Visual C++ 6.0 类参考详解

刘金龙 高兆法 巩玉国 编著

清华大学出版社
http://www.tsinghua.edu.cn

Visual C++ 6.0 类参考详解

刘金龙　高兆法　巩玉国　　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

Visual C++ 6.0 是微软公司 C++ 语言开发环境的最新版本。本书分为两部分:常用部分和高级部分,两部分互相配合使用。Visual C++ 6.0 的类库被称为 Microsoft Foundation Class Library 6.0,基本部分介绍类库中的常用类,使用这些常用类,可以完成一般应用程序的开发。本书在介绍这些常用类时,详细说明了类库中每个类的功能、用法、原型所在的头文件,同时详细描述了这些类中所有成员函数和数据成员。高级部分介绍类库中的高级类,使用这些类,可以完成一般高级应用程序的开发,而一般的程序设计,是很少能使用它们的。本书在介绍这些高级类的时候,详细说明了类库中每个类的功能、用法、原型所在的头文件,同时详细描述了这些类中所有成员函数和数据成员。

本书适合所有使用 Visual C++ 6.0 开发软件的用户阅读,对其程序设计将起到很好的帮助作用。

图书在版编目(CIP)数据

Visual C++ 6.0 类参考详解/刘金龙等编著. - 北京:清华大学出版社,1999
ISBN 7-302-03772-8

Ⅰ.V… Ⅱ.刘… Ⅲ.C 语言,C++6.0-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(1999)第 62015 号

出 版 者: 清华大学出版社(北京清华大学学研楼,邮编 100084)
http://www.tup.tsinghua.edu.cn
印 刷 者: 昌平环球印刷厂
责任编辑: 赵从棉 景丽娟
发 行 者: 新华书店总店北京发行所
开 本: 787×1092 1/16 **印 张**: 35.5 **字 数**: 1391 千字
版 次: 1999 年 12 月第 1 版 1999 年 12 月第 1 次印刷
书 号: ISBN 7-302-03772-8/TP·2117
印 数: 0001 ~ 5000
定 价: 62.00 元

前　言

C++是在C语言的基础上，吸收了BCPL、Simula 67和Alogl68等语言中的精华而逐渐发展起来的通用程序设计语言，是为适应90年代开发和维护复杂应用软件的需要而开发的。其目标是为程序员提供一个良好的程序开发环境和设计理论，以产生重用性和维护性良好的程序。同时，C++又非常强调代码的有效性和紧凑性。在程序设计方面，C++既支持传统的面向过程的程序设计方法，又支持面向对象的程序设计方法。自1983年贝尔实验室公布C++以来，它在计算机界引起了广泛的重视，在系统程序设计、计算机辅助设计、仿真、数据库和人工智能等领域得到了广泛应用。

微软公司成功地将C++"可视化"为Visual C++，现在的最新版本是6.0。自推出以来，Visual C++就以其完善的类库、友好的面向对象的程序界面和对标准C/C++的完整支持深得广大编程工作者的喜爱，在Windows 95/NT环境下，Visual C++几乎已经成了C++的代名词。作为Visual C++的核心，Microsoft基本类库(Microsoft Foundation Class Library)是学好、用好Visual C++的关键，在Visual C++ 6.0中，Microsoft基本类库的版本是6.0版。

Microsoft基本类库封装了Windows 95/NT环境下面向对象的程序开发界面(Microsoft Windows Application Programming Interface)，它提供了大量预先编写好的类及支持代码，可用于处理大多数标准的Windows编程任务，利用它可以大大降低Windows环境下编程的工作量。

Microsoft基本类库包含200多个类，可以分为若干组类，分别应用于一些特定操作，这些操作包括应用管理、可视控制、图形和打印、数据结构、文件和数据库管理、因特网和网络管理、调试和异常处理等方面。Microsoft基本类库的分类如下：

- 应用框架结构类
- 窗口类
- 图形和打印类
- 集合类
- 文件和数据库类
- OLE支持类
- Internet和网络类
- 调试和异常类
- 各种辅助类

与Microsoft基本类库的上一个版本4.22版(对应于Visual C++ 5.0)相比，Microsoft基本类库6.0版增加了许多功能，主要扩充了对日益流行的Internet网络编程的支持，包括动

态 HTML 控制的 CHtmlView、Internet Explorer4.0 通用控件类等，使利用 Visual C++编写网络应用程序更加方便。另外，Microsoft 基本类库 6.0 版还增加了 OLE 数据库类、活动文档容器等功能。

编写本手册的目的主要是想从实用的角度出发，为 Visual C++程序员和爱好者提供 Microsoft 基本类库完整的参考手册。为了方便读者使用，在手册的第一部分对 MFC 类及其成员函数分别做了简单介绍，并指明了每个成员函数具体说明所在的页码，起到索引的作用，类是按照功能划分排列的；第二部分则将所有类按字母索引顺序排列，并详细说明了每个类的功能、用法以及使用时需要包含的头文件，并具体说明了其所有成员函数(包括函数、参数和返回值等)和数据成员。

Microsoft 基本类库包含的类数量庞大，类之间的关系非常复杂，往往使初学者难以理出头绪。实际上，在编程中，经常使用的类只是一部分，只要熟悉常用的类即可完成 Visual C++的大部分编程工作。该书针对初学者，主要介绍 Microsoft 基本类库中常用类，如应用框架类、Windows 95/NT 的标准控件类等，对于一般的程序设计这是完全够用的；而针对高级程序设计人员，主要介绍 Microsoft 基本类库中的高级类，它们是进行高级程序设计(如，对系统资源操作的程序)所必需的。这些类的划分方法是作者根据多年的经验而定的，如有不当之处，还望读者提出建议。

本手册由长期使用 Visual C++的开发人员编写而成。由于水平有限，时间仓促，书中难免有错误之处，恳请读者予以指正。

作　者

1999 年 8 月

目　录

Microsoft 基本类库

CArchive　　# include < afx.h >

CArchive 没有基类。

CArchive 允许以永久的二进制形式(通常使用磁盘存储)保存复杂的对象网络。这些对象被删除后存储器仍保存它们。以后程序员可以从保留存储器中装入这些数据,在内存中重组这些对象。这种保持数据的过程被称作"串行化"。

可以将档案对象看作一种二进制流。同输入/输出流一样,档案与文件相关联,并允许对存储器进行缓冲式读写数据。输入/输出流用于处理 ASCII 字符序列,而档案则以高效、无冗余的格式来处理二进制的对象数据。

程序员必须在创建 CArchive 对象之前构造一个 CFile 对象,另外,还必须保证装入/保存档案的状态与文件打开的方式兼容。一个文件只能使用一个活动档案。

当构造一个 CArcive 对象时,应将其关联上一个代表打开文件的 CFile 或其派生类的对象 CFile。程序员还必须指定档案是否将用于装入或存储。CArchive 对象不仅可以处理基本的数据类型,而且还可以处理为串行伦设计的 CObject 派生类的对象。一个可串行化的类通常拥有 Serialize 成员函数,而且如同 CObject 类的描述,往往使用了宏 DECLARE _ SERIAL 和 IMPLEMENT _ SERIAL。

重载的抽取(> >)和插入(< <)操作符是简便的档案编程接口,它支持基本的数据类型和 CObject 的派生类。

CArchive 还支持 MFC Windows Sockets 类中的 CSocket 和 CSocketFile 的编程。数据成员 IsBufferEmpty 支持了这一用法。

参　见　CFile, CObject, CSocket, CSocketFile。

成员函数

CArchive::Abort

```
void Abort ( );
```

调用此函数可以不发出异常而关闭档案。CArchive 的析构函数通常将调用 Close 成员函数,会刷新没有存储到关联的 CFile 对象中的数据。该操作会导致异常。

捕获这些异常时使用 Abort,可以将 CArchive 对象析构而且防止进一步发出异常。与 CArchive::Close 不同, Abort 忽略了失败,所以发生失败时不会发出异常。

如果使用了 new 来在堆中分配 CArchive 对象,则程序员在关闭文件之后必须删除它。

参　见　CArchive::Close, CFile::Close。

CArchive::CArchive

```
CArchive( CFile * pFile, UINT nMode, int nBufSize = 4096,
void * lpBuf = NULL );
throw( CMemoryException, CArchiveException, CFileException );
```

参　数　**pFile**　CFile 对象的指针,该对象是永久数据的最终源或目的。

nMode　标志位,用于指定对象是从档案中读出还是写入。参数 nMode 必须是下列值中的一个:

- CArchive::load　从档案中装入数据。只要求 CFile 读允许。
- CArchive::store　将数据保存到档案中,要求 CFile 写允许。
- CArchive::bNoFlushOnDelete　当调用析构函数时防止档案自动地调用 Flush。如果设置了该标志位,则程序员负责在析构函数调用之前显式地调用 Close。如果不这样做,数据将会破坏。

nBufSize　用于指定内部文件缓冲区的字节数大小。注意:默认的缓冲区大小是 4096 字节。当按常规归档大对象时,如果使用数倍于文件缓冲区的更大缓冲区,则会提高性能。

lpBuf　可选指针,指向用户提供的大小为 nBufSize 的缓冲区。如果没有指定该参数,则档案从局部的堆中分配一个缓冲区,并且当删除对象时释放该缓冲区。档案不能释放用户提供的缓冲区。

构造 CArchive 对象并指定它是否用于装入或保存对象。程序员在创建该对象后不能改变这一属性。

程序员在关闭文件之前,不能使用 CFile 的操作函数来改变文件的状态。任何类似的操作都将毁坏档案的完整性。可以在串行化的任何时候通过 GetFile 成员函数获得档案文件对象,然后通过调用 CFile::GetPosition 函数实现访问文件的指定位置。程序员应在获得文件指针的位置之前调用 CArchive::Flush。

参　见　CArchive::Close, CArchive::Flush, CFile::Close。

CArchive::Close

```
void Close( );
throw( CArchiveException, CFileException );
```

刷新所有保留在缓冲区中的数据,关闭档案,并解除档案和文件的连接。禁止对档案进行进一步操作。在关闭档案之后,程序员可以创建同一文件的另一个档案或者关闭文件。

该成员函数保证将所有的数据从档案传送到文件中,并使档案不可用。为完成从文件到存储介质的传送,程序员必须首先使用 CFile::Close,然后删除 CFile 对象。

参　见　CArchive::Flush, CArchive::Abort。

CArchive::Flush

void Flush( );
throw(CFileException);

强制将所有留在档案缓冲区中的数据写到文件中。程序员必须调用 CFile::Close,完成从文件到存储媒体的传输。

参 见 CArchive::Close, CFile::Flush, CFile::Close。

CArchive::GetFile

CFile * GetFile( ) const;

为该档案获得 CFile 对象的指针。程序员必须在使用 GetFile 之前刷新档案。

返回值 一个指向使用中的 CFile 对象的常量指针。

参 见 CArchive::Flush。

CArchive::GetObjectSchema

UINT GetObjectSchema( );

从 Serialize 函数中调用此函数,来确定当前正在进行反串行化对象的版本。只有当 CArchive 对象正在被装入(CArchive::IsLoading 返回非 0)时,调用此函数才有效。应在 Serialize 函数中首先调用该函数;此函数只能调用一次。UINT 型的返回值为 -1 时表明版本号未知。

CObject 派生类可以通过使用位操作符将 VERSIONABLE_SCHEMA 和自身的模式版本号相结合,创建"可版本化的对象"。也就是说,该对象的 Serialize 成员函数可以读出多个版本。当版本不匹配时,默认的框架功能(没有 VERSIONABLE_SCHEMA)将发出一个异常。

返回值 在串行化期间,返回正在进行读操作的对象的版本。

参 见 CObject::Serialize, CObject::IsSerializable, IMPLEMENT_SERIAL, DECLARE_SERIAL, CArchive::IsLoading。

CArchive::IsBufferEmpty

BOOL IsBufferEmpty( ) const;

调用此成员函数可以确定档案对象的内部缓冲区是否为空。提供此函数是为了支持同 MFC Windows Sockets 类 CSocketFile 一起编程。程序员不需要为与 CFile 对象关联的档案而使用它。

与 CSocketFile 对象关联的档案才能使用 IsBufferEmpty,其使用原因是档案缓冲区可能保存不止一条的信息或记录。在接收到一条信息后,程序员应使用 IsBufferEmpty 来控制循环,该循环能继续接收数据直到缓冲区为空时。更多的信息参见 CAsyncSocket 类的成员函数 Receive。

返回值 如果档案的缓冲区不为空,则返回非 0 值;否则,返回 0。

参 见 CSocketFile, CAsyncSocket::Receive。

CArchive::IsLoading

BOOL IsLoading( ) const;

确定档案是否正在用于装入数据。此成员函数由档案类的成员函数 Serialize 来调用。

返回值 如果档案当前正在用于装入数据,返回非 0 值;否则,返回 0。

参 见 CArchive::IsStoring。

CArchive::IsStoring

BOOL IsStoring( ) const;

确定档案是否正在用于存储数据。此成员函数由档案类的成员函数 Serialize 来调用。

如果档案的 IsStoring 状态为非 0,则 IsLoading 的状态为 0;反之依然。

返回值 如果档案当前正在用于存储,返回非 0 值;否则,返回 0。

参 见 CArchive::IsLoading。

CArchive::MapObject

void MapObject(const CObject * pOb);

参 数 pOb 要存储对象的常量指针。

调用此成员函数可以将对象放到映射表中。虽然该对象实际上并没有串行化到文件,但是可以被子对象引用。例如:程序员虽然不需要串行化一个文档,但可能需要对文档中的一部分项目进行串行化。通过调用 MapObject,可允许这些项目或子对象引用该文档。

当从 CArchive 对象装入或存储数据时,可以调用 MapObject。在串行化和反串行化期间,MapObject 增加指定的对象到 CArchive 对象维护的内部数据结构中。但与 ReadObject 和 WriteObject 不同,MapObject 对象不调用串行化操作。

参 见 CArchive::ReadObject, CArchive::WriteObject。

CArchive::Read

UINT Read(void * lpBuf, UINT nMax);
throw(CFileException);

参 数 lpBuf 指向用户提供的缓冲区的指针,该缓冲区用于从档案中接收数据。

nMax 无符号整数,用于指定从档案中读取的字节数。

从档案中读出指定字节数的字节,档案并不解释字节。程序员可以在 Serialize 函数中使用 Read 成员函数,以读入保存在此对象中的一般结构。

返回值 无符号整数,用于保存实际读入的字节数。如果返回值小于申请的数值,则说明已经到达文件的尾部。在到达文件尾部的情况下不会发出异常。

CArchive::ReadClass

CRuntimeClass * ReadClass(const CRuntimeClass * pClassRefRequested = NULL, UINT * pSchema = NULL, DWORD * obTag = NULL);
Throw CArchiveException;
Throw CNotSupportedException;

参 数 pClassRefRequested CRuntimeClass 的结构指针,该结构与申请的对象项对应,可以为空。

pSchema 指向先前存储的运行时对象模式的指针。

obTag 作为对象唯一的标签的数字,通过执行 ReadObject 而在内部使用此参数,仅在高级编程使用。ObTag 通常应为

空。

调用此成员函数,可以读入先前通过 WriteClass 而存储的类的引用。

如果 pClassRefRequested 不是空,则 ReadClass 校验档案类的信息是否与运行时类相兼容。如果不兼容,则 ReadClass 将发出一个异常。

运行时类必须使用 DECLARE _ SERIAL 和 IMPLEMENT _ SERIAL;否则,ReadClass 将发出 CNotSupp ortedException 异常。

如果 pSchema 为空,则通过调用 CArchive: : GetObjectSchema 可以检取存储类的模式;否则 pSchema 将保存先前存储的运行时类的模式。

程序员可以使用 SerializeClass 来取代 ReadClass。类引用的读和写 ReadClass 都可处理。

返回值 CRuntimeClass 的结构指针。

参 见 CArchive: : WriteClass, CArchive: : GetObjectSchema, CArchive: : SetObjectSchema, CArchiveException, CNotSupportedException, CArchive: : SerializeClass。

CArchive: : ReadObject

CObject * ReadObject(const CRuntimeClass * pClass);

throw(CFileException, CArchiveException, CMemoryException);

参 数 pClass 指向 CRuntimeClass 结构的常量指针,该对象与将要读的对象相对应。

从档案中读入对象数据,并构造一个合适类型的对象。通常此函数由 CObject 指针 CArchive 抽取操作符(> >)调用。随后,ReadObject 将调用档案类的成员函数 Serialize。pClass 参数从宏 RUNTIME _ CLASS 中获得;如果程序员提供了一个非 0 的 pClass 参数,则该函数将校验档案对象的运行时类。前提是程序员在类的实现中已经使用宏 IMPLEMENT _ SERIAL。

返回值 CObject 指针。它必须通过调用 CObject: : IsKindOf 强制地转换为正确的派生类。

参 见 CArchive: : WriteObject, CObject: : IsKindOf。

CArchive: : ReadString

Bool ReadString(CString& rString);

LPTSTR ReadString(LPTSTR lpsz, UINT nMax);

throw(CArchiveException);

参 数 rString CString 的引用,用于保存从与 CArchive 对象关联的文件中读入的字符串。

lpsz 指向缓冲区的指针,该缓冲区是由用户提供的,将接收一个以 null 结尾的文本字符串。

nMax 指定读入的最大字符数。此参数应比 lpsz 指定的缓冲区小。

调用此成员函数可以将文本数据从与 CArchive 对象关联的文件读入缓冲区中。在此成员函数带 nMax 参数的版本中,缓冲区最多能保存 nMax - 1 个字符。当读到一个回车换行符时就停止读入,并在读入的字符串末尾追加一个 null 字符('\0')。

CArchive: : ReadString 还可以用于文本方式的输入,但是遇到回车换行符时并不终止读入。

返回值 在返回 Bool 的版本中,如果成功执行,返回 True;否则,返回 False。

在返回 LPTSTR 的版本中,返回保存文本数据的缓冲区指针;如果到达文件的尾部,则返回空。

参 见 CArchive: : Read, CArchive: : Write, CArchive: : WriteString, CArchiveException。

CArchive: : SerializeClass

void SerializeClass(const CRuntimeClass * pRuntimeClass);

参 数 pRuntimeClass 指向基类的运行时类对象的指针。

当希望存储或装入基类的版本信息时,调用此成员函数。根据 CArchive 的方向, SerializeClass 从 CArchive 对象读出或写入类的引用。使用 SerializeClass 来替代 ReadClass 和 WriteClass,可以简便地实现基类对象的串行化。SerializeClass 需要更少的代码和参数。

同 ReadClass 一样,SerializeClass 将验证档案类信息是否与运行时类相兼容。如果不兼容,则 SerializeClass 将发出 CArchiveException 异常。

运行时类必须使用 DECLARE _ SERIAL 和 IMPLEMENT _ SERIAL;否则,SerializeClass 将发出 CNotSupportedException 异常。

使用宏 RUNTIME _ CLASS 可以检取 pRuntimeClass 参数的值。基类必须已使用宏 IMPLEMENT _ SERIAL。

参 见 CArchive: : ReadClass, CArchive: : WriteClass, CArchive: : GetObjectSchema, CArchive: : SetObjectSchema, CArchiveException, CNotSupportedException。

CArchive: : SetLoadParams

void SetLoadParams(UINT nGrowBy = 1024);

参 数 nGrowBy 如果需要增大装载数组的尺寸时,指定要分配的元素空位的最小值。

当程序员要从档案中读入大量的 CObject 派生类对象时调用 SetLoadParams。CArchive 使用装载数组来分解储存在档案中的对象的引用。SetLoadParams 允许程序员设置装载数组的增长尺寸。

在装入了对象或者调用了 MapObject 或 ReadObject 后,不能调用 SetLoadParams。

参 见 CArchive: : SetStoreParams。

CArchive: : SetObjectSchema

void SetObjectSchema(UINT nSchema);

参 数 nSchema 指定对象的模式。

调用此成员函数可以设置对象的模式。该模式存储在档案对象的 nSchema 中。设置后可通过调用 GetObjectSchema,返回在 nSchema 存储的值。

一般在高级版本中使用 SetObjectSchema 设置。例如,当希望在派生类的 Serialize 函数中,强制读入某种特殊的版本时。

参 见 CArchive::GetObjectSchema。

CArchive::SetStoreParams

void SetStoreParams(UINT nHashSize = 2053, UINT nBlockSize = 128);

参 数 nHashSize 接口指针映射表的哈希表大小。

nBlockSize 为扩展参数指定内存分配的间隔尺寸。为了获得最佳特性,此参数应为 2 的幂次方。

当向档案存储大量的 CObject 派生类对象时,使用 SetStoreParams。

SetStoreParams 允许设置哈希表的大小和映射表的块尺寸;该映射表用于在串行化期间标识唯一的对象。

在任何对象存储之后,或者在 MapObject 或 WriteObject 之后程序员不能调用 SetStoreParams。

参 见 CArchive::SetLoadParams。

CArchive::Write

void Write(const void * lpBuf, UINT nMax);
throw(CFileException);

参 数 lpBuf 指定指向缓冲区的指针,该缓冲区由用户提供,保存要写入到档案中的数据。

nMax 整数,用于指定写入到档案中的字节数。

写入指定字节数的内容到档案中。该档案并不格式化字节。

可以在自己的 Serialize 函数中使用 Write 成员函数以写入保存在对象中的普通结构。

参 见 CArchive::Read。

CArchive::WriteClass

void WriteClass(const CRuntimeClass * pClassRef);

参 数 pClassRef CRuntimeClass 的结构指针,该结构与申请的类引用相对应。

使用 WriteClass 可以在派生类串行化过程中存储基类的版本和类信息。WriteClass 为基类将 CRuntimeClass 的引用写入到 CArchive 中。使用 ReadClass 可检取该引用。

WriteClass 验证档案类信息是否与运行时类相兼容。如果不兼容,WriteClass 将发出 CArchiveException 异常。

运行时类必须使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL;否则,WriteClass 将发出一个 CNotSupportedException 异常。

可以使用 SerializeClass 来替代 WriteClass,SerializeClass 可以用来处理类引用的读和写。

参 见 CArchive::ReadClass, CArchive::GetObjectSchema, CArchive::SetObjectSchema, CArchive::SerializeClass, CArchiveException, CNotSupportedException。

CArchive::WriteObject

void WriteObject(const CObject * pOb);
throw(CFileException, CArchiveException);

参 数 pOb 正在存储的对象的常量指针。

将指定的 CObject 存储到档案中。

通常,此函数由为 CObject 重载的 CArchive 插入操作符(<<)调用。随后,将调用档案类的函数 Serialize。

程序员必须使用宏 IMPLEMENT_SERIAL 来允许归档。WriteObject 将 ASCII 类名写入到档案中,该类名在以后装入时用于验证。为防止不必要的将一个类的多个对象的类名重复写入,会采用一种特殊的编码方案。该方案将防止对象的冗余存储。

注意:在开始归档对象之前应完成所有对对象的创建、删除和更新操作。如果归档时混入对对象的修改,会破坏档案。

参 见 CArchive::ReadObject。

CArchive::WriteString

void WriteString(LPCTSTR lpsz);
throw(CFileException);

参 数 lpsz 指定缓冲区的指针,该缓冲区保存一个以 null 结尾的文本字符串。

使用此成员函数可以将数据从缓冲区写入到与 CArchive 对象关联的文件中。结尾的空字符('\0')将不会写到文件中。

在磁盘已满的情况下,该函数会发出异常。

Write 也允许写入一定数目的字节到文件中,但最好以 null 字符结束。

参 见 CArchive::Write, CArchive::Read, CArchive::ReadString, CFileException。

操作符

CArchive::operator <<

friend CArchive& operator << (CArchive& ar, const CObject * pOb);
throw(CArchiveException, CFileException);
CArchive& operator << (BYTE by);
throw(CArchiveException, CFileException);
CArchive& operator << (WORD w);
throw(CArchiveException, CFileException);
CArchive& operator << (int i);
throw(CArchiveException, CFileException);
CArchive& operator << (LONG l);
throw(CArchiveException, CFileException);
CArchive& operator << (DWORD dw);
throw(CArchiveException, CFileException);
CArchive& operator << (float f);
throw(CArchiveException, CFileException);
CArchive& operator << (double d);
throw(CArchiveException, CFileException);

将显式对象或基本数据类型存储到档案中。

如果在类实现中使用了宏 IMPLEMENT_SERIAL,那么为 CObject 重载的插入操作符将调用保护型的 WriteObject。此函数随后会调用该类的 Serialize 函数。

返回值 一个 CArchive 引用,允许在一行中使用多个插入操作符。

参 见 CArchive::WriteObject, CObject::Serialize。

CArchive::operator > >

```
friend CArchive& operator > > ( CArchive& ar, CObject * & pOb);
throw( CArchiveException, CFileException, CMemoryException );
friend CArchive& operator > > ( CArchive& ar, const CObject * & pOb );
throw( CArchiveException, CFileException, CMemoryException );
CArchive& operator > > ( BYTE& by );
throw( CArchiveException, CFileException );
CArchive& operator > > ( WORD& w );
throw( CArchiveException, CFileException );
CArchive& operator > > ( int& i );
throw( CArchiveException, CFileException );
CArchive& operator > > ( LONG& l );
throw( CArchiveException, CFileException );
CArchive& operator > > ( DWORD& dw );
throw( CArchiveException, CFileException );
CArchive& operator > > ( float& f );
throw( CArchiveException, CFileException );
CArchive& operator > > ( double& d );
throw( CArchiveException, CFileException );
```

从档案中装入显式对象或基本数据类型。

如果在类实现中使用了宏 IMPLEMENT _ SERIAL,那么为 CObject 重载的抽取操作符将调用保护型的函数 ReadObject(以非 0 的运行时类指针为参数)。此函数随后会调用该类的 Serialize 函数。

返回值 CArchive 的引用,允许在一行中使用多个抽取操作符。

参 见 CArchive::ReadObject, CObject::Serialize。

数据成员

CArchive::m _ pDocument

默认值为空。这一指向 CDocument 的指针能被设置为 CArchive 实例用户需要的任何值。该指针通常用于向要串行化的对象传送串行化过程的附加信息,这是通过利用正在串行化的文档(一个 CDocument 的派生类)初始化该指针来实现的。如果需要的话,文档里的对象可通过这种方式访问文档。

在串行化过程中,该指针还可被 COleClientItem 对象使用。当用户发出一个打开或关闭文件命令时,工作框架将 m _ pDocument 设置为正串行化的文档。如果程序员因为其他非文件打开和保存命令而串行化一个对象链接和嵌入(OLE)的容器文档,必须明确地设置 m _ pDocument。例如:当串行化一个容器文档到粘贴板时可能需要这样做。

参 见 CDocument, COleClientItem。

CArchiveException # include < afx.h >

CArchiveException 对象说明存在串行化异常。CArchiveException 类包含了一个说明异常原因的公共数据成员。

CArchiveException 对象是在 CArchive 成员函数内构造或激发的。可通过 CATCH 宏来访问 CArchhiveException 对象。

参 见 CArchive, AfxThrowArchiveException, Exception Processing

成员函数

CArchiveException::CArchiveException

```
CArchiveException( int cause = CArchiveException::none );
```

参 数 cause 一个说明异常处理原因的枚举类型变量。要了解枚举列表,参见 m _ cause 数据成员。

构造保存原因值的 CArchiveException 对象。CArchiveException 对象既可以在堆中自己创建并引发,也可以利用全局函数 AfxThrowArchiveException 来创建。

一般不直接使用这个构造函数,而是调用全局函数 AfxThrowArchiveException。

CArchiveException::m _ cause

用于指明异常发生的原因。该成员变量是枚举型。

该枚举型的值如下:

CArchiveException::none	无错误发生
CArchiveException::generic	不确定错误
CArchiveException::readOnly	试图向只读文档写入
CArchiveException::endOfFile	读入对象时读到文件末尾
CArchiveException::writeOnly	试图从只为写入而打开的文档中读出数据
CArchiveException::badIndex	非法文件格式
CArchiveException::badClass	试图将对象读到一个错误类型对象中
CArchiveException::badSchema	试图读一个不同版本的类的对象

注意:CArchiveException 枚举类型的值与 CFileException 是有区别的。

CArray # include < afxtempl.h >

```
template < class TYPE, class ARG _ TYPE > class CArray : public CObject
```

参 数 TYPE 模板参数,用来指明存储在数组中的对象的类型,是 CArray 返回的参数。

ARG _ TYPE 用来访问存储在数组中对象的参数类型。传递到 CArray 的参数通常是 TYPE 的引用。

CArray 类支持的数组与 C 语言中的数组相似,但可以按需要动态减小和扩大。

数组下标总是从 0 开始,上界既可以固定,也可以不断扩大以存储更多的元素。即使中间一些元素是空的,CArray 也会按上界连续分配内存地址。

在使用数组前,应先用 SetSize 设定数组的大小并分配好内存地址。如果没有使用 SetSize,在数组中增加元素时就会造成不断地重分配内存和拷贝。不断地重分配内存和拷

贝效率很低并会造成内存碎片。

当需要转储数组中的某一个元素时，必须将 CDumpContext 对象的深度设置为大于等于 1 的数。数组一些成员函数需要调用全局帮助函数，在大部分使用 CArray 类的场合必须对全局帮助函数进行定制。

当从 CArray 对象中删去元素时，会调用帮助函数 DestructElements。而要增加元素时，则会调用帮助函数 ConstructElements。

数组类的派生类类似于链表类的派生类。

参　见　CObArray，DestructElements，ConstructElements，Collection Class Helpers。

成员函数

CArray::Add

int Add(ARG_TYPE newElement);

throw(CMemoryException);

参　数　**ARG_TYPE**　模板参数，用于指定数组引用元素参数的类型。

newElement　添加到数组的元素。

在数组末尾添加一个新元素，数组大小增加 1。若已使用 SetSize 将数组的大小设置成大于 1，那么可能要额外分配内存。但数组上界仅增加 1。

返回值　添加元素的索引值。

参　见　CArray::SetAt，CArray::SetAtGrow，CArray::InsertAt，CArray::operator [ ]。

CArray::Append

int Append(const CArray& src);

参　数　**src**　要添加到当前数组的元素源。

该成员函数用于将一个数组全部加到另一个数组的末尾，应注意这两个数组必须为同一类型。Append 会对添加到数组中的元素自动分配内存。

返回值　第一个被添加元素的索引值。

参　见　CCArray::Copy。

CArray::CArray

CArray( );

构造空数组。数组一次增加一个元素。

参　见　CObArray::CobArray。

CArray::Copy

void Copy(const CArray& src);

参　数　**src**　拷贝到数组的元素源。

这个成员函数可以把一个数组的元素拷贝到另一个数组中。该函数会用一个数组的元素覆盖另一个数组的元素。拷贝不释放内存。如果必要，需额外分配内存以保证元素都能拷贝到数组中。

参　见　CArray::Append。

CArray::ElementAt

TYPE& ElementAt(int nIndex);

参　数　**TYPE**　模板参数，指定了数组元素的类型。

nIndex　整数索引值，大于或等于 0 并且小于或等于 GetUpperBound 返回值。

返回数组内的指定元素的临时引用。用来实现数组左边赋值操作符。

返回值　指定数组元素的引用。

参　见　CArray::operator [ ]。

CArray::FreeExtra

void FreeExtra( );

释放增大数组时额外分配的内存。这个函数对数组的大小和上界没有影响。

CArray::GetAt

TYPE GetAt(int nIndex) const;

参　数　**TYPE**　模板参数，指定数组元素的类型。

nIndex　整数索引值，大于或等于 0 并且小于或等于 GetUpperBound 返回值。

按指定索引返回数组元素。

注意：若出现负值或比 GetUpperBound 返回的值大时，则会导致错误断言。

返回值　当前索引的数组元素。

参　见　CArray::SetAt，CArray::operator [ ]，ConstructElements。

CArray::GetData

const TYPE * GetData( ) const;

TYPE * GetData( );

参　数　**TYPE**　模板参数，指定数组元素类型。

用这个成员函数可直接访问数组元素。若没有元素可访问，则 GetData 返回空值。直接访问数组元素可使工作更快，当调用 GetData 时要小心，用户制造的任何错误都会影响数组元素。

返回值　指向数组元素的指针。

参　见　CArray::GetAt，CArray::SetAt，CArray::ElementAt。

CArray::GetSize

int GetSize( ) const;

返回数组的大小。因为索引从 0 开始，其值比最大的索引值大 1。

参　见　CArray::GetUpperBound，CArray::SetSize。

CArray::GetUpperBound

int GetUpperBound( ) const;

用于返回数组的上界。因为数组索引从 0 开始，这个函数返回值比 GetSize 少 1。当 GetUpperBound 返回 -1 时，说明数组中没有元素。

参　见　CArray::GetSize，CArray::SetSize。

CArray::InsertAt

void InsertAt(int nIndex, ARG_TYPE newElement, int nCount = 1);

throw(CMemoryException);

void InsertAt(int nStartIndex, CArray * pNewArray);
throw(CMemoryException);
参 数 nIndex 整数索引值,可能比 GetUpperBound 返回值要大。
ARG _ TYPE 模板参数,指定数组元素类型。
newElement 要插入数组中的元素。
nCount 元素插入的次数(默认为 1)。
nStartIndex 整数索引值,可能比 GetUpperBound 返回值要大。
pNewArray 一个数组,包含了要加入到数组中的元素。

前一个 InsertAt 函数是在数组的指定索引处 InsertAt 插入一个元素(或一个元素的多个拷贝),并将后面的元素依次向上移动。后一个 InsertAt 是在 nStartIndex 指定的索引处将另一个 CArray collection 所有元素插入到当前数组中。
参 见 GetUpperBound, CArray::SetAt, CArray::RemoveAt。

CArray::RemoveAll
void RemoveAll( );
将数组中的所有元素删除。

CArray::RemoveAt
void RemoveAt(int nIndex, int nCount = 1);
参 数 nIndex 整数索引值,大于或等于 0 并且小于或等于 GetUpperBound 返回值。
nCount 要删除的元素的数目。

从数组指定的索引处删除元素。被删除元素后的所有元素会向下移动。数组的上界会减小但内存不会释放。

如果要删除的元素数大于数组中 nIndex 之后的元素数,Debug 版将会发出警告。
参 见 CArray::SetAt, CArray::SetAtGrow, CArray::InsertAt。

CArray::SetAt
void SetAt(int nIndex, ARG _ TYPE newElement);
参 数 nIndex 整数索引值,大于或等于 0 并且小于或等于 GetUpperBound 返回值。
ARG _ TYPE 模板参数,指定引用数组元素的参数类型。
newElement 要放置在指定位置的新元素。

在指定索引处设置数组元素。SetAt 不会使数组增加,要使数组自动增加可以调用 SetAtGrow。指定的索引值应在数组上界内,否则 Debug 会发出警告。
参 见 CArray::GetAt, CArray::SetAtGrow, CArray::ElementAt, CArray::operator [ ]。

CArray::SetAtGrow
void SetAtGrow(int nIndex, ARG _ TYPE newElement);
throw(CMemoryException);
参 数 nIndex 大于或等于 0 的整数索引值。
ARG _ TYPE 模板参数,指定了数组元素的类型。
newElement 要添加到数组中的元素,可以是空值。

在指定索引处设置数组元素。如有必要,数组大小会自动增加。
参 见 CArray::GetAt, CArray::SetAt, CArray::ElementAt, CArray::operator [ ]。

CArray::SetSize
void SetSize(int nNewSize, int nGrowBy = -1);
throw(CMemoryException);
参 数 nNewSize 数组的大小,必须大于或等于 0。
nGrowBy 在需要增加元素的情况下,要分配的元素数的最小数目。

为一个数组设定大小,如有必要会分配内存。如果所设置的大小比原来的小,那么数组会被截短,不用的内存将被释放掉。在使用数组前应该用这个函数设置数组大小。若不用 SetSize,在数组中增加元素时会导致频繁的重新分配内存和拷贝。频繁的重新分配内存和拷贝效率低并造成内存碎片。当数组的大小增加时,nGrowBy 参数会影响内部内存分配,但它绝不会影响 GetSize 和 GetUpperBound 返回的数组大小。若 nGrowBy 使用默认值,MFC 通过优化来决定如何分配内存。
参 见 CArray::GetUpperBound, CArray::GetSize。

CArray::operator [ ]
TYPE& operator [ ](int nIndex);
TYPE operator [ ](int nIndex) const;
参 数 TYPE 模板参数,指定数组元素的类型。
nIndex 被访问元素的索引值。

用[]操作符可以代替 SetAt 和 GetAt 函数。如果不是常量数组,[]操作符既可用在赋值语句的左边也可用在右边。但若是常量数组,[]操作符只能用在右边。
参 见 CArray::GetAt, CArray::SetAt, CArray::ElementAt。

CBitmap # include < afxwin.h >

CBitmap 类封装了 Windows 图形设备界面(GDI)位图,提供了处理位图的成员函数。

成员函数

CBitmap::CBitmap
CBitmap( );
构造函数,创建 CBitmap 对象。
参 见 CBitmap::LoadBitmap, CBitmap::LoadOEMB itmap, CBitmap:: CreateBitmap, CBitmap:: CreateBitmapIndirect, CBitmap:: CreateCompatibleBitmap, Bitmap:: CreateDiscardabeBitmap。

CBitmap::CreateBitmap
BOOL CreateBitmap(int nWidth, int nHeight, UINT nPlanes, UINT nBitcount, const void * lpBits);
参 数 nWidth 指定位图的宽度(以像素为单位)。
nHeight 指定位图的高度(以像素为单位)。
nPlanes 指定位图中颜色平面的数目。

nBitcount 指定每个显示像素中颜色位的数目。

lpBits 指向一短整形数组,包含初始位图位的值,若该参数值为空,则新位图不被初始化。

初始化指定宽度、高度及式样的设备相关内存位图。彩色位图中 nPlanes 或 nBitcount 应设为 1。如果这两个参数都被设为 1,将创建单色位图。虽然不能直接把位图拷贝到显示设备上,但是通过 CDC::SelectObject 将位图选进内存设备场境后,可以使用 CDC::BitBlt 函数将它从内存设备场境拷贝到任何兼容的设备场境中,包括显示场境。必须首先把位图从设备场境中选出,然后才能删除 CreateBitmap 函数创建的 CBitmap 对象。

返回值 函数调用成功返回非 0 值;否则返回值为 0。

参 见 CDC:: SelectObject, CGdiObject:: DeleteObject, CDC::BitBlt, ::CreateBitmap。

CBitmap::CreateBitmapIndirect

BOOL CreateBitmapIndirect(LPBITMAP lpBitmap);

参 数 lpBitmap 指向 BITMAP 结构的指针,该结构包含位图的高度、宽度和式样。

初始化由 lpBitmap 指向的结构中给定宽度、高度及式样的位图。虽然不能直接把位图拷贝到显示设备上,但是通过 CDC::SelectObject 将位图选进内存设备场境后,可以使用 CDC::BitBlt,函数将它从内存设备场境拷贝到任何兼容的设备场境中,包括显示场境(CDC::PatBlt 函数可将当前画刷的位图直接拷贝到显示设备场境中)。

如果 lpBitmap 指向的 BITMAP 结构已经通过 GetObject 函数进行了填充,则位图的一些位并未指定,这时位图是未初始化的。为了进行初始化,在应用程序中,可以通过诸如 CDC::BitBlt 或 CDC::SetDIBits 等函数,把位从 CGdiObject::GetObject 的第一个参数所标识的位图拷贝到由 CreateBitmapIndirect 创建的位图中。必须首先把位图从设备场境中选出,然后才能删除 CreateBitmap 函数创建的 CBitmap 对象。

返回值 函数调用成功返回非 0 值;否则返回值为 0。

参 见 CDC:: SelectObject, CDC:: BitBlt, CGdiObject:: DeleteObject, CGdiObject::GetObject, ::CreateBitmapIndirect。

CBitmap::CreateCompatibleBitmap

BOOL CreateCompatibleBitmap(CDC * pDC, int nWidth, int nHeight);

参 数 pDC 指定设备场境。

nWidth 指定位图宽度(以像素为单位)。

nHeight 指定位图高度(以像素为单位)。

初始化与 pDC 指定设备具有同样颜色平面或同样像素位格式的位图。如果 pDC 指向内存设备场境,则返回的位图与当前在设备场境中选择的位图具有相同的形式。用它在内存中准备图像,然后再拷贝到兼容设备的实际显示平面。创建内存设备场境后,GDI 自动为它选择一单色位图。因为彩色内存设备场境有彩色和单色位图两种选择,所以 CreateCompatibleBitmap 函数返回的位图形式是不同的。但是,非内存设备场境的兼容位图却总是以设备位图的形式出现。必须首先把位图从设备场境中选出,然后才能删除 CreateBitmap 函数创建的 CBitmap 对象。

返回值 函数调用成功返回非 0 值;否则返回值为 0。

参 见 ::CreateCompatibleBitmap, CGdiObject::DeleteObject。

CBitmap::CreateDiscardableBitmap

BOOL CreateDiscardableBitmap(CDC * pDC, int nWidth, int nHeight);

参 数 pDC 指定设备场境。

nWidth 指定位图宽度(以位为单位)。

nHeight 指定位图高度(以位为单位)。

初始化与 pDC 指定设备场境兼容的可丢弃位图。位图具有与指定设备场境相同颜色平面数或相同的像素位方式。只有应用程序未把位图选入显示场境时,Windows 才可删除此函数创建的位图。如果 Windows 删除了未选中的位图,当应用程序再选用它时,CDC::SelectObject 函数将返回空。必须首先把位图从设备场境中选出,然后才能删除 CreateBitmap 函数创建的 CBitmap 对象。

返回值 函数调用成功返回非 0 值;否则返回值为 0。

参 见 ::CreateDiscardableBitmap, CGdiObject::DeleteObject。

CBitmap::FromHandle

static CBitmap * PASCAL FromHandle(HBITMAP hBitmap);

参 数 hBitmap 指定 Windows GDI 位图句柄。

获取与指定位图句柄相关联的 CBitmap 对象指针。如果指定句柄没有相关联的 CBitmap 对象,将创建一个临时 Cbitmap 对象与之关联。

返回值 函数调用成功返回指向 CBitmap 对象的指针;否则返回值为空。

CBitmap::GetBitmap

int GetBitmap(BITMAP * pBitMap);

参 数 pBitMap 指向 BITMAP 结构的指针,不能为空。

获取 CBitmap 相关信息。信息由 pBitmap 指定的 BITMAP 结构返回。

返回值 若检索到 CBitmap 信息返回非 0 值;否则返回值为 0。

参 见 BITMAP。

CBitmap::GetBitmapBits

DWORD GetBitmapBits(DWORD dwCount, LPVOID lpBits) const;

参 数 dwCount 指定要拷贝的字节数。

lpBits 指向接收位图的缓冲区,位图是字节数组,这个数组形成的水平扫描行是 16 的倍数。

将指定位图的位拷贝到 lpBits. 指向的缓冲区,参数 dwCount 指定要拷贝的字节数。可以通过 CGdiObject::GetObject 来确定给定位图的正确 dwCount 值。

返回值 返回位图中的实际字节数;出错时返回值为 0。

参 见 CGdiObject::GetObject, ::GetBitmapBits。

CBitmap::GetBitmapDimension

CSize GetBitmapDimension( ) const;

返回位图的宽度和高度。假设宽度和高度已经由 SetBitmapDimension 成员函数设置。

返回值 返回 CSize 对象,高度在 CSize 对象的 cy 成员中,宽度在 cx 成员中,以 0.1mm 为单位。如果位图的宽度和高度没有设置,则返回值为 0。

参 见 CBitmap::SetBitmapDimension。

CBitmap::LoadBitmap

BOOL LoadBitmap(LPCTSTR lpszResourceName);

BOOL LoadBitmap(UINT nIDResource);

参 数 lpszResourceName 指向以 null 结尾的字符串的指针变量,指定资源名。

nIDResource 指定位图资源 ID 值。

加载位图资源。位图由 lpszResourceName 命名或由应用程序可执行文件的 nIDResource 中 ID 值标识。加载后的位图与 CBitmap 对象关联。如果由 lpszResourceName 标识的位图不存在或没有足够内存加载位图时,函数返回值为 0。可用 CGdiObject::DeleteObject 函数或 CBitmap 析构函数,来删除 LoadBitmap 函数加载的位图。

返回值 若加载成功返回非 0 值;否则返回值为 0。

参 见 CBitmap::LoadOEMBitmap,::LoadBitmap, CGdiObject::DeleteObject。

CBitmap::LoadMappedBitmap

BOOL LoadMappedBitmap(UINT nIDBitmap, UINT nFlags = 0, LPCOLORMAP lpColorMap = NULL, int nMapSize = 0);

参 数 nIDBitmap 指定位图资源 ID 值。

nFlags 指定位图标志,可以是 0 或 CMB - MASKED。

lpColorMap 指向用于映射位图的 COLORMAP 结构的指针,如果这个参数值为空,函数就使用默认颜色映射。

加载位图,并将颜色映射到当前系统的颜色。默认情况下,LoadMappedBitmap 将映射按钮字型中经常使用的颜色。

返回值 若加载成功返回非 0 值;否则返回值为 0。

参 见 ::LoadBitmap,::CreateMappedBitmap。

CBitmap::LoadOEMBitmap

BOOL LoadOEMBitmap(UINT nIDBitmap);

参 数 nIDBitmap 指定位图资源 ID 值,取值为预定义的 Windows 位图 ID 值,下面列出了 WINDOWS.H 中所有的值:

OBM_BTNCORNERS	OBM_OLD_RESTORE
OBM_BTSIZE	OBM_OLD_RGARROW
OBM_CHECK	OBM_OLD_UPARROW
OBM_CHECKBOXES	OBM_OLD_ZOOM
OBM_CLOSE	OBM_REDUCE
OBM_COMBO	OBM_REDUCED
OBM_DNARROW	OBM_RESTORE
OBM_DNARROWD	OBM_RESTORED
OBM_DNARROWI	OBM_RGARROW
OBM_LFARROW	OBM_RGARROWD
OBM_LFARROWD	OBM_RGARROWI
OBM_LFARROWI	OBM_SIZE
OBM_MNARROW	OBM_UPARROW
OBM_OLD_CLOSE	OBM_UPARROWD
OBM_OLD_DNARROW	OBM_UPARROW
OBM_OLD_LFARROW	OBM_ZOOM
OBM_OLD_REDUCE	OBM_ZOOMD

加载 Windows 预定义的位图。以 OBM_OLD 开头的位图名代表由 Windows 3.0 以前的版本使用。常数 OEMRESOUR CE 必须在 # include WINDOWS.H 语句之前使用,这样就能使用任何 OBM_常数。

返回值 若加载成功返回非 0 值;否则返回值为 0。

参 见 CBitmap::LoadBitmap,::LoadBitmap。

CBitmap::operator HBITMAP

operator HBITMAP( ) const;

操作符,获取与 CBitmap 对象关联的 Windows GDI 句柄。这个操作符是类型强制转换型的,所以可直接用于 HBITMAP 对象。

返回值 函数调用成功返回与 CBitmap 对象关联的 Windows GDI 句柄;否则返回值空。

CBitmap::SetBitmapBits

DWORD SetBitmapBits(DWORD dwCount, const void * lpBits);

参 数 dwCount 指定 lpBits 指向的缓冲区的字节数。

lpBits 指向 BYTE 数组的指针,指定拷贝到 CBitmap 对象的位值。

将指定位图的位设置成 lpBits 指定值。

返回值 函数调用成功返回用于设置位图位的字节数;否则返回值为 0。

参 见 CBitmap::SetBitmapBits。

CBitmap::SetBitmapDimension

CSize SetBitmapDimension(int nWidth, int nHeight);

参 数 nWidth 指定位图的宽度(单位 0.1mm)。

nHeight 指定位图的高度(单位 0.1mm)。

设置位图的宽度和高度。GDI 并不使用这些数值,只是当应用程序调用 GetBitmapDimension 成员函数时将它们返回。

返回值 返回设置前位图的尺寸。高度存在 CSize 对象的 cy 成员中,宽度存在 cx 成员变量中。

参 见 CBitmap::GetBitmapDimension。

CBrush # include < afxwin.h >

CBrush 类封装了 Windows 图形设备界面(GDI)的画刷。画刷可以是实心的,带阴影线的或其他图形的。

参 见 CBitmap, CDC。