

这种方式的主要问题是其侵入性。为了调试代码必须修改代码，还得十分小心避免遗留 DebugBreak()。如果组件产品中有 DebugBreak()，运行时将弹出对话框并锁住每一个用户使其脱离Web服务器，等待用户按下 OK或Cancel。这是很糟的。

如果组件在ASP中使用，修改然后重新编译就会出现如下错误：

**LINK: fatal error LNK1168: cannot Debug/ASPComponent.dll for writing
Error executing link.exe**

服务器组件在 Web服务器的进程空间中运行，如果得到上述错误，意味着服务器仍有此组件的引用。更确切地说，服务器正引用组件中的 DLL，所以需要解除对DLL的引用。但是，做到这些需要关闭并重新启动 WWW服务。然而，如果对象是在 MTS中运行，这个过程就简单多了。

COM+调试

首先，确定设定的激活属性(activation property)是专用的服务器进程而不是库进程。这样才能确保调试器绑定到所要调试的组件的进程。在 Project Settings选项卡上，将Executable for debug session项设置为dllhost.exe，将Program arguments项设置为ProcessID:<进程的ID>，如图17-11所示。

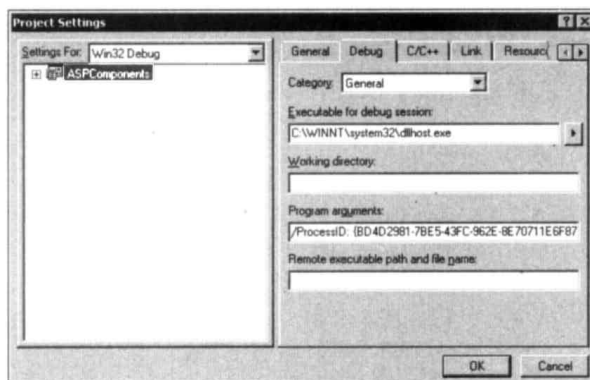


图17-11 设置激活属性

进程的ID可以通过在 Component Services Explorer中右键单击应用程序图标，并选择 Properties来获得，如图 17-12所示。

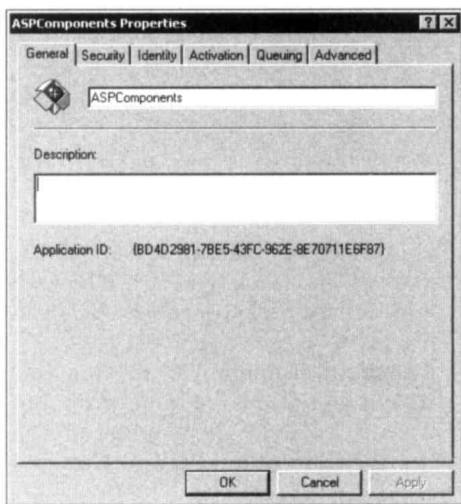


图17-12 获得进程的ID的界面

关闭组件所在的应用程序的服务器进程，确保组件当前不会驻留在内存中。否则，可能会使用没有绑定调试器的组件实例。确被建立组件的调试版本，在调试版本中设置所需断点并从Build菜单中执行程序。

运行带有程序参数的 `dllhost.exe`，以指示它载入包含被调试组件的应用程序。因为 COM+ 进程绑定了调试器，无论什么时候访问组件，调试器都能在断点处中断调用。

如果对象失败，有关 C++ 中的组件的其他情况都自动地放在事件日志中。

17.4 小结

本章介绍了 C++ 的起源以及设置与开发 ASP 组件相关的环境。读者已经学习了一部分 C++，如 STL 和 ATL，这些都是创建 ASP 组件的有用工具。这些工具都能满足服务器组件的设计需要，其目的就是减少存储空间和访问时间。为了介绍它们的使用方法，本章创建了一个能用于任何应用程序的通用组件。

组件的可利用能力的一个重要方面是其错误处理。C++ 异常处理提供了在组件中捕获错误的有效方法。要想在应用程序调用之外报告错误，组件需要支持 `ISupportErrorInfo`。

下一章将分析在 Visual C++ 中支持 ASP 和 COM+ 的集成，也将讨论在 C++ 中访问数据的两个选项。