

Contents

以 C++/CLI 進行 .NET 程式設計

逐步解說：編譯以 CLR 為目標的 C++/CLI 程式

C++/CLI 工作

C++/CLI 工作

作法：建立 CLR 空專案

作法：建立 CLR 主控台應用程式

作法：在 C++/CLI 中使用追蹤參考

作法：在 C++/CLI 中使用陣列

作法：定義與使用類別和結構

參考型別的 C++ 堆疊語意

使用者定義的運算子

使用者定義轉換

initonly

作法：定義和使用委派

作法：在 C++/CLI 中定義和使用列舉

作法：在 C++/CLI 中使用事件

作法：定義介面靜態建構函式

作法：在原生編譯中宣告覆寫規範

作法：在 C++/CLI 中使用屬性

作法：在 C++/CLI 中使用 safe_cast

規則運算式

檔案處理和 I/O

圖形作業

Windows 作業

使用 ADO.NET 進行資料存取

原生和 .NET 互通性

原生和 .NET 互通性

與其他 .NET 語言間的互通性

混合（原生和受控）組件

[混合 \(原生和受控\) 組件](#)

[作法:移轉至 -clr](#)

[作法:使用 -clr 編譯 MFC 和 ATL 程式碼](#)

[混合組件的初始化](#)

[混合組件的程式庫支援](#)

[Interop 的效能考量 \(C++\)](#)

[應用程式定義域和 Visual C++](#)

[Double Thunking \(C++\)](#)

[當使用以 /clr 所建置 COM 物件時防止 CLR 關閉的例外狀況](#)

[作法:移除對 CRT 程式庫 DLL 的相依性以建立部分信任應用程式](#)

[在 MFC 中使用 Windows Form 使用者控制項](#)

[在 MFC 中使用 Windows Form 使用者控制項](#)

[Windows Forms/MFC 程式設計的差異](#)

[將 Windows Form 使用者控制項裝載至 MFC 對話方塊中](#)

[將 Windows Form 使用者控制項裝載至 MFC 對話方塊中](#)

[作法:建立使用者控制項並裝載至對話方塊中](#)

[作法:使用 Windows Forms 執行 DDX/DDV 資料繫結](#)

[作法:從原生 C++ 類別接收 Windows Forms 事件](#)

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

[作法:建立使用者控制項並裝載 MDI 檢視](#)

[作法:新增命令傳送至 Windows Forms 控制項](#)

[作法:呼叫 Windows Forms 控制項的屬性和方法](#)

[將 Windows Form 使用者控制項裝載成 MFC 對話方塊](#)

[從受控碼呼叫原生函式](#)

[從受控碼呼叫原生函式](#)

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

[作法:使用 PInvoke 從受控碼呼叫原生 DLL](#)

[作法:使用 PInvoke 封送處理字串](#)

[作法:使用 PInvoke 封送處理結構](#)

[作法:使用 PInvoke 封送處理陣列](#)

作法: 使用 PInvoke 封送處理函式指標

作法: 使用 PInvoke 封送處理內嵌指標

使用 C++ Interop (隱含 PInvoke)

 使用 C++ Interop (隱含 PInvoke)

 作法: 使用 C++ Interop 封送處理 ANSI 字串

 作法: 使用 C++ Interop 封送處理 Unicode 字串

 作法: 使用 C++ Interop 封送處理 COM 字串

 作法: 使用 C++ Interop 封送處理結構

 作法: 使用 C++ Interop 封送處理陣列

 作法: 使用 C++ Interop 封送處理回呼和委派

 作法: 使用 C++ Interop 封送處理內嵌指標

 作法: 擴充封送處理程式庫

 作法: 存取 System::String 中的字元

 作法: 將 char * 字串轉換為 System::Byte 陣列

 作法: 將 System::String 轉換為 wchar_t* 或 char*

 作法: 將 System::String 轉換為標準字串

 作法: 將標準字串轉換為 System::String

 作法: 取得位元組陣列的指標

 作法: 將非受控資源載入至位元組陣列

 作法: 修改原生函式中的參考類別

 作法: 判斷映像為原生或 CLR

 作法: 將原生 DLL 新增至全域組件快取

 作法: 以原生類型存放實值型別的參考

 作法: 在非受控記憶體中存放物件參考

 作法: 偵測 -clr 編譯

 作法: 在 System::Guid 和 _GUID 之間轉換

 作法: 指定 out 參數

 作法: 在 -clr 編譯中使用原生類型

 作法: 以原生類型宣告控制代碼

 作法: 包裝原生類別以供 C# 使用

純粹和可驗證的程式碼

 純粹和可驗證的程式碼

作法:建立可驗證的 C++ 專案
搭配 SQL Server 使用可驗證的組件
將專案從混合模式轉換為純中繼語言
序列化
Friend 組件 (C#)
受控類型
反射
強式名稱組件 (組件簽署)
偵錯類別
STL/CLR 程式庫參考
 STL/CLR 程式庫參考
 cliext 命名空間
 STL/CLR 容器
 STL/CLR 容器項目的需求
作法:從 .NET 集合轉換為 STL/CLR 容器
作法:從 STL/CLR 容器轉換為 .NET 集合
作法:從組件公開 STL/CLR 容器
for each, in
 adapter (STL/CLR)
 algorithm (STL/CLR)
 deque (STL/CLR)
 functional (STL/CLR)
 hash_map (STL/CLR)
 hash_multimap (STL/CLR)
 hash_multiset (STL/CLR)
 hash_set (STL/CLR)
 list (STL/CLR)
 map (STL/CLR)
 multimap (STL/CLR)
 multiset (STL/CLR)
 numeric (STL/CLR)
 priority_queue (STL/CLR)

[queue \(STL/CLR\)](#)

[set \(STL/CLR\)](#)

[stack \(STL/CLR\)](#)

[utility \(STL/CLR\)](#)

[vector \(STL/CLR\)](#)

[C++ 支援程式庫](#)

[C++ 支援程式庫](#)

[C++ 中的封送處理概觀](#)

[C++ 中的封送處理概觀](#)

[marshal_as](#)

[marshal_context 類別](#)

[msclr 命名空間](#)

[資源管理類別](#)

[資源管理類別](#)

[auto_gcroot](#)

[auto_gcroot](#)

[auto_gcroot 類別](#)

[swap 函式 \(auto_gcroot\)](#)

[auto_handle](#)

[auto_handle](#)

[auto_handle 類別](#)

[swap 函式 \(auto_handle\)](#)

[同步處理 \(lock 類別\)](#)

[同步處理 \(lock 類別\)](#)

[鎖定](#)

[鎖定](#)

[lock 類別](#)

[lock_when 列舉](#)

[呼叫特定應用程式定義域的函式](#)

[呼叫特定應用程式定義域的函式](#)

[call_in_appdomain 函式](#)

[com::ptr](#)

com::ptr

com::ptr 類別

C++/CLI 中的例外狀況

C++/CLI 中的例外狀況

使用受控例外狀況的基本概念

在 /clr 之下例外狀況處理行為的差異

finally

作法：攔截 MSIL 所擲回機器碼的例外狀況

作法：定義與安裝全域例外狀況處理常式

Boxing

Boxing

作法：明確要求 Boxing

作法：使用 gcnew 建立實值型別及使用隱含 Boxing

作法：Unbox

標準轉換和隱含 Boxing

以 C++/CLI 進行 .NET 程式設計

2020/12/10 • [Edit Online](#)

根據預設，使用 Visual Studio 2015 所建立的 CLR 專案會以 .NET Framework 4.5.2 為目標。當您建立新專案時，可以將目標設為 .NET Framework 4.6。在 [新增專案] 對話方塊中，變更對話方塊頂端中間的下拉式清單中的目標 framework。若要變更現有專案的目標 framework，請關閉專案、編輯專案檔 (`.vcxproj`)，然後將目標 Framework 版本的值變更為 4.6。變更會在您下次開啟專案時生效。

在 Visual Studio 2017 中，預設的目標 .NET Framework 是 4.6.1。[Framework 版本選取器] 位於 [新增專案] 對話方塊的底部。

在 Visual Studio 2017 中安裝 c++/CLI 支援

當您安裝 Visual Studio c++ 工作負載時，預設不會安裝 c++/CLI 本身。若要在安裝 Visual Studio 之後安裝元件，請開啟 Visual Studio 安裝程式。選擇您安裝的 Visual Studio 版本旁的 [修改] 按鈕。選取 [已安裝的元件] 索引標籤。向下捲到 [編譯器]、[組建工具] 和 [運行 時間] 區段，然後選取 [c++/cli 支援] 選擇 [修改] 以更新 Visual Studio。

在 Visual Studio 2019 中，.NET Core 專案的預設目標 framework 是 5.0。針對 .NET Framework 專案，預設值為 4.7.2。.NET Framework 版本選取器位於 [建立新專案] 對話方塊的 [設定您的新專案] 頁面上。

在 Visual Studio 2019 中安裝 c++/CLI 支援

當您安裝 Visual Studio c++ 工作負載時，預設不會安裝 c++/CLI 本身。若要在安裝 Visual Studio 之後安裝元件，請開啟 Visual Studio 安裝程式。選擇您安裝的 Visual Studio 版本旁的 [修改] 按鈕。選取 [已安裝的元件] 索引標籤。向下滾動至 [編譯器]、[組建工具] 和 [運行 時間] 區段，然後選取最新的 c++/cli 支援適用於 v142 build tools 元件。選擇 [修改] 以更新 Visual Studio。

本節內容

[C++/CLI 工作](#)

[原生和 .NET 互通性](#)

[純和可驗證的程式碼 \(c++/CLI\)](#)

[\(c++/CLI\) 的正則運算式](#)

[檔案處理和 i/o \(c++/CLI\)](#)

[\(c++/CLI\) 的圖形作業](#)

[Windows 作業 \(c++/CLI\)](#)

[使用 ADO.NET 的資料存取 \(c++/CLI\)](#)

[與其他 .NET 語言的互通性 \(c++/CLI\)](#)

[序列化 \(C++/CLI\)](#)

[Managed 類型 \(c++/CLI\)](#)

[反映 \(C++/CLI\)](#)

[強式名稱元件 \(元件簽署\) \(c++/CLI\)](#)

[Debug 類別 \(c + +/CLI\)](#)

[STL/CLR 程式庫參考](#)

[C++ 支援程式庫](#)

[C++/CLI 中的例外狀況](#)

[\(c + +/CLI\) 的裝箱](#)

另請參閱

[原生和 .NET 互通性](#)

逐步解說：在 Visual Studio 中編譯以 CLR 為目標的 C + +/CLI 程式

2020/11/2 • [Edit Online](#)

您可以使用 c + +/CLI 來建立 c + + 程式，以使用 .NET 類別和原生 c + + 類型。C + +/CLI 適用于主控台應用程式，以及包裝原生 c + + 程式碼並可從 .NET 程式存取的 DLL 中。若要建立以 .NET 為基礎的 Windows 使用者介面，請使用 c # 或 Visual Basic。

針對此程式，您可以輸入自己的 c + + 程式，或使用其中一個範例程式。我們在此程序中使用的範例程式會建立名為 `textfile.txt` 的文字檔，並將它儲存至專案目錄。

Prerequisites

- 對 C++ 語言基本知識的了解。
- 在 Visual Studio 2017 和更新版本中，c + +/CLI 支援是選擇性元件。若要安裝它，請從 Windows [開始] 功能表開啟 Visual Studio 安裝程式。確定已核取 [使用 c + + 進行桌面開發] 碑，並在 [選用元件] 區段中，檢查 c + +/cli 支援。

建立新專案

下列步驟會依您使用的 Visual Studio 版本而略有不同。若要查看您慣用 Visual Studio 版本的檔，請使用 **版本** 選擇器控制項。您可在此頁面的目錄頂端找到此檔案。

若要在 Visual Studio 2019 中建立 c + +/CLI 專案

1. 在 **方案總管** 中，以滑鼠右鍵按一下頂端以開啟 **[建立新專案]** 對話方塊。
2. 在對話方塊頂端的 **[搜尋]** 方塊中輸入 `clr`，然後從結果清單中選擇 **[CLR 空專案]**。
3. 選擇 **[建立]** 按鈕以建立專案。

若要在 Visual Studio 2017 中建立 c + +/CLI 專案

1. 建立新專案。在 **[檔案]** 功能表上，指向 **[開啟檔案]**，然後按一下 **[專案]**。
2. 從 Visual C++ 專案類型，按一下 **[CLR]**，然後按一下 **[CLR 空專案]**。
3. 鍵入專案名稱。根據預設，包含專案的方案與新專案具有相同的名稱，但您可以輸入不同的名稱。如果您想要，也可以輸入不同的專案位置。
4. 按一下 **[確定]** 建立新專案。

若要在 Visual Studio 2015 中建立 c + +/CLI 專案

1. 建立新專案。在 **[檔案]** 功能表上，指向 **[開啟檔案]**，然後按一下 **[專案]**。
2. 從 Visual C++ 專案類型，按一下 **[CLR]**，然後按一下 **[CLR 空專案]**。
3. 鍵入專案名稱。根據預設，包含專案的方案與新專案具有相同的名稱，但您可以輸入不同的名稱。如果您想要，也可以輸入不同的專案位置。
4. 按一下 **[確定]** 建立新專案。

新增原始檔

1. 如果未顯示 **[方案總管]**，請按一下 **[檢視]** 功能表上的 **[方案總管]**。

2. 新增原始程式檔至專案：

- 以滑鼠右鍵按一下 方案總管 中的 [來源 檔案] 資料夾，指向 [加入]，然後按一下 [新增專案]。
- 按一下 [C++ 檔 (.cpp)] 並鍵入檔案名稱，然後按一下 [新增]。

.Cpp 檔會出現在 方案總管 的 [原始 程式檔] 資料夾中，而且會出現索引標籤式視窗，讓您在該檔案中輸入想要的程式碼。

3. 按一下 Visual Studio 中新建立的索引標籤，然後鍵入有效的 Visual C++ 程式，或複製並貼上其中一個範例程式。

例如，您可以使用 [如何：寫入文字檔 \(C++/CLI\)](#) 範例程式 (位於《程式設計指南》的 檔案處理和 I/O 節點中)。

如果您使用範例程式，請注意，`gcnew` 當您建立 .net 物件時，會使用關鍵字而不是，而 `new` 會傳回 `gcnew` 控制碼 (`^`) 而非指標 (`*`)：

```
StreamWriter^ sw = gcnew StreamWriter(fileName);
```

如需 C++/CLI 語法的詳細資訊，請參閱 [執行時間平臺的元件擴充](#) 功能。

4. 在 [建置] 功能表上，按一下 [建置方案]。

[輸出] 視窗會顯示編譯進度的相關資訊，例如組建記錄檔的位置，以及指出組建狀態的訊息。

如果您進行變更，然後執行程式而不進行建置，對話方塊可能會指出專案已過期。如果您想要 Visual Studio 一律使用目前版本的檔案，而不是每次建置應用程式都提示您，請先選取此對話方塊上的核取方塊，再按一下 [確定]。

5. 在 [偵錯] 功能表上，按一下 [啟動但不偵錯]。

6. 如果您使用範例程式，當您執行程式時，就會顯示命令視窗指出已建立文字檔。

`textfile.txt` 文字檔現在位於專案目錄中。您可以使用 [記事本] 開啟此檔案。

NOTE

選擇空白的 CLR 專案範本時，會自動設定 `/clr` 編譯器選項。若要進行確認，請以滑鼠右鍵按一下 [方案總管] 中的專案並按一下 [屬性]，然後在 [組態屬性] 的 [一般] 節點中，核取 [Common Language Runtime 支援]。

請參閱

[C++ 語言參考](#)

[專案與建置系統](#)

C++/CLI 工作

2020/3/25 • [Edit Online](#)

本檔的這一節中的文章說明如何使用/Cli 的C++各種功能

本節內容

- [如何 : 建立 CLR 空專案](#)
- [如何 : 建立 CLR 主控台應用程式 \(C++/CLI\)](#)
- [如何 : 在 C++/CLI 中使用追蹤參考](#)
- [如何 : 在 C++/CLI 中使用陣列](#)
- [如何 : 定義與使用類別和結構 \(C++/CLI\)](#)
- [參考型別的 C++ 堆疊語意](#)
- [使用者定義的運算子 \(C++/CLI\)](#)
- [使用者定義轉換 \(C++/CLI\)](#)
- [initonly \(C++/CLI\)](#)
- [如何 : 定義和使用委派 \(C++/CLI\)](#)
- [如何 : 在 C++/CLI 中定義和使用列舉](#)
- [如何 : 在 C++/CLI 中使用事件](#)
- [如何 : 定義介面靜態建構函式 \(C++/CLI\)](#)
- [如何 : 在原生編譯中宣告覆寫指定名稱 \(C++/CLI\)](#)
- [如何 : 在 C++/CLI 中使用屬性](#)
- [如何 : 在 C++/CLI 中使用 safe_cast](#)

相關章節

- [以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

如何：建立 CLR 空專案

2020/4/15 • [Edit Online](#)

要建立 CLR 空專案,請使用CLR 空專案範本,可從「**新專案**」對話框中獲取。

NOTE

IDE 中功能的外觀可能取決於您的活動設置或版本,並且可能與"說明"中描述的功能不同。若要變更您的設定,請在 [工具]**** 功能表上選擇 [匯入和匯出設定]****。有關詳細資訊,請參閱[個人化可視化工作室 IDE](#)。

建立 CLR 空專案

1. 在 [檔案] 功能表上按一下 [新增], 然後按一下 [專案]。

此時會出現 [新增專案]**** 對話方塊。

2. 在 "已安裝範本" 下,按一下可視化C++ 節點;然後單擊CLR 節點。選擇CLR 空項目圖示。
3. 在 [名稱] **** 方塊中輸入應用程式的唯一名稱。

NOTE

您還可以從 ■ 對話方塊中指定其他專案和解決方案設置,但不需要這些設置。

4. 按一下 [確定]。

另請參閱

[Visual Studio 中的 C++ 專案類型](#)

[除錯C++專案](#)

如何：建立 CLR 主控台應用程式 (C++/CLI)

2020/12/10 • [Edit Online](#)

您可以使用 [新增專案] 對話方塊中的 [CLR 主控台應用程式] 範本，建立已經有基本專案參考和檔案的主控台應用程式專案。

您可以使用 [新增專案] 對話方塊中的 [CLR 主控台應用程式] 範本，建立已經有基本專案參考和檔案的主控台應用程式專案。

當您安裝 Visual Studio c++ 工作負載時，預設不會安裝 c++/CLI 支援。如果您在 [新增專案] 對話方塊中的 [Visual C++] 沒有看到 [CLR] 標題，您可能需要安裝 c++/cli 支援。如需詳細資訊，請參閱 [使用 c++/cli 進行 .NET 程式設計](#)。

您可以使用 [建立新專案] 對話方塊中的 [CLR 主控台應用程式 (.NET Framework)] 範本，建立已經有基本專案參考和檔案的主控台應用程式專案。

當您安裝 Visual Studio c++ 工作負載時，預設不會安裝 c++/CLI 支援。如果您在 [建立新專案] 對話方塊中看不到 CLR 專案範本，您可能需要安裝 c++/cli 支援。如需詳細資訊，請參閱 [使用 c++/cli 進行 .NET 程式設計](#)。

通常，主控台應用程式會編譯成獨立的可執行檔，但並不具圖形使用者介面。使用者會在命令提示字元中執行主控台應用程式。他們可以使用命令列來發出指示給執行中的應用程式。應用程式會在命令視窗中提供輸出資訊做為文字。主控台應用程式的立即意見反應，讓它成為學習程式設計的絕佳方法。您不需要擔心如何執行圖形化使用者介面。

當您使用 CLR 主控台應用程式範本建立專案時，它會自動加入這些參考和檔案：

- 這些 .NET Framework 命名空間的參考：
 - `System`, `System.Data`, `System.Xml` : 這些參考包含定義常用類型、事件、介面、屬性和例外狀況的基本類別。
 - `mscorLib.dll` : 支援 .NET Framework 開發的元件 DLL。
- 原始程式檔：
 - `ConsoleApplicationName.cpp` : 應用程式的主要來源檔案和進入點。這個檔案具有您為專案指定的基底名稱。它會識別專案 DLL 檔和專案命名空間。在這個檔案中提供您自己的程式碼。
 - `AssemblyInfo.cpp` : 包含屬性和設定，您可以使用這些屬性來修改專案的元件中繼資料。如需詳細資訊，請參閱 [元件內容](#)。
 - `stdafx.cpp` : 用來建立名為的先行編譯標頭檔，`ConsoleApplicationName.pch` 以及名為的先行編譯類型檔案 `stdafx.obj` 。
- 標頭檔：
 - `stdafx.h` : 用來建立名為的先行編譯標頭檔，`ConsoleApplicationName.pch` 以及名為的先行編譯類型檔案 `stdafx.obj` 。
 - `resource.h` : 產生的 include 檔 `app.rc` 。
- 資源檔：
 - `app.rc` : 程式的資源腳本檔案。
 - `app.ico` : 程式的圖示檔。

- `ReadMe.txt` : 描述專案中的檔案。

當您使用 CLR 主控台應用程式範本建立專案時，它會自動加入這些參考和檔案：

- 這些 .NET Framework 命名空間的參考：

- `System`, `System.Data` `System.Xml` : 這些參考包含定義常用類型、事件、介面、屬性和例外狀況的基本類別。
- `mscorlib.dll` : 支援 .NET Framework 開發的元件 DLL。

- 原始程式檔：

- `ConsoleApplicationName.cpp` : 應用程式的主要來源檔案和進入點。這個檔案具有您為專案指定的基本名稱。它會識別專案 DLL 檔和專案命名空間。在這個檔案中提供您自己的程式碼。
- `AssemblyInfo.cpp` : 包含屬性和設定，您可以使用這些屬性來修改專案的元件中繼資料。如需詳細資訊，請參閱 [元件內容](#)。
- `pch.cpp` : 用來建立名為的先行編譯標頭檔，`ConsoleApplicationName.pch` 以及名為的先行編譯類型檔案 `pch.obj`。

- 標頭檔：

- `pch.h` : 用來建立名為的先行編譯標頭檔，`ConsoleApplicationName.pch` 以及名為的先行編譯類型檔案 `pch.obj`。
- `Resource.h` : 產生的 include 檔 `app.rc`。

- 資源檔：

- `app.rc` : 程式的資源腳本檔案。
- `app.ico` : 程式的圖示檔。

若要建立 CLR 主控台應用程式專案

1. 在功能表列上，選擇 [檔案 > 新增 > 專案]。
2. 在 [新增專案] 對話方塊中，選取 [已安裝 的 > 範本] > Visual C++ > clr 節點，然後選取 [clr 主控台應用程式] 範本。
3. 在 [名稱] 方塊中輸入應用程式的唯一名稱。

您可以指定其他專案和方案設定，但不是必要的。

4. 選擇 [確定] 按鈕，以產生專案和來源檔案。

1. 在功能表列上，選擇 [檔案 > 新增 > 專案]。
2. 在 [新增專案] 對話方塊中，選取 已安裝 的 > Visual C++ > clr 節點，然後選取 [clr 主控台應用程式] 範本。
3. 在 [名稱] 方塊中輸入應用程式的唯一名稱。

您可以指定其他專案和方案設定，但不是必要的。

4. 選擇 [確定] 按鈕，以產生專案和來源檔案。

1. 在功能表列上，選擇 [檔案 > 新增 > 專案]。
2. 在 [建立新專案] 對話方塊中，于搜尋方塊中輸入「clr 主控台」。選取 (.NET Framework) 範本的 CLR 主

控台應用程式，然後選擇 [下一步]。

3. 在 [名稱] 方塊中輸入應用程式的唯一名稱。

您可以指定其他專案和方案設定，但不是必要的。

4. 選擇 [建立] 按鈕以產生專案和來源檔案。

另請參閱

[CLR 專案](#)

HOW TO : 使用追蹤參考的C++/CLI

2019/12/2 • [Edit Online](#)

這篇文章示範如何使用追蹤參考 (%) 在 C++/CLI 以傳址方式傳遞 common language runtime (CLR) 型別。

依參考傳遞 CLR 類型

下列範例示範如何依參考傳遞 CLR 型別搭配使用追蹤參考。

```
// tracking_reference_handles.cpp
// compile with: /clr
using namespace System;

ref struct City {
private:
    Int16 zip_;

public:
    City (int zip) : zip_(zip) {};
    property Int16 zip {
        Int16 get(void) {
            return zip_;
        } // get
    } // property
};

void passByRef (City ^% myCity) {
    // cast required so this pointer in City struct is "const City"
    if (myCity->zip == 20100)
        Console::WriteLine("zip == 20100");
    else
        Console::WriteLine("zip != 20100");
}

ref class G {
public:
    int i;
};

void Test(int % i) {
    i++;
}

int main() {
    G ^ g1 = gcnew G;
    G ^% g2 = g1;
    g1 -> i = 12;

    Test(g2->i); // g2->i will be changed in Test2()

    City ^ Milano = gcnew City(20100);
    passByRef(Milano);
}
```

```
zip == 20100
```

下一個範例會顯示該採取的追蹤參考的位址傳回 [interior_ptr \(C++/CLI\)](#)，並示範如何修改，並透過追蹤參考來存取資料。

```

// tracking_reference_data.cpp
// compile with: /clr
using namespace System;

public ref class R {
public:
    R(int i) : m_i(i) {
        Console::WriteLine("ctor: R(int)");
    }

    int m_i;
};

class N {
public:
    N(int i) : m_i (i) {
        Console::WriteLine("ctor: N(int i)");
    }

    int m_i;
};

int main() {
    R ^hr = gcnew R('r');
    R ^%thr = hr;
    N n('n');
    N %tn = n;

    // Declare interior pointers
    interior_ptr<R^> iphr = &thr;
    interior_ptr<N> ipn = &tn;

    // Modify data through interior pointer
    (*iphr)->m_i = 1;    // (*iphr)->m_i == thr->m_i
    ipn->m_i = 4;    // ipn->m_i == tn.m_i

    ++thr-> m_i;    // hr->m_i == thr->m_i
    ++tn. m_i;    // n.m_i == tn.m_i

    ++hr-> m_i;    // (*iphr)->m_i == hr->m_i
    ++n. m_i;    // ipn->m_i == n.m_i
}

```

```

ctor: R(int)
ctor: N(int i)

```

追蹤參考和內部指標

下列程式碼範例示範您可以追蹤參考和內部指標之間進行轉換。

```

// tracking_reference_interior_ptr.cpp
// compile with: /clr
using namespace System;

public ref class R {
public:
    R(int i) : m_i(i) {
        Console::WriteLine("ctor: R(int)");
    }

    int m_i;
};

class N {
public:
    N(int i) : m_i(i) {
        Console::WriteLine("ctor: N(int i)");
    }

    int m_i;
};

int main() {
    R ^hr = gcnew R('r');
    N n('n');

    R ^%thr = hr;
    N %tn = n;

    // Declare interior pointers
    interior_ptr<R^> iphr = &hr;
    interior_ptr<N> ipn = &n;

    // Modify data through interior pointer
    (*iphr)->m_i = 1;    // (*iphr)->m_i == thr->m_i
    ipn->m_i = 4;    // ipn->m_i == tn.m_i

    ++thr->m_i;    // hr->m_i == thr->m_i
    ++tn.m_i;    // n.m_i == tn.m_i

    ++hr->m_i;    // (*iphr)->m_i == hr->m_i
    ++n.m_i;    // ipn->m_i == n.m_i
}

```

```

ctor: R(int)
ctor: N(int i)

```

追蹤參考和實值型別

這個範例會示範簡單的 boxing 實值類型的追蹤參考透過：

```

// tracking_reference_valuetypes_1.cpp
// compile with: /clr

using namespace System;

int main() {
    int i = 10;
    int % j = i;
    Object ^ o = j;    // j is implicitly boxed and assigned to o
}

```

下一個範例會顯示您可以有追蹤參考和實值類型的原生參考。

```
// tracking_reference_valuetypes_2.cpp
// compile with: /clr
using namespace System;
int main() {
    int i = 10;
    int & j = i;
    int % k = j;
    i++; // 11
    j++; // 12
    k++; // 13
    Console::WriteLine(i);
    Console::WriteLine(j);
    Console::WriteLine(k);
}
```

```
13
13
13
```

下列範例會示範您可以使用追蹤參考，以及實值型別和原生型別。

```
// tracking_reference_valuetypes_3.cpp
// compile with: /clr
value struct G {
    int i;
};

struct H {
    int i;
};

int main() {
    G g;
    G % v = g;
    v.i = 4;
    System::Console::WriteLine(v.i);
    System::Console::WriteLine(g.i);

    H h;
    H % w = h;
    w.i = 5;
    System::Console::WriteLine(w.i);
    System::Console::WriteLine(h.i);
}
```

```
4
4
5
5
```

這個範例會示範您可以在記憶體回收堆積上結合實值類型的追蹤參考：

```
// tracking_reference_valuetypes_4.cpp
// compile with: /clr
using namespace System;
value struct V {
    int i;
};

void Test(V^ hv) {    // hv boxes another copy of original V on GC heap
    Console::WriteLine("Boxed new copy V: {0}", hv->i);
}

int main() {
    V v;    // V on the stack
    v.i = 1;
    V ^hv1 = v;    // v is boxed and assigned to hv1
    v.i = 2;
    V % trv = *hv1;    // trv is bound to boxed v, the v on the gc heap.
    Console::WriteLine("Original V: {0}, Tracking reference to boxed V: {1}", v.i, trv.i);
    V ^hv2 = trv;    // hv2 boxes another copy of boxed v on the GC heap
    hv2->i = 3;
    Console::WriteLine("Tracking reference to boxed V: {0}", hv2->i);
    Test(trv);
    v.i = 4;
    V ^% trhv = hv1;    // creates tracking reference to boxed type handle
    Console::WriteLine("Original V: {0}, Reference to handle of originally boxed V: {1}", v.i, trhv->i);
}
```

```
Original V: 2, Tracking reference to boxed V: 1
Tracking reference to boxed V: 3
Boxed new copy V: 1
Original V: 4, Reference to handle of originally boxed V: 1
```

範本函式採用原生、值或參考參數

使用追蹤參考的樣板函式的簽章中，您確定，可以類型為原生參數、CLR 值或 CLR 參考所呼叫函式。

```
// tracking_reference_template.cpp
// compile with: /clr
using namespace System;

class Temp {
public:
    // template functions
    template<typename T>
    static int f1(T% tt) {    // works for object in any location
        Console::WriteLine("T %");
        return 0;
    }

    template<typename T>
    static int f2(T& rt) {    // won't work for object on the gc heap
        Console::WriteLine("T &");
        return 1;
    }
};

// Class Definitions
ref struct R {
    int i;
};

int main() {
    R ^hr = gcnew R;
    int i = 1;

    Temp::f1(i); // ok
    Temp::f1(hr->i); // ok
    Temp::f2(i); // ok

    // error can't track object on gc heap with a native reference
    // Temp::f2(hr->i);
}
```

```
T %
T %
T &
```

另請參閱

[追蹤參考運算子](#)

HOW TO : 使用中的陣列C++/CLI

2019/12/2 • • [Edit Online](#)

這篇文章說明如何使用中的陣列C++/CLI。

一維陣列

下列範例示範如何建立一維陣列的參考、值和原生指標型別。此外，它也會示範如何從函式會傳回一維陣列，以及如何將單一維度陣列做為引數傳遞至函式。

```
// mcppv2_sdarrays.cpp
// compile with: /clr
using namespace System;

#define ARRAY_SIZE 2

value struct MyStruct {
    int m_i;
};

ref class MyClass {
public:
    int m_i;
};

struct MyNativeClass {
    int m_i;
};

// Returns a managed array of a reference type.
array<MyClass^>^ Test0() {
    int i;
    array< MyClass^ >^ local = gcnew array< MyClass^ >(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        local[i] = gcnew MyClass;
        local[i] -> m_i = i;
    }
    return local;
}

// Returns a managed array of Int32.
array<Int32>^ Test1() {
    int i;
    array< Int32 >^ local = gcnew array< Int32 >(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        local[i] = i + 10;
    return local;
}

// Modifies an array.
void Test2(array< MyNativeClass * >^ local) {
    for (int i = 0 ; i < ARRAY_SIZE ; i++)
        local[i] -> m_i = local[i] -> m_i + 2;
}

int main() {
    int i;

    // Declares an array of user-defined reference types
```

```

// and uses a function to initialize.
array< MyClass^ >^ MyClass0;
MyClass0 = Test0();

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i] -> m_i);
Console::WriteLine();

// Declares an array of value types and uses a function to initialize.
array< Int32 >^ IntArray;
IntArray = Test1();

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("IntArray[{0}] = {1}", i, IntArray[i]);
Console::WriteLine();

// Declares and initializes an array of user-defined
// reference types.
array< MyClass^ >^ MyClass1 = gcnew array< MyClass^ >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyClass1[i] = gcnew MyClass;
    MyClass1[i] -> m_i = i + 20;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass1[{0}] = {1}", i, MyClass1[i] -> m_i);
Console::WriteLine();

// Declares and initializes an array of pointers to a native type.
array< MyNativeClass * >^ MyClass2 = gcnew array<
    MyNativeClass * >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyClass2[i] = new MyNativeClass();
    MyClass2[i] -> m_i = i + 30;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass2[{0}] = {1}", i, MyClass2[i]->m_i);
Console::WriteLine();

Test2(MyClass2);
for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyClass2[{0}] = {1}", i, MyClass2[i]->m_i);
Console::WriteLine();

delete[] MyClass2[0];
delete[] MyClass2[1];

// Declares and initializes an array of user-defined value types.
array< MyStruct >^ MyStruct1 = gcnew array< MyStruct >(ARRAY_SIZE);
for (i = 0 ; i < ARRAY_SIZE ; i++) {
    MyStruct1[i] = MyStruct();
    MyStruct1[i].m_i = i + 40;
}

for (i = 0 ; i < ARRAY_SIZE ; i++)
    Console::WriteLine("MyStruct1[{0}] = {1}", i, MyStruct1[i].m_i);
}

```

```

MyClass0[0] = 0
MyClass0[1] = 1

IntArray[0] = 10
IntArray[1] = 11

 MyClass1[0] = 20
 MyClass1[1] = 21

 MyClass2[0] = 30
 MyClass2[1] = 31

 MyClass2[0] = 32
 MyClass2[1] = 33

 MyStruct1[0] = 40
 MyStruct1[1] = 41

```

下一個範例示範如何在一維 managed 陣列上執行彙總初始化。

```

// mcppv2_sdarrays_aggregate_init.cpp
// compile with: /clr
using namespace System;

ref class G {
public:
    G(int i) {}
};

value class V {
public:
    V(int i) {}
};

class N {
public:
    N(int i) {}
};

int main() {
    // Aggregate initialize a single-dimension managed array.
    array<String^>^ gc1 = gcnew array<String^>{"one", "two", "three"};
    array<String^>^ gc2 = {"one", "two", "three"};

    array<G^>^ gc3 = gcnew array<G^>{gcnew G(0), gcnew G(1), gcnew G(2)};
    array<G^>^ gc4 = {gcnew G(0), gcnew G(1), gcnew G(2)};

    array<Int32>^ value1 = gcnew array<Int32>{0, 1, 2};
    array<Int32>^ value2 = {0, 1, 2};

    array<V>^ value3 = gcnew array<V>{V(0), V(1), V(2)};
    array<V>^ value4 = {V(0), V(1), V(2)};

    array<N*>^ native1 = gcnew array<N*>{new N(0), new N(1), new N(2)};
    array<N*>^ native2 = {new N(0), new N(1), new N(2)};
}

```

```

 MyClass0[0, 0] = 0
 MyClass0[0, 1] = 0
 MyClass0[1, 0] = 1
 MyClass0[1, 1] = 1

 IntArray[0, 0] = 10
 IntArray[0, 1] = 10
 IntArray[1, 0] = 11
 IntArray[1, 1] = 11

```

此範例示範如何在多維度的 managed 陣列上執行彙總初始設定：

```

// mcppv2_mdarrays_aggregate_initialization.cpp
// compile with: /clr
using namespace System;

ref class G {
public:
    G(int i) {}
};

value class V {
public:
    V(int i) {}
};

class N {
public:
    N(int i) {}
};

int main() {
    // Aggregate initialize a multidimension managed array.
    array<String^, 2>^ gc1 = gcnew array<String^, 2>{ {"one", "two"},
        {"three", "four"} };
    array<String^, 2>^ gc2 = { {"one", "two"}, {"three", "four"} };

    array<G^, 2>^ gc3 = gcnew array<G^, 2>{ {gcnew G(0), gcnew G(1)},
        {gcnew G(2), gcnew G(3)} };
    array<G^, 2>^ gc4 = { {gcnew G(0), gcnew G(1)}, {gcnew G(2), gcnew G(3)} };

    array<Int32, 2>^ value1 = gcnew array<Int32, 2>{ {0, 1}, {2, 3} };
    array<Int32, 2>^ value2 = { {0, 1}, {2, 3} };

    array<V, 2>^ value3 = gcnew array<V, 2>{ {V(0), V(1)}, {V(2), V(3)} };
    array<V, 2>^ value4 = { {V(0), V(1)}, {V(2), V(3)} };

    array<N*, 2>^ native1 = gcnew array<N*, 2>{ {new N(0), new N(1)},
        {new N(2), new N(3)} };
    array<N*, 2>^ native2 = { {new N(0), new N(1)}, {new N(2), new N(3)} };
}

```

不規則陣列

本節說明如何建立 managed 陣列的參考、值和原生指標類型的一維陣列。此外，它也會示範如何從函式中傳回 managed 陣列的一維陣列，以及如何將單一維度陣列做為引數傳遞至函式。

```

// mcppv2_array_of_arrays.cpp
// compile with: /clr
using namespace System;

#define ARRAY_SIZE 2

```

```

value struct MyStruct {
    int m_i;
};

ref class MyClass {
public:
    int m_i;
};

// Returns an array of managed arrays of a reference type.
array<array<MyClass^>^>^ Test0() {
    int size_of_array = 4;
    array<array<MyClass^>^>^ local = gcnew
        array<array<MyClass^>^>(ARRAY_SIZE);

    for (int i = 0 ; i < ARRAY_SIZE ; i++, size_of_array += 4) {
        local[i] = gcnew array<MyClass^>(size_of_array);
        for (int k = 0; k < size_of_array ; k++) {
            local[i][k] = gcnew MyClass;
            local[i][k] -> m_i = i;
        }
    }

    return local;
}

// Returns a managed array of Int32.
array<array<Int32>^>^ Test1() {
    int i;
    array<array<Int32>^>^ local = gcnew array<array< Int32 >^>(ARRAY_SIZE);

    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        local[i] = gcnew array< Int32 >(ARRAY_SIZE);
        for ( int j = 0 ; j < ARRAY_SIZE ; j++ )
            local[i][j] = i + 10;
    }
    return local;
}

int main() {
    int i, j;

    // Declares an array of user-defined reference types
    // and uses a function to initialize.
    array< array< MyClass^ >>^ MyClass0;
    MyClass0 = Test0();

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        for ( j = 0 ; j < ARRAY_SIZE ; j++ )
            Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i][j] -> m_i);
    Console::WriteLine();

    // Declares an array of value types and uses a function to initialize.
    array< array< Int32 >>^ IntArray;
    IntArray = Test1();

    for (i = 0 ; i < ARRAY_SIZE ; i++)
        for ( j = 0 ; j < ARRAY_SIZE ; j++ )
            Console::WriteLine("IntArray[{0}] = {1}", i, IntArray[i][j]);
    Console::WriteLine();

    // Declares and initializes an array of user-defined value types.
    array< MyStruct >^ MyStruct1 = gcnew array< MyStruct >(ARRAY_SIZE);
    for (i = 0 ; i < ARRAY_SIZE ; i++) {
        MyStruct1[i] = MyStruct();
        MyStruct1[i].m_i = i + 40;
    }

    for (i = 0 : i < ARRAY_SIZE : i++)

```

```
    }
}
```

```
MyClass0[0] = 0
MyClass0[0] = 0
MyClass0[1] = 1
MyClass0[1] = 1

IntArray[0] = 10
IntArray[0] = 10
IntArray[1] = 11
IntArray[1] = 11

40
41
```

下列範例示範如何執行彙總初始化不規則陣列。

```
// mcppv2_array_of_arrays_aggregate_init.cpp
// compile with: /clr
using namespace System;
#define ARRAY_SIZE 2
int size_of_array = 4;
int count = 0;

ref class MyClass {
public:
    int m_i;
};

struct MyNativeClass {
    int m_i;
};

int main() {
    // Declares an array of user-defined reference types
    // and performs an aggregate initialization.
    array< array< MyClass^ >^ >^ MyClass0 = gcnew array<array<MyClass^>>^ {
        gcnew array<MyClass^>{ gcnew MyClass(), gcnew MyClass() },
        gcnew array<MyClass^>{ gcnew MyClass(), gcnew MyClass() }
    };

    for ( int i = 0 ; i < ARRAY_SIZE ; i++ , size_of_array += 4 )
        for ( int k = 0 ; k < ARRAY_SIZE ; k++ )
            MyClass0[i][k] -> m_i = i;

    for ( int i = 0 ; i < ARRAY_SIZE ; i++ )
        for ( int j = 0 ; j < ARRAY_SIZE ; j++ )
            Console::WriteLine("MyClass0[{0}] = {1}", i, MyClass0[i][j] -> m_i);
    Console::WriteLine();

    // Declares an array of value types and performs an aggregate initialization.
    array< array< Int32 >^ >^ IntArray = gcnew array<array< Int32 >>^ {
        gcnew array<Int32>{1,2},
        gcnew array<Int32>{3,4,5}
    };

    for each ( array<int>^ outer in IntArray ) {
        Console::Write("[");
        for each( int i in outer )
            Console::Write(" {0}", i);
        Console::Write(" ]");
    }
}
```

```

}

Console::WriteLine();

// Declares and initializes an array of pointers to a native type.
array<array< MyNativeClass * >^ > ^ MyClass2 =
    gcnew array<array< MyNativeClass * > ^ > {
        gcnew array<MyNativeClass *>{ new MyNativeClass(), new MyNativeClass() },
        gcnew array<MyNativeClass *>{ new MyNativeClass(), new MyNativeClass(), new MyNativeClass() }
    };

for each ( array<MyNativeClass * > ^ outer in MyClass2 )
    for each( MyNativeClass* i in outer )
        i->m_i = count++;

for each ( array<MyNativeClass * > ^ outer in MyClass2 ) {
    Console::Write("[");
    for each( MyNativeClass* i in outer )
        Console::Write(" {0}", i->m_i);
    Console::Write(" ]");
    Console::WriteLine();
}
Console::WriteLine();

// Declares and initializes an array of two-dimensional arrays of strings.
array<array<String ^,2> ^ > ^gc3 = gcnew array<array<String ^,2> ^ >{
    gcnew array<String ^>{ {"a","b"}, {"c", "d"}, {"e","f"} },
    gcnew array<String ^>{ {"g", "h"} }
};

for each ( array<String^, 2> ^ outer in gc3 ){
    Console::Write("[");
    for each( String ^ i in outer )
        Console::Write(" {0}", i);
    Console::Write(" ]");
    Console::WriteLine();
}
}
}

```

```

MyClass0[0] = 0
MyClass0[0] = 0
MyClass0[1] = 1
MyClass0[1] = 1

```

```

[ 1 2 ]
[ 3 4 5 ]

```

```

[ 0 1 ]
[ 2 3 4 ]

```

```

[ a b c d e f ]
[ g h ]

```

Managed 陣列當做樣板類型參數

此範例示範如何使用 managed 的陣列做為範本的參數：

```

// mcppv2_template_type_params.cpp
// compile with: /clr
using namespace System;
template <class T>
class TA {
public:
    array<array<T>^>^ f() {
        array<array<T>^>^ larr = gcnew array<array<T>^>(10);
        return larr;
    }
};

int main() {
    int retval = 0;
    TA<array<array<Int32>^>^* ta1 = new TA<array<array<Int32>^>^>();
    array<array<array<Int32>^>^>^ larr = ta1->f();
    retval += larr->Length - 10;
    Console::WriteLine("Return Code: {0}", retval);
}

```

Return Code: 0

managed 陣列的 typedef

此範例示範如何讓 managed 陣列的 typedef:

```

// mcppv2_typedef_arrays.cpp
// compile with: /clr
using namespace System;
ref class G {};

typedef array<array<G^>^> jagged_array;

int main() {
    jagged_array ^ MyArr = gcnew jagged_array (10);
}

```

排序陣列

不同於標準C++陣列，managed 陣列隱含衍生自其繼承的通用行為陣列基底類別。例如，`Sort`方法，可用來排序任何陣列中的項目。

對於包含基本的內建類型的陣列，您可以呼叫`Sort`方法。您可以覆寫的排序準則，這動作需要時您想要排序陣列的複雜型別。在此情況下，陣列項目類型必須實作`CompareTo`方法。

```

// array_sort.cpp
// compile with: /clr
using namespace System;

int main() {
    array<int>^ a = { 5, 4, 1, 3, 2 };
    Array::Sort( a );
    for (int i=0; i < a->Length; i++)
        Console::Write("{0} ", a[i] );
}

```

藉由使用自訂準則排序陣列

若要排序陣列，其中包含基本的內建類型，只要呼叫 `Array::Sort` 方法。不過，以排序陣列，包含複雜型別，或若要覆寫預設的排序準則，覆寫 `CompareTo` 方法。

在下列範例中，結構名為 `Element` 衍生自 `IComparable`，並提供寫入 `CompareTo` 兩個整數的平均值做為排序準則的方法。

```
using namespace System;

value struct Element : public IComparable {
    int v1, v2;

    virtual int CompareTo(Object^ obj) {
        Element^ o = dynamic_cast<Element^>(obj);
        if (o) {
            int thisAverage = (v1 + v2) / 2;
            int thatAverage = (o->v1 + o->v2) / 2;
            if (thisAverage < thatAverage)
                return -1;
            else if (thisAverage > thatAverage)
                return 1;
            return 0;
        }
        else
            throw gcnew ArgumentException
            ("Object must be of type 'Element'");
    }
};

int main() {
    array<Element>^ a = gcnew array<Element>(10);
    Random^ r = gcnew Random;

    for (int i=0; i < a->Length; i++) {
        a[i].v1 = r->Next() % 100;
        a[i].v2 = r->Next() % 100;
    }

    Array::Sort( a );
    for (int i=0; i < a->Length; i++) {
        int v1 = a[i].v1;
        int v2 = a[i].v2;
        int v = (v1 + v2) / 2;
        Console::WriteLine("{0} ({1}+{2})/2 ", v, v1, v2);
    }
}
```

陣列共變數

指定參考類別 D 具有直接或間接基底類別 B, D 型別的陣列可以指派給陣列變數的型別 b。

```
// clr_array_covariance.cpp
// compile with: /clr
using namespace System;

int main() {
    // String derives from Object.
    array<Object^>^ oa = gcnew array<String^>(20);
}
```

指派給陣列項目應該與指派相容具有動態類型的陣列。指派至具有不相容的類型的陣列項目會導致 `System::ArrayTypeMismatchException` 擲回。

陣列共變數不適用於實值類別類型的陣列。例如，無法轉換的 Int32 陣列物件 ^ 陣列，甚至不是藉由使用 boxing。

```
// clr_array_covariance2.cpp
// compile with: /clr
using namespace System;

ref struct Base { int i; };
ref struct Derived : Base {};
ref struct Derived2 : Base {};
ref struct Derived3 : Derived {};
ref struct Other { short s; };

int main() {
    // Derived* d[] = new Derived*[100];
    array<Derived^> ^ d = gcnew array<Derived^>(100);

    // ok by array covariance
    array<Base ^> ^ b = d;

    // invalid
    // b[0] = new Other;

    // error (runtime exception)
    // b[1] = gcnew Derived2;

    // error (runtime exception),
    // must be "at least" a Derived.
    // b[0] = gcnew Base;

    b[1] = gcnew Derived;
    b[0] = gcnew Derived3;
}
```

另請參閱

[陣列](#)

如何：定義和使用類別和結構 (c + +/CLI)

2020/11/2 • [Edit Online](#)

本文說明如何在 c + +/CLI 中定義和使用使用者定義的參考型別和實數值型別

物件具現化

參考 (ref) 類型只能在 managed 堆積上具現化，而不是在堆疊上或在原生堆積上具現化。實值型別可在堆疊或 managed 堆積上具現化。

```
// mcppv2_ref_class2.cpp
// compile with: /clr
ref class MyClass {
public:
    int i;

    // nested class
    ref class MyClass2 {
public:
    int i;
};

    // nested interface
    interface struct MyInterface {
        void f();
    };
};

ref class MyClass2 : public MyClass::MyInterface {
public:
    virtual void f() {
        System::Console::WriteLine("test");
    }
};

public value struct MyStruct {
    void f() {
        System::Console::WriteLine("test");
    }
};

int main() {
    // instantiate ref type on garbage-collected heap
    MyClass ^ p_MyClass = gcnew MyClass;
    p_MyClass -> i = 4;

    // instantiate value type on garbage-collected heap
    MyStruct ^ p_MyStruct = gcnew MyStruct;
    p_MyStruct -> f();

    // instantiate value type on the stack
    MyStruct p_MyStruct2;
    p_MyStruct2.f();

    // instantiate nested ref type on garbage-collected heap
    MyClass::MyClass2 ^ p_MyClass2 = gcnew MyClass::MyClass2;
    p_MyClass2 -> i = 5;
}
```

隱含抽象類別

隱含抽象類別無法具現化。當下列情況時，類別會隱含抽象化：

- 類別的基底類型是介面，而
- 類別不會執行所有介面的成員函式。

您可能無法從衍生自介面的類別來建立物件。原因可能是類別是隱含抽象的。如需抽象類別的詳細資訊，請參閱[抽象](#)。

下列程式碼範例示範無法具現化類別，因為函式並未實作為函式 `MyClass` `MyClass::func2`。若要讓範例進行編譯，請取消批註 `MyClass::func2`。

```
// mcppv2_ref_class5.cpp
// compile with: /clr
interface struct MyInterface {
    void func1();
    void func2();
};

ref class MyClass : public MyInterface {
public:
    void func1(){}
    // void func2(){}
};

int main() {
    MyClass ^ h_MyClass = gcnew MyClass;    // C2259
                                            // To resolve, uncomment MyClass::func2.
}
```

類型可見度

您可以控制 common language runtime (CLR) 類型的可見度。參考您的元件時，您可以控制元件中的類型是可見的，還是在元件外部看不到。

`public` 指出包含型別之元件的指示詞的任何原始程式檔都可以看見型別 `#using`。`private` 指出包含型別之元件的指示詞的原始程式檔看不到型別 `#using`。不過，私用類型會顯示在相同的元件中。根據預設，類別的可見度為 `private`。

根據預設，在 Visual Studio 2005 之前，原生類型在元件外部具有公用存取範圍。啟用[編譯器警告 \(層級 1\) C4692](#)，以協助您瞭解私用原生類型的使用方式不正確。使用 `make_public` pragma，將公用存取範圍提供給無法修改之原始程式碼檔中的原生類型。

如需詳細資訊，請參閱[#using 指示詞](#)。

下列範例示範如何宣告型別並指定其存取範圍，然後在元件記憶體取這些類型。如果使用參考具有私用類型的元件，則 `#using` 只會顯示元件中的公用類型。

```

// type_visibility.cpp
// compile with: /clr
using namespace System;
// public type, visible inside and outside assembly
public ref struct Public_Class {
    void Test(){Console::WriteLine("in Public_Class");}
};

// private type, visible inside but not outside assembly
private ref struct Private_Class {
    void Test(){Console::WriteLine("in Private_Class");}
};

// default accessibility is private
ref class Private_Class_2 {
public:
    void Test(){Console::WriteLine("in Private_Class_2");}
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    a->Test();

    Private_Class ^ b = gcnew Private_Class;
    b->Test();

    Private_Class_2 ^ c = gcnew Private_Class_2;
    c->Test();
}

```

輸出

```

in Public_Class
in Private_Class
in Private_Class_2

```

現在，讓我們重寫先前的範例，讓它以 DLL 的形式建立。

```

// type_visibility_2.cpp
// compile with: /clr /LD
using namespace System;
// public type, visible inside and outside the assembly
public ref struct Public_Class {
    void Test(){Console::WriteLine("in Public_Class");}
};

// private type, visible inside but not outside the assembly
private ref struct Private_Class {
    void Test(){Console::WriteLine("in Private_Class");}
};

// by default, accessibility is private
ref class Private_Class_2 {
public:
    void Test(){Console::WriteLine("in Private_Class_2");}
};

```

下一個範例顯示如何存取元件之外的型別。在此範例中，用戶端會使用先前範例中所建立的元件。

```
// type_visibility_3.cpp
// compile with: /clr
#using "type_visibility_2.dll"
int main() {
    Public_Class ^ a = gcnew Public_Class;
    a->Test();

    // private types not accessible outside the assembly
    // Private_Class ^ b = gcnew Private_Class;
    // Private_Class_2 ^ c = gcnew Private_Class_2;
}
```

輸出

```
in Public_Class
```

成員可見度

您可以使用存取規範的配對，從元件外部存取公用類別的成員，而不是從元件外部存取該成員。[public](#) [protected](#)

**** [private](#)

下表摘要說明各種存取規範的效果：

public	private
public	成員可在元件內部和外部進行存取。如需詳細資訊，請參閱 public 。
private	成員在元件內部和外部無法存取。如需詳細資訊，請參閱 private 。
protected	成員可在元件內部和外部存取，但僅適用於衍生類型。如需詳細資訊，請參閱 protected 。
internal	成員在元件內是公用的，但在元件外部是私用的。 internal 是即時線上關鍵字。如需詳細資訊，請參閱 內容相關性關鍵字 。
public protected -或- protected public	成員在元件內是公用的，但在元件外部受到保護。
private protected -或- protected private	成員在元件內部受到保護，但在元件外部是私用的。

下列範例顯示的公用類型具有使用不同存取規範宣告的成員。然後，它會顯示從元件內部對這些成員的存取。

```

// compile with: /clr
using namespace System;
// public type, visible inside and outside the assembly
public ref class Public_Class {
public:
    void Public_Function(){System::Console::WriteLine("in Public_Function");}
private:
    void Private_Function(){System::Console::WriteLine("in Private_Function");}
protected:
    void Protected_Function(){System::Console::WriteLine("in Protected_Function");}
internal:
    void Internal_Function(){System::Console::WriteLine("in Internal_Function");}
protected public:
    void Protected_Public_Function(){System::Console::WriteLine("in Protected_Public_Function");}
public protected:
    void Public_Protected_Function(){System::Console::WriteLine("in Public_Protected_Function");}
private protected:
    void Private_Protected_Function(){System::Console::WriteLine("in Private_Protected_Function");}
protected private:
    void Protected_Private_Function(){System::Console::WriteLine("in Protected_Private_Function");}
};

// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
void Test() {
    Console::WriteLine("=====");
    Console::WriteLine("in function of derived class");
    Protected_Function();
    Protected_Private_Function();
    Private_Protected_Function();
    Console::WriteLine("exiting function of derived class");
    Console::WriteLine("=====");
}
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    MyClass ^ b = gcnew MyClass;
    a->Public_Function();
    a->Protected_Public_Function();
    a->Public_Protected_Function();

    // accessible inside but not outside the assembly
    a->Internal_Function();

    // call protected functions
    b->Test();

    // not accessible inside or outside the assembly
    // a->Private_Function();
}

```

輸出

```
in Public_Function
in Protected_Public_Function
in Public_Protected_Function
in Internal_Function
=====
in function of derived class
in Protected_Function
in Protected_Private_Function
in Private_Protected_Function
existing function of derived class
=====
```

現在讓我們以 DLL 的形式建立先前的範例。

```
// compile with: /clr /LD
using namespace System;
// public type, visible inside and outside the assembly
public ref class Public_Class {
public:
    void Public_Function(){System::Console::WriteLine("in Public_Function");}

private:
    void Private_Function(){System::Console::WriteLine("in Private_Function");}

protected:
    void Protected_Function(){System::Console::WriteLine("in Protected_Function");}

internal:
    void Internal_Function(){System::Console::WriteLine("in Internal_Function");}

protected public:
    void Protected_Public_Function(){System::Console::WriteLine("in Protected_Public_Function");}

public protected:
    void Public_Protected_Function(){System::Console::WriteLine("in Public_Protected_Function");}

private protected:
    void Private_Protected_Function(){System::Console::WriteLine("in Private_Protected_Function");}

protected private:
    void Protected_Private_Function(){System::Console::WriteLine("in Protected_Private_Function");}
};

// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
    void Test() {
        Console::WriteLine("=====");
        Console::WriteLine("in function of derived class");
        Protected_Function();
        Protected_Private_Function();
        Private_Protected_Function();
        Console::WriteLine("existing function of derived class");
        Console::WriteLine("=====");
    }
};
```

下列範例會使用在上一個範例中建立的元件。它會顯示如何從元件外部存取成員。

```

// compile with: /clr
#using "type_member_visibility_2.dll"
using namespace System;
// a derived type, calls protected functions
ref struct MyClass : public Public_Class {
    void Test() {
        Console::WriteLine("=====");
        Console::WriteLine("in function of derived class");
        Protected_Function();
        Protected_Public_Function();
        Public_Protected_Function();
        Console::WriteLine("exiting function of derived class");
        Console::WriteLine("=====");
    }
};

int main() {
    Public_Class ^ a = gcnew Public_Class;
    MyClass ^ b = gcnew MyClass;
    a->Public_Function();

    // call protected functions
    b->Test();

    // can't be called outside the assembly
    // a->Private_Function();
    // a->Internal_Function();
    // a->Protected_Private_Function();
    // a->Private_Protected_Function();
}

```

輸出

```

in Public_Function
=====
in function of derived class
in Protected_Function
in Protected_Public_Function
in Public_Protected_Function
exiting function of derived class
=====

```

公用和私用原生類別

您可以從 managed 類型參考原生類型。例如，managed 型別中的函式可以採用其型別為原生結構的參數。如果 managed 型別和函式在元件中是公用的，則原生型別也必須是公用的。

```

// native type
public struct N {
    N(){}
    int i;
};

```

接下來，建立使用原生類型的原始程式碼檔：

```
// compile with: /clr /LD
#include "mcppv2_ref_class3.h"
// public managed type
public ref struct R {
    // public function that takes a native type
    void f(N nn) {}
};


```

現在，編譯用戶端：

```
// compile with: /clr
#using "mcppv2_ref_class3.dll"

#include "mcppv2_ref_class3.h"

int main() {
    R ^r = gcnew R;
    N n;
    r->f(n);
}
```

靜態函式

CLR 類型(例如類別或結構)可以有可用於初始化靜態資料成員的靜態函式。靜態的函式最多隻會呼叫一次，而且會在第一次存取型別的任何靜態成員之前呼叫。

實例的函式一律會在靜態函式之後執行。

如果類別具有靜態的函式，則編譯器無法內嵌對函式的呼叫。如果類別是實值型別、具有靜態的函式，而且沒有實例的函式，則編譯器無法內嵌對任何成員函式的呼叫。CLR 可以內嵌呼叫，但編譯器無法。

將靜態的函式定義為私用成員函式，因為它的目的只是要由 CLR 呼叫。

如需靜態函式的詳細資訊，請參閱 [如何：定義介面靜態函式 \(c + +/cli\)](#)。

```
// compile with: /clr
using namespace System;

ref class MyClass {
private:
    static int i = 0;

    static MyClass() {
        Console::WriteLine("in static constructor");
        i = 9;
    }

public:
    static void Test() {
        i++;
        Console::WriteLine(i);
    }
};

int main() {
    MyClass::Test();
    MyClass::Test();
}
```

輸出

```
in static constructor  
10  
11
```

指標的語義 `this`

當您使用 c++\CLI 來定義型別時，`this` 參考型別中的指標是型別 **控制碼**。實 `this` 值型別中的指標是 **內部指標類型**。

`this` 當呼叫預設索引子時，指標的這些不同語義可能會導致非預期的行為。下一個範例會顯示在 `ref` 型別和實值型別中存取預設索引子的正確方式。

如需詳細資訊，請參閱 [物件運算子控制碼 \(^\)](#) 和 [Interior_ptr \(c++/cli\)](#)

```
// compile with: /clr  
using namespace System;  
  
ref struct A {  
    property Double default[Double] {  
        Double get(Double data) {  
            return data*data;  
        }  
    }  
  
    A() {  
        // accessing default indexer  
        Console::WriteLine("{0}", this[3.3]);  
    }  
};  
  
value struct B {  
    property Double default[Double] {  
        Double get(Double data) {  
            return data*data;  
        }  
    }  
    void Test() {  
        // accessing default indexer  
        Console::WriteLine("{0}", this->default[3.3]);  
    }  
};  
  
int main() {  
    A ^ mya = gcnew A();  
    B ^ myb = gcnew B();  
    myb->Test();  
}
```

輸出

```
10.89  
10.89
```

依簽章隱藏函式

在標準 c++ 中，衍生類別中的函式會隱藏基類中的函式，即使衍生類別函數沒有相同類型或數目的參數也是一樣。這就是所謂的 **依名稱隱藏語義**。在參考型別中，如果名稱和參數清單相同，則基類中的函式只會被衍生類別中的函式所隱藏。這就是所謂的 **隱藏簽章語義**。

當類別的所有函式在中繼資料中標記為時，就會將類別視為隱藏簽章類別 `hidebysig`。根據預設，在下建立的所有類別 `/clr` 都具有函式 `hidebysig`。當某個類別具有函式時 `hidebysig`，編譯器不會在任何直接基類中依名稱隱藏函式，但是如果編譯器在繼承鏈中遇到依名稱隱藏的類別，則會繼續進行依名稱的隱藏行為。

在隱藏簽章的語法下，當在物件上呼叫函式時，編譯器會識別最常衍生的類別，其中包含可滿足函式呼叫的函式。如果類別中只有一個可滿足呼叫的函式，則編譯器會呼叫該函式。如果類別中有一個以上可滿足呼叫的函式，編譯器會使用多載解析規則來判斷要呼叫的函式。如需多載規則的詳細資訊，[請參閱函式多載](#)。

針對指定的函式呼叫，基類中的函式可能會有簽章，使其比衍生類別中的函式更好。但是，如果函式是在衍生類別的物件上明確呼叫，則會呼叫衍生類別中的函式。

由於傳回值不會被視為函式簽章的一部分，因此，如果基類函式具有相同的名稱，並採用與衍生類別函式相同的種類和數目的引數，即使它不同于傳回值的類型，也會被隱藏。

下列範例顯示基類中的函式不會隱藏基類中的函式。

```
// compile with: /clr
using namespace System;
ref struct Base {
    void Test() {
        Console::WriteLine("Base::Test");
    }
};

ref struct Derived : public Base {
    void Test(int i) {
        Console::WriteLine("Derived::Test");
    }
};

int main() {
    Derived ^ t = gcnew Derived;
    // Test() in the base class will not be hidden
    t->Test();
}
```

輸出

```
Base::Test
```

下一個範例顯示 Microsoft C++ 編譯器會呼叫最常衍生類別中的函式，即使需要轉換來比對一或多個參數，也不會在基類中呼叫函式，而該函式呼叫會比較符合函式呼叫。

```

// compile with: /clr
using namespace System;
ref struct Base {
    void Test2(Single d) {
        Console::WriteLine("Base::Test2");
    }
};

ref struct Derived : public Base {
    void Test2(Double f) {
        Console::WriteLine("Derived::Test2");
    }
};

int main() {
    Derived ^ t = gcnew Derived;
    // Base::Test2 is a better match, but the compiler
    // calls a function in the derived class if possible
    t->Test2(3.14f);
}

```

輸出

```
Derived::Test2
```

下列範例顯示即使基類的簽章與衍生類別相同，也可以隱藏函式。

```

// compile with: /clr
using namespace System;
ref struct Base {
    int Test4() {
        Console::WriteLine("Base::Test4");
        return 9;
    }
};

ref struct Derived : public Base {
    char Test4() {
        Console::WriteLine("Derived::Test4");
        return 'a';
    }
};

int main() {
    Derived ^ t = gcnew Derived;

    // Base::Test4 is hidden
    int i = t->Test4();
    Console::WriteLine(i);
}

```

輸出

```
Derived::Test4
97
```

複製函式

C++ 標準指出當移動物件時，會呼叫複製的函式，以便在相同的位址建立和終結物件。

不過，當編譯為 MSIL 的函式呼叫原生函式，其中的原生類別（或多個）以傳值方式傳遞，且原生類別具有複製函式或析構函式時，不會呼叫複製的函式，而且物件會在與建立時不同的位址終結。如果類別具有指向自身的指標，或如果程式碼是依位址追蹤物件，此行為可能會造成問題。

如需詳細資訊，請參閱 [/clr \(Common Language Runtime 編譯\)](#)。

下列範例會示範何時不產生複製的函式。

```
// compile with: /clr
#include<stdio.h>

struct S {
    int i;
    static int n;

    S() : i(n++) {
        printf_s("S object %d being constructed, this=%p\n", i, this);
    }

    S(S const& rhs) : i(n++) {
        printf_s("S object %d being copy constructed from S object "
                 "%d, this=%p\n", i, rhs.i, this);
    }

    ~S() {
        printf_s("S object %d being destroyed, this=%p\n", i, this);
    }
};

int S::n = 0;

#pragma managed(push,off)
void f(S s1, S s2) {
    printf_s("in function f\n");
}
#pragma managed(pop)

int main() {
    S s;
    S t;
    f(s,t);
}
```

輸出

```
S object 0 being constructed, this=0018F378
S object 1 being constructed, this=0018F37C
S object 2 being copy constructed from S object 1, this=0018F380
S object 3 being copy constructed from S object 0, this=0018F384
S object 4 being copy constructed from S object 2, this=0018F2E4
S object 2 being destroyed, this=0018F380
S object 5 being copy constructed from S object 3, this=0018F2E0
S object 3 being destroyed, this=0018F384
in function f
S object 5 being destroyed, this=0018F2E0
S object 4 being destroyed, this=0018F2E4
S object 1 being destroyed, this=0018F37C
S object 0 being destroyed, this=0018F378
```

析構函數和完成項

參考型別中的析構程式會對資源進行決定性的清除。完成項清除未受管理的資源，並可由函式以決定性的方式呼

叫，或由垃圾收集行程非確定性地。如需標準 C++ 中之析構函數的詳細資訊，請參閱「[析構](#)」。

```
class classname {
    ~classname() {} // destructor
    !classname() {} // finalizer
};
```

CLR 垃圾收集行程會刪除未使用的 managed 物件，並在不再需要時釋放其記憶體。不過，型別可能會使用垃圾收集行程不知道如何釋放的資源。這些資源稱為 [非受控資源](#)（原生檔案控制代碼，例如）。建議您釋放完成項中的所有非受控資源。垃圾收集行程會釋出受控資源非確定性地，因此在完成項中參考 managed 資源是不安全的。這是因為垃圾收集行程可能已經清除它們。

Visual C++ 完成項與 [Finalize](#) 方法不同。（CLR 檔會使用完成項和 [Finalize](#) 方法同義）。此 [Finalize](#) 方法是由垃圾收集行程所呼叫，它會叫用類別繼承鏈中的每個完成項。不同于 Visual C++ 的析構函數，衍生類別的完成項呼叫不會導致編譯器在所有基類中叫用完成項。

因為 Microsoft C++ 編譯器支援確定性的資源釋放，所以請勿嘗試實作為 [Dispose](#) 或 [Finalize](#) 方法。但是，如果您熟悉這些方法，以下是 Visual C++ 完成項和呼叫完成項的函式對應至模式的方式 [Dispose](#)：

```
// Visual C++ code
ref class T {
    ~T() { this->!T(); } // destructor calls finalizer
    !T() {} // finalizer
};

// equivalent to the Dispose pattern
void Dispose(bool disposing) {
    if (disposing) {
        ~T();
    } else {
        !T();
    }
}
```

Managed 類型也可以使用您想要以決定性的方式發行的 managed 資源。在不再需要物件之後，您可能不想讓垃圾收集行程釋放物件非確定性地。確定性的資源版本可以大幅提升效能。

Microsoft C++ 編譯器可讓函式的定義以決定性的方式清除物件。使用「函式」來釋出您想要以決定性方式發行的所有資源。如果有完成項，請從函式呼叫它，以避免程式碼重複。

```
// compile with: /clr /c
ref struct A {
    // destructor cleans up all resources
    ~A() {
        // clean up code to release managed resource
        // ...
        // to avoid code duplication,
        // call finalizer to release unmanaged resources
        this->!A();
    }

    // finalizer cleans up unmanaged resources
    // destructor or garbage collector will
    // clean up managed resources
    !A() {
        // clean up code to release unmanaged resources
        // ...
    }
};
```

如果使用您的型別的程式碼不會呼叫該函式，垃圾收集行程最後會釋放所有 managed 資源。

函式的存在不表示完成項的存在。不過，完成項表示您必須定義一個自訂函式，並從該函式呼叫完成項。此呼叫提供非受控資源的決定性版本。

呼叫的函式會使用完成物件的結束，來抑制 [SuppressFinalize](#)。如果未呼叫此函式，則最後會由垃圾收集行程呼叫您的型別完成項。

您可以藉由呼叫「函式」來明確清除物件的資源，而不是讓 CLR 非確定性地完成物件，藉以改善效能。

如果有下列情況，則以 Visual C++ 撰寫並使用編譯的程式碼會 `/clr` 執行類型的函式：

- 使用堆疊語義建立的物件會超出範圍。如需詳細資訊，請參閱參考型別的 [C++ 堆疊語義](#)。
- 物件的結構中會擲回例外狀況。
- 物件是正在執行其執行程式之物件中的成員。
- 您可以在控制碼上呼叫 `delete` 運算子，(控制碼運算子的控制碼 (^))。
- 您明確地呼叫該函式。

如果以其他語言撰寫的用戶端使用您的型別，則會呼叫此函式，如下所示：

- 呼叫時 `Dispose`。
- 在類型的呼叫上 `Dispose(void)`。
- 如果類型超出 C# 語句中的範圍 `using`。

如果您未對參考型別使用堆疊語義，並且在 managed 堆積上建立參考型別的物件，請使用 [try-finally](#) 語法，以確保例外狀況不會讓此函式無法執行。

```
// compile with: /clr
ref struct A {
    ~A() {}
};

int main() {
    A ^ MyA = gcnew A;
    try {
        // use MyA
    }
    finally {
        delete MyA;
    }
}
```

如果您的類型有一個函式，編譯器會產生可執行檔 `Dispose` 方法 [IDisposable](#)。如果以 Visual C++ 撰寫的型別，而且具有從另一種語言取用的函式，則 `IDisposable::Dispose` 在該型別上呼叫會導致呼叫類型的函式。從 Visual C++ 用戶端取用型別時，您不能直接呼叫，`Dispose` 而是使用運算子呼叫此函式 `delete`。

如果您的型別具有完成項，則編譯器 `Finalize(void)` 會產生可覆寫的方法 [Finalize](#)。

如果型別具有完成項或「函式」，則編譯器會 `Dispose(bool)` 根據設計模式產生方法。(需詳細資訊，請參閱 [處置模式](#))。您無法在 Visual C++ 中明確撰寫或呼叫 `Dispose(bool)`。

如果類型具有符合設計模式的基類，則會在呼叫衍生類別的函式時呼叫所有基類的析構函式。(如果您的類型是以 Visual C++ 撰寫，則編譯器會確保您的型別會實作為此模式。) 換句話說，參考類別的函式會根據 C++ 標準所指定的基底和成員來連結。首先，會執行類別的函式。然後，其成員的析構函數會以它們的建立順序相反的循序執行。最後，其基類的析構函數會以它們的建立順序相反的循序執行。

數值型別或介面中不允許有析構函數和完成項。

只能在參考型別中定義或宣告完成項。和函式和函式一樣，完成項沒有傳回型別。

在物件的完成項執行之後，也會呼叫任何基類中的完成項，從最不衍生的類型開始。資料成員的完成項不會由類別的完成項自動連結到。

如果完成項刪除 managed 類型中的原生指標，您必須確定不會提早收集或透過原生指標的參考。在 managed 類型上呼叫此函式，而不是使用 [KeepAlive](#)。

您可以在編譯時期偵測型別是否具有完成項或函式。如需詳細資訊，請參閱[類型特徵的編譯器支援](#)。

下一個範例會示範兩種類型：一個具有未受管理的資源，另一個則具有以決定性方式釋放的 managed 資源。

```

// compile with: /clr
#include <vcclr.h>
#include <stdio.h>
using namespace System;
using namespace System::IO;

ref class SystemFileWriter {
    FileStream ^ file;
    array<Byte> ^ arr;
    int bufLen;

public:
    SystemFileWriter(String ^ name) : file(File::Open(name, FileMode::Append)),
                                         arr(gcnew array<Byte>(1024)) {}

    void Flush() {
        file->Write(arr, 0, bufLen);
        bufLen = 0;
    }

    ~SystemFileWriter() {
        Flush();
        delete file;
    }
};

ref class CRTFileWriter {
    FILE * file;
    array<Byte> ^ arr;
    int bufLen;

    static FILE * getFile(String ^ n) {
        pin_ptr<const wchar_t> name = PtrToStringChars(n);
        FILE * ret = 0;
        _wfopen_s(&ret, name, L"ab");
        return ret;
    }

public:
    CRTFileWriter(String ^ name) : file(getFile(name)), arr(gcnew array<Byte>(1024)) {}

    void Flush() {
        pin_ptr<Byte> buf = &arr[0];
        fwrite(buf, 1, bufLen, file);
        bufLen = 0;
    }

    ~CRTFileWriter() {
        this->!CRTFileWriter();
    }

    !CRTFileWriter() {
        Flush();
        fclose(file);
    }
};

int main() {
    SystemFileWriter w("systest.txt");
    CRTFileWriter ^ w2 = gcnew CRTFileWriter("crttest.txt");
}

```

另請參閱

[類別和結構](#)

參考類型的 C++ 堆疊語意

2020/11/2 • [Edit Online](#)

在 Visual Studio 2005 之前，只能使用運算子來建立參考型別的實例 `new`，而這會在垃圾收集堆積上建立物件。不過，您現在可以使用與您用來在堆疊上建立原生類型之執行個體相同的語法建立參考類型的執行個體。因此，您不需要使用`ref new`、`gcnew`來建立參考型別的物件。然後，當物件超出範圍時，編譯器會呼叫物件的解構函式。

備註

當您使用堆疊語義建立參考型別的實例時，編譯器會在內部于垃圾收集堆積上建立實例（使用 `gcnew`）。

當函式的簽章或傳回型別是一個傳值參考類型的執行個體時，函式在中繼資料中會被標記為需要進行特殊處理（使用 `modreq`）。這項特殊處理目前只提供給 Visual C++ 用戶端使用，其他語言目前不支援使用以堆疊語意建立之參考類型的函式或資料。

`gcnew` 如果類型沒有任何析構函式，則使用（動態配置）而非堆疊語義的其中一個原因是。此外，如果想要讓 Visual C++ 以外的語言使用函式，則無法在函式簽章中使用以堆疊語意建立的參考類型。

編譯器不會為參考類型產生複製建構函式。因此，如果您在簽章中定義了一個使用傳值參考類型的函式，則必須定義參考類型的複製建構函式。參考類型的複製建構函式具有下列形式的簽章：`R(R%){}.`

編譯器不會為參考類型產生預設指派運算子。指派運算子可讓您建立使用堆疊語意的物件，並以使用堆疊語意建立的現有物件對其進行初始化。參考類型的指派運算子具有下列形式的簽章：`void operator=(R%){}`。

如果您的型別的「析構函式」釋放重要資源，而且您對引用型別使用堆疊語義，則不需要明確地呼叫「析構函式」（或呼叫 `delete`）。如需參考型別中之析構函數的詳細資訊，請參閱[如何：定義和使用類別和結構\(c + +/cli\)中的析構函數和完成項](#)。

由編譯器產生的指派運算子會遵循一般的標準 C++ 規則，並提供下列新增功能：

- 類型為參考類型之控制代碼的任何非靜態資料成員將會進行淺層複製（視為指標類型的非靜態資料成員）。
- 類型為實值類型的任何非靜態資料成員都會進行淺層複製。
- 類型為參考類型之執行個體的任何非靜態資料成員都會叫用參考類型之複製建構函式的呼叫。

編譯器也提供 `%` 一元運算子，可將使用堆疊語意建立之參考類型的執行個體轉換為其基礎控制代碼類型。

下列參考類型不能與堆疊語意搭配使用：

- [delegate \(C++ 元件擴充功能\)](#)
- [陣列](#)
- [String](#)

範例

描述

下列程式碼範例說明如何使用堆疊語意宣告參考類型的執行個體、指派運算子和複製建構函式的運作方式，以及如何初始化使用堆疊語意建立之參考類型的追蹤參考。

程式碼

```

// stack_semantics_for_reference_types.cpp
// compile with: /clr
ref class R {
public:
    int i;
    R(){}
    // assignment operator
    void operator=(R% r) {
        i = r.i;
    }
    // copy constructor
    R(R% r) : i(r.i) {}
};

void Test(R r) {} // requires copy constructor

int main() {
    R r1;
    r1.i = 98;

    R r2(r1); // requires copy constructor
    System::Console::WriteLine(r1.i);
    System::Console::WriteLine(r2.i);

    // use % unary operator to convert instance using stack semantics
    // to its underlying handle
    R ^ r3 = %r1;
    System::Console::WriteLine(r3->i);

    Test(r1);

    R r4;
    R r5;
    r5.i = 13;
    r4 = r5; // requires a user-defined assignment operator
    System::Console::WriteLine(r4.i);

    // initialize tracking reference
    R % r6 = r4;
    System::Console::WriteLine(r6.i);
}

```

輸出

```

98
98
98
13
13

```

另請參閱

[類別和結構](#)

使用者定義的運算子 (C++/CLI)

2020/11/2 • [Edit Online](#)

Managed 類型的使用者定義運算子允許為靜態成員或是執行個體成員，或位於全域範圍。不過，不是以 Visual C++ 語言撰寫的程式中，只有靜態運算子可以透過用戶端的中繼資料可取得。

在參考類型中，其中一個靜態使用者定義運算子的參數必須是下列其中一個：

- 封入類型的執行個體之控制代碼 (`type ^`)。
- 參考型別間接 (`type ^&` 或 `類型 ^%`) 至封入型別的實例控制碼。

在實值類型中，其中一個靜態使用者定義運算子的參數必須是下列其中一個：

- 與封入實值類型相同的類型。
- 對封入類型的指標類型進行間接取值 (`type ^`)。
- 參考型別間接 (`type %` 或 `type &`) 為封入類型。
- 參考型別間接取值 (`type ^%` 或 `type ^&`) 至控制碼。

您可以定義下列運算子：

運算子	參照類型？
!	一元
!=	Binary
%	Binary
&	一元和二元
&&	Binary
*	一元和二元
+	一元和二元
++	一元
,	Binary
-	一元和二元
--	一元
->	一元
/	Binary

<code>TokenType</code>	<code>LexicalCategory</code>
<	Binary
<<	Binary
<=	Binary
=	Binary
==	Binary
>	Binary
>=	Binary
>>	Binary
^	Binary
false	一元
true	一元
	Binary
	Binary
~	一元

範例：使用者定義的運算子

```

// mcppv2_user-defined_operators.cpp
// compile with: /clr
using namespace System;
public ref struct X {
    X(int i) : m_i(i) {}
    X() {}

    int m_i;

    // static, binary, user-defined operator
    static X ^ operator +(X^ me, int i) {
        return (gcnew X(me -> m_i + i));
    }

    // instance, binary, user-defined operator
    X^ operator -( int i ) {
        return gcnew X(this->m_i - i);
    }

    // instance, unary, user-defined pre-increment operator
    X^ operator ++() {
        return gcnew X(this->m_i++);
    }

    // instance, unary, user-defined post-increment operator
    X^ operator ++(int i) {
        return gcnew X(this->m_i++);
    }

    // static, unary user-defined pre- and post-increment operator
    static X^ operator-- (X^ me) {
        return (gcnew X(me -> m_i - 1));
    }
};

int main() {
    X ^hX = gcnew X(-5);
    System::Console::WriteLine(hX -> m_i);

    hX = hX + 1;
    System::Console::WriteLine(hX -> m_i);

    hX = hX - (-1);
    System::Console::WriteLine(hX -> m_i);

    ++hX;
    System::Console::WriteLine(hX -> m_i);

    hX++;
    System::Console::WriteLine(hX -> m_i);

    hX--;
    System::Console::WriteLine(hX -> m_i);

    --hX;
    System::Console::WriteLine(hX -> m_i);
}

```

```

-5
-4
-3
-2
-1
-2
-3

```

範例：運算子合成

下列範例示範操作員合成，只有當您使用 /clr 進行編譯時才可使用。如果沒有定義任何一個二元運算子的工作表單，運算子合成會建立它，其中指派運算子左方會使用 CLR 類型。

```
// mcppv2_user-defined_operators_2.cpp
// compile with: /clr
ref struct A {
    A(int n) : m_n(n) {};
    static A^ operator + (A^ r1, A^ r2) {
        return gcnew A( r1->m_n + r2->m_n);
    };
    int m_n;
};

int main() {
    A^ a1 = gcnew A(10);
    A^ a2 = gcnew A(20);

    a1 += a2;    // a1 = a1 + a2    += not defined in source
    System::Console::WriteLine(a1->m_n);
}
```

30

另請參閱

[類別和結構](#)

使用者定義轉換 (C++/CLI)

2019/12/10 • [Edit Online](#)

當轉換中的其中一個類型是實數值型別或參考型別的參考或實例時，本節將討論使用者定義的轉換(UDC)。

隱含和明確轉換

使用者定義的轉換可以是隱含或明確的。如果轉換不會導致資訊遺失，則 UDC 應該是隱含的。否則，應該定義明確的 UDC。

原生類別的函式可以用來將參考或實值型別轉換成原生類別。

如需轉換的詳細資訊，請參閱[裝箱和標準轉換](#)。

```
// mcpp_User Defined_Conversions.cpp
// compile with: /clr
#include "stdio.h"
ref class R;
class N;

value class V {
    static operator V(R^) {
        return V();
    }
};

ref class R {
public:
    static operator N(R^);
    static operator V(R^) {
        System::Console::WriteLine("in R::operator N");
        return V();
    }
};

class N {
public:
    N(R^) {
        printf("in N::N\n");
    }
};

R::operator N(R^) {
    System::Console::WriteLine("in R::operator N");
    return N(nullptr);
}

int main() {
    // Direct initialization:
    R ^r2;
    N n2(r2);    // direct initialization, calls constructor
    static_cast<N>(r2);    // also direct initialization

    R ^r3;
    // ambiguous V::operator V(R^) and R::operator V(R^)
    // static_cast<V>(r3);
}
```

Output

```
in N::N  
in N::N
```

轉換來源運算子

Convert-from 運算子會建立類別的物件，其中運算子是從其他類別的物件所定義。

標準C++不支援 convert from 運算子；標準C++針對此用途使用了函數。不過，使用CLR型別時，C++視覺效果會提供呼叫 convert from 運算子的語法支援。

若要與其他符合CLS標準的語言互通，您可能想要使用對應的convert from 運算子，來包裝指定類別的每個使用者定義一元函式。

從運算子轉換：

- 應定義為靜態函式。
- 當可能會遺失有效位數時，可以是隱含的（適用於不會遺失有效位數的轉換，例如short to int）或 explicit。
- 應傳回包含類別的物件。
- 應將 "from" 類型當做唯一的參數類型。

下列範例顯示隱含和明確的「轉換自」、使用者定義的轉換(UDC)運算子。

```
// clr_udc_convert_from.cpp  
// compile with: /clr  
value struct MyDouble {  
    double d;  
  
    MyDouble(int i) {  
        d = static_cast<double>(i);  
        System::Console::WriteLine("in constructor");  
    }  
  
    // Wrap the constructor with a convert-from operator.  
    // implicit UDC because conversion cannot lose precision  
    static operator MyDouble (int i) {  
        System::Console::WriteLine("in operator");  
        // call the constructor  
        MyDouble d(i);  
        return d;  
    }  
  
    // an explicit user-defined conversion operator  
    static explicit operator signed short int (MyDouble) {  
        return 1;  
    }  
};  
  
int main() {  
    int i = 10;  
    MyDouble md = i;  
    System::Console::WriteLine(md.d);  
  
    // using explicit user-defined conversion operator requires a cast  
    unsigned short int j = static_cast<unsigned short int>(md);  
    System::Console::WriteLine(j);  
}
```

Output

```
in operator  
in constructor  
10  
1
```

轉換成運算子

轉換成運算子會將定義運算子的類別之物件轉換為其他物件。下列範例顯示隱含的轉換成使用者定義的轉換運算子：

```
// clr_udc_convert_to.cpp  
// compile with: /clr  
using namespace System;  
value struct MyInt {  
    Int32 i;  
  
    // convert MyInt to String^  
    static operator String^ ( MyInt val ) {  
        return val.i.ToString();  
    }  
  
    MyInt(int _i) : i(_i) {}  
};  
  
int main() {  
    MyInt mi(10);  
    String ^s = mi;  
    Console::WriteLine(s);  
}
```

Output

```
10
```

明確使用者定義的轉換轉換運算子適用於可能會以某種方式遺失資料的轉換。若要叫用明確的轉換成運算子，則必須使用 cast。

```
// clr_udc_convert_to_2.cpp  
// compile with: /clr  
value struct MyDouble {  
    double d;  
    // convert MyDouble to Int32  
    static explicit operator System::Int32 ( MyDouble val ) {  
        return (int)val.d;  
    }  
};  
  
int main() {  
    MyDouble d;  
    d.d = 10.3;  
    System::Console::WriteLine(d.d);  
    int i = 0;  
    i = static_cast<int>(d);  
    System::Console::WriteLine(i);  
}
```

Output

10.3

10

轉換泛型類別

您可以將泛型類別轉換成 T。

```
// clr_udc_generics.cpp
// compile with: /clr
generic<class T>
public value struct V {
    T mem;
    static operator T(V v) {
        return v.mem;
    }

    void f(T t) {
        mem = t;
    }
};

int main() {
    V<int> v;
    v.f(42);
    int i = v;
    i += v;
    System::Console::WriteLine(i == (42 * 2));
}
```

Output

```
True
```

轉換的函式會採用類型，並使用它來建立物件。轉換的函式只會使用直接初始化來呼叫；轉換不會叫用轉換的函式。根據預設，轉換為 CLR 類型的函式是明確的。

```
// clr_udc_converting_constructors.cpp
// compile with: /clr
public ref struct R {
    int m;
    char c;

    R(int i) : m(i) { }
    R(char j) : c(j) { }
};

public value struct V {
    R^ ptr;
    int m;

    V(R^ r) : ptr(r) { }
    V(int i) : m(i) { }
};

int main() {
    R^ r = gcnew R(5);

    System::Console::WriteLine( V(5).m);
    System::Console::WriteLine( V(r).ptr);
}
```

Output

```
5
R
```

在此程式碼範例中，隱含靜態轉換函式的作用與明確轉換的程式相同。

```
public value struct V {
    int m;
    V(int i) : m(i) {}
    static operator V(int i) {
        V v(i*100);
        return v;
    }
};

public ref struct R {
    int m;
    R(int i) : m(i) {}
    static operator R^(int i) {
        return gcnew R(i*100);
    }
};

int main() {
    V v(13);    // explicit
    R^ r = gcnew R(12);    // explicit

    System::Console::WriteLine(v.m);
    System::Console::WriteLine(r->m);

    // explicit ctor can't be called here: not ambiguous
    v = 5;
    r = 20;

    System::Console::WriteLine(v.m);
    System::Console::WriteLine(r->m);
}
```

Output

```
13
12
500
2000
```

請參閱

[類別和結構](#)

initonly (C++/CLI)

2019/12/10 • [Edit Online](#)

initonly是一個內容相關的關鍵字，表示變數指派只可以做為宣告的一部分或在相同類別的靜態函式中發生。

下列範例顯示如何使用 `initonly`：

```
// mcpp_initonly.cpp
// compile with: /clr /c
ref struct Y1 {
    initonly
    static int staticConst1;

    initonly
    static int staticConst2 = 0;

    static Y1() {
        staticConst1 = 0;
    }
};
```

請參閱

[類別和結構](#)

如何：定義和使用委派 (C++/CLI)

2020/11/2 • [Edit Online](#)

本文說明如何在 C++/CLI 中定義和使用委派。

雖然 .NET Framework 會提供一些委派，但有時候您可能必須定義新的委派。

下列程式碼範例會定義名為的委派 `MyCallback`。事件處理常式代碼(引發這個新委派時所呼叫的函式)必須具有的傳回型別 `void`，並接受 `String` 參考。

Main 函式會使用所定義的靜態方法來具現 `SomeClass` 化 `MyCallback` 委派。委派接著會變成呼叫此函式的替代方法，如藉由將字串 "single" 傳送至委派物件所示範。接下來，會將的其他實例 `MyCallback` 連結在一起，然後藉由一次呼叫委派物件來執行。

```

// use_delegate.cpp
// compile with: /clr
using namespace System;

ref class SomeClass
{
public:
    static void Func(String^ str)
    {
        Console::WriteLine("static SomeClass::Func - {0}", str);
    }
};

ref class OtherClass
{
public:
    OtherClass( Int32 n )
    {
        num = n;
    }

    void Method(String^ str)
    {
        Console::WriteLine("OtherClass::Method - {0}, num = {1}",
                           str, num);
    }

    Int32 num;
};

delegate void MyCallback(String^ str);

int main( )
{
    MyCallback^ callback = gcnew MyCallback(SomeClass::Func);
    callback("single");

    callback += gcnew MyCallback(SomeClass::Func);

    OtherClass^ f = gcnew OtherClass(99);
    callback += gcnew MyCallback(f, &OtherClass::Method);

    f = gcnew OtherClass(100);
    callback += gcnew MyCallback(f, &OtherClass::Method);

    callback("chained");

    return 0;
}

```

```

static SomeClass::Func - single
static SomeClass::Func - chained
static SomeClass::Func - chained
OtherClass::Method - chained, num = 99
OtherClass::Method - chained, num = 100

```

下一個程式碼範例顯示如何將委派與實值類別的成員產生關聯。

```
// mcppv2_del_mem_value_class.cpp
// compile with: /clr
using namespace System;
public delegate void MyDel();

value class A {
public:
    void func1() {
        Console::WriteLine("test");
    }
};

int main() {
    A a;
    A^ ah = a;
    MyDel^ f = gcnew MyDel(a, &A::func1); // implicit box of a
    f();
    MyDel^ f2 = gcnew MyDel(ah, &A::func1);
    f2();
}
```

```
test
test
```

如何撰寫委派

您可以使用 " - " 運算子，從組成的委派中移除元件委派。

```
// mcppv2_compose_delegates.cpp
// compile with: /clr
using namespace System;

delegate void MyDelegate(String ^ s);

ref class MyClass {
public:
    static void Hello(String ^ s) {
        Console::WriteLine("Hello, {0}!", s);
    }

    static void Goodbye(String ^ s) {
        Console::WriteLine(" Goodbye, {0}!", s);
    }
};

int main() {

    MyDelegate ^ a = gcnew MyDelegate(MyClass::Hello);
    MyDelegate ^ b = gcnew MyDelegate(MyClass::Goodbye);
    MyDelegate ^ c = a + b;
    MyDelegate ^ d = c - a;

    Console::WriteLine("Invoking delegate a:");
    a("A");
    Console::WriteLine("Invoking delegate b:");
    b("B");
    Console::WriteLine("Invoking delegate c:");
    c("C");
    Console::WriteLine("Invoking delegate d:");
    d("D");
}
```

輸出

```
Invoking delegate a:  
Hello, A!  
Invoking delegate b:  
Goodbye, B!  
Invoking delegate c:  
Hello, C!  
Goodbye, C!  
Invoking delegate d:  
Goodbye, D!
```

將委派 ^ 傳遞至預期函式指標的原生函式

從 managed 元件中，您可以使用函式指標參數呼叫原生函式，其中原生函式可以呼叫 managed 元件之委派的成員函式。

這個範例會建立匯出原生函式的 .dll：

```
// delegate_to_native_function.cpp  
// compile with: /LD  
#include < windows.h >  
extern "C" {  
    __declspec(dllexport)  
    void nativeFunction(void (CALLBACK *mgdFunc)(const char* str)) {  
        mgdFunc("Call to Managed Function");  
    }  
}
```

下一個範例會使用 .dll，並將委派控制碼傳遞至預期函式指標的原生函式。

```
// delegate_to_native_function_2.cpp  
// compile with: /clr  
using namespace System;  
using namespace System::Runtime::InteropServices;  
  
delegate void Del(String ^s);  
public ref class A {  
public:  
    void delMember(String ^s) {  
        Console::WriteLine(s);  
    }  
};  
  
[DllImportAttribute("delegate_to_native_function", CharSet=CharSet::Ansi)]  
extern "C" void nativeFunction(Del ^d);  
  
int main() {  
    A ^a = gcnew A;  
    Del ^d = gcnew Del(a, &A::delMember);  
    nativeFunction(d); // Call to native function  
}
```

輸出

```
Call to Managed Function
```

將委派與非受控函式產生關聯

若要讓委派與原生函式產生關聯，您必須將原生函式包裝在 managed 類型中，並宣告要透過叫用的函式 [PInvoke](#)

。

```
// mcppv2_del_to_umnangd_func.cpp
// compile with: /clr
#pragma unmanaged
extern "C" void printf(const char*, ...);
class A {
public:
    static void func(char* s) {
        printf(s);
    }
};

#pragma managed
public delegate void func(char*);

ref class B {
A* ap;

public:
    B(A* ap):ap(ap) {}
    void func(char* s) {
        ap->func(s);
    }
};

int main() {
    A* a = new A;
    B^ b = gcnew B(a);
    func^ f = gcnew func(b, &B::func);
    f("hello");
    delete a;
}
```

輸出

```
hello
```

若要使用未系結的委派

您可以使用未系結的委派，在呼叫委派時，傳遞您想要呼叫其函式之類型的實例。

如果您想要逐一查看集合中的物件（藉由使用 [for each](#)、[in](#) 關鍵字），並在每個實例上呼叫成員函式，則未系結的委派會特別有用。

以下是宣告、具現化和呼叫系結和未系結委派的方法：

宣告	具現化	呼叫
	委派簽章必須符合您想要透過委派呼叫之函式的簽章。	委派簽章的第一個參數是您想要呼叫之物件的型別 this 。 在第一個參數之後，委派簽章必須符合您想要透過委派呼叫之函式的簽章。

化	<p>當您具現化系結委派時，您可以指定實例函數或全域或靜態成員函式。</p> <p>若要指定實例函式，第一個參數是您想要呼叫其成員函式之類型的實例，而第二個參數則是您想要呼叫之函式的位址。</p> <p>如果您想要呼叫全域或靜態成員函式，只要傳遞全域函式的名稱或靜態成員函式的名稱即可。</p>	當您具現化未系結的委派時，只要傳遞您要呼叫之函式的位址即可。
呼叫	<p>當您呼叫系結委派時，只要傳遞委派簽章所需的參數即可。</p>	與系結委派相同，但請記住，第一個參數必須是物件的實例，其中包含您想要呼叫的函數。

這個範例示範如何宣告、具現化和呼叫未系結的委派：

```

// unbound_delegates.cpp
// compile with: /clr
ref struct A {
    A(){}
    A(int i) : m_i(i){}
    void Print(int i) { System::Console::WriteLine(m_i + i);}

private:
    int m_i;
};

value struct V {
    void Print() { System::Console::WriteLine(m_i);}
    int m_i;
};

delegate void Delegate1(A^, int i);
delegate void Delegate2(A%, int i);

delegate void Delegate3(interior_ptr<V>);
delegate void Delegate4(V%);

delegate void Delegate5(int i);
delegate void Delegate6();

int main() {
    A^ a1 = gcnew A(1);
    A% a2 = *gcnew A(2);

    Delegate1 ^ Unbound_Delegate1 = gcnew Delegate1(&A::Print);
    // delegate takes a handle
    Unbound_Delegate1(a1, 1);
    Unbound_Delegate1(%a2, 1);

    Delegate2 ^ Unbound_Delegate2 = gcnew Delegate2(&A::Print);
    // delegate takes a tracking reference (must deference the handle)
    Unbound_Delegate2(*a1, 1);
    Unbound_Delegate2(a2, 1);

    // instantiate a bound delegate to an instance member function
    Delegate5 ^ Bound_Del = gcnew Delegate5(a1, &A::Print);
    Bound_Del(1);

    // instantiate value types
    V v1 = {7};
    V v2 = {8};

    Delegate3 ^ Unbound_Delegate3 = gcnew Delegate3(&V::Print);
    Unbound_Delegate3(&v1);
    Unbound_Delegate3(&v2);

    Delegate4 ^ Unbound_Delegate4 = gcnew Delegate4(&V::Print);
    Unbound_Delegate4(v1);
    Unbound_Delegate4(v2);

    Delegate6 ^ Bound_Delegate3 = gcnew Delegate6(v1, &V::Print);
    Bound_Delegate3();
}

```

輸出

```
2
3
2
3
2
7
8
7
8
7
```

下一個範例示範如何使用未系結的委派和for each, 在關鍵字中逐一查看集合中的物件，並在每個實例上呼叫成員函式。

```
// unbound_delegates_2.cpp
// compile with: /clr
using namespace System;

ref class RefClass {
    String^ _Str;

public:
    RefClass( String^ str ) : _Str( str ) {}
    void Print() { Console::Write( _Str ); }
};

delegate void PrintDelegate( RefClass^ );

int main() {
    PrintDelegate^ d = gcnew PrintDelegate( &RefClass::Print );

    array< RefClass^ >^ a = gcnew array<RefClass^>( 10 );

    for ( int i = 0; i < a->Length; ++i )
        a[i] = gcnew RefClass( i.ToString() );

    for each ( RefClass^ R in a )
        d( R );

    Console::WriteLine();
}
```

這個範例會建立屬性存取子函式的未系結委派：

```

// unbound_delegates_3.cpp
// compile with: /clr
ref struct B {
    property int P1 {
        int get() { return m_i; }
        void set(int i) { m_i = i; }
    }

private:
    int m_i;
};

delegate void DelBSet(B^, int);
delegate int DelBGet(B^);

int main() {
    B^ b = gcnew B;

    DelBSet^ delBSet = gcnew DelBSet(&B::P1::set);
    delBSet(b, 11);

    DelBGet^ delBGet = gcnew DelBGet(&B::P1::get);
    System::Console::WriteLine(delBGet(b));
}

```

輸出

```
11
```

下列範例示範如何叫用多播委派，其中一個實例已系結，另一個實例已解除系結。

```

// unbound_delegates_4.cpp
// compile with: /clr
ref class R {
public:
    R(int i) : m_i(i) {}

    void f(R ^ r) {
        System::Console::WriteLine("in f(R ^ r)");
    }

    void f() {
        System::Console::WriteLine("in f()");
    }

private:
    int m_i;
};

delegate void Del(R ^);

int main() {
    R ^r1 = gcnew R(11);
    R ^r2 = gcnew R(12);

    Del^ d = gcnew Del(r1, &R::f);
    d += gcnew Del(&R::f);
    d(r2);
}

```

輸出

```
in f(R ^ r)
in f()
```

下一個範例顯示如何建立和呼叫未系結的泛型委派。

```
// unbound_delegates_5.cpp
// compile with: /clr
ref struct R {
    R(int i) : m_i(i) {}

    int f(R ^) { return 999; }
    int f() { return m_i + 5; }

    int m_i;
};

value struct V {
    int f(V%) { return 999; }
    int f() { return m_i + 5; }

    int m_i;
};

generic <typename T>
delegate int Del(T t);

generic <typename T>
delegate int DelV(T% t);

int main() {
    R^ hr = gcnew R(7);
    System::Console::WriteLine((gcnew Del<R^>(&R::f))(hr));

    V v;
    v.m_i = 9;
    System::Console::WriteLine((gcnew DelV<V >(&V::f))(v) );
}
```

輸出

```
12
14
```

另請參閱

[delegate \(C++ 元件擴充功能\)](#)

作法：在 C++/CLI 中定義和使用列舉

2020/11/2 • [Edit Online](#)

本主題討論 c + +/CLI 中的列舉

指定列舉的基本類型

根據預設，列舉的基本類型是 `int`。不過，您可以指定要簽署的類型或 `int`、`short`、`long`、`_int32` 或的不帶正負號形式 `_int64`。您也可以使用 `char`。

```
// mcppv2_enum_3.cpp
// compile with: /clr
public enum class day_char : char {sun, mon, tue, wed, thu, fri, sat};

int main() {
    // fully qualified names, enumerator not injected into scope
    day_char d = day_char::sun, e = day_char::mon;
    System::Console::WriteLine(d);
    char f = (char)d;
    System::Console::WriteLine(f);
    f = (char)e;
    System::Console::WriteLine(f);
    e = day_char::tue;
    f = (char)e;
    System::Console::WriteLine(f);
}
```

輸出

```
sun
0
1
2
```

如何在 managed 和標準列舉之間轉換

列舉和整數類資料類型之間沒有標準轉換;需要轉換。

```
// mcppv2_enum_4.cpp
// compile with: /clr
enum class day {sun, mon, tue, wed, thu, fri, sat};
enum {sun, mon, tue, wed, thu, fri, sat} day2; // unnamed std enum

int main() {
    day a = day::sun;
    day2 = sun;
    if ((int)a == day2)
        // or...
    // if (a == (day)day2)
        System::Console::WriteLine("a and day2 are the same");
    else
        System::Console::WriteLine("a and day2 are not the same");
}
```

輸出

```
a and day2 are the same
```

運算子和列舉

下列運算子在 C++/CLI 的列舉中是有效的：

!!!

`== != < > <= >=`

`+ -`

`| ^ & ~`

`++ --`

`sizeof`

運算子 `| ^ & ~ + --` 僅針對具有整數基礎類型的列舉(不包括 `bool`)定義。這兩個運算元都必須是列舉型別。

編譯器不會對列舉運算的結果進行靜態或動態檢查；作業可能會導致值不在列舉的有效枚舉器範圍內。

NOTE

C++11 引進了非受控程式碼中的列舉類別類型，這與 C++/CLI 中的 managed 列舉類別截然不同。特別是 C++11 列舉類別類型不支援與 C++/CLI 中的 managed 列舉類別類型相同的運算子，而 C++/CLI 原始程式碼必須在 managed enum 類別宣告中提供存取範圍規範，才能區別它們與非受控(C++11)列舉類別宣告。如需 C++/CLI、C++/CX 和 C++11 中列舉類別的詳細資訊，請參閱[enum 類別](#)。

```
// mcppv2_enum_5.cpp
// compile with: /clr
private enum class E { a, b } e, mask;
int main() {
    if ( e & mask )    // C2451 no E->bool conversion
    ;
    if ( ( e & mask ) != 0 )   // C3063 no operator!= (E, int)
    ;
    if ( ( e & mask ) != E() )   // OK
    ;
}
```

```
// mcppv2_enum_6.cpp
// compile with: /clr
private enum class day : int {sun, mon};
enum : bool {sun = true, mon = false} day2;

int main() {
    day a = day::sun, b = day::mon;
    day2 = sun;

    System::Console::WriteLine(sizeof(a));
    System::Console::WriteLine(sizeof(day2));
    a++;
    System::Console::WriteLine(a == b);
}
```

輸出

```
4
1
True
```

另請參閱

[enum 類別](#)

如何：在 C++/CLI 中使用事件

2020/11/2 • [Edit Online](#)

本文說明如何使用宣告事件的介面，以及用來叫用該事件的函式，以及用來執行介面的類別和事件處理常式。

介面事件

下列程式碼範例會加入事件處理常式，叫用事件，這會導致事件處理常式將其名稱寫入主控台，然後移除事件處理常式。

```
// mcppv2_events2.cpp
// compile with: /clr
using namespace System;

delegate void Del(int, float);

// interface that has an event and a function to invoke the event
interface struct I {
public:
    event Del ^ E;
    void fire(int, float);
};

// class that implements the interface event and function
ref class EventSource: public I {
public:
    virtual event Del^ E;
    virtual void fire(int i, float f) {
        E(i, f);
    }
};

// class that defines the event handler
ref class EventReceiver {
public:
    void Handler(int i , float f) {
        Console::WriteLine("EventReceiver::Handler");
    }
};

int main () {
    I^ es = gcnew EventSource();
    EventReceiver^ er = gcnew EventReceiver();

    // hook the handler to the event
    es->E += gcnew Del(er, &EventReceiver::Handler);

    // call the event
    es -> fire(1, 3.14);

    // unhook the handler from the event
    es->E -= gcnew Del(er, &EventReceiver::Handler);
}
```

輸出

```
EventReceiver::Handler
```

自訂存取子方法

下列範例示範如何在新增或移除處理常式時，以及引發事件時，定義事件的行為。

```
// mcppv2_events6.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

ref class EventSource {
public:
    MyDel ^ pE;
    MyDel2 ^ pE2;

    event MyDel^ E {
        void add(MyDel^ p) {
            pE = static_cast<MyDel^> (Delegate::Combine(pE, p));
            // cannot refer directly to the event
            // E = static_cast<MyDel^> (Delegate::Combine(pE, p));    // error
        }
        void remove(MyDel^ p) {
            pE = static_cast<MyDel^> (Delegate::Remove(pE, p));
        }
        void raise() {
            if (pE != nullptr)
                pE->Invoke();
        }
    } // E event block

    event MyDel2^ E2 {
        void add(MyDel2^ p2) {
            pE2 = static_cast<MyDel2^> (Delegate::Combine(pE2, p2));
        }
        void remove(MyDel2^ p2) {
            pE2 = static_cast<MyDel2^> (Delegate::Remove(pE2, p2));
        }
        int raise(int i, float f) {
            if (pE2 != nullptr) {
                return pE2->Invoke(i, f);
            }
            return 1;
        }
    } // E2 event block
};

public ref struct EventReceiver {
    void H1() {
        Console::WriteLine("In event handler H1");
    }

    int H2(int i, float f) {
        Console::WriteLine("In event handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};

int main() {
    EventSource ^ pE = gcnew EventSource;
    EventReceiver ^ pR = gcnew EventReceiver;

    // hook event handlers
    pE->E += pR->H1;
    pE2->E2 += pR->H2;
}
```

```
pE->E += gcnew MyDel(pR, &EventReceiver::H1);
pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

// raise events
pE->E();
pE->E2::raise(1, 2.2); // call event through scope path

// unhook event handlers
pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

// raise events, but no handlers
pE->E();
pE->E2::raise(1, 2.5);
}
```

輸出

```
In event handler H1
In event handler H2 with args 1 and 2.2
```

覆寫 add、remove 和 raise 存取子上的預設存取

這個範例會示範如何覆寫 add、remove 和 raise 事件方法上的預設存取：

```

// mcppv2_events3.cpp
// compile with: /clr
public delegate void f(int);

public ref struct E {
    f ^ _E;
public:
    void handler(int i) {
        System::Console::WriteLine(i);
    }

    E() {
        _E = nullptr;
    }

    event f^ Event {
        void add(f ^ d) {
            _E += d;
        }
    private:
        void remove(f ^ d) {
            _E -= d;
        }
    }

protected:
    void raise(int i) {
        if (_E) {
            _E->Invoke(i);
        }
    }
}

// a member function to access all event methods
static void Go() {
    E^ pE = gcnew E;
    pE->Event += gcnew f(pE, &E::handler);
    pE->Event(17); // prints 17
    pE->Event -= gcnew f(pE, &E::handler);
    pE->Event(17); // no output
}
};

int main() {
    E::Go();
}

```

輸出

17

多個事件處理常式

事件接收器或任何其他用戶端程式代碼，可以將一個或多個處理常式加入至事件。

```

// mcppv2_events4.cpp
// compile with: /clr
using namespace System;
#include <stdio.h>

delegate void ClickEventHandler(int, double);
delegate void DblClickEventHandler(String^);

ref class EventSource {
public:
    event ClickEventHandler^ OnClick;
    event DblClickEventHandler^ OnDblClick;

    void FireEvents() {
        OnClick(7, 3.14159);
        OnDblClick("Started");
    }
};

ref struct EventReceiver {
public:
    void Handler1(int x, double y) {
        System::Console::Write("Click(x={0},y={1})\n", x, y);
    }

    void Handler2(String^ s) {
        System::Console::Write("DblClick(s={0})\n", s);
    }

    void Handler3(String^ s) {
        System::Console::WriteLine("DblClickAgain(s={0})\n", s);
    }

    void AddHandlers(EventSource^ pES) {
        pES->OnClick +=
            gcnew ClickEventHandler(this,&EventReceiver::Handler1);
        pES->OnDblClick +=
            gcnew DblClickEventHandler(this,&EventReceiver::Handler2);
        pES->OnDblClick +=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler3);
    }

    void RemoveHandlers(EventSource^ pES) {
        pES->OnClick -=
            gcnew ClickEventHandler(this, &EventReceiver::Handler1);
        pES->OnDblClick -=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler2);
        pES->OnDblClick -=
            gcnew DblClickEventHandler(this, &EventReceiver::Handler3);
    }
};

int main() {
    EventSource^ pES = gcnew EventSource;
    EventReceiver^ pER = gcnew EventReceiver;

    // add handlers
    pER->AddHandlers(pES);

    pES->FireEvents();

    // remove handlers
    pER->RemoveHandlers(pES);
}

```

輸出

```
Click(x=7,y=3.14159)
DblClick(s=System.Char[])
DblClickAgain(s=System.Char[])
```

靜態事件

下列範例顯示如何定義和使用靜態事件。

```
// mcppv2_events7.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

ref class EventSource {
public:
    static MyDel ^ psE;
    static event MyDel2 ^ E2; // event keyword, compiler generates add,
                           // remove, and Invoke

    static event MyDel ^ E {
        static void add(MyDel ^ p) {
            psE = static_cast<MyDel^> (Delegate::Combine(psE, p));
        }
    }

    static void remove(MyDel^ p) {
        psE = static_cast<MyDel^> (Delegate::Remove(psE, p));
    }

    static void raise() {
        if (psE != nullptr) //psE!=0 -> C2679, use nullptr
            psE->Invoke();
    }
};

static int Fire_E2(int i, float f) {
    return E2(i, f);
}

public ref struct EventReceiver {
    void H1() {
        Console::WriteLine("In event handler H1");
    }

    int H2(int i, float f) {
        Console::WriteLine("In event handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};

int main() {
    EventSource^ pE = gcnew EventSource;
    EventReceiver^ pR = gcnew EventReceiver;

    // Called with "this"
    // hook event handlers
    pE->E += gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events
    pE->E();
    pE->Fire_E2(11, 11.11);
```

```

// unhook event handlers
pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

// Not called with "this"
// hook event handler
EventSource::E += gcnew MyDel(pR, &EventReceiver::H1);
EventSource::E2 += gcnew MyDel2(pR, &EventReceiver::H2);

// raise events
EventSource::E();
EventSource::Fire_E2(22, 22.22);

// unhook event handlers
EventSource::E -= gcnew MyDel(pR, &EventReceiver::H1);
EventSource::E2 -= gcnew MyDel2(pR, &EventReceiver::H2);
}

```

輸出

```

In event handler H1
In event handler H2 with args 11 and 11.11
In event handler H1
In event handler H2 with args 22 and 22.22

```

虛擬事件

這個範例會在介面和類別中，執行虛擬的 managed 事件：

```

// mcppv2_events5.cpp
// compile with: /clr
using namespace System;

public delegate void MyDel();
public delegate int MyDel2(int, float);

// managed class that has a virtual event
ref class IEFace {
public:
    virtual event MyDel ^ E; // declares three accessors (add, remove, and raise)
};

// managed interface that has a virtual event
public interface struct IEFace2 {
public:
    event MyDel2 ^ E2; // declares two accessors (add and remove)
};

// implement virtual events
ref class EventSource : public IEFace, public IEFace2 {
public:
    virtual event MyDel2 ^ E2;

    void Fire_E() {
        E();
    }

    int Fire_E2(int i, float f) {
        try {
            return E2(i, f);
        }
        catch(System::NullReferenceException^) {
            return 0; // no handlers
        }
    }
}

```

```

}

// class to hold event handlers, the event receiver
public ref struct EventReceiver {
    // first handler
    void H1() {
        Console::WriteLine("In handler H1");
    }

    // second handler
    int H2(int i, float f) {
        Console::WriteLine("In handler H2 with args {0} and {1}", i.ToString(), f.ToString());
        return 0;
    }
};

int main() {
    EventSource ^ pE = gcnew EventSource;
    EventReceiver ^ pR = gcnew EventReceiver;

    // add event handlers
    pE->E += gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 += gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events
    pE->Fire_E();
    pE->Fire_E2(1, 2.2);

    // remove event handlers
    pE->E -= gcnew MyDel(pR, &EventReceiver::H1);
    pE->E2 -= gcnew MyDel2(pR, &EventReceiver::H2);

    // raise events, but no handlers; so, no effect
    pE->Fire_E();
    pE->Fire_E2(1, 2.5);
}

```

輸出

```

In handler H1
In handler H2 with args 1 and 2.2

```

不能指定簡單事件來覆寫或隱藏基類事件。您必須定義所有事件的存取子函式，然後 `new` `override` 在每個存取子函數上指定或關鍵字。

```

// mcppv2_events5_a.cpp
// compile with: /clr /c
delegate void Del();

ref struct A {
    virtual event Del ^E;
    virtual event Del ^E2;
};

ref struct B : A {
    virtual event Del ^E override; // C3797
    virtual event Del ^E2 new; // C3797
};

ref struct C : B {
    virtual event Del ^E { // OK
        void raise() override {}
        void add(Del ^) override {}
        void remove(Del^) override {}
    }

    virtual event Del ^E2 { // OK
        void raise() new {}
        void add(Del ^) new {}
        void remove(Del^) new {}
    }
};

```

抽象事件

下列範例顯示如何執行抽象事件。

```

// mcppv2_events10.cpp
// compile with: /clr /W1
using namespace System;
public delegate void Del();
public delegate void Del2(String^ s);

interface struct IEvent {
public:
    // in this case, no raised method is defined
    event Del^ Event1;

    event Del2^ Event2 {
public:
        void add(Del2^ _d);
        void remove(Del2^ _d);
        void raise(String^ s);
    }

    void fire();
};

ref class EventSource: public IEvent {
public:
    virtual event Del^ Event1;
    event Del2^ Event2 {
        virtual void add(Del2^ _d) {
            d = safe_cast<Del2^>(System::Delegate::Combine(d, _d));
        }

        virtual void remove(Del2^ _d) {
            d = safe_cast<Del2^>(System::Delegate::Remove(d, _d));
        }
    }
};

```

```

        virtual void raise(String^ s) {
            if (d) {
                d->Invoke(s);
            }
        }

        virtual void fire() {
            return Event1();
        }

private:
    Del2^ d;
};

ref class EventReceiver {
public:
    void func() {
        Console::WriteLine("hi");
    }

    void func(String^ str) {
        Console::WriteLine(str);
    }
};

int main () {
    IEvent^ es = gcnew EventSource;
    EventReceiver^ er = gcnew EventReceiver;
    es->Event1 += gcnew Del(er, &EventReceiver::func);
    es->Event2 += gcnew Del2(er, &EventReceiver::func);

    es->fire();
    es->Event2("hello from Event2");
    es->Event1 -= gcnew Del(er, &EventReceiver::func);
    es->Event2 -= gcnew Del2(er, &EventReceiver::func);
    es->Event2("hello from Event2");
}

```

輸出

```

hi
hello from Event2

```

引發在不同元件中定義的事件

事件和事件處理常式可以在一個元件中定義，並由另一個元件使用。

```

// mcppv2_events8.cpp
// compile with: /LD /clr
using namespace System;

public delegate void Del(String^ s);

public ref class Source {
public:
    event Del^ Event;
    void Fire(String^ s) {
        Event(s);
    }
};

```

此用戶端程式代碼會使用事件：

```
// mcppv2_events9.cpp
// compile with: /clr
#using "mcppv2_events8.dll"
using namespace System;

ref class Receiver {
public:
    void Handler(String^ s) {
        Console::WriteLine(s);
    }
};

int main() {
    Source^ src = gcnew Source;
    Receiver^ rc1 = gcnew Receiver;
    Receiver^ rc2 = gcnew Receiver;
    src -> Event += gcnew Del(rc1, &Receiver::Handler);
    src -> Event += gcnew Del(rc2, &Receiver::Handler);
    src->Fire("hello");
    src -> Event -= gcnew Del(rc1, &Receiver::Handler);
    src -> Event -= gcnew Del(rc2, &Receiver::Handler);
}
```

輸出

```
hello
hello
```

另請參閱

[event](#)

如何：定義介面靜態建構函式 (C++/CLI)

2019/12/10 • [Edit Online](#)

可具備靜態建構函式的介面，您可用它來初始化靜態資料成員。靜態建構函式最多只能呼叫一次，而且會在第一次存取靜態介面成員之前呼叫。

範例

```
// mcppv2_interface_class2.cpp
// compile with: /clr
using namespace System;

interface struct MyInterface {
    static int i;
    static void Test() {
        Console::WriteLine(i);
    }

    static MyInterface() {
        Console::WriteLine("in MyInterface static constructor");
        i = 99;
    }
};

ref class MyClass : public MyInterface {};

int main() {
    MyInterface::Test();
    MyClass::MyInterface::Test();

    MyInterface ^ mi = gcnew MyClass;
    mi->Test();
}
```

```
in MyInterface static constructor
99
99
99
```

請參閱

[介面類別](#)

如何：在原生編譯中宣告覆寫指定名稱 (C++/CLI)

2020/11/2 • [Edit Online](#)

密封、抽象和覆寫適用於未使用 /ZW 或 /clr 的編譯。

NOTE

ISO C++11 標準語言有覆寫識別碼和最終識別碼，在 Visual Studio 用途中 `final`（而不是以 `sealed` 原生方式編譯的程式碼）支援這兩者。

範例：sealed 有效

描述

下列範例顯示 `sealed` 在原生編譯中是有效的。

程式碼

```
// sealed_native_keyword.cpp
#include <stdio.h>
__interface I1 {
    virtual void f();
    virtual void g();
};

class X : public I1 {
public:
    virtual void g() sealed {};
};

class Y : public X {
public:

    // the following override generates a compiler error
    virtual void g() {} // C3248 X::g is sealed!
};
```

範例：覆寫有效

描述

下一個範例顯示 `override` 在原生編譯中是有效的。

程式碼

```
// override_native_keyword.cpp
#include <stdio.h>
__interface I1 {
    virtual void f();
};

class X : public I1 {
public:
    virtual void f() override {} // OK
    virtual void g() override {} // C3668 I1::g does not exist
};
```

範例：abstract 有效

描述

此範例顯示 `abstract` 在原生編譯中是有效的。

程式碼

```
// abstract_native_keyword.cpp
class X abstract {};

int main() {
    X * MyX = new X;    // C3622 cannot instantiate abstract class
}
```

另請參閱

[覆寫規範](#)

如何：在 C++/CLI 中使用屬性

2020/11/2 • [Edit Online](#)

本文說明如何使用 C++/CLI 中的屬性

基本特性

對於僅指派和取出私用資料成員的基本屬性(property)，您不需要明確定義 get 和 set 存取子函式，因為編譯器會在僅提供屬性的資料類型時，自動提供它們。此程式碼示範基本屬性：

```
// SimpleProperties.cpp
// compile with: /clr
using namespace System;

ref class C {
public:
    property int Size;
};

int main() {
    C^ c = gcnew C;
    c->Size = 111;
    Console::WriteLine("c->Size = {0}", c->Size);
}
```

```
c->Size = 111
```

靜態屬性

這個程式碼範例會示範如何宣告和使用靜態屬性。靜態屬性只能存取其類別的靜態成員。

```
// mcppv2_property_3.cpp
// compile with: /clr
using namespace System;

ref class StaticProperties {
    static int MyInt;
    static int MyInt2;

public:
    static property int Static_Data_Member_Property;

    static property int Static_Block_Property {
        int get() {
            return MyInt;
        }

        void set(int value) {
            MyInt = value;
        }
    }
};

int main() {
    StaticProperties::Static_Data_Member_Property = 96;
    Console::WriteLine(StaticProperties::Static_Data_Member_Property);

    StaticProperties::Static_Block_Property = 47;
    Console::WriteLine(StaticProperties::Static_Block_Property);
}
```

96
47

索引屬性

索引屬性通常會公開使用注標運算子存取的資料結構。

如果您使用預設的索引屬性，只要參考類別名稱就可以存取資料結構，但是如果您使用使用者定義的索引屬性，就必須指定屬性名稱來存取資料結構。

如需如何使用以 c# 撰寫之索引子的詳細資訊，請參閱 [如何：使用 c# 索引子 \(c++/cli\)](#)。

這個程式碼範例會示範如何使用預設值和使用者定義的索引屬性：

```

// mcppv2_property_2.cpp
// compile with: /clr
using namespace System;
public ref class C {
    array<int>^ MyArr;

public:
    C() {
        MyArr = gcnew array<int>(5);
    }

    // default indexer
    property int default[int] {
        int get(int index) {
            return MyArr[index];
        }
        void set(int index, int value) {
            MyArr[index] = value;
        }
    }

    // user-defined indexer
    property int indexer1[int] {
        int get(int index) {
            return MyArr[index];
        }
        void set(int index, int value) {
            MyArr[index] = value;
        }
    }
};

int main() {
    C ^ MyC = gcnew C();

    // use the default indexer
    Console::Write("[ ");
    for (int i = 0 ; i < 5 ; i++) {
        MyC[i] = i;
        Console::Write("{0} ", MyC[i]);
    }

    Console::WriteLine("]");
}

// use the user-defined indexer
Console::Write("[ ");
for (int i = 0 ; i < 5 ; i++) {
    MyC->indexer1[i] = i * 2;
    Console::Write("{0} ", MyC->indexer1[i]);
}

Console::WriteLine("]");
}

```

```
[ 0 1 2 3 4 ]
[ 0 2 4 6 8 ]
```

下一個範例顯示如何使用指標呼叫預設索引子 `this`。

```

// call_default_indexer_through_this_pointer.cpp
// compile with: /clr /c
value class Position {
public:
    Position(int x, int y) : position(gcnew array<int, 2>(100, 100)) {
        this->default[x, y] = 1;
    }

    property int default[int, int] {
        int get(int x, int y) {
            return position[x, y];
        }

        void set(int x, int y, int value) {}
    }

private:
    array<int, 2> ^ position;
};

```

這個範例會示範如何使用 [DefaultMemberAttribute](#) 來指定預設的索引子：

```

// specify_default_indexer.cpp
// compile with: /LD /clr
using namespace System;
[Reflection::DefaultMember("XXX")]
public ref struct Squares {
    property Double XXX[Double] {
        Double get(Double data) {
            return data*data;
        }
    }
};

```

下一個範例會使用在上一個範例中建立的中繼資料。

```

// consume_default_indexer.cpp
// compile with: /clr
#using "specify_default_indexer.dll"
int main() {
    Squares ^ square = gcnew Squares();
    System::Console::WriteLine("{0}", square[3]);
}

```

9

虛擬屬性

此程式碼範例示範如何宣告和使用虛擬屬性：

```

// mcppv2_property_4.cpp
// compile with: /clr
using namespace System;
interface struct IFFace {
public:
    property int VirtualProperty1;
    property int VirtualProperty2 {
        int get();
        void set(int i);
    }
};

// implement virtual events
ref class PropImpl : public IFFace {
    int MyInt;
public:
    virtual property int VirtualProperty1;

    virtual property int VirtualProperty2 {
        int get() {
            return MyInt;
        }
        void set(int i) {
            MyInt = i;
        }
    }
};

int main() {
    PropImpl ^ MyPI = gcnew PropImpl();
    MyPI->VirtualProperty1 = 93;
    Console::WriteLine(MyPI->VirtualProperty1);

    MyPI->VirtualProperty2 = 43;
    Console::WriteLine(MyPI->VirtualProperty2);
}

```

93

43

Abstract 和 sealed 屬性

雖然 `abstract` 和 `SEALED` 關鍵字在 ECMA C++/CLI 規格中指定為有效，但是針對 Microsoft C++ 編譯器，您無法在簡單屬性上指定它們，也不能在非一般屬性的屬性宣告上指定它們。

若要宣告密封或抽象屬性，您必須定義非一般屬性，然後 `abstract` `sealed` 在 `get` 和 `set` 存取子函式上指定或關鍵字。

```

// properties_abstract_sealed.cpp
// compile with: /clr
ref struct A {
protected:
    int m_i;

public:
    A() { m_i = 87; }

    // define abstract property
    property int Prop_1 {
        virtual int get() abstract;
        virtual void set(int i) abstract;
    }
};

ref struct B : A {
private:
    int m_i;

public:
    B() { m_i = 86; }

    // implement abstract property
    property int Prop_1 {
        virtual int get() override { return m_i; }
        virtual void set(int i) override { m_i = i; }
    }
};

ref struct C {
private:
    int m_i;

public:
    C() { m_i = 87; }

    // define sealed property
    property int Prop_2 {
        virtual int get() sealed { return m_i; }
        virtual void set(int i) sealed { m_i = i; }
    }
};

int main() {
    B b1;
    // call implementation of abstract property
    System::Console::WriteLine(b1.Prop_1);

    C c1;
    // call sealed property
    System::Console::WriteLine(c1.Prop_2);
}

```

86
87

多維度屬性

您可以使用多維度屬性來定義採用非標準參數的屬性存取子方法。

```
// mcppv2_property_5.cpp
// compile with: /clr
ref class X {
    double d;
public:
    X() : d(0) {}
    property double MultiDimProp[int, int, int] {
        double get(int, int, int) {
            return d;
        }
        void set(int i, int j, int k, double l) {
            // do something with those ints
            d = l;
        }
    }

    property double MultiDimProp2[int] {
        double get(int) {
            return d;
        }
        void set(int i, double l) {
            // do something with those ints
            d = l;
        }
    }
};

int main() {
    X ^ MyX = gcnew X();
    MyX->MultiDimProp[0,0,0] = 1.1;
    System::Console::WriteLine(MyX->MultiDimProp[0, 0, 0]);
}
```

1.1

多載屬性存取子

下列範例示範如何多載已編制索引的屬性。

```
// mcppv2_property_6.cpp
// compile with: /clr
ref class X {
    double d;
public:
    X() : d(0.0) {}
    property double MyProp[int] {
        double get(int i) {
            return d;
        }

        double get(System::String ^ i) {
            return 2*d;
        }

        void set(int i, double l) {
            d = i * l;
        }
    } // end MyProp definition
};

int main() {
    X ^ MyX = gcnew X();
    MyX->MyProp[2] = 1.7;
    System::Console::WriteLine(MyX->MyProp[1]);
    System::Console::WriteLine(MyX->MyProp["test"]);
}
```

3.4
6.8

另請參閱

[property](#)

作法：在 C++/CLI 中使用 safe_cast

2020/11/2 • [Edit Online](#)

本文說明如何在 c++/CLI 應用程式中使用 safe_cast。如需 c++/CX 中 safe_cast 的詳細資訊，請參閱[safe_cast](#)。

Upcasting

向上轉換是從衍生類型轉型為它的其中一個基類。這種轉換是安全的，不需要明確的轉換標記法。下列範例顯示如何使用和不搭配執行向上轉換 `safe_cast`。

```
// safe_upcast.cpp
// compile with: /clr
using namespace System;
interface class A {
    void Test();
};

ref struct B : public A {
    virtual void Test() {
        Console::WriteLine("in B::Test");
    }
    void Test2() {
        Console::WriteLine("in B::Test2");
    }
};

ref struct C : public B {
    virtual void Test() override {
        Console::WriteLine("in C::Test");
    }
};

int main() {
    C ^ c = gcnew C;

    // implicit upcast
    B ^ b = c;
    b->Test();
    b->Test2();

    // upcast with safe_cast
    b = nullptr;
    b = safe_cast<B^>(c);
    b->Test();
    b->Test2();
}
```

```
in C::Test
in B::Test2
in C::Test
in B::Test2
```

向下檢視

轉換是從基類轉換成衍生自基類的類別。只有在執行時間所定址的物件實際定址衍生類別物件時，向下轉換才是安

全的。不同 `static_cast` `safe_cast` 于，會執行動態檢查，並 `InvalidCastException` 在轉換失敗時擲回。

```
// safe_downcast.cpp
// compile with: /clr
using namespace System;

interface class A { void Test(); };

ref struct B : public A {
    virtual void Test() {
        Console::WriteLine("in B::Test()");
    }
};

void Test2() {
    Console::WriteLine("in B::Test2()");
}
};

ref struct C : public B {
    virtual void Test() override {
        Console::WriteLine("in C::Test()");
    }
};
};

interface class I {};

value struct V : public I {};

int main() {
    A^ a = gcnew C();
    a->Test();
    B^ b = safe_cast<B^>(a);
    b->Test();
    b->Test2();

    V v;
    I^ i = v; // i boxes V
    V^ refv = safe_cast<V^>(i);

    Object^ o = gcnew B;
    A^ a2= safe_cast<A^>(o);
}
```

```
in C::Test()
in C::Test()
in B::Test2()
```

使用者定義的轉換 `safe_cast`

下一個範例會顯示如何使用來叫用 `safe_cast` 使用者定義的轉換。

```

// safe_cast_udc.cpp
// compile with: /clr
using namespace System;
value struct V;

ref struct R {
    int x;
    R() {
        x = 1;
    }

    R(int argx) {
        x = argx;
    }

    static operator R^>(R^ r);
};

value struct V {
    int x;
    static operator R^>(V& v) {
        Console::WriteLine("in operator R^>(V& v)");
        R^ r = gcnew R();
        r->x = v.x;
        return r;
    }

    V(int argx) {
        x = argx;
    }
};

R::operator V^>(R^ r) {
    Console::WriteLine("in operator V^>(R^ r)");
    return gcnew V(r->x);
}

int main() {
    bool fReturnVal = false;
    V v(2);
    R^ r = safe_cast<R^>(v);    // should invoke UDC
    V^ v2 = safe_cast<V^>(r);    // should invoke UDC
}

```

```

in operator R^>(V& v
in operator V^>(R^ r)

```

safe_cast 和裝箱作業

Box 處理

「裝箱」定義為編譯器插入的使用者定義轉換。因此，您可以使用 `safe_cast` 來 BOX CLR 堆積上的值。

下列範例示範如何使用簡單和使用者定義的實數值型別來進行裝箱。一個方塊，它是 `safe_cast` 原生堆疊上的實值型別變數，可以將它指派給垃圾收集堆積上的變數。

```

// safe_cast_boxing.cpp
// compile with: /clr
using namespace System;

interface struct I {};

value struct V : public I {
    int m_x;

    V(int i) : m_x(i) {}

};

int main() {
    // box a value type
    V v(100);
    I^ i = safe_cast<I^>(v);

    int x = 100;
    V^ refv = safe_cast<V^>(v);
    int^ refi = safe_cast<int^>(x);
}

```

下一個範例顯示，在作業中，在使用者定義的轉換上，該裝箱的優先順序高於 `safe_cast`。

```

// safe_cast_boxing_2.cpp
// compile with: /clr
static bool fRetval = true;

interface struct I {};
value struct V : public I {
    int x;

    V(int argx) {
        x = argx;
    }

    static operator I^(V v) {
        fRetval = false;
        I^ pi = v;
        return pi;
    }
};

ref struct R {
    R() {}
    R(V^ pv) {}
};

int main() {
    V v(10);
    I^ pv = safe_cast<I^>(v);    // boxing will occur, not UDC "operator I^"
}

```

Unbox 處理

取消裝箱會定義為編譯器插入的使用者定義轉換。因此，您可以使用 `safe_cast` 將 CLR 堆積上的值取消裝箱。

取消裝箱是使用者定義的轉換，但與裝箱不同的是，取消裝箱必須是明確的，也就是必須由 C 樣式的轉換來執行，`static_cast` 或者，無法以隱含方式 `safe_cast` 執行取消錄製。

```
// safe_cast_unboxing.cpp
// compile with: /clr
int main() {
    System::Object ^ o = 42;
    int x = safe_cast<int>(o);
}
```

下列範例會顯示具有實數值型別和基本類型的取消裝箱。

```
// safe_cast_unboxing_2.cpp
// compile with: /clr
using namespace System;

interface struct I {};

value struct VI : public I {};

void test1() {
    Object^ o = 5;
    int x = safe_cast<Int32>(o);
}

value struct V {
    int x;
    String^ s;
};

void test2() {
    V localv;
    Object^ o = localv;
    V unboxv = safe_cast<V>(o);
}

void test3() {
    V localv;
    V^ o2 = localv;
    V unboxv2 = safe_cast<V>(o2);
}

void test4() {
    I^ refi = VI();
    VI vi = safe_cast<VI>(refi);
}

int main() {
    test1();
    test2();
    test3();
    test4();
}
```

safe_cast 和泛型型別

下一個範例會示範如何使用 `safe_cast` 來執行具有泛型型別的向下轉換。

```
// safe_cast_generic_types.cpp
// compile with: /clr
interface struct I {};

generic<class T> where T:I
ref struct Base {
    T t;
    void test1() {}
};

generic<class T> where T:I
ref struct Derived:public Base <T> {};

ref struct R:public I {};

typedef Base<R^> GBase_R;
typedef Derived<R^> GDerived_R;

int main() {
    GBase_R^ br = gcnew GDerived_R();
    GDerived_R^ dr = safe_cast<GDerived_R^>(br);
}
```

另請參閱

[safe_cast](#)

規則運算式 (C++/CLI)

2019/12/2 • [Edit Online](#)

示範如何使用.NET Framework 中的規則運算式類別的各種字串作業。

下列主題示範如何使用.NET Framework [System.Text.RegularExpressions](#) 命名空間 (並在其中一個案例 [System.String.Split](#) 方法) 來搜尋、剖析和修改字串。

使用規則運算式剖析字串

下列程式碼範例示範簡單的字串剖析使用 [Regex](#) 類別中 [System.Text.RegularExpressions](#) 命名空間。會建構包含多種類型的文字分隔符號的字串。使用再剖析字串 [Regex](#) 類別搭配 [Match](#) 類別。然後，會個別顯示每個單字的句子。

範例

```
// regex_parse.cpp
// compile with: /clr
#using <system.dll>

using namespace System;
using namespace System::Text::.RegularExpressions;

int main( )
{
    int words = 0;
    String^ pattern = "[a-zA-Z]*";
    Console::WriteLine( "pattern : '{0}'", pattern );
    Regex^ regex = gcnew Regex( pattern );

    String^ line = "one\ttwo three:four,five six  seven";
    Console::WriteLine( "text : '{0}'", line );
    for( Match^ match = regex->Match( line );
         match->Success; match = match->NextMatch( ) )
    {
        if( match->Value->Length > 0 )
        {
            words++;
            Console::WriteLine( "{0}", match->Value );
        }
    }
    Console::WriteLine( "Number of Words : {0}", words );
}

return 0;
}
```

使用 Split 方法剖析字串

下列程式碼範例示範如何使用 [System.String.Split](#) 方法，以從字串擷取每個字。字串，包含多種類型的文字分隔符號是建構和剖析藉由呼叫 [Split](#) 分隔符號的清單。然後，會個別顯示每個單字的句子。

範例

```

// regex_split.cpp
// compile with: /clr
using namespace System;

int main()
{
    String^ delimStr = " .:\t";
    Console::WriteLine( "delimiter : '{0}'", delimStr );
    array<Char>^ delimiter = delimStr->ToCharArray();
    array<String>^ words;
    String^ line = "one\ttwo three:four,five six seven";

    Console::WriteLine( "text : '{0}'", line );
    words = line->Split( delimiter );
    Console::WriteLine( "Number of Words : {0}", words->Length );
    for (int word=0; word<words->Length; word++)
        Console::WriteLine( "{0}", words[word] );

    return 0;
}

```

使用規則運算式進行簡單對應

下列程式碼範例會使用規則運算式來尋找確切的子字串相符項目。搜尋由靜態`IsMatch`方法，這個方法會採用兩個字串做為輸入。第一個是要搜尋的字串和第二個是要搜尋的模式。

範例

```

// regex_simple.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace System::Text::RegularExpressions;

int main()
{
    array<String>^ sentence =
    {
        "cow over the moon",
        "Betsy the Cow",
        "cowering in the corner",
        "no match here"
    };

    String^ matchStr = "cow";
    for (int i=0; i<sentence->Length; i++)
    {
        Console::Write( "{0,24}", sentence[i] );
        if ( Regex::IsMatch( sentence[i], matchStr,
            RegexOptions::IgnoreCase ) )
            Console::WriteLine( "(match for '{0}' found)", matchStr );
        else
            Console::WriteLine("");
    }
    return 0;
}

```

使用規則運算式來擷取資料欄位

下列程式碼範例示範如何從格式化字串中擷取資料的規則運算式的使用。下列程式碼範例使用`Regex`類別，以指定的電子郵件地址對應的模式。這個模式會包含可用來擷取使用者和主機名稱部分，每個電子郵件地址的欄位識別

碼。[Match](#)類別用來執行實際的模式比對。如果指定的電子郵件地址是有效的會擷取並顯示的使用者名稱和主機名稱。

範例

```
// Regex_extract.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace System::Text::RegularExpressions;

int main()
{
    array<String^>^ address=
    {
        "jay@southridgevideo.com",
        "barry@adatum.com",
        "treyresearch.net",
        "karen@proseware.com"
    };

    Regex^ emailregex = gcnew Regex("(?<user>[^@]+)@(?:host>.+)");

    for (int i=0; i<address->Length; i++)
    {
        Match^ m = emailregex->Match( address[i] );
        Console::Write("\n{0,25}", address[i]);

        if ( m->Success )
        {
            Console::Write("    User='{0}'",
                m->Groups["user"]->Value);
            Console::Write("    Host='{0}'",
                m->Groups["host"]->Value);
        }
        else
            Console::Write("    (invalid email address)");
    }

    Console::WriteLine("");
    return 0;
}
```

使用規則運算式重新整理資料

下列程式碼範例示範如何使用.NET Framework 規則運算式支援，以重新排列或重新格式化資料。下列程式碼範例會使用[Regex](#)和[Match](#)類別以從字串中擷取第一個和最後一個名稱，並以反向順序顯示這些名稱項目。

[Regex](#)類別用來建構描述目前的資料格式的規則運算式。這兩個名稱會假設為以逗號分隔，而且可以使用任何數量的逗號周圍的空白。[Match](#)方法可用來分析每個字串。如果成功，會從擷取名字和姓氏[Match](#)物件，並顯示。

範例

```
// regex_reordered.cpp
// compile with: /clr
#using <System.dll>
using namespace System;
using namespace Text::RegularExpressions;

int main()
{
    array<String^>^ name =
    {
        "Abolrous, Sam",
        "Berg,Matt",
        "Berry , Jo",
        "www.contoso.com"
    };

    Regex^ reg = gcnew Regex("(?<last>\\w*)\\s*,\\s*(?<first>\\w*)");

    for ( int i=0; i < name->Length; i++ )
    {
        Console::Write( "{0,-20}", name[i] );
        Match^ m = reg->Match( name[i] );
        if ( m->Success )
        {
            String^ first = m->Groups["first"]->Value;
            String^ last = m->Groups["last"]->Value;
            Console::WriteLine("{0} {1}", first, last);
        }
        else
            Console::WriteLine("(invalid)");
    }
    return 0;
}
```

使用規則運算式進行搜尋和取代

下列程式碼範例示範如何規則運算式類別[Regex](#)可用來執行搜尋和取代。做法是使用[Replace](#)方法。使用的版本會採用兩個字串做為輸入：修改字串和要插入取代的區段（如果有的話）的字串，符合模式提供給[Regex](#)物件。

此程式碼會以底線()取代字串中的所有數字，然後再取代具有空字串，以便有效地移除它們。相同的效果，即可在單一步驟中，但兩個步驟這裡基於示範用途使用。

範例

```
// regex_replace.cpp
// compile with: /clr
#using <System.dll>
using namespace System::Text::RegularExpressions;
using namespace System;

int main()
{
    String^ before = "The q43uick bro254wn f00x ju4mped";
    Console::WriteLine("original : {0}", before);

    Regex^ digitRegex = gcnew Regex("(?<digit>[0-9])");
    String^ after = digitRegex->Replace(before, "_");
    Console::WriteLine("1st regex : {0}", after);

    Regex^ underbarRegex = gcnew Regex("_");
    String^ after2 = underbarRegex->Replace(after, "");
    Console::WriteLine("2nd regex : {0}", after2);

    return 0;
}
```

使用規則運算式驗證資料格式

下列程式碼範例示範如何使用規則運算式來驗證字串的格式。在下列程式碼範例中，字串應該包含有效的電話號碼。下列程式碼範例會使用字串"\d{3}-\d{3}-\d{4}"來指出每個欄位代表有效的電話號碼。在字串中的"d"表示數字，和"d"後面的引數指出必須要有的數字數目。在此情況下，數字，才能以連字號分隔。

範例

```
// regex_validate.cpp
// compile with: /clr
#using <System.dll>

using namespace System;
using namespace Text::RegularExpressions;

int main()
{
    array<String^>^ number =
    {
        "123-456-7890",
        "444-234-22450",
        "690-203-6578",
        "146-893-232",
        "146-839-2322",
        "4007-295-1111",
        "407-295-1111",
        "407-2-5555",
    };

    String^ regStr = "^\\d{3}-\\d{3}-\\d{4}$";

    for ( int i = 0; i < number->Length; i++ )
    {
        Console::Write( "{0,14}", number[i] );

        if ( Regex::IsMatch( number[i], regStr ) )
            Console::WriteLine(" - valid");
        else
            Console::WriteLine(" - invalid");
    }
    return 0;
}
```

相關章節

[.NET Framework 規則運算式](#)

另請參閱

[以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

檔案處理和 I/O (C++/CLI)

2020/11/2 • [Edit Online](#)

示範使用 .NET Framework 的各種檔案作業。

下列主題示範如何使用命名空間中定義的類別 [System.IO](#) 來執行各種檔案作業。

列舉目錄中的檔案

下列程式碼範例示範如何取出目錄中的檔案清單。此外，也會列舉子目錄。下列程式碼範例會使用 [GetFiles](#) [GetDirectories](#) 方法來顯示 C:\Windows 目錄的內容。

範例

```
// enum_files.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    String^ folder = "C:\\";
    array<String^>^ dir = Directory::GetDirectories( folder );
    Console::WriteLine("==== Directories inside '{0}' ====", folder);
    for ( int i=0; i<dir->Length; i++ )
        Console::WriteLine(dir[i]);

    array<String^>^ file = Directory::.GetFiles( folder );
    Console::WriteLine("==== Files inside '{0}' ====", folder);
    for ( int i=0; i<file->Length; i++ )
        Console::WriteLine(file[i]);

    return 0;
}
```

監視檔案系統變更

下列程式碼範例會使用 [FileSystemWatcher](#) 來註冊對應至所建立、變更、刪除或重新命名之檔案的事件。您可以使用 [FileSystemWatcher](#) 類別，在偵測到變更時引發事件，而不是定期輪詢目錄以變更檔案。

範例

```

// monitor_fs.cpp
// compile with: /clr
#using <system.dll>

using namespace System;
using namespace System::IO;

ref class FSEventHandler
{
public:
    void OnChanged (Object^ source, FileSystemEventArgs^ e)
    {
        Console::WriteLine("File: {0} {1}",
                           e->FullPath, e->ChangeType);
    }
    void OnRenamed(Object^ source, RenamedEventArgs^ e)
    {
        Console::WriteLine("File: {0} renamed to {1}",
                           e->OldFullPath, e->FullPath);
    }
};

int main()
{
    array<String^>^ args = Environment::GetCommandLineArgs();

    if(args->Length < 2)
    {
        Console::WriteLine("Usage: Watcher.exe <directory>");
        return -1;
    }

    FileSystemWatcher^ fsWatcher = gcnew FileSystemWatcher( );
    fsWatcher->Path = args[1];
    fsWatcher->NotifyFilter = static_cast<NotifyFilters>
        (NotifyFilters::FileName |
         NotifyFilters::Attributes |
         NotifyFilters::LastAccess |
         NotifyFilters::LastWrite |
         NotifyFilters::Security |
         NotifyFilters::Size );

    FSEventHandler^ handler = gcnew FSEventHandler();
    fsWatcher->Changed += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Created += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Deleted += gcnew FileSystemEventHandler(
        handler, &FSEventHandler::OnChanged);
    fsWatcher->Renamed += gcnew RenamedEventHandler(
        handler, &FSEventHandler::OnRenamed);

    fsWatcher->EnableRaisingEvents = true;

    Console::WriteLine("Press Enter to quit the sample.");
    Console::ReadLine( );
}

```

讀取二進位檔案

下列程式碼範例顯示如何使用命名空間中的兩個類別，從檔案讀取二進位資料 [System.IO : FileStream](#) 和 [BinaryReader](#)。[FileStream](#) 表示實際的檔案。[BinaryReader](#) 提供允許二進位存取之資料流程的介面。

程式碼範例會讀取名為 `data` 的檔案，並包含二進位格式的整數。如需這種檔案的相關資訊，請參閱 [如何: \(c + +/Cli 寫入二進位檔案\)](#)。

範例

```
// binary_read.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "data.bin";
    try
    {
        FileStream^ fs = gcnew FileStream(fileName, FileMode::Open);
        BinaryReader^ br = gcnew BinaryReader(fs);

        Console::WriteLine("contents of {0}:", fileName);
        while (br->BaseStream->Position < br->BaseStream->Length)
            Console::WriteLine(br->ReadInt32().ToString());

        fs->Close();
    }
    catch (Exception^ e)
    {
        if (dynamic_cast<FileNotFoundException^>(e))
            Console::WriteLine("File '{0}' not found", fileName);
        else
            Console::WriteLine("Exception: ({0})", e);
        return -1;
    }
    return 0;
}
```

讀取文字檔

下列程式碼範例示範如何使用 [StreamReader](#) 命名空間中定義的類別，一次開啟和讀取文字檔 [System.IO](#)。這個類別的實例是用來開啟文字檔，然後 [System.IO.StreamReader.ReadLine](#) 使用方法來取出每一行。

這個程式碼範例會讀取名為 `textfile.txt` 的檔案，並包含文字。如需這種檔案的相關資訊，請參閱 [如何：\(c + +/Cli 撰寫文字檔\)](#)。

範例

```
// text_read.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "textfile.txt";
    try
    {
        Console::WriteLine("trying to open file {0}...", fileName);
        StreamReader^ din = File::OpenText(fileName);

        String^ str;
        int count = 0;
        while ((str = din->ReadLine()) != nullptr)
        {
            count++;
            Console::WriteLine("line {0}: {1}", count, str );
        }
    }
    catch (Exception^ e)
    {
        if (dynamic_cast<FileNotFoundException^>(e))
            Console::WriteLine("file '{0}' not found", fileName);
        else
            Console::WriteLine("problem reading file '{0}'", fileName);
    }

    return 0;
}
```

取出檔案資訊

下列程式碼範例將示範 [FileInfo](#) 類別。當您擁有檔案的名稱時，您可以使用這個類別來取得檔案的相關資訊，例如檔案大小、目錄、完整名稱，以及建立的日期和時間，以及上次修改的日期和時間。

此程式碼會抓取 Notepad.exe 的檔案資訊。

範例

```
// file_info.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    array<String^>^ args = Environment::GetCommandLineArgs();
    if (args->Length < 2)
    {
        Console::WriteLine("\nUSAGE : file_info <filename>\n\n");
        return -1;
    }

    FileInfo^ fi = gcnew FileInfo( args[1] );

    Console::WriteLine("file size: {0}", fi->Length );

    Console::Write("File creation date:  ");
    Console::Write(fi->CreationTime.Month.ToString());
    Console::Write(".{0}", fi->CreationTime.Day.ToString());
    Console::WriteLine(".{0}", fi->CreationTime.Year.ToString());

    Console::Write("Last access date:  ");
    Console::Write(fi->>LastAccessTime.Month.ToString());
    Console::Write(".{0}", fi->>LastAccessTime.Day.ToString());
    Console::WriteLine(".{0}", fi->>LastAccessTime.Year.ToString());

    return 0;
}
```

寫入二進位檔案

下列程式碼範例示範如何將二進位資料寫入檔案。使用命名空間的兩個類別 [System.IO : FileStream](#) 和 [BinaryWriter](#)。[FileStream](#) 表示實際的檔案，並 [BinaryWriter](#) 提供允許二進位存取之資料流程的介面。

下列程式碼範例會寫入檔案，其中包含二進位格式的整數。您可以使用 [如何:讀取二進位檔案 \(c + +/cli\)](#) 中的程式碼來讀取此檔案。

範例

```
// binary_write.cpp
// compile with: /clr
#using<system.dll>
using namespace System;
using namespace System::IO;

int main()
{
    array<Int32>^ data = {1, 2, 3, 10000};

    FileStream^ fs = gcnew FileStream("data.bin", FileMode::Create);
    BinaryWriter^ w = gcnew BinaryWriter(fs);

    try
    {
        Console::WriteLine("writing data to file:");
        for (int i=0; i<data->Length; i++)
        {
            Console::WriteLine(data[i]);
            w->Write(data[i]);
        }
    }
    catch (Exception^)
    {
        Console::WriteLine("data could not be written");
        fs->Close();
        return -1;
    }

    fs->Close();
    return 0;
}
```

寫入文字檔

下列程式碼範例示範如何建立文字檔，並使用在 [StreamWriter](#) 命名空間中定義的類別，將文字寫入其中 [System.IO](#)。此函式 [StreamWriter](#) 會使用要建立的檔案名。如果檔案存在，則會覆寫 (除非您傳遞 `True` 做為第二個函式 [StringWriter](#) 引數)。

然後，會使用和函式來將檔案歸檔 [Write](#) [WriteLine](#)。

範例

```
// text_write.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

int main()
{
    String^ fileName = "textfile.txt";

    StreamWriter^ sw = gcnew StreamWriter(fileName);
    sw->WriteLine("A text file is born!");
    sw->Write("You can use WriteLine");
    sw->WriteLine("...or just Write");
    sw->WriteLine("and do {0} output too.", "formatted");
    sw->WriteLine("You can also send non-text objects:");
    sw->WriteLine(DateTime::Now);
    sw->Close();
    Console::WriteLine("a new file ('{0}') has been written", fileName);

    return 0;
}
```

另請參閱

[使用 c + +/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

[檔案和資料流程 i/o](#)

[System.IO 命名空間](#)

圖形作業 (C++/CLI)

2019/12/2 • [Edit Online](#)

示範如何使用 Windows SDK 的影像操作。

下列主題示範如何使用[System.Drawing.Image](#)執行影像操作的類別。

使用.NET Framework 顯示影像

下列程式碼範例會修改 OnPaint 事件處理常式，來擷取變數的指標，[Graphics](#)主要表單的物件。[OnPaint](#)函式適用於 Windows Forms 應用程式，很可能使用精靈建立的 Visual Studio 應用程式。

表示影像[Image](#)類別。映像載入的資料會從jpg 檔案使用[System.Drawing.Image.FromFile](#)方法。繪製至表單的影像之前，表單會調整大小以容納影像。繪製影像利用[System.Drawing.Graphics.DrawImage](#)方法。

[Graphics](#)並[Image](#)類別都在[System.Drawing](#)命名空間。

範例

```
#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;

protected:
virtual Void Form1::OnPaint(PaintEventArgs^ pe) override
{
    Graphics^ g = pe->Graphics;
    Image^ image = Image::FromFile("SampleImage.jpg");
    Form::ClientSize = image->Size;
    g->DrawImage( image, 0, 0, image->Size.Width, image->Size.Height );
}
```

使用.NET Framework 繪製圖案

下列程式碼範例會使用[Graphics](#)類別，以修改[OnPaint](#)事件處理常式來擷取變數的指標，[Graphics](#)主要表單的物件。設定表單的背景色彩來繪製線條和弧形使用再使用此指標[System.Drawing.Graphics.DrawLine](#)和[DrawArc](#)方法。

範例

```

#using <system.drawing.dll>
using namespace System;
using namespace System::Drawing;
// ...
protected:
virtual Void Form1::OnPaint(PaintEventArgs^ pe ) override
{
    Graphics^ g = pe->Graphics;
    g->Clear(Color::AntiqueWhite);

    Rectangle rect = Form::ClientRectangle;
    Rectangle smallRect;
    smallRect.X = rect.X + rect.Width / 4;
    smallRect.Y = rect.Y + rect.Height / 4;
    smallRect.Width = rect.Width / 2;
    smallRect.Height = rect.Height / 2;

    Pen^ redPen = gcnew Pen(Color::Red);
    redPen->Width = 4;
    g->DrawLine(redPen, 0, 0, rect.Width, rect.Height);

    Pen^ bluePen = gcnew Pen(Color::Blue);
    bluePen->Width = 10;
    g->DrawArc( bluePen, smallRect, 90, 270 );
}

```

使用.NET Framework 旋轉影像

下列程式碼範例示範使用[System.Drawing.Image](#)從磁碟載入影像、旋轉 90 度，並將它儲存為新的.jpg 檔案的類別。

範例

```

#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;

int main()
{
    Image^ image = Image::FromFile("SampleImage.jpg");
    image->RotateFlip( RotateFlipType::Rotate90FlipNone );
    image->Save("SampleImage_rotated.jpg");
    return 0;
}

```

轉換影像檔案格式，使用.NET Framework

下列程式碼範例示範[System.Drawing.Image](#)類別和[System.Drawing.Imaging.ImageFormat](#)列舉，用來轉換和儲存影像檔。下列程式碼會從.jpg 檔案載入影像，並再將它儲存在.gif 和.bmp 檔案格式。

範例

```
#using <system.drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Drawing::Imaging;

int main()
{
    Image^ image = Image::FromFile("SampleImage.jpg");
    image->Save("SampleImage.png", ImageFormat::Png);
    image->Save("SampleImage.bmp", ImageFormat::Bmp);

    return 0;
}
```

相關章節

[圖形程式設計入門](#)

[關於 GDI+ Managed 程式碼](#)

另請參閱

[以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

[System.Drawing](#)

Windows 作業 (C++/CLI)

2020/11/2 • [Edit Online](#)

使用 Windows SDK 示範各種 Windows 特定的工作。

下列主題示範使用 Visual C++ 搭配 Windows SDK 執行的各種 Windows 作業。

判斷關機是否已啟動

下列程式碼範例示範如何判斷應用程式或 .NET Framework 目前是否正在終止。這有助於存取 .NET Framework 中的靜態元素，因為在關機期間，這些結構會由系統完成，而且無法可靠地使用。藉由 [HasShutdownStarted](#) 先檢查屬性，您就可以不存取這些元素來避免可能的失敗。

範例

```
// check_shutdown.cpp
// compile with: /clr
using namespace System;
int main()
{
    if (Environment::HasShutdownStarted)
        Console::WriteLine("Shutting down.");
    else
        Console::WriteLine("Not shutting down.");
    return 0;
}
```

判斷使用者互動狀態

下列程式碼範例示範如何判斷程式碼是否正在使用者互動的內容中執行。如果 [UserInteractive](#) 為 false，則程式碼會以服務進程的形式執行，或從 Web 應用程式內部執行，在這種情況下，您不應該嘗試與使用者互動。

範例

```
// user_interactive.cpp
// compile with: /clr
using namespace System;

int main()
{
    if ( Environment::UserInteractive )
        Console::WriteLine("User interactive");
    else
        Console::WriteLine("Noninteractive");
    return 0;
}
```

從 Windows 登錄讀取資料

下列程式碼範例會使用 [CurrentUser](#) 金鑰來讀取 Windows 登錄中的資料。首先，系統會使用方法來列舉子機碼 [GetSubKeyNames](#)，然後使用方法開啟身分識別子機碼 [OpenSubKey](#)。和根金鑰一樣，每個子機碼都是由 [RegistryKey](#) 類別表示。最後，新的 [RegistryKey](#) 物件會用來列舉索引鍵/值組。

範例

```
// registry_read.cpp
// compile with: /clr
using namespace System;
using namespace Microsoft::Win32;

int main( )
{
    array<String^>^ key = Registry::CurrentUser->GetSubKeyNames( );

    Console::WriteLine("Subkeys within CurrentUser root key:");
    for (int i=0; i<key->Length; i++)
    {
        Console::WriteLine("    {0}", key[i]);
    }

    Console::WriteLine("Opening subkey 'Identities'...");
    RegistryKey^ rk = nullptr;
    rk = Registry::CurrentUser->OpenSubKey("Identities");
    if (rk==nullptr)
    {
        Console::WriteLine("Registry key not found - aborting");
        return -1;
    }

    Console::WriteLine("Key/value pairs within 'Identities' key:");
    array<String^>^ name = rk->GetValueNames( );
    for (int i=0; i<name->Length; i++)
    {
        String^ value = rk->GetValue(name[i])->ToString();
        Console::WriteLine("    {0} = {1}", name[i], value);
    }

    return 0;
}
```

備註

Registry 類別只是靜態實例的容器 RegistryKey。每個實例都代表一個根登錄節點。實例為 ClassesRoot 、 CurrentConfig CurrentUser 、 LocalMachine 和 Users 。

除了為靜態，類別內的物件 Registry 也是唯讀的。此外， RegistryKey 為了存取登錄物件的內容而建立的類別實例也是唯讀的。如需如何覆寫此行為的範例，請參閱 [如何：將資料寫入 Windows 登錄 \(c++/cli\)](#) 。

類別中有兩個額外的物件 Registry : DynData 和 PerformanceData 。兩者都是類別的實例 RegistryKey 。此 DynData 物件包含動態登錄資訊，只有在 Windows 98 和 Windows Me 中才支援。 PerformanceData 物件可以用來存取使用 Windows 效能監視系統之應用程式的效能計數器資訊。 PerformanceData 節點代表未實際儲存在登錄中，因此無法使用 Regedit.exe 來查看的資訊。

讀取 Windows 效能計數器

某些應用程式和 Windows 子系統會透過 Windows 效能系統公開效能資料。您可以使用 PerformanceCounterCategory PerformanceCounter 位於命名空間中的和類別來存取這些計數器 System.Diagnostics 。

下列程式碼範例會使用這些類別來取出並顯示由 Windows 更新的計數器，以指出處理器忙碌的時間百分比。

NOTE

這個範例需要系統管理權限才能在 Windows Vista 上執行。

範例

```

// processor_timer.cpp
// compile with: /clr
#pragma once

using namespace System;
using namespace System::Threading;
using namespace System::Diagnostics;
using namespace System::Timers;

ref struct TimerObject
{
public:
    static String^ m_instanceName;
    static PerformanceCounter^ m_theCounter;

public:
    static void OnTimer(Object^ source, ElapsedEventArgs^ e)
    {
        try
        {
            Console::WriteLine("CPU time used: {0,6} ",
                m_theCounter->NextValue().ToString("f"));
        }
        catch(Exception^ e)
        {
            if (dynamic_cast<InvalidOperationException^>(e))
            {
                Console::WriteLine("Instance '{0}' does not exist",
                    m_instanceName);
                return;
            }
            else
            {
                Console::WriteLine("Unknown exception... ('q' to quit)");
                return;
            }
        }
    }
};

int main()
{
    String^ objectName = "Processor";
    String^ counterName = "% Processor Time";
    String^ instanceName = "_Total";

    try
    {
        if ( !PerformanceCounterCategory::Exists(objectName) )
        {
            Console::WriteLine("Object {0} does not exist", objectName);
            return -1;
        }
    }
    catch (UnauthorizedAccessException ^ex)
    {
        Console::WriteLine("You are not authorized to access this information.");
        Console::Write("If you are using Windows Vista, run the application with ");
        Console::WriteLine("administrative privileges.");
        Console::WriteLine(ex->Message);
        return -1;
    }

    if ( !PerformanceCounterCategory::CounterExists(
        counterName, objectName) )
    {
        Console::WriteLine("Counter {0} does not exist", counterName);
        return -1;
    }
}

```

```

}

TimerObject::m_instanceName = instanceName;
TimerObject::m_theCounter = gcnew PerformanceCounter(
    objectName, counterName, instanceName);

System::Timers::Timer^ aTimer = gcnew System::Timers::Timer();
aTimer->Elapsed += gcnew ElapsedEventHandler(&TimerObject::OnTimer);
aTimer->Interval = 1000;
aTimer->Enabled = true;
aTimer->AutoReset = true;

Console::WriteLine("reporting CPU usage for the next 10 seconds");
Thread::Sleep(10000);
return 0;
}

```

從剪貼簿取出文字

下列程式碼範例會使用 [GetDataObject](#) 成員函式，將指標傳回至 [IDataObject](#) 介面。然後可以查詢此介面，以取得資料的格式，並用來取得實際的資料。

範例

```

// read_clipboard.cpp
// compile with: /clr
#using <system.dll>
#using <system.Drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;

[STAThread] int main( )
{
    IDataObject^ data = Clipboard::GetDataObject( );

    if (data)
    {
        if (data->GetDataPresent(DataFormats::Text))
        {
            String^ text = static_cast<String^>
                (data->GetData(DataFormats::Text));
            Console::WriteLine(text);
        }
        else
            Console::WriteLine("Nontext data is in the Clipboard.");
    }
    else
        Console::WriteLine("No data was found in the Clipboard.");
}

return 0;
}

```

取出目前的使用者名稱

下列程式碼範例將示範如何抓取目前的使用者名稱，(登入 Windows) 的使用者名稱。名稱會儲存在 [UserName](#) 命名空間中定義的字串中 [Environment](#)。

範例

```
// username.cpp
// compile with: /clr
using namespace System;

int main()
{
    Console::WriteLine("\nCurrent user: {0}", Environment::UserName);
    return 0;
}
```

取出 .NET Framework 版本

下列程式碼範例示範如何使用屬性來判斷目前安裝 .NET Framework 的版本 [Version](#)，這是 [Version](#) 包含版本資訊之物件的指標。

範例

```
// dotnet_ver.cpp
// compile with: /clr
using namespace System;
int main()
{
    Version^ version = Environment::Version;
    if (version)
    {
        int build = version->Build;
        int major = version->Major;
        int minor = version->Minor;
        int revision = Environment::Version->Revision;
        Console::Write(".NET Framework version: ");
        Console::WriteLine("{0}.{1}.{2}.{3}",
                           build, major, minor, revision);
    }
    return 0;
}
```

取出本機電腦名稱稱

下列程式碼範例將示範如何抓取本機電腦名稱稱，(在網路) 上顯示的電腦名稱稱。您可以藉由取得在 [MachineName](#) 命名空間中定義的字串來完成此動作 [Environment](#)。

範例

```
// machine_name.cpp
// compile with: /clr
using namespace System;

int main()
{
    Console::WriteLine("\nMachineName: {0}", Environment::MachineName);
    return 0;
}
```

取出 Windows 版本

下列程式碼範例示範如何取出目前作業系統的平臺和版本資訊。這項資訊會儲存在 [System.Environment.OSVersion](#) 屬性中，並包含一個列舉，描述廣泛詞彙的 Windows 版本，以及 [Version](#) 包合作業系統確切組建的物件。

範例

```
// os_ver.cpp
// compile with: /clr
using namespace System;

int main()
{
    OperatingSystem^ osv = Environment::OSVersion;
    PlatformID id = osv->Platform;
    Console::Write("Operating system: ");

    if (id == PlatformID::Win32NT)
        Console::WriteLine("Win32NT");
    else if (id == PlatformID::Win32S)
        Console::WriteLine("Win32S");
    else if (id == PlatformID::Win32Windows)
        Console::WriteLine("Win32Windows");
    else
        Console::WriteLine("WinCE");

    Version^ version = osv->Version;
    if (version)
    {
        int build = version->Build;
        int major = version->Major;
        int minor = version->Minor;
        int revision = Environment::Version->Revision;
        Console::Write("OS Version: ");
        Console::WriteLine("{0}.{1}.{2}.{3}",
                           build, major, minor, revision);
    }
}

return 0;
}
```

從啟動以來取出經過的時間

下列程式碼範例示範如何判斷滴答計數，或自啟動 Windows 以來經過的毫秒數。此值會儲存在 [System.Environment.TickCount](#) 成員中，而且因為它是32位的值，所以大約每隔24.9 天重設為零。

範例

```
// startup_time.cpp
// compile with: /clr
using namespace System;

int main( )
{
    Int32 tc = Environment::TickCount;
    Int32 seconds = tc / 1000;
    Int32 minutes = seconds / 60;
    float hours = static_cast<float>(minutes) / 60;
    float days = hours / 24;

    Console::WriteLine("Milliseconds since startup: {0}", tc);
    Console::WriteLine("Seconds since startup: {0}", seconds);
    Console::WriteLine("Minutes since startup: {0}", minutes);
    Console::WriteLine("Hours since startup: {0}", hours);
    Console::WriteLine("Days since startup: {0}", days);

    return 0;
}
```

儲存剪貼簿中的文字

下列程式碼範例使用 [Clipboard](#) 命名空間中定義的物件 [System.Windows.Forms](#) 來儲存字串。這個物件會提供兩個成員函式：[SetDataObject](#) 和 [GetDataObject](#)。資料會藉由傳送任何衍生自的物件，儲存在剪貼簿中 [Object](#) [SetDataObject](#)。

範例

```
// store_clipboard.cpp
// compile with: /clr
#using <System.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

[STAThread] int main()
{
    String^ str = "This text is copied into the Clipboard.";

    // Use 'true' as the second argument if
    // the data is to remain in the clipboard
    // after the program terminates.
    Clipboard::SetDataObject(str, true);

    Console::WriteLine("Added text to the Clipboard.");

    return 0;
}
```

將資料寫入 Windows 登錄

下列程式碼範例會使用 [CurrentUser](#) 金鑰來建立 [RegistryKey](#) 對應至 [軟體](#) 金鑰之類別的可寫入實例。[CreateSubKey](#) 接著會使用方法來建立新的索引鍵，並將其新增至索引鍵/值組。

範例

```
// registry_write.cpp
// compile with: /clr
using namespace System;
using namespace Microsoft::Win32;

int main()
{
    // The second OpenSubKey argument indicates that
    // the subkey should be writable.
    RegistryKey^ rk;
    rk = Registry::CurrentUser->OpenSubKey("Software", true);
    if (!rk)
    {
        Console::WriteLine("Failed to open CurrentUser/Software key");
        return -1;
    }

    RegistryKey^ nk = rk->CreateSubKey("NewRegKey");
    if (!nk)
    {
        Console::WriteLine("Failed to create 'NewRegKey'");
        return -1;
    }

    String^ newValue = "NewValue";
    try
    {
        nk->SetValue("NewKey", newValue);
        nk->SetValue("NewKey2", 44);
    }
    catch (Exception^)
    {
        Console::WriteLine("Failed to set new values in 'NewRegKey'");
        return -1;
    }

    Console::WriteLine("New key created.");
    Console::Write("Use REGEDIT.EXE to verify ");
    Console::WriteLine("'"CURRENTUSER/Software/NewRegKey'\n");
    return 0;
}
```

備註

您可以使用 .NET Framework 來存取具有 [Registry](#) 和 [RegistryKey](#) 類別的登錄，這些類別都是在 [Microsoft.Win32](#) 命名空間中定義。登錄 類別是 類別靜態實例的容器 [RegistryKey](#)。每個實例都代表一個根登錄節點。實例為 [ClassesRoot](#)、[CurrentUser](#)、[LocalMachine](#) 和 [Users](#)。

相關章節

[Environment](#)

另請參閱

[使用 c++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

使用 ADO.NET 進行資料存取 (C++/CLI)

2020/11/2 • [Edit Online](#)

ADO.NET 是資料存取所需的 .NET Framework API，可讓先前的資料存取解決方案更容易使用並不相符。本節說明 Visual C++ 使用者特有的 ADO.NET，例如封送處理原生類型的一些問題。

ADO.NET 會在 Common Language Runtime (CLR) 下執行。因此，與 ADO.NET 互動的任何應用程式也必須將目標設為 CLR。不過，這並不表示原生應用程式無法使用 ADO.NET。這些範例將示範如何從機器碼與 ADO.NET 資料庫互動。

封送處理 ADO.NET 的 ANSI 字串

示範如何將原生字串 (`char *`) 加入至資料庫，以及如何將資料庫的封送處理 `System.String` 至原生字串。

範例

在此範例中，會建立類別 `DatabaseClass` 來與 ADO.NET 物件互動 `DataTable`。請注意，這個類別是原生 c++ `class` (相較于 `ref class` 或 `value class`)。這是必要的，因為我們想要從機器碼使用這個類別，而且您無法在機器碼中使用 managed 類型。這個類別會編譯成以 CLR 為目標，如同在類別宣告前面的指示詞所表示 `#pragma managed`。如需此指示詞的詳細資訊，請參閱[managed、非受控](#)。

請注意 `DatabaseClass` 類別的私用成員：`gcroot<DataTable ^> table`。由於原生類型不能包含 managed 類型，因此 `gcroot` 必須要有關鍵詞。如需的詳細資訊 `gcroot`，請參閱[如何：在原生類型中宣告控制碼](#)。

本範例中的其餘程式碼是原生 c++ 程式碼，如前面的指示詞所表示 `#pragma unmanaged main`。在此範例中，我們會建立 `DatabaseClass` 的新實例，並呼叫其方法來建立資料表，並在資料表中填入部分資料列。請注意，原生 c++ 字串會當做資料庫資料行 `StringCol` 的值來傳遞。在 `DatabaseClass` 內，這些字串會使用命名空間中找到的封送處理功能封送處理至 managed 字串 `System.Runtime.InteropServices`。具體而言，方法 `PtrToStringAnsi` 是用來將封送處理 `char *` 至 `String`，而方法 `StringToHGlobalAnsi` 則是用來將封送處理 `String` 至 `char *`。

NOTE

所配置的記憶體 `StringToHGlobalAnsi` 必須藉由呼叫或解除 `FreeHGlobal` 分配 `GlobalFree`。

```
// adonet_marshal_string_native.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(char *stringColValue)
    {
        table->Rows->Add(stringColValue);
    }
};
```

```

    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["StringCol"] = Marshal::PtrToStringAnsi(
            (IntPtr)stringValue);
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("StringCol",
            Type::GetType("System.String"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(char *dataColumn, char **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringAnsi(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed string
            // to a char *.
            values[i] = (char *)Marshal::StringToHGlobalAnsi(
                (String ^)rows[i][columnStr]).ToPointer();
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();
    db->AddRow("This is string 1.");
    db->AddRow("This is string 2.");

    // Now retrieve the rows and display their contents.
    char *values[MAXCOLS];
    int len = db->GetValuesForColumn(
        "StringCol", values, MAXCOLS);
    for (int i = 0; i < len; i++)
    {
        cout << "StringCol: " << values[i] << endl;

        // Deallocate the memory allocated using
        // Marshal::StringToHGlobalAnsi.
        GlobalFree(values[i]);
    }
}

```

```
    delete db;  
  
    return 0;  
}
```

```
StringCol: This is string 1.  
StringCol: This is string 2.
```

編譯程式碼

- 若要從命令列編譯器代碼，請將程式碼範例儲存在名為 `adonet_marshal_string_native` 的檔案中，然後輸入下列語句：

```
cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshal_string_native.cpp
```

封送處理 ADO.NET 的 BSTR 字串

示範如何將 COM 字串()加入 `BSTR` 至資料庫，以及如何將資料庫的封送處理 `System.String` 至 `BSTR`。

範例

在此範例中，會建立類別 `DatabaseClass` 來與 ADO.NET 物件互動 `DataTable`。請注意，這個類別是原生 c++ `class` (相較于 `ref class` 或 `value class`)。這是必要的，因為我們想要從機器碼使用這個類別，而且您無法在機器碼中使用 managed 類型。這個類別會編譯成以 CLR 為目標，如同在類別宣告前面的指示詞所表示 `#pragma managed`。如需此指示詞的詳細資訊，請參閱[managed、非受控](#)。

請注意 `DatabaseClass` 類別的私用成員：`gcroot<DataTable ^> table`。由於原生類型不能包含 managed 類型，因此 `gcroot` 必須要有關鍵詞。如需的詳細資訊 `gcroot`，請參閱[如何：在原生類型中宣告控制碼](#)。

本範例中的其餘程式碼是原生 c++ 程式碼，如前面的指示詞所表示 `#pragma unmanaged main`。在此範例中，我們會建立 `DatabaseClass` 的新實例，並呼叫其方法來建立資料表，並在資料表中填入部分資料列。請注意，COM 字串會當做資料庫資料行 `StringCol` 的值來傳遞。在 `DatabaseClass` 內，這些字串會使用命名空間中找到的封送處理功能封送處理至 managed 字串 `System.Runtime.InteropServices`。具體而言，方法 `PtrToStringBSTR` 是用來將封送處理 `BSTR` 至 `String`，而方法 `StringToBSTR` 則是用來將封送處理 `String` 至 `BSTR`。

NOTE

所配置的記憶體 `StringToBSTR` 必須藉由呼叫或解除 `FreeBSTR` 分配 `SysFreeString`。

```
// adonet_marshal_string_bstr.cpp  
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll  
#include <comdef.h>  
#include <gcroot.h>  
#include <iostream>  
using namespace std;  
  
#using <System.Data.dll>  
using namespace System;  
using namespace System::Data;  
using namespace System::Runtime::InteropServices;  
  
#define MAXCOLS 100  
  
#pragma managed  
class DatabaseClass  
{  
public:
```

```

DatabaseClass() : table(nullptr) { }

void AddRow(BSTR stringColValue)
{
    // Add a row to the table.
    DataRow ^row = table->NewRow();
    row["StringCol"] = Marshal::PtrToStringBSTR(
        (IntPtr)stringColValue);
    table->Rows->Add(row);
}

void CreateAndPopulateTable()
{
    // Create a simple DataTable.
    table = gcnew DataTable("SampleTable");

    // Add a column of type String to the table.
    DataColumn ^column1 = gcnew DataColumn("StringCol",
        Type::GetType("System.String"));
    table->Columns->Add(column1);
}

int GetValuesForColumn(BSTR dataColumn, BSTR *values,
    int valuesLength)
{
    // Marshal the name of the column to a managed
    // String.
    String ^columnStr = Marshal::PtrToStringBSTR(
        (IntPtr)dataColumn);

    // Get all rows in the table.
    array<DataRow ^> ^rows = table->Select();
    int len = rows->Length;
    len = (len > valuesLength) ? valuesLength : len;
    for (int i = 0; i < len; i++)
    {
        // Marshal each column value from a managed string
        // to a BSTR.
        values[i] = (BSTR)Marshal::StringToBSTR(
            (String ^)rows[i][columnStr]).ToPointer();
    }

    return len;
}

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();

    BSTR str1 = SysAllocString(L"This is string 1.");
    db->AddRow(str1);

    BSTR str2 = SysAllocString(L"This is string 2.");
    db->AddRow(str2);

    // Now retrieve the rows and display their contents.
    BSTR values[MAXCOLS];
    BSTR str3 = SysAllocString(L"StringCol");
    int len = db->GetValuesForColumn(
        str3, values, MAXCOLS);
}

```

```

for (int i = 0; i < len; i++)
{
    wcout << "StringCol: " << values[i] << endl;

    // Deallocate the memory allocated using
    // Marshal::StringToBSTR.
    SysFreeString(values[i]);
}

SysFreeString(str1);
SysFreeString(str2);
SysFreeString(str3);
delete db;

return 0;
}

```

```

StringCol: This is string 1.
StringCol: This is string 2.

```

編譯程式碼

- 若要從命令列編譯器代碼，請將程式碼範例儲存在名為 `adonet_marshal_string_native` 的檔案中，然後輸入下列語句：

```

cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshal_string_native.cpp

```

封送處理 ADO.NET 的 Unicode 字串

示範如何將原生 Unicode 字串 (`wchar_t *`) 加入至資料庫，以及如何將資料庫的封送處理 `System.String` 至原生 unicode 字串。

範例

在此範例中，會建立類別 `DatabaseClass` 來與 ADO.NET 物件互動 `DataTable`。請注意，這個類別是原生 c++ `class` (相較于 `ref class` 或 `value class`)。這是必要的，因為我們想要從機器碼使用這個類別，而且您無法在機器碼中使用 managed 類型。這個類別會編譯成以 CLR 為目標，如同在類別宣告前面的指示詞所表示 `#pragma managed`。如需此指示詞的詳細資訊，請參閱[managed、非受控](#)。

請注意 `DatabaseClass` 類別的私用成員：`gcroot<DataTable ^> table`。由於原生類型不能包含 managed 類型，因此 `gcroot` 必須要有關鍵詞。如需的詳細資訊 `gcroot`，請參閱[如何：在原生類型中宣告控制碼](#)。

本範例中的其餘程式碼是原生 c++ 程式碼，如前面的指示詞所表示 `#pragma unmanaged main`。在此範例中，我們會建立 `DatabaseClass` 的新實例，並呼叫其方法來建立資料表，並在資料表中填入部分資料列。請注意，Unicode c++ 字串會當做資料庫資料行 `StringCol` 的值傳遞。在 `DatabaseClass` 內，這些字串會使用命名空間中找到的封送處理功能封送處理至 managed 字串 `System.Runtime.InteropServices`。具體而言，方法 `PtrToStringUni` 是用來將封送處理 `wchar_t *` 至 `String`，而方法 `StringToHGlobalUni` 則是用來將封送處理 `String` 至 `wchar_t *`。

NOTE

所配置的記憶體 `StringToHGlobalUni` 必須藉由呼叫或解除 `FreeHGlobal` 分配 `GlobalFree`。

```

// adonet_marshal_string_wide.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>

```

```

using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(wchar_t *stringColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["StringCol"] = Marshal::PtrToStringUni(
            (IntPtr)stringColValue);
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("StringCol",
            Type::GetType("System.String"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, wchar_t **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed string
            // to a wchar_t *.
            values[i] = (wchar_t *)Marshal::StringToHGlobalUni(
                (String ^)rows[i][columnStr]).ToPointer();
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
}

```

```

db->CreateAndPopulateTable();
db->AddRow(L"This is string 1.");
db->AddRow(L"This is string 2.");

// Now retrieve the rows and display their contents.
wchar_t *values[MAXCOLS];
int len = db->GetValuesForColumn(
    L"StringCol", values, MAXCOLS);
for (int i = 0; i < len; i++)
{
    wcout << "StringCol: " << values[i] << endl;

    // Deallocate the memory allocated using
    // Marshal::StringToHGlobalUni.
    GlobalFree(values[i]);
}

delete db;

return 0;
}

```

```

StringCol: This is string 1.
StringCol: This is string 2.

```

編譯程式碼

- 若要從命令列編譯器代碼，請將程式碼範例儲存在名為 `adonet_marshal_string_wide` 的檔案中，然後輸入下列語句：

```

cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshal_string_wide.cpp

```

封送處理 ADO.NET 的 VARIANT

示範如何將原生新增 `VARIANT` 至資料庫，以及如何將資料庫的封送處理 `System.Object` 至原生 `VARIANT`。

範例

在此範例中，會建立類別 `DatabaseClass` 來與 ADO.NET 物件互動 `DataTable`。請注意，這個類別是原生 c++ `class` (相較于 `ref class` 或 `value class`)。這是必要的，因為我們想要從機器碼使用這個類別，而且您無法在機器碼中使用 managed 類型。這個類別會編譯成以 CLR 為目標，如同在類別宣告前面的指示詞所表示 `#pragma managed`。如需此指示詞的詳細資訊，請參閱[managed、非受控](#)。

請注意 `DatabaseClass` 類別的私用成員：`gcroot<DataTable ^> table`。由於原生類型不能包含 managed 類型，因此 `gcroot` 必須要有關鍵詞。如需的詳細資訊 `gcroot`，請參閱[如何：在原生類型中宣告控制碼](#)。

本範例中的其餘程式碼是原生 c++ 程式碼，如前面的指示詞所表示 `#pragma unmanaged main`。在此範例中，我們會建立 `DatabaseClass` 的新實例，並呼叫其方法來建立資料表，並在資料表中填入部分資料列。請注意，原生 `VARIANT` 類型會當做資料庫資料行 `ObjectCol` 的值來傳遞。在 `DatabaseClass` 內，`VARIANT` 會使用命名空間中的封送處理功能，將這些類型封送處理至 managed 物件 `System.Runtime.InteropServices`。具體而言，方法 `GetObjectForNativeVariant` 是用來將封送處理 `VARIANT` 至 `Object`，而方法 `GetNativeVariantForObject` 則是用來將封送處理 `Object` 至 `VARIANT`。

```

// adonet_marshal_variant.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

```

```

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(VARIANT *objectColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        row["ObjectCol"] = Marshal::GetObjectForNativeVariant(
            IntPtr(objectColValue));
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("ObjectCol",
            Type::GetType("System.Object"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, VARIANT *values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed object
            // to a VARIANT.
            Marshal::GetNativeVariantForObject(
                rows[i][columnStr], IntPtr(&values[i]));
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();
}

```

```

BSTR bstr1 = SysAllocString(L"This is a BSTR in a VARIANT.");
VARIANT v1;
v1.vt = VT_BSTR;
v1.bstrVal = bstr1;
db->AddRow(&v1);

int i = 42;
VARIANT v2;
v2.vt = VT_I4;
v2.lVal = i;
db->AddRow(&v2);

// Now retrieve the rows and display their contents.
VARIANT values[MAXCOLS];
int len = db->GetValuesForColumn(
    L"ObjectCol", values, MAXCOLS);
for (int i = 0; i < len; i++)
{
    switch (values[i].vt)
    {
        case VT_BSTR:
            wcout << L"ObjectCol: " << values[i].bstrVal << endl;
            break;
        case VT_I4:
            cout << "ObjectCol: " << values[i].lVal << endl;
            break;
        default:
            break;
    }
}

SysFreeString(bstr1);
delete db;

return 0;
}

```

```

ObjectCol: This is a BSTR in a VARIANT.
ObjectCol: 42

```

編譯程式碼

- 若要從命令列編譯器代碼，請將程式碼範例儲存在名為 `adonet_marshall_variant` 的檔案中，然後輸入下列語句：

```

cl /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshall_variant.cpp

```

封送處理 ADO.NET 的 SAFEARRAY

示範如何將原生新增 `SAFEARRAY` 至資料庫，以及如何將 managed 陣列從資料庫封送處理至原生 `SAFEARRAY`。

範例

在此範例中，會建立類別 `DatabaseClass` 來與 ADO.NET 物件互動 `DataTable`。請注意，這個類別是原生 C++ `class` (相較于 `ref class` 或 `value class`)。這是必要的，因為我們想要從機器碼使用這個類別，而且您無法在機器碼中使用 managed 類型。這個類別會編譯成以 CLR 為目標，如同在類別宣告前面的指示詞所表示 `#pragma managed`。如需此指示詞的詳細資訊，請參閱[managed、非受控](#)。

請注意 `DatabaseClass` 類別的私用成員：`gcroot<DataTable ^> table`。由於原生類型不能包含 managed 類型，因此 `gcroot` 必須要有關鍵詞。如需的詳細資訊 `gcroot`，請參閱[如何：在原生類型中宣告控制碼](#)。

本範例中的其餘程式碼是原生 c++ 程式碼，如前面的指示詞所表示 `#pragma unmanaged` `main`。在此範例中，我們會建立 DatabaseClass 的新實例，並呼叫其方法來建立資料表，並在資料表中填入部分資料列。請注意，原生 `SAFEARRAY` 類型會當做資料庫資料行 `ArrayIntsCol` 的值來傳遞。在 DatabaseClass 內，`SAFEARRAY` 會使用命名空間中的封送處理功能，將這些類型封送處理至 managed 物件 `System.Runtime.InteropServices`。具體而言，方法 `Copy` 是用來將封送處理 `SAFEARRAY` 至整數的 managed 陣列，而方法 `Copy` 則是用來將整數的 managed 陣列封送處理至 `SAFEARRAY`。

```
// adonet_marshal_safearray.cpp
// compile with: /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll
#include <comdef.h>
#include <gcroot.h>
#include <iostream>
using namespace std;

#using <System.Data.dll>
using namespace System;
using namespace System::Data;
using namespace System::Runtime::InteropServices;

#define MAXCOLS 100

#pragma managed
class DatabaseClass
{
public:
    DatabaseClass() : table(nullptr) { }

    void AddRow(SAFEARRAY *arrayIntsColValue)
    {
        // Add a row to the table.
        DataRow ^row = table->NewRow();
        int len = arrayIntsColValue->nrgsabound[0].cElements;
        array<int> ^arr = gcnew array<int>(len);

        int *pData;
        SafeArrayAccessData(arrayIntsColValue, (void **)&pData);
        Marshal::Copy(IntPtr(pData), arr, 0, len);
        SafeArrayUnaccessData(arrayIntsColValue);

        row["ArrayIntsCol"] = arr;
        table->Rows->Add(row);
    }

    void CreateAndPopulateTable()
    {
        // Create a simple DataTable.
        table = gcnew DataTable("SampleTable");

        // Add a column of type String to the table.
        DataColumn ^column1 = gcnew DataColumn("ArrayIntsCol",
            Type::GetType("System.Int32[]"));
        table->Columns->Add(column1);
    }

    int GetValuesForColumn(wchar_t *dataColumn, SAFEARRAY **values,
        int valuesLength)
    {
        // Marshal the name of the column to a managed
        // String.
        String ^columnStr = Marshal::PtrToStringUni(
            (IntPtr)dataColumn);

        // Get all rows in the table.
        array<DataRow ^> ^rows = table->Select();
        int len = rows->Length;
        len = (len > valuesLength) ? valuesLength : len;
```

```

        for (int i = 0; i < len; i++)
        {
            // Marshal each column value from a managed array
            // of Int32s to a SAFEARRAY of type VT_I4.
            values[i] = SafeArrayCreateVector(VT_I4, 0, 10);
            int *pData;
            SafeArrayAccessData(values[i], (void **)&pData);
            Marshal::Copy((array<int> ^)rows[i][columnStr], 0,
                IntPtr(pData), 10);
            SafeArrayUnaccessData(values[i]);
        }

        return len;
    }

private:
    // Using gcroot, you can use a managed type in
    // a native class.
    gcroot<DataTable ^> table;
};

#pragma unmanaged
int main()
{
    // Create a table and add a few rows to it.
    DatabaseClass *db = new DatabaseClass();
    db->CreateAndPopulateTable();

    // Create a standard array.
    int originalArray[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    // Create a SAFEARRAY.
    SAFEARRAY *psa;
    psa = SafeArrayCreateVector(VT_I4, 0, 10);

    // Copy the data from the original array to the SAFEARRAY.
    int *pData;
    HRESULT hr = SafeArrayAccessData(psa, (void **)&pData);
    memcpy(pData, &originalArray, 40);
    SafeArrayUnaccessData(psa);
    db->AddRow(psa);

    // Now retrieve the rows and display their contents.
    SAFEARRAY *values[MAXCOLS];
    int len = db->GetValuesForColumn(
        L"ArrayIntsCol", values, MAXCOLS);
    for (int i = 0; i < len; i++)
    {
        int *pData;
        SafeArrayAccessData(values[i], (void **)&pData);
        for (int j = 0; j < 10; j++)
        {
            cout << pData[j] << " ";
        }
        cout << endl;
        SafeArrayUnaccessData(values[i]);
    }

    // Deallocate the memory allocated using
    // SafeArrayCreateVector.
    SafeArrayDestroy(values[i]);
}

SafeArrayDestroy(psa);
delete db;

return 0;
}

```

編譯程式碼

- 若要從命令列編譯器代碼，請將程式碼範例儲存在名為 adonet_marshall_safearray 的檔案中，然後輸入下列語句：

```
c1 /clr /FU System.dll /FU System.Data.dll /FU System.Xml.dll adonet_marshall_safearray.cpp
```

.NET Framework 安全性

如需有關 ADO.NET 的安全性問題的資訊，請參閱[保護 ADO.NET 應用程式](#)。

相關章節

II	II
ADO.NET	提供 ADO.NET 的總覽，這是將資料存取服務公開給 .NET 程式設計人員的一組類別。

另請參閱

[使用 c++/CLI 進行 .NET 程式設計\(Visual C++\)](#)

[原生和 .NET 互通性](#)

[System.Runtime.InteropServices](#)

[互通性](#)

原生和 .NET 互通性

2019/12/2 • [Edit Online](#)

視覺化C++支援的互通性功能，可讓 managed 和 unmanaged 結構，以同時存在，而且交互操作內相同的組件，並甚至在相同的檔案。一小部分的這項功能，例如 P/Invoke，受到其他.NET 語言，但大部分的互通性支援提供視覺效果C++不適用於其他語言。

本節內容

[混合 \(原生和 Managed\) 組件](#)

描述使用產生的組件[/clr \(Common Language Runtime 編譯\)](#)編譯器選項，同時包含 managed 和 unmanaged 功能。

[在 MFC 中使用 Windows Forms 使用者控制項](#)

討論如何使用 MFC Windows Form 支援類別來裝載 Windows Forms 控制項，在您的 MFC 應用程式。

[從 Managed 程式碼呼叫原生函式](#)

描述如何使用非 CLR Dll，從.NET 應用程式。

與其他 .NET 程式設計語言間的互通性 (C++/CLI)

2019/12/2 • [Edit Online](#)

在本節中的主題示範如何建立視覺效果中的組件C++, 從取用或提供功能給撰寫的組件C#或 Visual Basic。

使用 C# 索引子

視覺化C++不包含索引子;它有索引的屬性。若要使用 C# 索引子, 存取索引子, 如同它是索引的屬性。

如需索引子的詳細資訊, 請參閱:

- [索引子](#)

範例

下列 C# 程式會定義索引子。

```
// consume_cs_indexers.cs
// compile with: /target:library
using System;
public class IndexerClass {
    private int [] myArray = new int[100];
    public int this [int index] {    // Indexer declaration
        get {
            // Check the index limits.
            if (index < 0 || index >= 100)
                return 0;
            else
                return myArray[index];
        }
        set {
            if (!(index < 0 || index >= 100))
                myArray[index] = value;
        }
    }
}
/*
// code to consume the indexer
public class MainClass {
    public static void Main() {
        IndexerClass b = new IndexerClass();

        // Call indexer to initialize elements 3 and 5
        b[3] = 256;
        b[5] = 1024;
        for (int i = 0 ; i <= 10 ; i++)
            Console.WriteLine("Element #{0} = {1}", i, b[i]);
    }
}
*/
```

範例

此視覺效果C++程式會使用索引子。

```

// consume_cs_indexers_2.cpp
// compile with: /clr
#using "consume_cs_indexers.dll"
using namespace System;

int main() {
    IndexerClass ^ ic = gcnew IndexerClass;
    ic->default[0] = 21;
    for (int i = 0 ; i <= 10 ; i++)
        Console::WriteLine("Element #{0} = {1}", i, ic->default[i]);
}

```

```

Element #0 = 21
Element #1 = 0
Element #2 = 0
Element #3 = 0
Element #4 = 0
Element #5 = 0
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0

```

實作 is 和 as C# 關鍵字

本主題說明如何實作的功能 `is` 並 `as` C# 視覺效果中的關鍵字 C++。

範例

```

// CS_is_as.cpp
// compile with: /clr
using namespace System;

interface class I {
public:
    void F();
};

ref struct C : public I {
    virtual void F( void ) { }
};

template < class T, class U >
Boolean isinst(U u) {
    return dynamic_cast< T >(u) != nullptr;
}

int main() {
    C ^ c = gcnew C();
    I ^ i = safe_cast< I ^ >(c);    // is (maps to castclass in IL)
    I ^ ii = dynamic_cast< I ^ >(c); // as (maps to isinst in IL)

    // simulate 'as':
    Object ^ o = "f";
    if ( isinst< String ^ >(o) )
        Console::WriteLine("o is a string");
}

```

```

o is a string

```

實作鎖定 C# 關鍵字

本主題說明如何實作C# `lock` 視覺效果中的關鍵字C++。

您也可以使用 `lock` 類別中C++支援程式庫。請參閱[同步處理 \(lock 類別\)](#)如需詳細資訊。

範例

```
// CS_lock_in_CPP.cpp
// compile with: /clr
using namespace System::Threading;
ref class Lock {
    Object^ m_pObject;
public:
    Lock( Object ^ pObject ) : m_pObject( pObject ) {
        Monitor::Enter( m_pObject );
    }
    ~Lock() {
        Monitor::Exit( m_pObject );
    }
};

ref struct LockHelper {
    void DoSomething();
};

void LockHelper::DoSomething() {
    // Note: Reference type with stack allocation semantics to provide
    // deterministic finalization

    Lock lock( this );
    // LockHelper instance is locked
}

int main()
{
    LockHelper lockHelper;
    lockHelper.DoSomething();
    return 0;
}
```

另請參閱

[以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

混合 (原生和受控) 組件

2020/11/2 • [Edit Online](#)

混合元件能夠同時包含未受管理的機器指令和 MSIL 指令。這可讓它們呼叫並由 .NET 元件呼叫，同時保留與原生 c++ 程式庫的相容性。開發人員可以使用混合的元件，使用 .NET 和原生 c++ 程式碼的混合來撰寫應用程式。

例如，透過使用 /clr 編譯器參數只重新編譯一個模組，可將完全由原生 c++ 程式碼組成的現有程式庫帶入 .net 平臺。此模組接著可以使用 .NET 功能，但仍可與應用程式的其餘部分相容。您甚至可以在同一個檔案內逐函式來決定 managed 和原生編譯之間的功能(請參閱[managed、非受控](#))。

Visual C++ 只支援使用 /clr 編譯器選項來產生混合的 managed 元件。/Clr: pure 和 /clr: safe 編譯器選項在 Visual Studio 2015 中已被取代，在 Visual Studio 2017 中不支援。如果您需要純粹或可驗證的 managed 元件，建議您使用 c# 來建立它們。

舊版 Microsoft c++ 編譯器工具組支援三種不同類型的 managed 元件產生：混合、單純和可驗證。後面兩個程式碼會在純粹和可驗證的程式碼中討論([c++/cli](#))。

本節內容

[如何:遷移至/clr](#)

說明在應用程式中引進或升級 .NET 功能的建議步驟。

[如何:使用/clr 編譯 MFC 和 ATL 程式碼](#)

討論如何編譯現有的 MFC 和 ATL 程式，以將目標設為通用語言執行時間。

[混合元件的初始化](#)

說明「載入器鎖定」的問題和解決方案。

[混合元件的程式庫支援](#)

討論如何在 /clr 編譯中使用原生程式庫。

[效能考慮](#)

說明混合元件和資料封送處理的效能影響。

[應用程式域和 Visual C++](#)

討論 Visual C++ 應用程式域的支援。

[雙重 Thunking](#)

討論 managed 函式的原生進入點的效能影響。

[避免在使用以/clr 建立的 COM 物件時，CLR 關閉的例外狀況](#)

討論如何確保使用以 /clr 編譯之 COM 物件的 managed 應用程式正常關機。

[如何:移除 CRT 程式庫 DLL 的相依性以建立部分信任的應用程式](#)

討論如何藉由移除 msxml3.dll 的相依性，以使用 Visual C++ 建立部分信任的 Common Language Runtime 應用程式。

如需混合元件之程式碼撰寫方針的詳細資訊，請參閱[Managed/非受控碼互通性的總覽](#)。

另請參閱

- [原生和 .NET 互通性](#)

如何：移轉至 /clr

2020/11/2 • [Edit Online](#)

本主題討論使用 /clr 編譯機器碼時所發生的問題(如需詳細資訊，請參閱[/Clr \(Common Language Runtime 編譯\)](#))。除了其他原生 C++ 程式碼之外，/clr 允許原生 C++ 程式碼叫用並從 .NET 元件叫用。如需使用 /clr 進行編譯之優點的詳細資訊，請參閱[混合\(原生和 Managed\)元件](#)和[原生和 .NET 互通性](#)。

使用 /clr 編譯程式庫專案的已知問題

Visual Studio 包含使用 /clr 編譯程式庫專案時的一些已知問題：

- 您的程式碼可能會在執行時間使用 `CRuntimeClass::FromName` 查詢類型。不過，如果類型是在 MSIL.dll 中(以 /clr 編譯)，則呼叫 `FromName` 可能會失敗(如果是在 managed.dll 中執行靜態的程式碼之前)(如果 `FromName` 呼叫在 managed.dll 中執行後發生，則不會看到這個問題)。若要解決這個問題，您可以藉由在 managed.dll 中定義函式、將其匯出，以及從原生 MFC 應用程式叫用它，來強制結構管理靜態函式。例如：

```
// MFC extension DLL Header file:  
_declspec( dllexport ) void EnsureManagedInitialization () {  
    // managed code that won't be optimized away  
    System::GC::KeepAlive(System::Int32::.MaxValue);  
}
```

使用 Visual C++ 進行編譯

在專案的任何模組上使用 /clr 之前，請先編譯並連結您的原生專案與 Visual Studio 2010。

下列步驟(後面接著順序)提供 /clr 編譯的最簡單路徑。在每個步驟之後，請務必編譯並執行您的專案。

Visual Studio 2003 之前的版本

如果您要從 Visual Studio 2003 之前的版本升級至 Visual Studio 2010，您可能會在 Visual Studio 2003 中看到與增強型 C++ 標準一致性相關的編譯器錯誤

從 Visual Studio 2003 升級

先前以 Visual Studio 2003 建立的專案應該先使用 /clr 編譯，因為 Visual Studio 現在增加了 ANSI/ISO 合規性和一些重大變更。可能需要最多注意的變更是[CRT 中的安全性功能](#)。使用 CRT 的程式碼很可能會產生取代警告。您可以隱藏這些警告，但最好是遷移至新的[安全性增強版本的 CRT 函式](#)，因為它們提供較佳的安全性，而且可能會顯示您程式碼中的安全性問題。

從 Managed Extensions for C++ 升級

從 Visual Studio 2005 開始，以 Managed Extensions for C++ 撰寫的程式碼不會在 /clr 下編譯。

將 C 程式碼轉換成 C++

雖然 Visual Studio 將會編譯 C 檔案，但必須將它們轉換成 C++ 以進行 /clr 編譯。實際的檔案名不需要變更；您可以使用 /tp (請參閱[/tc, /tp, /tc, /Tp \(指定來源檔案類型\)](#))。請注意，雖然 /clr 需要 C++ 原始程式碼檔案，但不需要重新考慮您的程式碼來使用物件導向的範例。

C 程式碼在編譯為 C++ 檔案時，很可能需要變更。C++ 型別安全性規則是嚴格的，因此型別轉換必須使用轉換來明確地進行。例如，`malloc` 會傳回 `void` 指標，但是可以透過 `cast` 指派給 C 中任何類型的指標：

```
int* a = malloc(sizeof(int)); // C code  
int* b = (int*)malloc(sizeof(int)); // C++ equivalent
```

函式指標在 C++ 中也是完全安全的類型，因此下列 C 程式碼需要修改。在 C++ 中，最好是建立 `typedef` 定義函式指標類型的，然後使用該類型來轉型函數指標：

```
NewFunc1 = GetProcAddress( hLib, "Func1" ); // C code  
typedef int(*MYPROC)(int); // C++ equivalent  
NewFunc2 = (MYPROC)GetProcAddress( hLib, "Func2" );
```

C++ 也要求函式必須是原型或完整定義，才能加以參考或叫用。

在 C 程式碼中，用來做為 C++ 中關鍵字的識別碼（例如 `virtual`、`new`、`delete`、`bool`、`true`、等 `false`）必須重新命名。這通常可以透過簡單的搜尋與取代作業來完成。

```
COMObj1->lpVtbl->Method(COMObj, args); // C code  
COMObj2->Method(args); // C++ equivalent
```

重新設定專案設定

在您的專案編譯並于 Visual Studio 2010 中執行之後，您應該建立 /clr 的新專案設定，而不是修改預設設定。/clr 與某些編譯器選項不相容，而建立不同的設定可讓您將專案建立為原生或受控。在 [屬性頁] 對話方塊中選取 /clr 時，會停用與 /clr 不相容的專案設定（如果後續未選取 /clr，則不會自動還原已停用的選項）。

建立新的專案設定

您可以使用 [新增專案設定] 對話方塊中的 [複製設定] 選項（[組建 Configuration Manager 使用中的方案設定] [> Configuration Manager > Active Solution Configuration > 新增]），根據現有的專案設定建立專案設定。針對 Debug 設定執行此動作一次，並針對發行設定進行一次。接下來的變更只會套用到 /clr 特定的設定，讓原始專案設定保持不變。

使用自訂群組建規則的專案可能需要特別注意。

針對使用 makefile 的專案，此步驟會有不同的含意。在此情況下，您可以設定個別的組建目標，或者可以從原始的複本建立 /clr 編譯的特定版本。

變更專案設定

您可以遵循 [/clr \(Common Language Runtime 編譯\)](#) 中的指示，在開發環境中選取 /clr。如先前所述，此步驟會自動停用衝突的專案設定。

NOTE

從 Visual Studio 2003 升級 managed 程式庫或 web 服務專案時，會將 /zI 編譯器選項新增至 [] 屬性頁。這會造成 LNK2001。從 [] 屬性頁中移除 /zI 以解決。如需詳細資訊，請參閱 [/zI \(省略預設程式庫名稱\)](#) 和 [設定編譯器和組建屬性](#)。或者，將 msrvct.lib 和 msrvcurt.lib 新增至連結器的 [] 相依性屬性。

若為以 makefile 建立的專案，則在加入 /clr 之後，必須手動停用不相容的編譯器選項。如需與 /clr 不相容之編譯器選項的詳細資訊，請參閱 [/clr 限制](#)。

先行編譯標頭檔

/Clr 下支援先行編譯的標頭。不過，如果您只使用 /clr 編譯一些 CPP 檔案（將其餘程式編譯為原生），則需要進行一些變更，因為以 /clr 產生的先行編譯標頭檔與未使用 /clr 產生的不相容。這種不相容的原因是 /clr 產生的事實，而且需要中繼資料。因此，已編譯 /clr 的模組可能不會使用不包含中繼資料的先行編譯標頭檔，而且非 /clr 模組無法使用包含中繼資料的先行編譯標頭檔。

若要編譯專案(其中有些模組是以 /clr 編譯), 最簡單的方式就是完全停用先行編譯的標頭。(在 [專案屬性頁] 對話方塊中, 開啟 [C/C++] 節點, 然後選取 [先行編譯標頭檔]。然後將 [建立/使用先行編譯標頭檔] 屬性變更為 [不使用先行編譯標頭檔])。

不過, 特別是大型專案時, 先行編譯的標頭會提供更好的編譯速度, 因此不需要停用這項功能。在此情況下, 最好將 /clr 和非 /clr 檔案設定為使用個別的先行編譯標頭檔。您可以使用****方案總管, 以滑鼠右鍵按一下群組, 然後選取 [屬性], 藉此在一個步驟中完成這項作業。然後, 變更 [透過檔案建立/使用 PCH] 和 [先行編譯頭檔案屬性], 分別使用不同的標頭檔名稱和 PCH 檔案。

修正錯誤

使用 /clr 進行編譯可能會導致編譯器、連結器或執行階段錯誤。本節討論最常見的問題。

中繼資料合併

不同版本的資料類型可能會導致連結器失敗, 因為針對這兩個類型所產生的中繼資料不相符。(這通常是因為有條件地定義類型的成員, 但使用該類型的所有 CPP 檔案的條件並不相同)。在此情況下, 連結器會失敗, 只報告符號名稱和類型定義所在之第二個 OBJ 檔案的名稱。將 OBJ 檔案傳送至連結器的順序, 通常會很有用, 因為它會探索其他版本資料類型的位置。

載入器鎖定鎖死

「載入器鎖定鎖死」可能會發生, 但具決定性, 而且會在執行時間偵測並回報。如需詳細的背景、指引和解決方案, 請參閱[混合元件的初始化](#)。

資料匯出

匯出 DLL 資料很容易出錯, 而且不建議您這麼做。這是因為在執行 DLL 的某些 managed 部分之前, 不保證會初始化 DLL 的資料區段。參考具有#using 指示詞的中繼資料。

類型可視性

原生類型預設為私用。這可能會導致原生類型不會顯示在 DLL 外部。藉由將加入 `public` 至這些類型來解決此錯誤。

浮點和對齊問題

`_controlfp` 通用語言執行時間不支援(如需詳細資訊, 請參閱[_control87](#)、[_controlfp](#)、[_control87_2](#))。CLR 也不會遵循。

COM 初始化

通用語言執行平臺會在模組初始化時自動初始化 COM (當 COM 自動初始化時, 它會以 MTA 的方式完成)。因此, 明確初始化 COM 會產生傳回碼, 指出 COM 已經初始化。當 CLR 已經將 COM 初始化為另一個執行緒模型時, 嘗試使用一個執行緒模型明確初始化 COM, 可能會導致您的應用程式失敗。

通用語言執行時間預設會將 COM 當做 MTA 啟動; 使用[/CLRTREADATTRIBUTE](#) (設定 CLR 執行緒屬性) 來修改此。

效能問題

當產生給 MSIL 的原生 C++ 方法間接呼叫(虛擬函式呼叫或使用函式指標)時, 您可能會看到效能降低。若要深入瞭解這方面的資訊, 請參閱[雙重 Thunking](#)。

從原生移至 MSIL 時, 您會注意到工作集的大小增加。這是因為 common language runtime 提供許多功能, 以確保程式正確執行。如果您的 /clr 應用程式未正確執行, 您可能會想要啟用 C4793 (預設為關閉), 請參閱[編譯器警告 \(層級1和3\)](#) C4793 以取得詳細資訊。

關機時程式損毀

在某些情況下, CLR 可以在您的 managed 程式碼完成執行之前關閉。使用 `std::set_terminate` 和 `SIGTERM` 可能會造成此問題。如需詳細資訊, 請參閱[信號常數](#) 和 `set_terminate`。

使用新的 Visual C++ 功能

在您的應用程式編譯、連結和執行之後，您就可以開始在任何以 /clr 編譯的模組中使用 .NET 功能。如需詳細資訊，請參閱[執行階段平台的元件延伸模組](#)。

如需 Visual C++ .NET 程式設計的詳細資訊，請參閱：

- [使用 c++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)
- [原生和 .NET 互通性](#)
- [執行階段平台的元件延伸模組](#)

另請參閱

[混合 \(原生和 Managed\) 元件](#)

HOW TO : 編譯 MFC 和 ATL 程式碼使用 /clr

2019/12/2 • [Edit Online](#)

本主題討論如何編譯現有的 MFC 和 ATL 程式為目標的通用語言執行平台。

若要使用 /clr 編譯 MFC 可執行檔或一般 MFC DLL

1. 在專案上按一下滑鼠右鍵方案總管，然後按一下屬性。
2. 在專案屬性對話方塊中，依序展開節點旁組態屬性，然後選取一般。在右窗格中，在專案預設值，將Common Language Runtime 支援來Common Language Runtime 支援 (/ clr)。
在相同的窗格中，請確定MFC 用法設為使用 MFC 的共用 dll。
3. 底下組態屬性，依序展開節點旁C /C++，然後選取一般。請確定偵錯資訊格式設為程式資料庫 /Zi (不/ZI)。
4. 選取 程式碼產生節點。設定啟用最少重建要否 (/ /gm-)。Ssprop_param_table_default基本執行階段會檢查要預設。
5. 底下組態屬性，選取C /C++ 再程式碼產生。請確定執行階段程式庫會設為多執行緒偵錯 DLL (/ /mdd) 或是多執行緒 DLL (/ MD)。
6. 在 Stdafx.h 中加入下面這一行。

```
#using <System.Windows.Forms.dll>
```

若要使用 /clr 編譯 MFC 擴充 DLL

1. 請依照「若要編譯 MFC 可執行檔或一般 MFC DLL 使用 /clr 」中的步驟。
2. 底下組態屬性，依序展開節點旁C /C++，然後選取先行編譯標頭。設定建立/使用先行編譯標頭要未使用先行編譯標頭。
或者，在方案總管Stdafx.cpp 上按一下滑鼠右鍵，然後按一下 屬性。底下組態屬性，依序展開節點旁C /C++，然後選取一般。設定Common Language Runtime 支援編譯要Common Language Runtime 不支援。
3. 檔案包含 DllMain 和任何項目呼叫，在方案總管，以滑鼠右鍵按一下檔案，然後按一下屬性。底下組態屬性，依序展開節點旁C /C++，然後選取一般。在右窗格中下,專案預設值，將Common Language Runtime 支援編譯來No Common Language Runtime 支援。

若要使用 /clr 編譯 ATL 可執行檔

1. 在 方案總管，以滑鼠右鍵按一下專案，然後按一下屬性。
2. 在專案屬性對話方塊中，依序展開節點旁組態屬性，然後選取一般。在右窗格中，在專案預設值，將Common Language Runtime 支援來Common Language Runtime 支援 (/ clr)。
3. 底下組態屬性，依序展開節點旁C /C++，然後選取一般。請確定偵錯資訊格式設為程式資料庫 /Zi (不/ZI)。
4. 選取 程式碼產生節點。設定啟用最少重建要否 (/ /gm-)。Ssprop_param_table_default基本執行階段會檢查要預設。
5. 底下組態屬性，選取C /C++ 再程式碼產生。請確定執行階段程式庫會設為多執行緒偵錯 DLL (/ /mdd) 或是多執行緒 DLL (/ MD)。

- 針對每個 MIDL 產生檔案 (C 檔案)，以滑鼠右鍵按一下中的檔案方案總管，然後按一下屬性。底下組態屬性，依序展開節點旁 C /C++，然後選取一般。設定 Common Language Runtime 支援編譯要 Common Language Runtime 不支援。

若要使用 /clr 編譯 ATL DLL

- 請遵循 < 使用 /clr 編譯 ATL 可執行檔 > 一節中的步驟。
- 底下組態屬性，依序展開節點旁 C /C++，然後選取先行編譯標頭。設定建立/使用先行編譯標頭要未使用先行編譯標頭。

或者，在方案總管 Stdafx.cpp 上按一下滑鼠右鍵，然後按一下 屬性。底下組態屬性，依序展開節點旁 C /C++，然後選取一般。設定 Common Language Runtime 支援編譯要 Common Language Runtime 不支援。
- 檔案包含 DllMain 和任何項目呼叫，在方案總管，以滑鼠右鍵按一下檔案，然後按一下屬性。底下組態屬性，依序展開節點旁 C /C++，然後選取一般。在右窗格中下專案預設值，將 Common Language Runtime 支援編譯來 No Common Language Runtime 支援。

另請參閱

[混合 \(原生和 Managed\) 組件](#)

混合組件的初始化

2020/11/2 • [Edit Online](#)

在期間執行程式碼時，Windows 開發人員一定要小心載入載入器鎖定 `DllMain`。不過，在處理 C++/CLI 混合模式元件時，需要考慮一些其他問題。

`DllMain` 內的程式碼不得存取 .NET Common Language RUNTIME (CLR)。這表示，不 `DllMain` 應該直接或間接呼叫 managed 函式；不應該在中宣告或執行 managed 程式碼，也 `DllMain` 不應該在中進行垃圾收集或自動程式庫載入 `DllMain`。

載入器鎖定的原因

隨著 .NET 平臺的推出，有兩個不同的機制可將執行模組載入 (EXE 或 DLL)：一個適用於非受控模組，另一個用於 CLR，它會載入 .NET 元件。混合 DLL 載入問題主要與 Microsoft Windows 作業系統載入器相關。

當僅包含 .NET 結構的元件載入至進程時，CLR 載入器可以執行所有必要的載入和初始化工作本身。不過，若要載入可包含機器碼和資料的混合元件，也必須使用 Windows 載入器。

Windows 載入器保證在初始化程式碼之前，沒有任何程式碼可以存取該 DLL 中的程式碼或資料。也可確保程式碼在部分初始化時，不會重複載入 DLL。若要這樣做，Windows 載入器會使用進程全域關鍵區段 (通常稱為「載入器鎖定」)，可在模組初始化期間防止不安全的存取。因此，載入程序很容易受到許多典型死結案例的危害。若是混合組件，下列兩種案例會增加發生死結的風險：

- 首先，如果使用者嘗試執行編譯為 Microsoft 中繼語言的函式 (MSIL) 當載入器鎖定 (自 `DllMain` 或靜態初始化運算式 (例如)) 時，它可能會造成鎖死。請考慮 MSIL 函數在尚未載入的元件中參考型別的情況。CLR 會嘗試自動載入該組件，這可能需要 Windows 載入器對載入器鎖定進行封鎖。因為先前在呼叫順序中的程式碼已持有載入器鎖定，所以會發生鎖死。不過，在載入器鎖定下執行 MSIL 不保證會發生鎖死。這就是這種情況很難診斷和修正。在某些情況下 (例如，當參考型別的 DLL 未包含原生結構，而且其所有相依性不包含任何原生結構) 時，不需要 Windows 載入器載入所參考型別的 .NET 元件。此外，必要的組件或其混合的原生 (.NET) 相依性可能已由其他程式碼載入。因此，死結的發生不僅很難預測，也會因目標電腦的組態而異。
- 第二，在 .NET Framework 的 1.0 和 1.1 版中載入 DII 時，CLR 會假設載入器鎖定不會被保留，並且會在載入器鎖定下採取數個不正確動作。假設未持有載入器鎖定是單純 .NET DII 的有效假設。但是由於混合 DII 會執行原生初始化常式，因此需要原生的 Windows 載入器，因此也需要載入器鎖定。因此，即使開發人員在 DLL 初始化期間未嘗試執行任何 MSIL 函數，在 .NET Framework 版本 1.0 和 1.1 中仍有很小的非決定性鎖死。

在混合 DLL 載入程序中，所有不具決定性問題都已獲得解決。這項變更已完成，但有下列變更：

- 載入混合 DLL 時，CLR 不會再做出錯誤的假設。
- 非受控和受控初始化是在兩個不同的階段中完成。非受控初始化會先 `DllMain` 透過) 進行 (，而且之後會透過來進行 managed 初始化。支援 NET 的 `.cctor` 結構。除非使用或，否則後者對使用者而言是完全透明的 `/ZI` `/NODEFAULTLIB`。如需詳細資訊，請參閱 `/NODEFAULTLIB` (忽略程式庫) 和 `/ZI` (省略預設程式庫名稱)。

載入器鎖定仍然有可能發生，但現在能夠重現並偵測所發生的情況。如果 `DllMain` 包含 MSIL 指令，編譯器會產生警告 [編譯器警告 \(層級 1\) C4747](#)。此外，CRT 或 CLR 會嘗試偵測並回報在載入器鎖定下嘗試執行 MSIL。CRT 偵測會產生執行階段診斷 C 執行階段錯誤 R6033。

本文的其餘部分將說明可在載入器鎖定下執行 MSIL 的其餘案例。它會示範如何解決每個案例的問題，以及偵錯工具的方法。

案例和因應措施

在幾種不同的情況下，使用者程式碼可能會在載入器鎖定下執行 MSIL。開發人員必須確保使用者程式碼的執行不會在每一種情況下嘗試執行 MSIL 指令。下列各節說明所有可能性，並討論如何解決最常見案例中的問題。

DllMain

`DllMain` 函數是 DLL 的使用者定義進入點。除非使用者另外指定，否則每次處理序或執行緒附加至所包含的 DLL 或與其中斷連結時，都會叫用 `DllMain`。因為持有載入器鎖定時會發生此引動過程，所以使用者提供的 `DllMain` 函式不應該編譯為 MSIL。此外，根目錄為 `DllMain` 的呼叫樹狀結構中的函式不可以編譯為 MSIL。若要解決此問題，您應該使用修改定義的程式碼區塊 `DllMain #pragma unmanaged`。針對 `DllMain` 呼叫的每個函式，都應該執行相同的作業。

如果這些函式必須呼叫需要 MSIL 實作為其他呼叫內容的函式，您可以使用重複的策略，同時建立 .NET 和原生版本的相同函式。

或者，如果不需要 `DllMain` 執行，或不需要在載入器鎖定下執行，您可以移除使用者提供的 `DllMain` 執行，以排除問題。

如果 `DllMain` 嘗試直接執行 MSIL，則會產生 [編譯器警告 \(層級 1\) C4747](#)。不過，編譯器無法偵測 `DllMain` 呼叫另一個模組中的函式，進而嘗試執行 MSIL 的情況。

如需此案例的詳細資訊，請參閱 [診斷的阻礙](#)。

初始化靜態物件

如果需要動態初始設定式，初始化靜態物件就可能會產生死結。簡單的案例（例如，當您將編譯時間已知的值指派給靜態變數時）不需要動態初始化，因此不會有鎖死的風險。不過，某些靜態變數會由函式呼叫、函式調用，或無法在編譯時期評估的運算式初始化。這些變數都需要在模組初始化期間執行程式碼。

下列程式碼示範需要動態初始化的靜態初始設定式：函式呼叫、物件建構和指標初始化（這些範例並不是靜態的，但假設有全域範圍中的定義，其效果也相同。）

```
// dynamic initializer function generated
int a = init();
CObject o(arg1, arg2);
CObject* op = new CObject(arg1, arg2);
```

這種鎖死的風險取決於所包含的模組是否使用編譯 `/clr`，以及是否會執行 MSIL。具體而言，如果靜態變數未 `/clr`（或在區塊）中進行編譯 `#pragma unmanaged`，而將它初始化所需的動態初始化運算式導致執行 MSIL 指令，則可能會發生鎖死。這是因為在不使用編譯的模組中，會 `/clr` 由 `DllMain` 執行靜態變數的初始化。相反地，使用編譯的靜態變數 `/clr` 會在 `.cctor` 未受管理的初始化階段完成並釋放載入器鎖定之後，由初始化。

動態初始化靜態變數時，有許多因應鎖死的解決方案。這些問題大約會依照修正問題所需的時間順序排列在此處：

- 包含靜態變數的來源檔案可以使用編譯 `/clr`。
- 靜態變數所呼叫的所有函式都可以使用指示詞編譯為機器碼 `#pragma unmanaged`。
- 以手動方式複製靜態變數所依賴的程式碼，並為 .NET 和原生版本指定不同的名稱。接著，開發人員可從原生靜態初始設定式呼叫原生版本，並從別處呼叫 .NET 版本。

影響啟動的使用者提供函式

程式庫在啟動期間的初始化，會依賴數個使用者提供的函式。例如，在 c++ 中全域多載運算子（例如 `new` 和 `delete` 運算子）時，使用者提供的版本會在所有地方使用，包括 c++ 標準程式庫初始化和銷毀。因此，c++ 標準程式庫和使用者提供的靜態初始化運算式將會叫用這些運算子的任何使用者提供版本。

如果使用者提供的版本會編譯為 MSIL，則這些初始設定式會在持有載入器鎖定時嘗試執行 MSIL 指令。使用者提供 `malloc` 的結果相同。若要解決這個問題，您必須使用指示詞，將這些多載或使用者提供的定義實作為原生程式

碼 `#pragma unmanaged`。

如需此案例的詳細資訊，請參閱 [診斷的阻礙](#)。

自訂地區設定

如果使用者提供自訂的全域地區設定，則會使用此地區設定來初始化所有未來的 i/o 資料流程，包括靜態初始化的資料流程。如果此全域地區設定物件會編譯為 MSIL，就可能在持有載入器鎖定時，叫用編譯為 MSIL 的地區設定物件成員函式。

有三個解決此問題的選項：

包含所有全域 i/o 資料流程定義的來源檔案，可以使用選項進行編譯 `/clr`。它會防止在載入器鎖定下執行其靜態初始化運算式。

您可以使用指示詞，將自訂地區設定函式定義編譯成機器碼 `#pragma unmanaged`。

請等到釋放載入器鎖定之後，再將自訂地區設定設定為全域地區設定。然後再明確設定初始化期間使用自訂地區設定建立的 I/O 資料流。

診斷的阻礙

在某些情況下，很難偵測到鎖死的來源。下列各節將討論這些案例及解決這些問題的方法。

標頭中的實作

在選取的案例中，標頭檔內的函式實作可能會使診斷變得複雜。內嵌函式和範本程式碼都要求必須在標頭檔中指定函式。C++ 語言會指定「一個定義規則」，強制所有同名的函式實作在語意上相等。因此，當合併的物件檔案具有指定函式的重複實作時，C++ 連結器不需要進行任何特殊的考量。

在 Visual Studio 2005 之前的 Visual Studio 版本中，連結器只會選擇這些語義相等定義的最大值。這樣做的目的是為了配合向前宣告，以及針對不同的原始程式檔使用不同的優化選項時的案例。它會為混合的原生和 .NET DII 建立問題。

因為 C++ 檔案可以同時包含相同的標頭 (`/clr` 啟用和停用)，或者 `#include` 可以包裝在 `#pragma unmanaged` 區塊內，所以可以有 MSIL 和原生版本的函式可在標頭中提供實作為。MSIL 和原生實作為載入器鎖定下的初始化有不同的語法，這實際上違反了一個定義規則。因此，當連結器選擇最大的實值時，它可能會選擇 MSIL 版本的函式，即使在其他地方使用指示詞明確地編譯成機器碼 `#pragma unmanaged`。為了確保不會在載入器鎖定下呼叫 MSIL 版本的範本或內嵌函式，在載入器鎖定下呼叫的每個這類函式的每個定義都必須使用指示詞來修改 `#pragma unmanaged`。如果標頭檔是來自協力廠商，則進行這項變更最簡單的方式就是在 `#pragma unmanaged` 有問題的標頭檔的 `#include` 指示詞周圍推送和 `pop` 指示詞。(請參閱 [受控、非受控](#) 的範例。) 不過，此策略不適用於包含其他必須直接呼叫 .NET API 之程式碼的標頭。

為方便使用者處理載入器鎖定，若原生實作和 Managed 實作同時存在，連結器會優先選擇原生實作。此預設值可避免上述問題。不過，在此版本中，這項規則有兩個例外狀況，因為編譯器有兩個未解決的問題：

- 內嵌函式的呼叫是透過全域靜態函式指標。這個案例 `isn'table` 的原因是虛擬函式是透過全域函式指標呼叫。例如，

```

#include "definesmyObject.h"
#include "definesclassC.h"

typedef void (*function_pointer_t)();

function_pointer_t myObject_p = &myObject;

#pragma unmanaged
void DuringLoaderlock(C & c)
{
    // Either of these calls could resolve to a managed implementation,
    // at link-time, even if a native implementation also exists.
    c.VirtualMember();
    myObject_p();
}

```

在偵錯模式中診斷

載入器鎖定問題的所有診斷都應該使用偵錯組建來完成。發行組建可能不會產生診斷。此外，在發行模式中進行的優化，可能會在載入器鎖定案例下遮罩某些 MSIL。

如何偵錯工具載入器鎖定問題

在叫用 MSIL 函式時，CLR 所產生的診斷會造成 CLR 暫停執行。接著也會在執行中的偵錯工具時，使 Visual C++ 混合模式偵錯工具暫停。不過，當附加至進程時，it'sn't 可能會使用混合偵錯工具取得偵錯工具的 managed 呼叫堆疊。

若要識別在載入器鎖定下所呼叫的特定 MSIL 函式，開發人員應該完成下列步驟：

- 確定可以使用 mscoree.dll 和 mscorewks.dll 的符號。

您可以利用兩種方式來提供符號。首先，您可以將 mscoree.dll 和 mscorewks.dll 的 PDB 加入符號搜尋路徑。

若要加入它們，請開啟 [符號搜尋路徑選項] 對話方塊。從 [工具] 功能表 (，選擇 [選項]。在 [選項] 對話方塊的左窗格中，開啟 [調試 程式] 節點，然後選擇 [符號]。) 將 mscoree.dll 和 mscorewks.dll PDB 檔案的路徑新增至搜尋清單。這些 PDB 會安裝到 %VSINSTALLDIR%\SDK\v2.0\symbols。選擇 [確定]。

其次，您可以從 Microsoft 符號伺服器下載 mscoree.dll 和 mscorewks.dll 的 PDB。若要設定符號伺服器，請開啟 [symbol search path options] (符號搜尋路徑選項) 對話方塊。從 [工具] 功能表 (，選擇 [選項]。在 [選項] 對話方塊的左窗格中，開啟 [調試 程式] 節點，然後選擇 [符號]。) 將此搜尋路徑新增至搜尋清單：
<https://msdl.microsoft.com/download/symbols>。將符號快取目錄加入符號伺服器快取文字方塊。選擇 [確定]。

- 將偵錯工具模式設定為僅限原生模式。

在方案中開啟啟始專案的 [屬性] 方格。選取設定屬性 > 的調試。將 [偵錯工具類型] 屬性設定為 [僅限原生]。

- 啟動偵錯工具 (F5)。

- 當 `/clr` 診斷產生時，請選擇 [重試]，然後選擇 [中斷]。
- 開啟呼叫堆疊視窗 (在功能表列上，選擇 [Debug > Windows > 呼叫堆疊])。`DllMain` 有問題或靜態初始化運算式以綠色箭號識別。如果未識別出違規的函式，則必須採取下列步驟來尋找它。
- 開啟功能表列上的 [即時運算視窗] (，選擇 [Debug > Windows > immediate])。
- `.load sos.dll` 在 [即時運算] 視窗中輸入，以載入 SOS 調試服務。
- 在 `!dumpstack` [即時Immediate運算] 視窗中輸入，以取得內部堆疊的完整清單 `/clr`。
- 尋找最接近) 堆疊最底部的第一個實例 (`_CorDIIMain` (如果 `DllMain` 靜態初始化運算式造成問題)，則

`_VTableBootstrapThunkInitHelperStub` 或 `GetTargetForVTableEntry()`。此呼叫正下方的堆疊項目，會是在載入器鎖定下嘗試執行之 MSIL 實作函式的引動過程。

10. 移至在上一個步驟中識別的原始程式檔和行號，並使用案例一節中所述的案例和解決方案來修正問題。

範例

描述

下列範例顯示如何將程式碼從移 `DllMain` 至全域物件的函式，以避免載入器鎖定。

在此範例中，有一個全域 managed 物件，其函式包含原本在中的 managed 物件 `DllMain`。此範例的第二個部分會參考元件，並建立 managed 物件的實例來叫用執行初始化的模組函式。

程式碼

```
// initializing_mixed_assemblies.cpp
// compile with: /clr /LD
#pragma once
#include <stdio.h>
#include <windows.h>
struct __declspec(dllexport) A {
    A() {
        System::Console::WriteLine("Module ctor initializing based on global instance of class.\n");
    }

    void Test() {
        printf_s("Test called so linker doesn't throw away unused object.\n");
    }
};

#pragma unmanaged
// Global instance of object
A obj;

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved) {
    // Remove all managed code from here and put it in constructor of A.
    return true;
}
```

此範例示範混合元件的初始化問題：

```
// initializing_mixed_assemblies_2.cpp
// compile with: /clr initializing_mixed_assemblies.lib
#include <windows.h>
using namespace System;
#include <stdio.h>
#using "initializing_mixed_assemblies.dll"
struct __declspec(dllimport) A {
    void Test();
};

int main() {
    A obj;
    obj.Test();
}
```

此程式碼會產生下列輸出：

Module ctor initializing based on global instance of class.

Test called so linker doesn't throw away unused object.

另請參閱

[混合 \(原生和 Managed\) 元件](#)

混合組件的程式庫支援

2019/12/2 • [Edit Online](#)

視覺化C++支援使用C++標準程式庫、C執行階段程式庫(CRT)、ATL和MFC編譯的應用程式/[clr \(Common Language Runtime 編譯\)](#)。這可讓使用.NET Framework功能，以及使用這些程式庫的現有應用程式。

IMPORTANT

/Clr: pure並/Clr: safe編譯器選項是在Visual Studio 2015中已被取代，而且不支援的Visual Studio 2017中。

這項支援包括下列 DLL 和匯入程式庫：

- 如果您使用編譯的Msxml.dll /clr。此匯入程式庫混合組件連結。

這項支援可提供數個相關的優點：

- CRT與C++標準程式庫可供混合程式碼。CRT與C++標準程式庫會提供不是可驗證;最後，您呼叫仍會路由至相同的CRT與C++為您的標準程式庫從原生程式碼中使用。
- 修正在混合的映像中的統一的例外狀況處理。
- 靜態初始設定的C++在混合的映像中的變數。
- Managed程式碼中的每個AppDomain和處理序專屬變數的支援。
- 解析用於編譯Visual Studio 2003及更早版本的混合DII載入器鎖定問題。

此外，這項支援存在下列限制：

- 只將CRT DLL的模型才支援以編譯程式碼 /clr。有不支援的靜態CRT程式庫 /clr組建。

另請參閱

- [混合\(原生和Managed\)組件](#)

Interop 的效能考量 (C++)

2019/12/2 • [Edit Online](#)

本主題會提供指導方針，以降低管理/未受管理的執行階段效能的 interop 轉換的效果。

視覺化 C++ 支援相同的互通性機制，與其他.NET 語言，例如 Visual Basic 和 C#(P/Invoke)，但它也提供特定視覺效果的 interop 支援 C++(C++ interop)。針對效能關鍵的應用程式，請務必了解每個 interop 技術的效能含意。

不論所使用的 interop 技巧，特殊的轉換序列，稱為 thunk，都需要每次 managed 函式呼叫 unmanaged 的函式，反之亦然。Microsoft 會自動插入這些 thunk C++ 編譯器，但它是要牢記在心，累積，這些轉換可能耗用大量效能很重要。

減少轉換

若要避免或減少的 interop thunk 成本的一個方式是重構所牽涉到 managed/unmanaged 轉換降到最低的介面。將目標設為多對話介面，是指涉及經常呼叫跨 managed/unmanaged 界限可大幅提升效能。Managed 函式呼叫 unmanaged 函式在緊密迴圈中，比方說，是很好的候選重構。如果迴圈本身會移至未受管理端，或如果受管理的替代方案，以建立未受管理的呼叫（或許是對 unmanaged 端的併列資料，然後封送處理其 unmanaged api 回圈後，全部一次），轉換的數目可以降低 significantly。

P/Invoke vs. C++ Interop

適用於.NET 的語言，例如 Visual Basic 和 C# 中，以原生元件相互操作的指定的方法是 P/Invoke。因為.NET Framework 中，視覺效果支援 P/Invoke C++ 支援它，但 Visual C++ 也提供自己的互通性支援，指 C++ Interop。C++ Interop 建議，透過 P/Invoke 因為 P/Invoke 不是類型安全。如此一來，錯誤主要會報告在執行階段，但 C++ Interop 也有透過 P/Invoke 的效能優點。

這兩種技術需要時 managed 函式會呼叫 unmanaged 函式的幾件事：

- 函式呼叫引數是以原生類型，封送處理從 CLR。
- 執行 managed 至 unmanaged 的 thunk。
- Unmanaged 函式會呼叫（使用原生的版本引數）。
- 執行非受控至 managed 的 thunk。
- 傳回的型別和任何「發送」或「在 out」引數會從 CLR 類型的原生封送處理。

Managed/unmanaged 的 thunk 所需的 interop 運作，但資料封送處理所需取決於所涉及的資料類型、函式簽章，以及如何使用資料。

資料封送處理由執行 C++ Interop 是最簡單的可能形式：參數只會複製跨 managed/unmanaged 的界限，以位元的方式，在所有不執行任何轉換。P/Invoke，這是僅為 true，如果所有參數都簡單，blittable 類型。否則，P/Invoke 會執行非常強大的步驟，將每個受管理的參數轉換成適當的原生型別，而且反之亦然引數會標示為 "out"，或 "in"，"out"。

換句話說，C++ Interop 使用的資料封送處理，最快的可能方法，而 P/Invoke 使用的最強大的方法。這表示，C++ Interop (以一般方式 C++) 提供最佳效能，根據預設，程式設計人員是負責處理，這個行為並不安全或不適當的情況。

C++ Interop 因此需要封送處理的資料必須明確地提供，但優點是程式設計人員可以自由地決定適當的資料，本質以及它所要使用的方式。此外，雖然的 P/Invoke 資料封送處理行為可以修改在自訂的程度，C++ Interop 讓封送處理至呼叫藉由呼叫基礎上自訂的資料。這是不可能使用 P/Invoke。

如需詳細資訊C++ Interop, 請參閱[Using C++ Interop \(隱含 PInvoke\)](#)。

另請參閱

[混合 \(原生和 Managed\) 組件](#)

應用程式定義域和 Visual C++

2020/3/25 • [Edit Online](#)

如果您有 `__clrcall` 虛擬函式，則 vtable 將會是每個應用程式域(appdomain)。如果您在一個 appdomain 中建立物件，您只能從該 appdomain 內呼叫虛擬函式。在混合模式(`/clr`)中，如果您的類型沒有 `__clrcall` 的虛擬函式，您就會有每個進程的 vtable。`/Clr: pure`和`/clr: safe`編譯器選項在 Visual Studio 2015 中已被取代，在 Visual Studio 2017 中不支援。

如需詳細資訊，請參閱

- [appdomain](#)
- [__clrcall](#)
- [process](#)

另請參閱

- [混合 \(原生和 Managed\) 組件](#)

Double Thunking (C++)

2020/11/2 • [Edit Online](#)

雙 Thunking 是指當 managed 內容中的函式呼叫呼叫 Visual C++ managed 函式，而程式執行呼叫函式的原生進入點以呼叫 managed 函式時，您可能會遇到的效能損失。本主題討論雙 Thunking 發生的位置，以及如何避免它以改善效能。

備註

根據預設，當使用 /clr 進行編譯時，managed 函式的定義會導致編譯器產生 managed 進入點和原生進入點。這可讓您從原生和 managed 呼叫網站呼叫 managed 函式。不過，當原生進入點存在時，它可以是函數所有呼叫的進入點。如果呼叫的函式是受管理的，則原生進入點接著會呼叫 managed 進入點。實際上，需要兩個呼叫來叫用函式（因此，會）雙 Thunking。例如，一律會透過原生進入點呼叫虛擬函式。

其中一個解決方法是告知編譯器不要產生 managed 函式的原生進入點，而且只會使用 `_clrcall` 呼叫慣例，從 managed 內容呼叫函式。

同樣地，如果您將 (`dllexport`、`dllimport`) managed 函式，就會產生原生進入點，而且任何匯入和呼叫該函式的函式都會透過原生進入點呼叫。若要避免在這種情況下進行雙重 Thunking，請勿使用原生匯出/匯入語義；只要透過（參考中繼資料，`#using` 請參閱 `#using` 指示詞）。

編譯器已更新，可減少不必要的雙 Thunking。例如，簽章中具有 managed 型別的任何函式（包括傳回型別）都會隱含地標示為 `_clrcall`。

範例：Double Thunking

描述

下列範例示範雙 Thunking。當編譯的原生（不含 /clr）時，對中的虛擬函式的呼叫會 `main` 產生對 `T` 的複製函式的呼叫，以及對該呼叫者的一個呼叫。使用 /clr 和來宣告虛擬函式時，就可以達到類似的行為 `_clrcall`。但是，當您只使用 /clr 編譯時，函式呼叫會產生對複製函式的呼叫，但由於原生對 managed Thunk 的緣故，所以會有另一個對複製函式的呼叫。

程式碼

```

// double_thunking.cpp
// compile with: /clr
#include <stdio.h>
struct T {
    T() {
        puts(__FUNCSIG__);
    }

    T(const T&) {
        puts(__FUNCSIG__);
    }

    ~T() {
        puts(__FUNCSIG__);
    }

    T& operator=(const T&) {
        puts(__FUNCSIG__);
        return *this;
    }
};

struct S {
    virtual void /* __clrcall */ f(T t) {};
} s;

int main() {
    S* pS = &s;
    T t;

    printf("calling struct S\n");
    pS->f(t);
    printf("after calling struct S\n");
}

```

範例輸出

```

__thiscall T::T(void)
calling struct S
__thiscall T::T(const struct T &)
__thiscall T::T(const struct T &)
__thiscall T::~T(void)
__thiscall T::~T(void)
after calling struct S
__thiscall T::~T(void)

```

範例: 雙 Thunking 的效果

描述

先前的範例示範了雙 Thunking 的存在。此範例顯示其效果。`for` 迴圈會呼叫虛擬函式，而程式會報告執行時間。以 `/clr` 編譯器時，會報告最慢的時間。如果編譯時沒有 `/clr`，或以宣告虛擬函式時，就會回報最快的時間 `__clrcall`。

程式碼

```
// double_thunking_2.cpp
// compile with: /clr
#include <time.h>
#include <stdio.h>

#pragma unmanaged
struct T {
    T() {}
    T(const T&) {}
    ~T() {}
    T& operator=(const T&) { return *this; }
};

struct S {
    virtual void /* __clrcall */ f(T t) {};
} s;

int main() {
    S* pS = &s;
    T t;
    clock_t start, finish;
    double duration;
    start = clock();

    for ( int i = 0 ; i < 1000000 ; i++ )
        pS->f(t);

    finish = clock();
    duration = (double)(finish - start) / (CLOCKS_PER_SEC);
    printf( "%2.1f seconds\n", duration );
    printf("after calling struct S\n");
}
```

範例輸出

```
4.2 seconds
after calling struct S
```

另請參閱

[混合 \(原生和 Managed\) 元件](#)

當使用以 /clr 建置的 COM 物件時防止 CLR 關閉之例外狀況

2019/12/2 • [Edit Online](#)

一旦 common language runtime (CLR) 會進入關機模式，原生函式具有有限存取權 CLR 服務。COM 物件時嘗試呼叫版本上使用編譯 /clrCLR 會轉換成原生程式碼，再轉換回 iunknown:: Release 呼叫（這定義在 managed 程式碼）提供服務的 managed 程式碼。CLR 可防止呼叫至 managed 程式碼，因為它是在關機模式。

若要解決此問題，請確定從發行方法呼叫的解構函式只能包含原生程式碼。

另請參閱

[混合 \(原生和 Managed\) 組件](#)

如何：移除 CRT 程式庫 DLL 的相依性以建立部分信任的應用程式

2020/4/15 • [Edit Online](#)

本主題討論如何通過刪除對 msxml.dll 的依賴項,使用 Visual C++ 創建部分受信任的通用語言運行時應用程式。

使用 /clr 構建的可視 C++ 應用程式將依賴於 msxml.dll,msxml.dll 是 C-執行時庫的一部分。當您希望應用程式在部分信任環境中使用時,CLR 將在 DLL 上強制實施某些代碼訪問安全規則。因此,有必要刪除此依賴項,因為 msxml.dll 包含本機代碼,並且無法對它強制實施代碼存取安全策略。

如果應用程式不使用 C-Runtime 庫的任何功能,並且希望從代碼中刪除對此庫的依賴項,則必須使用 /NODEFAULTLIB:msvcrt.lib 連結器選項並與 ptrustm.lib 或 ptrustmd.lib 連結。這些庫包含用於應用程式初始化和非初始化的物件檔、初始化代碼使用的異常類以及託管異常處理代碼。在這些庫中連結將刪除對 msxml.dll 的任何依賴。

NOTE

對於使用 ptrust 庫的應用程式,程式集取消初始化的順序可能不同。對於普通應用程式,程式集通常按載入程式集的相反順序卸載,但這不能保證。對於部分信任應用程式,程式集的卸載順序通常與載入程式集的順序相同。這也不能保證這一點。

建立部分受信任的混合 (/clr) 應用程式

- 要刪除對 msxml.dll 的依賴項,必須指定連結器不要使用 /NODEFAULTLIB:msvcrt.lib 連結器選項來包括此庫。有關如何使用可視化工作室開發環境或以程式設計方式執行此操作的資訊,請參閱 [/NODEFAULTLIB\(忽略庫\)](#)。
- 將其中一個 ptrustm 庫添加到連結器輸入依賴項。如果要在發佈模式下構建應用程式,請使用 ptrustm.lib。對於調試模式,請使用 ptrustmd.lib。有關如何使用 Visual Studio 開發環境或以程式設計方式執行此操作的資訊,請參閱 [Lib 檔案作為連結器輸入](#)。

另請參閱

[混合 \(原生和 Managed\) 組件](#)

[混合程式集的初始化](#)

[混合組件的程式庫支援](#)

[/link \(傳遞選項給連結器\)](#)

在 MFC 中使用 Windows Form 使用者控制項

2019/12/2 • [Edit Online](#)

使用 MFC Windows Forms 支援類別，您可以將 MFC 應用程式內的 Windows Forms 控制項裝載為 MFC 對話方塊或視圖中的 ActiveX 控制項。此外，Windows Forms 表單也可以裝載為 MFC 對話方塊。

下列各節說明如何：

- 在 MFC 對話方塊中裝載 Windows Forms 控制項。
- 將 Windows Forms 的使用者控制項裝載為 MFC 視圖。
- 將 Windows Forms 表單裝載為 MFC 對話方塊。

NOTE

Mfc Windows Forms 整合只適用於以 mfc (定義的專案 `_AFXDLL`) 動態連結的專案。

NOTE

當您使用 MFC Windows Forms 介面 DLL (mfcmifc80) 的私用 (已修改) 複本來建立應用程式時，除非您使用自己的廠商金鑰來取代 Microsoft 金鑰，否則它將無法安裝在 GAC 中。如需元件簽署的詳細資訊，請參閱[使用元件](#)進行程式設計和[強式名稱元件 \(C++ 元件簽署\) \(/cli\)](#)。

如果您的 MFC 應用程式使用 Windows Forms，您需要使用您的應用程式轉散發 mfcmifc80。如需詳細資訊，請參閱[轉散發 MFC 程式庫](#)。

本節內容

[將 Windows Forms 使用者控制項裝載至 MFC 對話方塊中](#)

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

[將 Windows Forms 使用者控制項裝載成 MFC 對話方塊](#)

參考資料

[CWinFormsControl 類別](#)

[CWinFormsDialog 類別](#)

[CWinFormsView 類別](#)

[ICommandSource 介面](#)

[ICommandTarget 介面](#)

[ICommandUI 介面](#)

[IView 介面](#)

[CommandHandler](#)

[DDX_ManagedControl](#)

[UICheckState](#)

相關章節

[Windows Forms](#)

[Windows Forms 控制項](#)

另請參閱

[使用者介面元素](#)

[表單檢視](#)

Windows Form/MFC 程式設計的差異

2019/12/2 • [Edit Online](#)

在 MFC 中使用 Windows Form 使用者控制項中的主題會描述 WINDOWS FORMS 的 MFC 支援。如果您不熟悉 .NET Framework 或 MFC 程式設計，本主題會提供兩者之間的程式設計差異的背景資訊。

Windows Forms 是用來在 .NET Framework 上建立 Microsoft Windows 應用程式。此架構提供了一組現代化、物件導向、可擴充的類別，可讓您開發豐富的 Windows 應用程式。使用 Windows Forms，您可以建立豐富的用戶端應用程式，以存取各種不同的資料來源，並使用 Windows Forms 控制項來提供資料顯示和資料編輯工具。

不過，如果您習慣使用 MFC，您可能會用來建立 Windows Forms 中尚未明確支援的特定類型應用程式。Windows Forms 的應用程式相當於 MFC 對話應用程式。不過，它們不會提供直接支援其他 MFC 應用程式類型的基礎結構，例如 OLE 檔案伺服器/容器、ActiveX 檔、單一檔介面(SDI)的檔/視圖支援、多重文件介面(MDI)，以及多個頂層介面(MTI)。您可以撰寫自己的邏輯來建立這些應用程式。

如需 Windows Forms 應用程式的詳細資訊，請參閱[Windows Forms 簡介](#)。

如需顯示與 MFC 搭配使用之 Windows Forms 的範例應用程式，請參閱[mfc 和 Windows Forms 整合](#)。

下列 MFC 視圖或檔和命令路由功能在 Windows Forms 中沒有對等專案：

- Shell 整合

當您以滑鼠右鍵按一下檔，然後選取 [開啟]、[編輯] 或 [列印] 這類動詞時，MFC 會處理 shell 使用的動態資料交換(DDE)命令和命令列引數。Windows Forms 沒有 shell 整合，而且不會回應 shell 動詞命令。

- 檔範本

在 MFC 中，檔範本會將包含在框架視窗(在 MDI、SDI 或 MTI 模式中)的視圖與您開啟的檔產生關聯。Windows Forms 不等於檔範本。

- 文件

MFC 會在從 shell 開啟檔時，註冊檔檔類型並處理檔案類型。Windows Forms 沒有檔支援。

- 檔狀態

MFC 會維護檔的已變更狀態。因此，當您關閉應用程式時，請關閉包含應用程式的最後一個視圖，或從 Windows 結束，MFC 會提示您儲存檔。Windows Forms 沒有對等的支援。

- 命令

MFC 具有命令的概念。功能表列、工具列和內容功能表都可以叫用相同的命令，例如，剪下和複製。在 Windows Forms 中，命令會從特定的 UI 元素(例如功能表項目)緊密系結事件；因此，您必須明確地連結所有命令事件。您也可以在 Windows Forms 中使用單一處理程式來處理多個事件。如需詳細資訊，請參閱[在 Windows Forms 中將多個事件連接到單一事件處理常式](#)。

- 命令路由

MFC 命令路由可讓使用中的 view 或 document 來處理命令。因為相同的命令通常對於不同的視圖具有不同的意義(例如，複製在文字編輯檢視中的行為不同于圖形編輯器)，所以必須由現用視圖來處理命令。因為 Windows Forms 功能表和工具列並不會對使用中的視圖有任何固有的瞭解，所以您的 MenuItem 的每個檢視類型不能有不同的處理常式。請按一下[事件]，而不需要撰寫額外的內部

- 命令更新機制

MFC 有一個命令更新機制。因此，即時檢視或檔會負責 UI 元素的狀態(例如，啟用或停用功能表項目或工具

按鈕，以及已核取的狀態）。Windows Forms 沒有對等的命令更新機制。

請參閱

[在 MFC 中使用 Windows Forms 使用者控制項](#)

將 Windows Form 使用者控制項裝載至 MFC 對話方塊中

2020/4/15 • [Edit Online](#)

MFC 將 Windows 窗體控制項模組託管為一種特殊的 ActiveX 控制項，並使用 [ControlActiveX](#) 介面以及 類的屬性和方法與控制元件通訊。我們建議您使用 .NET Framework 屬性和方法對控制項進行操作。

有關顯示與 MFC 一起使用的 Windows 窗體的範例應用程式，請參考[MFC 與 Windows 元件整合](#)。

NOTE

在目前的版本中，`CDialogBar` 物件無法承載 Windows 窗體控制件。

本節內容

[如何：建立使用者控制項並裝載至對話方塊中](#)

[如何：使用 Windows Form 執行 DDX/DDV 資料繫結](#)

[如何：從原生 C++ 類別接收 Windows Form 事件](#)

參考

[CWinForms 控制類](#) | [CDialog 類](#) | [Cwnd 類](#) | [Control](#)

另請參閱

[在 MFC 中使用 Windows Forms 使用者控制項](#)

[視窗表單/MFC 程式設計差異](#)

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

[將 Windows Form 使用者控制項裝載成 MFC 對話方塊](#)

HOW TO：在對話方塊中建立使用者控制項並裝載

2019/12/2 • [Edit Online](#)

這篇文章中的步驟假設您要建立對話方塊架構 ([CDialog 類別](#)) Microsoft Foundation Classes (MFC) 專案，但是您也可以將支援的 Windows Form 控制項加入至現有的 MFC 對話方塊。

若要建立.NET 使用者控制項

1. 建立 Visual C# Windows Form 控制項程式庫專案，名為 `WindowsFormsControlLibrary1`。

在 [檔案] 功能表上，按一下 [新增] 及 [專案]。在 Visual C# 資料夾中，選取 **Windows Forms 控制項程式庫**。

接受 `WindowsFormsControlLibrary1` 專案名稱，依序按一下 [確定]。

根據預設，.NET 控制項的名稱會是 `UserControl1`。

2. 加入子控制項 `UserControl1`。

在 工具箱，開啟所有的 Windows Form 清單。拖曳** 按鈕**若要控制 `UserControl1` 設計介面。

也加入 `TextBox` 控制項。

3. 在 方案總管，按兩下 `UserControl1.Designer.cs` 以開啟它進行編輯。變更文字方塊和按鈕的宣告
`private` 至 `public`。

4. 建置專案。

在 [建置] 功能表上，按一下 [建置方案]。

若要建立 MFC 主應用程式

1. 建立 MFC 應用程式專案。

在 [檔案] 功能表上，按一下 [新增] 及 [專案]。在 Visual C++ 資料夾中，選取 **MFC 應用程式**。

在 [名稱] 方塊中，輸入 `MFC01`。將方案設定變更為加入至方案。按一下 [確定]。

在 **MFC 應用程式精靈**，針對應用程式類型，選取 **採用對話方塊**。接受其餘的預設設定，然後按 **完成**。這會建立具有 MFC 對話方塊的 MFC 應用程式。

2. MFC 對話方塊中，加入預留位置控制項。

在 檢視功能表上，按一下 **資源檢視**。在 **資源檢視**，展開 **對話方塊** 資料夾，然後按兩下 `IDD_MFC01_DIALOG`。對話方塊資源隨即出現在 **資源編輯器**。

在 工具箱，開啟 **對話方塊編輯器** 清單。拖曳 **靜態文字** 對話方塊資源的控制。靜態文字控制項將會當做.NET Windows Forms 控制項的預留位置。其大小調整為大約 Windows Form 控制項的大小。

在 **屬性** 視窗中，變更識別碼的靜態文字若要控制 `IDC_CTRL1` 並變更 **TabStop** 屬性，則為 `true`。

3. 設定專案的 Common Language Runtime (CLR) 支援。

在 **方案總管**，以滑鼠右鍵按一下 `MFC01` 專案節點，然後按一下 **屬性**。

在 **屬性** 頁 **對話方塊** 的 **組態屬性**，選取 **一般**。在 **專案預設值** 區段中，將 **Common Language Runtime 支援**來 **Common Language Runtime 支援 (/clr)**。

底下 **組態屬性**，展開 **C/C++**，然後選取 **一般** 節點。設定 **偵錯資訊格式** 要 **程式資料庫 (/Zi)**。

選取 程式碼產生節點。設定啟用最少重建要否 (/ /gm-)。Ssprop_param_table_default 基本執行階段會檢查要預設。

按一下 確定 以套用變更。

4. 新增.NET 控制項的參考。

在 方案總管，以滑鼠右鍵按一下 MFC01 專案節點，然後按一下新增，參考。上屬性頁，按一下加入新參考，選取 WindowsFormsControlLibrary1 (下專案 索引標籤)，然後按一下確定。這會將參考新增的形式/FU 編譯器選項，如此將會編譯程式。它還可讓 WindowsFormsControlLibrary1.dll 的複本 \MFC01\ 專案資料夾中，讓程式得以執行。

5. 在 Stdafx.h 中尋找此行：

```
#endif // _AFX_NO_AFXCMN_SUPPORT
```

它的上方，新增下列行：

```
#include <afxwinforms.h> // MFC Windows Forms support
```

6. 加入建立 managed 的控制項的程式碼。

首先，宣告 managed 的控制項。在 MFC01Dlg.h 中，移至對話方塊類別的宣告，在 Protected 範圍內，加入使用者控制項的資料成員時，也將，如下所示。

```
class CMFC01Dlg : public CDialog
{
    ...
    // Data member for the .NET User Control:
    CWinFormsControl<WindowsFormsControlLibrary1::UserControl1> m_ctrl1;
```

接下來，提供 managed 控制項的實作。在 mfc01dlg.cpp 內，在對話方塊中的覆寫

CMFC01Dlg::DoDataExchange MFC 應用程式精靈所產生 (不 CAutoDlg::DoDataExchange，這是在相同的檔案)，新增下列程式碼，建立 managed 的控制項，並將它與靜態的預留位置 IDC_CTRL1 產生關聯。

```
void CMFC01Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_ManagedControl(pDX, IDC_CTRL1, m_ctrl1);
}
```

7. 建置並執行專案。

在 方案總管，以滑鼠右鍵按一下 MFC01，然後按一下 設定為啟始專案。

在 [建置] 功能表上，按一下 [建置方案]。

在 偵錯功能表上，按一下 啟動但不偵錯。在 MFC 對話方塊中應該會顯示在 Windows Form 控制項。

另請參閱

[將 Windows Forms 使用者控制項裝載至 MFC 對話方塊中](#)

如何：使用 Windows Form 執行 DDX/DDV 資料繫結

2020/11/2 • [Edit Online](#)

`DDX_ManagedControl` 會呼叫 `CWinFormsControl::CreateManagedControl` 來建立符合資源控制識別碼的控制項。如果您 `DDX_ManagedControl` `CWinFormsControl` 在 wizard 產生的程式碼中使用 (控制項，您就不應該 `CreateManagedControl` 針對相同的控制項明確呼叫。

`DDX_ManagedControl` 在 `CWnd` 中呼叫 `::DoDataExchange` 從資源識別碼建立控制項。針對資料交換，您不需要搭配 Windows Forms 控制項來使用 DDX/DDV 函數。相反地，您可以將程式碼放在 `DoDataExchange` 對話 (或 view) 類別的方法中，以存取 managed 控制項的屬性，如下列範例所示。

下列範例顯示如何將原生 C++ 字串系結至 .NET 使用者控制項。

範例：DDX/DDV 資料系結

以下範例是 `m_str` 使用 .NET 使用者控制項的使用者定義屬性之 MFC 字串的 DDX/DDV 資料系結 `NameText`。

當第一次 `CDialog::OnInitDialog` 呼叫時，會建立控制項 `CMyDlg::DoDataExchange`，因此任何參考的程式碼都 `m_UserControl` 必須在 `DDX_ManagedControl` 呼叫之後。

您可以在 [如何：在對話方塊中建立使用者控制項和主機] 中建立的 MFC01 應用程式中，執行此程式碼。

將下列程式碼放在 CMFC01Dlg 的宣告中：

```
class CMFC01Dlg : public CDialog
{
    CWinFormsControl<WindowsFormsControlLibrary1::UserControl1> m_MyControl;
    CString m_str;
};
```

範例：執行 DoDataExchange (# A1)

將下列程式碼放在 CMFC01Dlg 的執行中：

```
void CMFC01Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_ManagedControl(pDX, IDC_CTRL1, m_MyControl);

    if (pDX->m_bSaveAndValidate) {
        m_str = m_MyControl->textBox1->Text;
    } else {
        m_MyControl->textBox1->Text = gcnew System::String(m_str);
    }
}
```

範例：新增處理常式方法

現在，我們要新增處理常式方法，以按一下 [確定] 按鈕。按一下 [資源檢視] 索引標籤。在資源檢視中，按兩下 `IDD_MFC01_DIALOG`。對話方塊資源會出現在資源編輯器中。然後按兩下 [確定] 按鈕。

定義處理常式，如下所示。

```
void CMFC01Dlg::OnBnClickedOk()
{
    AfxMessageBox(CString(m_MyControl.GetControl()->textBox1->Text));
    OnOK();
}
```

範例：設定 textBox 文字

然後將下列這一行新增至 BOOL CMFC01Dlg::OnInitDialog (# A1 的實值。

```
m_MyControl.GetControl()->textBox1->Text = "hello";
```

您現在可以建置並執行應用程式。請注意，當應用程式關閉時，文字方塊中的任何文字都會顯示在快顯訊息方塊中。

另請參閱

[CWinFormsControl 類別](#)

[DDX_ManagedControl](#)

[CWnd::DoDataExchange](#)

HOW TO : 從原生接收 Windows Forms 事件C++類別

2019/12/2 • [Edit Online](#)

您可以啟用原生C++類別，以接收回撥，從受管理的Windows Form 控制項或其他使用MFC 巨集對應格式的形式從引發的事件。檢視和對話方塊中的接收事件是類似於在控制項相同的工作。

若要這樣做，您需要：

- 附加 `OnClick` 控制項使用的事件處理常式`MAKE_DELEGATE`。
- 建立委派對應 using `BEGIN_DELEGATE_MAP`, `END_DELEGATE_MAP`, 並`EVENT_DELEGATE_ENTRY`。

此範例會繼續在您執行的工作[How to:執行 DDX/DDV 資料繫結 Windows form](#)。

現在，您將建立MFC 控制項的關聯 (`m_MyControl`) 使用稱為「受管理的事件處理常式委派 `OnClick` managedClick 事件」。

若要連結的 `OnClick` 事件處理常式：

1. BOOL CMFC01Dlg::OnInitDialog 的實作中加入下列程式碼：

```
m_MyControl.GetControl()->button1->Click += MAKE_DELEGATE( System::EventHandler, OnClick );
```

2. 將下列程式碼新增至類別 CMFC01Dlg 宣告中的 public 區段：公用 CDialog。

```
// delegate map
BEGIN_DELEGATE_MAP( CMFC01Dlg )
EVENT_DELEGATE_ENTRY( OnClick, System::Object^, System::EventArgs^ )
END_DELEGATE_MAP()

void OnClick( System::Object^ sender, System::EventArgs^ e );
```

3. 最後，新增的實作 `OnClick` CMFC01Dlg.cpp 來：

```
void CMFC01Dlg::OnClick(System::Object^ sender, System::EventArgs^ e)
{
    AfxMessageBox(_T("Button clicked"));
}
```

另請參閱

[MAKE_DELEGATE](#)
[BEGIN_DELEGATE_MAP](#)
[END_DELEGATE_MAP](#)
[EVENT_DELEGATE_ENTRY](#)

將 Windows Form 使用者控制項裝載為 MFC 檢視

2019/12/2 • [Edit Online](#)

MFC 會使用 CWinFormsView 類別，在 MFC 視圖中裝載 Windows Forms 的使用者控制項。MFC Windows Forms views 是 ActiveX 控制項。使用者控制項會裝載為原生視圖的子系，並佔用原生視圖的整個工作區。

最後的結果會與[CFormView 類別](#)所使用的模型類似。這可讓您利用 Windows Forms 的設計工具和執行時間來建立以表單為基礎的豐富視圖。

因為 MFC Windows Forms views 是 ActiveX 控制項，所以沒有與 MFC views 相同的 `hwnd`。此外，它們也無法當做[CView](#)視圖的指標來傳遞。一般來說，請使用 .NET Framework 方法來處理 Windows Forms 的 views，並在 Win32 上依賴較少的。

如需顯示與 MFC 搭配使用之 Windows Forms 的範例應用程式，請參閱[mfc 和 Windows Forms 整合](#)。

本章節內容

[如何 : 建立使用者控制項並裝載 MDI 檢視](#)

[如何 : 新增命令傳送至 Windows Forms 控制項](#)

[如何 : 呼叫 Windows Forms 控制項的屬性和方法](#)

請參閱

[在 MFC 中使用 Windows Forms 使用者控制項](#)

[操作說明 : 撰寫複合控制項](#)

如何：建立使用者控制項並裝載 MDI 檢視

2020/4/15 • [Edit Online](#)

以下步驟演示如何創建 .NET Framework 使用者控件，在控件類庫中創作使用者控制件（特別是 Windows 控件庫專案），然後將專案編譯為程式集。然後，可以從使用從 [CView 類](#) 和 [CWinFormsView 類](#) 派生的類的 MFC 應用程式使用該控制項。

有關如何建立 Windows 表單使用者控制項和創作控制項類別的庫的資訊，請參考[如何：作者使用者控制件](#)。

NOTE

在某些情況下，Windows 表單控制項控制件（如第三方網格控件）在 MFC 應用程式中託管時可能無法可靠地運行。建議的解決方法是在 MFC 應用程式中放置 Windows 窗體使用者控件，並將第三方網格控件放在使用者控制項中。

此過程假定您創建了名為 WindowsFormsControlLibrary1 的 Windows 窗體控件庫專案，根據[「如何：在對話框中創建使用者控件和主機」](#)中的過程。

建立 MFC 主機應用程式

1. 創建 MFC 應用程式專案。

在「檔案」選單上，選擇「新建」，然後按下「專案」。在視覺 C++ 資料夾中，選擇 MFC 應用程式。

在「名稱」框 **MFC02** 中，輸入並變更「解決方案」設定以新增到解決方案。按一下 [確定]。

在 MFC 應用程式精靈中，接受所有預設值，然後單擊「完成」。。這將創建具有多個文檔介面的 MFC 應用程式。

2. 為通用語言運行時 (CLR) 支援配置專案。

在解決方案資源管理器中，右鍵單擊 **MFC01** 擊 專案節點，並從上下文菜單中選擇「屬性」。。將顯示「屬性頁」對話框。

在「設定屬性」下，選擇「一般」。在「項目預設值」部分下，將「通用語言運行時支援」設置為通用語言執行時支援 (/clr)。

在「設定屬性」下，展開 C/C++ 並單擊常規節點。將除錯資訊格式設定為程式資料庫 (/Zi)。

單擊代碼生成節點。將開啟最小重建設定為否 (/Gm-)。還將基本運行時檢查設置為「預設」。

按下「確定」以應用更改。

3. 在 *pch.h*(Visual Studio 2017 和更早版本中的 *stdafx.h*) 中，添加以下行：

```
#using <System.Windows.Forms.dll>
```

4. 添加對 .NET 控制項的引用。

在解決方案資源管理員中，右鍵按 **MFC02** 下專案節點並選擇「新增」。在「屬性頁」中，按一下「添加新參考」，選擇 WindowsFormsControlLibrary1（在「專案」選項卡下），然後單擊“確定”。這將以 /FU 編譯器選項的形式添加引用，以便程式將編譯；它還將 WindowsFormsControlLibrary1.dll **MFC02** 複製到 專案目錄中，以便程式將運行。

5. 在 *stdafx.h* 中，找到此行：

```
#endif // _AFX_NO_AFXCMN_SUPPORT
```

在上面新增以下行:

```
#include <afxwinforms.h> // MFC Windows Forms support
```

6. 修改檢視類,以便從[CWinFormsView](#)繼承。

在 MFC02View.h 中,將[CView](#)替換為[CWinFormsView](#),以便代碼如下所示:

```
class CMFC02View : public CWinFormsView
{
};
```

如果要向 MDI 應用程式添加其他檢視,則需要為創建的每個檢視調用[CWinApp::AddDocTemplate](#)。

7. 修改 MFC02View.cpp 檔,在IMPLEMENT_DYNCREATE宏和消息映射中將 CView 更改為 CWinFormsView,並將現有的空構造函數替換為如下所示的構造函數:

```
IMPLEMENT_DYNCREATE(CMFC02View, CWinFormsView)

CMFC02View::CMFC02View(): CWinFormsView(WindowsFormsControlLibrary1::UserControl1::typeid)
{
}
BEGIN_MESSAGE_MAP(CMFC02View, CWinFormsView)
//leave existing body as is
END_MESSAGE_MAP()
```

8. 建置並執行專案。

在解決方案資源管理器中,右鍵單擊 MFC02 並選擇「設置為啟動專案」。。

在 [建置]**** 功能表上,按一下 [建置方案]****。

在「調試」選單上,按一下「不調試即可開始」。

另請參閱

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

如何：新增命令傳送至 Windows Form 控制項

2020/4/15 • [Edit Online](#)

[CWinFormsView](#) 將指令和更新命令 UI 消息路由到使用者控制項，以允許它處理 MFC 命令（例如，框架功能表項和工具列按鈕）。

使用者控制項使用 [ICommandTarget::初始化](#) 來儲存對命令來源 `m_CmdSrc` 物件的引用，如下範例所示。要使用 `ICommandTarget`，必須添加對 `mfcmifc80.dll` 的引用。

[CWinFormsView](#) 通過將多個常見的 MFC 檢視通知轉發到託管使用者控件來處理它們。這些通知包括「[初始更新](#)」、「[更新](#)」和「[啟動檢視](#)」方法。

這個主題假定您以前已完成了「[如何操作：在對話框中建立使用者控制和主機](#)」以及「[如何：建立使用者控制和主機 MDI 檢視](#)」。

建立 MFC 主機應用程式

- 開啟您在「[如何操作](#)」中建立的 Windows 窗體控制件庫：[在對話框中建立使用者控制件和主機](#)。
- 新增對 `mfcmifc80.dll` 的引用，您可以通過右鍵單擊解決方案資源管理器中的專案節點，選擇「添加」，參考，然後流覽到 Microsoft Visual Studio 10.0\VC_atlmfc_lib。
- 開啟 `UserControl1.Designer.cs` 並添加以下使用語句：

```
using Microsoft.VisualStudio.MFC;
```

- 此外，在 `UserControl1.Designer.cs` 中，更改此行：

```
partial class UserControl1
```

變更為以下程式碼：

```
partial class UserControl1 : System.Windows.Forms.UserControl, ICommandTarget
```

- 將新增為 `UserControl1` 的類別定義的第一行：

```
private ICommandSource m_CmdSrc;
```

- 將以下方法定義加入 `UserControl1`（我們將在下一步中建立 MFC 控制件的 ID）：

```
public void Initialize (ICommandSource cmdSrc)
{
    m_CmdSrc = cmdSrc;
    // need ID of control in MFC dialog and callback function
    m_CmdSrc.AddCommandHandler(32771, new CommandHandler (singleMenuHandler));
}

private void singleMenuHandler (uint cmdUI)
{
    // User command handler code
    System.Windows.MessageBox.Show("Custom menu option was clicked.");
}
```

7. 開啟您在「如何建立」的 MFC 應用程式:建立使用者控制和主機 MDI 檢視。

8. 添加將調用 `singleMenuHandler` 的功能表選項。

跳到資源檢視(Ctrl_Shift_E),展開選單資料夾,然後按兩下IDR_MFC02TYPE。這將顯示功能表編輯器。

在「檢視」選單底部添加功能表選項。請注意「屬性」視窗中選單選項的 ID。儲存檔案。

在解決方案資源管理員中,打開 Resource.h 檔,複製剛剛新增的選單選項的 ID 值,並將該值

`m_CmdSrc.AddCommandHandler` 作為第一個參數貼上 `Initialize C# 32771` 專案方法中的呼叫(如有必要取代)。

9. 建置並執行專案。

在 [建置]**** 功能表上,按一下 [建置方案]****。

在「調試」選單上,按一下「不調試即可開始」。

選擇您添加的功能表選項。請注意,調用 .dll 中的方法。

另請參閱

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

[ICommandSource 介面](#)

[ICommandTarget 介面](#)

HOW TO : 呼叫控制項屬性和方法的 Windows Form

2019/12/2 • [Edit Online](#)

因為 `CWinFormsView::GetControl` 傳回的指標 `System.Windows.Forms.Control`, 並不是指標 `WindowsControlLibrary1::UserControl1`, 建議您新增的使用者控制項類型的成員, 並將它在初始化 `IView::OnInitialUpdate`. 現在您可以呼叫方法和屬性使用 `m_ViewControl`。

本主題假設您先前已完成 [How to: 在對話方塊中建立使用者控制項並裝載](#) 和 [How to: 建立使用者控制項並裝載 MDI 檢視](#)。

若要建立 MFC 主應用程式

- 開啟您建立的 MFC 應用程式 [How to: 建立使用者控制項並裝載 MDI 檢視](#)。
- 將下行新增到公用的覆寫區段 `CMFC02View` 類別在 `MFC02View.h` 中的宣告。

```
gcroot<WindowsFormsControlLibrary1::UserControl1 ^> m_ViewControl;
```

- 新增 `OnInitialupdate` 覆寫。

顯示屬性視窗 (F4)。在 **類別檢視** (CTRL + SHIFT + C), 選取 `CMFC02View` 類別。在 **[屬性]** 視窗中, 選取用於覆寫的圖示。往下到清單 `OnInitialUpdate` Scroll。按一下下拉式清單並選取 <新增>。在 `MFC02View.cpp`。請確定 `OnInitialUpdate` 函式的主體, 如下所示:

```
CWinFormsView::OnInitialUpdate();  
m_ViewControl = safe_cast<WindowsFormsControlLibrary1::UserControl1 ^>(this->GetControl());  
m_ViewControl->textBox1->Text = gcnew System::String("hi");
```

- 建置並執行專案。

在 **[建置]** 功能表上, 按一下 **[建置方案]**。

在 **偵錯** 功能表上, 按一下 **啟動但不偵錯**。

請注意, 文字方塊現在已初始化。

另請參閱

[將 Windows Form 使用者控制項裝載為 MFC 檢視](#)

將 Windows Form 使用者控制項裝載成 MFC 對話方塊

2019/12/2 • [Edit Online](#)

MFC 提供樣板類別 `CWinFormsDialog`，讓您可以在強制回應或無模式 MFC 對話方塊中裝載 Windows Forms 使用者控制項 (`UserControl`)。`CWinFormsDialog` 衍生自 MFC 類別 `CDialog`，因此可以將對話方塊啟動為強制回應或非強制回應。

`CWinFormsDialog` 用來裝載使用者控制項的程式，與在 [MFC 對話方塊中裝載 Windows Form 使用者控制項](#) 中所述的程式相同。不過，`CWinFormsDialog` 會管理使用者控制項的初始化和裝載，使其不必以手動方式進行程式設計。

如需顯示與 MFC 搭配使用之 Windows Forms 的範例應用程式，請參閱 [mfc 和 Windows Forms 整合](#)。

建立 MFC 主應用程式

1. 建立 MFC 應用程式專案。

在 [檔案] 功能表上，選取 [新增]，然後按一下 [專案]。在 [視覺C++效果] 資料夾中，選取 [MFC 應用程式]。

在 [名稱] 方塊中，輸入 `MFC03`，並將解決方案設定變更為 [加入方案]。按一下 [確定]。

在 MFC 應用程式精靈中，接受所有預設值，然後按一下 [完成]。這會建立具有多個檔介面的 MFC 應用程式。

2. 設定專案。

在方案總管中，以滑鼠右鍵按一下 MFC03 專案節點，然後選擇 [屬性]。[屬性頁] 對話方塊隨即出現。

在 [屬性頁] 對話方塊的 [設定屬性] 樹狀結構控制項中，選取 [一般]，然後在 [專案預設值] 區段中，將 [Common Language Runtime 支援] 設定為 [common language runtime 支援 (/clr)]。按一下 [確定]。

3. 加入 .NET 控制項的參考。

在方案總管中，以滑鼠右鍵按一下 MFC03 專案節點，然後選擇 [加入]、[參考]。在屬性頁中，按一下 [加入新參考]，選取 [WindowsControlLibrary1]（在 [專案] 索引標籤下），然後按一下 [確定]。這會以 `/FU` 編譯器選項的形式加入參考，讓程式進行編譯；它也會將 WindowsControlLibrary1 複製到 `MFC03` 專案目錄中，以便程式執行。

4. 在現有的 `#include` 語句結尾處，將 `#include <afxwinforms.h>` 新增至 `pch` (Visual Studio 2017 和更早版本中的 `stdafx.h`)。

5. 加入子類別 `CDialog` 的新類別。

以滑鼠右鍵按一下專案名稱，並加入子類別 `CDialog` 的 MFC 類別（稱為 `CHostForWinForm`）。由於您不需要對話方塊資源，您可以刪除資源識別碼（選取 [資源檢視]，展開對話方塊資料夾，然後刪除 [`IDD_HOSTFORWINFORM` 資源]）。然後，在程式碼中移除對識別碼的任何參考）。

6. 以 `CWinFormsDialog<WindowsControlLibrary1::UserControl1>` 取代 `CHostForWinForm` 中的 `CDialog` 和 `CHostForWinForm.cpp` 檔案。

7. 在 `CHostForWinForm` 類別上呼叫 `DoModal`。

在 `MFC03` 中，新增 `#include "HostForWinForm.h"`。

在 CMFC03App::InitInstance 定義中的 return 語句之前，加入：

```
CHostForWinForm m_HostForWinForm;
m_HostForWinForm.DoModal();
```

8. 建置並執行專案。

在 [建置] 功能表上，按一下 [建置方案]。

在 [調試] 功能表上，按一下 [啟動但不進行調試]。

接下來，您將加入程式碼，以從 MFC 應用程式監視 Windows Forms 上控制項的狀態。

9. 新增 OnInitDialog 的處理常式。

顯示 [屬性] 視窗(F4)。在類別檢視中，選取 [CHostForWinForm]。在 [屬性] 視窗中，選取 [覆寫]，並在 OnInitDialog 的資料列中按一下左側資料行，然後選取 [<新增>]。這會將下列這一行新增至 CHostForWinForm：

```
virtual BOOL OnInitDialog();
```

10. 定義 OnInitDialog (在 CHostForWinForm 中)，如下所示：

```
BOOL CHostForWinForm::OnInitDialog() {
    CWinFormsDialog<WindowsControlLibrary1::UserControl1>::OnInitDialog();
    GetControl()->button1->Click += MAKE_DELEGATE(System::EventHandler, OnButton1);
    return TRUE;
}
```

11. 接下來，新增 OnButton1 處理常式。將下列幾行新增至 CHostForWinForm 中 CHostForWinForm 類別的 public 區段：

```
virtual void OnButton1( System::Object^ sender, System::EventArgs^ e );

BEGIN_DELEGATE_MAP( CHostForWinForm )
    EVENT_DELEGATE_ENTRY( OnButton1, System::Object^, System::EventArgs^ );
END_DELEGATE_MAP()
```

在 CHostForWinForm 中，新增下列定義：

```
void CHostForWinForm::OnButton1( System::Object^ sender, System::EventArgs^ e )
{
    System::Windows::Forms::MessageBox::Show("test");
}
```

12. 建置並執行專案。當您按一下 [Windows] 表單上的按鈕時，MFC 應用程式中的程式碼就會執行。

接下來，您將新增程式碼，以便在 Windows Form 的文字方塊中，從 MFC 程式碼顯示值。

13. 在 CHostForWinForm 的 CHostForWinForm 類別的 public 區段中，新增下列宣告：

```
CString m_sEditBoxOnWinForm;
```

14. 在 CHostForWinForm 的 DoDataExchange 定義中，將下列三行新增至函式的結尾：

```
if (pDX->m_bSaveAndValidate)
    m_sEditBoxOnWinForm = CString( GetControl()->textBox1->Text);
else
    GetControl()->textBox1->Text = gcnew System::String(m_sEditBoxOnWinForm);
```

15. 在 CHostForWinForm 的 OnButton1 定義中，將下列三行新增至函式的結尾：

```
this->UpdateData(TRUE);
System::String ^ z = gcnew System::String(m_sEditBoxOnWinForm);
System::Windows::Forms::MessageBox::Show(z);
```

16. 建置並執行專案。

另請參閱

[在 MFC 中使用 Windows Form 使用者控制項System.Windows.Forms.UserControl](#)

從 Managed 程式碼呼叫原生函式

2020/4/15 • [Edit Online](#)

通用語言運行時提供平臺調用服務(PIInvoke),使託管代碼能夠在本機動態連結庫(DLL)中調用 C 樣式函數。相同的數據封送與 COM 與運行時的互通性以及「它只是工作」或 IJW 機制一樣。

如需詳細資訊, 請參閱

- [在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)
- [使用 C++ Interop \(隱含 PInvoke\)](#)

目前目前設定的範例僅說明如何 `PInvoke` 使用。`PInvoke` 可以簡化自定義的數據封送,因為您在屬性中以聲明性方式提供封送資訊,而不是編寫過程封送代碼。

NOTE

封送庫提供了一種以優化方式在本機環境和託管環境之間封送數據的替代方法。有關封送庫的詳細資訊,請參閱[C++ 中的封送概述](#)。封送庫僅可用於數據,不適用於函數。

PIvoke 與 Dll 匯入屬性

下面的範例顯示可檢視C++程式中的使用 `PInvoke`。本機函數放置在 `msvcrt.dll` 中定義。`DllImport` 屬性用於報放。

```
// platform_invocation_services.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[DllImport("msvcrt", CharSet=CharSet::Ansi)]
extern "C" int puts(String ^);

int main() {
    String ^ pStr = "Hello World!";
    puts(pStr);
}
```

以下示例等效於前面的示例,但使用IJW。

```
// platform_invocation_services_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#include <stdio.h>

int main() {
    String ^ pStr = "Hello World!";
    char* pChars = (char*)Marshal::StringToHGlobalAnsi(pStr).ToPointer();
    puts(pChars);

    Marshal::FreeHGlobal((IntPtr)pChars);
}
```

IJW 的優勢

- 無需為程式使用的非託管 `DLLImport` API 編寫屬性聲明。只需包括標頭檔和與導入庫的連結。
- IJW 機制稍微快一些(例如,IJW 存根不需要檢查是否需要固定或複製數據項,因為開發人員會顯式執行)。
- 它清楚地說明瞭性能問題。在這種情況下,您正在從 Unicode 字串轉換為 ANSI 字串,並且具有相應的記憶體分配和處理。在這種情況下,使用 IJW 編寫代碼的開發人員會意識到 `_putws` 調用 `PtrToStringChars` 和使用 將更具有性能。
- 如果使用相同的數據調用許多非託管 API,則將其封送一次並傳遞封送副本比每次重新封送效率高得多。

IJW的缺點

- 封送必須在代碼中顯式指定,而不是按屬性(通常具有適當的預設值)指定。
- 封送代碼是內聯的,在應用程式邏輯的流中它更具侵入性。
- 由於顯式封送 API `IntPtr` 返回 32 位元到 64 位 `ToPointer` 元可移植性的類型,因此必須 使用額外的調用。

C++公開的特定方法是更高效、更明確的方法,但代價是一些額外的複雜性。

如果應用程式主要使用非託管數據類型,或者調用比 .NET 框架 API 更多的非託管 API,我們建議您使用 IJW 功能。要在大多數託管應用程式中調用偶爾的非託管 API,選擇更為微妙。

使用 Windows API 呼叫

PInvoke 便於在 Windows 中調用函數。

在此示例中,可視化C++程式與作為 Win32 API 一部分的 MessageBox 函數交互。

```
// platform_invocation_services_4.cpp
// compile with: /clr /c
using namespace System;
using namespace System::Runtime::InteropServices;
typedef void* HWND;
[DllImport("user32", CharSet=CharSet::Ansi]
extern "C" int MessageBox(HWND hWnd, String ^ pText, String ^ pCaption, unsigned int uType);

int main() {
    String ^ pText = "Hello World!";
    String ^ pCaption = "PInvoke Test";
    MessageBox(0, pText, pCaption, 0);
}
```

輸出是一個消息框,標題為 PInvoke 測試,並包含文本 Hello World!

PInvoke 還使用封送資訊來查找 DLL 中的函數。在 user32.dll 中,實際上沒有 MessageBox 函數,但 CharSet_CharSet::Ansi 使 PInvoke 能夠使用 MessageBoxA(ANSI 版本)而不是 MessageBoxW(即 Unicode 版本)。通常,我們建議您使用非託管 API 的 Unicode 版本,因為這消除了從 .NET Framework 字串物件的本機 Unicode 格式到 ANSI 的轉換花費。

何時不使用 PInvoke

使用 PInvoke 不適用於 DLL 中的所有 C 樣式函數。例如,假設在 mylib.dll 中聲明了一個函數「使特殊」,如下所示:

```
char * MakeSpecial(char * pszString);
```

如果我們在 Visual C++ 應用程式中使用 PInvoke,我們可能會編寫類似於以下內容的內容:

```
[DllImport("mylib")]
extern "C" String * MakeSpecial([MarshalAs(UnmanagedType::LPStr)] String ^);
```

這裡的困難是,我們不能刪除 MakeSpecial 返回的非託管字串的記憶體。通過 PInvoke 調用的其他函數返回指向內部緩衝區的指標,該指標不必由使用者處理。在這種情況下,使用 IJW 功能是明顯的選擇。

PInvoke 的限制

不能從作為參數的本機函數返回完全相同的指標。如果本機函數返回已由 PInvoke 封送到它的指標,則記憶體損壞和異常可能會隨之而來。

```
__declspec(dllexport)
char* fstringA(char* param) {
    return param;
}
```

下面的示例顯示了此問題,即使程式似乎給出了正確的輸出,輸出來自己釋放的記憶體。

```
// platform_invocation_services_5.cpp
// compile with: /clr /c
using namespace System;
using namespace System::Runtime::InteropServices;
#include <limits.h>

ref struct MyPInvokeWrap {
public:
    [DllImport("user32.dll", EntryPoint = "CharLower", CharSet = CharSet::Ansi) ]
    static String^ CharLower([In, Out] String ^);
};

int main() {
    String ^ strout = "AabCc";
    Console::WriteLine(strout);
    strout = MyPInvokeWrap::CharLower(strout);
    Console::WriteLine(strout);
}
```

封送參數

使用 `PInvoke` 時,在託管和 C++ 具有相同窗體的本機基元類型之間不需要封送。例如,Int32 和 int 之間或 Double 和 Double 之間不需要封送。

但是,必須對具有相同窗體的類型進行封送。這包括字元、字串和結構類型。下表顯示了封送器用於各種類型的對應:

WTYPES.H	VISUAL C++	I / CLR IIfC++	COMMON LANGUAGE RUNTIME
HANDLE	無效*	無效*	IntPtr, UIntPtr
BYTE	unsigned char	unsigned char	Byte
SHORT	short	short	Int16
WORD	unsigned short	unsigned short	UInt16

WTYPES.H	VISUAL C++	I / CLR IIC++	COMMON LANGUAGE RUNTIME
INT	int	int	Int32
UINT	不帶正負號的整數	不帶正負號的整數	UInt32
LONG	long	long	Int32
BOOL	long	bool	Boolean
DWORD	unsigned long	unsigned long	UInt32
ULONG	unsigned long	unsigned long	UInt32
CHAR	char	char	Char
LPSTR	字元*	字串 = 輸入,字串產生器 = [輸入、出]	字串 = 輸入,字串產生器 = [輸入、出]
LPCSTR	康斯特字元*	字串	String
LPWSTR	wchar_t *	字串 = 輸入,字串產生器 = [輸入、出]	字串 = 輸入,字串產生器 = [輸入、出]
LPCWSTR	康斯特wchar_t*	字串	String
FLOAT	FLOAT	FLOAT	Single
DOUBLE	double	double	Double

如果伺服器的地址傳遞給非託管函數,則封送器會自動將分配給運行時堆的記憶體固定。固定可防止垃圾回收器在壓縮過程中移動分配的記憶體塊。

在本主題前面所示的示例中,DllImport 的 CharSet 參數指定如何封送託管字串;否則,應如何對託管字串進行封送處理。在這種情況下,它們應被封送到本機端的 ANSI 字串。

可以使用 MarshalAs 屬性為本機函數的各個參數指定封送資訊。對字串*參數進行封送有多種選擇:BStr、ANSIBStr、TBStr、LPStr、LPWStr 和 LPTStr。默認值為 LPStr。

在此範例中,字串作為雙位元組 Unicode 字串 LPWStr 進行封送。輸出是你好世界的第一個字母! 因為封送字串的第二個字節為空,並且將此解釋為字串結尾標記。

```
// platform_invocation_services_3.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[DllImport("msvcrt", EntryPoint="puts")]
extern "C" int puts([MarshalAs(UnmanagedType::LPWStr)] String ^);

int main() {
    String ^ pStr = "Hello World!";
    puts(pStr);
}
```

MarshalAs 屬性位於系統::運行時::互通服務命名空間。該屬性可與其他數據類型(如陣列)一起使用。

如本主題前面所述,封送庫提供了一種新的優化數據在本機環境和託管環境之間封送的方法。有關詳細資訊,請參閱[C++ 中的封送概述](#)。

效能考量

PInvoke 的開銷介於每個呼叫 10 到 30 x86 指令之間。除了此固定成本外,封送還會產生額外的開銷。在託管代碼和非託管代碼中具有相同的表示形式的可聲明類型之間沒有封送成本。例如,int 和 Int32 之間沒有翻譯費用。

為了更好的性能,減少 PInvoke 調用,以盡可能多地封送數據,而不是使用更多調用來為每個調用封送更少的數據。

另請參閱

[本機與 .NET 互通性](#)

在 C++ 中使用明確的 PInvoke (DllImport 屬性)

2019/12/2 • [Edit Online](#)

.NET Framework 會提供明確的平台叫用 (或 PInvoke) 功能與 `DllImport` 屬性，以允許呼叫 unmanaged 函式的 DLL 內封裝的 managed 應用程式。需要的情況下，需要 unmanaged 的 API 會封裝為 DLL 原始程式碼沒有明確的 PInvoke。呼叫 Win32 函式，例如，需要 PInvoke。否則，請使用隱含的 P {叫用；請參閱[使用C++ Interop \(隱含 PInvoke\)](#)如需詳細資訊。

PInvoke 的運作方式是使用 `DllImportAttribute`。這個屬性，會使用第一個引數為 DLL 的名稱，被放在將用於每個 DLL 進入點函式宣告之前。函式的簽章必須符合由 DLL 匯出的函式的名稱 (但可以隱含地執行一些類型轉換，藉由定義 `DllImport` 根據 managed 類型的宣告。)

結果是必要的轉換程式碼 (或 thunk) 包含每個原生 DLL 函式的 managed 的進入點和簡單資料轉換。透過這些進入點 dll，就可以呼叫 managed 函式。完全受控的 PInvoke 結果插入模組的程式碼。

本節內容

- [從 Managed 程式碼呼叫原生函式](#)
- [如何：使用 PInvoke 從受控程式碼呼叫原生 DLL](#)
- [如何：使用 PInvoke 封送處理字串](#)
- [如何：使用 PInvoke 封送處理結構](#)
- [如何：使用 PInvoke 封送處理陣列](#)
- [如何：使用 PInvoke 封送處理函式指標](#)
- [如何：使用 PInvoke 封送處理內嵌指標](#)

另請參閱

[從 Managed 程式碼呼叫原生函式](#)

如何：使用 P/Invoke 從 Managed 程式碼呼叫原生 DLL

2019/12/10 • [Edit Online](#)

您可以使用平台叫用(P/Invoke)功能，從 managed 程式碼呼叫未受管理的 DLL 中所實作用的函式。如果 DLL 的原始程式碼無法使用，P/Invoke 是唯一可進行交互操作的選項。不過，不同于其他 .NET 語言，C++ Visual 提供 P/Invoke 的替代方案。如需詳細資訊，請參閱[使用C++ Interop \(隱含 P/Invoke\)](#)。

範例

下列程式碼範例會使用 Win32 [GetSystemMetrics](#)函式來抓取螢幕的目前解析度(以圖元為單位)。

針對只使用內建類型做為引數和傳回值的函式，不需要額外的工作。其他資料類型(例如函式指標、陣列和結構)則需要額外的屬性，以確保正確的資料封送處理。

雖然並非必要，但最好是讓 P/Invoke 告知成為實值類別的靜態成員，使其不存在於全域命名空間中，如下列範例所示。

```
// pinvoke_basic.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value class Win32 {
public:
    [DllImport("User32.dll")]
    static int GetSystemMetrics(int);

    enum class SystemMetricIndex {
        // Same values as those defined in winuser.h.
        SM_CXSCREEN = 0,
        SM_CYSCREEN = 1
    };
};

int main() {
    int hRes = Win32::GetSystemMetrics( safe_cast<int>(Win32::SystemMetricIndex::SM_CXSCREEN) );
    int vRes = Win32::GetSystemMetrics( safe_cast<int>(Win32::SystemMetricIndex::SM_CYSCREEN) );
    Console::WriteLine("screen resolution: {0},{1}", hRes, vRes);
}
```

請參閱

[在 C++ 中使用明確的 P/Invoke \(DllImport 屬性\)](#)

如何：使用 P/Invoke 封送處理字串

2019/12/10 • [Edit Online](#)

本主題說明如何使用 CLR 字串類型 `System::String` 來呼叫接受 C 樣式字串的原生函式，以 .NET Framework 平台叫用支援。建議 C++ Visual 程式設計人員改用 C++ Interop 功能（可能的話），因為 P/Invoke 提供的編譯階段錯誤報表很少，而且不是型別安全，而且可能很繁瑣。如果非受控 API 封裝為 DLL，而原始程式碼無法使用，則 P/Invoke 是唯一的選項，否則請參閱[使用 C++ Interop \(隱含 P/Invoke\)](#)。

受控和非受控字串在記憶體中的配置方式不同，因此將字串從 managed 傳遞至非受控函式需要 `MarshalAsAttribute` 屬性來指示編譯器插入必要的轉換機制，以便正確且安全地封送處理字串資料。

就像只使用內建資料類型的函式一樣，`DllImportAttribute` 是用來將 managed 進入點宣告為原生函式，但--用於傳遞字串，而不是將這些進入點定義為採用 C 樣式字串，而是可以改用 `String` 類型的控制碼。這會提示編譯器插入執行必要轉換的程式碼。對於接受字串之非受控函式中的每個函式引數，應該使用 `MarshalAsAttribute` 屬性來指示字串物件應該以 C 樣式字串的形式封送處理至原生函式。

封送處理器會將非受控函式的呼叫包裝在隱藏的包裝函式常式中，將 managed 字串釘選並複製到非受控內容中的本機配置字串，然後再傳遞給非受控函式。當非受控函式傳回時，包裝函式會刪除資源，如果它是在堆疊上，則會在包裝函式超出範圍時回收。非受控函式不負責此記憶體。未受管理的程式碼只會建立和刪除由它自己的 CRT 所設定之堆積中的記憶體，因此使用不同 CRT 版本的封送處理器並不會有任何問題。

如果您的非受控函式傳回字串（做為傳回值或 `out` 參數），封送處理器會將它複製到新的 managed 字串，然後釋放記憶體。如需詳細資訊，請參閱[預設封送處理行為](#)和[使用平台叫用封送處理資料](#)。

範例

下列程式碼是由不受管理的和受控模組所組成。非受控模組是定義名為 `TakesAString` 之函式的 DLL，其接受以 `char*` 格式表示的 C 樣式 ANSI 字串。受控模組是一種命令列應用程式，它會匯入 `TakesAString` 函式，但會將它定義為接受 managed `System.String`，而不是 `char*`。`MarshalAsAttribute` 屬性是用來指示呼叫 `TakesAString` 時，managed 字串應該如何封送處理。

```
// TraditionalDll2.cpp
// compile with: /LD /EHsc
#include <windows.h>
#include <stdio.h>
#include <iostream>

using namespace std;

#define TRADITIONALDLL_EXPORTS
#ifdef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    TRADITIONALDLL_API void TakesAString(char*);
}

void TakesAString(char* p) {
    printf_s("[unmanaged] %s\n", p);
}
```

```
// MarshalString.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value struct TraditionalDLL
{
    [DllImport("TraditionalDLL2.dll")]
    static public void
    TakesAString([MarshalAs(UnmanagedType::LPStr)]String^);
};

int main() {
    String^ s = gcnew String("sample string");
    Console::WriteLine("[managed] passing managed string to unmanaged function...");
    TraditionalDLL::TakesAString(s);
    Console::WriteLine("[managed] {0}", s);
}
```

這項技術會導致在非受控堆積上建立字串的複本，因此原生函式對字串所做的變更將不會反映在字串的 managed 複本中。

請注意，不會透過傳統的 #include 指示詞，將 DLL 的任何部分公開給 managed 程式碼。事實上，DLL 只能在執行時間存取，因此使用 `DllImport` 匯入的函式會在編譯時期偵測到問題。

請參閱

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

如何：使用 P/Invoke 封送處理結構

2020/11/2 • [Edit Online](#)

本檔說明如何使用 P/Invoke，從 managed 函式呼叫接受 C 樣式結構的原生函式。雖然我們建議您使用 c + + Interop 功能，而不是 P/Invoke，因為 P/Invoke 提供的編譯時期錯誤報表很少，而且不是型別安全，而且可能相當繁瑣，如果未受管理的 API 封裝為 DLL 且無法使用原始程式碼，則 P/Invoke 是唯一的選項。否則，請參閱下列檔：

- [使用 c + + Interop \(隱含 P/Invoke\)](#)
- [如何：使用 P/Invoke 封送處理字串](#)

根據預設，原生和 managed 結構在記憶體中的配置方式不同，因此在受控/非受控界限之間成功傳遞結構需要額外的步驟，以保留資料的完整性。

本檔說明定義原生結構的受控對等專案所需的步驟，以及如何將產生的結構傳遞至非受控函數。本檔假設使用簡單結構（不含字串或指標的結構）。如需非後端互通性的詳細資訊，請參閱 [使用 c + + Interop \(隱含的 P/Invoke\)](#)。P/Invoke 不能有非類型的型別做為傳回值。在受控和非受控程式碼中，可集中式的類型具有相同的標記法。如需詳細資訊，[請參閱我的全像](#)

封送處理跨受控/非受控界限的簡單、可大量管理的結構，首先需要定義每個原生結構的受控版本。這些結構可以具有任何法定名稱；除了資料版面配置以外，這兩個結構的原生和 managed 版本之間沒有任何關聯性。因此，受控版本必須包含大小相同且順序與原生版本相同的欄位，是很重要的。（沒有機制可確保 managed 和原生版本的結構都相等，所以在執行時間之前不會明顯發生不相容的情況。程式設計人員必須負責確保兩個結構具有相同的資料版面配置。）

由於 managed 結構的成員有時候會基於效能考慮而重新排列，因此必須使用 [StructLayoutAttribute](#) 屬性來指出結構是依序排列的。將結構封裝設定明確設定為與原生結構所使用的相同，也是個不錯的主意。（雖然預設 Visual C++ 同時針對兩個 managed 程式碼使用 8 位元組結構封裝。）

1. 接下來，使用 [DllImportAttribute](#) 來宣告與接受結構的任何未受管理函式對應的進入點，但在函式簽章中使用結構的 managed 版本，如果您針對兩個版本的結構使用相同的名稱，則為想法點。
2. 現在 managed 程式碼可以將 managed 版本的結構傳遞至非受控函式，就好像它們實際上是 managed 函數一樣。這些結構可以透過傳值方式或傳址方式傳遞，如下列範例所示。

非受控和受控模組

下列程式碼包含非受控和受控模組。非受控模組是一個 DLL，它會定義名為 Location 的結構，以及一個名為 GetDistance 的函式，以接受位置結構的兩個實例。第二個模組是受管理的命令列應用程式，它會匯入 GetDistance 函式，但會根據位置結構 MLocation 的受控對等專案來定義它。在實務上，這兩個版本的結構可能會使用相同的名稱。不過，此處會使用不同的名稱來示範如何根據受控版本來定義 DllImport 原型。

請注意，使用傳統的 #include 指示詞不會對 managed 程式碼公開任何部分的 DLL。事實上，DLL 只會在執行時間存取，因此在編譯時期不會偵測到以 DllImport 匯入之函式的問題。

範例：非受控 DLL 模組

```
// TraditionalDll3.cpp
// compile with: /LD /EHsc
#include <iostream>
#include <stdio.h>
#include <math.h>

#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

#pragma pack(push, 8)
struct Location {
    int x;
    int y;
};
#pragma pack(pop)

extern "C" {
    TRADITIONALDLL_API double GetDistance(Location, Location);
    TRADITIONALDLL_API void InitLocation(Location*);
}

double GetDistance(Location loc1, Location loc2) {
    printf_s("[unmanaged] loc1(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2(%d,%d)\n", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y = loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    printf_s("[unmanaged] Initializing location...\n");
    lp->x = 50;
    lp->y = 50;
}
```

範例：Managed 命令列應用程式模組

```
// MarshalStruct_pi.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, Pack=8)]
value struct MLocation {
    int x;
    int y;
};

value struct TraditionalDLL {
    [DllImport("TraditionalDLL3.dll")]
    static public double GetDistance(MLocation, MLocation);
    [DllImport("TraditionalDLL3.dll")]
    static public double InitLocation(MLocation*);
};

int main() {
    MLocation loc1;
    loc1.x = 0;
    loc1.y = 0;

    MLocation loc2;
    loc2.x = 100;
    loc2.y = 100;

    double dist = TraditionalDLL::GetDistance(loc1, loc2);
    Console::WriteLine("[managed] distance = {0}", dist);

    MLocation loc3;
    TraditionalDLL::InitLocation(&loc3);
    Console::WriteLine("[managed] x={0} y={1}", loc3.x, loc3.y);
}
```

```
[unmanaged] loc1(0,0) loc2(100,100)
[managed] distance = 141.42135623731
[unmanaged] Initializing location...
[managed] x=50 y=50
```

另請參閱

[在 c++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

HOW TO : 封送處理陣列使用 PInvoke

2019/12/2 • [Edit Online](#)

本主題說明如何在原生函式會接受 C 樣式字串可使用 CLR 字串型別呼叫[String](#)使用.NET Framework 平台叫用的支援。視覺化C++ 程式設計人員是鼓勵使用C+++Interop 功能，而是（如果可能）因為 P/Invoke 提供報告，不是類型安全，並可能會非常繁瑣實作小小的編譯時期錯誤。如果未受管理的 API 會封裝成 DLL，而且沒有可用的原始程式碼，P/Invoke 是唯一的選項（否則請參閱[使用C+++Interop \(隱含 PInvoke\)](#)）。

範例

原生和 managed 陣列是以不同的方式在記憶體中配置，因為跨 managed/unmanaged 界限成功傳遞需要轉換，或封送處理。本主題示範如何簡單 (blitable) 項目的陣列可以傳遞至原生函式從 managed 程式碼。

如為 true 的 managed/unmanaged 資料封送處理一般情況下，[DllImportAttribute](#)屬性用來建立將用於每個原生函式的 managed 的進入點。在採用陣列當做引數，函式的情況下[MarshalAsAttribute](#)屬性必須也用來指定編譯器如何將資料會被封送處理。在下列範例中，[UnmanagedType](#)列舉型別用來表示，managed 的陣列會封送處理為 C 樣式陣列。

下列程式碼是由 unmanaged 和 managed 的模組所組成。未受管理的模組會定義接受整數陣列的函式的 DLL。第二個模組是受管理的命令列應用程式匯入此函式，但定義方面的受管理的陣列，並使用[MarshalAsAttribute](#)屬性來指定陣列，應該轉換成原生陣列時呼叫。

受管理的模組是使用 /clr 所編譯。

```
// TraditionalDll4.cpp
// compile with: /LD /EHsc
#include <iostream>

#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    TRADITIONALDLL_API void TakesAnArray(int len, int[]);
}

void TakesAnArray(int len, int a[]) {
    printf_s("[unmanaged]\n");
    for (int i=0; i<len; i++)
        printf("%d = %d\n", i, a[i]);
}
```

```
// MarshalBlitArray.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

value struct TraditionalDLL {
    [DllImport("TraditionalDLL4.dll")]
    static public void TakesAnArray(
        int len,[MarshalAs(UnmanagedType::LPArray)]array<int>^);
};

int main() {
    array<int>^ b = gcnew array<int>(3);
    b[0] = 11;
    b[1] = 33;
    b[2] = 55;
    TraditionalDLL::TakesAnArray(3, b);

    Console::WriteLine("[managed]");
    for (int i=0; i<3; i++)
        Console::WriteLine("{0} = {1}", i, b[i]);
}
```

請注意沒有部分的 dll 會公開給 managed 程式碼，透過傳統 #include 指示詞。事實上，DLL 會在執行階段只存取，因此問題函式匯入與[DllImportAttribute](#)將不會在編譯時期偵測。

另請參閱

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

HOW TO : 封送處理函式指標使用 P/Invoke

2019/12/2 • [Edit Online](#)

本主題說明如何在受管理的委派時與相互操作 unmanaged 函式使用.NET Framework P/Invoke 功能可用來取代函式指標。不過，VisualC++ 程式設計人員是鼓勵使用C++ Interop 功能，而是（如果可能）因為 P/Invoke 提供報告，不是類型安全，並可能會非常繁瑣實作小小的編譯時期錯誤。如果未受管理的 API 會封裝成 DLL，而且沒有可用的原始程式碼，P/Invoke 就會是唯一的選項。否則，請參閱下列主題：

- [使用 C++ Interop \(隱含 P/Invoke\)](#)
- [如何：使用 C++ Interop 封送處理回呼和委派](#)

Unmanaged 的 API，稱為 引數可以是從 managed 程式碼取代原生函式指標的受管理的委派，採用函式指標。編譯器會自動封送處理至 unmanaged 函式的委派函式指標，並將必要的 managed/unmanaged 轉換程式碼插入。

範例

下列程式碼是由 unmanaged 和 managed 的模組所組成。未受管理的模組會定義稱為 TakesCallback 接受函式指標的函式的 DLL。此位址用來執行函式。

受管理的模組定義委派，其中會封送處理函式指標的原生程式碼，並使用 [DllImportAttribute](#) 公開給 managed 程式碼的原生 TakesCallback 函式的屬性。在主要函數中，是建立委派的執行個體，並將其傳遞給 TakesCallback 函式中。程式輸出示範執行此函式時，取得原生 TakesCallback 函式。

Managed 函式會隱藏的受管理的委派，以避免.NET Framework 記憶體回收原生函式執行時，重新放置委派記憶體回收。

```
// TraditionalDll5.cpp
// compile with: /LD /EHsc
#include <iostream>
#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

extern "C" {
    /* Declare an unmanaged function type that takes two int arguments
       Note the use of __stdcall for compatibility with managed code */
    typedef int (__stdcall *CALLBACK)(int);
    TRADITIONALDLL_API int TakesCallback(CALLBACK fp, int);
}

int TakesCallback(CALLBACK fp, int n) {
    printf_s("[unmanaged] got callback address, calling it...\n");
    return fp(n);
}
```

```
// MarshalDelegate.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

public delegate int GetTheAnswerDelegate(int);
public value struct TraditionalDLL {
    [DllImport("TraditionalDLL5.dll")]
    static public int TakesCallback(GetTheAnswerDelegate^ pfn, int n);
};

int GetNumber(int n) {
    Console::WriteLine("[managed] callback!");
    static int x = 0;
    ++x;
    return x + n;
}

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);
    pin_ptr<GetTheAnswerDelegate^> pp = &fp;
    Console::WriteLine("[managed] sending delegate as callback...");

    int answer = TraditionalDLL::TakesCallback(fp, 42);
}
```

請注意沒有部分的 dll 會使用傳統的 managed 程式碼以 #include 指示詞。事實上，DLL 會在執行階段存取，因此問題函式匯入與[DllImportAttribute](#)將不會在編譯時期偵測。

另請參閱

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

HOW TO : 封送處理內嵌指標使用 P/Invoke

2019/12/2 • [Edit Online](#)

從 managed 程式碼使用平台叫用 (P/Invoke) 的功能，可以呼叫 unmanaged DLL 中實作的函式。如果無法使用 DLL 的原始程式碼，P/Invoke 是交互操作的唯一選項。不過，不同於其他.NET 語言，視覺效果 C++ 提供 P/Invoke 的替代方案。如需詳細資訊，請參閱 < [使用 C++ Interop \(隱含 P/Invoke\)](#) 並 [How to: 封送處理內嵌指標使用 C++ Interop](#)。

範例

將結構傳遞至原生程式碼需要，會建立受管理的結構，等於是根據資料配置的原生結構。不過，結構，其中包含指標需要特殊處理。對於原生結構中每個內嵌指標，結構的 managed 的版本應該包含的執行個體 `IntPtr` 型別。此外，記憶體為這些執行個體都必須明確配置，初始化，而且使用 `AllocCoTaskMem`、`StructureToPtr`，和 `FreeCoTaskMem` 方法。

下列程式碼是由 unmanaged 和 managed 的模組所組成。未受管理的模組是定義接受結構，稱為 `ListString` 包含指標的函式和呼叫 `TakesListStruct` 函式的 DLL。受管理的模組是命令列應用程式匯入 `TakesListStruct` 函式，並定義結構，稱為 `MListStruct`，相當於原生 `ListStruct` 不同之處在於雙精度浮點數 * 代表與 `IntPtr` 執行個體。在呼叫前 `TakesListStruct`，`main` 函式配置，並初始化此欄位會參考的記憶體。

```
// TraditionalDll6.cpp
// compile with: /EHsc /LD
#include <stdio.h>
#include <iostream>
#define TRADITIONALDLL_EXPORTS
#ifndef TRADITIONALDLL_EXPORTS
#define TRADITIONALDLL_API __declspec(dllexport)
#else
#define TRADITIONALDLL_API __declspec(dllimport)
#endif

#pragma pack(push, 8)
struct ListStruct {
    int count;
    double* item;
};
#pragma pack(pop)

extern "C" {
    TRADITIONALDLL_API void TakesListStruct(ListStruct);
}

void TakesListStruct(ListStruct list) {
    printf_s("[unmanaged] count = %d\n", list.count);
    for (int i=0; i<list.count; i++)
        printf_s("array[%d] = %f\n", i, list.item[i]);
}
```

```

// EmbeddedPointerMarshalling.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, Pack=8)]
value struct MListStruct {
    int count;
    IntPtr item;
};

value struct TraditionalDLL {
    [DllImport("TraditionalDLL6.dll")]
    static public void TakesListStruct(MListStruct);
};

int main() {
    array<double>^ parray = gcnew array<double>(10);
    Console::WriteLine("[managed] count = {0}", parray->Length);

    Random^ r = gcnew Random();
    for (int i=0; i<parray->Length; i++) {
        parray[i] = r->NextDouble() * 100.0;
        Console::WriteLine("array[{0}] = {1}", i, parray[i]);
    }

    int size = Marshal::SizeOf(double::typeid);
    MListStruct list;
    list.count = parray->Length;
    list.item = Marshal::AllocCoTaskMem(size * parray->Length);

    for (int i=0; i<parray->Length; i++) {
        IntPtr t = IntPtr(list.item.ToInt32() + i * size);
        Marshal::StructureToPtr(parray[i], t, false);
    }

    TraditionalDLL::TakesListStruct( list );
    Marshal::FreeCoTaskMem(list.item);
}

```

請注意沒有部分的 dll 會使用傳統的 managed 程式碼以 #include 指示詞。事實上，DLL 會在執行階段存取，因此問題函式匯入與[DllImportAttribute](#)將不會在編譯時期偵測。

另請參閱

[在 C++ 中使用明確的 PInvoke \(DllImport 屬性\)](#)

使用 C++ Interop (隱含 PInvoke)

2020/1/8 • [Edit Online](#)

不同于其他 .NET 語言，C++ 視覺效果具有互通性支援，可讓受控和非受控程式碼存在於相同的應用程式中，甚至是在同一個檔案中（使用[受控、未受管理的 pragma](#)）。這可讓 C++ 視覺效果開發人員將 .net 功能整合到現有的視覺效果應用程式，而不會干擾應用程式的其餘部分。

您也可以使用 `dllexport dllimport`，從 managed 編譯模組呼叫非受控函式。

當您不需要指定如何封送處理函式參數，或在明確呼叫 `DllImportAttribute` 時指定的任何其他詳細資料時，隱含的 `PInvoke` 就很有用。

[C++ 視覺效果] 提供兩種方式，讓 managed 和非受控函式相交互操作：

- 在 C++ 中使用明確的 `PInvoke` (`DllImport` 屬性)

明確的 `PInvoke` 受到 .NET Framework 的支援，而且在大部分的 .NET 語言中都有提供。但正如其名，C++ Interop 是視覺效果 C++ 特有的。

C++ Interop

C++ Interop 提供較佳的型別安全，而執行起來通常較不繁瑣。不過，C++ 如果無法使用非受控原始程式碼或跨平臺專案，Interop 就不是選項。

C++ COM Interop

視覺效果 C++ 所支援的互通性功能，在與 COM 元件互通時，會提供與其他 .net 語言的特殊優勢。Interop 不受限於 .NET Framework `tlbimp.exe`（[類型程式庫匯入工具](#)）的限制，例如有限的資料類型支援，以及每個 COM 介面之每個成員的強制性公開，C++ interop 可讓 COM 元件進行存取，而不需要個別的 Interop 元件。不同于 Visual Basic C# 和，C++ Visual 可以直接使用使用一般 Com 機制（例如 `CoCreateInstance` 和 `QueryInterface`）的 com 物件。這是可能的，C++ 因為 Interop 功能會導致編譯器自動插入轉換程式碼，以便從 managed 移至非受控函式，然後再次返回。

使用 C++ Interop 時，可以使用 COM 元件，因為它們通常會使用，也可以包裝在 C++ 類別內。這些包裝函式類別稱為「自訂執行時間可呼叫包裝函式」（CRCWs），而且在應用程式代碼中與直接使用 COM 相比有兩個優點：

- 產生的類別可以從視覺效果 C++ 以外的語言使用。
- 您可以從 managed 用戶端程式代碼隱藏 COM 介面的詳細資料。.NET 資料類型可以用來取代原生類型，而資料封送處理的細節則可以在 CRCW 內以透明的方式執行。

無論是直接使用 COM 或透過 CRCW，都必須封送處理簡單、可直接執行的引數類型。

可直接輸入的類型

針對使用簡單、內建類型的非受控 API（請參閱[可轉型和非類型的類型](#)），不需要特殊程式碼，因為這些資料類型在記憶體中有相同的標記法，但更複雜的資料類型需要明確的資料封送處理。如需範例，請參閱[如何：使用 PInvoke 從 Managed 程式碼呼叫原生 dll](#)。

範例

```
// vcmcppv2_impl_dllimp.cpp
// compile with: /clr:pure user32.lib
using namespace System::Runtime::InteropServices;

// Implicit DLLImport specifying calling convention
extern "C" int __stdcall MessageBeep(int);

// explicit DLLImport needed here to use P/Invoke marshalling because
// System::String ^ is not the type of the first parameter to printf
[DllImport("msvcrt.dll", EntryPoint = "printf", CallingConvention = CallingConvention::Cdecl, CharSet = CharSet::Ansi)]
// or just
// [DllImport("msvcrt.dll")]
int printf(System::String ^, ...);

int main() {
    // (string literals are System::String by default)
    printf("Begin beep\n");
    MessageBeep(100000);
    printf("Done\n");
}
```

```
Begin beep
Done
```

本章節內容

- [如何: 使用 C++ Interop 封送處理 ANSI 字串](#)
- [如何: 使用 C++ Interop 封送處理 Unicode 字串](#)
- [如何: 使用 C++ Interop 封送處理 COM 字串](#)
- [如何: 使用 C++ Interop 封送處理結構](#)
- [如何: 使用 C++ Interop 封送處理陣列](#)
- [如何: 使用 C++ Interop 封送處理回呼和委派](#)
- [如何: 使用 C++ Interop 封送處理內嵌指標](#)
- [如何: 存取 System::String 中的字元](#)
- [如何: 將 char * 字串轉換為 System::Byte 陣列](#)
- [如何: 將 System::String 轉換成 wchar_t * 或 char*](#)
- [如何: 將 System::String 轉換為標準字串](#)
- [如何: 將標準字串轉換為 System::String](#)
- [如何: 取得位元組陣列的指標](#)
- [如何: 將 Unmanaged 資源載入至位元組陣列](#)
- [如何: 修改原生函式中的參考類別](#)
- [如何: 判斷影像是否為原生或 CLR](#)
- [如何: 將原生 DLL 新增至全域組件快取](#)
- [如何: 以原生類型存放實值型別的參考](#)

- [如何 : 在 Unmanaged 記憶體中存放物件參考](#)
- [如何 : 偵測/clr 編譯](#)
- [如何 : 在 System::Guid 和 _GUID 之間轉換](#)
- [如何 : 指定 out 參數](#)
- [如何 : 在/clr 編譯中使用原生類型](#)
- [如何 : 以原生類型宣告控制代碼](#)
- [如何 : 包裝原生類別以便讓 C# 使用](#)

如需在 interop 案例中使用委派的詳細資訊，請參閱[委派\(C++元件擴充功能\)](#)。

請參閱

- [從 Managed 程式碼呼叫原生函式](#)

如何：使用 C++ Interop 封送處理 ANSI 字串

2020/11/2 • [Edit Online](#)

本主題將示範如何使用 C++ Interop 來傳遞 ANSI 字串，但 .NET Framework [String](#) 代表 Unicode 格式的字串，因此轉換成 ANSI 是一個額外的步驟。如需與其他字串類型交互操作，請參閱下列主題：

- [如何：使用 C++ Interop 封送處理 Unicode 字串](#)
- [如何：使用 C++ Interop 封送處理 COM 字串](#)

下列程式碼範例使用 [managed](#)、[非受控](#) #pragma 指示詞，在相同的檔案中執行 managed 和非受控函式，但這些函式在個別檔案中定義時，會以相同的方式相交互操作。因為只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯，所以它們可以保留其效能特性。

範例：傳遞 ANSI 字串

此範例示範如何使用，將來自 managed 的 ANSI 字串傳遞至非受控函數 [StringToHGlobalAnsi](#)。這個方法會在非受控堆積上配置記憶體，並在執行轉換之後傳回位址。這表示沒有必要的釘選（因為 GC 堆積上的記憶體未傳遞至非受控函式），而且從傳回的 IntPtr [StringToHGlobalAnsi](#) 必須明確釋放或記憶體流失結果。

```
// MarshalANSI1.cpp
// compile with: /clr
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(const char* p) {
    printf_s("(native) received '%s'\n", p);
}

#pragma managed

int main() {
    String^ s = gcnew String("sample string");
    IntPtr ip = Marshal::StringToHGlobalAnsi(s);
    const char* str = static_cast<const char*>(ip.ToPointer());

    Console::WriteLine("(managed) passing string...");
    NativeTakesAString( str );

    Marshal::FreeHGlobal( ip );
}
```

範例：存取 ANSI 字串所需的資料封送處理

下列範例示範在非受控函式所呼叫的 managed 函式中存取 ANSI 字串所需的資料封送處理。Managed 函式在接收原生字串時，可以直接使用它，或使用方法將它轉換成 managed 字串 [PtrToStringAnsi](#)，如下所示。

```
// MarshalANSI2.cpp
// compile with: /clr
#include <iostream>
#include <vcclr.h>

using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedStringFunc(char* s) {
    String^ ms = Marshal::PtrToStringAnsi(static_cast<IntPtr>(s));
    Console::WriteLine("(managed): received '{0}'", ms);
}

#pragma unmanaged

void NativeProvidesAString() {
    cout << "(native) calling managed func...\n";
    ManagedStringFunc("test string");
}

#pragma managed

int main() {
    NativeProvidesAString();
}
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理 Unicode 字串

2020/11/2 • [Edit Online](#)

本主題將示範 Visual C++ 互通性的一個 facet。如需詳細資訊，請參閱 [使用 c + + Interop \(隱含的 PInvoke\)](#)。

下列程式碼範例使用 **managed**、**非** 受控 #pragma 指示詞，在相同的檔案中執行 managed 和非受控函式，但這些函式在個別檔案中定義時，會以相同的方式相交互操作。只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯。

本主題將示範如何將 Unicode 字串從 managed 傳遞至非受控函式，反之亦然。如需與其他字串類型交互操作，請參閱下列主題：

- [如何：使用 c + + Interop 封送處理 ANSI 字串](#)
- [如何：使用 c + + Interop 封送處理 COM 字串](#)

範例：將 Unicode 字串從 managed 傳遞至非受控函式

若要將 Unicode 字串從 managed 傳遞至非受控函式，在 Vcclr 中宣告的 PtrToStringChars 函式（可以用來存取儲存 managed 字串的記憶體。因為此位址會傳遞至原生函式，所以將記憶體釘選 [pin_ptr \(c + +/cli\)](#) 以避免在執行非受控函式時，發生垃圾收集迴圈的情況下，將字串資料重新放置，是很重要的。

```
// MarshalUnicode1.cpp
// compile with: /clr
#include <iostream>
#include <stdio.h>
#include <vcclr.h>

using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(const wchar_t* p) {
    printf_s("(native) received '%S'\n", p);
}

#pragma managed

int main() {
    String^ s = gcnew String("test string");
    pin_ptr<const wchar_t> str = PtrToStringChars(s);

    Console::WriteLine("(managed) passing string to native func...");
    NativeTakesAString( str );
}
```

範例：存取 Unicode 字串所需的資料封送處理

下列範例示範在非受控函式所呼叫的 managed 函式中存取 Unicode 字串所需的資料封送處理。Managed 函式在接收原生 Unicode 字串時，會使用方法將它轉換成 managed 字串 [PtrToStringUni](#)。

```
// MarshalUnicode2.cpp
// compile with: /clr
#include <iostream>

using namespace std;
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedStringFunc(wchar_t* s) {
    String^ ms = Marshal::PtrToStringUni((IntPtr)s);
    Console::WriteLine("(managed) received '{0}'", ms);
}

#pragma unmanaged

void NativeProvidesAString() {
    cout << "(unmanaged) calling managed func...\n";
    ManagedStringFunc(L"test string");
}

#pragma managed

int main() {
    NativeProvidesAString();
}
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理 COM 字串

2020/11/2 • [Edit Online](#)

本主題示範 BSTR 如何 (COM 程式設計中偏好的基底字元串格式) 可以從 managed 傳遞至非受控函式，反之亦然。如需與其他字串類型交互操作，請參閱下列主題：

- [如何：使用 C++ Interop 封送處理 Unicode 字串](#)
- [如何：使用 C++ Interop 封送處理 ANSI 字串](#)

下列程式碼範例使用 [managed](#)、[非受控](#) #pragma 指示詞，在相同的檔案中執行 managed 和非受控函式，但這些函式在個別檔案中定義時，會以相同的方式相交互操作。只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯。

範例：將 BSTR 從 managed 傳遞至非受控函式

下列範例示範 BSTR 如何 (COM 程式設計中使用的字串格式) 可從 managed 傳遞至非受控函式。呼叫 managed 函式會使用 [StringToBSTR](#) 來取得 .Net system.string 內容之 BSTR 表示的位址。這個指標是使用 [pin_ptr \(C++/CLI\)](#) 來確保其實體位址在非受控函式執行期間，不會在垃圾收集週期中變更。在 [pin_ptr \(C++/CLI\)](#) 超出範圍之前，禁止垃圾收集行程移動記憶體。

```
// MarshalBSTR1.cpp
// compile with: /clr
#define WINVER 0x0502
#define _AFXDLL
#include <afxwin.h>

#include <iostream>
using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

void NativeTakesAString(BSTR bstr) {
    printf_s("%S", bstr);
}

#pragma managed

int main() {
    String^ s = "test string";

    IntPtr ip = Marshal::StringToBSTR(s);
    BSTR bs = static_cast<BSTR>(ip.ToPointer());
    pin_ptr<BSTR> b = &bs;

    NativeTakesAString( bs );
    Marshal::FreeBSTR(ip);
}
```

範例：將 BSTR 從非受控函式傳遞給 managed 函式

下列範例會示範如何將 BSTR 從非受控函式傳遞至 managed 函式。接收 managed 函式可以使用中的字串做為 BSTR，或是用 [PtrToStringBSTR](#) 來將它轉換成，[String](#) 以便與其他 managed 函數搭配使用。因為代表 BSTR 的記憶

體是在非受控堆積上配置的，所以不需要釘選，因為非受控堆積上沒有垃圾收集。

```
// MarshalBSTR2.cpp
// compile with: /clr
#define WINVER 0x0502
#define _AFXDLL
#include <afxwin.h>

#include <iostream>
using namespace std;

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed

void ManagedTakesAString(BSTR bstr) {
    String^ s = Marshal::PtrToStringBSTR(static_cast<IntPtr>(bstr));
    Console::WriteLine("(managed) converted BSTR to String: '{0}'", s);
}

#pragma unmanaged

void UnManagedFunc() {
    BSTR bs = SysAllocString(L"test string");
    printf_s("(unmanaged) passing BSTR to managed func...\n");
    ManagedTakesAString(bs);
}

#pragma managed

int main() {
    UnManagedFunc();
}
```

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理結構

2020/11/2 • [Edit Online](#)

本主題將示範 Visual C++ 互通性的一個 facet。如需詳細資訊，請參閱 [使用 c++ Interop \(隱含的 PInvoke\)](#)。

下列程式碼範例使用 **managed**、**非** 受控 #pragma 指示詞，在相同的檔案中執行 managed 和非受控函式，但這些函式在個別檔案中定義時，會以相同的方式相交互操作。只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯。

範例：將結構從 managed 傳遞至非受控函式

下列範例示範如何透過傳值和傳址方式，將結構從 managed 傳遞至非受控函式。因為此範例中的結構只包含簡單的內建資料類型（看到）的全像是，所以不需要特殊的封送處理。若要封送處理非多型的結構（例如包含指標的結構），請參閱 [如何：使用 c++ Interop 封送處理內嵌指標](#)。

```

// PassStruct1.cpp
// compile with: /clr

#include <stdio.h>
#include <math.h>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

struct Location {
    int x;
    int y;
};

double GetDistance(Location loc1, Location loc2) {
    printf_s("[unmanaged] loc1(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2(%d,%d)\n", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y = loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    printf_s("[unmanaged] Initializing location...\n");
    lp->x = 50;
    lp->y = 50;
}

#pragma managed

int main() {
    Location loc1;
    loc1.x = 0;
    loc1.y = 0;

    Location loc2;
    loc2.x = 100;
    loc2.y = 100;

    double dist = GetDistance(loc1, loc2);
    Console::WriteLine("[managed] distance = {0}", dist);

    Location loc3;
    InitLocation(&loc3);
    Console::WriteLine("[managed] x={0} y={1}", loc3.x, loc3.y);
}

```

範例：從非受控函式傳遞結構至 managed 函式

下列範例示範如何透過傳值和傳址方式，將結構從非受控函式傳遞至 managed 函式。因為此範例中的結構只包含簡單的內建資料類型（[看到](#) 的全像是），所以不需要特殊的封送處理。若要封送處理非多型的結構（例如包含指標的結構），請參閱 [如何：使用 C++ Interop 封送處理內嵌指標](#)。

```

// PassStruct2.cpp
// compile with: /clr
#include <stdio.h>
#include <math.h>
using namespace System;

// native structure definition
struct Location {
    int x;
    int y;
};

#pragma managed

double GetDistance(Location loc1, Location loc2) {
    Console::Write("[managed] got loc1({0},{1})", loc1.x, loc1.y);
    Console::WriteLine(" loc2({0},{1})", loc2.x, loc2.y);

    double h = loc1.x - loc2.x;
    double v = loc1.y - loc2.y;
    double dist = sqrt( pow(h,2) + pow(v,2) );

    return dist;
}

void InitLocation(Location* lp) {
    Console::WriteLine("[managed] Initializing location...");
    lp->x = 50;
    lp->y = 50;
}

#pragma unmanaged

int UnmanagedFunc() {
    Location loc1;
    loc1.x = 0;
    loc1.y = 0;

    Location loc2;
    loc2.x = 100;
    loc2.y = 100;

    printf_s("(unmanaged) loc1=(%d,%d)", loc1.x, loc1.y);
    printf_s(" loc2=(%d,%d)\n", loc2.x, loc2.y);

    double dist = GetDistance(loc1, loc2);
    printf_s("[unmanaged] distance = %f\n", dist);

    Location loc3;
    InitLocation(&loc3);
    printf_s("[unmanaged] got x=%d y=%d\n", loc3.x, loc3.y);

    return 0;
}

#pragma managed

int main() {
    UnmanagedFunc();
}

```

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理陣列

2020/11/2 • [Edit Online](#)

本主題將示範 Visual C++ 互通性的一個 facet。如需詳細資訊，請參閱 [使用 c++ Interop \(隱含的 PInvoke\)](#)。

下列程式碼範例使用 **managed**、**非** 受控 #pragma 指示詞，在相同的檔案中執行 managed 和非受控函式，但這些函式在個別檔案中定義時，會以相同的方式相交互操作。只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯。

範例：將受控陣列傳遞至非受控函式

下列範例示範如何將 managed 陣列傳遞至非受控函數。Managed 函式在呼叫非受控函式之前，會使用 `pin_ptr(c++/cli)` 來隱藏陣列的垃圾收集。藉由提供未受管理的函式與 GC 堆積的固定指標，可避免複製陣列的額外負荷。為了示範非受控函式正在存取 GC 堆積記憶體，它會修改陣列的內容，並在 managed 函式繼續控制時反映變更。

```

// PassArray1.cpp
// compile with: /clr
#ifndef _CRT_RAND_S
#define _CRT_RAND_S
#endif

#include <iostream>
#include <stdlib.h>
using namespace std;

using namespace System;

#pragma unmanaged

void TakesAnArray(int* a, int c) {
    cout << "(unmanaged) array received:\n";
    for (int i=0; i<c; i++)
        cout << "a[" << i << "] = " << a[i] << "\n";

    unsigned int number;
    errno_t err;

    cout << "(unmanaged) modifying array contents...\n";
    for (int i=0; i<c; i++) {
        err = rand_s( &number );
        if ( err == 0 )
            a[i] = number % 100;
    }
}

#pragma managed

int main() {
    array<int>^ nums = gcnew array<int>(5);

    nums[0] = 0;
    nums[1] = 1;
    nums[2] = 2;
    nums[3] = 3;
    nums[4] = 4;

    Console::WriteLine("(managed) array created:");
    for (int i=0; i<5; i++)
        Console::WriteLine("a[{0}] = {1}", i, nums[i]);

    pin_ptr<int> pp = &nums[0];
    TakesAnArray(pp, 5);

    Console::WriteLine("(managed) contents:");
    for (int i=0; i<5; i++)
        Console::WriteLine("a[{0}] = {1}", i, nums[i]);
}

```

範例：傳遞非受控陣列至 managed 函數

下列範例示範如何將非受控陣列傳遞至 managed 函數。Managed 函式會直接存取陣列記憶體 (而不是建立 managed 陣列並複製陣列內容)，這樣可讓 managed 函式所做的變更在未受管理的函式重新取得控制權時反映在該函式中。

```

// PassArray2.cpp
// compile with: /clr
#include <iostream>
using namespace std;

using namespace System;

#pragma managed

void ManagedTakesAnArray(int* a, int c) {
    Console::WriteLine("(managed) array received:");
    for (int i=0; i<c; i++)
        Console::WriteLine("a[{0}] = {1}", i, a[i]);

    cout << "(managed) modifying array contents...\n";
    Random^ r = gcnew Random(DateTime::Now.Second);
    for (int i=0; i<c; i++)
        a[i] = r->Next(100);
}

#pragma unmanaged

void NativeFunc() {
    int nums[5] = { 0, 1, 2, 3, 4 };

    printf_s("(unmanaged) array created:\n");
    for (int i=0; i<5; i++)
        printf_s("a[%d] = %d\n", i, nums[i]);

    ManagedTakesAnArray(nums, 5);

    printf_s("(unmanaged) contents:\n");
    for (int i=0; i<5; i++)
        printf_s("a[%d] = %d\n", i, nums[i]);
}

#pragma managed

int main() {
    NativeFunc();
}

```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理回呼和委派

2020/11/2 • [Edit Online](#)

本主題將示範如何使用 Visual C++, 將回呼和委派的封送處理 (受控和非受控碼之間的 managed 版本)。

下列程式碼範例使用 **managed**、**非** 受控 `#pragma` 指示詞，在相同的檔案中執行 managed 和非受控函式，但是也可以在個別的檔案中定義函數。只包含非受控函式的檔案不需要使用 [/clr \(Common Language Runtime 編譯\)](#) 來進行編譯。

範例：設定非受控 API 來觸發受管理的委派

下列範例示範如何設定非受控 API 來觸發受管理的委派。建立 managed 委派，並使用其中一個 interop 方法 [GetFunctionPointerForDelegate](#) 來取得委派的基礎進入點。此位址接著會傳遞至非受控函式，而不需要知道它是實作為 managed 函式的事實。

請注意，有可能（但不是必要）使用 [pin_ptr \(c++/cli\)](#) 來釘選委派，以防止垃圾收集行程重新找出或處置委派。需要預防過多的垃圾收集，但釘選可提供比所需更多的保護，因為它會防止收集但也會防止重新配置。

如果委派是由垃圾收集重新置放，則不會影響 underlaying managed 回呼，因此 [Alloc](#) 會用來加入委派的參考，以允許重新配置委派，但防止處置。使用 GCHandle 而非 pin_ptr 可減少 managed 堆積的分散潛力。

```

// MarshalDelegate1.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

// Declare an unmanaged function type that takes two int arguments
// Note the use of __stdcall for compatibility with managed code
typedef int (__stdcall *ANSWERCB)(int, int);

int TakesCallback(ANSWERCB fp, int n, int m) {
    printf_s("[unmanaged] got callback address, calling it...\n");
    return fp(n, m);
}

#pragma managed

public delegate int GetTheAnswerDelegate(int, int);

int GetNumber(int n, int m) {
    Console::WriteLine("[managed] callback!");
    return n + m;
}

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);
    GCHandle gch = GCHandle::Alloc(fp);
    IntPtr ip = Marshal::GetFunctionPointerForDelegate(fp);
    ANSWERCB cb = static_cast<ANSWERCB>(ip.ToPointer());
    Console::WriteLine("[managed] sending delegate as callback...");

    // force garbage collection cycle to prove
    // that the delegate doesn't get disposed
    GC::Collect();

    int answer = TakesCallback(cb, 243, 257);

    // release reference to delegate
    gch.Free();
}

```

範例：非受控 API 儲存的函式指標

下列範例與上一個範例類似，但在此情況下，未受管理的 API 會儲存提供的函式指標，因此可以隨時叫用，要求將垃圾收集視為任意時間長度。因此，下列範例會使用的全域實例 [GCHandle](#)，以防止委派重新放置，與函數範圍無關。如第一個範例中所述，這些範例不需要使用 pin_ptr，但在此情況下，不會有任何作用，因為 pin_ptr 的範圍僅限於單一函式。

```

// MarshalDelegate2.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

#pragma unmanaged

// Declare an unmanaged function type that takes two int arguments
// Note the use of __stdcall for compatibility with managed code
typedef int (__stdcall *ANSWERCB)(int, int);
static ANSWERCB cb;

int TakesCallback(ANSWERCB fp, int n, int m) {
    cb = fp;
    if (cb) {
        printf_s("[unmanaged] got callback address (%d), calling it...\n", cb);
        return cb(n, m);
    }
    printf_s("[unmanaged] unregistering callback");
    return 0;
}

#pragma managed

public delegate int GetTheAnswerDelegate(int, int);

int GetNumber(int n, int m) {
    Console::WriteLine("[managed] callback!");
    static int x = 0;
    ++x;

    return n + m + x;
}

static GCHandle gch;

int main() {
    GetTheAnswerDelegate^ fp = gcnew GetTheAnswerDelegate(GetNumber);

    gch = GCHandle::Alloc(fp);

    IntPtr ip = Marshal::GetFunctionPointerForDelegate(fp);
    ANSWERCB cb = static_cast<ANSWERCB>(ip.ToPointer());
    Console::WriteLine("[managed] sending delegate as callback...");

    int answer = TakesCallback(cb, 243, 257);

    // possibly much later (in another function)...

    Console::WriteLine("[managed] releasing callback mechanisms...");
    TakesCallback(0, 243, 257);
    gch.Free();
}

```

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：使用 C++ Interop 封送處理內嵌指標

2019/12/10 • [Edit Online](#)

下列程式碼範例會使用managed、非受控 #pragma 指示詞，在同一個檔案中執行 managed 和非受控函式，但如果在個別的檔案中定義，則這些函式會以相同的方式進行交互作用。僅包含非受控函式的檔案不需要使用`/clr`（Common Language Runtime 編譯）進行編譯。

範例

下列範例示範如何從 managed 函式呼叫接受包含指標之結構的非受控函式。Managed 函式會建立結構的實例，並使用 new 關鍵字（而不是ref new、gcnew 關鍵字）初始化內嵌指標。因為這會在原生堆積上配置記憶體，所以不需要釘選陣列來抑制垃圾收集。不過，必須明確刪除記憶體，以避免記憶體流失。

```
// marshal_embedded_pointer.cpp
// compile with: /clr
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices;

// unmanaged struct
struct ListStruct {
    int count;
    double* item;
};

#pragma unmanaged

void UnmanagedTakesListStruct(ListStruct list) {
    printf_s("[unmanaged] count = %d\n", list.count);
    for (int i=0; i<list.count; i++)
        printf_s("array[%d] = %f\n", i, list.item[i]);
}

#pragma managed

int main() {
    ListStruct list;
    list.count = 10;
    list.item = new double[list.count];

    Console::WriteLine("[managed] count = {0}", list.count);
    Random^ r = gcnew Random(0);
    for (int i=0; i<list.count; i++) {
        list.item[i] = r->NextDouble() * 100.0;
        Console::WriteLine("array[{0}] = {1}", i, list.item[i]);
    }

    UnmanagedTakesListStruct( list );
    delete list.item;
}
```

```
[managed] count = 10
array[0] = 72.624326996796
array[1] = 81.7325359590969
array[2] = 76.8022689394663
array[3] = 55.8161191436537
array[4] = 20.6033154021033
array[5] = 55.8884794618415
array[6] = 90.6027066011926
array[7] = 44.2177873310716
array[8] = 97.754975314138
array[9] = 27.370445768987
[unmanaged] count = 10
array[0] = 72.624327
array[1] = 81.732536
array[2] = 76.802269
array[3] = 55.816119
array[4] = 20.603315
array[5] = 55.888479
array[6] = 90.602707
array[7] = 44.217787
array[8] = 97.754975
array[9] = 27.370446
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：擴充封送處理程式庫

2020/11/2 • [Edit Online](#)

本主題說明如何擴充封送處理程式庫，以提供資料類型之間的更多轉換。使用者可以擴充封送處理程式庫，以找出程式庫目前不支援的任何資料轉換。

您可以使用兩種方式之一來擴充封送處理程式庫，不論是否有 [Marshal_coNtext 類別](#)。請參閱 [c++ 主題中的封送處理總覽](#)，以判斷新的轉換是否需要內容。

在這兩種情況下，您必須先為新的封送處理轉換建立檔案。若要保留標準封送處理程式庫檔案的完整性，您可以這麼做。如果您想要將專案移植到另一部電腦或其他程式設計人員，您必須將新的封送處理檔案連同專案的其餘部分一起複製。如此一來，將保證收到專案的使用者會收到新的轉換，而不需要修改任何程式庫檔案。

若要使用不需要內容的轉換擴充封送處理程式庫

1. 建立檔案以儲存新的封送處理函數，例如 MyMarshal。
2. 包含一或多個封送處理程式庫檔案：
 - 基底類型的封送處理 .h。
 - 適用于 windows 資料類型的 marshal_windows.h。
 - c++ 標準程式庫資料類型的 marshal_cppstd.h。
 - ATL 資料類型的 marshal_atl.h。
3. 在這些步驟的結尾使用程式碼，以撰寫轉換函式。在此程式碼中，TO 是要轉換成的型別，從是要轉換的型別，而 `from` 是要轉換的參數。
4. 將關於轉換邏輯的批註取代為程式碼，將參數轉換為 `from` 的物件，以輸入並傳回已轉換的物件。

```
namespace msclr {
    namespace interop {
        template<>
        inline TO marshal_as<TO, FROM> (const FROM& from) {
            // Insert conversion logic here, and return a TO parameter.
        }
    }
}
```

使用需要內容的轉換擴充封送處理程式庫

1. 建立檔案以儲存新的封送處理函式，例如 MyMarshal.h
2. 包含一或多個封送處理程式庫檔案：
 - 基底類型的封送處理 .h。
 - 適用于 windows 資料類型的 marshal_windows.h。
 - c++ 標準程式庫資料類型的 marshal_cppstd.h。
 - ATL 資料類型的 marshal_atl.h。
3. 在這些步驟的結尾使用程式碼，以撰寫轉換函式。在此程式碼中，TO 是要轉換成的型別，從是轉換來源的型別，`toObject` 是用來儲存結果的指標，`fromObject` 而是要轉換的參數。

4. 將使用程式碼初始化的批註取代為 `toPtr` 適當的空白值。例如，如果它是指標，請將它設定為 `NULL`。
5. 以程式碼取代轉換邏輯的相關批註，以將參數轉換為 `from` 類型 `TO` 的物件。這個已轉換的物件將會儲存在中 `toPtr`。
6. 將關於設定的批註取代為 `toObject` 要設定 `toObject` 為已轉換物件的程式碼。
7. 將清除原生資源的相關批註取代為程式碼，以釋放任何配置的記憶體 `toPtr`。如果 `toPtr` 使用配置的記憶體 `new`，請使用 `delete` 釋放記憶體。

```

namespace msclr {
    namespace interop {
        template<>
        ref class context_node<TO, FROM> : public context_node_base
        {
        private:
            TO toPtr;
        public:
            context_node(TO& toObject, FROM fromObject)
            {
                // (Step 4) Initialize toPtr to the appropriate empty value.
                // (Step 5) Insert conversion logic here.
                // (Step 6) Set toObject to the converted parameter.
            }
            ~context_node()
            {
                this->!context_node();
            }
        protected:
            !context_node()
            {
                // (Step 7) Clean up native resources.
            }
        };
    }
}

```

範例：擴充封送處理程式庫

下列範例會以不需要內容的轉換擴充封送處理程式庫。在此範例中，程式碼會將員工資訊從原生資料類型轉換為 managed 資料類型。

```

// MyMarshalNoContext.cpp
// compile with: /clr
#include <msclr/marshal.h>

value struct ManagedEmp {
    System::String^ name;
    System::String^ address;
    int zipCode;
};

struct NativeEmp {
    char* name;
    char* address;
    int zipCode;
};

namespace msclr {
    namespace interop {
        template<>
        inline ManagedEmp^ marshal_as<ManagedEmp^, NativeEmp> (const NativeEmp& from) {
            ManagedEmp^ toValue = gcnew ManagedEmp;
            toValue->name = marshal_as<System::String^>(from.name);
            toValue->address = marshal_as<System::String^>(from.address);
            toValue->zipCode = from.zipCode;
            return toValue;
        }
    }
}

using namespace System;
using namespace msclr::interop;

int main() {
    NativeEmp employee;

    employee.name = "Jeff Smith";
    employee.address = "123 Main Street";
    employee.zipCode = 98111;

    ManagedEmp^ result = marshal_as<ManagedEmp^>(employee);

    Console::WriteLine("Managed name: {0}", result->name);
    Console::WriteLine("Managed address: {0}", result->address);
    Console::WriteLine("Managed zip code: {0}", result->zipCode);

    return 0;
}

```

在上述範例中，函數會傳回已 `marshal_as` 轉換資料的控制碼。這樣做的目的是為了避免建立額外的資料複本。直接傳回變數會有與其相關聯的不必要效能成本。

```

Managed name: Jeff Smith
Managed address: 123 Main Street
Managed zip code: 98111

```

範例：轉換員工資訊

下列範例會將員工資訊從 managed 資料類型轉換成原生資料類型。這種轉換需要封送處理內容。

```

// MyMarshalContext.cpp
// compile with: /clr
#include <stdlib.h>
#include <string.h>

```

```

#include <msclr/marshal.h>

value struct ManagedEmp {
    System::String^ name;
    System::String^ address;
    int zipCode;
};

struct NativeEmp {
    const char* name;
    const char* address;
    int zipCode;
};

namespace msclr {
    namespace interop {
        template<>
        ref class context_node<NativeEmp*, ManagedEmp^> : public context_node_base
        {
        private:
            NativeEmp* toPtr;
            marshal_context context;
        public:
            context_node(NativeEmp*& toObject, ManagedEmp^ fromObject)
            {
                // Conversion logic starts here
                toPtr = NULL;

                const char* nativeName;
                const char* nativeAddress;

                // Convert the name from String^ to const char*.
                System::String^ tempValue = fromObject->name;
                nativeName = context.marshal_as<const char*>(tempValue);

                // Convert the address from String^ to const char*.
                tempValue = fromObject->address;
                nativeAddress = context.marshal_as<const char*>(tempValue);

                toPtr = new NativeEmp();
                toPtr->name = nativeName;
                toPtr->address = nativeAddress;
                toPtr->zipCode = fromObject->zipCode;

                toObject = toPtr;
            }
            ~context_node()
            {
                this->!context_node();
            }
        protected:
            !context_node()
            {
                // When the context is deleted, it will free the memory
                // allocated for toPtr->name and toPtr->address, so toPtr
                // is the only memory that needs to be freed.
                if (toPtr != NULL) {
                    delete toPtr;
                    toPtr = NULL;
                }
            }
        };
    };
}

using namespace System;
using namespace msclr::interop;

int main() {

```

```
ManagedEmp^ employee = gcnew ManagedEmp();

employee->name = gcnew String("Jeff Smith");
employee->address = gcnew String("123 Main Street");
employee->zipCode = 98111;

marshal_context context;
NativeEmp* result = context.marshal_as<NativeEmp*>(employee);

if (result != NULL) {
    printf_s("Native name: %s\nNative address: %s\nNative zip code: %d\n",
            result->name, result->address, result->zipCode);
}

return 0;
}
```

```
Native name: Jeff Smith
Native address: 123 Main Street
Native zip code: 98111
```

另請參閱

[C++ 中的封送處理總覽](#)

如何：存取 System::String 中的字元

2020/11/2 • [Edit Online](#)

您可以存取物件的字元，以對 [String](#) 採用字串的非受控函式進行高效能呼叫 `wchar_t*`。方法會產生物件之第一個字元的內部指標 [String](#)。此指標可以直接操作或釘選，並傳遞至預期一般字串的函式 `wchar_t`。

範例

`PtrToStringChars` 傳回 [Char](#)，也就是內部指標(也稱為 `byref`)。因此，它會受限於垃圾收集。除非您要將它傳遞給原生函式，否則您不需要釘選此指標。

請考慮下列程式碼：因為是內部指標，所以不需要釘選，`ppchar` 而且如果垃圾收集行程移動它所指向的字串，它也會更新 `ppchar`。若沒有 `pin_ptr(c + +/cli)`，程式碼將可運作，而且不會因為釘選而造成潛在的效能衝擊。

如果您傳遞 `ppchar` 至原生函式，則它必須是釘選指標；垃圾收集行程將無法更新非受控堆疊框架上的任何指標。

```
// PtrToStringChars.cpp
// compile with: /clr
#include<vcclr.h>
using namespace System;

int main() {
    String ^ mystring = "abcdefg";

    interior_ptr<const Char> ppchar = PtrToStringChars( mystring );

    for ( ; *ppchar != L'\0'; ++ppchar )
        Console::Write(*ppchar);
}
```

```
abcdefg
```

此範例顯示需要釘選的位置。

```
// PtrToStringChars_2.cpp
// compile with: /clr
#include <string.h>
#include <vcclr.h>
// using namespace System;

size_t getlen(System::String ^ s) {
    // Since this is an outside string, we want to be secure.
    // To be secure, we need a maximum size.
    size_t maxsize = 256;
    // make sure it doesn't move during the unmanaged call
    pin_ptr<const wchar_t> pinchars = PtrToStringChars(s);
    return wcsnlen(pinchars, maxsize);
};

int main() {
    System::Console::WriteLine(getlen("testing"));
}
```

內部指標具有原生 C++ 指標的所有屬性。例如，您可以使用它來引導連結的資料結構，並只使用一個指標進行插入和刪除：

```
// PtrToStringChars_3.cpp
// compile with: /clr /LD
using namespace System;
ref struct ListNode {
    Int32 elem;
    ListNode ^ Next;
};

void deleteNode( ListNode ^ list, Int32 e ) {
    interior_ptr<ListNode ^> ptrToNext = &list;
    while (*ptrToNext != nullptr) {
        if ( (*ptrToNext) -> elem == e )
            *ptrToNext = (*ptrToNext) -> Next;    // delete node
        else
            ptrToNext = &(*ptrToNext) -> Next;    // move to next node
    }
}
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：將 char * 字串轉換為 System::Byte 陣列

2019/12/10 • [Edit Online](#)

將 `char *` 字串轉換成 `Byte` 陣列最有效率的方式，就是使用 `Marshal` 類別。

範例

```
// convert_native_string_to_Byte_array.cpp
// compile with: /clr
#include <string.h>

using namespace System;
using namespace System::Runtime::InteropServices;

int main() {
    char buf[] = "Native String";
    int len = strlen(buf);

    array< Byte >^ byteArray = gcnew array< Byte >(len + 2);

    // convert native pointer to System::IntPtr with C-Style cast
    Marshal::Copy((IntPtr)buf,byteArray, 0, len);

    for ( int i = byteArray->GetLowerBound(0); i <= byteArray->GetUpperBound(0); i++ ) {
        char dc = *(Byte^) byteArray->GetValue(i);
        Console::Write((Char)dc);
    }

    Console::WriteLine();
}
```

Native String

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：將 System::String 轉換為 wchar_t* 或 char*

2019/12/10 • [Edit Online](#)

您可以使用 Vcclr.h 中的 `PtrToStringChars`，將 `String` 轉換為原生 (Native) `wchar_t *` 或 `char *`。這一定會傳回寬 Unicode 字串指標，因為內部的 CLR 字串即為 Unicode。接著，您可以轉換寬指標，如下列範例所示。

範例

```
// convert_string_to_wchar.cpp
// compile with: /clr
#include < stdio.h >
#include < stdlib.h >
#include < vcclr.h >

using namespace System;

int main() {
    String ^str = "Hello";

    // Pin memory so GC can't move it while native function is called
    pin_ptr<const wchar_t> wch = PtrToStringChars(str);
    printf_s("%S\n", wch);

    // Conversion to char* :
    // Can just convert wchar_t* to char* using one of the
    // conversion functions such as:
    // WideCharToMultiByte()
    // wcstombs_s()
    // ...
    size_t convertedChars = 0;
    size_t sizeInBytes = ((str->Length + 1) * 2);
    errno_t err = 0;
    char *ch = (char *)malloc(sizeInBytes);

    err = wcstombs_s(&convertedChars,
                     ch, sizeInBytes,
                     wch, sizeInBytes);
    if (err != 0)
        printf_s("wcstombs_s failed!\n");

    printf_s("%s\n", ch);
}
```

```
Hello
Hello
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：將 System::String 轉換為標準字串

2019/12/10 • [Edit Online](#)

您可以將 `String` 轉換成 `std::string` 或 `std::wstring`，而不需在 Vcclr 中使用 `PtrToStringChars`。

範例

```
// convert_system_string.cpp
// compile with: /clr
#include <string>
#include <iostream>
using namespace std;
using namespace System;

void MarshalString ( String ^ s, string& os ) {
    using namespace Runtime::InteropServices;
    const char* chars =
        (const char*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();
    os = chars;
    Marshal::FreeHGlobal(IntPtr((void*)chars));
}

void MarshalString ( String ^ s, wstring& os ) {
    using namespace Runtime::InteropServices;
    const wchar_t* chars =
        (const wchar_t*)(Marshal::StringToHGlobalUni(s)).ToPointer();
    os = chars;
    Marshal::FreeHGlobal(IntPtr((void*)chars));
}

int main() {
    string a = "test";
    wstring b = L"test2";
    String ^ c = gcnew String("abcd");

    cout << a << endl;
    MarshalString(c, a);
    c = "efgh";
    MarshalString(c, b);
    cout << a << endl;
    wcout << b << endl;
}
```

```
test
abcd
efgh
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：將標準字串轉換為 System::String

2019/12/10 • [Edit Online](#)

本主題說明如何將C++標準程式庫字串(<字串>)轉換成 String。

範例

```
// convert_standard_string_to_system_string.cpp
// compile with: /clr
#include <string>
#include <iostream>
using namespace System;
using namespace std;

int main() {
    string str = "test";
    cout << str << endl;
    String^ str2 = gcnew String(str.c_str());
    Console::WriteLine(str2);

    // alternatively
    String^ str3 = gcnew String(str.c_str());
    Console::WriteLine(str3);
}
```

```
test
test
test
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：取得位元組陣列的指標

2019/12/10 • [Edit Online](#)

您可以取得第一個專案的位址，並將它指派給指標，以取得 `Byte` 陣列中陣列區塊的指標。

範例

```
// pointer_to_Byte_array.cpp
// compile with: /clr
using namespace System;
int main() {
    Byte bArr[] = {1, 2, 3};
    Byte* pbArr = &bArr[0];

    array<Byte> ^ bArr2 = gcnew array<Byte>{1,2,3};
    interior_ptr<Byte> pbArr2 = &bArr2[0];
}
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：將 Unmanaged 資源載入至位元組陣列

2020/11/2 • [Edit Online](#)

本主題討論幾種將非受控資源載入陣列的方式 Byte。

範例

如果您知道非受控資源的大小，您可以預先配置 CLR 陣列，然後使用 CLR 陣列的陣列區塊指標將資源載入陣列中。

```
// load_unmanaged_resources_into_Byte_array.cpp
// compile with: /clr
using namespace System;
void unmanaged_func( unsigned char * p ) {
    for ( int i = 0; i < 10; i++ )
        p[ i ] = i;
}

public ref class A {
public:
    void func() {
        array<Byte> ^b = gcnew array<Byte>(10);
        pin_ptr<Byte> p = &b[ 0 ];
        Byte * np = p;
        unmanaged_func( np ); // pass pointer to the block of CLR array.
        for ( int i = 0; i < 10; i++ )
            Console::Write( b[ i ] );
        Console::WriteLine();
    }
};

int main() {
    A^ g = gcnew A;
    g->func();
}
```

0123456789

此範例示範如何將資料從非受控記憶體區塊複製到 managed 陣列。

```
// load_unmanaged_resources_into_Byte_array_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#include <string.h>
int main() {
    char buf[] = "Native String";
    int len = strlen(buf);
    array<Byte> ^byteArray = gcnew array<Byte>(len + 2);

    // convert any native pointer to IntPtr by doing C-Style cast
    Marshal::Copy( (IntPtr)buf, byteArray, 0, len );
}
```

另請參閱

使用 c + + Interop (隱含 PInvoke)

如何：修改原生函式中的參考類別

2020/11/2 • [Edit Online](#)

您可以將含有 CLR 陣列的參考類別傳遞給原生函式，並使用 PInvoke 服務修改類別。

範例

編譯下列原生程式庫。

```
// modify_ref_class_in_native_function.cpp
// compile with: /LD
#include <stdio.h>
#include <windows.h>

struct S {
    wchar_t* str;
    int intarr[2];
};

extern "C" {
    __declspec(dllexport) int bar(S* param) {
        printf_s("str: %S\n", param->str);
        fflush(stdin);
        fflush(stdout);
        printf_s("In native: intarr: %d, %d\n",
            param->intarr[0], param->intarr[1]);
        fflush(stdin);
        fflush(stdout);
        param->intarr[0]=300;param->intarr[1]=400;
        return 0;
    }
}
```

編譯下列元件。

```
// modify_ref_class_in_native_function_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

[StructLayout(LayoutKind::Sequential, CharSet = CharSet::Unicode)]
ref class S {
public:
    [MarshalAs(UnmanagedType::LPWStr)]
    String ^ str;
    [MarshalAs(UnmanagedType::ByValArray,
        ArraySubType=UnmanagedType::I4, SizeConst=2)]
    array<Int32> ^ intarr;
};

[DllImport("modify_ref_class_in_native_function.dll",
    CharSet=CharSet::Unicode)]
int bar([In][Out] S ^ param);

int main() {
    S ^ param = gcnew S;
    param->str = "Hello";
    param->intarr = gcnew array<Int32>(2);
    param->intarr[0] = 100;
    param->intarr[1] = 200;
    bar(param); // Call to native function
    Console::WriteLine("In managed: intarr: {0}, {1}",
        param->intarr[0], param->intarr[1]);
}
```

```
str: Hello
In native: intarr: 100, 200
In managed: intarr: 300, 400
```

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：判斷影像是否為原生或 CLR

2019/12/10 • [Edit Online](#)

判斷是否已為 common language runtime 建立影像的一種方式是使用dumpbin/[CLRHEADER](#)。

您也可以透過程式設計方式檢查是否已為 common language runtime 建立影像。如需詳細資訊，請參閱[如何：偵測/Clr 編譯](#)。

範例

下列範例會判斷是否已建立影像，以在 common language runtime 上執行。

```

// detect_image_type.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;

enum class CompilationMode {Invalid, Native, CLR };

static CompilationMode IsManaged(String^ filename) {
    try {
        array<Byte>^ data = gcnew array<Byte>(4096);
        FileInfo^ file = gcnew FileInfo(filename);
        Stream^ fin = file->Open(FileMode::Open, FileAccess::Read);
        Int32 iRead = fin->Read(data, 0, 4096);
        fin->Close();

        // Verify this is a executable/dll
        if ((data[1] << 8 | data[0]) != 0x5a4d)
            return CompilationMode::Invalid;

        // This will get the address for the WinNT header
        Int32 iWinNTHdr = data[63]<<24 | data[62]<<16 | data[61] << 8 | data[60];

        // Verify this is an NT address
        if ((data[iWinNTHdr+3] << 24 | data[iWinNTHdr+2] << 16 | data[iWinNTHdr+1] << 8 | data[iWinNTHdr]) != 0x00004550)
            return CompilationMode::Invalid;

        Int32 iLightningAddr = iWinNTHdr + 24 + 208;
        Int32 iSum = 0;
        Int32 iTop = iLightningAddr + 8;

        for (int i = iLightningAddr; i < iTop; ++i)
            iSum |= data[i];

        if (iSum == 0)
            return CompilationMode::Native;
        else
            return CompilationMode::CLR;
    }
    catch(Exception ^e) {
        throw(e);
    }
}

int main() {
    array<String^>^ args = Environment::GetCommandLineArgs();

    if (args->Length < 2) {
        Console::WriteLine("USAGE : detect_clr <assembly_name>\n");
        return -1;
    }

    Console::WriteLine("{0} is compiled {1}", args[1], IsManaged(args[1]));
}

```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

HOW TO : 將原生 DLL 加入全域組件快取

2019/12/2 • [Edit Online](#)

您可以將原生 DLL (而非 COM) 放入全域組件快取。

範例

/ASSEMBLYLINKRESOURCE可讓您的組件中嵌入原生 DLL。

如需詳細資訊，請參閱 [/ASSEMBLYLINKRESOURCE \(連結到 .NET Framework 資源\)](#)。

```
/ASSEMBLYLINKRESOURCE:MyComponent.dll
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：以原生類型存放實值類型的參考

2019/12/10 • [Edit Online](#)

使用已裝箱類型上的 `gcroot` 來保存原生類型中實值型別的參考。

範例

```
// reference_to_value_in_native.cpp
// compile with: /clr
#using <mscorlib.dll>
#include <vcclr.h>

using namespace System;

public value struct V {
    String ^str;
};

class Native {
public:
    gcroot< V^ > v_handle;
};

int main() {
    Native native;
    V v;
    native.v_handle = v;
    native.v_handle->str = "Hello";
    Console::WriteLine("String in V: {0}", native.v_handle->str);
}
```

```
String in V: Hello
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：在 Unmanaged 記憶體中存放物件參考

2020/11/2 • [Edit Online](#)

您可以使用 `gcroot GCHandle` 來將 CLR 物件參考保存在非受控記憶體中。或者，您也可以 `GCHandle` 直接使用。

範例

```
// hold_object_reference.cpp
// compile with: /clr
#include "gcroot.h"
using namespace System;

#pragma managed
class StringWrapper {

private:
    gcroot<String ^> x;

public:
    StringWrapper() {
        String ^ str = gcnew String("ManagedString");
        x = str;
    }

    void PrintString() {
        String ^ targetStr = x;
        Console::WriteLine("StringWrapper::x == {0}", targetStr);
    }
};

#pragma unmanaged
int main() {
    StringWrapper s;
    s.PrintString();
}
```

```
StringWrapper::x == ManagedString
```

`GCHandle` 提供在非受控記憶體中保存受管理物件參考的方法。您可以使用 `Alloc` 方法來建立受管理物件的不透明控制碼，並 `Free` 加以釋放。此外，此 `Target` 方法可讓您從 managed 程式碼中的控制碼取得物件參考。

```
// hold_object_reference_2.cpp
// compile with: /clr
using namespace System;
using namespace System::Runtime::InteropServices;

#pragma managed
class StringWrapper {
    IntPtr m_handle;
public:
    StringWrapper() {
        String ^ str = gcnew String("ManagedString");
        m_handle = static_cast<IntPtr>(GCHandle::Alloc(str));
    }
    ~StringWrapper() {
        static_cast<GCHandle>(m_handle).Free();
    }

    void PrintString() {
        String ^ targetStr = safe_cast< String ^ >(static_cast<GCHandle>(m_handle).Target);
        Console::WriteLine("StringWrapper::m_handle == {0}", targetStr);
    }
};

#pragma unmanaged
int main() {
    StringWrapper s;
    s.PrintString();
}
```

```
StringWrapper::m_handle == ManagedString
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：偵測 /clr 編譯

2019/12/10 • [Edit Online](#)

使用 `_MANAGED` 或 `_M_CEE` 宏來查看模組是否使用 `/clr` 進行編譯。如需詳細資訊，請參閱 [/clr \(Common Language Runtime 編譯\)](#)。

如需宏的詳細資訊，請參閱 [預先定義的宏](#)。

範例

```
// detect_CLR_compilation.cpp
// compile with: /clr
#include <stdio.h>

int main() {
    #if (_MANAGED == 1) || (_M_CEE == 1)
        printf_s("compiling with /clr\n");
    #else
        printf_s("compiling without /clr\n");
    #endif
}
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：在 System::Guid 和 _GUID 之間轉換

2019/12/10 • [Edit Online](#)

下列程式碼範例示範如何在 Guid 和 _GUID 之間進行轉換。

範例

```
// convert_guids.cpp
// compile with: /clr
#include <windows.h>
#include <stdio.h>

using namespace System;

Guid FromGUID( _GUID& guid ) {
    return Guid( guid.Data1, guid.Data2, guid.Data3,
                 guid.Data4[ 0 ], guid.Data4[ 1 ],
                 guid.Data4[ 2 ], guid.Data4[ 3 ],
                 guid.Data4[ 4 ], guid.Data4[ 5 ],
                 guid.Data4[ 6 ], guid.Data4[ 7 ] );
}

_GUID ToGUID( Guid& guid ) {
    array<Byte>^ guidData = guid.ToByteArray();
    pin_ptr<Byte> data = &(guidData[ 0 ]);

    return *(_GUID *)data;
}

int main() {
    _GUID ng = {0x11111111,0x2222,0x3333,0x44,0x55,0x55,0x55,0x55,0x55,0x55};
    Guid mg;

    Console::WriteLine( (mg = FromGUID( ng )).ToString() );
    _GUID ng2 = ToGUID( mg );

    printf_s( "%x-%x-%x-", ng2.Data1, ng2.Data2, ng2.Data3 );
    for (int i = 0 ; i < 8 ; i++) {
        if (i == 2)
            printf_s("-");
        printf_s("%x", ng2.Data4[i]);
    }
    printf_s("\n");
}
```

```
11111111-2222-3333-4455-555555555555
11111111-2222-3333-4455-555555555555
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

How to: Specify an out parameter

2020/2/1 • [Edit Online](#)

This sample shows how to specify that a function parameter is an `out` parameter, and how to call that function from a C# program.

An `out` parameter is specified in C++ by using [OutAttribute](#).

範例

The first part of this sample creates a C++ DLL. It defines a type that contains a function with an `out` parameter.

```
// cpp_out_param.cpp
// compile with: /LD /clr
using namespace System;
public value struct TestStruct {
    static void Test([Runtime::InteropServices::Out] String^ %s) {
        s = "a string";
    }
};
```

This source file is a C# client that consumes the C++ component created in the previous example.

```
// cpp_out_param_2.cs
// compile with: /reference:cpp_out_param.dll
using System;
class TestClass {
    public static void Main() {
        String t;
        TestStruct.Test(out t);
        System.Console.WriteLine(t);
    }
}
```

```
a string
```

請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

如何：在 /clr 編譯中使用原生類型

2020/11/2 • [Edit Online](#)

您可以在 /clr 編譯中定義原生類型，而且該原生類型在元件內的任何使用都是有效的。不過，原生類型將無法從參考的中繼資料使用。

每個元件都必須包含它將使用的每個原生類型的定義。

如需詳細資訊，請參閱 [/clr \(Common Language Runtime 編譯\)](#)。

範例

這個範例會建立定義和使用原生類型的元件。

```
// use_native_type_in_clr.cpp
// compile with: /clr /LD
public struct NativeClass {
    static int Test() { return 98; }
};

public ref struct ManagedClass {
    static int i = NativeClass::Test();
    void Test() {
        System::Console::WriteLine(i);
    }
};
```

這個範例會定義使用元件的用戶端。請注意，除非在編譯單位中定義，否則存取原生類型會是錯誤。

```
// use_native_type_in_clr_2.cpp
// compile with: /clr
#using "use_native_type_in_clr.dll"
// Uncomment the following 3 lines to resolve.
// public struct NativeClass {
//     static int Test() { return 98; }
// };

int main() {
    ManagedClass x;
    x.Test();

    System::Console::WriteLine(NativeClass::Test()); // C2653
}
```

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：以原生類型宣告控制代碼

2020/11/2 • [Edit Online](#)

您無法以原生類型宣告控制碼類型。vcclr 提供型別安全包裝函式範本 `gcroot`，以從 C++ 堆積參考 CLR 物件。此範本可讓您以原生類型內嵌虛擬控制碼，並將其視為基礎類型。在大多數情況下，您可以使用 `gcroot` 物件做為內嵌類型，而不需要任何轉換。不過，在中，您必須使用 `static_cast` 來取得基礎 managed 參考。

此 `gcroot` 範本會使用實值類別 `System::Runtime::System.Runtime.InteropServices.OutAttribute::GCHandle` 的功能來執行，這會將「控制碼」提供給垃圾收集的堆積。請注意，處理常式本身不會進行垃圾收集，而且當類別中的函式不再使用時，會釋出 `gcroot`（無法以手動方式呼叫這個函式）。如果您在 `gcroot` 原生堆積上具現化物件，您必須在該資源上呼叫 `delete`。

執行時間會維護控制碼和 CLR 物件（其參考）之間的關聯。當 CLR 物件與垃圾收集堆積一起移動時，控制碼會傳回物件的新位址。變數在指派給範本之前，不需要先釘選 `gcroot`。

範例

此範例說明如何 `gcroot` 在原生堆疊上建立物件。

```
// mcpp_gcroot.cpp
// compile with: /clr
#include <vcclr.h>
using namespace System;

class CppClass {
public:
    gcroot<String^> str; // can use str as if it were String^
    CppClass() {}
};

int main() {
    CppClass c;
    c.str = gcnew String("hello");
    Console::WriteLine( c.str ); // no cast required
}
```

```
hello
```

此範例顯示如何 `gcroot` 在原生堆積上建立物件。

```

// mcpp_gcroot_2.cpp
// compile with: /clr
// compile with: /clr
#include <vcclr.h>
using namespace System;

struct CppClass {
    gcroot<String ^> * str;
    CppClass() : str(new gcroot<String ^>) {}

    ~CppClass() { delete str; }

};

int main() {
    CppClass c;
    *c.str = gcnew String("hello");
    Console::WriteLine( *c.str );
}

```

hello

這個範例會示範如何使用 `gcroot` 來保存實值型別的參考，(不會使用在已裝箱的型別上) 于原生類型中的參考型別 `gcroot`。

```

// mcpp_gcroot_3.cpp
// compile with: /clr
#include < vcclr.h >
using namespace System;

public value struct V {
    String^ str;
};

class Native {
public:
    gcroot< V^ > v_handle;
};

int main() {
    Native native;
    V v;
    native.v_handle = v;
    native.v_handle->str = "Hello";
    Console::WriteLine("String in V: {0}", native.v_handle->str);
}

```

String in V: Hello

另請參閱

[使用 c + + Interop \(隱含 PInvoke\)](#)

如何：包裝原生類別以供 C# 使用

2020/3/18 • [Edit Online](#)

這個範例示範如何包裝原生C++類別，使其可供以C#或其他 .net 語言撰寫的程式碼使用。

範例

```
// wrap_native_class_for_mgd_consumption.cpp
// compile with: /clr /LD
#include <windows.h>
#include <vcclr.h>
#using <System.dll>

using namespace System;

class UnmanagedClass {
public:
    LPCWSTR GetPropertyA() { return 0; }
    void MethodB( LPCWSTR ) {}

};

public ref class ManagedClass {
public:
    // Allocate the native object on the C++ Heap via a constructor
    ManagedClass() : m_Impl( new UnmanagedClass() ) {}

    // Deallocate the native object on a destructor
    ~ManagedClass() {
        delete m_Impl;
    }

protected:
    // Deallocate the native object on the finalizer just in case no destructor is called
    !ManagedClass() {
        delete m_Impl;
    }

public:
    property String ^ get_PropertyA {
        String ^ get() {
            return gcnew String( m_Impl->GetPropertyA() );
        }
    }

    void MethodB( String ^ theString ) {
        pin_ptr<const WCHAR> str = PtrToStringChars(theString);
        m_Impl->MethodB(str);
    }

private:
    UnmanagedClass * m_Impl;
};

}
```

另請參閱

[使用 C++ Interop \(隱含 PInvoke\)](#)

純粹的和可驗證的程式碼 (C++/CLI)

2019/12/2 • [Edit Online](#)

如需.NET 程式設計，視覺效果C++在 Visual Studio 2017 支援使用混合的組件的建立/[clr \(Common Language Runtime 編譯\)](#)編譯器選項。[/Clr: pure](#)並：[safe](#)選項是在 Visual Studio 2015 中已被取代，而且不支援的 Visual Studio 2017 中。如果您的程式碼必須是安全或驗證的則我們建議您移植到C#。

混合 (/ clr)

混合組件 (以編譯 /clr)、包含非受控和受控組件，讓他們能夠使用.NET 功能，但仍然包含原生程式碼。這可讓應用程式和元件更新，而不需要重寫整個專案中使用.NET 功能。使用視覺效果C++混合 managed 和原生程式碼，以這種方式呼叫C+++Interop。如需詳細資訊，請參閱 <[混合 \(原生和 Managed\) 組件並原生和.NET 互通性](#)>。

從 managed 組件對原生 DLL 透過 P/Invoke 的呼叫會進行編譯，但在執行階段根據安全性設定可能會失敗。

還有一個程式碼撰寫案例，將會通過編譯器，但會導致無法驗證的組件：呼叫虛擬函式，透過使用範圍解析運算子的物件執行個體。例如：`MyObj -> A::VirtualFunction();`。

另請參閱

- [以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

如何：建立可驗證C++專案 (C++/CLI)

2019/12/2 • [Edit Online](#)

視覺化C++應用程式精靈不會建立可驗證的專案。

IMPORTANT

已被取代的 visual Studio 2015 和 Visual Studio 2017 不支援 /clr: pure 並 /clr: safe 建立可驗證的專案。如果您需要可驗證程式碼，我們建議您將轉譯成 C# 程式碼。

不過，如果您使用較舊版本的 MicrosoftC++ 支援的編譯器工具組 /clr: pure 並 /clr: safe，專案可以轉換為可供驗證。本主題描述如何設定專案屬性並修改專案來源檔，來轉換您的 Visual StudioC++ 專案來產生可驗證的應用程式。

編譯器和連結器設定

根據預設，.NET 專案使用 /clr 編譯器旗標，並設定目標 x86 硬體連結器。可驗證的程式碼，您必須使用 /clr: safe 旗標，而且您必須指示連結器產生的 MSIL，而不是原生機器指令。

若要變更的編譯器和連結器設定

- 顯示專案屬性頁。如需詳細資訊，請參閱 <[設定編譯器和組建屬性](#)>。
- 在上一般頁面組態屬性節點，設定 Common Language Runtime 支援屬性設安全 MSIL Common Language 執行階段支援 (/: safe)。
- 在上進階頁面連結器節點，設定 CLR 映像類型屬性設強制安全 IL 映像 (/ /clrimagetype: safe)。

移除原生資料類型

因為原生資料類型非驗證的即使實際上使用，您必須移除所有標頭檔包含原生型別。

NOTE

下列程序適用於 Windows Forms 應用程式 (.NET) 和主控台應用程式 (.NET) 的專案。

若要移除原生資料類型的參考

- 標記為註解 Stdafx.h 檔案中的所有項目。

設定進入點

可驗證的應用程式無法使用 C 執行階段程式庫 (CRT)，因為它們不能相依於 CRT 呼叫 main 函式的標準的進入點。這表示您必須明確地提供給連結器一開始要呼叫的函式的名稱。（在本例中，main () 而不是 main () 或 _tmain() 用來表示非 CRT 進入點，但此名稱必須明確指定的進入點，因為是任意的）。

NOTE

下列程序適用於主控台應用程式 (.NET) 專案。

若要設定的進入點

- 將 _tmain() 變更專案的主要.cpp 檔案中的 main ()。

- 顯示專案屬性頁。如需詳細資訊，請參閱 <[設定編譯器和組建屬性](#)>。
- 在上進階頁面連結器節點中，輸入 `Main` 作為進入點屬性值。

另請參閱

- 純粹的和可驗證的程式碼 (C++/CLI)

使用可驗證的組件搭配 SQL Server (C++/CLI)

2019/12/2 • [Edit Online](#)

擴充預存程序，封裝成動態連結程式庫 (DLL)，可用來擴充 SQL Server 功能，透過開發視覺效果的函式 C++。擴充預存程序會實作為 DLL 內的函式。除了函數以外，擴充預存程序也可以定義使用者定義型別和彙總函式（例如，SUM 或 AVG）。

當用戶端執行擴充預存程序時，SQL Server DLL 的搜尋與擴充預存程序相關聯，並載入 DLL。SQL Server 會呼叫要求的擴充預存程序，並在指定的安全性內容下執行它。擴充預存程序，然後將結果集，並傳回至伺服器的參數。

SQL Server transact-sql (T-SQL) 可讓您可驗證的組件安裝到 SQL Server 提供延伸模組。SQL Server 權限集合指定的安全性內容，含有下列層級的安全性：

- 不受限制的模式：執行程式碼自行承擔風險；程式碼可能沒有可驗證的型別安全。
- 安全模式：執行可驗證的型別安全程式碼。使用 /clr: safe 編譯。

IMPORTANT

已被取代的 visual Studio 2015 和 Visual Studio 2017 不支援 /clr: pure 並 /clr: safe 建立可驗證的專案。如果您需要可驗證程式碼，我們建議您將轉譯成 C# 程式碼。

建立和可驗證的組件載入到 SQL Server，請使用 CREATE ASSEMBLY 和 DROP ASSEMBLY TRANSACT-SQL 命令，如下所示：

```
CREATE ASSEMBLY <assemblyName> FROM <'Assembly UNC Path'> WITH  
PERMISSION_SET <permissions>  
DROP ASSEMBLY <assemblyName>
```

PERMISSION_SET 命令指定的安全性內容中，並可以有不受限制、保險箱或擴充的值。

此外，您可以使用 CREATE FUNCTION 命令繫結至類別中的方法名稱：

```
CREATE FUNCTION <FunctionName>(<FunctionParams>)  
RETURNS returnType  
[EXTERNAL NAME <AssemblyName>:<ClassName>:<StaticMethodName>]
```

範例

下列 SQL 指令碼 (比方說，具名 "MyScript.sql") 將組件載入至 SQL Server，並提供類別的方法：

```
-- Create assembly without external access
drop assembly stockNoEA
go
create assembly stockNoEA
from
'c:\stockNoEA.dll'
with permission_set = safe

-- Create function on assembly with no external access
drop function GetQuoteNoEA
go
create function GetQuoteNoEA(@sym nvarchar(10))
returns real
external name stockNoEA:StockQuotes::GetQuote
go

-- To call the function
select dbo.GetQuoteNoEA('MSFT')
go
```

可以以互動方式執行 SQL 指令碼，在 SQL Query Analyzer 或 sqlcmd.exe 公用程式命令列。下列命令列連接到 MyServer，使用預設的資料庫、使用信任的連接，輸入 MyScript.sql 及輸出 MyResult.txt。

```
sqlcmd -S MyServer -E -i myScript.sql -o myResult.txt
```

另請參閱

[類別和結構](#)

將專案從混合模式轉換為純中繼語言

2020/3/25 • [Edit Online](#)

根據預設C++，所有Visual CLR專案都會連結到C執行時間程式庫。因此，這些專案會分類為混合模式應用程式，因為它們會將機器碼與以common language runtime (managed 程式碼)為目標的程式碼結合。編譯時，它們會編譯成中繼語言(IL)，也稱為Microsoft中繼語言(MSIL)。

IMPORTANT

Visual Studio 2015 已淘汰，Visual Studio 2017 不再支援為CLR應用程式建立 /clr: pure或 /clr: safe程式碼。如果您需要純或safe元件，建議您將應用程式轉譯為C#。

如果您使用的是支援 /clr: pure或C++ /Clr: safe的舊版Microsoft編譯器工具組，您可以使用這個程式將程式碼轉換成純MSIL：

將混合模式應用程式轉換為純中繼語言

1. 移除C執行時間程式庫(CRT)的連結：

- a. 在定義應用程式進入點的.cpp檔案中，將進入點變更為Main()。使用Main()表示您的專案未連結至CRT。
- b. 在方案總管中，以滑鼠右鍵按一下您的專案，然後選取快捷方式功能表上的[屬性]，以開啟應用程式的屬性頁。
- c. 在連結器的[Advanced project]屬性頁中，選取進入點，然後在此欄位中輸入Main。
- d. 對於主控台應用程式，請在連結器的[系統專案]屬性頁中選取[子系統]欄位，並將其變更為[主控台(/SUBSYSTEM:主控台)]。

NOTE

您不需要為Windows Forms應用程式設定此屬性，因為[]欄位預設會設定為[windows (/SUBSYSTEM:windows)]。

- e. 在stdafx.h中，將所有#include語句標記為批註。例如，在主控台應用程式中：

```
// #include <iostream>
// #include <tchar.h>
```

-或-

例如，在Windows Forms應用程式中：

```
// #include <stdlib.h>
// #include <malloc.h>
// #include <memory.h>
// #include <tchar.h>
```

- f. 對於Windows Forms應用程式，在form1.vb中，將參考Windows的#include語句標記為批註。例如：

```
// #include <windows.h>
```

2. 將下列程式碼新增至 *stdafx.h*:

```
#ifndef __FLTUSED__
#define __FLTUSED__
    extern "C" __declspec(selectany) int _fltused=1;
#endif
```

3. 移除所有非受控類型:

在適當的情況下，將非受控類型取代為[系統命名空間](#)中結構的參考。下表列出常見的 managed 類型：

類型	說明
Boolean	代表布林值。
Byte	代表 8 位元不帶正負號的整數。
Char	代表 Unicode 字元。
DateTime	表示時間的瞬間，通常以一天的日期和時間表示。
Decimal	代表十進位數字。
Double	代表雙精度浮點數。
Guid	代表全域唯一識別項 (GUID)。
Int16	代表 16 位元帶正負號的整數。
Int32	代表 32 位元帶正負號的整數。
Int64	代表 64 位元帶正負號的整數。
IntPtr	平台專用的類型，用以代表指標或控制代碼。
SByte	代表 8 位元帶正負號的整數。
Single	代表單精確度浮點數。
TimeSpan	代表時間間隔。
UInt16	代表 16 位元不帶正負號的整數。
UInt32	代表 32 位元不帶正負號的整數。
UInt64	代表 64 位元不帶正負號的整數。
UIntPtr	平台專用的類型，用以代表指標或控制代碼。

空位	表示不會傳回值的方法。也就是說，此方法具有 void 傳回型別。

序列化 (C++/CLI)

2019/12/10 • [Edit Online](#)

`SerializableAttribute` 和 `NonSerializedAttribute` 類別支援序列化(將物件或成員的狀態儲存到永久性媒體的處理常式)(包括個別欄位或屬性)。

備註

將`SerializableAttribute`自訂屬性套用至 managed 類別，以序列化整個類別，或只套用至特定欄位或屬性，以序列化 managed 類別的各個部分。使用`NonSerializedAttribute`自訂屬性來豁免 managed 類別的欄位或屬性，使其無法序列化。

範例

描述

在下列範例中，類別 `MyClass` (和屬性 `m_nCount`)標示為可序列化。不過，`m_nData` 屬性不會如非序列化自訂屬性所指示進行序列化：

程式碼

```
// serialization_and_mcpp.cpp
// compile with: /LD /clr
using namespace System;

[ Serializable ]
public ref class MyClass {
public:
    int m_nCount;
private:
    [ NonSerialized ]
    int m_nData;
};
```

註解

請注意，這兩個屬性都可以使用其「簡短名稱」(可序列化和非序列化)來參考。這會在套用屬性中進一步說明。

請參閱

[以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

Friend 組件 (C++)

2020/11/2 • [Edit Online](#)

針對適用的執行時間，`friend`元件語言功能可讓元件元件中的命名空間範圍或全域範圍內的類型，可供一或多個用戶端元件或 `.netmodule` 存取。

所有執行階段

備註

(不是所有執行階段都有支援這個語言功能)。

Windows 執行階段

備註

(Windows 執行階段不支援這個語言功能。)

需求

編譯器選項: `/ZW`

Common Language Runtime

備註

讓組件元件中在命名空間範圍或全域範圍的類型可存取用戶端組件或 `.netmodule`

1. 在元件中，指定組件屬性 `InternalsVisibleToAttribute`，並傳遞將存取元件中在命名空間範圍或全域範圍之類型的用戶端組件或 `.netmodule` 的名稱。您可以藉由指定其他屬性來指定多個用戶端組件或 `.netmodule`。
2. 在用戶端元件或 `.netmodule` 中，當您使用來參考元件元件時，`#using` 請傳遞 `as_friend` 屬性。如果您為 `as_friend` 未指定的元件指定屬性 `InternalsVisibleToAttribute`，則會在您嘗試存取元件中命名空間範圍或全域範圍的類型時，擲回執行時間例外狀況。

如果包含屬性的元件沒有 `InternalsVisibleToAttribute` 強式名稱，但使用該屬性的用戶端元件執行了，就會產生組建錯誤 `as_friend`。

雖然在命名空間範圍和全域範圍的類型可以讓用戶端組件或 `.netmodule` 知道，成員存取範圍仍然有效。例如，您無法存取私人成員。

必須明確授與組件中所有類型的存取權。例如，如果組件 C 參考組件 B，而組件 B 具有組件 A 中所有類型的存取權，則組件 C 沒有組件 A 中所有類型的存取權。

如需如何簽署(也就是如何提供強式名稱)(使用 Microsoft C++ 編譯器建立的元件)的相關資訊，請參閱[強式名稱元件\(元件簽署\)\(C++/CLI\)](#)。

對於使用 friend 組件功能的替代方案，您可以使用 `StrongNameIdentityPermission` 限制對個別類型的存取。

需求

編譯器選項: `/clr`

範例

下列程式碼範例會定義可指定具有元件類型存取權之用戶端組件的元件。

```
// friend_assemblies.cpp
// compile by using: /clr /LD
using namespace System::Runtime::CompilerServices;
using namespace System;
// an assembly attribute, not bound to a type
[assembly:InternalsVisibleTo("friend_assemblies_2")];

ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};
```

下一個程式碼範例會存取元件中的私用類型。

```
// friend_assemblies_2.cpp
// compile by using: /clr
#using "friend_assemblies.dll" as_friend

int main() {
    Class1 ^ a = gcnew Class1;
    a->Test_Public();
}
```

```
Class1::Test_Public
```

下一個程式碼範例會定義元件，但不指定具有元件中各類型之存取權的用戶端組件。

請注意，元件是使用 `/opt: noref` 所連結。這可確保私用類型在元件的中繼資料中發出，當 `InternalsVisibleTo` 屬性存在時則不需要。如需詳細資訊，請參閱[/opt \(優化\)](#)。

```
// friend_assemblies_3.cpp
// compile by using: /clr /LD /link /opt:noref
using namespace System;

ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};
```

下列程式碼範例會定義嘗試存取元件中私用類型（其未將存取權授與其私用類型）的用戶端。由於執行階段的行為，如果您要攔截例外狀況，您必須嘗試存取在 helper 函式的私用類型。

```

// friend_assemblies_4.cpp
// compile by using: /clr
#using "friend_assemblies_3.dll" as_friend
using namespace System;

void Test() {
    Class1 ^ a = gcnew Class1;
}

int main() {
    // to catch this kind of exception, use a helper function
    try {
        Test();
    }
    catch(MethodAccessException ^ e) {
        Console::WriteLine("caught an exception");
    }
}

```

caught an exception

下一個程式碼範例顯示如何建立可指定具有元件類型存取權之用戶端組件的強式名稱元件。

```

// friend_assemblies_5.cpp
// compile by using: /clr /link /keyfile:friend_assemblies.snk
using namespace System::Runtime::CompilerServices;
using namespace System;
// an assembly attribute, not bound to a type

[assembly:InternalsVisibleTo("friend_assemblies_6,
PublicKey=002400000480000094000000060200000240005253413100040000010001000bf45d77fd991f3bff0ef51af48a12d35699e
04616f27ba561195a69ebd3449c345389dc9603d65be8cd1987bc7ea48bdda35ac7d57d3d82c666b7fc1a5b79836d139ef0ac8c4e715434
211660f481612771a9f7059b9b742c3d8af00e01716ed4b872e6f1be0e94863eb5745224f0deaba5b137624d7049b6f2d87fba639fc5")]
;

private ref class Class1 {
public:
    void Test_Public() {
        Console::WriteLine("Class1::Test_Public");
    }
};

```

請注意，該元件必須指定其公開金鑰。建議您依序在命令提示字元執行下列命令，以建立金鑰組，並取得公開金鑰：

sn-d friend_assemblies .snk

sn-k friend_assemblies .snk

sn-i friend_assemblies .snk friend_assemblies .snk

sn-pc friend_assemblies .snk 金鑰. publickey

sn -tp key.publickey

下一個程式碼範例會存取在強式名稱元件中的私用類型。

```
// friend_assemblies_6.cpp
// compile by using: /clr /link /keyfile:friend_assemblies.snk
#using "friend_assemblies_5.dll" as_friend

int main() {
    Class1 ^ a = gcnew Class1;
    a->Test_Public();
}
```

```
Class1::Test_Public
```

另請參閱

[執行階段平台的元件延伸模組](#)

Managed 類型 (C++/CLI)

2020/11/2 • [Edit Online](#)

Visual C++ 允許透過 managed 類型存取 .NET 功能，以提供 common language runtime 功能的支援，並受限於執行時間的優點和限制。

Managed 類型和 main 函式

使用撰寫應用程式時 `/clr`，`main()` 函數的引數不能是 managed 類型。

適當簽章的範例如下：

```
// managed_types_and_main.cpp
// compile with: /clr
int main(int, char*[], char*[]) {}
```

C++ 原生類型的 .NET Framework 對等專案

下表顯示內建 Visual C++ 類型的關鍵字，這些是 System 命名空間中預先定義類型的別名。

VISUAL C++	.NET FRAMEWORK
<code>void</code>	<code>System.Void</code>
<code>bool</code>	<code>System.Boolean</code>
<code>signed char</code>	<code>System.SByte</code>
<code>unsigned char</code>	<code>System.Byte</code>
<code>wchar_t</code>	<code>System.Char</code>
<code>short</code> 和 ** <code>signed short</code>	<code>System.Int16</code>
<code>unsigned short</code>	<code>System.UInt16</code>
<code>int</code> 、 <code>signed int</code> 、 <code>long</code> 和 ** <code>signed long</code>	<code>System.Int32</code>
<code>unsigned int</code> 和 ** <code>unsigned long</code>	<code>System.UInt32</code>
<code>__int64</code> 和 ** <code>signed __int64</code>	<code>System.Int64</code>
<code>unsigned __int64</code>	<code>System.UInt64</code>
<code>float</code>	<code>System.Single</code>
<code>double</code> 和 ** <code>long double</code>	<code>System.Double</code>

如需編譯器選項預設為或的詳細資訊 `signed char` `unsigned char`，請參閱 [/J \(預設 char 類型為 unsigned\)](#)。

在原生類型中嵌套實數值型別的版本問題

請考慮用來建立用戶端元件的帶正負號(強式名稱)元件元件。元件包含實值型別，可在用戶端中用來做為原生等位、類別或陣列成員的型別。如果元件的未來版本變更了實數值型別的大小或配置，則必須重新編譯用戶端。

使用 `sn.exe ()` 建立 keyfile `sn -k mykey.snk`。

範例

下列範例是元件。

```
// nested_value_types.cpp
// compile with: /clr /LD
using namespace System::Reflection;
[assembly:AssemblyVersion("1.0.0.*"),
assembly:AssemblyKeyFile("mykey.snk")];

public value struct S {
    int i;
    void Test() {
        System::Console::WriteLine("S.i = {0}", i);
    }
};
```

範例

這個範例是用戶端：

```
// nested_value_types_2.cpp
// compile with: /clr
#using <nested_value_types.dll>

struct S2 {
    S MyS1, MyS2;
};

int main() {
    S2 MyS2a, MyS2b;
    MyS2a.MyS1.i = 5;
    MyS2a.MyS2.i = 6;
    MyS2b.MyS1.i = 10;
    MyS2b.MyS2.i = 11;

    MyS2a.MyS1.Test();
    MyS2a.MyS2.Test();
    MyS2b.MyS1.Test();
    MyS2b.MyS2.Test();
}
```

輸出

```
S.i = 5
S.i = 6
S.i = 10
S.i = 11
```

評價

不過，如果您在 `nested_value_types.cpp` 中將另一個成員加入至 `struct S` (例如 `double d;`)，並在不重新編譯用戶端的情況下重新編譯元件，則結果會是未處理的例外狀況(類型為 `System.IO.FileLoadException`)。

如何 : 測試是否相等

在下列範例中，使用 Managed Extensions for C++ 的相等測試是根據控制碼所參考的內容。

範例

```
// mcppv2_equality_test.cpp
// compile with: /clr /LD
using namespace System;

bool Test1() {
    String ^ str1 = "test";
    String ^ str2 = "test";
    return (str1 == str2);
}
```

此程式的 IL 顯示，傳回值是使用 op_Equality 的呼叫來執行。

```
IL_0012: call     bool [mscorlib]System.String::op_Equality(string,
                                         string)
```

如何 : 診斷和修正元件相容性問題

本主題說明在編譯時期參考的元件版本與執行時間所參考的元件版本不相符時，會發生什麼情況，以及如何避免這個問題。

編譯元件時，可能會使用語法來參考其他元件 `#using`。在編譯期間，編譯器會存取這些元件。這些元件中的資訊是用來做出優化決策。

不過，如果參考的元件已變更並重新編譯，而且您不重新編譯相依于它的參考元件，則元件可能仍然不相容。相對於新元件版本而言，第一次有效的優化決策可能不正確。可能會因為這些不相容而發生各種執行階段錯誤。在這種情況下，並不會產生特定的例外狀況。在執行時間報告失敗的方式，取決於造成問題之程式碼變更的本質。

只要針對產品的發行版本重建整個應用程式，這些錯誤就應該不會在最終的實際執行程式碼中發生問題。發行至公用的元件應該以官方版本號碼標示，這可確保避免這些問題。如需詳細資訊，請參閱[組件版本控制](#)。

診斷和修正不相容錯誤

1. 如果您遇到執行時間例外狀況，或是在程式碼中發生的其他錯誤情況會參考另一個元件，而且沒有其他識別的原因，您可能會處理過期的元件。
2. 首先，隔離並重現例外狀況或其他錯誤情況。因過期的例外狀況而發生的問題應該可以重現。
3. 檢查應用程式中所參考之任何元件的時間戳記。
4. 如果任何參考元件的時間戳記晚于應用程式上次編譯的時間戳記，則您的應用程式已過期。如果發生這種情況，請使用最新的元件重新編譯您的應用程式，並進行任何必要的程式碼變更。
5. 重新執行應用程式，並執行重現問題的步驟，並確認不會發生例外狀況。

範例

下列程式會藉由減少方法的存取範圍並嘗試在另一個元件中存取該方法，來說明問題，而不需要重新編譯。請先嘗試編譯 `changeaccess.cpp`。這是所參考的元件，將會變更。然後編譯 `referencing.cpp`。編譯成功。現在，減少所呼叫方法的存取範圍。使用旗標重新編譯 `changeaccess.cpp /DCHANGE_ACCESS`。這會讓方法受到保護，而不是私用，因此可以更長的合法方式呼叫。若未 `referencing.exe` 重新編譯，請重新執行應用程式。

`MethodAccessException` 將會產生例外狀況。

```

// changeaccess.cpp
// compile with: /clr:safe /LD
// After the initial compilation, add /DCHANGE_ACCESS and rerun
// referencing.exe to introduce an error at runtime. To correct
// the problem, recompile referencing.exe

public ref class Test {
#if defined(CHANGE_ACCESS)
protected:
#else
public:
#endif

    int access_me() {
        return 0;
    }

};


```

```

// referencing.cpp
// compile with: /clr:safe
#using <changeaccess.dll>

// Force the function to be inline, to override the compiler's own
// algorithm.
__forceinline
int CallMethod(Test^ t) {
    // The call is allowed only if access_me is declared public
    return t->access_me();
}

int main() {
    Test^ t = gcnew Test();
    try
    {
        CallMethod(t);
        System::Console::WriteLine("No exception.");
    }
    catch (System::Exception ^ e)
    {
        System::Console::WriteLine("Exception!");
    }
    return 0;
}

```

另請參閱

[使用 c++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

[與其他 .NET 語言的互通性 \(c++/CLI\)](#)

[Managed 類型 \(c++/CLI\)](#)

[#using 指示詞](#)

反映 (C++/CLI)

2019/12/2 • [Edit Online](#)

反映可讓您在執行階段檢查已知的資料類型。反映可讓資料類型的列舉中指定的組件，並可以探索到指定的類別或實值類型的成員。這是不論是否已知或在編譯時期參考的類型，則為 true。這可讓反映實用的功能進行開發和程式碼管理工具。

請注意，所提供的組件名稱的強式名稱（請參閱[建立和使用強式名稱組件](#)），包括組件版本、文化特性，以及簽章的資訊。也請注意，在其中定義資料類型的命名空間的名稱可以擷取，以及基底類別的名稱。

若要存取反映功能的最常見方式是透過[GetType](#)方法。這個方法藉由提供[System.Object](#)，從衍生所有記憶體回收的類別。

NOTE

反映在使用 Microsoft 建置的.exeC++ 編譯器才允許使用.exe 已內建 /clr: pure 或 /clr: safe 編譯器選項。/Clr: pure 並 /clr: safe 編譯器選項為已被取代，在 Visual Studio 2015 和 Visual Studio 2017 中無法使用。請參閱[/clr \(Common Language Runtime 編譯\)](#)如需詳細資訊。

如需詳細資訊，請參閱[System.Reflection](#)。

範例 : GetType

`GetType` 方法傳回的指標[Type](#)類別的物件，其中描述在當物件為基礎的型別。（型別物件不包含任何執行個體的特定資訊。）一個這類項目是型別，可以顯示，如下所示的完整名稱：

請注意類型名稱包含類型定義所在，包括命名空間的完整範圍，而且它會顯示在.NET 的語法，加上範圍解析運算子為一個點。

```
// vcpp_reflection.cpp
// compile with: /clr
using namespace System;
int main() {
    String ^ s = "sample string";
    Console::WriteLine("full type name of '{0}' is '{1}'", s, s->GetType());
}
```

```
full type name of 'sample string' is 'System.String'
```

範例 : boxed 實值類型

實值型別可以搭配 `GetType` 函式，但它們必須先將它 boxed。

```
// vcpp_reflection_2.cpp
// compile with: /clr
using namespace System;
int main() {
    Int32 i = 100;
    Object ^ o = i;
    Console::WriteLine("type of i = '{0}'", o->GetType());
}
```

```
type of i = 'System.Int32'
```

範例: typeid

如同 `GetType` 方法中, `typeid` 運算子會傳回的指標類型物件, 因此這個代碼表示的型別名稱 `System.Int32`。顯示型別名稱是反映的最基本功能, 但可能更有用的技巧是要檢查或探索有效的列舉型別值。做法是使用靜態 `enum::getNames` 函式, 它會傳回字串陣列, 每一個都包含文字格式的列舉值。下列範例會擷取描述的值列舉值的字串陣列選項(CLR) 列舉, 並在迴圈中顯示它們。

如果是第四個選項新增至選項列舉型別, 此程式碼會報告新的選項, 而不必重新編譯, 即使在不同的組件中已定義列舉。

```
// vcpp_reflection_3.cpp
// compile with: /clr
using namespace System;

enum class Options {    // not a native enum
    Option1, Option2, Option3
};

int main() {
    array<String^>^ names = Enum::GetNames(Options::typeid);

    Console::WriteLine("there are {0} options in enum '{1}'",
        names->Length, Options::typeid);

    for (int i = 0 ; i < names->Length ; i++)
        Console::WriteLine("{0}: {1}", i, names[i]);

    Options o = Options::Option2;
    Console::WriteLine("value of 'o' is {0}", o);
}
```

```
there are 3 options in enum 'Options'
0: Option1
1: Option2
2: Option3
value of 'o' is Option2
```

範例: GetType 成員和屬性

`GetType` 物件支援的成員和屬性, 可用來檢查類型。此程式碼會擷取並顯示某幾項資訊:

```

// vcpp_reflection_4.cpp
// compile with: /clr
using namespace System;
int main() {
    Console::WriteLine("type information for 'String':");
    Type ^ t = String::typeid;

    String ^ assemblyName = t->Assembly->FullName;
    Console::WriteLine("assembly name: {0}", assemblyName);

    String ^ nameSpace = t->Namespace;
    Console::WriteLine("namespace: {0}", nameSpace);

    String ^ baseType = t->BaseType->FullName;
    Console::WriteLine("base type: {0}", baseType);

    bool isArray = t->IsArray;
    Console::WriteLine("is array: {0}", isArray);

    bool isClass = t->IsClass;
    Console::WriteLine("is class: {0}", isClass);
}

```

```

type information for 'String':
assembly name: mscorlib, Version=1.0.5000.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089
namespace: System
base type: System.Object
is array: False
is class: True

```

類型的範例：列舉

反映也可讓組件內的型別和類別成員的列舉型別。若要示範這項功能，定義一個簡單的類別：

```

// vcpp_reflection_5.cpp
// compile with: /clr /LD
using namespace System;
public ref class TestClass {
    int m_i;
public:
    TestClass() {}
    void SimpleTestMember1() {}
    String ^ SimpleMember2(String ^ s) { return s; }
    int TestMember(int i) { return i; }
    property int Member {
        int get() { return m_i; }
        void set(int i) { m_i = i; }
    }
};

```

組件的範例：檢查

如果上述程式碼編譯成 DLL，稱為 vcpp_reflection_6.dll 時，您接著可以使用反映來檢查此組件的內容。這牽涉到使用靜態的反映 API 函式 [xref:System.Reflection.Assembly.Load%2A?displayProperty=nameWithType](#) 來載入組件。此函式傳回的位址組件接著可以查詢有關模組和類型內的物件。

一旦反映系統成功載入的組件的陣列型別物件擷取 [Assembly.GetTypes](#) 函式。雖然只有一個類別定義在此情況下，每個陣列元素會包含不同類型的相關資訊。使用迴圈，每個型別此陣列中查詢使用的型別成員相

關Type::GetMembers函式。此函式傳回的陣列MethodInfo物件，每個物件，包含成員函式、資料成員或類型中的屬性的相關資訊。

中的附註的方法清單包括函式明確地定義TestClass及函式會隱含地繼承自system:: object類別。在.NET中，而不是視覺效果中所描述的一部分C++語法中，屬性會顯示為基礎的資料成員存取的get/set函式。Get/set函式出現在此清單做為一般方法。不是由Microsoft透過common language runtime支援反映C++編譯器。

雖然您可以使用此程式碼來檢查您所定義的組件，您也可以使用此程式碼來檢查.NET組件。例如，如果您變更TestAssembly mscorlib 時，您會看到每個型別和 mscorev.dll 中定義的方法清單。

```
// vcpp_reflection_6.cpp
// compile with: /clr
using namespace System;
using namespace System::IO;
using namespace System::Reflection;
int main() {
    Assembly ^ a = nullptr;
    try {
        // load assembly -- do not use file extension
        // will look for .dll extension first
        // then .exe with the filename
        a = Assembly::Load("vcpp_reflection_5");
    }
    catch (FileNotFoundException ^ e) {
        Console::WriteLine(e->Message);
        return -1;
    }

    Console::WriteLine("assembly info:");
    Console::WriteLine(a->FullName);
    array<Type^>^ typeArray = a->GetTypes();

    Console::WriteLine("type info ({0} types):", typeArray->Length);

    int totalTypes = 0;
    int totalMembers = 0;
    for (int i = 0 ; i < typeArray->Length ; i++) {
        // retrieve array of member descriptions
        array<MemberInfo^>^ member = typeArray[i]->GetMembers();

        Console::WriteLine(" members of {0} ({1} members):",
        typeArray[i]->FullName, member->Length);
        for (int j = 0 ; j < member->Length ; j++) {
            Console::Write("      ({0})", member[j]->MemberType.ToString());
            Console::Write("{0} ", member[j]);
            Console::WriteLine("");
            totalMembers++;
        }
        totalTypes++;
    }
    Console::WriteLine("{0} total types, {1} total members",
    totalTypes, totalMembers);
}
```

如何：實作使用反映外掛程式元件架構

下列程式碼範例示範如何使用反映來實作簡單的「外掛程式」架構。第一個清單應用程式，而第二個外掛程式。應用程式是填入本身使用任何以 form 為基礎的類別做為命令列引數所提供的外掛程式 DLL 中找到的多個文件表單。

應用程式嘗試載入使用提供的組件System.Reflection.Assembly.Load方法。如果成功，組件內的型別列舉使用System.Reflection.Assembly.GetTypes方法。每個型別接著檢查相容性使用System.Type.IsAssignableFrom方法。在此範例中，提供的組件中找到的類別必須衍生自Form近似於外掛程式類別。

相容的類別會接著使用具現化 `System.Activator.CreateInstance` 方法，它接受 `Type` 做為引數和傳回的新執行個體的指標。然後附加到表單，顯示每個新的執行個體。

請注意，`Load` 方法不接受包含副檔名的組件名稱。因此下列程式碼範例適用於在任一情況下，應用程式中的 `main` 函式會修剪任何提供的延伸模組。

範例

下列程式碼會定義接受外掛程式的應用程式。必須提供組件名稱做為第一個引數。這個組件應該包含至少一個公用 `Form` 衍生型別。

```
// plugin_application.cpp
// compile with: /clr /c
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Reflection;

ref class PluggableForm : public Form {
public:
    PluggableForm() {}
    PluggableForm(Assembly^ plugAssembly) {
        Text = "plug-in example";
        Size = Drawing::Size(400, 400);
        IsMdiContainer = true;

        array<Type^>^ types = plugAssembly->GetTypes();
        Type^ formType = Form::typeid;

        for (int i = 0 ; i < types->Length ; i++) {
            if (formType->IsAssignableFrom(types[i])) {
                // Create an instance given the type description.
                Form^ f = dynamic_cast<Form^>(Activator::CreateInstance(types[i]));
                if (f) {
                    f->Text = types[i]->ToString();
                    f->MdiParent = this;
                    f->>Show();
                }
            }
        }
    }
};

int main() {
    Assembly^ a = Assembly::LoadFrom("plugin_application.exe");
    Application::Run(gcnew PluggableForm(a));
}
```

範例

下列程式碼會定義三個類別衍生自 `Form`。時產生的組件名稱傳遞至先前的清單中的可執行檔，每個三個類別將可探索和具現化，儘管它們是裝載的應用程式，在編譯時期未知。

```

// plugin_assembly.cpp
// compile with: /clr /LD
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Reflection;
using namespace System::Drawing;

public ref class BlueForm : public Form {
public:
    BlueForm() {
        BackColor = Color::Blue;
    }
};

public ref class CircleForm : public Form {
protected:
    virtual void OnPaint(PaintEventArgs^ args) override {
        args->Graphics->FillEllipse(Brushes::Green, ClientRectangle);
    }
};

public ref class StarburstForm : public Form {
public:
    StarburstForm(){
        BackColor = Color::Black;
    }
protected:
    virtual void OnPaint(PaintEventArgs^ args) override {
        Pen^ p = gcnew Pen(Color::Red, 2);
        Random^ r = gcnew Random( );
        Int32 w = ClientSize.Width;
        Int32 h = ClientSize.Height;
        for (int i=0; i<100; i++) {
            float x1 = w / 2;
            float y1 = h / 2;
            float x2 = r->Next(w);
            float y2 = r->Next(h);
            args->Graphics->DrawLine(p, x1, y1, x2, y2);
        }
    }
};

```

如何:列舉使用反映的組件中的資料類型

下列程式碼示範的公用類型和成員使用列舉[System.Reflection](#)。

指定組件的名稱之後的本機目錄中或在 GAC 中，下列程式碼會嘗試開啟組件，並擷取描述。如果成功的話，每個類型會顯示其公用成員。

請注意，[System.Reflection.Assembly.Load](#)要求使用的不具副檔名。因此，使用「mscorlib.dll」作為命令列引數時將會失敗，使用只是「mscorlib」會產生.NET Framework 類型的顯示。如果不提供任何組件名稱，程式碼會偵測並報告目前的組件中的類型（此程式碼所產生的 EXE）。

範例

```

// self_reflection.cpp
// compile with: /clr
using namespace System;
using namespace System::Reflection;
using namespace System::Collections;

public ref class ExampleType {
public:
    ExampleType() {}
    void Func() {}
};

int main() {
    String^ delimStr = " ";
    array<Char>^ delimiter = delimStr->ToCharArray( );
    array<String^>^ args = Environment::CommandLine->Split( delimiter );

    // replace "self_reflection.exe" with an assembly from either the local
    // directory or the GAC
    Assembly^ a = Assembly::LoadFrom("self_reflection.exe");
    Console::WriteLine(a);

    int count = 0;
    array<Type^>^ types = a->GetTypes();
    IEnumrator^ typeIter = types->GetEnumerator();

    while ( typeIter->MoveNext() ) {
        Type^ t = dynamic_cast<Type^>(typeIter->Current);
        Console::WriteLine("    {0}", t->ToString());

        array<MemberInfo^>^ members = t->GetMembers();
        IEnumrator^ memberIter = members->GetEnumerator();
        while ( memberIter->MoveNext() ) {
            MemberInfo^ mi = dynamic_cast<MemberInfo^>(memberIter->Current);
            Console::Write("        {0}", mi->ToString( ) );
            if ( mi->MemberType == MemberTypes::Constructor )
                Console::Write("    (constructor)");

            Console::WriteLine();
        }
        count++;
    }
    Console::WriteLine("{0} types found", count);
}

```

另請參閱

- [以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

強式名稱組件 (組件簽署) (C++/CLI)

2019/12/2 • [Edit Online](#)

本主題討論如何登入您的組件，通常是指讓您的組件強式名稱。

備註

當使用視覺效果C++，使用連結器選項來登入您的組件，以避免發生與CLR屬性，以簽署組件相關的問題：

- [AssemblyDelaySignAttribute](#)
- [AssemblyKeyFileAttribute](#)
- [AssemblyKeyNameAttribute](#)

不使用屬性的原因包括索引鍵的名稱會顯示在組件中繼資料，可能會有安全性風險，如果檔案名稱中包含機密資訊的事實。此外，視覺效果使用建置流程C++開發環境將會失效，如果您使用CLR屬性來指定組件的強式的名稱，並接著執行組件上的後置處理 mt.exe 等工具，已簽署組件的索引鍵。

如果您在命令列建置，使用連結器選項來簽署組件，然後再執行後置處理工具 (mt.exe)，您必須重新簽署組件使用sn.exe。或者，您可以建置和延遲簽署組件並執行後置處理的工具之後，再完成簽署。

如果在開發環境中建置時，您會使用簽署的屬性，您可以成功登入的組件藉由明確呼叫 sn.exe ([Sn.exe \(強式名稱工具\)](#)) 在建置後事件。如需詳細資訊，請參閱[指定建置事件](#)。如果您使用屬性和建置後事件，相較於使用連結器選項，建置時間可能較少。

下列連結器選項支援簽署的組件：

- [/DELAYSIGN \(部分簽署組件\)](#)
- [/KEYFILE \(指定金鑰或金鑰組以簽署組件\)](#)
- [/KEYCONTAINER \(指定金鑰容器以簽署組件\)](#)

如需有關強式的組件的詳細資訊，請參閱 < [建立和使用強式名稱組件](#)。

另請參閱

[以 C++/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

Debug 類別 (C++/CLI)

2020/11/2 • [Edit Online](#)

Debug 在 Visual C++ 應用程式中使用時，其行為不會在 debug 和發行組建之間變更。

備註

的行為與 Trace Debug 類別的行為完全相同，但相依於所定義的符號追蹤。這表示您必須 `#ifdef` 有任何追蹤相關的程式碼，以防止發行組建中的 debug 行為。

範例：一律執行 output 語句

描述

下列範例一律會執行 output 語句，無論您是使用 /DDEBUG 或 /DTRACE 編譯。

程式碼

```
// mcpp_debug_class.cpp
// compile with: /clr
#using <system.dll>
using namespace System::Diagnostics;
using namespace System;

int main() {
    Trace::Listeners->Add( gcnew TextWriterTraceListener( Console::Out ) );
    Trace::AutoFlush = true;
    Trace::Indent();
    Trace::WriteLine( "Entering Main" );
    Console::WriteLine( "Hello World." );
    Trace::WriteLine( "Exiting Main" );
    Trace::Unindent();

    Debug::WriteLine("test");
}
```

輸出

```
Entering Main
Hello World.
Exiting Main
test
```

範例：使用 #ifdef 和 #endif 指示詞

描述

若要取得預期的行為（也就是，針對發行組建）不會列印任何「測試」輸出，您必須使用和指示詞 `#ifdef` `#endif`。先前的程式碼範例修改如下以示範此修正：

程式碼

```
// mcpp_debug_class2.cpp
// compile with: /clr
#using <system.dll>
using namespace System::Diagnostics;
using namespace System;

int main() {
    Trace::Listeners->Add( gcnew TextWriterTraceListener( Console::Out ) );
    Trace::AutoFlush = true;
    Trace::Indent();

#ifndef TRACE // checks for a debug build
    Trace::WriteLine( "Entering Main" );
    Console::WriteLine( "Hello World." );
    Trace::WriteLine( "Exiting Main" );
#endif
    Trace::Unindent();

#ifndef DEBUG // checks for a debug build
    Debug::WriteLine("test");
#endif //ends the conditional block
}
```

另請參閱

[使用 c + +/CLI 進行 .NET 程式設計 \(Visual C++\)](#)

STL/CLR 程式庫參考

2020/11/2 • [Edit Online](#)

STL/CLR 程式庫提供的介面類別似于 C++ 標準程式庫容器，可搭配 C++ 和 .NET Framework common language runtime (CLR) 使用。STL/CLR 和 C++ 標準程式庫的 Microsoft 執行完全不同。STL/CLR 會針對舊版支援進行維護，但不會以 C++ 標準保持最新狀態。我們強烈建議盡可能使用原生 [C++ 標準程式庫](#) 容器，而不是 STL/CLR。

若要使用 STL/CLR：

- 包含 `cliext` 包含子目錄中的標頭，而不是一般的 C++ 標準程式庫對應專案。
- 限定程式庫名稱，`cliext::` 而不是 `std::`。

STL/CLR 程式庫提供類似 STL 的介面，可搭配 C++ 和 .NET Framework common language runtime (CLR) 使用。此程式庫會針對舊版支援進行維護，但不會與 C++ 標準保持最新狀態。我們強烈建議使用原生 [C++ 標準程式庫](#) 容器，而不是 STL/CLR。

本節內容

[cliext 命名空間](#)

討論包含所有 STL/CLR 程式庫類型的命名空間。

[STL/CLR 容器](#)

提供在 C++ 標準程式庫中找到之容器的總覽，包括容器元素的需求、可插入的元素類型，以及擁有權問題。

[STL/CLR 容器專案的需求](#)

描述插入 C++ 標準程式庫容器的所有參考型別的最低需求。

[如何：從 .NET 集合轉換為 STL/CLR 容器](#)

描述如何將 .NET 集合轉換為 STL/CLR 容器。

[如何：從 STL/CLR 容器轉換為 .NET 集合](#)

描述如何將 STL/CLR 容器轉換為 .NET 集合。

[如何：從元件公開 STL/CLR 容器](#)

示範如何顯示以 C++ 元件撰寫的數個 STL/CLR 容器的元素。

此外，本節也描述 STL/CLR 的下列元件：

`adapter` (STL/CLR)
`algorithm` (STL/CLR)
`deque` (STL/CLR)
`for each` , `in`
`functional` (STL/CLR)
`hash_map` (STL/CLR)
`hash_multimap` (STL/CLR)
`hash_multiset` (STL/CLR)
`hash_set` (STL/CLR)
`list` (STL/CLR) \

`map` (STL/CLR)
`multimap` (STL/CLR)
`multiset` (STL/CLR)

[numeric](#) (STL/CLR)

[priority_queue](#) (STL/CLR)

[queue](#) (STL/CLR)

[set](#) (STL/CLR)

[stack](#) (STL/CLR)

[utility](#) (STL/CLR)

[vector](#) (STL/CLR) \

另請參閱

[C++ 標準程式庫](#)

cliext 命名空間

2019/12/2 • [Edit Online](#)

`cliext` 命名空間包含所有類型的 STL/CLR 程式庫。如需所有這些類型和 STL/CLR 類型的詳細資訊連結的清單，請參閱 <[STL/CLR 程式庫參考](#)。

另請參閱

[STL/CLR 程式庫參考](#)

STL/CLR 容器

2020/11/2 • [Edit Online](#)

STL/CLR 程式庫是由類似于 C++ 標準程式庫中的容器所組成，但它會在 .NET Framework 的 managed 環境中執行。它不會與實際的 C++ 標準程式庫保持在最新狀態，並且會針對舊版支援加以維護。

本文件提供了 STL/CLR 容器的概觀，例如容器項目的需求、您可以插入容器的項目類型，以及容器中項目的擁有權問題。在適當的情況下，會提及原生 C++ 標準程式庫和 STL/CLR 之間的差異。

容器項目的需求

插入 STL/CLR 容器中的所有元素都必須遵守特定的指導方針。如需詳細資訊，請參閱[STL/CLR 容器元素的需求](#)。

有效的容器項目

STL/CLR 容器可保留兩種類型之一的項目：

- 參考類型的控制代碼。
- 參考型別。
- Unboxed 實值類型。

您不能將 Boxed 實值類型插入至任何 STL/CLR 容器。

參考類型的控制代碼

您可以將參考類型的控制代碼插入至 STL/CLR 容器。C++ 中以 CLR 為目標的控制代碼與原生 C++ 的指標類似。如需詳細資訊，請參閱[Handle To Object Operator \(^\)](#)。

範例

下列範例顯示如何將 Employee 物件的控制碼插入至 `cliext::set`。

```
// cliext_container_valid_reference_handle.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

ref class Employee
{
public:
    // STL/CLR containers might require a public constructor, so it
    // is a good idea to define one.
    Employee() :
        name(nullptr),
        employeeNumber(0) { }

    // All STL/CLR containers require a public copy constructor.
    Employee(const Employee% orig) :
        name(orig.name),
        employeeNumber(orig.employeeNumber) { }

    // All STL/CLR containers require a public assignment operator.
    Employee% operator=(const Employee% orig)
    {
        if (this != %orig)
```

```

{
    name = orig.name;
    employeeNumber = orig.employeeNumber;
}

return *this;
}

// All STL/CLR containers require a public destructor.
~Employee() { }

// Associative containers such as maps and sets
// require a comparison operator to be defined
// to determine proper ordering.
bool operator<(const Employee^ rhs)
{
    return (employeeNumber < rhs->employeeNumber);
}

// The employee's name.
property String^ Name
{
    String^ get() { return name; }
    void set(String^ value) { name = value; }
}

// The employee's employee number.
property int EmployeeNumber
{
    int get() { return employeeNumber; }
    void set(int value) { employeeNumber = value; }
}

private:
    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee^ empl1419 = gcnew Employee();
    empl1419->Name = L"Darin Lockert";
    empl1419->EmployeeNumber = 1419;

    // Add the employee to the set of all employees.
    set<Employee^>^ emplSet = gcnew set<Employee^>();
    emplSet->insert(empl1419);

    // List all employees of the company.
    for each (Employee^ empl in emplSet)
    {
        Console::WriteLine("Employee Number {0}: {1}",
                           empl->EmployeeNumber, empl->Name);
    }

    return 0;
}

```

參考類型

您也可以將參考類型 (而不是參考類型的控制代碼) 插入至 STL/CLR 容器。此處的主要差異在於，當參考類型的容器遭到刪除時，會呼叫容器內部所有項目的解構函式。在參考類型之控制代碼的容器中，將不會呼叫這些項目的解構函式。

範例

下列範例說明如何將 Employee 物件插入至 `cliext::set` 內。

```

// cliext_container_valid_reference.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

ref class Employee
{
public:
    // STL/CLR containers might require a public constructor, so it
    // is a good idea to define one.
    Employee() :
        name(nullptr),
        employeeNumber(0) { }

    // All STL/CLR containers require a public copy constructor.
    Employee(const Employee% orig) :
        name(orig.name),
        employeeNumber(orig.employeeNumber) { }

    // All STL/CLR containers require a public assignment operator.
    Employee% operator=(const Employee% orig)
    {
        if (this != %orig)
        {
            name = orig.name;
            employeeNumber = orig.employeeNumber;
        }

        return *this;
    }

    // All STL/CLR containers require a public destructor.
    ~Employee() { }

    // Associative containers such as maps and sets
    // require a comparison operator to be defined
    // to determine proper ordering.
    bool operator<(const Employee^ rhs)
    {
        return (employeeNumber < rhs->employeeNumber);
    }

    // The employee's name.
    property String^ Name
    {
        String^ get() { return name; }
        void set(String^ value) { name = value; }
    }

    // The employee's employee number.
    property int EmployeeNumber
    {
        int get() { return employeeNumber; }
        void set(int value) { employeeNumber = value; }
    }

private:
    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee empl1419;
}

```

```
empl1419.Name = L"Darin Lockert";
empl1419.EmployeeNumber = 1419;

// Add the employee to the set of all employees.
set<Employee>^ emplSet = gcnew set<Employee>();
emplSet->insert(empl1419);

// List all employees of the company.
for each (Employee^ empl in emplSet)
{
    Console::WriteLine("Employee Number {0}: {1}",
        empl->EmployeeNumber, empl->Name);
}

return 0;
}
```

Unboxed 實值類型

您也可以將 Unboxed 實值類型插入至 STL/CLR 容器。未裝箱的實值型別是尚未封裝成引用型別的實值型別。

實數值型別專案可以是其中一個標準數值型別(例如 `int`)，也可以是使用者定義的實數值型別，例如

`value class`。如需詳細資訊，請參閱[類別和結構](#)

範例

下列範例會修改第一個範例，將 `Employee` 類別改為實值類型。接著再將這個實值類型插入至 `cliext::set`，如第一個範例中所示。

```

// cliext_container_valid_valuetype.cpp
// compile with: /clr

#include <cliext/set>

using namespace cliext;
using namespace System;

value class Employee
{
public:
    // Associative containers such as maps and sets
    // require a comparison operator to be defined
    // to determine proper ordering.
    bool operator<(const Employee^ rhs)
    {
        return (employeeNumber < rhs->employeeNumber);
    }

    // The employee's name.
    property String^ Name
    {
        String^ get() { return name; }
        void set(String^ value) { name = value; }
    }

    // The employee's employee number.
    property int EmployeeNumber
    {
        int get() { return employeeNumber; }
        void set(int value) { employeeNumber = value; }
    }

private:
    String^ name;
    int employeeNumber;
};

int main()
{
    // Create a new employee object.
    Employee empl1419;
    empl1419.Name = L"Darin Lockert";
    empl1419.EmployeeNumber = 1419;

    // Add the employee to the set of all employees.
    set<Employee>^ emplSet = gcnew set<Employee>();
    emplSet->insert(empl1419);

    // List all employees of the company.
    for each (Employee empl in emplSet)
    {
        Console::WriteLine("Employee Number {0}: {1}",
                           empl.EmployeeNumber, empl.Name);
    }

    return 0;
}

```

如果您嘗試將實數值型別的控制碼插入容器中，就會產生[編譯器錯誤 C3225](#)。

效能和記憶體含意

在決定是要使用參考類型的控制代碼或實值類型作為容器項目時，您必須考量許多因素。如果您決定使用實值類型，請記得在每次將項目插入至容器時建立這個項目的複本。若為小型物件，這應該不會造成問題，但是如果插入的物件很大，則可能會減損效能。此外，如果您使用實值類型，則無法同時在多個容器中儲存一個項目，因為每個容

器將會擁有其項目的複本。

如果您決定使用參考類型的控制代碼，則可能會提升效能，因為在將項目插入至容器時，不需要製作項目的複本。此外，不同於實值類型，其同一個項目可以存在於多個容器中。不過，如果您決定使用控制代碼，則必須確保控制代碼有效，並且它所參考的物件沒有在程式的其他地方遭到刪除。

容器的擁有權問題

STL/CLR 中的容器會處理實值語意。每當您將項目插入至容器時，也會插入該項目的複本。如果您要取得類似參考的語意，可以插入物件的控制代碼物件而不是物件本身。

當您呼叫控制代碼物件的容器之清除或清理方法時，不會從記憶體釋放控制代碼所參考的物件。您必須明確地刪除該物件，或者（因為這些物件位於 Managed 堆積中）在決定不再使用物件時允許記憶體回收行程釋放記憶體。

另請參閱

[C++ 標準程式庫參考](#)

STL/CLR 容器項目的需求

2019/12/2 • [Edit Online](#)

插入 STL/CLR 容器的所有參考類型至少必須擁有下列項目：

- 公用的複製建構函式。
- 公用的指派運算子。
- 公用的解構函式。

此外，這類的關聯容器[設定並地圖](#)必須具有公用的比較運算子定義，也就是 `operator<` 預設。容器的某些作業可能也需要公用預設建構函式和公用等價運算子定義。

和參考類型一樣，實值類型及要插入一個關聯容器之參考類型的控制代碼必須具有比較運算子，例如 `operator<` 定義。公用複製建構函式、公用指派運算子和公用解構函式的需求中未包含實值類型或參考類型的控制代碼。

另請參閱

[C++ 標準程式庫參考](#)

如何：從 .NET 集合轉換為 STL/CLR 容器

2020/11/2 • [Edit Online](#)

本主題說明如何將 .NET 集合轉換成其對等的 STL/CLR 容器。舉例來說，我們會示範如何將 .NET 轉換成 `List<T>` stl/clr 向量，以及如何將 .NET 轉換成 `Dictionary< TKey, TValue >` stl/clr map，但是所有集合和容器的程式都很類似。

從集合建立容器

1. 若要轉換整個集合，請建立 STL/CLR 容器，並將集合傳遞至該函式。

第一個範例會示範此程式。

-或-

1. 藉由建立 `collection_adapter` 物件，建立泛型 STL/CLR 容器。此範本類別會將 .NET 集合介面視為引數。若要確認支援的介面，請參閱 [collection_adapter \(STL/CLR\)](#)。
2. 將 .NET 集合的內容複寫到容器。這可以藉由使用 STL/CLR 演算法來完成，或逐一查看 .net 集合，並將每個元素的複本插入 STL/clr 容器中。

第二個範例會示範此程式。

範例

在此範例中，我們會建立泛型 `List<T>` 並將5個元素加入其中。然後，我們 `vector` 會使用接受 `IEnumerable<T>` 做為引數的函式來建立。

```
// cliext_convert_list_to_vector.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/algorithm>
#include <cliext/vector>

using namespace System;
using namespace System::Collections;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    List<int> ^primeNumbersColl = gcnew List<int>();
    primeNumbersColl->Add(2);
    primeNumbersColl->Add(3);
    primeNumbersColl->Add(5);
    primeNumbersColl->Add(7);
    primeNumbersColl->Add(11);

    cliext::vector<int> ^primeNumbersCont =
        gcnew cliext::vector<int>(primeNumbersColl);

    Console::WriteLine("The contents of the cliext::vector are:");
    cliext::vector<int>::const_iterator it;
    for (it = primeNumbersCont->begin(); it != primeNumbersCont->end(); it++)
    {
        Console::WriteLine(*it);
    }
}
```

```
The contents of the cliext::vector are:  
2  
3  
5  
7  
11
```

在此範例中，我們會建立泛型 `Dictionary< TKey, TValue >` 並將5個元素加入其中。接著，我們會建立，`collection_adapter` 以將 `Dictionary< TKey, TValue >` 當作簡單的 STL/CLR 容器來包裝。最後，我們會建立，並逐一查看，將的 `map` 內容複寫 `Dictionary< TKey, TValue >` 到 `map` `collection_adapter`。在這個過程中，我們會使用函式來建立新的配對 `make_pair`，並將新的配對直接插入至 `map`。

```
// cliext_convert_dictionary_to_map.cpp  
// compile with: /clr  
  
#include <cliext/adapter>  
#include <cliext/algorithm>  
#include <cliext/map>  
  
using namespace System;  
using namespace System::Collections;  
using namespace System::Collections::Generic;  
  
int main(array<System::String ^> ^args)  
{  
    System::Collections::Generic::Dictionary<float, int> ^dict =  
        gcnew System::Collections::Generic::Dictionary<float, int>();  
    dict->Add(42.0, 42);  
    dict->Add(13.0, 13);  
    dict->Add(74.0, 74);  
    dict->Add(22.0, 22);  
    dict->Add(0.0, 0);  
  
    cliext::collection_adapter<System::Collections::Generic::IDictionary<float, int>> dictAdapter(dict);  
    cliext::map<float, int> aMap;  
    for each (KeyValuePair<float, int> ^kvp in dictAdapter)  
    {  
        cliext::pair<float, int> aPair = cliext::make_pair(kvp->Key, kvp->Value);  
        aMap.insert(aPair);  
    }  
  
    Console::WriteLine("The contents of the cliext::map are:");  
    cliext::map<float, int>::const_iterator it;  
    for (it = aMap.begin(); it != aMap.end(); it++)  
    {  
        Console::WriteLine("Key: {0:F} Value: {1}", it->first, it->second);  
    }  
}
```

```
The contents of the cliext::map are:  
Key: 0.00 Value: 0  
Key: 13.00 Value: 13  
Key: 22.00 Value: 22  
Key: 42.00 Value: 42  
Key: 74.00 Value: 74
```

另請參閱

[STL/CLR 程式庫參考](#)
[adapter \(STL/CLR\)](#)

如何：從 STL/CLR 容器轉換為 .NET 集合

2020/11/2 • [Edit Online](#)

本主題說明如何將 STL/CLR 容器轉換成其對等的 .NET 集合。舉例而言，我們會示範如何將 STL/CLR 向量 轉換成 .net, `ICollection<T>` 以及如何將 stl/clr 對應 轉換成 .net `IDictionary< TKey, TValue >`，但是所有集合和容器的程式都很類似。

從容器建立集合

1. 請使用下列其中一個方法：

- 若要轉換容器的部分，請呼叫 `make_collection` 函式，並傳遞 STL/CLR 容器的 begin iterator 和 end 反覆運算器，以複製到 .net 集合中。此範本函式會採用 STL/CLR iterator 作為樣板引數。第一個範例會示範這個方法。
- 若要轉換整個容器，請將容器轉換成適當的 .NET 集合介面或介面集合。第二個範例會示範這個方法。

範例

在此範例中，我們會建立 STL/CLR `vector`，並在其中加入5個元素。然後，我們會藉由呼叫函數來建立 .NET 集合 `make_collection`。最後，我們會顯示新建立之集合的內容。

```
// cliext_convert_vector_to_icollection.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/vector>

using namespace cliext;
using namespace System;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    cliext::vector<int> primeNumbersCont;
    primeNumbersCont.push_back(2);
    primeNumbersCont.push_back(3);
    primeNumbersCont.push_back(5);
    primeNumbersCont.push_back(7);
    primeNumbersCont.push_back(11);

    System::Collections::Generic::ICollection<int> ^iColl =
        make_collection<cliext::vector<int>::iterator>(
            primeNumbersCont.begin() + 1,
            primeNumbersCont.end() - 1);

    Console::WriteLine("The contents of the System::Collections::Generic::ICollection are:");
    for each (int i in iColl)
    {
        Console::WriteLine(i);
    }
}
```

```
The contents of the System::Collections::Generic::ICollection are:
```

```
3
```

```
5
```

```
7
```

在此範例中，我們會建立 STL/CLR `map`，並在其中加入5個元素。接著，我們會建立 .NET `IDictionary< TKey, TValue >` 並直接指派給 `map` 它。最後，我們會顯示新建立之集合的內容。

```
// cliext_convert_map_to_idictionary.cpp
// compile with: /clr

#include <cliext/adapter>
#include <cliext/map>

using namespace cliext;
using namespace System;
using namespace System::Collections::Generic;

int main(array<System::String ^> ^args)
{
    cliext::map<float, int> ^aMap = gcnew cliext::map<float, int>;
    aMap->insert(cliext::make_pair<float, int>(42.0, 42));
    aMap->insert(cliext::make_pair<float, int>(13.0, 13));
    aMap->insert(cliext::make_pair<float, int>(74.0, 74));
    aMap->insert(cliext::make_pair<float, int>(22.0, 22));
    aMap->insert(cliext::make_pair<float, int>(0.0, 0));

    System::Collections::Generic::IDictionary<float, int> ^iDict = aMap;

    Console::WriteLine("The contents of the IDictionary are:");
    for each (KeyValuePair<float, int> ^kvp in iDict)
    {
        Console::WriteLine("Key: {0:F} Value: {1}", kvp->Key, kvp->Value);
    }
}
```

```
The contents of the IDictionary are:
```

```
Key: 0.00 Value: 0
```

```
Key: 13.00 Value: 13
```

```
Key: 22.00 Value: 22
```

```
Key: 42.00 Value: 42
```

```
Key: 74.00 Value: 74
```

另請參閱

[STL/CLR 程式庫參考](#)

[如何：從 .NET 集合轉換為 STL/CLR 容器](#)

[range_adapter \(STL/CLR\)](#)

HOW TO : 公開 STL/CLR 容器從組件

2019/12/2 • [Edit Online](#)

STL/CLR 容器，例如 `list` 和 `map` 會實作為樣板 ref 類別。因為 C++ 範本會在編譯時期具現化，有完全相同的簽章，但位於不同的組件的兩個範本類別其實是不同類型。這表示範本類別，不可跨組件界限使用。

若要進行跨組件共用，STL/CLR 容器會實作泛型介面 `ICollection<T>`。利用這個泛型介面，所有語言的都支援泛型，包括 C++、C#、和 Visual Basic 中，可以存取 STL/CLR 容器。

本主題說明如何顯示數個以撰寫的 STL/CLR 容器項目 C++ 名為組件 `StlClrClassLibrary`。我們顯示兩個組件存取 `StlClrClassLibrary`。第一個組件以 C++，並在第二個 C#。

如果兩個組件以 C++，您可以使用來存取容器的泛型介面及其 `generic_container` `typedef`。例如，如果您有容器的型別 `cliext::vector<int>`，則其泛型介面是：`cliext::vector<int>::generic_container`。同樣地，使用泛型介面取得迭代器 `generic_iterator` `typedef`，如下所示：`cliext::vector<int>::generic_iterator`。

因為這些 `typedef` 告訴中 C++ 標頭檔，以其他語言撰寫的組件無法使用它們。因此，若要存取的泛型介面 `cliext::vector<int>` 在 C# 或任何其他.NET 語言中，使用 `System.Collections.Generic.ICollection<int>`。若要逐一查看這個集合，使用 `foreach` 迴圈。

下表列出每一個 STL/CLR 容器會實作泛型介面：

STL/CLR	CLR
<code>deque<T></code>	<code>ICollection<T></code>
<code>hash_map<K, V></code>	<code>IDictionary<K, V></code>
<code>hash_multimap<K, V></code>	<code>IDictionary<K, V></code>
<code>hash_multiset<T></code>	<code>ICollection<T></code>
<code>hash_set<T></code>	<code>ICollection<T></code>
<code>list<T></code>	<code>ICollection<T></code>
<code>map<K, V></code>	<code>IDictionary<K, V></code>
<code>multimap<K, V></code>	<code>IDictionary<K, V></code>
<code>multiset<T></code>	<code>ICollection<T></code>
<code>set<T></code>	<code>ICollection<T></code>
<code>vector<T></code>	<code>ICollection<T></code>

NOTE

因為 `queue`、`priority_queue`，和 `stack` 容器不支援迭代器，它們不會實作泛型介面，並不能存取的跨組件。

範例 1

描述

在此範例中，我們宣告C++類別，其中包含私用的STL/CLR成員資料。然後，我們會宣告授與存取權的私用集合類別的公用方法。我們是採用兩種不同的方式，一個用於C++用戶端，另一個用於其他.NET用戶端。

程式碼

```
// StlClrClassLibrary.h
#pragma once

#include <cliext/deque>
#include <cliext/list>
#include <cliext/map>
#include <cliext/set>
#include <cliext/stack>
#include <cliext/vector>

using namespace System;
using namespace System::Collections::Generic;
using namespace cliext;

namespace StlClrClassLibrary {

    public ref class StlClrClass
    {
    public:
        StlClrClass();

        // These methods can be called by a C++ class
        // in another assembly to get access to the
        // private STL/CLR types defined below.
        deque<wchar_t>::generic_container ^GetDequeCpp();
        list<float>::generic_container ^GetListCpp();
        map<int, String ^>::generic_container ^GetMapCpp();
        set<double>::generic_container ^GetSetCpp();
        vector<int>::generic_container ^GetVectorCpp();

        // These methods can be called by a non-C++ class
        // in another assembly to get access to the
        // private STL/CLR types defined below.
        ICollection<wchar_t> ^GetDequeCs();
        ICollection<float> ^GetListCs();
        IDictionary<int, String ^> ^GetMapCs();
        ICollection<double> ^GetSetCs();
        ICollection<int> ^GetVectorCs();

    private:
        deque<wchar_t> ^aDeque;
        list<float> ^aList;
        map<int, String ^> ^aMap;
        set<double> ^aSet;
        vector<int> ^aVector;
    };
}
```

範例 2

描述

在此範例中，我們會實作範例1中所宣告的類別。為了讓用戶端使用此類別庫，我們會使用資訊清單工具mt.exe在DLL中內嵌資訊清單檔案。如需詳細資訊，請參閱程式碼註解。

如需有關的資訊清單工具和並排顯示組件的詳細資訊，請參閱 <[建置C/C++隔離的應用程式和並排顯示組件](#)>。

程式碼

```
// StlClrClassLibrary.cpp
// compile with: /clr /LD /link /manifest
// post-build command: (attrib -r StlClrClassLibrary.dll & mt /manifest StlClrClassLibrary.dll.manifest
// /outputresource:StlClrClassLibrary.dll;#2 & attrib +r StlClrClassLibrary.dll)

#include "StlClrClassLibrary.h"

namespace StlClrClassLibrary
{
    StlClrClass::StlClrClass()
    {
        aDeque = gcnew deque<wchar_t>();
        aDeque->push_back(L'a');
        aDeque->push_back(L'b');

        aList = gcnew list<float>();
        aList->push_back(3.14159f);
        aList->push_back(2.71828f);

        aMap = gcnew map<int, String ^>();
        aMap[0] = "Hello";
        aMap[1] = "World";

        aSet = gcnew set<double>();
        aSet->insert(3.14159);
        aSet->insert(2.71828);

        aVector = gcnew vector<int>();
        aVector->push_back(10);
        aVector->push_back(20);
    }

    deque<wchar_t>::generic_container ^StlClrClass::GetDequeCpp()
    {
        return aDeque;
    }

    list<float>::generic_container ^StlClrClass::GetListCpp()
    {
        return aList;
    }

    map<int, String ^>::generic_container ^StlClrClass::GetMapCpp()
    {
        return aMap;
    }

    set<double>::generic_container ^StlClrClass::GetSetCpp()
    {
        return aSet;
    }

    vector<int>::generic_container ^StlClrClass::GetVectorCpp()
    {
        return aVector;
    }

    ICollection<wchar_t> ^StlClrClass::GetDequeCs()
    {
        return aDeque;
    }

    ICollection<float> ^StlClrClass::GetListCs()
    {
        return aList;
    }
}
```

```
IDictionary<int, String ^> ^StlClrClass::GetMapCs()
{
    return aMap;
}

ICollection<double> ^StlClrClass::GetSetCs()
{
    return aSet;
}

ICollection<int> ^StlClrClass::GetVectorCs()
{
    return aVector;
}
```

範例 3

描述

在此範例中，我們會建立C++會使用範例 1 和 2 中建立的類別程式庫的用戶端。此用戶端會使用 `generic_container` 逐一查看的容器，並顯示其內容的 STL/CLR 容器的 `typedef`。

程式碼

```

// CppConsoleApp.cpp
// compile with: /clr /FUStlClrClassLibrary.dll

#include <cliext/deque>
#include <cliext/list>
#include <cliext/map>
#include <cliext/set>
#include <cliext/vector>

using namespace System;
using namespace StlClrClassLibrary;
using namespace cliext;

int main(array<System::String ^> ^args)
{
    StlClrClass theClass;

    Console::WriteLine("cliext::deque contents:");
    deque<wchar_t>::generic_container ^aDeque = theClass.GetDequeCpp();
    for each (wchar_t wc in aDeque)
    {
        Console::WriteLine(wc);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::list contents:");
    list<float>::generic_container ^aList = theClass.GetListCpp();
    for each (float f in aList)
    {
        Console::WriteLine(f);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::map contents:");
    map<int, String ^>::generic_container ^aMap = theClass.GetMapCpp();
    for each (map<int, String ^>::value_type rp in aMap)
    {
        Console::WriteLine("{0} {1}", rp->first, rp->second);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::set contents:");
    set<double>::generic_container ^aSet = theClass.GetSetCpp();
    for each (double d in aSet)
    {
        Console::WriteLine(d);
    }
    Console::WriteLine();

    Console::WriteLine("cliext::vector contents:");
    vector<int>::generic_container ^aVector = theClass.GetVectorCpp();
    for each (int i in aVector)
    {
        Console::WriteLine(i);
    }
    Console::WriteLine();

    return 0;
}

```

Output

```
cliext::deque contents:  
a  
b  
  
cliext::list contents:  
3.14159  
2.71828  
  
cliext::map contents:  
0 Hello  
1 World  
  
cliext::set contents:  
2.71828  
3.14159  
  
cliext::vector contents:  
10  
20
```

範例 4

描述

在此範例中，我們會建立使用範例 1 和 2 中建立的類別程式庫的 C# 用戶端。此用戶端會使用`ICollection<T>`STL/CLR 容器來逐一查看的容器，並顯示其內容的方法。

程式碼

```

// CsConsoleApp.cs
// compile with: /r:Microsoft.VisualBasic.dll /r:StlClrClassLibrary.dll /r:System.dll

using System;
using System.Collections.Generic;
using StlClrClassLibrary;
using cliext;

namespace CsConsoleApp
{
    class Program
    {
        static int Main(string[] args)
        {
            StlClrClass theClass = new StlClrClass();

            Console.WriteLine("cliext::deque contents:");
            ICollection<char> iCollChar = theClass.GetDequeCs();
            foreach (char c in iCollChar)
            {
                Console.WriteLine(c);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::list contents:");
            ICollection<float> iCollFloat = theClass.GetListCs();
            foreach (float f in iCollFloat)
            {
                Console.WriteLine(f);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::map contents:");
            IDictionary<int, string> iDict = theClass.GetMapCs();
            foreach (KeyValuePair<int, string> kvp in iDict)
            {
                Console.WriteLine("{0} {1}", kvp.Key, kvp.Value);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::set contents:");
            ICollection<double> iCollDouble = theClass.GetSetCs();
            foreach (double d in iCollDouble)
            {
                Console.WriteLine(d);
            }
            Console.WriteLine();

            Console.WriteLine("cliext::vector contents:");
            ICollection<int> iCollInt = theClass.GetVectorCs();
            foreach (int i in iCollInt)
            {
                Console.WriteLine(i);
            }
            Console.WriteLine();

            return 0;
        }
    }
}

```

Output

```
cliext::deque contents:  
a  
b  
  
cliext::list contents:  
3.14159  
2.71828  
  
cliext::map contents:  
0 Hello  
1 World  
  
cliext::set contents:  
2.71828  
3.14159  
  
cliext::vector contents:  
10  
20
```

另請參閱

[STL/CLR 程式庫參考](#)

for each, in

2020/11/2 • [Edit Online](#)

逐一查看陣列或集合。此非標準關鍵字在 C++/CLI 和原生 C++ 專案中皆可用。但是，不建議使用它。請考慮改用以標準 [範圍為基礎的 For 語句 \(c++\)](#)。

所有執行階段

語法

```
針對運算式***中****的每個 (* 類型**識別碼) {
    語句
}
```

參數

type

`identifier` 的類型。

識別碼

表示集合項目的反覆項目變數。當 `identifier` 是 [追蹤參考運算子](#) 時，您可以修改元素。

expression

陣列運算式或集合。集合項目必須如此，編譯器才能將其轉換為 `identifier` 類型。

語句

要執行的一個或多個陳述式。

備註

`for each` 陳述式可用來逐一查看集合。您可以修改集合中的元素，但無法新增或刪除專案。

系統會針對陣列或集合中的每個元素執行這些 語句。在完成集合中所有項目的反覆項目之後，程式控制權會轉移到 `for each` 區塊之後的下一個陳述式。

`for each` 和 `in` 都是 [內容相關的關鍵字](#)。

Windows 執行階段

規格需求

編譯器選項: /ZW

範例

本範例示範如何使用 `for each` 逐一查看字串。

```

// for_each_string1.cpp
// compile with: /ZW
#include <stdio.h>
using namespace Platform;

ref struct MyClass {
    property String^ MyStringProperty;
};

int main() {
    String^ MyString = ref new String("abcd");

    for each ( char c in MyString )
        wprintf("%c", c);

    wprintf("/n");

    MyClass^ x = ref new MyClass();
    x->MyStringProperty = "Testing";

    for each( char c in x->MyStringProperty )
        wprintf("%c", c);
}

```

abcd

Testing

Common Language Runtime

備註

CLR 語法與 所有運行 時間語法相同，不同之處如下。

expression

Managed 陣列運算式或集合。Collection 元素必須如此，編譯器才能將它從轉換成 [Object](#) 識別碼型別。

運算式會評估為型別，這個型別會定義方法，這個型別會傳回型別，而該型別會傳回實 [IEnumerable](#) [IEnumerable<T>](#) 或宣告 [GetEnumerator](#) [IEnumerator](#) 中定義的所有方法 [IEnumerator](#) 。

規格需求

編譯器選項: [/clr](#)

範例

本範例示範如何使用 [for each](#) 逐一查看字串。

```
// for_each_string2.cpp
// compile with: /clr
using namespace System;

ref struct MyClass {
    property String ^ MyStringProperty;
};

int main() {
    String ^ MyString = gcnew String("abcd");

    for each ( Char c in MyString )
        Console::Write(c);

    Console::WriteLine();

    MyClass ^ x = gcnew MyClass();
    x->MyStringProperty = "Testing";

    for each( Char c in x->MyStringProperty )
        Console::Write(c);
}
```

```
abcd
```

```
Testing
```

另請參閱

[執行時間平臺的元件擴充功能](#)
[以範圍為基礎的 for 語句 \(C++\)](#)

adapter (STL/CLR)

2020/11/2 • [Edit Online](#)

STL/CLR 標頭會 `<cliext/adapter>` 指定兩個範本類別 (`collection_adapter` 和 `range_adapter`)，以及範本 `make_collection` 函式。

Syntax

```
#include <cliext/adapter>
```

需求

標頭: `<cliext/adapter>`

命名空間: `cliext`

宣告

II	II
<code>collection_adapter (STL/CLR)</code>	將基底類別庫 (BCL) 集合包裝為一個範圍。
<code>range_adapter (STL/CLR)</code>	將範圍包裝為 BCL 集合。
II	II
<code>make_collection (STL/CLR)</code>	使用反覆運算器配對建立範圍介面卡。

成員

`collection_adapter (STL/CLR)`

包裝 .NET 集合，以做為 STL/CLR 容器使用。`collection_adapter` 是描述簡單 STL/CLR 容器物件的範本類別。它會包裝基類程式庫 (BCL) 介面，並傳回您用來操作受控制序列的反覆運算器組。

語法

```

template<typename Coll>
ref class collection_adapter;

template<>
ref class collection_adapter<
    System::Collections::ICollection>;
template<>
ref class collection_adapter<
    System::Collections::IEnumerable>;
template<>
ref class collection_adapter<
    System::Collections::IList>;
template<>
ref class collection_adapter<
    System::Collections::IDictionary>;
template<typename Value>
ref class collection_adapter<
    System::Collections::Generic::ICollection<Value>>;
template<typename Value>
ref class collection_adapter<
    System::Collections::Generic::IEnumerable<Value>>;
template<typename Value>
ref class collection_adapter<
    System::Collections::Generic::IList<Value>>;
template<typename Key,
typename Value>
ref class collection_adapter<
    System::Collections::Generic::IDictionary<Key, Value>>;

```

參數

Coll

已包裝集合的型別。

特製化

III	II
IEnumerable	透過元素的順序。
ICollection	維護元素群組。
IList	維護已排序的元素群組。
IDictionary	維護一組 {key, value} 組。
IEnumerable<Value>	透過具型別元素的順序。
ICollection<Value>	維護一組具類型的元素。
IList<Value>	維護具類型專案的已排序群組。
IDictionary<Value>	維護一組具類型的 {key, value} 配對。

成員

III	II
collection_adapter::difference_type (STL/CLR)	兩個項目之間帶正負號距離的類型。

<p> </p> <p>collection_adapter::iterator (STL/CLR)</p> <p>collection_adapter::key_type (STL/CLR)</p> <p>collection_adapter::mapped_type (STL/CLR)</p> <p>collection_adapter::reference (STL/CLR)</p> <p>collection_adapter::size_type (STL/CLR)</p> <p>collection_adapter::value_type (STL/CLR)</p>	<p> </p> <p>受控制序列之迭代器的類型。</p> <p>字典索引鍵的類型。</p> <p>字典值的型別。</p> <p>項目的參考類型。</p> <p>兩個項目之間帶正負號距離的類型。</p> <p>項目的類型。</p>
<p> </p> <p>collection_adapter::base (STL/CLR)</p> <p>collection_adapter::begin (STL/CLR)</p> <p>collection_adapter::collection_adapter (STL/CLR)</p> <p>collection_adapter::end (STL/CLR)</p> <p>collection_adapter::size (STL/CLR)</p> <p>collection_adapter::swap (STL/CLR)</p>	<p> </p> <p>指定包裝的 BCL 介面。</p> <p>指定受控制序列的開頭。</p> <p>構造介面卡物件。</p> <p>指定受控制序列的結尾。</p> <p>計算元素的數目。</p> <p>交換兩個容器的內容。</p>
<p> </p> <p>collection_adapter::operator= (STL/CLR)</p>	<p> </p> <p>取代儲存的 BCL 控制碼。</p>

備註

您可以使用此範本類別，將 BCL 容器操作為 STL/CLR 容器。會將 `collection_adapter` 控制碼儲存至 BCL 介面，進而控制一連串的元素。物件會傳回 `collection_adapter` `X` 一組輸入反覆運算器，`X.begin()` 而 `X.end()` 您可以使用這些反覆運算器來依序流覽元素。某些特製化也可讓您撰寫 `X.size()`，以決定受控制序列的長度。

collection_adapter::base (STL/CLR)

指定包裝的 BCL 介面。

語法

```
Coll^ base();
```

備註

成員函式會傳回儲存的 BCL 介面控制碼。

範例

```
// cliext_collection_adapter_base.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');
    Mycoll c1(%d6x);

    // display initial contents "x x x x x x "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("base() same = {0}", c1.base() == %c1);
    return (0);
}
```

```
x x x x x x
base() same = True
```

collection_adapter:: begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回輸入反覆運算器，此反覆運算器會指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。

範例

```

// cliext_collection_adapter_begin.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mycoll::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b

```

collection_adapter:: collection_adapter (STL/CLR)

構造介面卡物件。

語法

```

collection_adapter();
collection_adapter(collection_adapter<Coll>% right);
collection_adapter(collection_adapter<Coll>^ right);
collection_adapter(Coll^ collection);

```

參數

收集

要包裝的 BCL 控制碼。

對

要複製的物件。

備註

函數：

```
collection_adapter();
```

使用初始化預存控制碼 `nullptr`。

函數：

```
collection_adapter(collection_adapter<Coll>% right);
```

使用 `right.` `collection_adapter:: BASE (STL/CLR)`，初始化預存控制碼 `()`。

函數：

```
collection_adapter(collection_adapter<Coll>^ right);
```

使用 `right->` `collection_adapter:: BASE (STL/CLR)`，初始化預存控制碼 `()`。

函數：

```
collection_adapter(Coll^ collection);
```

使用初始化預存控制碼 `collection`。

範例

```
// cliext_collection_adapter_construct.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');

    // construct an empty container
    Mycoll c1;
    System::Console::WriteLine("base() null = {0}", c1.base() == nullptr);

    // construct with a handle
    Mycoll c2(%d6x);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container
    Mycoll c3(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying a container handle
    Mycoll c4(%c3);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
base() null = True
x x x x x x
x x x x x x
x x x x x x
```

collection_adapter::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型會描述已簽署的元素計數。

範例

```
// cliext_collection_adapter_difference_type.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mycoll::difference_type diff = 0;
    Mycoll::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

collection_adapter:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回指向受控制序列結尾以外的輸入反覆運算器。

範例

```

// cliext_collection_adapter_end.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}

```

a b c

collection_adapter:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 **T1**，可作為受控制序列的輸入反覆運算器。

範例

```

// cliext_collection_adapter_iterator.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

collection_adapter:: key_type (STL/CLR)

字典索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數的同義字 `Key` (在或的特製化中), `IDictionary` `IDictionary<Value>` 否則不會定義。

範例

```
// cliext_collection_adapter_key_type.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef cliext::collection_adapter<
    System::Collections::Generic::IDictionary<wchar_t, int>> Mycoll;
typedef System::Collections::Generic::KeyValuePair<wchar_t,int> Mypair;
int main()
{
    Mymap d1;
    d1.insert(Mymap::make_value(L'a', 1));
    d1.insert(Mymap::make_value(L'b', 2));
    d1.insert(Mymap::make_value(L'c', 3));
    Mycoll c1(%d1);

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mypair elem in c1)
    {
        Mycoll::key_type key = elem.Key;
        Mycoll::mapped_type value = elem.Value;
        System::Console::Write("[{0} {1}] ", key, value);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

collection_adapter:: mapped_type (STL/CLR)

字典值的型別。

語法

```
typedef Value mapped_type;
```

備註

此類型是樣板參數的同義字 `Value` (在或的特製化中), `IDictionary` `IDictionary<Value>` 否則不會定義。

範例

```
// cliext_collection_adapter_mapped_type.cpp
// compile with: /clr
#include <cliext/adapters>
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef cliext::collection_adapter<
    System::Collections::Generic::IDictionary<wchar_t, int>> Mycoll;
typedef System::Collections::Generic::KeyValuePair<wchar_t,int> Mypair;
int main()
{
    Mymap d1;
    d1.insert(Mymap::make_value(L'a', 1));
    d1.insert(Mymap::make_value(L'b', 2));
    d1.insert(Mymap::make_value(L'c', 3));
    Mycoll c1(%d1);

    // display contents "[a 1] [b 2] [c 3]"
    for each (Mypair elem in c1)
    {
        Mycoll::key_type key = elem.Key;
        Mycoll::mapped_type value = elem.Value;
        System::Console::Write("[{0} {1}] ", key, value);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

collection_adapter:: operator = (STL/CLR)

取代儲存的 BCL 控制碼。

語法

```
collection_adapter<Coll>% operator=(collection_adapter<Coll>% right);
```

參數

對

要複製的介面卡。

備註

成員運算子會將 **右移至物件**，然後傳回 `*this`。您可以使用它來取代儲存的 BCL 控制碼，以及 **右邊儲存的 bcl 控制碼複本**。

範例

```
// cliext_collection_adapter_operator_as.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mycoll c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

collection_adapter:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```

// cliext_collection_adapter_reference.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents "a b c "
    Mycoll::iterator it = c1.begin();
    for ( ; it != c1.end(); ++it)
    {   // get a reference to an element
        Mycoll::reference ref = *it;
        System::Console::Write("{0} ", ref);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

collection_adapter:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。它並未定義于或的特製化 `IEnumerable` 中 `IEnumerable<Value>` 。

範例

```

// cliext_collection_adapter_size.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d6x(6, L'x');
    Mycoll c1(%d6x);

    // display initial contents "x x x x x x "
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```
x x x x x x  
size() = 6
```

collection_adapter:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```
// cliext_collection_adapter_size_type.cpp  
// compile with: /clr  
#include <cliext/adapter>  
#include <cliext/deque>  
  
typedef cliext::collection_adapter<  
    System::Collections::ICollection> Mycoll;  
int main()  
{  
    cliext::deque<wchar_t> d6x(6, L'x');  
    Mycoll c1(%d6x);  
  
    // display initial contents "x x x x x x"  
    for each (wchar_t elem in c1)  
        System::Console::Write("{0} ", elem);  
    System::Console::WriteLine();  
  
    Mycoll::size_type size = c1.size();  
    System::Console::WriteLine("size() = {0}", size);  
    return (0);  
}
```

```
x x x x x x  
size() = 6
```

collection_adapter:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(collection_adapter<Coll>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊儲存的 BCL 控制碼 `*this`。 `right`

範例

```
// cliext_collection_adapter_swap.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::deque<wchar_t> d2(5, L'x');
    Mycoll c2(%d2);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x x x x x
x x x x x
a b c
```

collection_adapter:: value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與樣板參數 **值** 同義 (如果存在於特製化中);否則就是的同義字 `System::Object^`。

範例

```

// cliext_collection_adapter_value_type.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display contents "a b c" using value_type
    for (Mycoll::iterator it = c1.begin();
        it != c1.end(); ++it)
    {   // store element in value_type object
        Mycoll::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

make_collection (STL/CLR)

`range_adapter` 從反覆運算器配對建立。

語法

```

template<typename Iter>
range_adapter<Iter> make_collection(Iter first, Iter last);

```

參數

Iter

已包裝反覆運算器的類型。

first

要包裝的第一個反覆運算器。

last

要包裝的第二個反覆運算器。

備註

此範本函式會傳回 `gcnew range_adapter<Iter>(first, last)`。您可以使用它來 `range_adapter<Iter>` 從一對反覆運算器建立物件。

範例

```

// cliext_make_collection.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in d1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Collections::ICollection^ p1 =
        cliext::make_collection(d1.begin(), d1.end());
    System::Console::WriteLine("Count = {0}", p1->Count);
    System::Console::WriteLine("IsSynchronized = {0}",
        p1->IsSynchronized);
    System::Console::WriteLine("SyncRoot not nullptr = {0}",
        p1->SyncRoot != nullptr);

    // copy the sequence
    cli::array<System::Object^>^ a1 = gcnew cli::array<System::Object^>(5);

    a1[0] = L'|';
    p1->CopyTo(a1, 1);
    a1[4] = L'|';
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

a b c
Count = 3
IsSynchronized = False
SyncRoot not nullptr = True
| a b c |

```

range_adapter (STL/CLR)

此樣板類別會包裝一組反覆運算器，用來將數個基底類別庫 (BCL) 介面。您可以使用 range_adapter 來操控 STL/CLR 範圍，就如同它是 BCL 集合一樣。

語法

```

template<typename Iter>
ref class range_adapter
{
    public
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<Value>,
    System::Collections::Generic::ICollection<Value>
    { .... };
}

```

參數

Iter

與包裝的反覆運算器相關聯的類型。

成員

■■■	■■
range_adapter::range_adapter (STL/CLR)	構造介面卡物件。
range_adapter::operator= (STL/CLR)	取代預存反覆運算器的配對。

介面

■■	■■
IEnumerable	逐一查看集合中的元素。
ICollection	維護元素群組。
IEnumerable<T>	逐一查看集合中的類型元素。
ICollection<T>	維護一組具類型的元素。

備註

Range_adapter 會儲存一組反覆運算器，進而分隔一系列的元素。物件會實作為四個 BCL 介面，可讓您依序逐一查看元素。您可以使用這個範本類別來操作 STL/CLR 範圍，就像 BCL 容器一樣。

range_adapter:: operator = (STL/CLR)

取代預存反覆運算器的配對。

語法

```
range_adapter<Iter>% operator=(range_adapter<Iter>% right);
```

參數

對

要複製的介面卡。

備註

成員運算子會將 右 移至物件，然後傳回 `*this`。您可以使用它，將預存 iterator 組取代為 右邊的預存反覆運算器 組複本。

範例

```

// cliext_range_adapter_operator_as.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Myrange c1(d1.begin(), d1.end());

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myrange c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

range_adapter::range_adapter (STL/CLR)

構造介面卡物件。

語法

```

range_adapter();
range_adapter(range_adapter<Iter>% right);
range_adapter(range_adapter<Iter>^ right);
range_adapter(Iter first, Iter last);

```

參數

first

要包裝的第一個反覆運算器。

last

要包裝的第二個反覆運算器。

對

要複製的物件。

備註

函數:

```
range_adapter();
```

使用預設的結構化反覆運算器，初始化預存反覆運算器的配對。

函數：

```
range_adapter(range_adapter<Iter>% right);
```

藉由複製儲存于 右邊的配對，初始化預存反覆運算器的配對。

函數：

```
range_adapter(range_adapter<Iter>^ right);
```

藉由複製儲存在中的配對，初始化預存反覆運算器的配對 `*right`。

函數：

```
range_adapter(Iter^ first, last);
```

使用 `first` 和 `last` 初始化預存反覆運算器的配對。

範例

```
// cliext_range_adapter_construct.cpp
// compile with: /clr
#include <cliext/adaptor>
#include <cliext/deque>

typedef cliext::deque<wchar_t> Mycont;
typedef cliext::range_adapter<Mycont::iterator> Myrange;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');

    // construct an empty adapter
    Myrange c1;

    // construct with an iterator pair
    Myrange c2(d1.begin(), d1.end());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another adapter
    Myrange c3(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying an adapter handle
    Myrange c4(%c3);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
a b c
a b c
a b c
```

algorithm (STL/CLR)

2020/3/25 • [Edit Online](#)

定義執行演算法的 STL/CLR 容器範本函式。

語法

```
#include <cliext/algorithm>
```

需求

標頭：`<cliext/演算法>`

命名空間：`cliext`

宣告

II	II
adjacent_find (STL/CLR)	搜尋兩個相等的相鄰元素。
binary_search (STL/CLR)	測試排序的序列是否包含指定的值。
copy (STL/CLR)	將值從來源範圍複製到目的範圍，並以正向方向逐一查看。
copy_backward (STL/CLR)	將來源範圍的值複製到目的範圍，並以回溯方向逐一查看。
count (STL/CLR)	傳回範圍中值符合指定值的項目數目。
count_if (STL/CLR)	傳回範圍中值符合指定條件的項目數目。
equal (STL/CLR)	比較兩個範圍，專案的元素。
equal_range (STL/CLR)	搜尋值的已排序序列，並傳回兩個位置，以分隔全都等於指定專案的值子序列。
fill (STL/CLR)	將相同的新值指派到指定範圍內的每個項目。
fill_n (STL/CLR)	將新值指派給範圍中以特定元素開頭的指定元素數目。
find (STL/CLR)	傳回指定值第一次出現的位置。
find_end (STL/CLR)	傳回範圍中與指定序列相同的最後一個子序列。
find_first_of (STL/CLR)	搜尋範圍，尋找任一指定專案範圍中第一個出現的專案。
find_if (STL/CLR)	傳回值序列中，元素符合指定條件的第一個元素的位置。

for_each (STL/CLR)	將指定的函式物件套用至值序列中的每個元素，並傳回函式物件。
generate (STL/CLR)	將函式物件產生的值指派給值序列中的每個元素。
generate_n (STL/CLR)	將函式物件產生的值指派給指定數目的元素。
includes (STL/CLR)	測試一個排序的範圍是否包含第二個排序範圍中的所有元素。
inplace_merge (STL/CLR)	將兩個連續排序範圍的元素結合成單一排序範圍。
iter_swap (STL/CLR)	交換由一組指定之迭代器所參考的兩個值。
lexicographical_compare (STL/CLR)	比較兩個序列，專案的元素，識別哪一個序列是兩者中較小者。
lower_bound (STL/CLR)	以排序的值序列，尋找值大於或等於指定值的第一個元素的位置。
make_heap (STL/CLR)	將指定範圍中的專案轉換為堆積，其中堆積上的第一個元素最大。
max (STL/CLR)	比較兩個物件，並傳回兩者中的最大。
max_element (STL/CLR)	尋找指定的值序列中最大的元素。
merge (STL/CLR)	將兩個已排序來源範圍中的所有元素結合成單一排序目的範圍。
min (STL/CLR)	比較兩個物件，並傳回兩者中的較小者。
min_element (STL/CLR)	尋找指定值序列中的最小元素。
mismatch (STL/CLR)	比較兩個範圍專案的元素，並傳回發生差異的第一個位置。
next_permutation (STL/CLR)	重新排序範圍中的專案，如此一來，詞典編纂下一個較大的排列（如果存在的話），就會取代原始的排序。
nth_element (STL/CLR)	分割一系列專案，正確地找出序列中的 n 個專案，讓它前面的所有元素都小於或等於它，而後面的所有元素都大於或等於它。
partial_sort (STL/CLR)	將範圍中指定數目的較小元素排列成遞減排列順序。
partial_sort_copy (STL/CLR)	從來源範圍將專案複製到目的範圍，以排序來源範圍中的元素。
partition (STL/CLR)	排列範圍中的專案，讓滿足一元述詞的專案在無法滿足它的專案之前。
pop_heap (STL/CLR)	將最大的元素從堆積的前端移至結尾，然後從其餘元素形成新的堆積。

prev_permutation (STL/CLR)	重新排序專案序列，讓原始順序由詞典編纂先前較大的排列取代(如果有的話)。
push_heap (STL/CLR)	將在範圍結尾的項目加入至由範圍中之前項目所組成的現有堆積。
random_shuffle (STL/CLR)	將範圍中的一系列 N 元素重新排列成其中一個 N ！排列方式隨機選取的其中一個。
remove (STL/CLR)	從給定的範圍中刪除指定的值，而不會干擾其餘專案的順序，並傳回沒有指定值的新範圍結尾。
remove_copy (STL/CLR)	從來源範圍將專案複製到目的範圍，但不會複製指定值的元素，而不會干擾其餘專案的順序。
remove_copy_if (STL/CLR)	從來源範圍將專案複製到目的範圍，但滿足述詞的專案除外，而不會干擾其餘元素的順序。
remove_if (STL/CLR)	從指定的範圍刪除滿足述詞的專案，而不會干擾其餘元素的順序。。
replace (STL/CLR)	以新值取代符合指定值之範圍中的元素。
replace_copy (STL/CLR)	從來源範圍將專案複製到目的範圍，以新值取代符合指定值的元素。
replace_copy_if (STL/CLR)	檢查來源範圍內的每個項目，如果滿足指定的述詞則予以取代，同時將結果複製到新目的範圍。
replace_if (STL/CLR)	檢查範圍內的每個項目，如果滿足指定的述詞則予以取代。
reverse (STL/CLR)	反轉範圍內項目的順序。
reverse_copy (STL/CLR)	反轉來源範圍內的專案順序，同時將它們複製到目的範圍。
rotate (STL/CLR)	交換兩個相鄰範圍的項目。
rotate_copy (STL/CLR)	交換來源範圍內兩個相鄰範圍的項目，並將結果複製到目的範圍。
search (STL/CLR)	在目標範圍中搜尋第一個序列，其項目等於指定項目序列中的項目，或在二元述詞指定的意義上，其項目相當於指定序列中的項目。
search_n (STL/CLR)	在範圍中搜尋包含指定項目數的第一個子序列，這些項目具有特定值或在二元述詞指定的意義上與該值關聯。
set_difference (STL/CLR)	將屬於一個排序來源範圍但不屬於第二個排序來源範圍的所有項目聯集為單一排序的目的範圍，其中順序準則可由二元述詞指定。
set_intersection (STL/CLR)	將屬於兩個排序來源範圍的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。

set_symmetric_difference (STL/CLR)	將屬於兩個排序來源範圍之一 (但非兩者) 的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。
set_union (STL/CLR)	將至少屬於兩個排序來源範圍之一的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。
sort (STL/CLR)	將在指定範圍中的項目排列成非遞減排列，或是依據二元述詞指定的順序準則。
sort_heap (STL/CLR)	將堆積轉換為排序的範圍。
stable_partition (STL/CLR)	將範圍中的項目分類為兩個斷續集合，而滿足一元述詞的項目在無法滿足一元述詞的項目之前，保留對等項目的相對順序。
stable_sort (STL/CLR)	將在指定範圍中的項目排列成非遞減排列，或是依據二元述詞指定的順序準則，並保留對等項目的相對順序。
swap (STL/CLR)	在兩種類型的物件之間交換項目的值，將第一個物件的內容指派給第二個物件，並將第一個物件的內容指派給第一個物件。
swap_ranges (STL/CLR)	將某個範圍的項目與另一個相等大小之範圍的項目交換。
transform (STL/CLR)	將指定的函式物件應用至來源範圍中的每個項目，或是一組來自兩個來源範圍的項目，並複製函式物件的傳回值到目的範圍。
unique (STL/CLR)	移除在指定範圍內彼此相鄰的重複項目。
unique_copy (STL/CLR)	將來源範圍的項目複製到目的範圍，但是彼此相鄰的重複項目除外。
upper_bound (STL/CLR)	在已排序範圍中尋找值大於指定值的第一個項目的位置，其中順序準則可由二元述詞指定。

成員

adjacent_find (STL/CLR)

搜尋等於或符合指定之條件的兩個相鄰項目。

語法

```
template<class _FwdIt> inline
    _FwdIt adjacent_find(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt adjacent_find(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `adjacent_find` 相同。如需詳細資訊，請參閱[adjacent_find](#)。

binary_search (STL/CLR)

測試已排序的範圍中是否有等於指定之值 (或在二元述詞指定的意義上，相當於該值) 的項目。

語法

```
template<class _FwdIt, class _Ty> inline  
    bool binary_search(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);  
template<class _FwdIt, class _Ty, class _Pr> inline  
    bool binary_search(_FwdIt _First, _FwdIt _Last,  
        const _Ty% _Val, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `binary_search` 相同。如需詳細資訊，請參閱[binary_search](#)。

copy (STL/CLR)

從來源範圍將項目的值指定到目的範圍，逐一查看項目的來源序列，並以正向方向指派它們新位置。

語法

```
template<class _InIt, class _OutIt> inline  
    _OutIt copy(_InIt _First, _InIt _Last, _OutIt _Dest);
```

備註

此函式的行為與C++標準程式庫函數 `copy` 相同。如需詳細資訊，請參閱[copy](#)。

copy_backward (STL/CLR)

從來源範圍將項目的值指定到目的範圍，逐一查看項目的來源序列，並以反向方向指派它們新位置。

語法

```
template<class _BidIt1, class _BidIt2> inline  
    _BidIt2 copy_backward(_BidIt1 _First, _BidIt1 _Last,  
        _BidIt2 _Dest);
```

備註

此函式的行為與C++標準程式庫函數 `copy_backward` 相同。如需詳細資訊，請參閱[copy_backward](#)。

count (STL/CLR)

傳回範圍中值符合指定值的項目數目。

語法

```
template<class _InIt, class _Ty> inline  
    typename iterator_traits<_InIt>::difference_type  
        count(_InIt _First, _InIt _Last, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `count` 相同。如需詳細資訊，請參閱[count](#)。

count_if (STL/CLR)

傳回範圍中值符合指定條件的項目數目。

語法

```
template<class _InIt, class _Pr> inline
    typename iterator_traits<_InIt>::difference_type
        count_if(_InIt _First, _InIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `count_if` 相同。如需詳細資訊，請參閱[count_if](#)。

等於 (STL/CLR)

逐一比較兩個範圍的每個項目是否相等 (或在二元述詞指定的意義上，是否對等)。

語法

```
template<class _InIt1, class _InIt2> inline
    bool equal(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2);
template<class _InIt1, class _InIt2, class _Pr> inline
    bool equal(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `equal` 相同。如需詳細資訊，請參閱[等於](#)。

equal_range (STL/CLR)

在已排序的範圍中尋找一對位置，第一個位置小於或等於指定項目的位置，第二個位置大於該項目的位置，其中用於建立序列中位置的等價或順序意義可由二元述詞指定。

語法

```
template<class _FwdIt, class _Ty> inline
    _PAIR_TYPE(_FwdIt) equal_range(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    _PAIR_TYPE(_FwdIt) equal_range(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `equal_range` 相同。如需詳細資訊，請參閱[equal_range](#)。

fill (STL/CLR)

將相同的新值指派到指定範圍內的每個項目。

語法

```
template<class _FwdIt, class _Ty> inline
    void fill(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `fill` 相同。如需詳細資訊，請參閱[fill](#)。

fill_n (STL/CLR)

將新值指派給範圍中以特定元素開頭的指定元素數目。

語法

```
template<class _OutIt, class _Diff, class _Ty> inline
void fill_n(_OutIt _First, _Diff _Count, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 [fill_n](#) 相同。如需詳細資訊，請參閱[fill_n](#)。

find (STL/CLR)

在範圍中找出有指定值的第一個項目的位置。

語法

```
template<class _InIt, class _Ty> inline
_InIt find(_InIt _First, _InIt _Last, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 [find](#) 相同。如需詳細資訊，請參閱[尋找](#)。

find_end (STL/CLR)

在範圍中尋找與指定序列相同 (或在二元述詞指定的意義上，相當於該序列) 的最後一個子序列。

語法

```
template<class _FwdIt1, class _FwdIt2> inline
_FwdIt1 find_end(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
_FwdIt1 find_end(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [find_end](#) 相同。如需詳細資訊，請參閱[find_end](#)。

find_first_of (STL/CLR)

在目標範圍內搜尋第一次出現的任何多個值，或第一次出現的任何多個項目 (在二元述詞指定的意義上，相當於指定之項目集合)。

語法

```
template<class _FwdIt1, class _FwdIt2> inline
_FwdIt1 find_first_of(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
_FwdIt1 find_first_of(_FwdIt1 _First1, _FwdIt1 _Last1,
    _FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [find_first_of](#) 相同。如需詳細資訊，請參閱[find_first_of](#)。

find_if (STL/CLR)

在範圍中找出滿足特定條件的第一個項目的位置。

語法

```
template<class _InIt, class _Pr> inline  
_InIt find_if(_InIt _First, _InIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `find_if` 相同。如需詳細資訊，請參閱[find_if](#)。

for_each (STL/CLR)

將指定的函式物件以正向順序套用至範圍內的每個項目，並傳回函式物件。

語法

```
template<class _InIt, class _Fn1> inline  
_Fn1 for_each(_InIt _First, _InIt _Last, _Fn1 _Func);
```

備註

此函式的行為與C++標準程式庫函數 `for_each` 相同。如需詳細資訊，請參閱[for_each](#)。

產生 (STL/CLR)

將函式物件產生的值指派給範圍內的每個項目。

語法

```
template<class _FwdIt, class _Fn0> inline  
void generate(_FwdIt _First, _FwdIt _Last, _Fn0 _Func);
```

備註

此函式的行為與C++標準程式庫函數 `generate` 相同。如需詳細資訊，請參閱[產生](#)。

generate_n (STL/CLR)

將函式物件產生的值指派給範圍內的指定項目數，並返回到超過最後一個指定值的位置。

語法

```
template<class _OutIt, class _Diff, class _Fn0> inline  
void generate_n(_OutIt _Dest, _Diff _Count, _Fn0 _Func);
```

備註

此函式的行為與C++標準程式庫函數 `generate_n` 相同。如需詳細資訊，請參閱[generate_n](#)。

包含 (STL/CLR)

測試一個排序範圍是否包含第二個排序範圍內的所有項目，其中項目之間的順序或等價準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2> inline
    bool includes(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2);
template<class _InIt1, class _InIt2, class _Pr> inline
    bool includes(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `includes` 相同。如需詳細資訊，請參閱[include](#)。

inplace_merge (STL/CLR)

將兩個連續排序範圍內的項目結合成單一排序範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _BidIt> inline
    void inplace_merge(_BidIt _First, _BidIt _Mid, _BidIt _Last);
template<class _BidIt, class _Pr> inline
    void inplace_merge(_BidIt _First, _BidIt _Mid, _BidIt _Last,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數相同 `inplace_merge` 如需詳細資訊，請參閱[inplace_merge](#)。

iter_swap (STL/CLR)

交換由一組指定之迭代器所參考的兩個值。

語法

```
template<class _FwdIt1, class _FwdIt2> inline
    void iter_swap(_FwdIt1 _Left, _FwdIt2 _Right);
```

備註

此函式的行為與C++標準程式庫函數 `iter_swap` 相同。如需詳細資訊，請參閱[iter_swap](#)。

lexicographical_compare (STL/CLR)

逐一比較兩個序列之間的每個項目，判斷兩者較小者。

語法

```
template<class _InIt1, class _InIt2> inline
    bool lexicographical_compare(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2);
template<class _InIt1, class _InIt2, class _Pr> inline
    bool lexicographical_compare(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `lexicographical_compare` 相同。如需詳細資訊，請參閱[lexicographical_compare](#)。

lower_bound (STL/CLR)

尋找具有小於或等於指定值之排序範圍中第一個元素的位置，其中順序準則可由二元述詞指定。

語法

```
template<class _FwdIt, class _Ty> inline  
    _FwdIt lower_bound(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);  
template<class _FwdIt, class _Ty, class _Pr> inline  
    _FwdIt lower_bound(_FwdIt _First, _FwdIt _Last,  
        const _Ty% _Val, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `lower_bound` 相同。如需詳細資訊，請參閱[lower_bound](#)。

make_heap (STL/CLR)

將在指定範圍內的項目轉換為堆積，其中第一個項目是最大，而且排序準則可由二元述詞指定。

語法

```
template<class _RanIt> inline  
    void make_heap(_RanIt _First, _RanIt _Last);  
template<class _RanIt, class _Pr> inline  
    void make_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `make_heap` 相同。如需詳細資訊，請參閱[make_heap](#)。

max (STL/CLR)

比較兩個物件並傳回兩者較大者，其中順序準則可由二元述詞指定。

語法

```
template<class _Ty> inline  
    const _Ty max(const _Ty% _Left, const _Ty% _Right);  
template<class _Ty, class _Pr> inline  
    const _Ty max(const _Ty% _Left, const _Ty% _Right, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `max` 相同。如需詳細資訊，請參閱[max](#)。

max_element (STL/CLR)

在指定的範圍內尋找第一個最大項目，其中順序準則可由二元述詞指定。

語法

```
template<class _FwdIt> inline  
    _FwdIt max_element(_FwdIt _First, _FwdIt _Last);  
template<class _FwdIt, class _Pr> inline  
    _FwdIt max_element(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `max_element` 相同。如需詳細資訊，請參閱[max_element](#)。

merge (STL/CLR)

將兩個排序來源範圍內的所有項目結合成單一排序目的範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt merge(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt merge(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `merge` 相同。如需詳細資訊，請參閱[merge](#)。

min (STL/CLR)

比較兩個物件並傳回兩者較小者，其中順序準則可由二元述詞指定。

語法

```
template<class _Ty> inline
    const _Ty min(const _Ty% _Left, const _Ty% _Right);
template<class _Ty, class _Pr> inline
    const _Ty min(const _Ty% _Left, const _Ty% _Right, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `min` 相同。如需詳細資訊，請參閱[min](#)。

min_element (STL/CLR)

在指定的範圍內尋找第一個最小項目，其中順序準則可由二元述詞指定。

語法

```
template<class _FwdIt> inline
    _FwdIt min_element(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt min_element(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `min_element` 相同。如需詳細資訊，請參閱[min_element](#)。

不相符 (STL/CLR)

逐一比較兩個範圍的每個項目是否相等 (或在二元述詞指定的意義上，是否對等)，而且找出差異發生的第一個位置。

語法

```
template<class _InIt1, class _InIt2> inline
    _PAIR_TYPE(_InIt1)
        mismatch(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2);
template<class _InIt1, class _InIt2, class _Pr> inline
    _PAIR_TYPE(_InIt1)
        mismatch(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
            _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [mismatch](#) 相同。如需詳細資訊，請參閱[不相符](#)。

next_permutation (STL/CLR)

重新排列範圍的項目，讓原始順序由語彙方面下一個較大的排列取代 (如果有的話)，其中下一個的意義可由二元述詞指定。

語法

```
template<class _BidIt> inline
    bool next_permutation(_BidIt _First, _BidIt _Last);
template<class _BidIt, class _Pr> inline
    bool next_permutation(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [next_permutation](#) 相同。如需詳細資訊，請參閱[next_permutation](#)。

nth_element (STL/CLR)

分割某個範圍的專案，並正確地找出範圍中序列的 n 第一個專案，讓它前面的所有元素小於或等於它，而且序列中後面的所有元素都大於或等於該元素。

語法

```
template<class _RanIt> inline
    void nth_element(_RanIt _First, _RanIt _Nth, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void nth_element(_RanIt _First, _RanIt _Nth, _RanIt _Last,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [nth_element](#) 相同。如需詳細資訊，請參閱[nth_element](#)。

partial_sort (STL/CLR)

將範圍中指定的較小項目數目排列成非遞減排列，或是依據二元述詞指定的順序準則。

語法

```
template<class _RanIt> inline
    void partial_sort(_RanIt _First, _RanIt _Mid, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void partial_sort(_RanIt _First, _RanIt _Mid, _RanIt _Last,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `partial_sort` 相同。如需詳細資訊，請參閱[partial_sort](#)。

partial_sort_copy (STL/CLR)

從來源範圍將項目複製到目的範圍，其中來源項目是依小於排序，或依據二元述詞指定的順序準則。

語法

```
template<class _InIt, class _RanIt> inline
    _RanIt partial_sort_copy(_InIt _First1, _InIt _Last1,
                           _RanIt _First2, _RanIt _Last2);
template<class _InIt, class _RanIt, class _Pr> inline
    _RanIt partial_sort_copy(_InIt _First1, _InIt _Last1,
                           _RanIt _First2, _RanIt _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `partial_sort_copy` 相同。如需詳細資訊，請參閱[partial_sort_copy](#)。

partition (STL/CLR)

將範圍中的項目分類為兩個斷續集合，而滿足一元述詞的項目在無法滿足一元述詞的項目之前。

語法

```
template<class _BidIt, class _Pr> inline
    _BidIt partition(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `partition` 相同。如需詳細資訊，請參閱[partition](#)。

pop_heap (STL/CLR)

從堆積的前面移動最大的項目至範圍的倒數第二個位置，然後從其餘項目形成新的堆積。

語法

```
template<class _RanIt> inline
    void pop_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void pop_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `pop_heap` 相同。如需詳細資訊，請參閱[pop_heap](#)。

prev_permutation (STL/CLR)

重新排列範圍的項目，讓原始順序由語彙方面下一個較大的排列取代 (如果有的話)，其中下一個的意義可由二元述詞指定。

語法

```
template<class _BidIt> inline
    bool prev_permutation(_BidIt _First, _BidIt _Last);
template<class _BidIt, class _Pr> inline
    bool prev_permutation(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `prev_permutation` 相同。如需詳細資訊，請參閱[prev_permutation](#)。

push_heap (STL/CLR)

將在範圍結尾的項目加入至由範圍中之前項目所組成的現有堆積。

語法

```
template<class _RanIt> inline
    void push_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void push_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `push_heap` 相同。如需詳細資訊，請參閱[push_heap](#)。

random_shuffle (STL/CLR)

將範圍中的一系列 `N` 元素重新排列成其中一個 `N`！排列方式隨機選取的其中一個。

語法

```
template<class _RanIt> inline
    void random_shuffle(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Fn1> inline
    void random_shuffle(_RanIt _First, _RanIt _Last, _Fn1% _Func);
```

備註

此函式的行為與C++標準程式庫函數 `random_shuffle` 相同。如需詳細資訊，請參閱[random_shuffle](#)。

remove (STL/CLR)

從指定範圍中排除指定的值，而不會干擾其餘項目的順序，並傳回沒有指定值、新範圍的結尾。

語法

```
template<class _FwdIt, class _Ty> inline
    _FwdIt remove(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `remove` 相同。如需詳細資訊，請參閱[remove](#)。

remove_copy (STL/CLR)

從來源範圍將項目複製到目的範圍，不過不會複製一個指定值的項目，也不會干擾其餘項目的順序，並傳回新目的範圍結尾。

語法

```
template<class _InIt, class _OutIt, class _Ty> inline
    _OutIt remove_copy(_InIt _First, _InIt _Last,
        _OutIt _Dest, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `remove_copy` 相同。如需詳細資訊，請參閱[remove_copy](#)。

remove_copy_if (STL/CLR)

從來源範圍將項目複製到目的範圍，不過不會複製滿足述詞的項目，也不會干擾其餘項目的順序，並傳回新目的範圍結尾。

語法

```
template<class _InIt, class _OutIt, class _Pr> inline
    _OutIt remove_copy_if(_InIt _First, _InIt _Last, _OutIt _Dest,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `remove_copy_if` 相同。如需詳細資訊，請參閱[remove_copy_if](#)。

remove_if (STL/CLR)

從指定範圍中排除滿足述詞的項目，而不會干擾其餘項目的順序，並傳回沒有指定值、新範圍的結尾。

語法

```
template<class _FwdIt, class _Pr> inline
    _FwdIt remove_if(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `remove_if` 相同。如需詳細資訊，請參閱[remove_if](#)。

replace (STL/CLR)

檢查範圍內的每個項目，如果符合指定的值則予以取代。

語法

```
template<class _FwdIt, class _Ty> inline
void replace(_FwdIt _First, _FwdIt _Last,
    const _Ty% _Oldval, const _Ty% _Newval);
```

備註

此函式的行為與C++標準程式庫函數 `replace` 相同。如需詳細資訊，請參閱[replace](#)。

replace_copy (STL/CLR)

檢查來源範圍內的每個項目，如果符合指定的值則予以取代，同時將結果複製到新目的範圍。

語法

```
template<class _InIt, class _OutIt, class _Ty> inline
    _OutIt replace_copy(_InIt _First, _InIt _Last, _OutIt _Dest,
        const _Ty% _Oldval, const _Ty% _Newval);
```

備註

此函式的行為與C++標準程式庫函數 `replace_copy` 相同。如需詳細資訊，請參閱[replace_copy](#)。

replace_copy_if (STL/CLR)

檢查來源範圍內的每個項目，如果滿足指定的述詞則予以取代，同時將結果複製到新目的範圍。

語法

```
template<class _InIt, class _OutIt, class _Pr, class _Ty> inline  
_OutIt replace_copy_if(_InIt _First, _InIt _Last, _OutIt _Dest,  
_Pr _Pred, const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `replace_copy_if` 相同。如需詳細資訊，請參閱[replace_copy_if](#)。

replace_if (STL/CLR)

檢查範圍內的每個項目，如果滿足指定的述詞則予以取代。

語法

```
template<class _FwdIt, class _Pr, class _Ty> inline  
void replace_if(_FwdIt _First, _FwdIt _Last, _Pr _Pred,  
const _Ty% _Val);
```

備註

此函式的行為與C++標準程式庫函數 `replace_if` 相同。如需詳細資訊，請參閱[replace_if](#)。

reverse (STL/CLR)

反轉範圍內項目的順序。

語法

```
template<class _BidIt> inline  
void reverse(_BidIt _First, _BidIt _Last);
```

備註

此函式的行為與C++標準程式庫函數 `reverse` 相同。如需詳細資訊，請參閱[reverse](#)。

reverse_copy (STL/CLR)

反轉來源範圍內的專案順序，同時將它們複製到目的範圍。

語法

```
template<class _BidIt, class _OutIt> inline  
_OutIt reverse_copy(_BidIt _First, _BidIt _Last, _OutIt _Dest);
```

備註

此函式的行為與C++標準程式庫函數 `reverse_copy` 相同。如需詳細資訊，請參閱[reverse_copy](#)。

旋轉(STL/CLR)

交換兩個相鄰範圍的項目。

語法

```
template<class _FwdIt> inline
void rotate(_FwdIt _First, _FwdIt _Mid, _FwdIt _Last);
```

備註

此函式的行為與C++標準程式庫函數 [rotate](#) 相同。如需詳細資訊，請參閱[旋轉](#)。

rotate_copy (STL/CLR)

交換來源範圍內兩個相鄰範圍的項目，並將結果複製到目的範圍。

語法

```
template<class _FwdIt, class _OutIt> inline
_OutIt rotate_copy(_FwdIt _First, _FwdIt _Mid, _FwdIt _Last,
_OutIt _Dest);
```

備註

此函式的行為與C++標準程式庫函數 [rotate_copy](#) 相同。如需詳細資訊，請參閱[rotate_copy](#)。

搜尋 (STL/CLR)

在目標範圍中搜尋第一個序列，其項目等於指定項目序列中的項目，或在二元述詞指定的意義上，其項目相當於指定序列中的項目。

語法

```
template<class _FwdIt1, class _FwdIt2> inline
_FwdIt1 search(_FwdIt1 _First1, _FwdIt1 _Last1,
_FwdIt2 _First2, _FwdIt2 _Last2);
template<class _FwdIt1, class _FwdIt2, class _Pr> inline
_FwdIt1 search(_FwdIt1 _First1, _FwdIt1 _Last1,
_FwdIt2 _First2, _FwdIt2 _Last2, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [search](#) 相同。如需詳細資訊，請參閱[搜尋](#)。

search_n (STL/CLR)

在範圍中搜尋包含指定項目數的第一個子序列，這些項目具有特定值或在二元述詞指定的意義上與該值關聯。

語法

```
template<class _FwdIt1, class _Diff2, class _Ty> inline
_FwdIt1 search_n(_FwdIt1 _First1, _FwdIt1 _Last1,
_Diff2 _Count, const _Ty& _Val);
template<class _FwdIt1, class _Diff2, class _Ty, class _Pr> inline
_FwdIt1 search_n(_FwdIt1 _First1, _FwdIt1 _Last1,
_Diff2 _Count, const _Ty& _Val, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [search_n](#) 相同。如需詳細資訊，請參閱[search_n](#)。

set_difference (STL/CLR)

將屬於一個排序來源範圍但不屬於第二個排序來源範圍的所有項目聯集為單一排序的目的範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [set_difference](#) 相同。如需詳細資訊，請參閱[set_difference](#)。

set_intersection (STL/CLR)

將屬於兩個排序來源範圍的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_intersection(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_intersection(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [set_intersection](#) 相同。如需詳細資訊，請參閱[set_intersection](#)。

set_symmetric_difference (STL/CLR)

將屬於兩個排序來源範圍之一 (但非兩者) 的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_symmetric_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_symmetric_difference(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [set_symmetric_difference](#) 相同。如需詳細資訊，請參閱[set_symmetric_difference](#)。

set_union (STL/CLR)

將至少屬於兩個排序來源範圍之一的所有項目聯集為單一排序目的範圍，其中順序準則可由二元述詞指定。

語法

```
template<class _InIt1, class _InIt2, class _OutIt> inline
    _OutIt set_union(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest);
template<class _InIt1, class _InIt2, class _OutIt, class _Pr> inline
    _OutIt set_union(_InIt1 _First1, _InIt1 _Last1,
        _InIt2 _First2, _InIt2 _Last2, _OutIt _Dest, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [set_union](#) 相同。如需詳細資訊，請參閱[set_union](#)。

sort (STL/CLR)

將在指定範圍中的項目排列成非遞減排列，或是依據二元述詞指定的順序準則。

語法

```
template<class _RanIt> inline
    void sort(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void sort(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [sort](#) 相同。如需詳細資訊，請參閱[sort](#)。

sort_heap (STL/CLR)

將堆積轉換為排序的範圍。

語法

```
template<class _RanIt> inline
    void sort_heap(_RanIt _First, _RanIt _Last);
template<class _RanIt, class _Pr> inline
    void sort_heap(_RanIt _First, _RanIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [sort_heap](#) 相同。如需詳細資訊，請參閱[sort_heap](#)。

stable_partition (STL/CLR)

將範圍中的項目分類為兩個斷續集合，而滿足一元述詞的項目在無法滿足一元述詞的項目之前，保留對等項目的相對順序。

語法

```
template<class _BidIt, class _Pr> inline
    _BidIt stable_partition(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [stable_partition](#) 相同。如需詳細資訊，請參閱[stable_partition](#)。

stable_sort (STL/CLR)

將在指定範圍中的項目排列成非遞減排列，或是依據二元述詞指定的順序準則，並保留對等項目的相對順序。

語法

```
template<class _BidIt> inline
void stable_sort(_BidIt _First, _BidIt _Last);
template<class _BidIt, class _Pr> inline
void stable_sort(_BidIt _First, _BidIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 `stable_sort` 相同。如需詳細資訊，請參閱[stable_sort](#)。

swap (STL/CLR)

在兩種類型的物件之間交換項目的值，將第一個物件的內容指派給第二個物件，並將第一個物件的內容指派給第一個物件。

語法

```
<class _Ty> inline
void swap(_Ty% _Left, _Ty% _Right);
```

備註

此函式的行為與C++標準程式庫函數 `swap` 相同。如需詳細資訊，請參閱[swap](#)。

swap_ranges (STL/CLR)

將某個範圍的項目與另一個相等大小之範圍的項目交換。

語法

```
template<class _FwdIt1, class _FwdIt2> inline
_FwdIt2 swap_ranges(_FwdIt1 _First1, _FwdIt1 _Last1,
_FwdIt2 _First2);
```

備註

此函式的行為與C++標準程式庫函數 `swap_ranges` 相同。如需詳細資訊，請參閱[swap_ranges](#)。

轉換 (STL/CLR)

將指定的函式物件應用至來源範圍中的每個項目，或是一組來自兩個來源範圍的項目，並複製函式物件的傳回值到目的範圍。

語法

```
template<class _InIt, class _OutIt, class _Fn1> inline
_OutIt transform(_InIt _First, _InIt _Last, _OutIt _Dest,
_Fn1 _Func);
template<class _InIt1, class _InIt2, class _OutIt, class _Fn2> inline
_OutIt transform(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
_OutIt _Dest, _Fn2 _Func);
```

備註

此函式的行為與C++標準程式庫函數 `transform` 相同。如需詳細資訊，請參閱[轉換](#)。

unique (STL/CLR)

移除在指定範圍內彼此相鄰的重複項目。

語法

```
template<class _FwdIt> inline
    _FwdIt unique(_FwdIt _First, _FwdIt _Last);
template<class _FwdIt, class _Pr> inline
    _FwdIt unique(_FwdIt _First, _FwdIt _Last, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [unique](#) 相同。如需詳細資訊，請參閱[unique](#)。

unique_copy (STL/CLR)

將來源範圍的項目複製到目的範圍，但是彼此相鄰的重複項目除外。

語法

```
template<class _InIt, class _OutIt> inline
    _OutIt unique_copy(_InIt _First, _InIt _Last, _OutIt _Dest);
template<class _InIt, class _OutIt, class _Pr> inline
    _OutIt unique_copy(_InIt _First, _InIt _Last, _OutIt _Dest,
        _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [unique_copy](#) 相同。如需詳細資訊，請參閱[unique_copy](#)。

upper_bound (STL/CLR)

在已排序範圍中尋找值大於指定值的第一個項目的位置，其中順序準則可由二元述詞指定。

語法

```
template<class _FwdIt, class _Ty> inline
    _FwdIt upper_bound(_FwdIt _First, _FwdIt _Last, const _Ty% _Val);
template<class _FwdIt, class _Ty, class _Pr> inline
    _FwdIt upper_bound(_FwdIt _First, _FwdIt _Last,
        const _Ty% _Val, _Pr _Pred);
```

備註

此函式的行為與C++標準程式庫函數 [upper_bound](#) 相同。如需詳細資訊，請參閱[upper_bound](#)。

deque (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別所描述的物件可控制具有隨機存取的變動長度專案序列。您可以使用容器 `deque` 來管理看起來像是連續儲存區塊的一系列專案，但它可以在任一端增加或縮小，而不需要複製任何剩餘的元素。因此，它可以有效率地執行 `double-ended queue`。（因此名稱）。

在下列描述中，與 `GValue` 相同，`Value` 除非後者是 `ref` 類型，在此情況下為 `Value^`。

語法

```
template<typename Value>
ref class deque
    : public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::IDeque<GValue>
{ .... };
```

參數

`GValue`

受控制序列中元素的泛型型別。

`Rep/Test1`

受控制序列中項目的類型。

需求

標頭：`<cliext/deque>`

命名空間：`cliext`

宣告

宣告	說明
<code>deque::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>deque::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>deque::const_reverse_iterator (STL/CLR)</code>	用於受控制序列的常數反向迭代器類型。
<code>deque::difference_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。
<code>deque::generic_container (STL/CLR)</code>	容器的泛型介面類別型。
<code>deque::generic_iterator (STL/CLR)</code>	容器之泛型介面的反覆運算器類型。

<p> </p> <p>deque::generic_reverse_iterator (STL/CLR)</p> <p>deque::generic_value (STL/CLR)</p> <p>deque::iterator (STL/CLR)</p> <p>deque::reference (STL/CLR)</p> <p>deque::reverse_iterator (STL/CLR)</p> <p>deque::size_type (STL/CLR)</p> <p>deque::value_type (STL/CLR)</p>	<p> </p> <p>容器的泛型介面之反向反覆運算器的類型。</p> <p>容器之泛型介面的元素類型。</p> <p>受控制序列之迭代器的類型。</p> <p>項目的參考類型。</p> <p>受控制序列的反向迭代器類型。</p> <p>兩個項目之間帶正負號距離的類型。</p> <p>項目的類型。</p>
<p> </p> <p>deque::assign (STL/CLR)</p> <p>deque::at (STL/CLR)</p> <p>deque::back (STL/CLR)</p> <p>deque::begin (STL/CLR)</p> <p>deque::clear (STL/CLR)</p> <p>deque::deque (STL/CLR)</p> <p>deque::empty (STL/CLR)</p> <p>deque::end (STL/CLR)</p> <p>deque::erase (STL/CLR)</p> <p>deque::front (STL/CLR)</p> <p>deque::insert (STL/CLR)</p> <p>deque::pop_back (STL/CLR)</p> <p>deque::pop_front (STL/CLR)</p> <p>deque::push_back (STL/CLR)</p> <p>deque::push_front (STL/CLR)</p> <p>deque::rbegin (STL/CLR)</p> <p>deque::rend (STL/CLR)</p>	<p> </p> <p>取代所有項目。</p> <p>存取指定位置的項目。</p> <p>存取最後一個項目。</p> <p>指定受控制序列的開頭。</p> <p>移除所有項目。</p> <p>建構容器物件。</p> <p>測試項目是否存在。</p> <p>指定受控制序列的結尾。</p> <p>移除位於指定位置的項目。</p> <p>存取第一個項目。</p> <p>在指定的位置加入專案。</p> <p>移除最後一個元素。</p> <p>移除第一個元素。</p> <p>加入新的最後一個元素。</p> <p>加入新的第一個元素。</p> <p>指定反向受控制序列的開頭。</p> <p>指定反向受控制序列的結尾。</p>

deque::resize (STL/CLR)	變更項目的數目。
deque::size (STL/CLR)	計算元素的數目。
deque::swap (STL/CLR)	交換兩個容器的內容。
deque::to_array (STL/CLR)	將受控制序列複製到新的陣列。
deque::back_item (STL/CLR)	存取最後一個項目。
deque::front_item (STL/CLR)	存取第一個項目。
deque::operator!= (STL/CLR)	判斷兩個 deque 物件是否不相等。
deque::operator(STL/CLR)	存取指定位置的項目。
operator< (deque) (STL/CLR)	判斷 deque 物件是否小於另一個 deque 物件。
operator<= (deque) (STL/CLR)	判斷 deque 物件是否小於或等於另一個 deque 物件。
operator = (deque) (STL/CLR)	取代受控制的序列。
operator== (deque) (STL/CLR)	判斷 deque 物件是否等於另一個 deque 物件。
operator> (deque) (STL/CLR)	判斷 deque 物件是否大於另一個 deque 物件。
operator>= (deque) (STL/CLR)	判斷 deque 物件是否大於或等於另一個 deque 物件。

介面

ICloneable	複製物件。
IEnumerable	透過元素進行序列。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的專案進行序列。
ICollection<T>	維護具類型的元素群組。
IList<T>	維護具類型元素的已排序群組。

`	`
IDeque<值>	維護一般容器。

備註

物件會透過指定專案區塊的已儲存控制碼陣列，配置並釋放它所控制之序列的儲存體 `value`。陣列會隨選成長。成長的發生方式，是將新專案前置或附加的成本固定為常數時間，而且不會干擾其餘的元素。您也可以在常數時間中移除任何結尾的專案，而不會干擾其餘的元素。因此，`deque` 是適用於樣板類別 [併列 \(stl/clr\)](#) 或樣板類別 [堆疊 \(STL/clr\)](#) 之基礎容器的理想候選。

`deque` 物件支援隨機存取反覆運算器，這表示您可以直接參考元素的數值位置，從零開始計算第一個 (`front`) 元素，到最後一個 (`back`) 元素的 `deque:: SIZE (STL/CLR) () - 1`。這也表示 `deque` 是樣板類別 [priority_queue \(STL/CLR\)](#) 之基礎容器的理想候選。

`Deque` 反覆運算器會儲存其相關聯 `deque` 物件的控制碼，以及它所指定之元素的偏差。您只能將反覆運算器與相關聯的容器物件搭配使用。`Deque` 元素的偏差不一定與其位置相同。第一個插入的元素具有偏差零，下一個附加的元素具有偏差1，但下一個前置的元素具有偏差-1。

在任一端插入或清除元素時，不會改變儲存于任何有效偏差的元素值。不過，插入或清除內部元素可以變更儲存在指定偏差的元素值，因此反覆運算器指定的值也可以變更。（容器可能必須在插入之前或下複製元素，以建立一個洞，或在清除之後填滿一個洞）。不過，`deque iterator` 會保持有效，只要其偏差指定有效的元素即可。此外，有效的反覆運算器仍然是 `dereferencable`--您可以使用它來存取或更改所指定的元素值，只要其偏差不等於所傳回反覆運算器的偏差即可 `end()`。

清除或移除元素會呼叫其預存值的析構函式。終結容器會清除所有元素。因此，其元素類型為 `ref` 類別的容器，可確保沒有任何元素 `outlive` 容器。不過要注意的是，控制碼容器並不會摧毀其元素。

成員

`deque:: assign (STL/CLR)`

取代所有項目。

語法

```
void assign(size_type count, value_type val);
template<typename InIt>
void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumerable<Value>^ right);
```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

再

要插入的列舉。

初始值

要插入之元素的值。

備註

第一個成員函式會將受控制的序列取代為值 *val* 的 *count* 元素的重複。您可以使用它來填滿具有相同值的所有專案容器。

如果 `Init` 是整數類型，第二個成員函式的行為會與相同 `assign((size_type)first, (value_type)last)`。否則，它會將受控制序列取代為序列 [`first` , `last`]。您可以使用它讓受控制的序列成為另一個序列的複本。

第三個成員函式會將受控制序列取代為列舉值右邊所指定的序列。您可以使用它，讓受控制的序列成為列舉值所描述之序列的複本。

範例

```
// cliext_deque_assign.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::deque<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    c2.assign(c1.begin(), c1.end() - 1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
x x x x x x
a b
a b c
```

deque:: at (STL/CLR)

存取指定位置的項目。

語法

```
reference at(size_type pos);
```

參數

採購

要存取的項目的位置。

備註

此成員函式會在位置 *pos* 傳回受控制序列之專案的參考。您可以使用它來讀取或寫入您知道其位置的元素。

範例

```
// cliext deque_at.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using at
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // change an entry and redisplay
    c1.at(1) = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a x c
```

deque:: back (STL/CLR)

存取最後一個項目。

語法

```
reference back();
```

備註

此成員函式會傳回受控制序列之最後一個元素的參考，該專案必須為非空白。當您知道它是否存在時，您可以使用它來存取最後一個元素。

範例

```
// cliext_deque_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back() = c
a b x
```

deque::back_item (STL/CLR)

存取最後一個項目。

語法

```
property value_type back_item;
```

備註

屬性會存取受控制序列中的最後一個專案，其必須為非空白。當您知道最後一個元素存在時，就可以使用它來讀取或寫入它。

範例

```
// cliext_deque_back_item.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back_item = c
a b x
```

deque::begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

此成員函式會傳回隨機存取反覆運算器，指定受控制序列的第一個元素，或在空序列結尾以外的專案。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_deque_begin.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::deque<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b
x y c

```

deque:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

此成員函式會有效地呼叫 `deque:: erase (stl/clr) [deque:: begin (stl/clr) () , deque:: end (stl/CLR) ()]`。您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_deque_clear.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

deque:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```

typedef T2 const_iterator;

```

備註

此類型描述未指定類型的物件 `T2`，可做為受控制序列的常數隨機存取反覆運算器。

範例

```

// cliext_deque_const_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::deque<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

deque:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

此類型描述專案的常數參考。

範例

```

// cliext_deque_const_reference.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::deque<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
    {
        // get a const reference to an element
        cliext::deque<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

deque:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

此類型描述未指定類型的物件 `T4`，可做為受控制序列的常數反向反覆運算器。

範例

```
// cliext_deque_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::deque<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::deque<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

deque:: deque (STL/CLR)

建構容器物件。

語法

```
deque();
deque(deque<Value>% right);
deque(deque<Value>^ right);
explicit deque(size_type count);
deque(size_type count, value_type val);
template<typename InIt>
deque(InIt first, InIt last);
deque(System::Collections::Generic::IEnumerable<Value>^ right);
```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

再

要插入的物件或範圍。

初始值

要插入之元素的值。

備註

此構造函式：

```
deque();
```

以沒有專案的方式，初始化受控制的序列。您可以使用它來指定空的初始受控制序列。

此構造函式：

```
deque(deque<Value>% right);
```

使用序列 [,) 初始化受控制的序列 `right.begin()` `right.end()`。您可以使用它來指定初始受控制序列，這是由 `deque` 物件許可權所控制的序列複本。如需反覆運算器的詳細資訊，請參閱[deque:: begin \(stl/clr\)](#) 和 [deque:: end \(stl/clr\)](#)。

此構造函式：

```
deque(deque<Value>^ right);
```

使用序列 [,) 初始化受控制的序列 `right->begin()` `right->end()`。您可以使用它來指定初始受控制序列，這是由 `deque` 物件（其控制碼為 `right`）所控制的序列複本。

此構造函式：

```
explicit deque(size_type count);
```

使用具有值的 `count` 元素，初始化受控制的序列 `value_type()`。您可以使用它來填滿具有預設值之專案的容器。

此構造函式：

```
deque(size_type count, value_type val);
```

使用具有值 `val` 的 `count` 元素，初始化受控制的序列。您可以使用它來填滿具有相同值的所有專案容器。

此構造函式：

```
template<typename InIt>
```

```
deque(InIt first, InIt last);
```

使用序列 [,) 初始化受控制的序列 `first` `last`。您可以使用它，讓受控制的序列成為另一個序列的複本。

此構造函式：

```
deque(System::Collections::Generic::IEnumerable<Value>^ right);
```

使用列舉值許可權所指定的順序，初始化受控制的序列。您可以使用它，讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```

// cliext_deque_construct.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
// construct an empty container
    cliext::deque<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

// construct with a repetition of default values
    cliext::deque<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// construct with a repetition of values
    cliext::deque<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    cliext::deque<wchar_t>::iterator it = c3.end();
    cliext::deque<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an enumeration
    cliext::deque<wchar_t> c5( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    cliext::deque<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying a container handle
    cliext::deque<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

size() = 0
0 0 0
x x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x

```

deque::difference_type (STL/CLR)

兩個元素之間帶正負號距離的類型。

語法

```
typedef int difference_type;
```

備註

此類型描述帶正負號的元素計數。

範例

```
// cliext_deque_difference_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::deque<wchar_t>::difference_type diff = 0;
    for (cliext::deque<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::deque<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
begin()-end() = -3
```

deque::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於 [deque::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 deque 是否是空的。

範例

```
// cliext_deque_empty.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

deque:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

此成員函式會傳回指向受控制序列結尾之外的隨機存取反覆運算器。您會用它來取得指定受控制序列之 `current` 結尾的 `Iterator`, 但是如果受控制序列的長度變更, 它的狀態也會變更。

範例

```

// cliext_deque_end.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::deque<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine(">-- --end() = {0}", *--it);
    System::Console::WriteLine("--end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
>-- --end() = b
--end() = c
a x y

```

deque:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

參數

first

要清除之範圍的開頭。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除 *where* 所指向之受控制序列的元素。您可以使用它來移除單一元素。

第二個成員函式會移除 [*first* , *last*) 範圍中受控制序列中的元素。您可以使用它來移除零個或多個連續元素。

這兩個成員函式會傳回反覆運算器，指定移除任何專案之後剩餘的第一個元素，如果沒有這類元素，則傳回`deque::end (STL/CLR) ()`。

清除元素時，元素複本的數目是抹除結尾和序列最接近端之間的元素數目的線性。(清除序列任一結尾的一個或多個專案時，不會複製任何專案)。

範例

```
// cliext_deque_erase.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::deque<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

deque::front (STL/CLR)

存取第一個項目。

語法

```
reference front();
```

備註

此成員函式會傳回受控制序列中第一個專案的參考，該專案必須為非空白。當您知道它是否存在時，您可以使用它來讀取或寫入第一個元素。

範例

```
// cliext_deque_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

deque:: front_item (STL/CLR)

存取第一個項目。

語法

```
property value_type front_item;
```

備註

屬性會存取受控制序列的第一個元素，其必須為非空白。當您知道它是否存在時，您可以使用它來讀取或寫入第一個元素。

範例

```
// cliext_deque_front_item.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front_item = a
x b c
```

deque:: generic_container (STL/CLR)

容器的泛型介面類別型。

語法

```
typedef Microsoft::VisualC::StlClr::
    IDeque<generic_value>
generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_deque_generic_container.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

deque::generic_iterator (STL/CLR)

用於容器之泛型介面的反覆運算器類型。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerRandomAccessIterator<generic_value> generic_iterator;

```

備註

此類型描述的泛型反覆運算器可與此樣板容器類別的泛型介面搭配使用。

範例

```

// cliext_deque_generic_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

deque:: generic_reverse_iterator (STL/CLR)

要與容器的泛型介面搭配使用的反向反覆運算器類型。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value> generic_reverse_iterator;

```

備註

此類型描述的泛型反向反覆運算器可與此樣板容器類別的泛型介面搭配使用。

範例

```

// cliext_deque_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c c

```

deque:: generic_value (STL/CLR)

要與容器的泛型介面搭配使用之元素的類型。

語法

```

typedef GValue generic_value;

```

備註

此類型描述類型的物件 `GValue`，其描述要與這個樣板容器類別的泛型介面搭配使用的預存元素值。

範例

```

// cliext_deque_generic_value.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::deque<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::deque<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::deque<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

deque:: insert (STL/CLR)

在指定的位置加入專案。

語法

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

再

要插入的列舉。

初始值

要插入之元素的值。

where

在容器中要插入的位置。

備註

每個成員函式都會在受控制序列中的專案(由其餘運算元所指定的序列)之前，插入所指向的元素之前。

第一個成員函式會插入具有值 *val* 的元素，並傳回反覆運算器，指定新插入的專案。您可以使用它，在反覆運算器所指定的位置之前插入單一元素。

第二個成員函式會插入 *value val* 的 *count* 元素的重複專案。您可以使用它來插入零個或多個連續元素，這些都是相同值的所有複本。

如果 `Init` 是整數類型，第三個成員函式的行為即與 `insert(where, (size_type)first, (value_type)last)` 相同。否則，它會插入序列 [`first` , `last`]。您可以使用它來插入從另一個序列複製的零個或多個連續元素。

第四個成員函式會插入右邊指定的序列。您可以使用它來插入列舉值所描述的序列。

當插入單一專案時，元素複本的數目是插入點和序列最接近端之間的元素數目的線性。(在序列的任一端插入一或多個專案時，不會複製任何元素)。如果 `Init` 是輸入反覆運算器，則第三個成員函式會針對序列中的每個元素，有效地執行單一插入。否則，在插入專案時 `N`，元素複本的數目會是線性，`N` 加上插入點和序列最接近端之間的元素數目。

範例

```

// cliext_deque_insert.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using iterator
    cliext::deque<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("insert(begin())+1, L'x') = {0}",
        *c1.insert(++it, L'x'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a repetition of values
    cliext::deque<wchar_t> c2;
    c2.insert(c2.begin(), 2, L'y');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    it = c1.end();
    c2.insert(c2.end(), c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    c2.insert(c2.begin(), // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(begin())+1, L'x') = x
a x b c
y y
y y a x b
a x b c y y a x b

```

deque:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

此類型描述未指定類型的物件 `T1`，可作為受控制序列的隨機存取反覆運算器。

範例

```
// cliext_deque_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::deque<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

deque:: operator != (STL/CLR)

Deque 不等於比較。

語法

```
template<typename Value>
bool operator!=(deque<Value>% left,
                 deque<Value>% right);
```

參數

左面

要比較的左容器。

再

要比較的右容器。

備註

運算子函式會傳回 `!(left == right)`。當兩個 deques 是以元素進行比較時，您可以使用它來測試是否將 `left` 與 `right` 排序。

範例

```

// cliext_deque_operator_ne.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

deque:: operator (STL/CLR)

存取指定位置的項目。

語法

```
reference operator[](size_type pos);
```

參數

採購

要存取的項目的位置。

備註

成員運算子會將其他參考資料傳回至位置 *pos* 的元素。您可以使用它來存取您知道其位置的元素。

範例

```
// cliext_deque_operator_sub.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using subscripting
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();

    // change an entry and redisplay
    c1[1] = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a x c
```

deque::pop_back (STL/CLR)

移除最後一個元素。

語法

```
void pop_back();
```

備註

此成員函式會移除受控制序列中的最後一個專案，此專案必須為非空白。您可以使用它來縮短背景上一個元素的 deque。

範例

```
// cliext_deque_pop_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b
```

deque::pop_front (STL/CLR)

移除第一個元素。

語法

```
void pop_front();
```

備註

此成員函式會移除受控制序列中的第一個元素，此專案必須為非空白。您可以使用它來縮短前端一個元素的 deque。

範例

```

// cliext_deque_pop_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_front();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

deque::push_back (STL/CLR)

加入新的最後一個元素。

語法

```
void push_back(value_type val);
```

備註

成員函式會將具有值的元素插入 `val` 受控制序列的結尾。您可以使用它將另一個元素附加至 `deque`。

範例

```

// cliext_deque_push_back.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

deque::push_front (STL/CLR)

加入新的第一個元素。

語法

```
void push_front(value_type val);
```

備註

成員函式會將具有值的元素插入 `val` 受控制序列的開頭。您可以用它來在 `deque` 前面加上另一個元素。

範例

```
// cliext_deque_push_front.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_front(L'a');
    c1.push_front(L'b');
    c1.push_front(L'c');

    // display contents " c b a"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

deque::rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

此成員函式會傳回反向反覆運算器，指定受控制序列的最後一個元素，或只在空白序列開頭以外的專案。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_deque_rbegin.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
a y x
```

deque:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

此類型描述專案的參考。

範例

```

// cliext_deque_reference.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::deque<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        cliext::deque<wchar_t>::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        { // get a reference to an element
        cliext::deque<wchar_t>::reference ref = *it;

        ref += (wchar_t)(L'A' - L'a');
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
A B C

```

deque:: rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

此成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 `Iterator` 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_deque_rend.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine(">-- --rend() = {0}", *--rit);
    System::Console::WriteLine(">--rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
"-- --rend() = b
"--rend() = a
y x c

```

deque:: resize (STL/CLR)

變更項目的數目。

語法

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

參數

new_size

受控制序列的新大小。

初始值

填補元素的值。

備註

成員函式可確保 [deque:: size \(STL/CLR\) \(\)](#) 因而需要會傳回 *new_size*。如果它必須讓受控制的序列變長，第一個成員函式會附加具有值的元素 [value_type\(\)](#)，而第二個成員函式會附加具有值 *val* 的元素。為了讓受控制的序列更短，這兩個成員函式會有效地清除最後一個元素 [deque:: size \(STL/CLR\) \(\) - new_size](#) 時間。您可以使用它來確保受控制的序列具有大小 *new_size*，方法是修剪或填補目前受控制的序列。

範例

```
// cliext_deque_resize.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
// construct an empty container and pad with default values
    cliext::deque<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

    // resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
0 0 0 0
size() = 0
x x x x x
```

deque:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_deque_reverse_iterator.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::deque<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
x b a
```

deque:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[deque:: empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_deque_size.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

deque:: size_type (STL/CLR)

兩個元素之間帶正負號距離的類型。

語法

```
typedef int size_type;
```

備註

此類型描述非負的元素計數。

範例

```
// cliext_deque_size_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::deque<wchar_t>::size_type diff = c1.end() - c1.begin();
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

deque:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(deque<Value>% right);
```

參數

再

要交換內容的容器。

備註

成員函式會在和 *right* 之間交換受控制的序列 `*this`。*right* 它會以常數時間執行，而且不會擲回任何例外狀況。您可以用它來快速交換兩個容器的內容。

範例

```

// cliext_deque_swap.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::deque<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
x x x x x
x x x x x
a b c

```

deque:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<Value>^ to_array();
```

備註

此成員函式會傳回陣列，其中包含受控制的序列。您可以用它來取得陣列表單中受控制序列的複本。

範例

```
// cliext_deque_to_array.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

deque::value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數值同義。

範例

```

// cliext_deque_value_type.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::deque<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {
        // store element in value_type object
        cliext::deque<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

operator < (deque) (STL/CLR)

Deque 小於比較。

語法

```

template<typename Value>
bool operator<(deque<Value>% left,
                 deque<Value>% right);

```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

如果的最低位置也為 true, 則運算子函數 `i` 會傳回 true `!(right[i] < left[i])` `left[i] < right[i]`。否則, 它會傳回 `left->size() < right->size()` 您用它來測試 `left` 當兩個 deques 是以元素進行比較時, 是否要在右邊排序。

範例

```

// cliext_deque_operator_lt.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}

```

```

a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True

```

operator < = (deque) (STL/CLR)

Deque 小於或等於比較。

語法

```

template<typename Value>
bool operator<=(deque<Value>% left,
                 deque<Value>% right);

```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

運算子函式會傳回 `!(right < left)`。當兩個 deques 是以元素進行比較時，您可以使用它來測試左側是否未在右邊排序。

範例

```
// cliext_deque_operator_le.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator = (deque) (STL/CLR)

取代受控制的序列。

語法

```
deque<Value>% operator=(deque<Value>% right);
```

參數

再

要複製的容器。

備註

成員運算子會將許可權複製到物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為右邊的受控制序列複本。

範例

```

// cliext_deque_operator_as.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c

```

operator == (deque) (STL/CLR)

Deque 相等比較。

語法

```

template<typename Value>
bool operator==(deque<Value>% left,
                  deque<Value>% right);

```

參數

左面

要比較的左容器。

再

要比較的右容器。

備註

只有當 *left* 和 *right* 所控制的序列具有相同的長度，且每個位置都有相同的時，運算子函數才會傳回 true *i*
`left[i] == right[i]`。您可以使用它來測試當兩個 deques 是以元素進行比較時，是否將 *left* 與 *right* 排序。

範例

```

// cliext_deque_operator_eq.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}

```

```

a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False

```

operator > (deque) (STL/CLR)

Deque 大於比較。

語法

```

template<typename Value>
bool operator>(deque<Value>% left,
                  deque<Value>% right);

```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

運算子函式會傳回 `right < left`。您可以使用它來測試當兩個 deques 是以元素進行比較時，是否要向**右排序。

範例

```
// cliext_deque_operator_gt.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (deque) (STL/CLR)

Deque 大於或等於比較。

語法

```
template<typename Value>
bool operator>=(deque<Value>% left,
                 deque<Value>% right);
```

參數

左面

要比較的左容器。

右面

要比較的右容器。

備註

運算子函式會傳回 `!(left < right)`。當兩個 deques 是以元素進行比較時，您可以使用它來測試左側是否未排序。

範例

```
// cliext_deque_operator_ge.cpp
// compile with: /clr
#include <cliext/deque>

int main()
{
    cliext::deque<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::deque<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

functional (STL/CLR)

2020/11/2 • [Edit Online](#)

包含 STL/CLR 標頭 `<cliext/functional>` 來定義一些範本類別以及相關的範本委派和函式。

Syntax

```
#include <functional>
```

需求

標頭 : `<cliext/functional>`

命名空間 : `cliext`

宣告

III	II
<code>binary_delegate (STL/CLR)</code>	雙引數的委派。
<code>binary_delegate_noreturn (STL/CLR)</code>	傳回兩個引數的委派 <code>void</code> 。
<code>unary_delegate (STL/CLR)</code>	單向引數的委派。
<code>unary_delegate_noreturn (STL/CLR)</code>	傳回一個引數的委派 <code>void</code> 。
II	II
<code>binary_negate (STL/CLR)</code>	仿函數，以否定雙引數仿函數。
<code>binder1st (STL/CLR)</code>	仿函數可將第一個引數系結至兩個引數仿函數。
<code>binder2nd (STL/CLR)</code>	仿函數可將第二個引數系結至兩個引數仿函數。
<code>divides (STL/CLR)</code>	除仿函數。
<code>equal_to (STL/CLR)</code>	相等的比較仿函數。
<code>greater (STL/CLR)</code>	更大的比較仿函數。
<code>greater_equal (STL/CLR)</code>	較大或相等的比較仿函數。
<code>less (STL/CLR)</code>	比較不仿函數。
<code>less_equal (STL/CLR)</code>	小於或等於比較仿函數。

logical_and (STL/CLR)	邏輯 AND 仿函數。
logical_not (STL/CLR)	邏輯 NOT 仿函數。
logical_or (STL/CLR)	邏輯 OR 仿函數。
minus (STL/CLR)	減去仿函數。
modulus (STL/CLR)	模數仿函數。
multiplies (STL/CLR)	將仿函數相乘。
negate (STL/CLR)	仿函數會傳回其引數否定。
not_equal_to (STL/CLR)	不等於比較仿函數。
plus (STL/CLR)	新增仿函數。
unary_negate (STL/CLR)	仿函數，以否定單一引數仿函數。

bind1st (STL/CLR)	產生引數和仿函數的 binder1st。
bind2nd (STL/CLR)	產生引數和仿函數的 binder2nd。
not1 (STL/CLR)	產生仿函數的 unary_negate。
not2 (STL/CLR)	產生仿函數的 binary_negate。

成員

binary_delegate (STL/CLR)

Genereic 類別描述兩個引數的委派。您可以使用它來指定委派的引數和傳回類型。

語法

```
generic<typename Arg1,
        typename Arg2,
        typename Result>
    delegate Result binary_delegate(Arg1, Arg2);
```

參數

Arg1

第一個引數的型別。

Arg2

第二個引數的型別。

結果

傳回類型。

備註

Genereic 委派描述兩個引數函數。

請注意：

```
binary_delegate<int, int, int> Fun1;
```

```
binary_delegate<int, int, int> Fun2;
```

型別 `Fun1` 和 `Fun2` 都是同義字，而針對：

```
delegate int Fun1(int, int);
```

```
delegate int Fun2(int, int);
```

它們的類型不同。

範例

```
// cliext_binary_delegate.cpp
// compile with: /clr
#include <cliext/functional>

bool key_compare(wchar_t left, wchar_t right)
{
    return (left < right);
}

typedef cliext::binary_delegate<wchar_t, wchar_t, bool> Mydelegate;
int main()
{
    Mydelegate^ kcomp = gcnew Mydelegate(&key_compare);

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}
```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

binary_delegate_noreturn (STL/CLR)

Genereic 類別描述傳回的雙引數委派 `void`。您可以使用它來指定委派的引數。

語法

```
generic<typename Arg1,
        typename Arg2>
    delegate void binary_delegate(Arg1, Arg2);
```

參數

Arg1

第一個引數的型別。

Arg2

第二個引數的型別。

備註

Genereic 委派描述傳回的雙引數函數 `void`。

請注意：

```
binary_delegate_noreturn<int, int> Fun1;
```

```
binary_delegate_noreturn<int, int> Fun2;
```

型別 `Fun1` 和 `Fun2` 都是同義字，而針對：

```
delegate void Fun1(int, int);
```

```
delegate void Fun2(int, int);
```

它們的類型不同。

範例

```
// cliext_binary_delegate_noreturn.cpp
// compile with: /clr
#include <cliext/functional>

void key_compare(wchar_t left, wchar_t right)
{
    System::Console::WriteLine("compare({0}, {1}) = {2}",
        left, right, left < right);
}

typedef cliext::binary_delegate_noreturn<wchar_t, wchar_t> Mydelegate;
int main()
{
    Mydelegate^ kcomp = gcnew Mydelegate(&key_compare);

    kcomp(L'a', L'a');
    kcomp(L'a', L'b');
    kcomp(L'b', L'a');
    System::Console::WriteLine();
    return (0);
}
```

```
compare(a, a) = False
compare(a, b) = True
compare(b, a) = False
```

binary_negate (STL/CLR)

此樣板類別描述的仿函數，會在呼叫時傳回其儲存的雙引數仿函數的邏輯 NOT。您可以使用它來根據其預存仿函數來指定函式物件。

語法

```

template<typename Fun>
ref class binary_negate
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    explicit binary_negate(Fun% functor);
    binary_negate(binary_negate<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

好玩

預存仿函數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
stored_function_type	仿函數的類型。
MEMBER	
binary_negate	結構仿函數。
operator()	計算所需的函數。
operator delegate_type ^ (# A1	將仿函數轉換為委派。

備註

此樣板類別描述兩個引數仿函數，可儲存另一個雙引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，它會傳回以兩個引數呼叫之預存仿函數的邏輯 NOT。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_binary_negate.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::less<int> less_op;

    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(),
        cliext::binary_negate<cliext::less<int> >(less_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::not2(less_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
1 0
1 0

```

bind1st (STL/CLR)

產生 `binder1st` 引數和仿函數的。

Syntax

```

template<typename Fun,
        typename Arg>
binder1st<Fun> bind1st(Fun% functor,
                        Arg left);

```

範本參數

精氨酸

引數型別。

好玩

仿函數的類型。

函數參數

函

要包裝的仿函數。

離開

要換行的第一個引數。

備註

範本函式會傳回**binder1st (STL/CLR)** `<Fun>(functor, left)`。您可以使用它做為將雙引數仿函數和其第一個引數包裝在使用第二個引數呼叫它的單一引數仿函數中的便利方法。

範例

```
// cliext_bind1st.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder1st<cliext::minus<int> > subfrom3(sub_op, 3);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     subfrom3);
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind1st(sub_op, 3));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
-1 0
-1 0
```

bind2nd (STL/CLR)

產生 `binder2nd` 引數和仿函數的。

Syntax

```
template<typename Fun,
         typename Arg>
binder2nd<Fun> bind2nd(Fun% functor,
                         Arg right);
```

範本參數

精氨酸

引數型別。

好玩

仿函數的類型。

函數參數

函

要包裝的仿函數。

對

要換行的第二個引數。

備註

範本函式會傳回 `binder2nd (STL/CLR) <Fun>(functor, right)`。您可以使用它做為將雙引數仿函數和第二個引數包裝在使用第一個引數呼叫它的單一引數仿函數中的便利方法。

範例

```

// cliext_bind2nd.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder2nd<cliext::minus<int> > sub4(sub_op, 4);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                     sub4);
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
                      bind2nd(sub_op, 4));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
0 -1
0 -1

```

binder1st (STL/CLR)

此樣板類別描述一個引數仿函數，在呼叫時，會傳回其預存的雙引數仿函數，並使用其預存的第一個引數和提供的第二個引數來呼叫它。您可以使用它來根據其預存仿函數來指定函式物件。

語法

```

template<typename Fun>
ref class binder1st
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef typename Fun::result_type result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        second_argument_type, result_type>
    delegate_type;

    binder1st(Fun% functor, first_argument_type left);
    binder1st(binder1st<Arg>% right);

    result_type operator()(second_argument_type right);
    operator delegate_type^();
};


```

參數

好玩

預存仿函數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
stored_function_type	仿函數的類型。
MEMBER	
binder1st	結構仿函數。
operator()	計算所需的函數。
operator delegate_type ^ (# A1	將仿函數轉換為委派。

備註

此樣板類別描述一個可儲存兩個引數仿函數和第一個引數的單一引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，就會傳回以預存的第一個引數和所提供的第二個引數來呼叫預存仿函數的結果。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_binder1st.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder1st<cliext::minus<int> > subfrom3(sub_op, 3);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        subfrom3);
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        bind1st(sub_op, 3));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
-1 0
-1 0

```

binder2nd (STL/CLR)

此樣板類別描述一個引數仿函數，在呼叫時，會傳回其預存的雙引數仿函數，並使用提供的第一個引數和其儲存的第二個引數來呼叫它。您可以使用它來根據其預存仿函數來指定函式物件。

語法

```

template<typename Fun>
ref class binder2nd
{ // wrap operator()
public:
    typedef Fun stored_function_type;
    typedef typename Fun::first_argument_type first_argument_type;
    typedef typename Fun::second_argument_type second_argument_type;
    typedef typename Fun::result_type result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        first_argument_type, result_type>
        delegate_type;

    binder2nd(Fun% functor, second_argument_type left);
    binder2nd(binder2nd<Arg>% right);

    result_type operator()(first_argument_type right);
    operator delegate_type^();
};


```

參數

好玩

預存仿函數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
stored_function_type	仿函數的類型。
MEMBER	
binder2nd	結構仿函數。
operator()	計算所需的函數。
operator delegate_type ^ (# A1	將仿函數轉換為委派。

備註

此樣板類別描述一個可儲存兩個引數仿函數和第二個引數的單一引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，就會傳回以提供的第一個引數和儲存的第二個引數來呼叫預存仿函數的結果。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_binder2nd.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::minus<int> sub_op;
    cliext::binder2nd<cliext::minus<int> > sub4(sub_op, 4);

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        sub4);
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        bind2nd(sub_op, 4));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
0 -1
0 -1

```

(STL/CLR)

此樣板類別描述的仿函數，會在呼叫時傳回第一個引數除以第二個引數。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class divides
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    divides();
    divides(divides<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數和傳回值的型別。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	
divides	結構仿函數。

operator()	計算所需的函數。
operator delegate_type ^ (# A1	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，就會傳回第一個引數除以第二個。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_divides.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::divides<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
2 3

```

equal_to (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數等於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class equal_to
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    equal_to();
    equal_to(equal_to<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
equal_to	結構仿函數。
operator()	計算所需的函數。
operator delegate_type ^ (# A1	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數等於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_equal_to.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::equal_to<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
1 0

```

更多 (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數大於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class greater
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    greater();
    greater(greater<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
MEMBER	
greater	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數大於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_greater.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 3 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::greater<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
3 3
1 0

```

greater_equal (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數大於或等於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class greater_equal
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    greater_equal();
    greater_equal(greater_equal<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
greater_equal	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數大於或等於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_greater_equal.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::greater_equal<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
1 0

```

less (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數小於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class less
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    less();
    less(less<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
MEMBER	
less	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數小於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_less.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::less<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
4 4
0 1

```

less_equal (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數小於或等於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class less_equal
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    less_equal();
    less_equal(less_equal<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	
less_equal	結構仿函數。

operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數小於或等於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_less_equal.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(3);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 3 3"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::less_equal<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
3 3
0 1

```

logical_and (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數和第二個測試都是 true 時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class logical_and
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    logical_and();
    logical_and(logical_and<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數的類型。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	
logical_and	結構仿函數。

operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數和第二個測試都是 true 時，才會傳回 true。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_logical_and.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(2);
    c1.push_back(0);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 1 0" and " 1 0"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::logical_and<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

2 0
3 0
1 0

```

logical_not (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有在其引數測試為 false 時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class logical_not
{ // wrap operator()
public:
    typedef Arg argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        argument_type, result_type>
        delegate_type;

    logical_not();
    logical_not(logical_not<Arg> %right);

    result_type operator()(argument_type left);
    operator delegate_type^();
};

```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
argument_type	仿函數引數的類型。
delegate_type	泛型委派的型別。
result_type	仿函數結果的類型。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述一個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有在其引數測試為 `false` 時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```
// cliext_logical_not.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                     c3.begin(), cliext::logical_not<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

4 0
0 1

logical_or (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數或第二個測試為 true 時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```
template<typename Arg>
ref class logical_or
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    logical_or();
    logical_or(logical_or<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};

};


```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	ff
logical_or	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數或第二個測試為 true 時，才會傳回 true。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```
// cliext_logical_or.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(2);
    c1.push_back(0);
    Myvector c2;
    c2.push_back(0);
    c2.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 2 0" and " 0 0"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::logical_or<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
2 0
0 0
1 0
```

減號 (STL/CLR)

此樣板類別描述的仿函數，會在呼叫時傳回第一個引數減去第二個引數。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class minus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    minus();
    minus(minus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數和傳回值的型別。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。
MEMBER	
minus	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，它會傳回第一個引數減去第二個引數。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_minus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::minus<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
2 2

```

模數 (STL/CLR)

此樣板類別描述的仿函數，會在呼叫時傳回第一個引數模數第二個引數。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class modulus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    modulus();
    modulus(modulus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數和傳回值的型別。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	
模數	結構仿函數。

operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，它會傳回第一個引數模數第二個引數。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_modulus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(2);
    Myvector c2;
    c2.push_back(3);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 2" and " 3 1"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::modulus<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 2
3 1
1 0

```

(STL/CLR) 相乘

此樣板類別描述的仿函數，在呼叫時，會傳回第二個引數的第二次。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class multiplies
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
    delegate_type;

    multiplies();
    multiplies(multiplies<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};


```

參數

精氨酸

引數和傳回值的型別。

成員函數

delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	
multiplies	結構仿函數。

operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，就會傳回第一個引數的第二次。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_multiplies.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::multiplies<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
8 3

```

(STL/CLR) 否定

此樣板類別描述的仿函數，會在呼叫時傳回其引數否定。您可以使用它來指定函式物件的引數類型。

語法

```

template<typename Arg>
ref class negate
{ // wrap operator()
public:
    typedef Arg argument_type;
    typedef bool result_type;
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<
        argument_type, result_type>
        delegate_type;

    negate();
    negate(negate<Arg>% right);

    result_type operator()(argument_type left);
    operator delegate_type^();
};

```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
argument_type	仿函數引數的類型。
delegate_type	泛型委派的型別。
result_type	仿函數結果的類型。
negate	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述一個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，就會傳回其引數否定。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```
// cliext_negate.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(-3);
    Myvector c3(2, 0);

    // display initial contents " 4 -3"
    for each (int elem in c1)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                     c3.begin(), cliext::negate<int>());
    for each (int elem in c3)
        System::Console::Write("{0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 -3  
-4 3
```

not_equal_to (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，只有當第一個引數不等於第二個引數時，才會傳回 true。您可以使用它來指定函式物件的引數類型。

語法

```
template<typename Arg>  
ref class not_equal_to  
{ // wrap operator()  
public:  
    typedef Arg first_argument_type;  
    typedef Arg second_argument_type;  
    typedef bool result_type;  
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<  
        first_argument_type, second_argument_type, result_type>  
    delegate_type;  
  
    not_equal_to();  
    not_equal_to(not_equal_to<Arg>% right);  
  
    result_type operator()(first_argument_type left,  
                           second_argument_type right);  
    operator delegate_type^();  
};
```

參數

精氨酸

引數的類型。

成員函數

MEMBER	ff
not_equal_to	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，只有當第一個引數不等於第二個引數時，才會傳回 `true`。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```
// cliext_not_equal_to.cpp
// compile with: /clr
#include <cliext/algorithms>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
        c2.begin(), c3.begin(), cliext::not_equal_to<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
4 4
0 1
```

not1 (STL/CLR)

`unary_negate` 為仿函數產生。

Syntax

```
template<typename Fun>
unary_negate<Fun> not1(Fun% functor);
```

範本參數

好玩

仿函數的類型。

函數參數

函

要包裝的仿函數。

備註

範本函式會傳回 [unary_negate \(STL/CLR\)](#) `<Fun>(functor)`。您可以使用它做為將單一引數仿函數包裝在仿函數中的方法，以提供邏輯 NOT。

範例

```
// cliext_not1.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::logical_not<int> not_op;

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::unary_negate<cliext::logical_not<int> >(not_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::not1(not_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 0
1 0
1 0
```

not2 (STL/CLR)

`binary_negate` 為仿函數產生。

Syntax

```
template<typename Fun>
binary_negate<Fun> not2(Fun% functor);
```

範本參數

好玩

仿函數的類型。

函數參數

函

要包裝的仿函數。

備註

範本函式會傳回[binary_negate \(STL/CLR\)](#) `<Fun>(functor)`。您可以使用它來將兩個引數仿函數包裝在仿函數中，以提供邏輯 NOT。

範例

```
// cliext_not2.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(4);
    c2.push_back(4);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 4 4"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::less<int> less_op;

    cliext::transform(c1.begin(), c1.begin() + 2,
                     c2.begin(), c3.begin(),
                     cliext::binary_negate<cliext::less<int> >(less_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2,
                     c2.begin(), c3.begin(), cliext::not2(less_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 3
4 4
1 0
1 0
```

加上 (STL/CLR)

此樣板類別描述的仿函數，會在呼叫時傳回第一個引數再加上第二個引數。您可以使用它來指定函式物件的引數類型。

語法

```
template<typename Arg>
ref class plus
{ // wrap operator()
public:
    typedef Arg first_argument_type;
    typedef Arg second_argument_type;
    typedef Arg result_type;
    typedef Microsoft::VisualC::StlClr::BinaryDelegate<
        first_argument_type, second_argument_type, result_type>
        delegate_type;

    plus();
    plus(plus<Arg>% right);

    result_type operator()(first_argument_type left,
                           second_argument_type right);
    operator delegate_type^();
};
```

參數

精氨酸

引數和傳回值的型別。

成員函數

MEMBER	DEFINITION
delegate_type	泛型委派的型別。
first_argument_type	仿函數第一個引數的型別。
result_type	仿函數結果的類型。
second_argument_type	仿函數第二個引數的類型。

MEMBER	DEFINITION
plus	結構仿函數。
operator()	計算所需的函數。
運算子 delegate_type ^	將仿函數轉換為委派。

備註

此範本類別描述兩個引數仿函數。它會定義成員運算子 `operator()`，如此一來，當物件被呼叫為函式時，它會傳回第一個引數再加上第二個引數。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```

// cliext_plus.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(3);
    Myvector c2;
    c2.push_back(2);
    c2.push_back(1);
    Myvector c3(2, 0);

    // display initial contents " 4 3" and " 2 1"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    for each (int elem in c2)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::transform(c1.begin(), c1.begin() + 2,
                      c2.begin(), c3.begin(), cliext::plus<int>());
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

4 3
2 1
6 4

```

unary_delegate (STL/CLR)

Genereic 類別描述一個引數的委派。您可以使用它來指定委派的引數和傳回類型。

語法

```

generic<typename Arg,
        typename Result>
delegate Result unary_delegate(Arg);

```

參數

精氨酸

引數型別。

結果

傳回類型。

備註

Genereic 委派會描述單一引數函數。

請注意：

```
unary_delegate<int, int> Fun1;
```

```
unary_delegate<int, int> Fun2;
```

型別 `Fun1` 和 `Fun2` 都是同義字，而針對：

```
delegate int Fun1(int);
```

```
delegate int Fun2(int);
```

它們的類型不同。

範例

```
// cliext_unary_delegate.cpp
// compile with: /clr
#include <cliext/functional>

int hash_val(wchar_t val)
{
    return ((val * 17 + 31) % 67);

typedef cliext::unary_delegate<wchar_t, int> Mydelegate;
int main()
{
    Mydelegate^ myhash = gcnew Mydelegate(&hash_val);

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 5
hash(L'b') = 22
```

unary_delegate_noreturn (STL/CLR)

Genereic 類別描述傳回的單一引數委派 `void`。您可以使用它來指定委派的引數類型。

語法

```
generic<typename Arg>
    delegate void unary_delegate_noreturn(Arg);
```

參數

精氨酸

引數型別。

備註

Genereic 委派會描述傳回的單一引數函數 `void`。

請注意：

```
unary_delegate_noreturn<int> Fun1;
```

```
unary_delegate_noreturn<int> Fun2;
```

型別 `Fun1` 和 `Fun2` 都是同義字，而針對：

```
delegate void Fun1(int);  
  
delegate void Fun2(int);
```

它們的類型不同。

範例

```
// cliext_unary_delegate_noreturn.cpp  
// compile with: /clr  
#include <cliext/functional>  
  
void hash_val(wchar_t val)  
{  
    System::Console::WriteLine("hash({0}) = {1}",  
        val, (val * 17 + 31) % 67);  
}  
  
typedef cliext::unary_delegate_noreturn<wchar_t> Mydelegate;  
int main()  
{  
    Mydelegate^ myhash = gcnew Mydelegate(&hash_val);  
  
    myhash(L'a');  
    myhash(L'b');  
    return (0);  
}
```

```
hash(a) = 5  
hash(b) = 22
```

unary_negate (STL/CLR)

此樣板類別描述的仿函數，在呼叫時，會傳回其預存單一引數仿函數的邏輯 NOT。您可以使用它來根據其預存仿函數來指定函式物件。

語法

```
template<typename Fun>  
ref class unary_negate  
{ // wrap operator()  
public:  
    typedef Fun stored_function_type;  
    typedef typename Fun::argument_type argument_type;  
    typedef bool result_type;  
    typedef Microsoft::VisualC::StlClr::UnaryDelegate<  
        argument_type, result_type>  
        delegate_type;  
  
    unary_negate(Fun% functor);  
    unary_negate(unary_negate<Fun>% right);  
  
    result_type operator()(argument_type left);  
    operator delegate_type^();  
};
```

參數

好玩

預存仿函數的類型。

成員函數

<table border="1"> <tr> <td>argument_type</td><td>仿函數引數的類型。</td></tr> <tr> <td>delegate_type</td><td>泛型委派的型別。</td></tr> <tr> <td>result_type</td><td>仿函數結果的類型。</td></tr> </table> <p>MEMBER</p> <table border="1"> <tr> <td>unary_negate</td><td>結構仿函數。</td></tr> </table>	argument_type	仿函數引數的類型。	delegate_type	泛型委派的型別。	result_type	仿函數結果的類型。	unary_negate	結構仿函數。	<table border="1"> <tr> <td>operator()</td><td>計算所需的函數。</td></tr> <tr> <td>delegate_type ^</td><td>將仿函數轉換為委派。</td></tr> </table>	operator()	計算所需的函數。	delegate_type ^	將仿函數轉換為委派。
argument_type	仿函數引數的類型。												
delegate_type	泛型委派的型別。												
result_type	仿函數結果的類型。												
unary_negate	結構仿函數。												
operator()	計算所需的函數。												
delegate_type ^	將仿函數轉換為委派。												

備註

此樣板類別描述儲存另一個引數仿函數的單一引數仿函數。它會定義成員運算子，`operator()` 如此一來，當物件被呼叫為函式時，它就會傳回使用引數呼叫之預存仿函數的邏輯 NOT。

您也可以將物件傳遞為其型別為的函式引數 `delegate_type^`，而且會適當地進行轉換。

範例

```
// cliext_unary_negate.cpp
// compile with: /clr
#include <cliext/algorithm>
#include <cliext/functional>
#include <cliext/vector>

typedef cliext::vector<int> Myvector;
int main()
{
    Myvector c1;
    c1.push_back(4);
    c1.push_back(0);
    Myvector c3(2, 0);

    // display initial contents " 4 0"
    for each (int elem in c1)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display
    cliext::logical_not<int> not_op;

    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::unary_negate<cliext::logical_not<int> >(not_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();

    // transform and display with function
    cliext::transform(c1.begin(), c1.begin() + 2, c3.begin(),
        cliext::not1(not_op));
    for each (int elem in c3)
        System::Console::Write(" {0}", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
4 0
1 0
1 0
```

hash_map (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `hash_map` 來管理一連串的專案做為雜湊表、每個資料表專案儲存雙向連結的節點清單，以及每個節點儲存一個元素。專案是由索引鍵(用來排序序列)和對應值所組成。

在下列描述中，與 `GValue` 相同：

```
Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>
```

其中：

`GKey` 與相同 `Key`，除非後者是 ref 型別，在這種情況下，它是 `Key^`

`GMapped` 與相同 `Mapped`，除非後者是 ref 型別，在這種情況下，它是 `Mapped^`

語法

```
template<typename Key,
         typename Mapped>
ref class hash_map
{
    public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    System::Collections::Generic::IDictionary<Gkey, GMapped>,
    Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

映射

受控制序列中元素的其他元件類型。

需求

標頭：`<cliext/hash_map>`

命名空間：`cliext`

宣告

宣告	說明
<code>hash_map::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>hash_map::const_reference (STL/CLR)</code>	項目的常數參考類型。

hash_map::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
hash_map::difference_type (STL/CLR)	(的類型可能簽署兩個專案之間的) 距離。
hash_map::generic_container (STL/CLR)	容器的泛型介面型別。
hash_map::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
hash_map::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
hash_map::generic_value (STL/CLR)	容器之泛型介面的元素類型。
hash_map::hasher (STL/CLR)	索引鍵的雜湊委派。
hash_map::iterator (STL/CLR)	受控制序列之迭代器的類型。
hash_map::key_compare (STL/CLR)	兩個索引鍵的排序委派。
hash_map::key_type (STL/CLR)	排序索引鍵的類型。
hash_map::mapped_type (STL/CLR)	與每個索引鍵相關聯之對應值的型別。
hash_map::reference (STL/CLR)	項目的參考類型。
hash_map::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
hash_map::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
hash_map::value_compare (STL/CLR)	兩個元素值的排序委派。
hash_map::value_type (STL/CLR)	項目的類型。

hash_map::begin (STL/CLR)	指定受控制序列的開頭。
hash_map::bucket_count (STL/CLR)	計算值區的數目。
hash_map::clear (STL/CLR)	移除所有項目。
hash_map::count (STL/CLR)	計算符合指定索引鍵的元素。
hash_map::empty (STL/CLR)	測試項目是否存在。
hash_map::end (STL/CLR)	指定受控制序列的結尾。
hash_map::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
hash_map::erase (STL/CLR)	移除位於指定位置的項目。

hash_map::find (STL/CLR)	尋找符合指定之索引鍵的元素。
hash_map::hash_delegate (STL/CLR)	複製索引鍵的雜湊委派。
hash_map::hash_map (STL/CLR)	建構容器物件。
hash_map::insert (STL/CLR)	加入項目。
hash_map::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。
hash_map::load_factor (STL/CLR)	計算每個值區的平均項目數。
hash_map::lower_bound (STL/CLR)	尋找符合指定索引鍵的範圍開頭。
hash_map::make_value (STL/CLR)	結構值物件。
hash_map::max_load_factor (STL/CLR)	取得或設定每個 Bucket 最大項目數。
hash_map::rbegin (STL/CLR)	指定反向受控制序列的開頭。
hash_map::rehash (STL/CLR)	重建雜湊資料表。
hash_map::rend (STL/CLR)	指定反向受控制序列的結尾。
hash_map::size (STL/CLR)	計算元素的數目。
hash_map::swap (STL/CLR)	交換兩個容器的內容。
hash_map::to_array (STL/CLR)	將受控制序列複製到新的陣列。
hash_map::upper_bound (STL/CLR)	尋找符合指定索引鍵的範圍結尾。
hash_map::value_comp (STL/CLR)	針對兩個元素值複製順序委派。
hash_map::operator= (STL/CLR)	取代受控制的序列。
hash_map::operator(STL/CLR)	將索引鍵對應至其相關聯的對應值。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。

<code>IEnumerable<T></code>	透過具類型的元素排序。
<code>ICollection<T></code>	維護具類型的元素群組。
<code>IDictionary< TKey, TValue ></code>	維護 {key, value} 組的群組。
<code>IHash<索引鍵, 值></code>	維護泛型容器。

備註

物件會在雙向連結清單中，配置並釋放它所控制之序列的儲存體，以作為個別節點。為了加速存取，物件也會在雜湊表) 的清單中維護不同長度的指標陣列，(雜湊表，有效地將整個清單視為清單子或值區的序列來管理。它會藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式，將專案插入至值區，以保持排序。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `hash_set:: key_compare` 類型的預存委派物件來排序每個值區，([STL/CLR](#))。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<=(key_type, key_type)`。

您可以藉由呼叫成員函式 `hash_set:: key_comp (STL/CLR)` 來存取儲存的委派物件 `()`。這類委派物件必須在 `hash_set:: key_type (STL/CLR)` 類型的索引鍵之間定義對等順序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(X, Y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(X, Y) && key_comp()(Y, X)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

任何行為類似 `operator<=(key_type, key_type)` `operator>=(key_type, key_type)` 或 `operator==(key_type, key_type)` 定義 equivalent 順序的排序規則。

請注意，容器只會確保其索引鍵具有對等順序的專案 (，以及相同整數值) 的雜湊在值區中相鄰。不同于樣板類別 `hash_multimap (STL/CLR)`，樣板類別的物件 `hash_map` 可確保所有元素的索引鍵都是唯一的。(沒有任何兩個索引鍵具有對等的排序。)

物件會藉由呼叫 `hash_set:: hasher (STL/CLR)` 類型的預存委派物件，判斷哪些值區應包含指定的排序索引鍵。您可以藉由呼叫成員函式 `hash_set:: hash_delegate (STL/CLR) ()` 取得相依于索引鍵值的整數值，來存取這個儲存的物件。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，則預設值為函數

`System::Object::hash_value(key_type)`。這表示對於任何金鑰 `x` 和 `y`：

`hash_delegate()(X)` 每次呼叫時，都會傳回相同的整數結果。

如果 `x` 和 `y` 具有對等順序，則應該傳回與 `hash_delegate()(X)` 相同的整數結果 `hash_delegate()(Y)`。

每個元素都包含個別的索引鍵和對應的值。順序的表示方式，可讓您查閱、插入和移除任意專案，而這些作業與序列中的專案數目無關，(常數時間) --至少在案例中的最大值。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

但是，如果雜湊值未一致地散發，雜湊表就可以進行退化。在最極端的情況下，雜湊函數一律會傳回相同的值(查閱、插入和移除)，與序列中的專案數目成正比 (線性時間)。容器會致力於選擇合理的雜湊函式、平均值區大小，以及雜湊表大小 (值區的總數)，但您可以覆寫任何或所有的選項。例如，請參閱函式 `hash_set:: max_load_factor (stl/clr)` 和 `hash_set:: rehash (stl/clr)`。

`Hash_map` 支援雙向反覆運算器，這表示您可以使用反覆運算器逐步執行連續的元素，以指定受控制序列中的元素。特殊的前端節點對應至 `hash_map:: end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將 `hash_map` 反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考指定其數位位置的 hash_map 專案，這需要隨機存取反覆運算器。

Hash_map 反覆運算器會將控制碼儲存至其相關聯的 hash_map 節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。Hash_map 反覆運算器會維持有效，只要其相關聯的 hash_map 節點與某些 hash_map 相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

hash_map:: begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_map_begin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_map::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]
```

hash_map:: bucket_count (STL/CLR)

計算值區的數目。

語法

```
int bucket_count();
```

備註

成員函式會傳回目前的 bucket 數目。您可以使用它來判斷雜湊表的大小。

範例

```
// cliext_hash_map_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25
```

hash_map:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫[hash_map:: erase \(stl/clr\)](#) ([hash_map:: begin \(stl/clr\)](#) (), [hash_map:: end \(stl/clr\)](#) ())。您可以使用它來確保受控制的序列是空的。

範例

```
// cliext_hash_map_clear.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

hash_map:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數雙向反覆運算器。

範例

```
// cliext_hash_map_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_map:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_hash_map_const_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        {   // get a const reference to an element
            Myhash_map::const_reference cref = *cit;
            System::Console::Write("[{0} {1}] ", cref->first, cref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

hash_map:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```

// cliext_hash_map_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_map::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[c 3] [b 2] [a 1]
```

hash_map:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引 鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_hash_map_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

hash_map::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```
// cliext_hash_map_difference_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_map::difference_type diff = 0;
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_map::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3
```

hash_map::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[hash_map::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 hash_map 是否為空白。

範例

```
// cliext_hash_map_empty.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        System::Console::WriteLine("size() = {0}", c1.size());
        System::Console::WriteLine("empty() = {0}", c1.empty());

        // clear the container and reinspect
        c1.clear();
        System::Console::WriteLine("size() = {0}", c1.size());
        System::Console::WriteLine("empty() = {0}", c1.empty());
        return (0);
    }
}
```

```
[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True
```

hash_map:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_hash_map_end.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // inspect last two items
        Myhash_map::iterator it = c1.end();
        --it;
        --it;
        System::Console::WriteLine("*-- --end() = [{0} {1}]",
            it->first, it->second);
        ++it;
        System::Console::WriteLine("*--end() = [{0} {1}]",
            it->first, it->second);
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
*-- --end() = [b 2]
*--end() = [c 3]

```

hash_map::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(lower_bound(key), upper_bound(key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_hash_map_equal_range.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
typedef Myhash_map::pair_iter iter Pairii;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

hash_map:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 `where` 所指向之受控制序列的元素，並傳回反覆運算器，指定移除專案之後的第一個元素，或 `hash_map::end (STL/CLR)` (`()` 如果沒有這類專案存在)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍 `[,)` 中受控制序列的元素，`first` `last` 並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或 `end()` 如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引 鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_hash_map_erase.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    cliext::hash_map<wchar_t, int> c1;
    c1.insert(cliext::hash_map<wchar_t, int>::make_value(L'a', 1));
    c1.insert(cliext::hash_map<wchar_t, int>::make_value(L'b', 2));
    c1.insert(cliext::hash_map<wchar_t, int>::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::hash_map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::hash_map<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::hash_map<wchar_t, int>::make_value(L'd', 4));
    c1.insert(cliext::hash_map<wchar_t, int>::make_value(L'e', 5));
    for each (cliext::hash_map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

hash_map:: find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[hash_map:: end \(STL/CLR\)](#) ()。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_hash_map_find.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Myhash_map::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

hash_map:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
    generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```
// cliext_hash_map_generic_container.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_map::generic_container^ gc1 = %c1;
        for each (Myhash_map::value_type elem in gc1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // modify generic and display original
        gc1->insert(Myhash_map::make_value(L'd', 4));
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // modify original and display generic
        c1.insert(Myhash_map::make_value(L'e', 5));
        for each (Myhash_map::value_type elem in gc1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

hash_map:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```
// cliext_hash_map_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_map::generic_container^ gc1 = %c1;
    for each (Myhash_map::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_map::generic_iterator gicit = gc1->begin();
    Myhash_map::generic_value gcval = *gicit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

hash_map:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
generic_reverse_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```
// cliext_hash_map_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/hash_map>  
  
typedef cliext::hash_map<wchar_t, int> Myhash_map;  
int main()  
{  
    Myhash_map c1;  
    c1.insert(Myhash_map::make_value(L'a', 1));  
    c1.insert(Myhash_map::make_value(L'b', 2));  
    c1.insert(Myhash_map::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Myhash_map::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Myhash_map::generic_container^ gc1 = %c1;  
    for each (Myhash_map::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Myhash_map::generic_reverse_iterator gcit = gc1->rbegin();  
    Myhash_map::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

hash_map:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_hash_map_generic_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_map::generic_container^ gc1 = %c1;
        for each (Myhash_map::value_type elem in gc1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // get an element and display it
        Myhash_map::generic_iterator gcit = gc1->begin();
        Myhash_map::generic_value gcval = *gcit;
        System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

hash_map:: hash_delegate (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```

hasher^ hash_delegate();

```

備註

成員函式會傳回用來將索引鍵值轉換為整數的委派。您可以使用它來雜湊索引鍵。

範例

```
// cliext_hash_map_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        Myhash_map::hasher^ myhash = c1.hash_delegate();

        System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
        System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
        return (0);
    }
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_map::hash_map (STL/CLR)

建構容器物件。

語法

```
hash_map();
explicit hash_map(key_compare^ pred);
hash_map(key_compare^ pred, hasher^ hashfn);
hash_map(hash_map<Key, Mapped>% right);
hash_map(hash_map<Key, Mapped>^ right);
template<typename InIter>
    hash_map(hash_map<InIter first, InIter last>);
template<typename InIter>
    hash_map(InIter first, InIter last,
            key_compare^ pred);
template<typename InIter>
    hash_map(InIter first, InIter last,
            key_compare^ pred, hasher^ hashfn);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right,
        key_compare^ pred);
hash_map(System::Collections::Generic::IEnumerable<GValue>^ right,
        key_compare^ pred, hasher^ hashfn);
```

參數

first

要插入的範圍開頭。

hashfn

將索引鍵對應至值區的雜湊函數。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
hash_map();
```

使用預設的順序述詞 `key_compare()` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及預設順序述詞和雜湊函數。

函數：

```
explicit hash_map(key_compare^ pred);
```

使用順序述詞 `pred` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_map(key_compare^ pred, hasher^ hashfn);
```

使用順序述詞 `pred`，以及雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和雜湊函數。

函數：

```
hash_map(hash_map<Key, Mapped>% right);
```

使用序列 [`right.begin()`、`right.end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_map` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_map(hash_map<Key, Mapped>^ right);
```

使用序列 [`right->begin()`、`right->end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_map` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_map(InIter first, InIter last);
```

使用序列 [`first`、`last`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它，將受控制的序列設為另一個序列的複本，並使用預設順序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_map(InIter first, InIter last, key_compare^ pred);
```

使用) 順序述詞 `first` `last` `pred` 和預設雜湊函數，初始化受控制的序列。您可以使用它，以指定的順序述詞和預設雜湊函數，讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> hash_map(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

使用) 順序述詞 `first` `last` `pred` 和雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它，以指定的順序述詞和雜湊函式，讓受控制的序列成為另一個序列的複本。

函數：

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right);
```

使用列舉值 右邊指定的序列、預設順序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred* 和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred*，以及雜湊函數 *hashfn*，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和雜湊函數。

範例

```
// cliext_hash_map_construct.cpp
// compile with: /clr
#include <cliext/hash_map>

int myfun(wchar_t key)
{ // hash a key
    return (key ^ 0xdeadbeef);
}

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
// construct an empty container
    Myhash_map c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an ordering rule
    Myhash_map c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an ordering rule and hash function
    Myhash_map c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_map::hasher(&myfun));
    System::Console::WriteLine("size() = {0}", c2h.size());

    c2h.insert(c1.begin(), c1.end());
    for each (Myhash_map::value_type elem in c2h)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine();

    // construct with an iterator range
    Myhash_map c3(c1.begin(), c1.end());
    for each (Myhash_map::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an iterator range and an ordering rule
    Myhash_map c4(c1.begin(), c1.end(),
```

```

        cliext::greater_equal<wchar_t>());
for each (Myhash_map::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_map c4h(c1.begin(), c1.end(),
               cliext::greater_equal<wchar_t>(),
               gcnew Myhash_map::hasher(&myfun));
for each (Myhash_map::value_type elem in c4h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_map c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3);
for each (Myhash_map::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_map c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Myhash_map::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_map c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_map::value_type>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_map::hasher(&myfun));
for each (Myhash_map::value_type elem in c6h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct by copying another container
Myhash_map c7(c4);
for each (Myhash_map::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Myhash_map c8(%c3);
for each (Myhash_map::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

hash_map:: hasher (STL/CLR)

索引鍵的雜湊委派。

語法

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
hasher;
```

備註

此類型說明將索引鍵值轉換為整數的委派。

範例

```
// cliext_hash_map_hasher.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_map:: insert (STL/CLR)

加入項目。

語法

```
client::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會致力於插入具有值的元素 `val`，並傳回一對值 `x`。如果 `x.second` 為 `true`，會 `x.first` 指定新插入的專案，否則會 `x.first` 指定具有對等順序的專案，而該專案已存在，且不會插入新的元素。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 `va` 的元素，並使用 `where` 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [`first` , `last`) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_hash_map_insert.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
typedef Myhash_map::pair_iter_bool Pairib;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 =
        c1.insert(Myhash_map::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    pair1 = c1.insert(Myhash_map::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    Myhash_map::iterator it =
        c1.insert(c1.begin(), Myhash_map::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_map c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_map c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
            IEnumerable<Myhash_map::value_type>)c1);
    for each (Myhash_map::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24] True
insert([L'b' 2]) = [b 2] False
[a 1] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [c 3] [x 24]
[a 1] [b 2] [c 3] [x 24] [y 25]
```

hash_map:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_hash_map_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_map:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_hash_map_key_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    Myhash_map::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_map c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_map:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_hash_map_key_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        Myhash_map::key_compare^ kcomp = c1.key_comp();

        System::Console::WriteLine("compare(L'a', L'a') = {0}",
            kcomp(L'a', L'a'));
        System::Console::WriteLine("compare(L'a', L'b') = {0}",
            kcomp(L'a', L'b'));
        System::Console::WriteLine("compare(L'b', L'a') = {0}",
            kcomp(L'b', L'a'));
        System::Console::WriteLine();

        // test a different ordering rule
        Myhash_map c2 = cliext::greater<wchar_t>();
        kcomp = c2.key_comp();

        System::Console::WriteLine("compare(L'a', L'a') = {0}",
            kcomp(L'a', L'a'));
        System::Console::WriteLine("compare(L'a', L'b') = {0}",
            kcomp(L'a', L'b'));
        System::Console::WriteLine("compare(L'b', L'a') = {0}",
            kcomp(L'b', L'a'));
        return (0);
    }
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_map:: key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_hash_map_key_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_map::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_map::load_factor (STL/CLR)

計算每個值區的平均項目數。

語法

```
float load_factor();
```

備註

成員函式會傳回 `(float) hash_map::size (stl/clr) () / hash_map::bucket_count (stl/clr) ()`。您可以使用它來判斷平均 bucket 大小。

範例

```

// cliext_hash_map_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_map::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，此專案 `x` 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案存在，則會傳回`hash_map::end (STL/CLR) ()`；否則會傳回指定的 iterator `x`。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```
// cliext_hash_map_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Myhash_map::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]
```

hash_map:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key, mapped_type mapped);
```

參數

key

要使用的索引鍵值。

已對應

要搜尋的對應值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵 且對應值為 對應的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```
// cliext_hash_map_make_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_map:: mapped_type (STL/CLR)

與每個索引鍵關聯的對應值類型。

語法

```
typedef Mapped mapped_type;
```

備註

此類型是樣板參數 `Mapped` 的同義字。

範例

```
// cliext_hash_map_mapped_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in mapped_type object
            Myhash_map::mapped_type val = it->second;

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

1 2 3

hash_map:: max_load_factor (STL/CLR)

取得或設定每個 Bucket 最大項目數。

語法

```
float max_load_factor();
void max_load_factor(float new_factor);
```

參數

new_factor

要儲存的新最大載入因數。

備註

第一個成員函式會傳回目前儲存的最大載入因數。您可以使用它來判斷平均 bucket 大小上限。

第二個成員函式會以 *new_factor*取代儲存區的最大載入因數。在後續的插入之前，不會發生自動重新雜湊。

範例

```

// cliext_hash_map_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_map:: operator = (STL/CLR)

取代受控制的序列。

語法

```
hash_map<Key, Mapped>% operator=(hash_map<Key, Mapped>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將 `右移` 至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_hash_map_operator_as.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_map c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

hash_map:: operator (STL/CLR)

將索引鍵對應至其相關聯的對應值。

語法

```
mapped_type operator[](key_type key);
```

參數

`key`

要搜尋的索引鍵值。

備註

成員函式會致力於尋找對索引鍵具有對等排序的元素。如果找到一個，就會傳回相關聯的對應值；否則，它會插入 `value_type(key, mapped_type())` 並傳回關聯的（預設）對應值。您可以使用它來查詢對應的值，並指定其相關聯的索引鍵，或者，如果找不到任何索引鍵，則確定該索引鍵的專案是否存在。

範例

```

// cliext_hash_map_operator_sub.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        System::Console::WriteLine("c1[{0}] = {1}",
            L'A', c1[L'A']);
        System::Console::WriteLine("c1[{0}] = {1}",
            L'b', c1[L'b']);

        // redisplay altered contents
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // alter mapped values and redisplay
        c1[L'A'] = 10;
        c1[L'c'] = 13;
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
c1[A] = 0
c1[b] = 2
[a 1] [A 0] [b 2] [c 3]
[a 1] [A 10] [b 2] [c 13]

```

hash_map:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_hash_map_rbegin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // inspect first two items in reversed sequence
        Myhash_map::reverse_iterator rit = c1.rbegin();
        System::Console::WriteLine("*rbegin() = [{0} {1}]",
            rit->first, rit->second);
        ++rit;
        System::Console::WriteLine("*++rbegin() = [{0} {1}]",
            rit->first, rit->second);
    }
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]
```

hash_map:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_hash_map_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_map::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
    {
        // get a reference to an element
        Myhash_map::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_map:: rehash (STL/CLR)

重建雜湊資料表。

語法

```
void rehash();
```

備註

成員函式會重建雜湊表，以確保`hash_map:: load_factor (stl/clr) () <= hash_map:: max_load_factor (stl/clr)`。否則，雜湊表只會在插入之後視需要增加大小。(不會自動縮減大小。) 您使用它來調整雜湊表的大小。

範例

```

// cliext_hash_map_rehash.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1 = gcnew Myhash_map;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_map:: rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 `Iterator` 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_map_rend.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_map::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine("*- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("*-rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*- --rend() = [b 2]
*-rend() = [a 1]
```

hash_map:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_hash_map_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_map::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

hash_map::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[hash_map::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_hash_map_size.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // clear the container and reinspect
        c1.clear();
        System::Console::WriteLine("size() = {0} after clearing", c1.size());

        // add elements and clear again
        c1.insert(Myhash_map::make_value(L'd', 4));
        c1.insert(Myhash_map::make_value(L'e', 5));
        System::Console::WriteLine("size() = {0} after adding 2", c1.size());
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

hash_map:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_hash_map_size_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_map::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_map::size_type diff = 0;
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```
[a 1] [b 2] [c 3]
end()-begin() = 3
```

hash_map:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(hash_map<Key, Mapped>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_hash_map_swap.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // construct another container with repetition of values
        Myhash_map c2;
        c2.insert(Myhash_map::make_value(L'd', 4));
        c2.insert(Myhash_map::make_value(L'e', 5));
        c2.insert(Myhash_map::make_value(L'f', 6));
        for each (Myhash_map::value_type elem in c2)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // swap and redisplay
        c1.swap(c2);
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        for each (Myhash_map::value_type elem in c2)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

hash_map:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_hash_map_to_array.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // copy the container and modify it
        cli::array<Myhash_map::value_type>^ a1 = c1.to_array();

        c1.insert(Myhash_map::make_value(L'd', 4));
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // display the earlier array copy
        for each (Myhash_map::value_type elem in a1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}

```

```
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]
```

hash_map::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```
iterator upper_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，此專案 *x* 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 [hash_map::END \(STL/CLR\) \(\)](#)；否則會傳回反覆運算器，指定超過的第一個元素 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_hash_map_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        c1.insert(Myhash_map::make_value(L'a', 1));
        c1.insert(Myhash_map::make_value(L'b', 2));
        c1.insert(Myhash_map::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_map::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
            c1.upper_bound(L'x') == c1.end());

        Myhash_map::iterator it = c1.upper_bound(L'a');
        System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
            it->first, it->second);
        it = c1.upper_bound(L'b');
        System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
            it->first, it->second);
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

hash_map::value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```

value_compare^ value_comp();

```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```
// cliext_hash_map_value_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        Myhash_map::value_compare^ kcomp = c1.value_comp();

        System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
            kcomp(Myhash_map::make_value(L'a', 1),
                Myhash_map::make_value(L'a', 1)));
        System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
            kcomp(Myhash_map::make_value(L'a', 1),
                Myhash_map::make_value(L'b', 2)));
        System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
            kcomp(Myhash_map::make_value(L'b', 2),
                Myhash_map::make_value(L'a', 1)));
        System::Console::WriteLine();
        return (0);
    }
}
```

```
compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

hash_map::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```
Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;
```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```
// cliext_hash_map_value_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    {
        Myhash_map c1;
        Myhash_map::value_compare^ kcomp = c1.value_comp();

        System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
            kcomp(Myhash_map::make_value(L'a', 1),
                Myhash_map::make_value(L'a', 1)));
        System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
            kcomp(Myhash_map::make_value(L'a', 1),
                Myhash_map::make_value(L'b', 2)));
        System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
            kcomp(Myhash_map::make_value(L'b', 2),
                Myhash_map::make_value(L'a', 1)));
        System::Console::WriteLine();
        return (0);
    }
}
```

```
compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = False
compare([L'b', 2], [L'a', 1]) = False
```

hash_map::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_hash_map_value_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_map;
int main()
{
    Myhash_map c1;
    c1.insert(Myhash_map::make_value(L'a', 1));
    c1.insert(Myhash_map::make_value(L'b', 2));
    c1.insert(Myhash_map::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Myhash_map::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in value_type object
            Myhash_map::value_type val = *it;
            System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multimap (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `hash_multimap` 來管理一連串的專案做為雜湊表、每個資料表專案儲存雙向連結的節點清單，以及每個節點儲存一個元素。專案是由索引鍵(用來排序序列)和對應值所組成。

在下列描述中，與 `GValue` 相同：

`Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>`

其中：

`GKey` 與索引 鍵相同，除非後者是 ref 型別，在這種情況下，它是 `Key^`

`GMapped` 與 對應相同，除非後者是 ref 型別，在這種情況下，它是 `Mapped^`

語法

```
template<typename Key,
         typename Mapped>
ref class hash_multimap
:   public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ ..... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

映射

受控制序列中元素的其他元件類型。

需求

標頭：`<cliext/hash_map>`

命名空間：`cliext`

宣告

宣告	說明
<code>hash_multimap::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>hash_multimap::const_reference (STL/CLR)</code>	項目的常數參考類型。

hash_multimap::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
hash_multimap::difference_type (STL/CLR)	(的類型可能簽署兩個專案之間的) 距離。
hash_multimap::generic_container (STL/CLR)	容器的泛型介面型別。
hash_multimap::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
hash_multimap::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
hash_multimap::generic_value (STL/CLR)	容器之泛型介面的元素類型。
hash_multimap::hasher (STL/CLR)	索引鍵的雜湊委派。
hash_multimap::iterator (STL/CLR)	受控制序列之迭代器的類型。
hash_multimap::key_compare (STL/CLR)	兩個索引鍵的排序委派。
hash_multimap::key_type (STL/CLR)	排序索引鍵的類型。
hash_multimap::mapped_type (STL/CLR)	與每個索引鍵相關聯之對應值的型別。
hash_multimap::reference (STL/CLR)	項目的參考類型。
hash_multimap::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
hash_multimap::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
hash_multimap::value_compare (STL/CLR)	兩個元素值的排序委派。
hash_multimap::value_type (STL/CLR)	項目的類型。

hash_multimap::begin (STL/CLR)	指定受控制序列的開頭。
hash_multimap::bucket_count (STL/CLR)	計算值區的數目。
hash_multimap::clear (STL/CLR)	移除所有項目。
hash_multimap::count (STL/CLR)	計算符合指定索引鍵的元素。
hash_multimap::empty (STL/CLR)	測試項目是否存在。
hash_multimap::end (STL/CLR)	指定受控制序列的結尾。
hash_multimap::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
hash_multimap::erase (STL/CLR)	移除位於指定位置的項目。

hash_multimap::find (STL/CLR)	尋找符合指定之索引鍵的元素。
hash_multimap::hash_delegate (STL/CLR)	複製索引鍵的雜湊委派。
hash_multimap::hash_multimap (STL/CLR)	建構容器物件。
hash_multimap::insert (STL/CLR)	加入項目。
hash_multimap::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。
hash_multimap::load_factor (STL/CLR)	計算每個值區的平均項目數。
hash_multimap::lower_bound (STL/CLR)	尋找符合指定索引鍵的範圍開頭。
hash_multimap::make_value (STL/CLR)	結構值物件。
hash_multimap::max_load_factor (STL/CLR)	取得或設定每個 Bucket 最大項目數。
hash_multimap::rbegin (STL/CLR)	指定反向受控制序列的開頭。
hash_multimap::rehash (STL/CLR)	重建雜湊資料表。
hash_multimap::rend (STL/CLR)	指定反向受控制序列的結尾。
hash_multimap::size (STL/CLR)	計算元素的數目。
hash_multimap::swap (STL/CLR)	交換兩個容器的內容。
hash_multimap::to_array (STL/CLR)	將受控制序列複製到新的陣列。
hash_multimap::upper_bound (STL/CLR)	尋找符合指定索引鍵的範圍結尾。
hash_multimap::value_comp (STL/CLR)	針對兩個元素值複製順序委派。
hash_multimap::operator= (STL/CLR)	取代受控制的序列。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的元素排序。

<code>ICollection<T></code>	維護具類型的元素群組。
<code>IHash<Key, Value></code>	維護泛型容器。

備註

物件會在雙向連結清單中，配置並釋放它所控制之序列的儲存體，以作為個別節點。為了加速存取，物件也會在雜湊表) 的清單中維護不同長度的指標陣列，(雜湊表，有效地將整個清單視為清單子或值區的序列來管理。它會藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式，將專案插入至值區，以保持排序。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `hash_set:: key_compare` 類型的預存委派物件來排序每個值區，([STL/CLR](#))。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<=(key_type, key_type)`。

您可以藉由呼叫成員函式 `hash_set:: key_comp (STL/CLR)` 來存取儲存的委派物件 `()`。這類委派物件必須在 `hash_set:: key_type (STL/CLR)` 類型的索引鍵之間定義對等順序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y) && key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

任何行為類似 `operator<=(key_type, key_type)` `operator>=(key_type, key_type)` 或 `operator==(key_type, key_type)` 定義 equivalent 順序的排序規則。

請注意，容器只會確保其索引鍵具有對等順序的專案 (以及相同整數值) 的雜湊在值區中相鄰。不同于樣板類別 `hash_map (STL/CLR)`，範本類別的物件不 `hash_multimap` 需要所有元素的索引鍵都是唯一的。(兩個以上的索引鍵可以有對等的順序。)

物件會藉由呼叫 `hash_set:: hasher (STL/CLR)` 類型的預存委派物件，判斷哪些值區應包含指定的排序索引鍵。您可以藉由呼叫成員函式 `hash_set:: hash_delegate (STL/CLR) ()` 取得相依于索引鍵值的整數值，來存取這個儲存的物件。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，則預設值為函數

`System::Object::hash_value(key_type)`。這表示對於任何金鑰 `x` 和 `y`：

`hash_delegate()(x)` 每次呼叫時，都會傳回相同的整數結果。

如果 `x` 和 `y` 具有對等順序，則應該傳回與 `hash_delegate()(x)` 相同的整數結果 `hash_delegate()(y)`。

每個元素都包含個別的索引鍵和對應的值。順序的表示方式，可讓您查閱、插入和移除任意專案，而這些作業與序列中的專案數目無關，(常數時間) --至少在案例中的最大值。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

但是，如果雜湊值未一致地散發，雜湊表就可以進行退化。在最極端的情況下，雜湊函數一律會傳回相同的值(查閱、插入和移除)，與序列中的專案數目成正比(線性時間)。容器會致力於選擇合理的雜湊函式、平均值區大小，以及雜湊表大小(值區的總數)，但您可以覆寫任何或所有的選項。例如，請參閱函式 `hash_set:: max_load_factor (stl/clr)` 和 `hash_set:: rehash (stl/clr)`。

`Hash_multimap` 支援雙向反覆運算器，這表示您可以使用反覆運算器逐步執行連續的元素，以指定受控制序列中的元素。特殊的前端節點對應至 `hash_multimap:: end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將 `hash_multimap` 反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考指定其數位位置的 `hash_multimap` 專案，這需要隨機存取反覆運算器。

`Hash_multimap` 反覆運算器會將控制碼儲存至其相關聯的 `hash_multimap` 節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。`Hash_multimap` 反覆運算器會維持有效，只要其相關聯

的 hash_multimap 節點與某些 hash_multimap 相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的 容器不會摧毀其元素。

成員

hash_multimap::begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_multimap_begin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multimap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]
```

hash_multimap::bucket_count (STL/CLR)

計算值區的數目。

語法

```
int bucket_count();
```

備註

成員函式會傳回目前的 bucket 數目。您可以使用它來判斷雜湊表的大小。

範例

```
// cliext_hash_multimap_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25
```

hash_multimap:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫[hash_multimap:: erase \(stl/clr\) \(\)](#) [hash_multimap:: begin \(stl/clr\) \(\)](#), [hash_multimap:: end \(stl/clr\) \(\)](#)。您可以使用它來確保受控制的序列是空的。

範例

```
// cliext_hash_multimap_clear.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

hash_multimap:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數雙向反覆運算器。

範例

```
// cliext_hash_multimap_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multimap:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_hash_multimap_const_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myhash_multimap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

hash_multimap:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```

// cliext_hash_multimap_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_multimap::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[c 3] [b 2] [a 1]
```

hash_multimap::count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引 鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_hash_multimap_count.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

hash_multimap::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```
// cliext_hash_multimap_difference_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multimap::difference_type diff = 0;
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myhash_multimap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3
```

hash_multimap::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[hash_multimap::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 hash_multimap 是否為空白。

範例

```

// cliext_hash_multimap_empty.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

hash_multimap:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_hash_multimap_end.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Myhash_multimap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("--- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("---end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
--- --end() = [b 2]
---end() = [c 3]

```

hash_multimap::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(hash_multimap::lower_bound (stl/clr) (key), hash_multimap::upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_hash_multimap_equal_range.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
typedef Myhash_multimap::pair_iter_iter Pairii;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

hash_multimap:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 *where* 所指向之受控制序列的元素，並傳回反覆運算器，指定移除專案之後的第一個元素，或 `hash_multimap::end (STL/CLR)` (`()` 如果沒有這類專案存在)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍 `[,)` 中受控制序列的元素，`first` `last` 並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或 `end()` 如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引 鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_hash_multimap_erase.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    cliext::hash_multimap<wchar_t, int> c1;
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'a', 1));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'b', 2));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::hash_multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::hash_multimap<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'd', 4));
    c1.insert(cliext::hash_multimap<wchar_t, int>::make_value(L'e', 5));
    for each (cliext::hash_multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

hash_multimap:: find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[hash_multimap:: end \(STL/CLR\)](#) ()。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_hash_multimap_find.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Myhash_multimap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

hash_multimap:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```
// cliext_hash_multimap_generic_container.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multimap::generic_container^ gc1 = %c1;
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Myhash_multimap::make_value(L'd', 4));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Myhash_multimap::make_value(L'e', 5));
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

hash_multimap:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```
// cliext_hash_multimap_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multimap::generic_container^ gc1 = %c1;
    for each (Myhash_multimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multimap::generic_iterator gcit = gc1->begin();
    Myhash_multimap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

hash_multimap:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
generic_reverse_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```
// cliext_hash_multimap_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/hash_map>  
  
typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;  
int main()  
{  
    Myhash_multimap c1;  
    c1.insert(Myhash_multimap::make_value(L'a', 1));  
    c1.insert(Myhash_multimap::make_value(L'b', 2));  
    c1.insert(Myhash_multimap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Myhash_multimap::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Myhash_multimap::generic_container^ gc1 = %c1;  
    for each (Myhash_multimap::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Myhash_multimap::generic_reverse_iterator gcit = gc1->rbegin();  
    Myhash_multimap::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

hash_multimap:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_hash_multimap_generic_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    {
        Myhash_multimap c1;
        c1.insert(Myhash_multimap::make_value(L'a', 1));
        c1.insert(Myhash_multimap::make_value(L'b', 2));
        c1.insert(Myhash_multimap::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_multimap::generic_container^ gc1 = %c1;
        for each (Myhash_multimap::value_type elem in gc1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // get an element and display it
        Myhash_multimap::generic_iterator gcit = gc1->begin();
        Myhash_multimap::generic_value gcval = *gcit;
        System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

hash_multimap:: hash_delegate (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
hasher^ hash_delegate();
```

備註

成員函式會傳回用來將索引鍵值轉換為整數的委派。您可以使用它來雜湊索引鍵。

範例

```

// cliext_hash_multimap_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}

```

```

hash(L'a') = 1616896120
hash(L'b') = 570892832

```

hash_multimap::hash_multimap (STL/CLR)

建構容器物件。

語法

```

hash_multimap();
explicit hash_multimap(key_compare^ pred);
hash_multimap(key_compare^ pred, hasher^ hashfn);
hash_multimap(hash_multimap<Key, Mapped>% right);
hash_multimap(hash_multimap<Key, Mapped>^ right);
template<typename InIter>
    hash_multimap(hash_multimap<InIter first, InIter last>);
template<typename InIter>
    hash_multimap(InIter first, InIter last,
                 key_compare^ pred);
template<typename InIter>
    hash_multimap(InIter first, InIter last,
                 key_compare^ pred, hasher^ hashfn);
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
             key_compare^ pred);
hash_multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
             key_compare^ pred, hasher^ hashfn);

```

參數

first

要插入的範圍開頭。

hashfn

將索引鍵對應至值區的雜湊函數。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
hash_multimap();
```

使用預設的順序述詞 `key_compare()` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及預設順序述詞和雜湊函數。

函數：

```
explicit hash_multimap(key_compare^ pred);
```

使用順序述詞 `pred` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_multimap(key_compare^ pred, hasher^ hashfn);
```

使用順序述詞 `pred`，以及雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和雜湊函數。

函數：

```
hash_multimap(hash_multimap<Key, Mapped>% right);
```

使用序列 [`right.begin()` 、 `right.end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_multimap` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_multimap(hash_multimap<Key, Mapped>^ right);
```

使用序列 [`right->begin()` 、 `right->end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_multimap` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_multimap(InIter first, InIter last);
```

使用序列 [`first` 、 `last`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它，將受控制的序列設為另一個序列的複本，並使用預設順序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_multimap(InIter first, InIter last, key_compare^ pred);
```

使用) 順序述詞 `first` `last` `pred` 和預設雜湊函數，初始化受控制的序列。您可以使用它，以指定的順序述詞和預設雜湊函數，讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> hash_multimap(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

使用) 順序述詞 `first` `last` `pred` 和雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它，以指定的順序述詞和雜湊函式，讓受控制的序列成為另一個序列的複本。

函數：

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right);
```

使用列舉值 右邊指定的序列、預設順序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的

序列成為列舉值所描述之另一個順序的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred* 和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred*，以及雜湊函數 *hashfn*，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和雜湊函數。

範例

```
// cliext_hash_multimap_construct.cpp
// compile with: /clr
#include <cliext/hash_map>

int myfun(wchar_t key)
    { // hash a key
    return (key ^ 0xdeadbeef);
    }

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
// construct an empty container
    Myhash_multimap c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

// construct with an ordering rule
    Myhash_multimap c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

// construct with an ordering rule and hash function
    Myhash_multimap c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_multimap::hasher(&myfun));
    System::Console::WriteLine("size() = {0}", c2h.size());

    c2h.insert(c1.begin(), c1.end());
    for each (Myhash_multimap::value_type elem in c2h)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine();

// construct with an iterator range
    Myhash_multimap c3(c1.begin(), c1.end());
    for each (Myhash_multimap::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
```

```

// construct with an iterator range and an ordering rule
Myhash_multimap c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (Myhash_multimap::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_multimap c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multimap::hasher(&myfun));
for each (Myhash_multimap::value_type elem in c4h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_multimap c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3);
for each (Myhash_multimap::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_multimap c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Myhash_multimap::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_multimap c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myhash_multimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multimap::hasher(&myfun));
for each (Myhash_multimap::value_type elem in c6h)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
System::Console::WriteLine();

// construct by copying another container
Myhash_multimap c7(c4);
for each (Myhash_multimap::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Myhash_multimap c8(%c3);
for each (Myhash_multimap::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

hash_multimap:: hasher (STL/CLR)

索引鍵的雜湊委派。

語法

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

備註

此類型說明將索引鍵值轉換為整數的委派。

範例

```
// cliext_hash_multimap_hasher.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_multimap:: insert (STL/CLR)

加入項目。

語法

```
iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會插入具有值 *va* 的元素，並傳回指定新插入之元素的反覆運算器。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 *va* 的元素，並使用 *where* 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [*first* , *last*) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_hash_multimap_insert.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    {
        Myhash_multimap c1;
        c1.insert(Myhash_multimap::make_value(L'a', 1));
        c1.insert(Myhash_multimap::make_value(L'b', 2));
        c1.insert(Myhash_multimap::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // insert a single value, unique and duplicate
        Myhash_multimap::iterator it =
            c1.insert(Myhash_multimap::make_value(L'x', 24));
        System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}]",
            it->first, it->second);

        it = c1.insert(Myhash_multimap::make_value(L'b', 2));
        System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}]",
            it->first, it->second);

        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // insert a single value with hint
        it = c1.insert(c1.begin(), Myhash_multimap::make_value(L'y', 25));
        System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
            it->first, it->second);
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // insert an iterator range
        Myhash_multimap c2;
        it = c1.end();
        c2.insert(c1.begin(), --it);
        for each (Myhash_multimap::value_type elem in c2)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // insert an enumeration
        Myhash_multimap c3;
        c3.insert( // NOTE: cast is not needed
            (System::Collections::Generic::
                IEnumerable<Myhash_multimap::value_type>)c1);
        for each (Myhash_multimap::value_type elem in c3)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24]
insert([L'b' 2]) = [b 2]
[a 1] [b 2] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
```

hash_multimap::iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_hash_multimap_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multimap::key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_hash_multimap_key_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_multimap:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_hash_multimap_key_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_multimap::key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_hash_multimap_key_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in key_type object
            Myhash_multimap::key_type val = it->first;

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multimap::load_factor (STL/CLR)

計算每個值區的平均項目數。

語法

```
float load_factor();
```

備註

成員函式會傳回 `(float) hash_multimap::size (stl/clr) () / hash_multimap::bucket_count (stl/clr) ()`。您可以使用它來判斷平均 bucket 大小。

範例

```

// cliext_hash_multimap_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multimap::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，此專案 `x` 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案存在，則會傳回`hash_multimap::end (STL/CLR)`(); 否則會傳回指定的 iterator `x`。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```
// cliext_hash_multimap_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Myhash_multimap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]
```

hash_multimap:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key, mapped_type mapped);
```

參數

key

要使用的索引鍵值。

已對應

要搜尋的對應值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵且對應值為 對應的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```
// cliext_hash_multimap_make_value.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multimap:: mapped_type (STL/CLR)

與每個索引鍵關聯的對應值類型。

語法

```
typedef Mapped mapped_type;
```

備註

此類型與 對應的範本參數同義。

範例

```

// cliext_hash_multimap_mapped_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in mapped_type object
            Myhash_multimap::mapped_type val = it->second;

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

1 2 3

hash_multimap:: max_load_factor (STL/CLR)

取得或設定每個 Bucket 最大項目數。

語法

```

float max_load_factor();
void max_load_factor(float new_factor);

```

參數

new_factor

要儲存的新最大載入因數。

備註

第一個成員函式會傳回目前儲存的最大載入因數。您可以使用它來判斷平均 bucket 大小上限。

第二個成員函式會以 *new_factor*取代儲存區的最大載入因數。在後續的插入之前，不會發生自動重新雜湊。

範例

```

// cliext_hash_multimap_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multimap:: operator = (STL/CLR)

取代受控制的序列。

語法

```
hash_multimap<Key, Mapped>% operator=(hash_multimap<Key, Mapped>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將 **右移** 至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 *right* 中受控制序列的複本。

範例

```
// cliext_hash_multimap_operator_as.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_multimap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

hash_multimap::rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_hash_multimap_rbegin.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    {
        Myhash_multimap c1;
        c1.insert(Myhash_multimap::make_value(L'a', 1));
        c1.insert(Myhash_multimap::make_value(L'b', 2));
        c1.insert(Myhash_multimap::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // inspect first two items in reversed sequence
        Myhash_multimap::reverse_iterator rit = c1.rbegin();
        System::Console::WriteLine("*rbegin() = [{0} {1}]",
            rit->first, rit->second);
        ++rit;
        System::Console::WriteLine("*++rbegin() = [{0} {1}]",
            rit->first, rit->second);
        return (0);
    }
}
```

```
[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]
```

hash_multimap:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_hash_multimap_reference.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Myhash_multimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
    {
        // get a reference to an element
        Myhash_multimap::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multimap:: rehash (STL/CLR)

重建雜湊資料表。

語法

```
void rehash();
```

備註

成員函式會重建雜湊表，以確保[hash_multimap:: load_factor \(stl/clr\)](#) $(0 \leq)$ [hash_multimap:: max_load_factor \(stl/clr\)](#)。否則，雜湊表只會在插入之後視需要增加大小。(不會自動縮減大小。) 您使用它來調整雜湊表的大小。

範例

```

// cliext_hash_multimap_rehash.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1 = gcnew Myhash_multimap;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multimap:: rend (STL/CLR)

指定反向受控制序列的结尾。

语法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_multimap_rend.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_multimap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine(" --- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine(" --- --rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
--- --rend() = [b 2]
---rend() = [a 1]
```

hash_multimap:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_hash_multimap_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Myhash_multimap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

hash_multimap::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[hash_multimap::empty \(STL/CLR\)](#) ()。

範例

```

// cliext_hash_multimap_size.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Myhash_multimap::make_value(L'd', 4));
    c1.insert(Myhash_multimap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

hash_multimap::size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_hash_multimap_size_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multimap::size_type diff = 0;
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

hash_multimap:: swap (STL/CLR)

交換兩個容器的內容。

語法

```

void swap(hash_multimap<Key, Mapped>% right);

```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_hash_multimap_swap.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    {
        Myhash_multimap c1;
        c1.insert(Myhash_multimap::make_value(L'a', 1));
        c1.insert(Myhash_multimap::make_value(L'b', 2));
        c1.insert(Myhash_multimap::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // construct another container with repetition of values
        Myhash_multimap c2;
        c2.insert(Myhash_multimap::make_value(L'd', 4));
        c2.insert(Myhash_multimap::make_value(L'e', 5));
        c2.insert(Myhash_multimap::make_value(L'f', 6));
        for each (Myhash_multimap::value_type elem in c2)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        // swap and redisplay
        c1.swap(c2);
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        for each (Myhash_multimap::value_type elem in c2)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

hash_multimap:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<value_type>^ to_array();
```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_hash_multimap_to_array.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Myhash_multimap::value_type>^ a1 = c1.to_array();

    c1.insert(Myhash_multimap::make_value(L'd', 4));
    for each (Myhash_multimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Myhash_multimap::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]
```

hash_multimap::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```
iterator upper_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，此專案 *x* 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 [hash_multimap::END \(STL/CLR\) \(\)](#)；否則會傳回反覆運算器，指定超過的第一個元素 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_hash_multimap_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    {
        Myhash_multimap c1;
        c1.insert(Myhash_multimap::make_value(L'a', 1));
        c1.insert(Myhash_multimap::make_value(L'b', 2));
        c1.insert(Myhash_multimap::make_value(L'c', 3));

        // display contents " [a 1] [b 2] [c 3]"
        for each (Myhash_multimap::value_type elem in c1)
            System::Console::Write("[{0} {1}] ", elem->first, elem->second);
        System::Console::WriteLine();

        System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
            c1.upper_bound(L'x') == c1.end());

        Myhash_multimap::iterator it = c1.upper_bound(L'a');
        System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
            it->first, it->second);
        it = c1.upper_bound(L'b');
        System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
            it->first, it->second);
        return (0);
    }
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

hash_multimap:: value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```
value_compare^ value_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```
// cliext_hash_multimap_value_comp.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'b', 2),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

hash_multimap::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```
Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
value_compare;
```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```
// cliext_hash_multimap_value_compare.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_map<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    Myhash_multimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Myhash_multimap::make_value(L'a', 1),
            Myhash_multimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Myhash_multimap::make_value(L'b', 2),
            Myhash_multimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = True
compare([L'a', 1], [L'b', 2]) = False
compare([L'b', 2], [L'a', 1]) = False
```

hash_multimap::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_hash_multimap_value_type.cpp
// compile with: /clr
#include <cliext/hash_map>

typedef cliext::hash_multimap<wchar_t, int> Myhash_multimap;
int main()
{
    Myhash_multimap c1;
    c1.insert(Myhash_multimap::make_value(L'a', 1));
    c1.insert(Myhash_multimap::make_value(L'b', 2));
    c1.insert(Myhash_multimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Myhash_multimap::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in value_type object
            Myhash_multimap::value_type val = *it;
            System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

hash_multiset (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `hash_multiset` 來管理一連串的專案做為雜湊表、每個資料表專案儲存雙向連結的節點清單，以及每個節點儲存一個元素。每個元素的值會用來作為排序次序的索引鍵。

在下列描述中，與 `GValue` 相同 `GKey`，除非後者是 ref 型別，否則它就會與索引 鍵相同，在這種情況下，則為 `Key^` 。

語法

```
template<typename Key>
ref class hash_multiset
: public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

需求

標頭 : <cliext/hash_set>

命名空間 : cliext

宣告

宣告	說明
<code>hash_multiset::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>hash_multiset::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>hash_multiset::const_reverse_iterator (STL/CLR)</code>	用於受控制序列的常數反向迭代器類型。
<code>hash_multiset::difference_type (STL/CLR)</code>	(的類型可能簽署兩個專案之間的) 距離。
<code>hash_multiset::generic_container (STL/CLR)</code>	容器的泛型介面型別。
<code>hash_multiset::generic_iterator (STL/CLR)</code>	容器之泛型介面的反覆運算器類型。

hash_multiset::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
hash_multiset::generic_value (STL/CLR)	容器之泛型介面的元素類型。
hash_multiset::hasher (STL/CLR)	索引鍵的雜湊委派。
hash_multiset::iterator (STL/CLR)	受控制序列之迭代器的類型。
hash_multiset::key_compare (STL/CLR)	兩個索引鍵的排序委派。
hash_multiset::key_type (STL/CLR)	排序索引鍵的類型。
hash_multiset::reference (STL/CLR)	項目的參考類型。
hash_multiset::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
hash_multiset::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
hash_multiset::value_compare (STL/CLR)	兩個元素值的排序委派。
hash_multiset::value_type (STL/CLR)	項目的類型。

hash_multiset::begin (STL/CLR)	指定受控制序列的開頭。
hash_multiset::bucket_count (STL/CLR)	計算值區的數目。
hash_multiset::clear (STL/CLR)	移除所有項目。
hash_multiset::count (STL/CLR)	計算符合指定索引鍵的元素。
hash_multiset::empty (STL/CLR)	測試項目是否不存在。
hash_multiset::end (STL/CLR)	指定受控制序列的結尾。
hash_multiset::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
hash_multiset::erase (STL/CLR)	移除位於指定位置的項目。
hash_multiset::find (STL/CLR)	尋找符合指定之索引鍵的元素。
hash_multiset::hash_delegate (STL/CLR)	複製索引鍵的雜湊委派。
hash_multiset::hash_multiset (STL/CLR)	建構容器物件。
hash_multiset::insert (STL/CLR)	加入項目。
hash_multiset::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。

hash_multiset::load_factor (STL/CLR)	計算每個值區的平均項目數。
hash_multiset::lower_bound (STL/CLR)	尋找符合指定索引鍵的範圍開頭。
hash_multiset::make_value (STL/CLR)	結構值物件。
hash_multiset::max_load_factor (STL/CLR)	取得或設定每個 Bucket 最大項目數。
hash_multiset::rbegin (STL/CLR)	指定反向受控制序列的開頭。
hash_multiset::rehash (STL/CLR)	重建雜湊資料表。
hash_multiset::rend (STL/CLR)	指定反向受控制序列的結尾。
hash_multiset::size (STL/CLR)	計算元素的數目。
hash_multiset::swap (STL/CLR)	交換兩個容器的內容。
hash_multiset::to_array (STL/CLR)	將受控制序列複製到新的陣列。
hash_multiset::upper_bound (STL/CLR)	尋找符合指定索引鍵的範圍結尾。
hash_multiset::value_comp (STL/CLR)	針對兩個元素值複製順序委派。
hash_multiset::operator= (STL/CLR)	取代受控制的序列。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的元素排序。
ICollection<T>	維護具類型的元素群組。
IHash<Key, Value>	維護泛型容器。

備註

物件會在雙向連結清單中，配置並釋放它所控制之序列的儲存體，以作為個別節點。為了加速存取，物件也會在雜湊表) 的清單中維護不同長度的指標陣列，(雜湊表，有效地將整個清單視為清單子或值區的序列來管理。它會藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式，將專案插入至值區，以保持排序。這表

示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `hash_set::key_compare` 類型的預存委派物件來排序每個值區，([STL/CLR](#))。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<=(key_type, key_type)`。

您可以藉由呼叫成員函式`hash_set::key_comp (STL/CLR)`來存取儲存的委派物件 `()`。這類委派物件必須在 `hash_set::key_type (STL/CLR)` 類型的索引鍵之間定義對等順序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y) && key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

任何行為類似 `operator<=(key_type, key_type)` `operator>=(key_type, key_type)` 或 `operator==(key_type, key_type)` 定義 equivalent 順序的排序規則。

請注意，容器只會確保其索引鍵具有對等順序的專案(，以及相同整數值)的雜湊在值區中相鄰。不同于樣板類別 `hash_set (STL/CLR)`，範本類別的物件不 `hash_multiset` 需要所有元素的索引鍵都是唯一的。(兩個以上的索引鍵可以有對等的順序。)

物件會藉由呼叫 `hash_set::hasher (STL/CLR)` 類型的預存委派物件，判斷哪些值區應包含指定的排序索引鍵。您可以藉由呼叫成員函式`hash_set::hash_delegate (STL/CLR) ()` 取得相依于索引鍵值的整數值，來存取這個儲存的物件。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，則預設值為函數

`System::Object::hash_value(key_type)`。這表示對於任何金鑰 `x` 和 `y`：

`hash_delegate()(x)` 每次呼叫時，都會傳回相同的整數結果。

如果 `x` 和 `y` 具有對等順序，則應該傳回與 `hash_delegate()(x)` 相同的整數結果 `hash_delegate()(y)`。

每個元素都可作為索引鍵和值。順序的表示方式，可讓您查閱、插入和移除任意專案，而這些作業與序列中的專案數目無關，(常數時間) --至少在案例中的最大值。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

但是，如果雜湊值未一致地散發，雜湊表就可以進行退化。在最極端的情況下，雜湊函數一律會傳回相同的值(查閱、插入和移除)，與序列中的專案數目成正比(線性時間)。容器會致力於選擇合理的雜湊函式、平均值區大小，以及雜湊表大小(值區的總數)，但您可以覆寫任何或所有的選項。例如，請參閱函式 `hash_set::max_load_factor (stl/clr)` 和 `hash_set::rehash (stl/clr)`。

`Hash_multiset` 支援雙向反覆運算器，這表示您可以使用反覆運算器逐步執行連續的元素，以指定受控制序列中的元素。特殊的前端節點對應至`hash_multiset::end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將 `hash_multiset` 反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考指定其數位位置的 `hash_multiset` 專案，這需要隨機存取反覆運算器。

`Hash_multiset` 反覆運算器會將控制碼儲存至其相關聯的 `hash_multiset` 節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。`Hash_multiset` 反覆運算器會維持有效，只要其相關聯的 `hash_multiset` 節點與某些 `hash_multiset` 相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

`hash_multiset::begin (STL/CLR)`

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_multiset_begin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multiset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
*++begin() = b
```

hash_multiset::bucket_count (STL/CLR)

計算值區的數目。

語法

```
int bucket_count();
```

備註

成員函式會傳回目前的 bucket 數目。您可以使用它來判斷雜湊表的大小。

範例

```

// cliext_hash_multiset_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multiset:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫 `hash_multiset:: erase (stl/clr) ()`, `hash_multiset:: begin (stl/clr) ()`, `hash_multiset:: end (stl/clr) ()`。您可以使用它來確保受控制的序列是空的。

範例

```
// cliext_hash_multiset_clear.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

hash_multiset:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數雙向反覆運算器。

範例

```
// cliext_hash_multiset_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_multiset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```
// cliext_hash_multiset_const_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_multiset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myhash_multiset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_hash_multiset_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_multiset::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

hash_multiset:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引 鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_hash_multiset_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

hash_multiset::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_hash_multiset_difference_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // compute positive difference
        Myhash_multiset::difference_type diff = 0;
        for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
            ++diff;
        System::Console::WriteLine("end()-begin() = {0}", diff);

        // compute negative difference
        diff = 0;
        for (Myhash_multiset::iterator it = c1.end(); it != c1.begin(); --it)
            --diff;
        System::Console::WriteLine("begin()-end() = {0}", diff);
        return (0);
    }
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

hash_multiset::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[hash_multiset::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 hash_multiset 是否為空白。

範例

```
// cliext_hash_multiset_empty.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        System::Console::WriteLine("size() = {0}", c1.size());
        System::Console::WriteLine("empty() = {0}", c1.empty());

        // clear the container and reinspect
        c1.clear();
        System::Console::WriteLine("size() = {0}", c1.size());
        System::Console::WriteLine("empty() = {0}", c1.empty());
        return (0);
    }
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

hash_multiset::end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_hash_multiset_end.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect last two items
        Myhash_multiset::iterator it = c1.end();
        --it;
        System::Console::WriteLine("!-- --end() = {0}", *--it);
        System::Console::WriteLine("!--end() = {0}", *++it);
        return (0);
    }
}

```

```

a b c
!-- --end() = b
!--end() = c

```

hash_multiset::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(hash_multiset::lower_bound (stl/clr) (key), hash_multiset::upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_hash_multiset_equal_range.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
typedef Myhash_multiset::pair_iter_pairii;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

hash_multiset:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由*where*所指向之受控制序列的元素，並傳回反覆運算器，指定移除專案之後的第一個元

素，或 `hash_multiset::end` (STL/CLR) (() 如果沒有這類專案存在)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍 [,) 中受控制序列的元素，`first` `last` 並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或 `end()` 如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引 鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_hash_multiset_erase.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myhash_multiset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

hash_multiset::find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[hash_multiset:: end \(STL/CLR\)](#)。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_hash_multiset_find.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

hash_multiset:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
    generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_hash_multiset_generic_container.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_multiset::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify generic and display original
        gc1->insert(L'd');
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify original and display generic
        c1.insert(L'e');
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
a b c
a b c d
a b c d e

```

hash_multiset::generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```

// cliext_hash_multiset_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_multiset::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // get an element and display it
        Myhash_multiset::generic_iterator gcit = gc1->begin();
        Myhash_multiset::generic_value gcval = *gcit;
        System::Console::WriteLine("{0} ", gcval);
        return (0);
    }
}

```

```

a b c
a b c
a

```

hash_multiset:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```

// cliext_hash_multiset_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_multiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_multiset::generic_reverse_iterator gcit = gc1->rbegin();
    Myhash_multiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

hash_multiset:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_hash_multiset_generic_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_multiset::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // get an element and display it
        Myhash_multiset::generic_iterator gcit = gc1->begin();
        Myhash_multiset::generic_value gcval = *gcit;
        System::Console::WriteLine("{0} ", gcval);
        return (0);
    }
}

```

```

a b c
a b c
a

```

hash_multiset:: hash_delegate (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```

hasher^ hash_delegate();

```

備註

成員函式會傳回用來將索引鍵值轉換為整數的委派。您可以使用它來雜湊索引鍵。

範例

```
// cliext_hash_multiset_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        Myhash_multiset::hasher^ myhash = c1.hash_delegate();

        System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
        System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
        return (0);
    }
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_multiset::hash_multiset (STL/CLR)

建構容器物件。

語法

```
hash_multiset();
explicit hash_multiset(key_compare^ pred);
hash_multiset(key_compare^ pred, hasher^ hashfn);
hash_multiset(hash_multiset<Key>% right);
hash_multiset(hash_multiset<Key>^ right);
template<typename InIter>
    hash_multiset(InIter first, InIter last);
template<typename InIter>
    hash_multiset(InIter first, InIter last,
                 key_compare^ pred);
template<typename InIter>
    hash_multiset(InIter first, InIter last,
                 key_compare^ pred, hasher^ hashfn);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
             key_compare^ pred);
hash_multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
             key_compare^ pred, hasher^ hashfn);
```

參數

first

要插入的範圍開頭。

hashfn

將索引鍵對應至值區的雜湊函數。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
hash_multiset();
```

使用預設的順序述詞 `key_compare()` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及預設順序述詞和雜湊函數。

函數：

```
explicit hash_multiset(key_compare^ pred);
```

使用順序述詞 `pred` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_multiset(key_compare^ pred, hasher^ hashfn);
```

使用順序述詞 `pred`，以及雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和雜湊函數。

函數：

```
hash_multiset(hash_multiset<Key>% right);
```

使用序列 [`right.begin()` 、 `right.end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_multiset` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_multiset(hash_multiset<Key>^ right);
```

使用序列 [`right->begin()` 、 `right->end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_multiset` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_multiset(InIter first, InIter last);
```

使用序列 [`first` 、 `last`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它，將受控制的序列設為另一個序列的複本，並使用預設順序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_multiset(InIter first, InIter last, key_compare^ pred);
```

使用) 順序述詞 `first` `last` `pred` 和預設雜湊函數，初始化受控制的序列。您可以使用它，以指定的順序述詞和預設雜湊函數，讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> hash_multiset(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

使用) 順序述詞 `first` `last` `pred` 和雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它，以指定的順序述詞和雜湊函式，讓受控制的序列成為另一個序列的複本。

函數：

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right);
```

使用列舉值 右邊指定的序列、預設順序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred* 和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred*，以及雜湊函數 *hashfn*，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和雜湊函數。

範例

```
// cliext_hash_multiset_construct.cpp
// compile with: /clr
#include <cliext/hash_set>

int myfun(wchar_t key)
{ // hash a key
    return (key ^ 0xdeadbeef);
}

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
// construct an empty container
    Myhash_multiset c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule
    Myhash_multiset c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule and hash function
    Myhash_multiset c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_multiset::hasher(&myfun));
    System::Console::WriteLine("size() = {0}", c2h.size());

    c2h.insert(c1.begin(), c1.end());
    for each (wchar_t elem in c2h)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine();

    // construct with an iterator range
    Myhash_multiset c3(c1.begin(), c1.end());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an iterator range and an ordering rule
    Myhash_multiset c4(c1.begin(), c1.end(),
```

```

        cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_multiset c4h(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multiset::hasher(&myfun));
for each (wchar_t elem in c4h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_multiset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_multiset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_multiset c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_multiset::hasher(&myfun));
for each (wchar_t elem in c6h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct from a generic container
Myhash_multiset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myhash_multiset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
a b c
size() = 0
c b a

a b c
a b c
c b a

a b c
a b c
c b a

a b c
a b c
```

hash_multiset:: hasher (STL/CLR)

索引鍵的雜湊委派。

語法

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

備註

此類型說明將索引鍵值轉換為整數的委派。

範例

```
// cliext_hash_multiset_hasher.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_multiset:: insert (STL/CLR)

加入項目。

語法

```
iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會插入具有值 *val* 的元素，並傳回指定新插入之元素的反覆運算器。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 *val* 的元素，並使用 *where* 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [*first* , *last*) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_hash_multiset_insert.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert a single value, unique and duplicate
        System::Console::WriteLine("insert(L'x') = {0}",
            *c1.insert(L'x'));

        System::Console::WriteLine("insert(L'b') = {0}",
            *c1.insert(L'b'));

        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert a single value with hint
        System::Console::WriteLine("insert(begin(), L'y') = {0}",
            *c1.insert(c1.begin(), L'y'));
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert an iterator range
        Myhash_multiset c2;
        Myhash_multiset::iterator it = c1.end();
        c2.insert(c1.begin(), --it);
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert an enumeration
        Myhash_multiset c3;
        c3.insert(   // NOTE: cast is not needed
            (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
        for each (wchar_t elem in c3)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
insert(L'x') = x
insert(L'b') = b
a b b c x
insert(begin(), L'y') = y
a b b c x y
a b b c x
a b b c x y

```

hash_multiset:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_hash_multiset_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_multiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset::key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_hash_multiset_key_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_multiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_multiset:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_hash_multiset_key_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        Myhash_multiset::key_compare^ kcomp = c1.key_comp();

        System::Console::WriteLine("compare(L'a', L'a') = {0}",
            kcomp(L'a', L'a'));
        System::Console::WriteLine("compare(L'a', L'b') = {0}",
            kcomp(L'a', L'b'));
        System::Console::WriteLine("compare(L'b', L'a') = {0}",
            kcomp(L'b', L'a'));
        System::Console::WriteLine();

        // test a different ordering rule
        Myhash_multiset c2 = cliext::greater<wchar_t>();
        kcomp = c2.key_comp();

        System::Console::WriteLine("compare(L'a', L'a') = {0}",
            kcomp(L'a', L'a'));
        System::Console::WriteLine("compare(L'a', L'b') = {0}",
            kcomp(L'a', L'b'));
        System::Console::WriteLine("compare(L'b', L'a') = {0}",
            kcomp(L'b', L'a'));
        return (0);
    }
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_multiset:: key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_hash_multiset_key_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_multiset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset::load_factor (STL/CLR)

計算每個值區的平均項目數。

語法

```
float load_factor();
```

備註

成員函式會傳回 `(float) hash_multiset::size (stl/clr) () / hash_multiset::bucket_count (stl/clr) ()`。您可以使用它來判斷平均 bucket 大小。

範例

```

// cliext_hash_multiset_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multiset::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，此專案 *x* 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案存在，則會傳回 `hash_multiset::end (STL/CLR)` ()；否則會傳回指定的 iterator *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```
// cliext_hash_multiset_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b'));
    return (0);
}
```

```
a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b
```

hash_multiset:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key);
```

參數

key

要使用的索引鍵值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```
// cliext_hash_multiset_make_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(Myhash_multiset::make_value(L'a'));
    c1.insert(Myhash_multiset::make_value(L'b'));
    c1.insert(Myhash_multiset::make_value(L'c'));

    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset::max_load_factor (STL/CLR)

取得或設定每個 Bucket 最大項目數。

語法

```
float max_load_factor();
void max_load_factor(float new_factor);
```

參數

new_factor

要儲存的新最大載入因數。

備註

第一個成員函式會傳回目前儲存的最大載入因數。您可以使用它來判斷平均 bucket 大小上限。

第二個成員函式會以 *new_factor*取代儲存區的最大載入因數。在後續的插入之前，不會發生自動重新雜湊。

範例

```

// cliext_hash_multiset_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multiset:: operator = (STL/CLR)

取代受控制的序列。

語法

```
hash_multiset<Key>% operator=(hash_multiset<Key>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將 `right` 移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_hash_multiset_operator_as.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_multiset c2;
    c2 = c1;
    // display contents " a b c"
    for each (Myhash_multiset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

hash_multiset:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_hash_multiset_rbegin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_multiset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
```

hash_multiset:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_hash_multiset_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_multiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        {   // get a reference to an element
            Myhash_multiset::reference ref = *it;
            System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_multiset:: rehash (STL/CLR)

重建雜湊資料表。

語法

```
void rehash();
```

備註

成員函式會重建雜湊表，以確保[hash_multiset:: load_factor \(stl/clr\) \(\) <= hash_multiset:: max_load_factor \(stl/clr\)](#)。否則，雜湊表只會在插入之後視需要增加大小。(不會自動縮減大小。) 您使用它來調整雜湊表的大小。

範例

```

// cliext_hash_multiset_rehash.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_multiset:: rend (STL/CLR)

指定反向受控制序列的结尾。

语法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_multiset_rend.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_multiset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("*-- --rend() = {0}", *--rit);
    System::Console::WriteLine("*--rend() = {0}", *++rit);
    return (0);
}
```

```
a b c
*-- --rend() = b
*--rend() = a
```

hash_multiset:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_hash_multiset_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c" reversed
        Myhash_multiset::reverse_iterator rit = c1.rbegin();
        for (; rit != c1.rend(); ++rit)
            System::Console::Write("{0} ", *rit);
        System::Console::WriteLine();
    }
    return (0);
}
```

```
c b a
```

hash_multiset::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[hash_multiset::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_hash_multiset_size.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

hash_multiset::size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```
// cliext_hash_multiset_size_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_multiset::size_type diff = 0;
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

hash_multiset:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(hash_multiset<Key>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_hash_multiset_swap.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct another container with repetition of values
        Myhash_multiset c2;
        c2.insert(L'd');
        c2.insert(L'e');
        c2.insert(L'f');
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // swap and redisplay
        c1.swap(c2);
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
d e f
d e f
a b c

```

hash_multiset:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_hash_multiset_to_array.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // copy the container and modify it
        cli::array<wchar_t>^ a1 = c1.to_array();

        c1.insert(L'd');
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // display the earlier array copy
        for each (wchar_t elem in a1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c d
a b c

```

hash_multiset::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```

iterator upper_bound(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，此專案 *x* 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 `hash_multiset::END (STL/CLR) ()`；否則會傳回反覆運算器，指定超過的第一個元素 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_hash_multiset_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    {
        Myhash_multiset c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
            c1.upper_bound(L'x') == c1.end());

        System::Console::WriteLine("*upper_bound(L'a') = {0}",
            *c1.upper_bound(L'a'));
        System::Console::WriteLine("*upper_bound(L'b') = {0}",
            *c1.upper_bound(L'b'));
        return (0);
    }
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

hash_multiset::value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```

value_compare^ value_comp();

```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```
// cliext_hash_multiset_value_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}
```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

hash_multiset::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```
Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;
```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```
// cliext_hash_multiset_value_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    Myhash_multiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}
```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

hash_multiset::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_hash_multiset_value_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_multiset<wchar_t> Myhash_multiset;
int main()
{
    Myhash_multiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myhash_multiset::iterator it = c1.begin(); it != c1.end(); ++it)
        {   // store element in value_type object
            Myhash_multiset::value_type val = *it;

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `hash_set` 來管理一連串的專案做為雜湊表、每個資料表專案儲存雙向連結的節點清單，以及每個節點儲存一個元素。每個元素的值會用來做為排序次序的索引鍵。

在下列描述中，與 `GValue` 相同 `GKey`，除非後者是 ref 型別，否則它就會與索引 鍵相同，在這種情況下，則為 `Key^`。

語法

```
template<typename Key>
ref class hash_set
: public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::IHash<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

需求

標頭 : <cliext/hash_set>

命名空間 : cliext

宣告

宣告	說明
<code>hash_set::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>hash_set::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>hash_set::const_reverse_iterator (STL/CLR)</code>	用於受控制序列的常數反向迭代器類型。
<code>hash_set::difference_type (STL/CLR)</code>	(的類型可能簽署兩個專案之間的) 距離。
<code>hash_set::generic_container (STL/CLR)</code>	容器的泛型介面型別。
<code>hash_set::generic_iterator (STL/CLR)</code>	容器之泛型介面的反覆運算器類型。

hash_set::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
hash_set::generic_value (STL/CLR)	容器之泛型介面的元素類型。
hash_set::hasher (STL/CLR)	索引鍵的雜湊委派。
hash_set::iterator (STL/CLR)	受控制序列之迭代器的類型。
hash_set::key_compare (STL/CLR)	兩個索引鍵的排序委派。
hash_set::key_type (STL/CLR)	排序索引鍵的類型。
hash_set::reference (STL/CLR)	項目的參考類型。
hash_set::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
hash_set::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
hash_set::value_compare (STL/CLR)	兩個元素值的排序委派。
hash_set::value_type (STL/CLR)	項目的類型。

hash_set::begin (STL/CLR)	指定受控制序列的開頭。
hash_set::bucket_count (STL/CLR)	計算值區的數目。
hash_set::clear (STL/CLR)	移除所有項目。
hash_set::count (STL/CLR)	計算符合指定索引鍵的元素。
hash_set::empty (STL/CLR)	測試項目是否不存在。
hash_set::end (STL/CLR)	指定受控制序列的結尾。
hash_set::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
hash_set::erase (STL/CLR)	移除位於指定位置的項目。
hash_set::find (STL/CLR)	尋找符合指定之索引鍵的元素。
hash_set::hash_delegate (STL/CLR)	複製索引鍵的雜湊委派。
hash_set::hash_set (STL/CLR)	建構容器物件。
hash_set::insert (STL/CLR)	加入項目。
hash_set::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。

 hash_set::load_factor (STL/CLR) hash_set::lower_bound (STL/CLR) hash_set::make_value (STL/CLR) hash_set::max_load_factor (STL/CLR) hash_set::rbegin (STL/CLR) hash_set::rehash (STL/CLR) hash_set::rend (STL/CLR) hash_set::size (STL/CLR) hash_set::swap (STL/CLR) hash_set::to_array (STL/CLR) hash_set::upper_bound (STL/CLR) hash_set::value_comp (STL/CLR)	 計算每個值區的平均項目數。 尋找符合指定索引鍵的範圍開頭。 結構值物件。 取得或設定每個 Bucket 最大項目數。 指定反向受控制序列的開頭。 重建雜湊資料表。 指定反向受控制序列的結尾。 計算元素的數目。 交換兩個容器的內容。 將受控制序列複製到新的陣列。 尋找符合指定索引鍵的範圍結尾。 針對兩個元素值複製順序委派。
 hash_set::operator= (STL/CLR)	 取代受控制的序列。

介面

 ICloneable IEnumerable ICollection IEnumerable<T> ICollection<T> IHash<Key, Value>	 複製物件。 排序元素。 維護元素群組。 透過具類型的元素排序。 維護具類型的元素群組。 維護泛型容器。
--	---

備註

物件會在雙向連結清單中，配置並釋放它所控制之序列的儲存體，以作為個別節點。為了加速存取，物件也會在雜湊表) 的清單中維護不同長度的指標陣列，(雜湊表，有效地將整個清單視為清單子或值區的序列來管理。它會藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式，將專案插入至值區，以保持排序。這

表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `hash_set::key_compare` 類型的預存委派物件來排序每個值區，([STL/CLR](#))。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<=(key_type, key_type)`。

您可以藉由呼叫成員函式`hash_set::key_comp (STL/CLR)`來存取儲存的委派物件 `()`。這類委派物件必須在 `hash_set::key_type (STL/CLR)` 類型的索引鍵之間定義對等順序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y) && key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

任何行為類似 `operator<=(key_type, key_type)` `operator>=(key_type, key_type)` 或 `operator==(key_type, key_type)` 定義 equivalent 順序的排序規則。

請注意，容器只會確保其索引鍵具有對等順序的專案（，以及相同整數值）的雜湊在值區中相鄰。不同于樣板類別 `hash_multiset (STL/CLR)`，樣板類別的物件 `hash_set` 可確保所有元素的索引鍵都是唯一的。（沒有任何兩個索引鍵具有對等的排序。）

物件會藉由呼叫 `hash_set::hasher (STL/CLR)` 類型的預存委派物件，判斷哪些值區應包含指定的排序索引鍵。您可以藉由呼叫成員函式`hash_set::hash_delegate (STL/CLR) ()` 取得相依于索引鍵值的整數值，來存取這個儲存的物件。當您建立 `hash_set` 時，可以指定預存的委派物件。如果您未指定委派物件，則預設值為函數

`System::Object::hash_value(key_type)`。這表示對於任何金鑰 `x` 和 `y`：

`hash_delegate()(x)` 每次呼叫時，都會傳回相同的整數結果。

如果 `x` 和 `y` 具有對等順序，則應該傳回與 `hash_delegate()(x)` 相同的整數結果 `hash_delegate()(y)`。

每個元素都可作為索引鍵和值。順序的表示方式，可讓您查閱、插入和移除任意專案，而這些作業與序列中的專案數目無關，（常數時間）--至少在案例中的最大值。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

但是，如果雜湊值未一致地散發，雜湊表就可以進行退化。在最極端的情況下，雜湊函數一律會傳回相同的值（查閱、插入和移除），與序列中的專案數目成正比（線性時間）。容器會致力於選擇合理的雜湊函式、平均值區大小，以及雜湊表大小（值區的總數），但您可以覆寫任何或所有的選項。例如，請參閱函式 `hash_set::max_load_factor (stl/clr)` 和 `hash_set::rehash (stl/clr)`。

`Hash_set` 支援雙向反覆運算器，這表示您可以使用反覆運算器逐步執行連續的元素，以指定受控制序列中的元素。特殊的前端節點對應至`hash_set::end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素（如果有的話）。您可以將 `hash_set` 反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考指定其數位位置的 `hash_set` 專案，這需要隨機存取反覆運算器。

`Hash_set` 反覆運算器會將控制碼儲存至其相關聯的 `hash_set` 節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。`Hash_set` 反覆運算器會維持有效，只要其相關聯的 `hash_set` 節點與某些 `hash_set` 相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

`hash_set::begin (STL/CLR)`

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_set_begin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_Set::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}
```

hash_set::bucket_count (STL/CLR)

計算值區的數目。

語法

```
int bucket_count();
```

備註

成員函式會傳回目前的 bucket 數目。您可以使用它來判斷雜湊表的大小。

範例

```

// cliext_hash_set_bucket_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    {
        Myhash_set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_set:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫[hash_set:: erase \(stl/clr\)](#) () , [hash_set:: begin \(stl/clr\)](#) (), [hash_set:: end \(stl/clr\)](#) () 。

您可以使用它來確保受控制的序列是空的。

範例

```
// cliext_hash_set_clear.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
size() = 0
a b
size() = 0
```

hash_set:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 [T2](#) , 可作為受控制序列的常數雙向反覆運算器。

範例

```
// cliext_hash_set_const_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_Set::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set::const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```
// cliext_hash_set_const_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_Set::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
    {
        // get a const reference to an element
        Myhash_Set::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_hash_set_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_set::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

hash_set:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_hash_set_count.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

hash_set::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_hash_set_difference_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // compute positive difference
        Myhash_Set::difference_type diff = 0;
        for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
            ++diff;
        System::Console::WriteLine("end()-begin() = {0}", diff);

        // compute negative difference
        diff = 0;
        for (Myhash_Set::iterator it = c1.end(); it != c1.begin(); --it)
            --diff;
        System::Console::WriteLine("begin()-end() = {0}", diff);
        return (0);
    }
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

hash_set::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[hash_set::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 hash_set 是否為空白。

範例

```
// cliext_hash_set_empty.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

hash_set:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_hash_set_end.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect last two items
        Myhash_Set::iterator it = c1.end();
        --it;
        System::Console::WriteLine("--- --end() = {0}", *--it);
        System::Console::WriteLine("---end() = {0}", *++it);
        return (0);
    }
}

```

```

a b c
--- --end() = b
---end() = c

```

hash_set::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(hash_set::lower_bound (stl/clr) (key), hash_set::upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_hash_set_equal_range.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
typedef Myhash_Set::pair_iter_iter Pairii;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

hash_set:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 *where* 所指向之受控制序列的元素，並傳回反覆運算器，指定移除專案之後的第一個元

素, 或 `hash_set::end` (STL/CLR) (() 如果沒有這類專案存在)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍 [,) 中受控制序列的元素, `first` `last` 並傳回反覆運算器, 此反覆運算器會指定移除任何專案之後剩餘的第一個元素, 或 `end()` 如果沒有這類專案存在, 則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引 鍵具有對等排序之受控制序列的任何元素, 並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_hash_set_erase.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myhash_Set::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

hash_set::find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[hash_set:: end \(STL/CLR\)](#)。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_hash_set_find.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

hash_set:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    IHash<GKey, GValue>
    generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_hash_set_generic_container.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_Set::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify generic and display original
        gc1->insert(L'd');
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify original and display generic
        c1.insert(L'e');
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
a b c
a b c d
a b c d e

```

hash_set::generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```

// cliext_hash_set_generic_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        Myhash_Set::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // get an element and display it
        Myhash_Set::generic_iterator gcit = gc1->begin();
        Myhash_Set::generic_value gcval = *gcit;
        System::Console::WriteLine("{0} ", gcval);
        return (0);
    }
}

```

```

a b c
a b c
a

```

hash_set::generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```

// cliext_hash_set_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_Set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_Set::generic_reverse_iterator gcit = gc1->rbegin();
    Myhash_Set::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

hash_set::generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_hash_set_generic_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myhash_Set::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myhash_Set::generic_iterator gcit = gc1->begin();
    Myhash_Set::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

hash_set::hash_delegate (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```

hasher^ hash_delegate();

```

備註

成員函式會傳回用來將索引鍵值轉換為整數的委派。您可以使用它來雜湊索引鍵。

範例

```
// cliext_hash_set_hash_delegate.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    Myhash_set::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_set::hash_set (STL/CLR)

建構容器物件。

語法

```
hash_set();
explicit hash_set(key_compare^ pred);
hash_set(key_compare^ pred, hasher^ hashfn);
hash_set(hash_set<Key>% right);
hash_set(hash_set<Key>^ right);
template<typename InIter>
    hash_sethash_set(InIter first, InIter last);
template<typename InIter>
    hash_set(InIter first, InIter last,
            key_compare^ pred);
template<typename InIter>
    hash_set(InIter first, InIter last,
            key_compare^ pred, hasher^ hashfn);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right,
        key_compare^ pred);
hash_set(System::Collections::Generic::IEnumerable<GValue>^ right,
        key_compare^ pred, hasher^ hashfn);
```

參數

first

要插入的範圍開頭。

hashfn

將索引鍵對應至值區的雜湊函數。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
hash_set();
```

使用預設的順序述詞 `key_compare()` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及預設順序述詞和雜湊函數。

函數：

```
explicit hash_set(key_compare^ pred);
```

使用順序述詞 `pred` 和預設雜湊函式，初始化受控制的序列，但不含任何元素。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_set(key_compare^ pred, hasher^ hashfn);
```

使用順序述詞 `pred`，以及雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞和雜湊函數。

函數：

```
hash_set(hash_set<Key>% right);
```

使用序列 [`right.begin()` 、 `right.end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_set` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_set(hash_set<Key>^ right);
```

使用序列 [`right->begin()` 、 `right->end()`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `hash_set` 物件 許可權所控制之序列的複本，以及預設排序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_set(InIter first, InIter last);
```

使用序列 [`first` 、 `last`]、預設排序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它，將受控制的序列設為另一個序列的複本，並使用預設順序述詞和雜湊函數。

函數：

```
template<typename InIter> hash_set(InIter first, InIter last, key_compare^ pred);
```

使用) 順序述詞 `first` `last` `pred` 和預設雜湊函數，初始化受控制的序列。您可以使用它，以指定的順序述詞和預設雜湊函數，讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> hash_set(InIter first, InIter last, key_compare^ pred, hasher^ hashfn);
```

使用) 順序述詞 `first` `last` `pred` 和雜湊函數 `hashfn`，初始化受控制的序列。您可以使用它，以指定的順序述詞和雜湊函式，讓受控制的序列成為另一個序列的複本。

函數：

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right);
```

使用列舉值 右邊指定的序列、預設順序述詞和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及預設排序述詞和雜湊函數。

函數：

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred* 和預設雜湊函數，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和預設雜湊函數。

函數：

```
hash_set(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred, hasher^ hashfn);
```

使用列舉值 右邊所指定的序列、順序述詞 *pred*，以及雜湊函數 *hashfn*，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及指定的順序述詞和雜湊函數。

範例

```
// cliext_hash_set_construct.cpp
// compile with: /clr
#include <cliext/hash_set>

int myfun(wchar_t key)
{ // hash a key
    return (key ^ 0xdeadbeef);
}

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
// construct an empty container
    Myhash_set c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule
    Myhash_set c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an ordering rule and hash function
    Myhash_set c2h(cliext::greater_equal<wchar_t>(),
        gcnew Myhash_set::hasher(&myfun));
    System::Console::WriteLine("size() = {0}", c2h.size());

    c2h.insert(c1.begin(), c1.end());
    for each (wchar_t elem in c2h)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine();

    // construct with an iterator range
    Myhash_set c3(c1.begin(), c1.end());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an iterator range and an ordering rule
    Myhash_set c4(c1.begin(), c1.end(),
```

```

        cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule and hash function
Myhash_set c4h(c1.begin(), c1.end(),
               cliext::greater_equal<wchar_t>(),
               gcnew Myhash_set::hasher(&myfun));
for each (wchar_t elem in c4h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct with an enumeration
Myhash_set c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myhash_set c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule and hash function
Myhash_set c6h( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>(),
    gcnew Myhash_set::hasher(&myfun));
for each (wchar_t elem in c6h)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
System::Console::WriteLine();

// construct from a generic container
Myhash_set c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myhash_set c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
a b c
size() = 0
c b a

a b c
a b c
c b a

a b c
a b c
c b a

a b c
a b c
```

hash_set:: hasher (STL/CLR)

索引鍵的雜湊委派。

語法

```
Microsoft::VisualC::StlClr::UnaryDelegate<GKey, int>
    hasher;
```

備註

此類型說明將索引鍵值轉換為整數的委派。

範例

```
// cliext_hash_set_hasher.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    Myhash_set::hasher^ myhash = c1.hash_delegate();

    System::Console::WriteLine("hash(L'a') = {0}", myhash(L'a'));
    System::Console::WriteLine("hash(L'b') = {0}", myhash(L'b'));
    return (0);
}
```

```
hash(L'a') = 1616896120
hash(L'b') = 570892832
```

hash_set:: insert (STL/CLR)

加入項目。

語法

```
cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);
```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會致力於插入具有值 *va* 的元素，並傳回一對值 *x*。如果 *x.second* 為 true，會 *x.first* 指定新插入的專案，否則會 *x.first* 指定具有對等順序的專案，而該專案已存在，且不會插入新的元素。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 *va* 的元素，並使用 *where* 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [*first* , *last*) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_hash_set_insert.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
typedef Myhash_set::pair_iter_bool Pairib;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 = c1.insert(L'x');
    System::Console::WriteLine("insert(L'x') = [{0} {1}]",
        *pair1.first, pair1.second);

    pair1 = c1.insert(L'b');
    System::Console::WriteLine("insert(L'b') = [{0} {1}]",
        *pair1.first, pair1.second);

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Myhash_set c2;
    Myhash_set::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Myhash_set c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = [x True]
insert(L'b') = [b False]
a b c x
insert(begin(), L'y') = y
a b c x y
a b c x
a b c x y

```

hash_set:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_hash_set_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myhash_Set::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_hash_set_key_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_Set c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_set:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_hash_set_key_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    Myhash_set::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myhash_set c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

hash_set::key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_hash_set_key_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myhash_Set::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set::load_factor (STL/CLR)

計算每個值區的平均項目數。

語法

```
float load_factor();
```

備註

成員函式會傳回 `(float) hash_set::size (stl/clr) () / hash_set::bucket_count (stl/clr) ()`。您可以使用它來判斷平均 bucket 大小。

範例

```

// cliext_hash_set_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_set::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，此專案 `x` 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案存在，則會傳回 `hash_set::end (STL/CLR)` `()`；否則會傳回指定的 iterator `x`。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```
// cliext_hash_set_lower_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b')));
    return (0);
}
```

```
a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b
```

hash_set:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key);
```

參數

key

要使用的索引鍵值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```
// cliext_hash_set_make_value.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(Myhash_set::make_value(L'a'));
    c1.insert(Myhash_set::make_value(L'b'));
    c1.insert(Myhash_set::make_value(L'c'));

    // display contents " a b c"
    for each (Myhash_set::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set:: max_load_factor (STL/CLR)

取得或設定每個 Bucket 最大項目數。

語法

```
float max_load_factor();
void max_load_factor(float new_factor);
```

參數

new_factor

要儲存的新最大載入因數。

備註

第一個成員函式會傳回目前儲存的最大載入因數。您可以使用它來判斷平均 bucket 大小上限。

第二個成員函式會以 *new_factor*取代儲存區的最大載入因數。在後續的插入之前，不會發生自動重新雜湊。

範例

```

// cliext_hash_set_max_load_factor.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect current parameters
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // change max_load_factor and redisplay
    c1.max_load_factor(0.25f);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    System::Console::WriteLine();

    // rehash and redisplay
    c1.rehash(100);
    System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
    System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
    System::Console::WriteLine("max_load_factor() = {0}",
        c1.max_load_factor());
    return (0);
}

```

hash_set:: operator = (STL/CLR)

取代受控制的序列。

語法

```
hash_set<Key>% operator=(hash_set<Key>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_hash_set_operator_as.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myhash_set::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myhash_set c2;
    c2 = c1;
    // display contents " a b c"
    for each (Myhash_set::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

hash_set:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_hash_set_rbegin.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myhash_Set::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
```

hash_set:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_hash_set_reference.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myhash_Set::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
    {   // get a reference to an element
        Myhash_Set::reference ref = *it;
        System::Console::Write("{0} ", ref);
    }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

hash_set:: rehash (STL/CLR)

重建雜湊資料表。

語法

```
void rehash();
```

備註

成員函式會重建雜湊表，以確保[hash_set:: load_factor \(stl/clr\)](#) (\leq) [hash_set:: max_load_factor \(stl/clr\)](#)。否則，雜湊表只會在插入之後視需要增加大小。(不會自動縮減大小。) 您使用它來調整雜湊表的大小。

範例

```

// cliext_hash_set_rehash.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    {
        Myhash_set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // inspect current parameters
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // change max_load_factor and redisplay
        c1.max_load_factor(0.25f);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        System::Console::WriteLine();

        // rehash and redisplay
        c1.rehash(100);
        System::Console::WriteLine("bucket_count() = {0}", c1.bucket_count());
        System::Console::WriteLine("load_factor() = {0}", c1.load_factor());
        System::Console::WriteLine("max_load_factor() = {0}",
            c1.max_load_factor());
        return (0);
    }
}

```

```

a b c
bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 4

bucket_count() = 16
load_factor() = 0.1875
max_load_factor() = 0.25

bucket_count() = 128
load_factor() = 0.0234375
max_load_factor() = 0.25

```

hash_set::rend (STL/CLR)

指定反向受控制序列的结尾。

语法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_hash_set_rend.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myhash_Set::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("--- --rend() = {0}", *--rit);
    System::Console::WriteLine("---rend() = {0}", *++rit);
    return (0);
}
```

```
a b c
--- --rend() = b
---rend() = a
```

hash_set:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_hash_set_reverse_iterator.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myhash_Set::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

hash_set::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[hash_set::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_hash_set_size.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

hash_set::size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```
// cliext_hash_set_size_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myhash_Set::size_type diff = 0;
    for (Myhash_Set::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

hash_set:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(hash_set<Key>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_hash_set_swap.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct another container with repetition of values
        Myhash_Set c2;
        c2.insert(L'd');
        c2.insert(L'e');
        c2.insert(L'f');
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // swap and redisplay
        c1.swap(c2);
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
d e f
d e f
a b c

```

hash_set:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_hash_set_to_array.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    {
        Myhash_Set c1;
        c1.insert(L'a');
        c1.insert(L'b');
        c1.insert(L'c');

        // copy the container and modify it
        cli::array<wchar_t>^ a1 = c1.to_array();

        c1.insert(L'd');
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // display the earlier array copy
        for each (wchar_t elem in a1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c d
a b c

```

hash_set::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```

iterator upper_bound(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，此專案 *x* 會雜湊到與索引 鍵相同的值區，並對索引 鍵具有對等的排序。如果沒有這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 **hash_set::END (STL/CLR) ()**；否則會傳回反覆運算器，指定超過的第一個元素 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_hash_set_upper_bound.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

hash_set::value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```
value_compare^ value_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```

// cliext_hash_set_value_comp.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

hash_set::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```

// cliext_hash_set_value_compare.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_Set;
int main()
{
    Myhash_Set c1;
    Myhash_Set::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = True
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

hash_set::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_hash_set_value_type.cpp
// compile with: /clr
#include <cliext/hash_set>

typedef cliext::hash_set<wchar_t> Myhash_set;
int main()
{
    Myhash_set c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myhash_set::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myhash_set::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

list (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `list` 以雙向連結的節點清單來管理專案序列，每個節點都會儲存一個元素。

在下列描述中，與 `GValue` 值相同，除非後者是 ref 型別，在這種情況下就是 `Value^`。

語法

```
template<typename Value>
ref class list
    : public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
Microsoft::VisualC::StlClr::IList<GValue>
{ .... };
```

參數

值

受控制序列中項目的類型。

需求

標頭: <cliext/list>

命名空間: cliext

宣告

<code>list::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>list::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>list::const_reverse_iterator (STL/CLR)</code>	用於受控制序列的常數反向迭代器類型。
<code>list::difference_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。
<code>list::generic_container (STL/CLR)</code>	容器的泛型介面型別。
<code>list::generic_iterator (STL/CLR)</code>	容器之泛型介面的反覆運算器類型。
<code>list::generic_reverse_iterator (STL/CLR)</code>	容器的泛型介面之反向反覆運算器的類型。
<code>list::generic_value (STL/CLR)</code>	容器之泛型介面的元素類型。

<p> </p> <p>list::iterator (STL/CLR)</p> <p>list::reference (STL/CLR)</p> <p>list::reverse_iterator (STL/CLR)</p> <p>list::size_type (STL/CLR)</p> <p>list::value_type (STL/CLR)</p>	<p> </p> <p>受控制序列之迭代器的類型。</p> <p>項目的參考類型。</p> <p>受控制序列的反向迭代器類型。</p> <p>兩個項目之間帶正負號距離的類型。</p> <p>項目的類型。</p>
<p> </p> <p>list::assign (STL/CLR)</p> <p>list::back (STL/CLR)</p> <p>list::begin (STL/CLR)</p> <p>list::clear (STL/CLR)</p> <p>list::empty (STL/CLR)</p> <p>list::end (STL/CLR)</p> <p>list::erase (STL/CLR)</p> <p>list::front (STL/CLR)</p> <p>list::insert (STL/CLR)</p> <p>list::list (STL/CLR)</p> <p>list::merge (STL/CLR)</p> <p>list::pop_back (STL/CLR)</p> <p>list::pop_front (STL/CLR)</p> <p>list::push_back (STL/CLR)</p> <p>list::push_front (STL/CLR)</p> <p>list::rbegin (STL/CLR)</p> <p>list::remove (STL/CLR)</p> <p>list::remove_if (STL/CLR)</p> <p>list::rend (STL/CLR)</p>	<p> </p> <p>取代所有項目。</p> <p>存取最後一個項目。</p> <p>指定受控制序列的開頭。</p> <p>移除所有項目。</p> <p>測試項目是否不存在。</p> <p>指定受控制序列的結尾。</p> <p>移除位於指定位置的項目。</p> <p>存取第一個項目。</p> <p>將專案加入至指定的位置。</p> <p>建構容器物件。</p> <p>合併兩個已排序的受控制序列。</p> <p>移除最後一個元素。</p> <p>移除第一個元素。</p> <p>加入新的最後一個元素。</p> <p>加入新的第一個元素。</p> <p>指定反向受控制序列的開頭。</p> <p>移除具有指定之值的元素。</p> <p>移除通過指定之測試的元素。</p> <p>指定反向受控制序列的結尾。</p>

list::resize (STL/CLR)	變更項目的數目。
list::reverse (STL/CLR)	反轉受控制的序列。
list::size (STL/CLR)	計算元素的數目。
list::sort (STL/CLR)	排序受控制序列。
list::splice (STL/CLR)	重新拼接節點之間的連結。
list::swap (STL/CLR)	交換兩個容器的內容。
list::to_array (STL/CLR)	將受控制序列複製到新的陣列。
list::unique (STL/CLR)	移除通過指定測試的相鄰項目。

list::back_item (STL/CLR)	存取最後一個項目。
list::front_item (STL/CLR)	存取第一個項目。

list::operator= (STL/CLR)	取代受控制的序列。
operator!= (list) (STL/CLR)	判斷物件是否 <code>list</code> 不等於另一個 <code>list</code> 物件。
operator< (清單) (STL/CLR)	判斷 <code>list</code> 物件是否小於另一個 <code>list</code> 物件。
operator<= (清單) (STL/CLR)	判斷 <code>list</code> 物件是否小於或等於另一個 <code>list</code> 物件。
operator = = (list) (STL/CLR)	判斷 <code>list</code> 物件是否等於另一個 <code>list</code> 物件。
operator> (清單) (STL/CLR)	判斷 <code>list</code> 物件是否大於另一個 <code>list</code> 物件。
operator>= (清單) (STL/CLR)	判斷 <code>list</code> 物件是否大於或等於另一個 <code>list</code> 物件。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。

<code>IEnumerable<T></code>	透過具類型的元素排序。
<code>ICollection<T></code>	維護具類型的元素群組。
<code>IList<Value></code>	維護泛型容器。

備註

物件會在雙向連結清單中，配置並釋放它所控制之序列的儲存體，以作為個別節點。它會藉由變更節點之間的連結來重新排列元素，絕不會將某個節點的內容複寫到另一個節點。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。因此，清單是樣板類別併列的基礎容器 ([stl/clr](#)) 或樣板類別 [堆疊 \(stl/clr\)](#) 的絕佳候選。

`list` 物件支援雙向反覆運算器，這表示您可以逐步執行指定受控制序列中元素的反覆運算器，以逐步執行連續的元素。特殊的前端節點對應至 `list:: end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將清單反覆運算器遞增以到達前端節點，然後比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考清單專案的數值位置(需要隨機存取反覆運算器)。因此，清單 無法 當做樣板類別 [priority_queue \(STL/CLR\)](#) 的基礎容器使用。

清單反覆運算器會將控制碼儲存至其相關聯的清單節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。清單 `iterator` 會維持有效，只要其相關聯的清單節點與某個清單相關聯。此外，有效的 `iterator` 是 `derefencable`--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 `ref` 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的 容器不會摧毀其元素。

成員

`list:: assign (STL/CLR)`

取代所有項目。

語法

```
void assign(size_type count, value_type val);
template<typename InIt>
void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumerable<Value>^ right);
```

參數

計數

要插入的元素數目。

`first`

要插入的範圍開頭。

`last`

要插入的範圍結尾。

`對`

要插入的列舉。

瓦爾

要插入的元素值。

備註

第一個成員函式會以值 *va* 的 *count* 元素重複來取代受控制的序列。您可以使用它來填滿具有相同值之元素的容器。

如果 `InIt` 是整數類型，則第二個成員函式的行為會與相同 `assign((size_type)first, (value_type)last)`。否則，它會以順序 `[,)` 取代受控制的序列 `first last`。您可以使用它來讓受控制序列的複製另一個序列。

第三個成員函式會將受控制序列取代為列舉值 右邊所指定的序列。您可以使用它來讓受控制的序列成為列舉值所描述之序列的複本。

範例

```
// cliext_list_assign.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::list<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    cliext::list<wchar_t>::iterator it = c1.end();
    c2.assign(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
x x x x x x
a b
a b c
```

list:: back (STL/CLR)

存取最後一個項目。

語法

```
reference back();
```

備註

成員函式會傳回受控制序列之最後一個元素的參考，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來存取最後一個專案。

範例

```
// cliext_list_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back() = c
a b x
```

list::back_item (STL/CLR)

存取最後一個項目。

語法

```
property value_type back_item;
```

備註

屬性會存取受控制序列的最後一個專案，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來讀取或寫入最後一個專案。

範例

```

// cliext_list_back_item.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back_item = c
a b x

```

list::begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回隨機存取反覆運算器，此反覆運算器會指定受控制序列的第一個專案，或在空序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_list_begin.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::list<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b
x y c

```

list::clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫[list::erase \(stl/clr\) \(\)](#)、[list::begin \(stl/clr\) \(\)](#)、[list::end \(stl/clr\) \(\)](#)。您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_list_clear.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    {
        cliext::list<wchar_t> c1;
        c1.push_back(L'a');
        c1.push_back(L'b');
        c1.push_back(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // clear the container and reinspect
        c1.clear();
        System::Console::WriteLine("size() = {0}", c1.size());

        // add elements and clear again
        c1.push_back(L'a');
        c1.push_back(L'b');

        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        c1.clear();
        System::Console::WriteLine("size() = {0}", c1.size());
    }
}

```

```

a b c
size() = 0
a b
size() = 0

```

list::const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```

typedef T2 const_iterator;

```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數隨機存取反覆運算器。

範例

```
// cliext_list_const_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::list<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

list:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```
// cliext_list_const_reference.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::list<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        cliext::list<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

list:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_list_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::list<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::list<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

清單 ::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型會描述已簽署的元素計數。

範例

```

// cliext_list_difference_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::list<wchar_t>::difference_type diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

list::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[list::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試清單是否為空的。

範例

```
// cliext_list_empty.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

list::end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回隨機存取反覆運算器，其指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_list_end.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::list<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine("*- --end() = {0}", *--it);
    System::Console::WriteLine("*-end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*- --end() = b
*-end() = c
a x y

```

list:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

參數

first

要清除的範圍開頭。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 *where* 所指向之受控制序列的元素。您可以使用它來移除單一專案。

第二個成員函式會移除 [*first*, *last*) 範圍中受控制序列中的元素。您可以使用它來移除零個或多個連續元素。

這兩個成員函式會傳回反覆運算器，此反覆運算器會指定任何移除的元素之外的第一個元素，如果沒有這類專案存在，則會將`list:: end (STL/CLR) ()`。

清除專案時，專案複本的數目會是抹除結尾和序列最接近端之間的元素數目的線性。(清除序列結尾的一或多個元素時，不會複製任何元素。)

範例

```
// cliext_list_erase.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::list<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

list:: front (STL/CLR)

存取第一個項目。

語法

```
reference front();
```

備註

成員函式會傳回受控制序列的第一個元素的參考，而該專案必須為非空白。當您知道第一個元素存在時，您可以使用它來讀取或寫入第一個專案。

範例

```
// cliext_list_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

list::front_item (STL/CLR)

存取第一個項目。

語法

```
property value_type front_item;
```

備註

屬性會存取受控制序列的第一個專案，其必須為非空白。當您知道第一個元素存在時，您可以使用它來讀取或寫入第一個專案。

範例

```
// cliext_list_front_item.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front_item = a
x b c
```

list:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    IList<generic_value>
generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_list_generic_container.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

list:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```

// cliext_list_generic_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    {
        cliext::list<wchar_t> c1;
        c1.push_back(L'a');
        c1.push_back(L'b');
        c1.push_back(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        cliext::list<wchar_t>::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify generic and display original
        cliext::list<wchar_t>::generic_iterator gcit = gc1->begin();
        cliext::list<wchar_t>::generic_value gcval = *gcit;
        *++gcit = gcval;
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
a b c
a a c

```

list::generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseBidirectionalIterator<generic_value> generic_reverse_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```
// cliext_list_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::list<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::list<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::list<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
a c c
```

list::generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_list_generic_value.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    {
        cliext::list<wchar_t> c1;
        c1.push_back(L'a');
        c1.push_back(L'b');
        c1.push_back(L'c');

        // display contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // construct a generic container
        cliext::list<wchar_t>::generic_container^ gc1 = %c1;
        for each (wchar_t elem in gc1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // modify generic and display original
        cliext::list<wchar_t>::generic_iterator gcit = gc1->begin();
        cliext::list<wchar_t>::generic_value gcval = *gcit;
        *++gcit = gcval;
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();
        return (0);
    }
}

```

```

a b c
a b c
a a c

```

list:: insert (STL/CLR)

將專案加入至指定的位置。

語法

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
    void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的元素值。

where

在容器中要插入的位置。

備註

每個成員函式會在受控制序列中的 *位置* 所指向的專案之前，插入由其餘運算元所指定的序列。

第一個成員函式會插入具有值 *va*/ 的元素，並傳回指定新插入之元素的反覆運算器。您可以使用它，在反覆運算器所指定的位置之前插入單一元素。

第二個成員函式會插入值 *va*/ 的 *count* 元素重複。您可以使用它來插入零個或多個連續的元素，這些都是相同值的所有複本。

如果 `Init` 是整數類型，第三個成員函式的行為即與 `insert(where, (size_type)first, (value_type)last)` 相同。否則，它會插入順序 [`first` , `last`)。您可以使用它來插入零個或多個從另一個序列複製的連續元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

插入單一專案時，專案複本的數目是在插入點與序列最接近端之間的專案數中的線性。(在序列的任一結尾插入一或多個專案時，不會複製任何專案。) 如果 `Init` 是輸入反覆運算器，則第三個成員函式會針對序列中的每個元素，有效地執行單一插入。否則，插入專案時 `N`，專案複本的數目會是線性，`N` 加上插入點與序列最接近端之間的元素數目。

範例

```

// cliext_list_insert.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    {
        cliext::list<wchar_t> c1;
        c1.push_back(L'a');
        c1.push_back(L'b');
        c1.push_back(L'c');

        // display initial contents " a b c"
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert a single value using iterator
        cliext::list<wchar_t>::iterator it = c1.begin();
        System::Console::WriteLine("insert(begin())+1, L'x') = {0}",
            *c1.insert(++it, L'x'));
        for each (wchar_t elem in c1)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert a repetition of values
        cliext::list<wchar_t> c2;
        c2.insert(c2.begin(), 2, L'y');
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert an iterator range
        it = c1.end();
        c2.insert(c2.end(), c1.begin(), --it);
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert an enumeration
        c2.insert(c2.begin(), // NOTE: cast is not needed
            (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        // insert a single value using index
        it = c2.begin();
        ++it, ++it, ++it;
        c2.insert(it, L'z');
        for each (wchar_t elem in c2)
            System::Console::Write("{0} ", elem);
        System::Console::WriteLine();

        return (0);
    }
}

```

```

a b c
insert(begin())+1, L'x') = x
a x b c
y y
y y a x b
a x b c y y a x b

```

list:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的隨機存取反覆運算器。

範例

```
// cliext_list_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::list<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

list:: list (STL/CLR)

建構容器物件。

語法

```
list();
list(list<Value>% right);
list(list<Value>^ right);
explicit list(size_type count);
list(size_type count, value_type val);
template<typename InIt>
list(InIt first, InIt last);
list(System::Collections::Generic::IEnumerable<Value>^ right);
```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的物件或範圍。

瓦爾

要插入的元素值。

備註

函數：

```
list();
```

使用沒有元素的方式，初始化受控制的序列。您可以使用它來指定空的初始受控制序列。

函數：

```
list(list<Value>% right);
```

使用序列 [,) 初始化受控制的序列 `right.begin()` `right.end()`。您可以使用它來指定初始受控制序列，這是清單物件 許可權所控制之序列的複本。

函數：

```
list(list<Value>^ right);
```

使用序列 [,) 初始化受控制的序列 `right->begin()` `right->end()`。您可以使用它來指定初始受控制序列，這是由其控制碼 正確的清單物件所控制的序列複本。

函數：

```
explicit list(size_type count);
```

使用 *count* 元素(具有值)初始化受控制的序列 `value_type()`。您可以使用它來填滿具有預設值的元素的容器。

函數：

```
list(size_type count, value_type val);
```

使用具有值 *val* 的 *count* 元素，初始化受控制的序列。您可以使用它來填滿具有相同值之元素的容器。

函數：

```
template<typename InIt>
```

```
list(InIt first, InIt last);
```

使用序列 [,) 初始化受控制的序列 `first` `last`。您可以使用它來讓受控制的序列成為另一個序列的複本。

函數：

```
list(System::Collections::Generic::IEnumerable<Value>^ right);
```

使用列舉值 右邊指定的順序，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```

// cliext_list_construct.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    // construct an empty container
    cliext::list<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    // construct with a repetition of default values
    cliext::list<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // construct with a repetition of values
    cliext::list<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an iterator range
    cliext::list<wchar_t>::iterator it = c3.end();
    cliext::list<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct with an enumeration
    cliext::list<wchar_t> c5( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container
    cliext::list<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying a container handle
    cliext::list<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

size() = 0
0 0 0
x x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x

```

list:: merge (STL/CLR)

合併兩個已排序的受控制序列。

語法

```
void merge(list<Value>% right);
template<typename Pred2>
void merge(list<Value>% right, Pred2 pred);
```

參數

Pred

元素配對的比較子。

對

要合併的容器。

備註

第一個成員函式會從 `右邊` 控制的序列中移除所有專案，並將其插入受控制的序列。這兩個序列必須先前以 `operator<` --元素排序，且在您進行任一順序時，不能以值減少。產生的序列也會依排序 `operator<`。您可以使用這個成員函式，將值增加的兩個序列合併為也會增加值的序列。

第二個成員函式的行為與第一個成員函式的行為相同，不同之處在於 `pred -- pred(X, Y) X` 順序中的元素後面的任何元素的排序次序都必須為 `false Y`。您可以使用它來合併以述詞函式或您指定之委派排序的兩個序列。

這兩個函式都會執行穩定合併--在產生的受控制序列中，不會反轉原始受控制序列中的任何一對元素。此外，如果成對的元素 `X` 和 `Y` 在產生的受控制序列中有對等的排序--- `!(X < Y) && !(Y < X)` -原始受控制序列中的專案，會出現在 `右邊` 所控制之序列的元素之前。

範例

```

// cliext_list_merge.cpp
// compile with: /clr
#include <cliext/list>

typedef cliext::list<wchar_t> Mylist;
int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'c');
    c1.push_back(L'e');

    // display initial contents " a c e"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    cliext::list<wchar_t> c2;
    c2.push_back(L'b');
    c2.push_back(L'd');
    c2.push_back(L'f');

    // display initial contents " b d f"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // merge and display
    cliext::list<wchar_t> c3(c1);
    c3.merge(c2);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c2.size() = {0}", c2.size());

    // sort descending, merge descending, and redisplay
    c1.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    c3.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    c3.merge(c1, cliext::greater<wchar_t>());
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c1.size() = {0}", c1.size());
    return (0);
}

```

```

a c e
b d f
a b c d e f
c2.size() = 0
e c a
f e d c b a
f e e d c c b a a
c1.size() = 0

```

list:: operator = (STL/CLR)

取代受控制的序列。

語法

```
list<Value>% operator=(list<Value>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_list_operator_as.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

清單::pop_back (STL/CLR)

移除最後一個元素。

語法

```
void pop_back();
```

備註

成員函式會移除受控制序列的最後一個專案，其必須為非空白。您可以用它來將清單縮短一回一個專案。

範例

```
// cliext_list_pop_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b
```

清單 ::p op_front (STL/CLR)

移除第一個元素。

語法

```
void pop_front();
```

備註

成員函式會移除受控制序列的第一個專案，其必須為非空白。您可以使用它來將清單縮短到前一個元素。

範例

```

// cliext_list_pop_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_front();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

清單 ::push_back (STL/CLR)

加入新的最後一個元素。

語法

```
void push_back(value_type val);
```

備註

成員函式會 `val` 在受控制序列的結尾插入具有值的元素。您可以使用它將另一個元素附加至清單。

範例

```

// cliext_list_push_back.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

清單 ::push_front (STL/CLR)

加入新的第一個元素。

語法

```
void push_front(value_type val);
```

備註

成員函式會 `val` 在受控制序列的開頭插入具有值的元素。您可以使用它，在清單前面加上另一個元素。

範例

```
// cliext_list_push_front.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_front(L'a');
    c1.push_front(L'b');
    c1.push_front(L'c');

    // display contents " c b a"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

list::rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```

// cliext_list_rbegin.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::list<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*rbegin() = c
*++rbegin() = b
a y x

```

list:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```

// cliext_list_reference.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::list<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        {   // get a reference to an element
            cliext::list<wchar_t>::reference ref = *it;
            System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        {   // get a reference to an element
            cliext::list<wchar_t>::reference ref = *it;

            ref += (wchar_t)(L'A' - L'a');
            System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
A B C

```

list::remove (STL/CLR)

移除具有指定之值的元素。

語法

```

void remove(value_type val);

```

參數

瓦爾

要移除之元素的值。

備註

成員函式會移除受控制序列中的元素, `((System::Object^)val)->Equals((System::Object^)x)` 如果有任何), 則為 true ()。您可以使用它來清除具有指定值的任意元素。

範例

```

// cliext_list_remove.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // fail to remove and redisplay
    c1.remove(L'A');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // remove and redisplay
    c1.remove(L'b');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c

```

list::remove_if (STL/CLR)

移除通過指定之測試的元素。

語法

```

template<typename Pred1>
void remove_if(Pred1 pred);

```

參數

Pred

測試要移除的元素。

備註

成員函式會從受控制的序列中移除 (清除) 每個專案 x $pred(x)$ 都是 true。您可以使用它來移除滿足您指定為函式或委派之條件的所有元素。

範例

```

// cliext_list_remove_if.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'b');
    c1.push_back(L'b');
    c1.push_back(L'b');

    // display initial contents " a b b b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // fail to remove and redisplay
    c1.remove_if(cliext::binder2nd<cliext::equal_to<wchar_t> >(
        cliext::equal_to<wchar_t>(), L'd'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // remove and redisplay
    c1.remove_if(cliext::binder2nd<cliext::not_equal_to<wchar_t> >(
        cliext::not_equal_to<wchar_t>(), L'b'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b b b c
a b b b c
b b b

```

list::rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_list_rend.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::list<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("*- --rend() = {0}", *--rit);
    System::Console::WriteLine("*-rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*- --rend() = b
*-rend() = a
y x c

```

list:: resize (STL/CLR)

變更項目的數目。

語法

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

參數

new_size

受控制序列的新大小。

瓦爾

填補元素的值。

備註

成員函式可確保 `list:: size (STL/CLR) ()` 因而需要傳回 *new_size*。如果它必須讓受控制序列的時間變長，第一個成員函式會附加具有值的元素 `value_type()`，而第二個成員函式會附加具有值 *val* 的元素。為了讓受控制的序列變得更短，這兩個成員函式會有效地清除最後一個元素 `list:: size (STL/CLR) () - new_size` 次。您可以使用它來確保受控制序列的大小 *new_size*，方法是修剪或填補目前的受控制序列。

範例

```
// cliext_list_resize.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
// construct an empty container and pad with default values
    cliext::list<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

    // resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

    // resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
0 0 0 0
size() = 0
x x x x x
```

list:: reverse (STL/CLR)

反轉受控制的序列。

語法

```
void reverse();
```

備註

成員函式會反轉受控制序列中所有元素的順序。您可以使用它來反映元素的清單。

範例

```
// cliext_list_reverse.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // reverse and redisplay
    c1.reverse();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
c b a
```

list::reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_list_reverse_iterator.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::list<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
x b a
```

list::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[list::empty \(STL/CLR\) \(\)](#)。

範例

```
// cliext_list_size.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}
```

```
a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2
```

list::size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_list_size_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::list<wchar_t>::size_type diff = 0;
    for (cliext::list<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

list:: sort (STL/CLR)

排序受控制序列。

語法

```

void sort();
template<typename Pred2>
void sort(Pred2 pred);

```

參數

Pred

元素配對的比較子。

備註

第一個成員函式會重新排列受控制序列中的專案，使其以--元素排序，不會 `operator<` 隨著您在順序中進行而減少值。您可以使用這個成員函式，以遞增順序排序次序。

第二個成員函式的行為與第一個成員函式的行為相同，不同之處在於 `pred -- pred(X, Y) X` 結果序列中元素後面的任何元素的順序是以 `false` 排序 `Y`。您可以使用它，以述詞函式或委派所指定的順序來排序序列。

這兩個函式都會執行穩定的排序--在產生的受控制序列中，不會對原始受控制序列中的任何專案進行反轉。

範例

```

// cliext_list_sort.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // sort descending and redisplay
    c1.sort(cliext::greater<wchar_t>());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // sort ascending and redisplay
    c1.sort();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
c b a
a b c

```

list::splice (STL/CLR)

Restitch-節點之間的連結。

語法

```

void splice(iterator where, list<Value>% right);
void splice(iterator where, list<Value>% right,
            iterator first);
void splice(iterator where, list<Value>% right,
            iterator first, iterator last);

```

參數

first

要拼接的範圍開頭。

last

要拼接的範圍結尾。

對

要作為拼接來源的容器。

where

容器中要先拼接的位置。

備註

第一個成員函式會在受控制序列中由 `where` 所指向的元素之前，插入所控制的序列。它也會移除右邊的所有元素。
(`%right` 不能相等 `this`。) 您使用它來將所有清單拼接到另一個清單。

第二個成員函式會先在由 `right` 控制的序列中移除所指向的專案，然後將它插入受控制序列中由 `where` 所指向的元素之前。(`where == first` || `where == ++first`，則不會發生任何變更。) 您使用它來將某個清單的單一專案拼接至另一個清單。

第三個成員函式會將 `[]` 中所指定的子範圍，插入受 `first` | `last` 控制序列中由 `where` 所指向的元素之前。它也會從右邊控制的序列中移除原始的子範圍。(如果 `right == this`，範圍 `[first, last]` 不能包含 `where` 所指向的元素。) 您使用它來將零個或多個專案的子序列，從一個清單拼接到另一個清單。

範例

```
// cliext_list_splice.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // splice to a new list
    cliext::list<wchar_t> c2;
    c2.splice(c2.begin(), c1);
    System::Console::WriteLine("c1.size() = {0}", c1.size());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // return one element
    c1.splice(c1.end(), c2, c2.begin());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // return remaining elements
    c1.splice(c1.begin(), c2, c2.begin(), c2.end());
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("c2.size() = {0}", c2.size());
    return (0);
}
```

```
a b c
c1.size() = 0
a b c
a
b c
b c a
c2.size() = 0
```

list:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(list<Value>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `*this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```
// cliext_list_swap.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::list<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x x x x x
x x x x x
a b c
```

list:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<Value>^ to_array();
```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```
// cliext_list_to_array.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

list:: unique (STL/CLR)

移除通過指定測試的相鄰項目。

語法

```
void unique();
template<typename Pred2>
void unique(Pred2 pred);
```

參數

Pred

元素配對的比較子。

備註

第一個成員函式會從受控制的序列中移除 (清除) 每個比較等於先前專案的元素，如果專案 x 在專案之前 y ，則成員函式會 $x == y$ 移除 y 。您可以使用它來移除所有比較相等的相鄰元素子序列的所有複本，而不是一個複本。請注意，如果受控制序列的排序，例如藉由呼叫[list:: sort \(STL/CLR\)](#) ()，則成員函式只會保留具有唯一值的元素。(也因此才名為終端方法)。

第二個成員函式的行為與第一個成員函式的行為相同，不同之處在於它會移除專案後面的每個元素 Y X 。您可以使用它來移除所有相鄰元素子序列的所有專案，這些專案符合您指定的述詞函式或委派。請注意，如果受控制序列的排序，例如藉由呼叫，成員函式只會將沒有對 `sort(pred)` 等順序的元素保留為任何其他元素。

範例

```
// cliext_list_unique.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display contents after unique
    cliext::list<wchar_t> c2(c1);
    c2.unique();
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display contents after unique(not_equal_to)
    c2 = c1;
    c2.unique(cliext::not_equal_to<wchar_t>());
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a a b c
a b c
a a
```

list::value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數 `Value` 同義。

範例

```

// cliext_list_value_type.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::list<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {
        // store element in value_type object
        cliext::list<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

operator != (list) (STL/CLR)

清單不等於比較。

語法

```

template<typename Value>
bool operator!=(list<Value>% left,
                 list<Value>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個清單是依專案進行比較時，左邊是否未以正確的順序排序。

範例

```

// cliext_list_operator_ne.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

運算子 < (list) (STL/CLR)

清單小於比較。

語法

```

template<typename Value>
bool operator<(list<Value>% left,
    list<Value>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 `i !(right[i] < left[i])` 而言也是 true `left[i] < right[i]`。否則，它會傳回 `left->size() < right->size()` 您使用它來測試當兩個清單是依專案進行比較時，是否要在 `right` 之前排序左。

範例

```
// cliext_list_operator_lt.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (list) (STL/CLR)

清單小於或等於比較。

語法

```
template<typename Value>
bool operator<=(list<Value>% left,
    list<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個清單是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_list_operator_le.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (list) (STL/CLR)

列出相等的比較。

語法

```
template<typename Value>
bool operator==(list<Value>% left,
                 list<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true $i \leftarrow left[i] == right[i]$ 。您可以使用它來測試當兩個清單是依專案進行比較時，左邊是否以相同的順序排序。

範例

```
// cliest_list_operator_eq.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

運算子 > (list) (STL/CLR)

清單大於比較。

語法

```
template<typename Value>
bool operator>(list<Value>% left,
    list<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個清單是依專案進行比較時，是否要以左至

右順序排序。

範例

```
// cliext_list_operator_gt.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (list) (STL/CLR)

列出大於或等於比較。

語法

```
template<typename Value>
bool operator>=(list<Value>% left,
    list<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個清單是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_list_operator_ge.cpp
// compile with: /clr
#include <cliext/list>

int main()
{
    cliext::list<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::list<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
        c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

map (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `map` 來管理一連串的專案，做為 (近) 的節點數目，每個節點都會儲存一個元素。專案是由索引鍵(用來排序序列)和對應值所組成。

在下列描述中，與 `GValue` 相同：

```
Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>
```

其中：

`GKey` 與索引 鍵相同，除非後者是 ref 型別，在這種情況下，它是 `Key^`

`GMapped` 與 對應相同，除非後者是 ref 型別，在這種情況下，它是 `Mapped^`

語法

```
template<typename Key,
         typename Mapped>
ref class map
: public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
System::Collections::Generic::IDictionary<Gkey, GMapped>,
Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

映射

受控制序列中元素的其他元件類型。

需求

標頭：`<cliext/map>`

命名空間：`cliext`

宣告

宣告	說明
<code>map::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>map::const_reference (STL/CLR)</code>	項目的常數參考類型。

map::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
map::difference_type (STL/CLR)	(的類型可能簽署兩個專案之間的) 距離。
map::generic_container (STL/CLR)	容器的泛型介面型別。
map::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
map::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
map::generic_value (STL/CLR)	容器之泛型介面的元素類型。
map::iterator (STL/CLR)	受控制序列之迭代器的類型。
map::key_compare (STL/CLR)	兩個索引鍵的排序委派。
map::key_type (STL/CLR)	排序索引鍵的類型。
map::mapped_type (STL/CLR)	與每個索引鍵相關聯之對應值的型別。
map::reference (STL/CLR)	項目的參考類型。
map::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
map::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
map::value_compare (STL/CLR)	兩個元素值的排序委派。
map::value_type (STL/CLR)	項目的類型。

map::begin (STL/CLR)	指定受控制序列的開頭。
map::clear (STL/CLR)	移除所有項目。
map::count (STL/CLR)	計算符合指定索引鍵的元素。
map::empty (STL/CLR)	測試項目是否不存在。
map::end (STL/CLR)	指定受控制序列的結尾。
map::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
map::erase (STL/CLR)	移除位於指定位置的項目。
map::find (STL/CLR)	尋找符合指定之索引鍵的元素。
map::insert (STL/CLR)	加入項目。

map::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。
map::lower_bound (STL/CLR)	尋找符合指定索引鍵的範圍開頭。
map::make_value (STL/CLR)	結構值物件。
map::map (STL/CLR)	建構容器物件。
map::rbegin (STL/CLR)	指定反向受控制序列的開頭。
map::rend (STL/CLR)	指定反向受控制序列的結尾。
map::size (STL/CLR)	計算元素的數目。
map::swap (STL/CLR)	交換兩個容器的內容。
map::to_array (STL/CLR)	將受控制序列複製到新的陣列。
map::upper_bound (STL/CLR)	尋找符合指定索引鍵的範圍結尾。
map::value_comp (STL/CLR)	針對兩個元素值複製順序委派。

map::operator= (STL/CLR)	取代受控制的序列。
map::operator (STL/CLR)	將索引鍵對應至其相關聯的對應值。
operator != (map) (STL/CLR)	判斷物件是否 map 不等於另一個 map 物件。
operator< (map) (STL/CLR)	判斷 map 物件是否小於另一個 map 物件。
operator<= (map) (STL/CLR)	判斷 map 物件是否小於或等於另一個 map 物件。
operator == (map) (STL/CLR)	判斷 map 物件是否等於另一個 map 物件。
operator> (map) (STL/CLR)	判斷 map 物件是否大於另一個 map 物件。
operator>= (map) (STL/CLR)	判斷 map 物件是否大於或等於另一個 map 物件。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。

<code>IEnumerable<T></code>	透過具類型的元素排序。
<code>ICollection<T></code>	維護具類型的元素群組。
<code>IDictionary< TKey, TValue ></code>	維護 {key, value} 組的群組。
<code>ITree<索引鍵, 值></code>	維護泛型容器。

備註

物件會針對其控制為個別節點的序列，配置和釋出儲存體。它會將元素插入 (近) 平衡的樹狀結構中，藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式來保持排序。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `map:: key_compare` 類型的預存委派物件來排序它所控制的序列，([STL/CLR](#))。當您建立對應時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<(key_type, key_type)`。您可以藉由呼叫成員函式`map:: key_comp (STL/CLR)`來存取這個儲存的物件 ()。

這類委派物件必須在類型 `map:: key_type (STL/CLR)` 的索引鍵上強制執行嚴格弱式排序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y)` 是 true，則 `key_comp()(y, x)` 必須為 false。

如果 `key_comp()(x, y)` 是 true，則 `x` 會被視為之前的排序 `y`。

如果 `!key_comp()(x, y) && !key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

若為 `x` `y` 受控制序列中的任何元素，`key_comp()(y, x)` 則為 false。(預設的委派物件，索引鍵的值永遠不會減少。) 與範本類別 **對應**不同，樣板類別的物件不需要所有專案的索引 `map` 鍵都是唯一的。(兩個以上的索引鍵可以有對等的順序。)

每個元素都包含個別的索引鍵和對應的值。序列的表示方式，可讓您查閱、插入和移除具有許多作業的任意元素，並以序列中專案數目的對數為比例，(對數時間)。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

地圖支援雙向反覆運算器，這表示您可以逐步執行指定受控制序列中專案的反覆運算器，以逐步執行連續的元素。特殊前端節點對應于`map:: end (STL/CLR)` 所傳回的反覆運算器 ()。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將地圖反覆運算器遞增以連接到前端節點，然後比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考其數位位置(需要隨機存取反覆運算器)的對應元素。

地圖反覆運算器會將控制碼儲存至其相關聯的地圖節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。Map iterator 會維持有效，只要其相關聯的對應節點與某個對應相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

map:: begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_map_begin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Mymap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]
```

map:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫`map:: erase (stl/clr) ()` `map:: begin (stl/clr) ()`，`map:: end (STL/clr) ()`。您可以使用它來確保受控制的序列是空的。

範例

```
// cliext_map_clear.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0
```

map:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 **T2**，可作為受控制序列的常數雙向反覆運算器。

範例

```

// cliext_map_const_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

map::const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_map_const_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
    {
        // get a const reference to an element
        Mymap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
    }
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
```

map:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_map_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymap::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("{0} {1}", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

map:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_map_count.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

map::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_map_difference_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymap::difference_type diff = 0;
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3

```

map::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[map::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試地圖是否為空的。

範例

```
// cliext_map_empty.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True
```

map:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_map_end.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Mymap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("*-- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*--end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

map:: equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(map:: lower_bound (stl/clr) (key), map:: upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_map_equal_range.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef Mymap::pair_iter_pair Pairii;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

map:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 `where` 所指向之受控制序列的專案，並傳回反覆運算器，指定移除專案之後的第一個元素，或 `map::end (STL/CLR)` (`()` 如果沒有這樣的元素的話)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍 `[,)` 中受控制序列的元素，`first` `last` 並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或 `end()` 如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引 鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_map_erase.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    cliext::map<wchar_t, int> c1;
    c1.insert(cliext::map<wchar_t, int>::make_value(L'a', 1));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'b', 2));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::map<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::map<wchar_t, int>::make_value(L'd', 4));
    c1.insert(cliext::map<wchar_t, int>::make_value(L'e', 5));
    for each (cliext::map<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

map:: find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[map:: end \(STL/CLR\)](#)。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_map_find.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Mymap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

map:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```
// cliext_map_generic_container.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Mymap::make_value(L'd', 4));
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Mymap::make_value(L'e', 5));
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

map:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```
// cliext_map_generic_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymap::generic_iterator gcit = gc1->begin();
    Mymap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

map:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```
// cliext_map_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/map>  
  
typedef cliext::map<wchar_t, int> Mymap;  
int main()  
{  
    Mymap c1;  
    c1.insert(Mymap::make_value(L'a', 1));  
    c1.insert(Mymap::make_value(L'b', 2));  
    c1.insert(Mymap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Mymap::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Mymap::generic_container^ gc1 = %c1;  
    for each (Mymap::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Mymap::generic_reverse_iterator gcit = gc1->rbegin();  
    Mymap::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

map:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_map_generic_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymap::generic_container^ gc1 = %c1;
    for each (Mymap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymap::generic_iterator gcit = gc1->begin();
    Mymap::generic_value gcval = *gcit;
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

map:: insert (STL/CLR)

加入項目。

語法

```

cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會致力於插入具有值 `val` 的元素，並傳回一對值 `x`。如果 `x.second` 為 `true`，會 `x.first` 指定新插入的專案，否則會 `x.first` 指定具有對等順序的專案，而該專案已存在，且不會插入新的元素。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 `val` 的元素，並使用 `where` 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [`first` , `last`) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_map_insert.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
typedef Mymap::pair_iter_bool Pairib;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    // insert a single value, success and failure
    Pairib pair1 = c1.insert(Mymap::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    pair1 = c1.insert(Mymap::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}] {2}",
        pair1.first->first, pair1.first->second, pair1.second);

    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    Mymap::iterator it =
        c1.insert(c1.begin(), Mymap::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Mymap c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Mymap c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
            IEnumerable<Mymap::value_type>)c1);
    for each (Mymap::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [[x 24] True]
insert([L'b' 2]) = [[b 2] False]
[a 1] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [c 3] [x 24]
[a 1] [b 2] [c 3] [x 24]
[a 1] [b 2] [c 3] [x 24] [y 25]
```

map:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_map_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("[{0} {1}] ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

map:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_map_key_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

map:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_map_key_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

map::key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```

// cliext_map_key_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymap::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

map::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，其對索引 *x* 鍵具有對等的排序。如果沒有這類元素，則會傳回 [map::end \(STL/CLR\)](#) *()*；否則會傳回指定的 iterator *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```

// cliext_map_lower_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Mymap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]

```

map:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key, mapped_type mapped);
```

參數

key

要使用的索引鍵值。

已對應

要搜尋的對應值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵且對應值為 對應的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```

// cliext_map_make_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

map:: map (STL/CLR)

建構容器物件。

語法

```

map();
explicit map(key_compare^ pred);
map(map<Key, Mapped>% right);
map(map<Key, Mapped>^ right);
template<typename InIter>
    mapmap(InIter first, InIter last);
template<typename InIter>
    map(InIter first, InIter last,
        key_compare^ pred);
map(System::Collections::Generic::IEnumerable<GValue>^ right);
map(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

`map();`

使用預設順序述詞，初始化沒有元素的受控制序列 `key_compare()`。您可以使用它來指定空的初始受控制序列，

以及預設順序述詞。

函數：

```
explicit map(key_compare^ pred);
```

使用順序述詞 *pred*, 初始化沒有元素的受控制序列。您可以使用它來指定空的初始受控制序列, 以及指定的順序述詞。

函數：

```
map(map<Key, Mapped>% right);
```

使用順序 [*right.begin()* , *right.end()*), 以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列, 這是由 *map* 物件 許可權所控制之序列的複本, 以及預設順序述詞。

函數：

```
map(map<Key, Mapped>^ right);
```

使用順序 [*right->begin()* , *right->end()*), 以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列, 這是由 *map* 物件 許可權所控制之序列的複本, 以及預設順序述詞。

函數：

```
template<typename InIter> map(InIter first, InIter last);
```

使用順序 [*first* , *last*), 以預設順序述詞初始化受控制的序列。您可以使用它, 以預設順序述詞, 讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> map(InIter first, InIter last, key_compare^ pred);
```

使用序列 [*first* , *last*), 以順序述詞 *pred* 初始化受控制的序列。您可以使用它, 利用指定的順序述詞, 讓受控制的序列成為另一個序列的複本。

函數：

```
map(System::Collections::Generic::IEnumerable<Key>^ right);
```

以列舉值 右邊指定的順序, 使用預設順序述詞, 初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本, 以及預設順序述詞。

函數：

```
map(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用由列舉值 右邊指定的序列, 並搭配順序述詞 *pred*, 初始化受控制的序列。您可以使用它, 透過指定的順序述詞, 讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```
// cliext_map_construct.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
// construct an empty container
    Mymap c1;
    System::Console::WriteLine("size() = {0}", c1.size());
```

```

c1.insert(Myimap::make_value(L'a', 1));
c1.insert(Myimap::make_value(L'b', 2));
c1.insert(Myimap::make_value(L'c', 3));
for each (Myimap::value_type elem in c1)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an ordering rule
Myimap c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (Myimap::value_type elem in c2)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range
Myimap c3(c1.begin(), c1.end());
for each (Myimap::value_type elem in c3)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myimap c4(c1.begin(), c1.end(),
          cliext::greater_equal<wchar_t>());
for each (Myimap::value_type elem in c4)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration
Myimap c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myimap::value_type>^)%c3);
for each (Myimap::value_type elem in c5)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myimap c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Myimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Myimap::value_type elem in c6)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying another container
Myimap c7(c4);
for each (Myimap::value_type elem in c7)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
Myimap c8(%c3);
for each (Myimap::value_type elem in c8)
    System::Console::Write("[{0} {1}] ", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
```

map:: mapped_type (STL/CLR)

與每個索引鍵關聯的對應值類型。

語法

```
typedef Mapped mapped_type;
```

備註

此類型與 [對應的範本參數](#)同義。

範例

```
// cliext_map_mapped_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents "[a 1] [b 2] [c 3]" using mapped_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
            Mymap::mapped_type val = it->second;

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
1 2 3
```

map:: operator = (STL/CLR)

取代受控制的序列。

語法

```
map<Key, Mapped>% operator=(map<Key, Mapped>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_map_operator_as.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

map:: operator (STL/CLR)

將索引鍵對應至其相關聯的對應值。

語法

```
mapped_type operator[](key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會致力於尋找對索引鍵具有對等排序的元素。如果找到一個，就會傳回相關聯的對應值；否則，它會插入 `value_type(key, mapped_type())` 並傳回關聯的（預設）對應值。您可以使用它來查詢對應的值，並指定其相關聯的索引鍵，或者，如果找不到任何索引鍵，則確定該索引鍵的專案是否存在。

範例

```

// cliext_map_operator_sub.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("c1[{0}] = {1}",
        L'A', c1[L'A']);
    System::Console::WriteLine("c1[{0}] = {1}",
        L'b', c1[L'b']);

    // redisplay altered contents
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // alter mapped values and redisplay
    c1[L'A'] = 10;
    c1[L'c'] = 13;
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
c1[A] = 0
c1[b] = 2
[A 0] [a 1] [b 2] [c 3]
[A 10] [a 1] [b 2] [c 13]

```

map:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_map_rbegin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymap::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("*++rbegin() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]
```

map:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_map_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        {   // get a reference to an element
            Mymap::reference ref = *it;
            System::Console::Write("[{0} {1}] ", ref->first, ref->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

map:: rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 `Iterator` 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_map_rend.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine("*-- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("*--rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*-- --rend() = [b 2]
*--rend() = [a 1]
```

map:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_map_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} {1} ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

map::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[map::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_map_size.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Mymap::make_value(L'd', 4));
    c1.insert(Mymap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

map::size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_map_size_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymap::size_type diff = 0;
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3

```

map:: swap (STL/CLR)

交換兩個容器的內容。

語法

```

void swap(map<Key, Mapped>% right);

```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_map_swap.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymap c2;
    c2.insert(Mymap::make_value(L'd', 4));
    c2.insert(Mymap::make_value(L'e', 5));
    c2.insert(Mymap::make_value(L'f', 6));
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

map:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_map_to_array.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Mymap::value_type>^ a1 = c1.to_array();

    c1.insert(Mymap::make_value(L'd', 4));
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Mymap::value_type elem in a1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]

```

map::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```

iterator upper_bound(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，其對索引 *x* 鍵具有對等的順序。如果不存在這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 [map::END \(STL/CLR\)](#) ()；否則會傳回指定第一個元素的反覆運算器 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_map_upper_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Mymap::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

map::value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```
value_compare^ value_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```
// cliext_map_value_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'b', 2),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

map::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```
Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;
```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```
// cliext_map_value_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    Mymap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymap::make_value(L'a', 1),
            Mymap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymap::make_value(L'b', 2),
            Mymap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

map::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```

// cliext_map_value_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Mymap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymap::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

operator != (map) (STL/CLR)

清單不等於比較。

語法

```

template<typename Key,
         typename Mapped>
bool operator!=(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個地圖依元素進行比較時，左邊是否未以正確的順序排序。

範例

```

// cliext_map_operator_ne.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

operator < (map) (STL/CLR)

清單小於比較。

語法

```

template<typename Key,
         typename Mapped>
bool operator<(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 i $!(right[i] < left[i])$ 而言也是 true $left[i] < right[i]$ 。

否則，它會傳回 `left->size() < right->size()` 您使用它來測試當兩個地圖`right`是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_map_operator_lt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (map) (STL/CLR)

清單小於或等於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator<=(map<Key, Mapped>% left,
                 map<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個地圖是依專案進行比較時，是否不會排序左。

範例

```
// cliext_map_operator_le.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (map) (STL/CLR)

列出相等的比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator==(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true `i left[i] == right[i]`。您可以使用它來測試當兩個地圖依元素進行比較時，左邊是否以相同的順序排序。

範例

```
// cliext_map_operator_eq.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

operator > (map) (STL/CLR)

清單大於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator>(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right` < `left`。您可以使用它來測試當兩個地圖依元素進行比較時，是否要將左方排序。

範例

```
// cliext_map_operator_gt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (map) (STL/CLR)

列出大於或等於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator>=(map<Key, Mapped>% left,
                  map<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個地圖是依專案進行比較時，是否要將 *left* 排在 *右邊*。

範例

```
// cliext_map_operator_ge.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymap;
int main()
{
    Mymap c1;
    c1.insert(Mymap::make_value(L'a', 1));
    c1.insert(Mymap::make_value(L'b', 2));
    c1.insert(Mymap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymap c2;
    c2.insert(Mymap::make_value(L'a', 1));
    c2.insert(Mymap::make_value(L'b', 2));
    c2.insert(Mymap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

multimap (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `multimap` 來管理一連串的專案，做為(近)的節點數目，每個節點都會儲存一個元素。專案是由索引鍵(用來排序序列)和對應值所組成。

在下列描述中，與 `GValue` 相同：

```
Microsoft::VisualC::StlClr::GenericPair<GKey, GMapped>
```

其中：

`GKey` 與索引 鍵相同，除非後者是 ref 型別，在這種情況下，它是 `Key^`

`GMapped` 與 對應相同，除非後者是 ref 型別，在這種情況下，它是 `Mapped^`

語法

```
template<typename Key,
         typename Mapped>
ref class multimap
{
    public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

映射

受控制序列中元素的其他元件類型。

需求

標頭：`<cliext/map>`

命名空間：`cliext`

宣告

████	████
<code>multimap::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>multimap::const_reference (STL/CLR)</code>	項目的常數參考類型。

multimap::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
multimap::difference_type (STL/CLR)	(的類型可能簽署兩個專案之間的) 距離。
multimap::generic_container (STL/CLR)	容器的泛型介面型別。
multimap::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
multimap::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
multimap::generic_value (STL/CLR)	容器之泛型介面的元素類型。
multimap::iterator (STL/CLR)	受控制序列之迭代器的類型。
multimap::key_compare (STL/CLR)	兩個索引鍵的排序委派。
multimap::key_type (STL/CLR)	排序索引鍵的類型。
multimap::mapped_type (STL/CLR)	與每個索引鍵相關聯之對應值的型別。
multimap::reference (STL/CLR)	項目的參考類型。
multimap::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
multimap::size_type (STL/CLR)	兩個元素之間 (非負) 距離的型別。
multimap::value_compare (STL/CLR)	兩個元素值的排序委派。
multimap::value_type (STL/CLR)	項目的類型。
multimap::begin (STL/CLR)	指定受控制序列的開頭。
multimap::clear (STL/CLR)	移除所有項目。
multimap::count (STL/CLR)	計算符合指定索引鍵的元素。
multimap::empty (STL/CLR)	測試項目是否不存在。
multimap::end (STL/CLR)	指定受控制序列的結尾。
multimap::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
multimap::erase (STL/CLR)	移除位於指定位置的項目。
multimap::find (STL/CLR)	尋找符合指定之索引鍵的元素。
multimap::insert (STL/CLR)	加入項目。

multimap::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。
multimap::lower_bound (STL/CLR)	尋找符合指定索引鍵的範圍開頭。
multimap::make_value (STL/CLR)	結構值物件。
multimap::multimap (STL/CLR)	建構容器物件。
multimap::rbegin (STL/CLR)	指定反向受控制序列的開頭。
multimap::rend (STL/CLR)	指定反向受控制序列的結尾。
multimap::size (STL/CLR)	計算元素的數目。
multimap::swap (STL/CLR)	交換兩個容器的內容。
multimap::to_array (STL/CLR)	將受控制序列複製到新的陣列。
multimap::upper_bound (STL/CLR)	尋找符合指定索引鍵的範圍結尾。
multimap::value_comp (STL/CLR)	針對兩個元素值複製順序委派。

multimap::operator= (STL/CLR)	取代受控制的序列。
operator!= (multimap) (STL/CLR)	判斷物件是否 <code>multimap</code> 不等於另一個 <code>multimap</code> 物件。
operator< (multimap) (STL/CLR)	判斷 <code>multimap</code> 物件是否小於另一個 <code>multimap</code> 物件。
operator<= (multimap) (STL/CLR)	判斷 <code>multimap</code> 物件是否小於或等於另一個 <code>multimap</code> 物件。
operator == (multimap) (STL/CLR)	判斷 <code>multimap</code> 物件是否等於另一個 <code>multimap</code> 物件。
operator> (multimap) (STL/CLR)	判斷 <code>multimap</code> 物件是否大於另一個 <code>multimap</code> 物件。
operator>= (multimap) (STL/CLR)	判斷 <code>multimap</code> 物件是否大於或等於另一個 <code>multimap</code> 物件。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。

<code>IEnumerable<T></code>	透過具類型的元素排序。
<code>ICollection<T></code>	維護具類型的元素群組。
<code>ITree<Key, Value></code>	維護泛型容器。

備註

物件會針對其控制為個別節點的序列，配置和釋出儲存體。它會將元素插入（近）平衡的樹狀結構中，藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式來保持排序。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `multimap::key_compare` 類型的預存委派物件來排序它所控制的序列，([STL/CLR](#))。當您建立 `multimap` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<(key_type, key_type)`。您可以藉由呼叫成員函式 `multimap::key_comp` ([STL/CLR](#)) 來存取這個儲存的物件 `()`。

這類委派物件必須在 `multimap::key_type` 類型的索引鍵上強制執行嚴格弱式排序，([STL/CLR](#))。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y)` 是 true，則 `key_comp()(y, x)` 必須為 false。

如果 `key_comp()(x, y)` 是 true，則 `x` 會被視為之前的排序 `y`。

如果 `!key_comp()(x, y) && !key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

若為 `x` `y` 受控制序列中的任何元素，`key_comp()(y, x)` 則為 false。(預設的委派物件，索引鍵的值永遠不會減少。) 與樣板類別 [對應](#) ([STL/CLR](#)) 不同，樣板類別的物件不 `multimap` 需要所有元素的索引鍵都是唯一的。(兩個以上的索引鍵可以有對等的順序。)

每個元素都包含個別的索引鍵和對應的值。序列的表示方式，可讓您查閱、插入和移除具有許多作業的任意元素，並以序列中專案數目的對數為比例，(對數時間)。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

`Multimap` 支援雙向反覆運算器，這表示您可以使用反覆運算器逐步執行連續的元素，以指定受控制序列中的元素。特殊的前端節點對應至 `multimap::end` ([STL/CLR](#)) 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將 `multimap` 反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考 `multimap` 元素的數位位置，而這需要隨機存取反覆運算器。

`Multimap` 反覆運算器會將控制碼儲存至其相關聯的 `multimap` 節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。`Multimap` 反覆運算器會維持有效，只要其相關聯的 `multimap` 節點與某些 `multimap` 相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的 容器不會摧毀其元素。

成員

`multimap::begin` ([STL/CLR](#))

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_multimap_begin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items
    Mymultimap::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*++begin() = [{0} {1}]",
        it->first, it->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*begin() = [a 1]
*++begin() = [b 2]
```

multimap:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫 `multimap:: erase (stl/clr) ()` `multimap:: begin (stl/clr) ()`, `multimap:: end (STL/clr) ()`。您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_multimap_clear.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));

    // display contents " [a 1] [b 2]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0
[a 1] [b 2]
size() = 0

```

multimap:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數雙向反覆運算器。

範例

```

// cliext_multimap_const_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("[{0} {1}] ", cit->first, cit->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

multimap:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_multimap_const_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
    {
        // get a const reference to an element
        Mymultimap::const_reference cref = *cit;
        System::Console::Write("[{0} {1}] ", cref->first, cref->second);
    }
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
```

multimap:: const_reverse_iterator (STL/CLR)

用於受控制序列的常數反向迭代器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_multimap_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymultimap::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("[{0} {1}] ", crit->first, crit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

multimap:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引 鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```

// cliext_multimap_count.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}

```

```

[a 1] [b 2] [c 3]
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0

```

multimap::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_multimap_difference_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymultimap::difference_type diff = 0;
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymultimap::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
end()-begin() = 3
begin()-end() = -3

```

multimap::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[multimap::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 multimap 是否是空的。

範例

```

// cliext_multimap_empty.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 3
empty() = False
size() = 0
empty() = True

```

multimap::end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_multimap_end.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect last two items
    Mymultimap::iterator it = c1.end();
    --it;
    --it;
    System::Console::WriteLine("*-- --end() = [{0} {1}]",
        it->first, it->second);
    ++it;
    System::Console::WriteLine("*--end() = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*-- --end() = [b 2]
*--end() = [c 3]

```

multimap::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

pair_iter_iter equal_range(key_type _Keyval);

```

參數

_Keyval

要搜尋的索引鍵值。

備註

方法會傳回一對反覆運算器 `- multimap::lower_bound (stl/clr) (_Keyval), multimap::upper_bound (stl/clr) (_Keyval)`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_multimap_equal_range.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
typedef Mymultimap::pair_iter iter Pairii;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("[{0} {1}] ",
            pair1.first->first, pair1.first->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
equal_range(L'x') empty = True
[b 2]

```

multimap:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
bool erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除所指向之受控制序列的專案，並傳回反覆運算器，指定移除專案之後的第一個專案，如果沒有這樣的元素，則會傳回`multimap::end (STL/CLR) ()`。您可以使用它來移除單一專案。

第二個成員函式會移除範圍`[,)`中受控制序列的元素，`first` `last` 並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或`end()` 如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_multimap_erase.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    cliext::multimap<wchar_t, int> c1;
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'a', 1));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'b', 2));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (cliext::multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase an element and reinspect
    cliext::multimap<wchar_t, int>::iterator it =
        c1.erase(c1.begin());
    System::Console::WriteLine("erase(begin()) = [{0} {1}]",
        it->first, it->second);

    // add elements and display " b c d e"
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'd', 4));
    c1.insert(cliext::multimap<wchar_t, int>::value_type(L'e', 5));
    for each (cliext::multimap<wchar_t, int>::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // erase all but end
    it = c1.end();
    it = c1.erase(c1.begin(), --it);
    System::Console::WriteLine("erase(begin(), end()-1) = [{0} {1}]",
        it->first, it->second);
    System::Console::WriteLine("size() = {0}", c1.size());

    // erase end
    System::Console::WriteLine("erase(L'x') = {0}", c1.erase(L'x'));
    System::Console::WriteLine("erase(L'e') = {0}", c1.erase(L'e'));
    return (0);
}
```

```
[a 1] [b 2] [c 3]
erase(begin()) = [b 2]
[b 2] [c 3] [d 4] [e 5]
erase(begin(), end()-1) = [e 5]
size() = 1
erase(L'x') = 0
erase(L'e') = 1
```

multimap:: find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回[multimap:: end \(STL/CLR\)](#) ()。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_multimap_find.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1} ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());

    Mymultimap::iterator it = c1.find(L'b');
    System::Console::WriteLine("find {0} = [{1} {2}]",
        L'b', it->first, it->second);

    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
[a 1] [b 2] [c 3]
find A = False
find b = [b 2]
find C = False
```

multimap:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
    generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```
// cliext_multimap_generic_container.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(Mymultimap::make_value(L'd', 4));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(Mymultimap::make_value(L'e', 5));
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3] [d 4] [e 5]
```

multimap:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```
// cliext_multimap_generic_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymultimap::generic_iterator gcit = gc1->begin();
    Mymultimap::generic_value gcval = *gcit;
    System::Console::Write("[{0} {1}] ", gcval->first, gcval->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]
```

multimap:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```
typedef Microsoft::VisualC::StlClr::Generic::  
    ReverseRandomAccessIterator<generic_value>  
    generic_reverse_iterator;
```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```
// cliext_multimap_generic_reverse_iterator.cpp  
// compile with: /clr  
#include <cliext/map>  
  
typedef cliext::multimap<wchar_t, int> Mymultimap;  
int main()  
{  
    Mymultimap c1;  
    c1.insert(Mymultimap::make_value(L'a', 1));  
    c1.insert(Mymultimap::make_value(L'b', 2));  
    c1.insert(Mymultimap::make_value(L'c', 3));  
  
    // display contents " [a 1] [b 2] [c 3]"  
    for each (Mymultimap::value_type elem in c1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // construct a generic container  
    Mymultimap::generic_container^ gc1 = %c1;  
    for each (Mymultimap::value_type elem in gc1)  
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);  
    System::Console::WriteLine();  
  
    // get an element and display it  
    Mymultimap::generic_reverse_iterator gcit = gc1->rbegin();  
    Mymultimap::generic_value gcval = *gcit;  
    System::Console::WriteLine("[{0} {1}] ", gcval->first, gcval->second);  
    return (0);  
}
```

```
[a 1] [b 2] [c 3]  
[a 1] [b 2] [c 3]  
[c 3]
```

multimap:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_multimap_generic_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct a generic container
    Mymultimap::generic_container^ gc1 = %c1;
    for each (Mymultimap::value_type elem in gc1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // get an element and display it
    Mymultimap::generic_iterator gcit = gc1->begin();
    Mymultimap::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} {1}", gcval->first, gcval->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
[a 1]

```

multimap:: insert (STL/CLR)

加入項目。

語法

```

iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會插入具有值 va 的元素，並傳回指定新插入之元素的反覆運算器。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 va 的元素，並使用 *where* 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [`first` , `last`) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_multimap_insert.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Mymultimap::iterator it =
        c1.insert(Mymultimap::make_value(L'x', 24));
    System::Console::WriteLine("insert([L'x' 24]) = [{0} {1}]",
        it->first, it->second);

    it = c1.insert(Mymultimap::make_value(L'b', 2));
    System::Console::WriteLine("insert([L'b' 2]) = [{0} {1}]",
        it->first, it->second);

    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert a single value with hint
    it = c1.insert(c1.begin(), Mymultimap::make_value(L'y', 25));
    System::Console::WriteLine("insert(begin(), [L'y' 25]) = [{0} {1}]",
        it->first, it->second);
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an iterator range
    Mymultimap c2;
    it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // insert an enumeration
    Mymultimap c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::
            IEnumerable<Mymultimap::value_type>^)%c1);
    for each (Mymultimap::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
insert([L'x' 24]) = [x 24]
insert([L'b' 2]) = [b 2]
[a 1] [b 2] [b 2] [c 3] [x 24]
insert(begin(), [L'y' 25]) = [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
[a 1] [b 2] [b 2] [c 3] [x 24] [y 25]
```

multimap:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_multimap_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} {1} ", it->first, it->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

multimap:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_multimap_key_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

multimap:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_multimap_key_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultimap c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

multimap:: key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_multimap_key_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using key_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymultimap::key_type val = it->first;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

multimap::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，其對索引 *x* 鍵具有對等的排序。如果沒有這類元素，則會傳回 [multimap::end \(STL/CLR\)](#) ()；否則會傳回指定的 iterator *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```

// cliext_multimap_lower_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    Mymultimap::iterator it = c1.lower_bound(L'a');
    System::Console::WriteLine("*lower_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.lower_bound(L'b');
    System::Console::WriteLine("*lower_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
lower_bound(L'x') == end() = True
*lower_bound(L'a') = [a 1]
*lower_bound(L'b') = [b 2]

```

multimap:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key, mapped_type mapped);
```

參數

key

要使用的索引鍵值。

已對應

要搜尋的對應值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵 且對應值為 對應的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```

// cliext_multimap_make_value.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

multimap:: mapped_type (STL/CLR)

與每個索引鍵關聯的對應值類型。

語法

```
typedef Mapped mapped_type;
```

備註

此類型與 **對應**的範本參數同義。

範例

```

// cliext_multimap_mapped_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using mapped_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in mapped_type object
        Mymultimap::mapped_type val = it->second;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

1 2 3

multimap:: multimap (STL/CLR)

建構容器物件。

語法

```
multimap();
explicit multimap(key_compare^ pred);
multimap(multimap<Key, Mapped>% right);
multimap(multimap<Key, Mapped>^ right);
template<typename InIter>
    multimap(multimap(InIter first, InIter last));
template<typename InIter>
    multimap(InIter first, InIter last,
        key_compare^ pred);
multimap(System::Collections::Generic::IEnumerable<GValue>^ right);
multimap(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);
```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
multimap();
```

使用預設順序述詞，初始化沒有元素的受控制序列 `key_compare()`。您可以使用它來指定空的初始受控制序列，以及預設順序述詞。

函數：

```
explicit multimap(key_compare^ pred);
```

使用順序述詞 *pred*，初始化沒有元素的受控制序列。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞。

函數：

```
multimap(multimap<Key, Mapped>% right);
```

使用順序 [`right.begin()` , `right.end()`)，以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `multimap` 物件 許可權所控制之序列的複本(具有預設順序述詞)。

函數：

```
multimap(multimap<Key, Mapped>^ right);
```

使用順序 [`right->begin()` , `right->end()`)，以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列，這是由 `multimap` 物件 許可權所控制之序列的複本(具有預設順序述詞)。

函數：

```
template<typename InIter> multimap(InIter first, InIter last);
```

使用順序 [`first` , `last`)，以預設順序述詞初始化受控制的序列。您可以使用它，以預設順序述詞，讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> multimap(InIter first, InIter last, key_compare^ pred);
```

使用序列 [`first` , `last`)，以順序述詞 `pred` 初始化受控制的序列。您可以使用它，利用指定的順序述詞，讓受控制的序列成為另一個序列的複本。

函數：

```
multimap(System::Collections::Generic::IEnumerable<Key>^ right);
```

以列舉值 右邊指定的順序，使用預設順序述詞，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本，以及預設順序述詞。

函數：

```
multimap(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用由列舉值 右邊指定的序列，並搭配順序述詞 `pred`，初始化受控制的序列。您可以使用它，透過指定的順序述詞，讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```
// cliext_multimap_construct.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    // construct an empty container
    Mymultimap c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an ordering rule
    Mymultimap c2 = cliext::greater_equal<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    c2.insert(c1.begin(), c1.end());
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an iterator range
    Mymultimap c3(c1.begin(), c1.end());
    for each (Mymultimap::value_type elem in c3)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // construct with an iterator range and an ordering rule
    Mymultimap c4(c1.begin(), c1.end(),
        cliext::greater_equal<wchar_t>());
    for each (Mymultimap::value_type elem in c4)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
```

```

System::Console::WriteLine();

// construct with an enumeration
Mymultimap c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Mymultimap::value_type>^)%c3);
for each (Mymultimap::value_type elem in c5)
    System::Console::Write("{0} {1}\n", elem->first, elem->second);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Mymultimap c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<
        Mymultimap::value_type>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (Mymultimap::value_type elem in c6)
    System::Console::Write("{0} {1}\n", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying another container
Mymultimap c7(c4);
for each (Mymultimap::value_type elem in c7)
    System::Console::Write("{0} {1}\n", elem->first, elem->second);
System::Console::WriteLine();

// construct by copying a container handle
My multimap c8(%c3);
for each (My multimap::value_type elem in c8)
    System::Console::Write("{0} {1}\n", elem->first, elem->second);
System::Console::WriteLine();
return (0);
}

```

```

size() = 0
[a 1] [b 2] [c 3]
size() = 0
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]
[c 3] [b 2] [a 1]
[c 3] [b 2] [a 1]
[a 1] [b 2] [c 3]

```

multimap:: operator = (STL/CLR)

取代受控制的序列。

語法

```
multimap<Key, Mapped>% operator=(multimap<Key, Mapped>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```

// cliext_multimap_operator_as.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2 = c1;
    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [c 3]
```

multimap::rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_multimap_rbegin.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultimap::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("*++rbegin() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
*rbegin() = [c 3]
*++rbegin() = [b 2]
```

multimap:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_multimap_reference.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    Mymultimap::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Mymultimap::reference ref = *it;
        System::Console::Write("[{0} {1}] ", ref->first, ref->second);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
[a 1] [b 2] [c 3]
```

multimap::rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_multimap_rend.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultimap::reverse_iterator rit = c1.rend();
    --rit;
    --rit;
    System::Console::WriteLine("*-- --rend() = [{0} {1}]",
        rit->first, rit->second);
    ++rit;
    System::Console::WriteLine("*--rend() = [{0} {1}]",
        rit->first, rit->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
*-- --rend() = [b 2]
*--rend() = [a 1]

```

multimap:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_multimap_reverse_iterator.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" reversed
    Mymultimap::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("[{0} {1}] ", rit->first, rit->second);
    System::Console::WriteLine();
    return (0);
}
```

```
[c 3] [b 2] [a 1]
```

multimap::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[multimap::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_multimap_size.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(Mymultimap::make_value(L'd', 4));
    c1.insert(Mymultimap::make_value(L'e', 5));
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

[a 1] [b 2] [c 3]
size() = 0 after clearing
size() = 2 after adding 2

```

multimap:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_multimap_size_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // compute positive difference
    Mymultimap::size_type diff = 0;
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```
[a 1] [b 2] [c 3]
end()-begin() = 3
```

multimap:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(multimap<Key, Mapped>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_multimap_swap.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'd', 4));
    c2.insert(Mymultimap::make_value(L'e', 5));
    c2.insert(Mymultimap::make_value(L'f', 6));
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[d 4] [e 5] [f 6]
[d 4] [e 5] [f 6]
[a 1] [b 2] [c 3]

```

multimap:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_multimap_to_array.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // copy the container and modify it
    cli::array<Mymultimap::value_type>^ a1 = c1.to_array();

    c1.insert(Mymultimap::make_value(L'd', 4));
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (Mymultimap::value_type elem in a1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();
    return (0);
}

```

```
[a 1] [b 2] [c 3] [d 4]
[a 1] [b 2] [c 3]
```

multimap::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```
iterator upper_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，其對索引 x 鍵具有對等的順序。如果沒有這類專案，或 x 為受控制序列中的最後一個專案，則會傳回 [multimap::END \(STL/CLR\)](#) $()$ ；否則會傳回反覆運算器，指定超過的第一個元素 x 。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_multimap_upper_bound.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    Mymultimap::iterator it = c1.upper_bound(L'a');
    System::Console::WriteLine("*upper_bound(L'a') = [{0} {1}]",
        it->first, it->second);
    it = c1.upper_bound(L'b');
    System::Console::WriteLine("*upper_bound(L'b') = [{0} {1}]",
        it->first, it->second);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
upper_bound(L'x') == end() = True
*upper_bound(L'a') = [b 2]
*upper_bound(L'b') = [c 3]

```

multimap::value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```
value_compare^ value_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```
// cliext_multimap_value_comp.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'b', 2),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

multimap::value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```
Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;
```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```
// cliext_multimap_value_compare.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::map<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    Mymultimap::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare([L'a', 1], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine("compare([L'a', 1], [L'b', 2]) = {0}",
        kcomp(Mymultimap::make_value(L'a', 1),
            Mymultimap::make_value(L'b', 2)));
    System::Console::WriteLine("compare([L'b', 2], [L'a', 1]) = {0}",
        kcomp(Mymultimap::make_value(L'b', 2),
            Mymultimap::make_value(L'a', 1)));
    System::Console::WriteLine();
    return (0);
}
```

```
compare([L'a', 1], [L'a', 1]) = False
compare([L'a', 1], [L'b', 2]) = True
compare([L'b', 2], [L'a', 1]) = False
```

multimap::value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```

// cliext_multimap_value_type.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]" using value_type
    for (Mymultimap::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymultimap::value_type val = *it;
        System::Console::Write("[{0} {1}] ", val->first, val->second);
        }
    System::Console::WriteLine();
    return (0);
}

```

[a 1] [b 2] [c 3]

operator != (multimap) (STL/CLR)

清單不等於比較。

語法

```

template<typename Key,
         typename Mapped>
bool operator!=(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試兩個 multimap 是依元素進行比較時，左邊是否未與右方排序。

範例

```

// cliext_multimap_operator_ne.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

operator < (multimap) (STL/CLR)

清單小於比較。

語法

```

template<typename Key,
         typename Mapped>
bool operator<(multimap<Key, Mapped>% left,
                 multimap<Key, Mapped>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 i $!(right[i] < left[i])$ 而言也是 true $left[i] < right[i]$ 。否

則，它會傳回 `left->size() < right->size()` 您使用它來測試兩個 multimap 是依元素進行比較時，是否要在 *right* 之前排序 *left*。

範例

```
// cliext_multimap_operator_lt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (multimap) (STL/CLR)

清單小於或等於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator<=(multimap<Key, Mapped>% left,
                 multimap<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個 multimap 是依元素進行比較時，是否不會排序 `left`。

範例

```
// cliext_multimap_operator_le.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (multimap) (STL/CLR)

列出相等的比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator==(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true $i \leftarrow left[i] == right[i]$ 。您可以使用它來測試當兩個 multimap 是依元素進行比較時，左邊是否以正確的順序排序。

範例

```
// cliext_multimap_operator_eq.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("[{0} {1}] ", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

operator > (multimap) (STL/CLR)

清單大於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator>(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個 multimap 是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_multimap_operator_gt.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (multimap) (STL/CLR)

列出大於或等於比較。

語法

```
template<typename Key,
         typename Mapped>
bool operator>=(multimap<Key, Mapped>% left,
                  multimap<Key, Mapped>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個 multimap 是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_multimap_operator_ge.cpp
// compile with: /clr
#include <cliext/map>

typedef cliext::multimap<wchar_t, int> Mymultimap;
int main()
{
    Mymultimap c1;
    c1.insert(Mymultimap::make_value(L'a', 1));
    c1.insert(Mymultimap::make_value(L'b', 2));
    c1.insert(Mymultimap::make_value(L'c', 3));

    // display contents " [a 1] [b 2] [c 3]"
    for each (Mymultimap::value_type elem in c1)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    // assign to a new container
    Mymultimap c2;
    c2.insert(Mymultimap::make_value(L'a', 1));
    c2.insert(Mymultimap::make_value(L'b', 2));
    c2.insert(Mymultimap::make_value(L'd', 4));

    // display contents " [a 1] [b 2] [d 4]"
    for each (Mymultimap::value_type elem in c2)
        System::Console::Write("{0} {1}\n", elem->first, elem->second);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
[a 1] [b 2] [c 3]
[a 1] [b 2] [d 4]
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

multiset (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別所描述的物件可控制具有雙向存取之元素的變動長度序列。您可以使用容器 `multiset` 來管理一連串的專案，當做節點的(幾乎)平衡的已排序樹狀結構，每個專案都儲存一個元素。

在下面的描述中，與相同 `GValue` `GKey`，除非後者是 ref 型別(在 `Key` 此情況下為) `Key^`。

語法

```
template<typename Key>
ref class multiset
{
    public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ .... };
```

參數

索引鍵

受控制序列中項目的主要元件型別。

需求

標頭: <cliext/set>

命名空間: cliext

宣告

III	II
multiset::const_iterator (STL/CLR)	用於受控制序列的常數迭代器類型。
multiset::const_reference (STL/CLR)	項目的常數參考類型。
multiset::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
multiset::difference_type (STL/CLR)	兩個元素之間的(可能已簽署)距離的類型。
multiset::generic_container (STL/CLR)	容器的泛型介面類別型。
multiset::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
multiset::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
multiset::generic_value (STL/CLR)	容器之泛型介面的元素類型。

multiset::iterator (STL/CLR)	受控制序列之迭代器的類型。
multiset::key_compare (STL/CLR)	兩個索引鍵的排序委派。
multiset::key_type (STL/CLR)	排序索引鍵的類型。
multiset::reference (STL/CLR)	項目的參考類型。
multiset::reverse_iterator (STL/CLR)	受控制序列的反向迭代器類型。
multiset::size_type (STL/CLR)	兩個元素之間的(非負)距離類型。
multiset::value_compare (STL/CLR)	兩個元素值的排序委派。
multiset::value_type (STL/CLR)	項目的類型。

multiset::begin (STL/CLR)	指定受控制序列的開頭。
multiset::clear (STL/CLR)	移除所有項目。
multiset::count (STL/CLR)	計算符合指定索引鍵的元素。
multiset::empty (STL/CLR)	測試項目是否存在。
multiset::end (STL/CLR)	指定受控制序列的結尾。
multiset::equal_range (STL/CLR)	尋找符合指定之索引鍵的範圍。
multiset::erase (STL/CLR)	移除位於指定位置的項目。
multiset::find (STL/CLR)	尋找符合指定之索引鍵的元素。
multiset::insert (STL/CLR)	加入項目。
multiset::key_comp (STL/CLR)	複製兩個索引鍵的排序委派。
multiset::lower_bound (STL/CLR)	尋找符合指定之索引鍵的範圍開頭。
multiset::make_value (STL/CLR)	構造值物件。
multiset::multiset (STL/CLR)	建構容器物件。
multiset::rbegin (STL/CLR)	指定反向受控制序列的開頭。
multiset::rend (STL/CLR)	指定反向受控制序列的結尾。
multiset::size (STL/CLR)	計算元素的數目。

multiset::swap (STL/CLR)	交換兩個容器的內容。
multiset::to_array (STL/CLR)	將受控制序列複製到新的陣列。
multiset::upper_bound (STL/CLR)	尋找符合指定之索引鍵的結束範圍。
multiset::value_comp (STL/CLR)	複製兩個元素值的順序委派。
multiset::operator= (STL/CLR)	取代受控制的序列。
operator != (多重集) (STL/CLR)	判斷物件是否 <code>multiset</code> 不等於另一個 <code>multiset</code> 物件。
運算子< (多重集) (STL/CLR)	判斷 <code>multiset</code> 物件是否小於另一個 <code>multiset</code> 物件。
operator<= (多重集) (STL/CLR)	判斷 <code>multiset</code> 物件是否小於或等於另一個 <code>multiset</code> 物件。
operator == (多重集) (STL/CLR)	判斷 <code>multiset</code> 物件是否等於另一個 <code>multiset</code> 物件。
operator> (multiset) (STL/CLR)	判斷 <code>multiset</code> 物件是否大於另一個 <code>multiset</code> 物件。
operator>= (多重集) (STL/CLR)	判斷 <code>multiset</code> 物件是否大於或等於另一個 <code>multiset</code> 物件。

介面

ICloneable	複製物件。
IEnumerable	透過元素進行序列。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的專案進行序列。
ICollection<T>	維護具類型的元素群組。
ITree<Key, Value>	維護一般容器。

備註

物件會為其所控制的序列配置並釋出儲存體，以作為個別節點。它會將專案插入（幾乎）平衡的樹狀結構中，藉由改變節點間的連結來保持排序，而不是將某個節點的內容複寫到另一個節點。這表示您可以自由地插入和移除專案，而不會干擾其餘元素。

物件會藉由呼叫類型為 `多重集::key_compare (STL/CLR)` 的預存委派物件，來排序它所控制的序列。當您建立多重

集時，可以指定預存的委派物件；如果您沒有指定委派物件，預設值就是比較 `operator<(key_type, key_type)`。您可以藉由呼叫成員函式 [多重集:: key_comp \(STL/CLR\)](#) 來存取這個儲存的物件 `()`。

這類委派物件必須對 [多重集:: key_type \(STL/CLR\)](#) 類型的索引鍵強制執行嚴格弱式排序。這表示，對於任何兩個索引鍵 `x` 和 `y`：

`key_comp()(x, y)` 會在每次呼叫時傳回相同的布林值結果。

如果 `key_comp()(x, y)` 為 true，則 `key_comp()(y, x)` 必須為 false。

如果 `key_comp()(x, y)` 為 true，則 `x` 表示在之前排序過 `y`。

如果 `!key_comp()(x, y) && !key_comp()(y, x)` 為 true，則 `x` 和 `y` 會被視為具有對等的順序。

針對位於 `x` `y` 受控制序列中的任何專案，`key_comp()(y, x)` 為 false。（對於預設委派物件，索引鍵永遠不會減少值）。不同于樣板類別 [集 \(STL/CLR\)](#)，樣板類別的物件 `multiset` 不需要所有元素的索引鍵都是唯一的。（兩個或多個索引鍵可以具有對等的順序）。

每個元素都會同時做為安永和值。序列的表示方式，允許查閱、插入和移除具有數個作業的任意專案，並與序列中專案數的對數成正比（對數時間）。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

多重集支援雙向反覆運算器，這表示您可以逐步執行連續的專案，方法是指定以受控制序列中的元素。特殊的前端節點對應至 [多重集:: end \(STL/CLR\)](#) 傳回的反覆運算器 `()`。您可以遞減這個反覆運算器，使其到達受控制序列中的最後一個元素（如果有的話）。您可以遞增多重集反覆運算器，使其到達前端節點，然後再比較是否等於 `end()`。但是，您無法對所傳回的反覆運算器進行取值 `end()`。

請注意，您無法直接參考多重集元素的數值位置（需要隨機存取反覆運算器）。

多重集反覆運算器會儲存其相關聯集節點的控制碼，然後再將控制碼儲存至其相關聯的容器。您只能將反覆運算器與相關聯的容器物件搭配使用。多重集反覆運算器會保持有效，只要其相關聯的多重集節點與某些多重集相關聯即可。此外，有效的反覆運算器也是 dereferencable--您可以使用它來存取或更改其指定的元素值，只要它不等於即可 `end()`。

清除或移除元素會呼叫其預存值的析構函式。終結容器會清除所有元素。因此，其元素類型為 `ref` 類別的容器，可確保沒有任何元素 outlive 容器。不過要注意的是，控制碼容器並不會摧毀其元素。

成員

多重集:: begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

此成員函式會傳回雙向反覆運算器，指定受控制序列的第一個元素，或在空序列結尾以外的專案。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_multiset_begin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mymultiset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
*++begin() = b
```

多重集:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

此成員函式會有效地呼叫[多重集:: erase \(stl/clr\) \(\)](#)，[多重集:: begin \(stl/clr\) \(\)](#)，[多重集:: end \(stl/clr\) \(\)](#)。
您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_multiset_clear.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

多重集:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```
typedef T2 const_iterator;
```

備註

此類型描述未指定類型的物件 `T2`，可做為受控制序列的常數雙向反覆運算器。

範例

```
// cliext_multiset_const_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Mymultiset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

多重集:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

此類型描述專案的常數參考。

範例

```
// cliext_multiset_const_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Mymultiset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Mymultiset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

多重集:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

此類型描述未指定類型的物件 `T4`，可做為受控制序列的常數反向反覆運算器。

範例

```
// cliext_multiset_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Mymultiset::const_reverse_iterator crit = c1.rbegin();
    for (; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

多重集:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

此成員函式會傳回受控制序列中具有對等順序與索引鍵的元素數目。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```

// cliext_multiset_count.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}

```

```

a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0

```

多重集::difference_type (STL/CLR)

兩個元素之間帶正負號距離的類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能為負的元素計數。

範例

```

// cliext_multiset_difference_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mymultiset::difference_type diff = 0;
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Mymultiset::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

多重集:: empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[多重集:: size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試多重集是否為空的。

範例

```
// cliext_multiset_empty.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

多重集:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

此成員函式會傳回指向受控制序列結尾之外的雙向反覆運算器。您可以使用它來取得反覆運算器，以指定受控制序列的結尾；如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_multiset_end.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Mymultiset::iterator it = c1.end();
    --it;
    System::Console::WriteLine("!-- --end() = {0}", *--it);
    System::Console::WriteLine("!--end() = {0}", *++it);
    return (0);
}

```

```

a b c
!-- --end() = b
!--end() = c

```

多重集:: equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

此成員函式會傳回一對反覆運算器 `cliext::pair<iterator, iterator>(多重集:: lower_bound (stl/clr) (key), 多重集:: upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中符合指定索引鍵的元素範圍。

範例

```

// cliext_multiset_equal_range.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
typedef Mymultiset::pair_iter_iter Pairii;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

多重集:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
size_type erase(key_type key)

```

參數

first

要清除之範圍的開頭。

key

要清除的機碼值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除所指向之受控制序列的專案，並傳回反覆運算器，指定移除的專案之後剩餘的第一個元素。

或如果沒有這類元素，則傳回多重集:: end (STL/CLR) ()。您可以使用它來移除單一元素。

第二個成員函式會移除範圍 [,)中受控制序列的專案 first last，並傳回反覆運算器，指定移除任何元素之後剩餘的第一個元素，或 end() 如果沒有這類元素存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除受控制序列中的任何專案，其索引鍵對索引鍵具有對等的順序，並傳回已移除的元素數計數。您可以使用它來移除和計算符合指定索引鍵的所有元素。

每個專案抹除的時間會與受控制序列中專案數的對數成正比。

範例

```
// cliext_multiset_erase.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Mymultiset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

多重集:: find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有對等的順序，則成員函式會傳回指定其中一個元素的反覆運算器；否則，它會傳回多重集::end (STL/CLR) `()`。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_multiset_find.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

多重集::generic_container (STL/CLR)

容器的泛型介面類別型。

語法

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
    generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_multiset_generic_container.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

多重集:: generic_iterator (STL/CLR)

用於容器之泛型介面的反覆運算器類型。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

備註

此類型描述的泛型反覆運算器可與此樣板容器類別的泛型介面搭配使用。

範例

```

// cliext_multiset_generic_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_iterator gcit = gc1->begin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

多重集:: generic_reverse_iterator (STL/CLR)

要與容器的泛型介面搭配使用的反向反覆運算器類型。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

備註

此類型描述的泛型反向反覆運算器可與此樣板容器類別的泛型介面搭配使用。

範例

```

// cliext_multiset_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_reverse_iterator gcit = gc1->rbegin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

多重集:: generic_value (STL/CLR)

要與容器的泛型介面搭配使用之元素的類型。

語法

```

typedef GValue generic_value;

```

備註

此類型描述類型的物件 `GValue`，其描述要與這個樣板容器類別的泛型介面搭配使用的預存元素值。

範例

```

// cliext_multiset_generic_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mymultiset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Mymultiset::generic_iterator gcit = gc1->begin();
    Mymultiset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

多重集:: insert (STL/CLR)

加入項目。

語法

```

iterator insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

再

要插入的列舉。

初始值

要插入的機碼值。

where

在容器中要插入的位置(僅提示)。

備註

每個成員函式都會插入由其餘運算元所指定的序列。

第一個成員函式會插入具有值 va 的元素，並傳回反覆運算器，指定新插入的專案。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 va 的元素，並使用 $where$ 做為提示(以改善效能)，並傳回反覆運算器，指定新插入的專案。您可以使用它來插入單一專案，這可能會與您知道的元素相鄰。

第三個成員函式會插入序列 [`first` , `last`)。您可以使用它來插入從另一個序列複製的零個或多個元素。

第四個成員函式會插入右邊指定的序列。您可以使用它來插入列舉值所描述的序列。

每個專案插入所花的時間，會與受控制序列中專案數的對數成正比。不過，若指定的提示會指定插入點旁邊的元素，則插入可能會在分攤的常數時間內發生。

範例

```

// cliext_multiset_insert.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    System::Console::WriteLine("insert(L'x') = {0}",
        *c1.insert(L'x'));

    System::Console::WriteLine("insert(L'b') = {0}",
        *c1.insert(L'b'));

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Mymultiset c2;
    Mymultiset::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Mymultiset c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = x
insert(L'b') = b
a b b c x
insert(begin(), L'y') = y
a b b c x y
a b b c x
a b b c x y

```

多重集:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

此類型描述未指定類型的物件 `T1`，可做為受控制序列的雙向反覆運算器。

範例

```
// cliext_multiset_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Mymultiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

多重集:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

此成員函式會傳回用來排序受控制序列的排序委派。您會用它來比較兩個索引鍵。

範例

```

// cliext_multiset_key_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

多重集:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_multiset_key_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mymultiset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

多重集:: key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引鍵的同義字。

範例

```

// cliext_multiset_key_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Mymultiset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

多重集:: lower_bound (STL/CLR)

尋找符合指定之索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式 `x` 會判斷受控制序列中具有對索引鍵之對等順序的第一個元素。如果沒有這類元素存在，則會傳回 **多重集:: end (STL/CLR)**，`()` 否則會傳回指定的反覆運算器 `x`。您可以使用它來尋找目前在受控制序列中符合指定索引鍵之專案序列的開頭。

範例

```

// cliext_multiset_lower_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b')));
    return (0);
}

```

```

a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b

```

多重集:: make_value (STL/CLR)

構造值物件。

語法

```
static value_type make_value(key_type key);
```

參數

key

要使用的索引鍵值。

備註

此成員函式 `value_type` 會傳回其索引鍵為 *key* 的物件。您可以使用它來撰寫適合搭配數個其他成員函式使用的物件。

範例

```

// cliext_multiset_make_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(Mymultiset::make_value(L'a'));
    c1.insert(Mymultiset::make_value(L'b'));
    c1.insert(Mymultiset::make_value(L'c'));

    // display contents " a b c"
    for each (Mymultiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

多重集::多重集(STL/CLR)

建構容器物件。

語法

```

multiset();
explicit multiset(key_compare^ pred);
multiset(multiset<Key>% right);
multiset(multiset<Key>^ right);
template<typename InIter>
multiset(multiset(InIter first, InIter last));
template<typename InIter>
multiset(InIter first, InIter last,
         key_compare^ pred);
multiset(System::Collections::Generic::IEnumerable<GValue>^ right);
multiset(System::Collections::Generic::IEnumerable<GValue>^ right,
         key_compare^ pred);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

pred

受控制序列的順序述詞。

再

要插入的物件或範圍。

備註

此構造函式：

```
multiset();
```

使用預設排序述詞，初始化沒有任何專案的受控制序列 `key_compare()`。您可以使用它來指定空的初始受控制序

列，並使用預設的順序述詞。

此構造函式：

```
explicit multiset(key_compare^ pred);
```

使用順序述詞 *pred*，初始化不含任何專案的受控制序列。您可以使用它來指定空的初始受控制序列，並指定順序述詞。

此構造函式：

```
multiset(multiset<Key>% right);
```

使用順序 [,) 初始化受控制的序列 *right.begin()* *right.end()*，並搭配預設的排序述詞。您可以使用它來指定初始受控制序列，這是由多重集物件許可權所控制之序列的複本，並具有預設排序述詞。

此構造函式：

```
multiset(multiset<Key>^ right);
```

使用順序 [,) 初始化受控制的序列 *right->begin()* *right->end()*，並搭配預設的排序述詞。您可以使用它來指定初始受控制序列，這是由多重集物件許可權所控制之序列的複本，並具有預設排序述詞。

此構造函式：

```
template<typename InIter> multiset(InIter first, InIter last);
```

使用順序 [,) 初始化受控制的序列 *first* *last*，並搭配預設的排序述詞。您可以使用它，讓受控制的序列成為另一個序列的複本，並使用預設的順序述詞。

此構造函式：

```
template<typename InIter> multiset(InIter first, InIter last, key_compare^ pred);
```

使用順序述詞 *pred*，初始化具有序列 [,) 的受控制序列 *first* *last*。*pred* 您可以使用它，讓受控制的序列成為另一個序列的複本，並指定順序述詞。

此構造函式：

```
multiset(System::Collections::Generic::IEnumerable<Key>^ right);
```

使用預設排序述詞，以枚舉器右邊指定的順序，初始化受控制的序列。您可以使用它，讓受控制的序列成為枚舉器所描述之另一個序列的複本，並使用預設的排序述詞。

此構造函式：

```
multiset(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用排序述詞 *pred*，以列舉值右邊所指定的順序，初始化受控制的序列。您可以使用它，透過指定的排序述詞，讓受控制的序列成為枚舉器所描述之另一個序列的複本。

範例

```
// cliext_multiset_construct.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
// construct an empty container
    Mymultiset c1;
    System::Console::WriteLine("size() = {0}", c1.size());
```

```

c1.insert(L'a');
c1.insert(L'b');
c1.insert(L'c');
for each (wchar_t elem in c1)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule
Mymultiset c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range
Mymultiset c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Mymultiset c4(c1.begin(), c1.end(),
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration
Mymultiset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Mymultiset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct from a generic container
Mymultiset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Mymultiset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
c b a
a b c
c b a
a b c
c b a
c b a
a b c
```

多重集:: operator = (STL/CLR)

取代受控制的序列。

語法

```
multiset<Key>% operator=(multiset<Key>% right);
```

參數

再

要複製的容器。

備註

成員運算子會將許可權複製到物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為右邊的受控制序列複本。

範例

```
// cliext_multiset_operator_as.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Mymultiset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2 = c1;
    // display contents " a b c"
    for each (Mymultiset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

多重集:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

此成員函式會傳回反向反覆運算器，指定受控制序列的最後一個元素，或只在空白序列開頭以外的專案。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 Iterator，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_multiset_rbegin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Mymultiset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
```

多重集:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

此類型描述專案的參考。

範例

```
// cliext_multiset_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Mymultiset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        { // get a reference to an element
        Mymultiset::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

多重集::rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

此成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 `Iterator` 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_multiset_rend.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Mymultiset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("!-- --rend() = {0}", *--rit);
    System::Console::WriteLine("!--rend() = {0}", *++rit);
    return (0);
}

```

```

a b c
!-- --rend() = b
!--rend() = a

```

多重集:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_multiset_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Mymultiset::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

多重集:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[多重集:: empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_multiset_size.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

多重集:: size_type (STL/CLR)

兩個元素之間帶正負號距離的類型。

語法

```
typedef int size_type;
```

備註

此類型描述非負的元素計數。

範例

```
// cliext_multiset_size_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mymultiset::size_type diff = 0;
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

多重集:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(multiset<Key>% right);
```

參數

再

要交換內容的容器。

備註

成員函式會在和 *right* 之間交換受控制的序列 `this`。*right* 它會以常數時間執行，而且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_multiset_swap.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Mymultiset c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

多重集:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

此成員函式會傳回陣列，其中包含受控制的序列。您可以用它來取得陣列表單中受控制序列的複本。

範例

```

// cliext_multiset_to_array.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

多重集:: upper_bound (STL/CLR)

尋找符合指定之索引鍵的結束範圍。

語法

```

iterator upper_bound(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式 `x` 會判斷受控制序列中，具有對索引鍵之對等順序的最後一個元素。如果沒有這類元素存在，或如果 `x` 是受控制序列中的最後一個專案，則會傳回[多重集:: END \(STL/CLR\)](#)，`()` 否則會傳回反覆運算器，指定超出的第一個元素 `x`。您可以使用它來找出目前在受控制序列中符合指定索引鍵之專案序列的結尾。

範例

```

// cliext_multiset_upper_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

多重集:: value_comp (STL/CLR)

複製兩個元素值的順序委派。

語法

```
value_compare^ value_comp();
```

備註

此成員函式會傳回用來排序受控制序列的排序委派。您可以使用它來比較兩個元素的值。

範例

```

// cliext_multiset_value_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

多重集:: value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```

// cliext_multiset_value_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    Mymultiset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

多重集:: value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_multiset_value_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Mymultiset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Mymultiset::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

operator != (多重集) (STL/CLR)

清單不等於比較。

語法

```
template<typename Key>
bool operator!=(multiset<Key>% left,
                 multiset<Key>% right);
```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

運算子函式會傳回 `!(left == right)`。您可以使用它來測試當兩個多重集是以元素進行比較時，是否要將 *left* 與 *right* 排序。

範例

```
// cliext_multiset_operator_ne.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

運算子 < (多重集) (STL/CLR)

清單小於比較。

語法

```
template<typename Key>
bool operator<(multiset<Key>% left,
                 multiset<Key>% right);
```

參數

左面

要比較的左容器。

再

要比較的右容器。

備註

如果的最低位置也為 true, 則運算子函數 `i` 會傳回 true `!(right[i] < left[i])` `left[i] < right[i]`。否則, 它會傳回 `left->size() < right->size()` 您用它來測試當兩個多重 *right* 集是以元素進行比較時, 左側是否已排序。

範例

```
// cliext_multiset_operator_lt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (多重集) (STL/CLR)

列出小於或等於比較。

語法

```
template<typename Key>
bool operator<=(multiset<Key>% left,
    multiset<Key>% right);
```

參數

左面

要比較的左容器。

再

要比較的右容器。

備註

運算子函式會傳回 `!(right < left)`。您可以使用它來測試當兩個多重集是以元素進行比較時，左側是否不是靠右排序。

範例

```
// cliext_multiset_operator_le.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (多重集) (STL/CLR)

列出相等比較。

語法

```
template<typename Key>
bool operator==(multiset<Key>% left,
    multiset<Key>% right);
```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

只有當 *left* 和 *right* 所控制的序列具有相同的長度，且每個位置都有相同的時，運算子函數才會傳回 true *i* *left[i] == right[i]*。您可以使用它來測試當兩個多重集是以元素進行比較時，是否要將 *left* 排序為正確。

範例

```
// cliext_multiset_operator_eq.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
        c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

運算子 > (多重集) (STL/CLR)

清單大於比較。

語法

```
template<typename Key>
bool operator>(multiset<Key>% left,
                  multiset<Key>% right);
```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

運算子函式會傳回 `right < left`。您可以使用它來測試當兩個多重集是依照元素進行比較時，是否要向右排序。

範例

```
// cliext_multiset_operator_gt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                           c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                           c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (多重集) (STL/CLR)

列出大於或等於比較。

語法

```
template<typename Key>
bool operator>=(multiset<Key>% left,
                  multiset<Key>% right);
```

參數

左面

要比較的左容器。

右

要比較的右容器。

備註

運算子函式會傳回 `!(left < right)`。您可以使用它來測試當兩個多重集是以元素進行比較時，*左側*是否未按右排序。

範例

```
// cliext_multiset_operator_ge.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::multiset<wchar_t> Mymultiset;
int main()
{
    Mymultiset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mymultiset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

numeric (STL/CLR)

2020/3/25 • [Edit Online](#)

定義容器範本函式，這些函式會執行為數值處理提供的演算法。

語法

```
#include <cliext/numeric>
```

需求

標頭：`<cliext/numeric>`

命名空間：`cliext`

宣告

<code>accumulate (STL/CLR)</code>	藉由計算連續的部分總和來計算指定範圍內所有元素 (包括某個初始值) 的總和，或是計算連續部分結果 (同樣是使用指定的二進位運算而非加總來計算出) 的結果。
<code>adjacent_difference (STL/CLR)</code>	計算在輸入範圍中每個項目及其前置項之間的後續差異並將結果輸出至目的範圍，或計算一般化程序的結果，其中由另一個指定的二進位運算取代差異作業。
<code>inner_product (STL/CLR)</code>	計算兩個範圍的元素乘積總和並將它加到指定的初始值，或計算一般化程序的結果，其中總和及乘積二進位運算會由其他指定的二進位運算取代。
<code>partial_sum (STL/CLR)</code>	在輸入範圍中，從第一個專案到 <code>i</code> 個專案計算一系列的總和，並將每個這類總和的結果儲存在目的範圍的 <code>i</code> 第個元素中，或計算一般化程式的結果，其中總和運算會由另一個指定的二進位運算取代。

成員

累積 (STL/CLR)

藉由計算連續的部分總和來計算指定範圍內所有元素 (包括某個初始值) 的總和，或是計算連續部分結果 (同樣是使用指定的二進位運算而非加總來計算出) 的結果。

語法

```
template<class _InIt, class _Ty> inline
    _Ty accumulate(_InIt _First, _InIt _Last, _Ty _Val);
template<class _InIt, class _Ty, class _Fn2> inline
    _Ty accumulate(_InIt _First, _InIt _Last, _Ty _Val, _Fn2 _Func);
```

備註

此函式的行為與C++標準程式庫數值函數 `accumulate` 相同。如需詳細資訊，請參閱[累積](#)。

adjacent_difference (STL/CLR)

計算在輸入範圍中每個項目及其前置項之間的後續差異並將結果輸出至目的範圍，或計算一般化程序的結果，其中由另一個指定的二進位運算取代差異作業。

語法

```
template<class _InIt, class _OutIt> inline
    _OutIt adjacent_difference(_InIt _First, _InIt _Last,
        _OutIt _Dest);
template<class _InIt, class _OutIt, class _Fn2> inline
    _OutIt adjacent_difference(_InIt _First, _InIt _Last,
        _OutIt _Dest, _Fn2 _Func);
```

備註

此函式的行為與C++標準程式庫數值函數 `adjacent_difference` 相同。如需詳細資訊，請參閱[adjacent_difference](#)。

inner_product (STL/CLR)

計算兩個範圍的元素乘積總和並將它加到指定的初始值，或計算一般化程序的結果，其中總和及乘積二進位運算會由其他指定的二進位運算取代。

語法

```
template<class _InIt1, class _InIt2, class _Ty> inline
    _Ty inner_product(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _Ty _Val);
template<class _InIt1, class _InIt2, class _Ty, class _Fn21,
    class _Fn22> inline
    _Ty inner_product(_InIt1 _First1, _InIt1 _Last1, _InIt2 _First2,
        _Ty _Val, _Fn21 _Func1, _Fn22 _Func2);
```

備註

此函式的行為與C++標準程式庫數值函數 `inner_product` 相同。如需詳細資訊，請參閱[inner_product](#)。

partial_sum (STL/CLR)

在輸入範圍中，從第一個專案到 `i` 個專案計算一系列的總和，並將每個這類總和的結果儲存在目的範圍的 `i` 第個元素中，或計算一般化程式的結果，其中總和運算會由另一個指定的二進位運算取代。

語法

```
template<class _InIt, class _OutIt> inline
    _OutIt partial_sum(_InIt _First, _InIt _Last, _OutIt _Dest);
template<class _InIt, class _OutIt, class _Fn2> inline
    _OutIt partial_sum(_InIt _First, _InIt _Last,
        _OutIt _Dest, _Fn2 _Func);
```

備註

此函式的行為與C++標準程式庫數值函數 `partial_sum` 相同。如需詳細資訊，請參閱[partial_sum](#)。

priority_queue (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有有限存取權之元素的不同長度排序次序。您可以使用容器介面卡 `priority_queue`，以優先順序佇列的形式管理基礎容器。

在下列描述中，與 `GValue` 值相同，除非後者是 ref 型別，在這種情況下就是 `Value^`。同樣地，與 `GContainer` 容器相同，除非後者是 ref 類型，在此情況下為 `Container^`。

語法

```
template<typename Value,
         typename Container>
ref class priority_queue
    System::ICloneable,
    Microsoft::VisualC::StlClr::IPriorityQueue<GValue, GContainer>
{ ..... };
```

參數

值

受控制序列中項目的類型。

容器

基礎容器的類型。

需求

標頭: <cliext/queue>

命名空間: cliext

宣告

■■■	■■
priority_queue::const_reference (STL/CLR)	項目的常數參考類型。
priority_queue::container_type (STL/CLR)	基礎容器的類型。
priority_queue::difference_type (STL/CLR)	兩個項目之間帶正負號距離的類型。
priority_queue::generic_container (STL/CLR)	容器介面卡的泛型介面型別。
priority_queue::generic_value (STL/CLR)	容器介面卡泛型介面的元素類型。
priority_queue::reference (STL/CLR)	項目的參考類型。
priority_queue::size_type (STL/CLR)	兩個項目之間帶正負號距離的類型。
priority_queue::value_compare (STL/CLR)	兩個元素的順序委派。

priority_queue::value_type (STL/CLR)	項目的類型。
priority_queue::assign (STL/CLR)	取代所有項目。
priority_queue::empty (STL/CLR)	測試項目是否不存在。
priority_queue::get_container (STL/CLR)	存取基礎容器。
priority_queue::pop (STL/CLR)	移除 highest-priority 元素。
priority_queue::priority_queue (STL/CLR)	建構容器物件。
priority_queue::push (STL/CLR)	新增元素。
priority_queue::size (STL/CLR)	計算元素的數目。
priority_queue::top (STL/CLR)	存取最高優先順序的元素。
priority_queue::to_array (STL/CLR)	將受控制序列複製到新的陣列。
priority_queue::value_comp (STL/CLR)	複製兩個項目的排序委派。
priority_queue::top_item (STL/CLR)	存取最高優先順序的元素。
priority_queue::operator= (STL/CLR)	取代受控制的序列。

介面

ICloneable	複製物件。
IPriorityQueue<Value, Container>	維護一般容器介面卡。

備註

物件會根據類型的基礎容器來配置和釋出其所控制之序列的儲存體，以 `Container` 儲存專案 `Value` 並依需求成長。它會將順序保持為堆積，而最高優先順序元素（最上層元素，）可以立即存取和移除。物件會限制存取推送新的元素，並只會快顯最高優先順序的專案，以執行優先順序佇列。

物件會藉由呼叫 `priority_queue::value_compare` 類型的預存委派物件，來排序它所控制的序列 (STL/CLR)。當您建立 `priority_queue` 時，可以指定預存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<(value_type, value_type)`。您可以藉由呼叫成員函式 `priority_queue::value_comp (STL/CLR)` 來存取這

個儲存的物件 `()`。

這類委派物件必須在類型 `priority_queue::value_type` 的值上強制執行嚴格弱式排序, ([STL/CLR](#))。這表示, 針對任何兩個金鑰, `x` 以及 `y` :

`value_comp()(x, y)` 每次呼叫時, 都會傳回相同的布林值結果。

如果 `value_comp()(x, y)` 是 true, 則 `value_comp()(y, x)` 必須為 false。

如果 `value_comp()(x, y)` 是 true, 則 `x` 會被視為之前的排序 `y`。

如果 `!value_comp()(x, y) && !value_comp()(y, x)` 是 true, 則 `x` 和 `y` 也稱為具有對等的排序。

若為 `x` `y` 受控制序列中的任何元素, `key_comp()(y, x)` 則為 false。(預設的委派物件, 索引鍵的值永遠不會減少。)

最高優先順序的元素就是其中一個未在任何其他專案之前排序的元素。

因為基礎容器會將元素保留為堆積:

容器必須支援隨機存取反覆運算器。

具有對等順序的元素, 可能會以不同于推送的順序來進行快顯。(排序不穩定。)

因此, 基礎容器的候選項目包括 [deque \(stl/clr\)](#) 和 [vector \(stl/clr\)](#)。

成員

`priority_queue::assign (STL/CLR)`

取代所有項目。

語法

```
void assign(priority_queue<Value, Container>% right);
```

參數

對

要插入的容器介面卡。

備註

成員函式會指派 `right.get_container()` 給基礎容器。您可以使用它來變更佇列的整個內容。

範例

```
// cliext_priority_queue_assign.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Mypriority_queue c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
c a b
```

priority_queue:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_priority_queue_const_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " c b a"
    for ( ; !c1.empty(); c1.pop())
        { // get a const reference to an element
        Mypriority_queue::const_reference cref = c1.top();
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

c b a

priority_queue:: container_type (STL/CLR)

基礎容器的類型。

語法

```
typedef Container value_type;
```

備註

此類型是樣板參數 `Container` 的同義字。

範例

```

// cliext_priority_queue_container_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mypriority_queue::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

c a b

priority_queue::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```
// cliext_priority_queue_difference_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Mypriority_queue::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}
```

```
c a b
pushing 2 = -2
popping 3 = 3
```

priority_queue::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[priority_queue:: size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試 priority_queue 是否為空白。

範例

```
// cliest_priority_queue_empty.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
c a b
size() = 3
empty() = False
size() = 0
empty() = True
```

priority_queue:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::IPriorityQueue<Value>
    generic_container;
```

備註

此類型描述此範本容器介面卡類別的泛型介面。

範例

```

// cliext_priority_queue_generic_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Mypriority_queue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
c a b
d c b a
e d b a c

```

priority_queue:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。（`GValue` 是，`value_type` 或者 `value_type^` `value_type` 是 ref 類型。）

範例

```

// cliext_priority_queue_generic_value.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mypriority_queue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in priority order using generic_value
    for ( ; !gc1->empty(); gc1->pop())
    {
        Mypriority_queue::generic_value elem = gc1->top();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
c a b
c b a

```

priority_queue:: get_container (STL/CLR)

存取基礎容器。

語法

```

container_type get_container();

```

備註

成員函式會傳回基礎容器。您可以使用它來略過容器包裝函式所加諸的限制。

範例

```
// cliext_priority_queue_get_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
```

priority_queue:: operator = (STL/CLR)

取代受控制的序列。

語法

```
priority_queue <Value, Container>% operator=(priority_queue <Value, Container>% right);
```

參數

對

要複製的容器介面卡。

備註

成員運算子會將 *右* 移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 *right* 中受控制序列的複本。

範例

```
// cliext_priority_queue_operator_as.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mypriority_queue c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
c a b
```

priority_queue::pop (STL/CLR)

移除最高 priority 的元素。

語法

```
void pop();
```

備註

成員函式會移除受控制序列中最高優先順序的元素，此專案必須為非空白。您可以使用它來將佇列縮短一回一個元素。

範例

```

// cliext_priority_queue_pop.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

c a b
b a

```

priority_queue::priority_queue (STL/CLR)

構造容器介面卡物件。

語法

```

priority_queue();
priority_queue(priority_queue<Value, Container> right);
priority_queue(priority_queue<Value, Container> right);
explicit priority_queue(value_compare^ pred);
priority_queue(value_compare^ pred, container_type% cont);
template<typename InIt>
priority_queue(InIt first, InIt last);
template<typename InIt>
priority_queue(InIt first, InIt last,
              value_compare^ pred);
template<typename InIt>
priority_queue(InIt first, InIt last,
              value_compare^ pred, container_type% cont);

```

參數

續

要複製的容器。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

```
priority_queue();
```

使用預設順序述詞，建立空的包裝容器。您可以使用它來指定空的初始受控制序列，以及預設順序述詞。

函數：

```
priority_queue(priority_queue<Value, Container>% right);
```

使用順序述詞建立包裝的容器，該容器為的複本 `right.get_container()` | `right.value_comp()`。您可以使用它來指定初始受控制序列，這是由併列物件 許可權所控制之序列的複本(具有相同的順序述詞)。

函數：

```
priority_queue(priority_queue<Value, Container>^ right);
```

使用順序述詞建立包裝的容器，該容器為的複本 `right->get_container()` | `right->value_comp()`。您可以使用它來指定初始受控制序列，其為併列物件所控制之序列的複本 `*right` (具有相同的順序述詞)。

函數：

```
explicit priority_queue(value_compare^ pred);
```

使用順序述詞 `pred`建立空的包裝容器。您可以使用它來指定空的初始受控制序列，以及指定的順序述詞。

函數：

```
priority_queue(value_compare^ pred, container_type cont);
```

使用指定的順序述詞，建立空的包裝容器(具有順序述詞 `pred`)，然後推播您使用它來指定現有容器的初始受控制序列。

函數：

```
template<typename InIt> priority_queue(InIt first, InIt last);
```

使用預設順序述詞來建立空的包裝容器，然後將序列 [`first` , `last`)推送。您可以使用它，透過指定的順序述詞，從指定的 sequence 指定初始受控制序列。

函數：

```
template<typename InIt> priority_queue(InIt first, InIt last, value_compare^ pred);
```

使用排序述詞 `pred`建立空的已包裝容器，然後將序列 [`first` ,)推送 `last` 。您可以使用它，透過指定的順序述詞，從指定的 sequence 指定初始受控制序列。

函數：

```
template<typename InIt> priority_queue(InIt first, InIt last, value_compare^ pred, container_type% cont);
```

使用排序述詞 `pred`建立空的包裝容器，然後將 [接續] 和 [.)的所有元素推入 `first` | `last` 。您可以使用它，透過指定的順序述詞，從現有的容器和指定的 sequence 指定初始受控制序列。

範例

```
// cliext_priority_queue_construct.cpp
// compile with: /clr
#include <cliext/queue>
```

```

#include <cliext/deque>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
typedef cliext::deque<wchar_t> Mydeque;
int main()
{
// construct an empty container
    Mypriority_queue c1;
    Mypriority_queue::container_type^ wc1 = c1.get_container();
    System::Console::WriteLine("size() = {0}", c1.size());

    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    System::Console::WriteLine("size() = {0}", c2.size());

    for each (wchar_t elem in wc1)
        c2.push(elem);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an ordering rule by copying an underlying container
    Mypriority_queue c2x =
        gcnew Mypriority_queue(cliext::greater<wchar_t>(), *wc1);
    for each (wchar_t elem in c2x.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    Mypriority_queue c3(wc1->begin(), wc1->end());
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range and an ordering rule
    Mypriority_queue c4(wc1->begin(), wc1->end(),
        cliext::greater<wchar_t>());
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range, another container, and an ordering rule
    Mypriority_queue c5(wc1->begin(), wc1->end(),
        cliext::greater<wchar_t>(), *wc1);
    for each (wchar_t elem in c5.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct from a generic container
    Mypriority_queue c6(c3);
    for each (wchar_t elem in c6.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    Mypriority_queue c7(%c3);
    for each (wchar_t elem in c7.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an ordering rule, by copying an underlying container
    Mypriority_queue c8 =

```

```
    gcnew Mypriority_queue(cliext::greater<wchar_t>(), *wc1);
    for each (wchar_t elem in c8.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}
```

```
size() = 0
c a b
size() = 0
a c b
a c b
c a b
a c b
a a b c c b
c a b
c a b
a c b
```

priority_queue::push (STL/CLR)

新增元素。

語法

```
void push(value_type val);
```

備註

成員函式會將具有值的元素插入 `val` 至受控制的序列中，並重新排序受控制的序列以維護堆積專業領域。您可以使用它來將另一個元素新增至佇列。

範例

```
// cliext_priority_queue_push.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
```

priority_queue::reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_priority_queue_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify top of priority_queue and redisplay
    Mypriority_queue::reference ref = c1.top();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
x a b
```

priority_queue:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[priority_queue:: empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_priority_queue_size.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // pop an item and reinspect
    c1.pop();
    System::Console::WriteLine("size() = {0} after popping", c1.size());

    // add two elements and reinspect
    c1.push(L'a');
    c1.push(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

c a b
size() = 3 starting with 3
size() = 2 after popping
size() = 4 after adding 2

```

priority_queue:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```

typedef int size_type;

```

備註

型別描述非負的元素計數。

範例

```
// cliext_priority_queue_size_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mypriority_queue::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}
```

```
c a b
size difference = 2
```

priority_queue:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<Value>^ to_array();
```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```
// cliext_priority_queue_to_array.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
d c b a
c a b
```

priority_queue:: top (STL/CLR)

存取最高優先順序的元素。

語法

```
reference top();
```

備註

成員函式會傳回受控制序列之 top (最高優先順序) 專案的參考，該專案必須為非空白。您可以使用它來存取最高優先順序的元素(當您知道它存在時)。

範例

```
// cliext_priority_queue_top.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top() = {0}", c1.top());

    // alter last item and reinspect
    c1.top() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

priority_queue:: top_item (STL/CLR)

存取最高優先順序的元素。

語法

```
property value_type back_item;
```

備註

屬性會存取受控制序列的 top (最高優先順序) 元素，而該專案必須為非空白。您可以使用它來讀取或寫入最高優先順序的元素(當您知道它存在時)。

範例

```
// cliext_priority_queue_top_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top_item = {0}", c1.top_item);

    // alter last item and reinspect
    c1.top_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
c a b
top_item = c
x a b
```

priority_queue:: value_comp (STL/CLR)

複製兩個項目的排序委派。

語法

```
value_compare^ value_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個值。

範例

```

// cliext_priority_queue_value_comp.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    Mypriority_queue::value_compare^ vcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    vcomp = c2.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

priority_queue:: value_compare (STL/CLR)

兩個值的順序委派。

語法

```
binary_delegate<value_type, value_type, int> value_compare;
```

備註

此類型是委派的同義字，可判斷第一個引數是否會在第二個引數之前排序。

範例

```

// cliext_priority_queue_value_compare.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    Mypriority_queue::value_compare^ vcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Mypriority_queue c2 = cliext::greater<wchar_t>();
    vcomp = c2.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        vcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        vcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        vcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

priority_queue:: value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數 *值* 同義。

範例

```
// cliext_priority_queue_value_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::priority_queue<wchar_t> Mypriority_queue;
int main()
{
    Mypriority_queue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        {   // store element in value_type object
            Mypriority_queue::value_type val = c1.top();

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

queue (STL/CLR)

2020/11/2 • [Edit Online](#)

此範本類別所描述的物件，可控制具有先進先出存取權的不同長度專案序列。您可以使用容器介面卡 `queue`，以併列的形式管理基礎容器。

在下列描述中，與 `GValue` 值相同，除非後者是 ref 型別，在這種情況下就是 `Value^`。同樣地，與 `GContainer` 容器相同，除非後者是 ref 類型，在此情況下為 `Container^`。

語法

```
template<typename Value,
         typename Container>
ref class queue
    : public
        System::ICloneable,
        Microsoft::VisualC::StlClr::IQueue<GValue, GContainer>
{ .... };
```

參數

值

受控制序列中項目的類型。

容器

基礎容器的類型。

需求

標頭: <cliext/queue>

命名空間: cliext

宣告

宣告	說明
<code>queue::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>queue::container_type (STL/CLR)</code>	基礎容器的類型。
<code>queue::difference_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。
<code>queue::generic_container (STL/CLR)</code>	容器介面卡的泛型介面型別。
<code>queue::generic_value (STL/CLR)</code>	容器介面卡泛型介面的元素類型。
<code>queue::reference (STL/CLR)</code>	項目的參考類型。
<code>queue::size_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。

queue::value_type (STL/CLR)	項目的類型。
queue::assign (STL/CLR)	取代所有項目。
queue::back (STL/CLR)	存取最後一個項目。
queue::empty (STL/CLR)	測試項目是否不存在。
queue::front (STL/CLR)	存取第一個項目。
queue::get_container (STL/CLR)	存取基礎容器。
queue::pop (STL/CLR)	移除第一個元素。
queue::push (STL/CLR)	加入新的最後一個元素。
queue::queue (STL/CLR)	建構容器物件。
queue::size (STL/CLR)	計算元素的數目。
queue::to_array (STL/CLR)	將受控制序列複製到新的陣列。
queue::back_item (STL/CLR)	存取最後一個項目。
queue::front_item (STL/CLR)	存取第一個項目。
queue::operator= (STL/CLR)	取代受控制的序列。
operator != (併列) (STL/CLR)	判斷物件是否 <code>queue</code> 不等於另一個 <code>queue</code> 物件。
operator < (併列) (STL/CLR)	判斷 <code>queue</code> 物件是否小於另一個 <code>queue</code> 物件。
operator <= (併列) (STL/CLR)	判斷 <code>queue</code> 物件是否小於或等於另一個 <code>queue</code> 物件。
operator == (併列) (STL/CLR)	判斷 <code>queue</code> 物件是否等於另一個 <code>queue</code> 物件。
operator > (併列) (STL/CLR)	判斷 <code>queue</code> 物件是否大於另一個 <code>queue</code> 物件。
operator >= (queue) (STL/CLR)	判斷 <code>queue</code> 物件是否大於或等於另一個 <code>queue</code> 物件。

介面

ICloneable	複製物件。
IQueue<Value, Container>	維護一般容器介面卡。

備註

物件會根據類型的基礎容器來配置和釋出其所控制之序列的儲存體，以 **Container** 儲存專案 **Value** 並依需求成長。物件會限制只推送第一個專案並取出最後一個專案的存取權，以執行先進先出佇列（也稱為 FIFO 佇列，或只是佇列）。

成員

queue:: assign (STL/CLR)

取代所有項目。

語法

```
void assign(queue<Value, Container>% right);
```

參數

對

要插入的容器介面卡。

備註

成員函式會指派 **right.get_container()** 給基礎容器。您可以使用它來變更佇列的整個內容。

範例

```
// cliext_queue_assign.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Myqueue c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c  
a b c
```

queue:: back (STL/CLR)

存取最後一個項目。

語法

```
reference back();
```

備註

成員函式會傳回受控制序列之最後一個元素的參考，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來存取最後一個專案。

範例

```
// cliext_queue_back.cpp  
// compile with: /clr  
#include <cliext/queue>  
  
typedef cliext::queue<wchar_t> Myqueue;  
int main()  
{  
    Myqueue c1;  
    c1.push(L'a');  
    c1.push(L'b');  
    c1.push(L'c');  
  
    // display initial contents " a b c"  
    for each (wchar_t elem in c1.get_container())  
        System::Console::Write("{0} ", elem);  
    System::Console::WriteLine();  
  
    // inspect last item  
    System::Console::WriteLine("back() = {0}", c1.back());  
  
    // alter last item and reinspect  
    c1.back() = L'x';  
    for each (wchar_t elem in c1.get_container())  
        System::Console::Write("{0} ", elem);  
    System::Console::WriteLine();  
    return (0);  
}
```

```
a b c  
back() = c  
a b x
```

queue:: back_item (STL/CLR)

存取最後一個項目。

語法

```
property value_type back_item;
```

備註

屬性會存取受控制序列的最後一個專案，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來讀取或寫入最後一個專案。

範例

```
// cliext_queue_back_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back_item = c
a b x
```

queue:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_queue_const_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for ( ; !c1.empty(); c1.pop())
        {   // get a const reference to an element
            Myqueue::const_reference cref = c1.front();
            System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

queue:: container_type (STL/CLR)

基礎容器的類型。

語法

```
typedef Container value_type;
```

備註

此類型是樣板參數 `Container` 的同義字。

範例

```

// cliext_queue_container_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Myqueue::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

a b c

queue::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```
// cliext_queue_difference_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Myqueue::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}
```

```
a b c
pushing 2 = -2
popping 3 = 3
```

queue::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於 `queue:: size (STL/CLR) () == 0`。您可以使用它來測試佇列是否為空的。

範例

```
// cliext_queue_empty.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

queue:: front (STL/CLR)

存取第一個項目。

語法

```
reference front();
```

備註

成員函式會傳回受控制序列的第一個元素的參考，而該專案必須為非空白。當您知道第一個元素存在時，您可以使用它來存取第一個專案。

範例

```
// cliext_queue_front.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

queue:: front_item (STL/CLR)

存取第一個項目。

語法

```
property value_type front_item;
```

備註

屬性會存取受控制序列的第一個專案，其必須為非空白。當您知道第一個元素存在時，您可以使用它來讀取或寫入第一個專案。

範例

```

// cliext_queue_front_item.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter last item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

queue:: generic_container (STL/CLR)

容器介面卡的泛型介面型別。

語法

```

typedef Microsoft::VisualC::StlClr::IQueue<Value>
generic_container;

```

備註

此類型描述此範本容器介面卡類別的泛型介面。

範例

```

// cliext_queue_generic_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myqueue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

queue:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。（`GValue` 是 `value_type` 或者 `value_type^` `value_type` 是 ref 類型。）

範例

```

// cliext_queue_generic_value.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Myqueue::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in order using generic_value
    for (; !gc1->empty(); gc1->pop())
    {
        Myqueue::generic_value elem = gc1->front();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c

```

queue:: get_container (STL/CLR)

存取基礎容器。

語法

```

container_type^ get_container();

```

備註

成員函式會傳回基礎容器。您可以使用它來略過容器包裝函式所加諸的限制。

範例

```
// cliext_queue_get_container.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

queue:: operator = (STL/CLR)

取代受控制的序列。

語法

```
queue <Value, Container>% operator=(queue <Value, Container>% right);
```

參數

對

要複製的容器介面卡。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_queue_operator_as.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

佇列 ::pop (STL/CLR)

移除最後一個元素。

語法

```
void pop();
```

備註

成員函式會移除受控制序列的最後一個專案，其必須為非空白。您可以使用它來將佇列縮短一回一個元素。

範例

```

// cliext_queue_pop.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
b c

```

queue::push (STL/CLR)

加入新的最後一個元素。

語法

```

void push(value_type val);

```

備註

成員函式會在佇列結尾新增具有值的元素 `val`。您可以使用它來將元素附加至佇列。

範例

```

// cliext_queue_push.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

queue:: queue (STL/CLR)

構造容器介面卡物件。

語法

```
queue();
queue(queue<Value, Container>% right);
queue(queue<Value, Container>^ right);
explicit queue(container_type% wrapped);
```

參數

對

要複製的物件。

包裹

要使用的包裝容器。

備註

函數：

```
queue();
```

建立空的包裝容器。您可以使用它來指定空的初始受控制序列。

函數：

```
queue(queue<Value, Container>% right);
```

建立包裝的容器，該容器為的複本 `right.get_container()`。您可以使用它來指定初始受控制序列，這是由併列物件 許可權所控制之序列的複本。

函數：

```
queue(queue<Value, Container>^ right);
```

建立包裝的容器，該容器為的複本 `right->get_container()`。您可以使用它來指定初始受控制序列，這是由併列物件所控制之序列的複本 `*right`。

函數：

```
explicit queue(container_type wrapped);
```

使用 包裝 為包裝容器的現有容器。您可以使用它來從現有的容器中建立併列。

範例

```

// cliext_queue_construct.cpp
// compile with: /clr
#include <cliext/queue>
#include <cliext/list>

typedef cliext::queue<wchar_t> Myqueue;
typedef cliext::list<wchar_t> Mylist;
typedef cliext::queue<wchar_t, Mylist> Myqueue_list;
int main()
{
    //
    // construct an empty container
    Myqueue c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    //
    // construct from an underlying container
    Mylist v2(5, L'x');
    Myqueue_list c2(v2);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    //
    // construct by copying another container
    Myqueue_list c3(c2);
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    //
    // construct by copying another container through handle
    Myqueue_list c4(%c2);
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

size() = 0
x x x x x
x x x x x
x x x x x

```

queue:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_queue_reference.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify back of queue and redisplay
    Myqueue::reference ref = c1.back();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b x
```

queue:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[queue:: empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_queue_size.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // pop an item and reinspect
    c1.pop();
    System::Console::WriteLine("size() = {0} after popping", c1.size());

    // add two elements and reinspect
    c1.push(L'a');
    c1.push(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 2 after popping
size() = 4 after adding 2

```

queue:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```

typedef int size_type;

```

備註

型別描述非負的元素計數。

範例

```
// cliext_queue_size_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myqueue::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}
```

```
a b c
size difference = 2
```

queue:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<Value>^ to_array();
```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```
// cliext_queue_to_array.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

queue:: value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數 *值* 同義。

範例

```

// cliext_queue_value_type.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        {   // store element in value_type object
            Myqueue::value_type val = c1.front();

            System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

operator != (併列) (STL/CLR)

併列不等於比較。

語法

```

template<typename Value,
         typename Container>
bool operator!=(queue<Value, Container>% left,
                  queue<Value, Container>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個併列是依元素進行比較時，**左邊**是否未以正確的順序排序。

範例

```

// cliext_queue_operator_ne.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

operator < (queue) (STL/CLR)

併列小於比較。

語法

```

template<typename Value,
         typename Container>
bool operator<(queue<Value, Container>% left,
                 queue<Value, Container>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 i $!(right[i] < left[i])$ 而言也是 true $left[i] < right[i]$ 。否

則，它會傳回 `left->queue::size(STL/CLR) () < right->size()` 您使用它來測試當兩個佇列 `right` 是依元素進行比較時，是否要將 `left` 排序。

範例

```
// cliext_queue_operator_lt.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (queue) (STL/CLR)

佇列小於或等於比較。

語法

```
template<typename Value,
         typename Container>
bool operator<=(queue<Value, Container>% left,
                 queue<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個併列是依元素進行比較時，左邊是否未排序。

範例

```
// cliext_queue_operator_le.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (併列) (STL/CLR)

併列相等比較。

語法

```
template<typename Value,
         typename Container>
bool operator==(queue<Value, Container>% left,
                  queue<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true `i` `left[i] == right[i]`。您可以使用它來測試當兩個併列是依元素進行比較時，左邊是否以相同的順序排序。

範例

```
// cliext_queue_operator_eq.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}", 
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}", 
        c1 == c2);
    return (0);
}
```

```
a b c  
a b d  
[a b c] == [a b c] is True  
[a b c] == [a b d] is False
```

operator > (queue) (STL/CLR)

佢列大於比較。

語法

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個佇列是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_queue_operator_gt.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (queue) (STL/CLR)

佇列大於或等於比較。

語法

```
template<typename Value,
         typename Container>
bool operator>=(queue<Value, Container>% left,
                  queue<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個佇列是依元素進行比較時，左邊是否未排序。

範例

```
// cliext_queue_operator_ge.cpp
// compile with: /clr
#include <cliext/queue>

typedef cliext::queue<wchar_t> Myqueue;
int main()
{
    Myqueue c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myqueue c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

set (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有雙向存取之元素的不同長度序列。您可以使用容器 `set` 來管理一連串的專案，做為 (近) 的節點數目，每個節點都會儲存一個元素。

在下列描述中，與 `GValue` 相同 `GKey`，除非後者是 `ref` 型別，否則它就會與索引 鍵相同，在這種情況下，則為 `Key^`。

語法

```
template<typename Key>
ref class set
: public
System::ICloneable,
System::Collections::IEnumerable,
System::Collections::ICollection,
System::Collections::Generic::IEnumerable<GValue>,
System::Collections::Generic::ICollection<GValue>,
System::Collections::Generic::IList<GValue>,
Microsoft::VisualC::StlClr::ITree<Gkey, GValue>
{ ..... };
```

參數

索引/鍵

受控制序列中項目的主要元件型別。

需求

標頭 : <cliext/set>

命名空間 : cliext

宣告

IIII	II
<code>set::const_iterator (STL/CLR)</code>	用於受控制序列的常數迭代器類型。
<code>set::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>set::const_reverse_iterator (STL/CLR)</code>	用於受控制序列的常數反向迭代器類型。
<code>set::difference_type (STL/CLR)</code>	(的類型可能簽署兩個專案之間的) 距離。
<code>set::generic_container (STL/CLR)</code>	容器的泛型介面型別。
<code>set::generic_iterator (STL/CLR)</code>	容器之泛型介面的反覆運算器類型。
<code>set::generic_reverse_iterator (STL/CLR)</code>	容器的泛型介面之反向反覆運算器的類型。

¶¶¶	¶
<code>set::generic_value (STL/CLR)</code>	容器之泛型介面的元素類型。
<code>set::iterator (STL/CLR)</code>	受控制序列之迭代器的類型。
<code>set::key_compare (STL/CLR)</code>	兩個索引鍵的排序委派。
<code>set::key_type (STL/CLR)</code>	排序索引鍵的類型。
<code>set::reference (STL/CLR)</code>	項目的參考類型。
<code>set::reverse_iterator (STL/CLR)</code>	受控制序列的反向迭代器類型。
<code>set::size_type (STL/CLR)</code>	兩個元素之間 (非負) 距離的型別。
<code>set::value_compare (STL/CLR)</code>	兩個元素值的排序委派。
<code>set::value_type (STL/CLR)</code>	項目的類型。

¶¶¶	¶
<code>set::begin (STL/CLR)</code>	指定受控制序列的開頭。
<code>set::clear (STL/CLR)</code>	移除所有項目。
<code>set::count (STL/CLR)</code>	計算符合指定索引鍵的元素。
<code>set::empty (STL/CLR)</code>	測試項目是否不存在。
<code>set::end (STL/CLR)</code>	指定受控制序列的結尾。
<code>set::equal_range (STL/CLR)</code>	尋找符合指定之索引鍵的範圍。
<code>set::erase (STL/CLR)</code>	移除位於指定位置的項目。
<code>set::find (STL/CLR)</code>	尋找符合指定之索引鍵的元素。
<code>set::insert (STL/CLR)</code>	加入項目。
<code>set::key_comp (STL/CLR)</code>	複製兩個索引鍵的排序委派。
<code>set::lower_bound (STL/CLR)</code>	尋找符合指定索引鍵的範圍開頭。
<code>set::make_value (STL/CLR)</code>	結構值物件。
<code>set::rbegin (STL/CLR)</code>	指定反向受控制序列的開頭。
<code>set::rend (STL/CLR)</code>	指定反向受控制序列的結尾。
<code>set::set (STL/CLR)</code>	建構容器物件。

set	set
<code>set::size (STL/CLR)</code>	計算元素的數目。
<code>set::swap (STL/CLR)</code>	交換兩個容器的內容。
<code>set::to_array (STL/CLR)</code>	將受控制序列複製到新的陣列。
<code>set::upper_bound (STL/CLR)</code>	尋找符合指定索引鍵的範圍結尾。
<code>set::value_comp (STL/CLR)</code>	針對兩個元素值複製順序委派。
set	set
<code>set::operator= (STL/CLR)</code>	取代受控制的序列。
<code>operator != (設定) (STL/CLR)</code>	判斷物件是否 <code>set</code> 不等於另一個 <code>set</code> 物件。
<code>operator< (set) (STL/CLR)</code>	判斷 <code>set</code> 物件是否小於另一個 <code>set</code> 物件。
<code>operator<= (set) (STL/CLR)</code>	判斷 <code>set</code> 物件是否小於或等於另一個 <code>set</code> 物件。
<code>operator== (set) (STL/CLR)</code>	判斷 <code>set</code> 物件是否等於另一個 <code>set</code> 物件。
<code>operator> (set) (STL/CLR)</code>	判斷 <code>set</code> 物件是否大於另一個 <code>set</code> 物件。
<code>operator>= (set) (STL/CLR)</code>	判斷 <code>set</code> 物件是否大於或等於另一個 <code>set</code> 物件。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的元素排序。
ICollection<T>	維護具類型的元素群組。
ITree<Key, Value>	維護泛型容器。

備註

物件會針對其控制為個別節點的序列，配置和釋出儲存體。它會將元素插入（近）平衡的樹狀結構中，藉由變更節點之間的連結，而不是藉由將節點的內容複寫到另一個節點的方式來保持排序。這表示您可以自由插入和移除專案，而不會干擾其餘的元素。

物件會藉由呼叫 `set::key_compare (STL/CLR)` 類型的預存委派物件，排序它所控制的序列。您可以在建立集合

時指定儲存的委派物件。如果您未指定委派物件，預設值就是比較 `operator<(key_type, key_type)`。您可以藉由呼叫成員函式 `set:: key_comp (STL/CLR)` 來存取這個儲存的物件 `()`。

這類委派物件必須對 `set:: key_type (STL/CLR)` 類型的索引鍵強制執行嚴格的弱式排序。這表示，針對任何兩個金鑰，`x` 以及 `y`：

`key_comp()(x, y)` 每次呼叫時，都會傳回相同的布林值結果。

如果 `key_comp()(x, y)` 是 true，則 `key_comp()(y, x)` 必須為 false。

如果 `key_comp()(x, y)` 是 true，則 `x` 會被視為之前的排序 `y`。

如果 `!key_comp()(x, y) && !key_comp()(y, x)` 是 true，則 `x` 和 `y` 也稱為具有對等的排序。

若為 `x` `y` 受控制序列中的任何元素，`key_comp()(y, x)` 則為 false。(預設的委派物件，索引鍵的值永遠不會減少。)不同于樣板類別 `集`，樣板類別的物件不 `set` 需要所有元素的索引鍵都是唯一的。(兩個以上的索引鍵可以有對等的順序。)

每個元素都可作為 ey 和值。序列的表示方式，可讓您查閱、插入和移除具有許多作業的任意元素，並以序列中專案數目的對數為比例，(對數時間)。此外，插入項目不會使任何迭代器無效，移除項目則僅會使指向被移除項目的迭代器無效。

集合支援雙向反覆運算器，這表示您可以逐步執行指定受控制序列中元素的反覆運算器，以逐步執行連續的元素。特殊的前端節點對應至 `set:: end (STL/CLR)` 所傳回的反覆運算器 `()`。您可以遞減此反覆運算器，以到達受控制序列中的最後一個元素(如果有的話)。您可以將集合反覆運算器遞增以到達前端節點，然後再比較是否等於 `end()`。但是，您無法取值傳回的反覆運算器 `end()`。

請注意，您不能直接參考指定其數位位置的 `set` 元素，這需要隨機存取反覆運算器。

集合反覆運算器會將控制碼儲存至其相關聯的集合節點，然後再將控制碼儲存至其相關聯的容器。您只能使用反覆運算器與其相關聯的容器物件。Set iterator 會維持有效，只要其相關聯的集合節點與某個集合相關聯。此外，有效的 iterator 是 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要它不等於就可以了 `end()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

`set:: begin (STL/CLR)`

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回雙向反覆運算器，其指定受控制序列的第一個專案，或空白序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_set_begin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myset::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);
    return (0);
}
```

```
a b c
*begin() = a
*++begin() = b
```

set:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫 `set:: erase (stl/clr) ()`、`set:: begin (stl/clr) ()`、`set:: end (STL/clr) ()`。您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_set_clear.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

set:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```

typedef T2 const_iterator;

```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數雙向反覆運算器。

範例

```

// cliext_set_const_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}

```

a b c

set:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```

// cliext_set_const_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myset::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        Myset::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}

```

a b c

set:: const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_set_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myset::const_reverse_iterator crit = c1.rbegin();
    for ( ; crit != c1.rend(); ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

c b a

set:: count (STL/CLR)

尋找符合指定索引鍵的項目數目。

語法

```
size_type count(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回受控制序列中的專案數目，其具有與索引鍵相等的排序。您會用它來判斷目前在受控制序列中，符合指定之索引鍵的項目數目。

範例

```
// cliext_set_count.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("count(L'A') = {0}", c1.count(L'A'));
    System::Console::WriteLine("count(L'b') = {0}", c1.count(L'b'));
    System::Console::WriteLine("count(L'C') = {0}", c1.count(L'C'));
    return (0);
}
```

```
a b c
count(L'A') = 0
count(L'b') = 1
count(L'C') = 0
```

set::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_set_difference_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myset::difference_type diff = 0;
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (Myset::iterator it = c1.end(); it != c1.begin(); --it)
        --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

set::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[set::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試集合是否為空白。

範例

```
// cliext_set_empty.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}
```

```
a b c
size() = 3
empty() = False
size() = 0
empty() = True
```

set:: end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回雙向反覆運算器，指向受控制序列的結尾以外的位置。您可以使用它來取得反覆運算器，以指定受控制序列的結尾。如果受控制序列的長度變更，其狀態不會變更。

範例

```

// cliext_set_end.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    Myset::iterator it = c1.end();
    --it;
    System::Console::WriteLine("*-- --end() = {0}", *--it);
    System::Console::WriteLine("*--end() = {0}", *++it);
    return (0);
}

```

```

a b c
*-- --end() = b
*--end() = c

```

set::equal_range (STL/CLR)

尋找符合指定之索引鍵的範圍。

語法

```

cliext::pair<iterator, iterator> equal_range(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會傳回一組反覆運算器 `cliext::pair<iterator, iterator>(set::lower_bound (stl/clr) (key), set::upper_bound (stl/clr) (key))`。您可以使用它來判斷目前在受控制序列中，符合指定索引鍵的元素範圍。

範例

```

// cliext_set_equal_range.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
typedef Myset::pair_iter_iter Pairii;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display results of failed search
    Pairii pair1 = c1.equal_range(L'x');
    System::Console::WriteLine("equal_range(L'x') empty = {0}",
        pair1.first == pair1.second);

    // display results of successful search
    pair1 = c1.equal_range(L'b');
    for (; pair1.first != pair1.second; ++pair1.first)
        System::Console::Write("{0} ", *pair1.first);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
equal_range(L'x') empty = True
b

```

set:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);
size_type erase(key_type key)

```

參數

first

要清除的範圍開頭。

key

要清除的索引鍵值。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除所指向之受控制序列的專案，並傳回反覆運算器，指定移除專案之後的第一個元素。

或`set::end(STL/CLR)`(`()`如果沒有這樣的元素的話)。您可以使用它來移除單一專案。

第二個成員函式會移除範圍`[,)`中受控制序列的元素，`first` `last`並傳回反覆運算器，此反覆運算器會指定移除任何專案之後剩餘的第一個元素，或`end()`如果沒有這類專案存在，則為。您可以使用它來移除零個或多個連續元素。

第三個成員函式會移除其索引鍵對索引鍵具有對等排序之受控制序列的任何元素，並傳回已移除的元素數目計數。您可以使用它來移除和計算所有符合指定索引鍵的元素。

每個專案清除都會花費時間與受控制序列中專案數目的對數成正比。

範例

```
// cliext_set_erase.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.insert(L'd');
    c1.insert(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    Myset::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

set::find (STL/CLR)

尋找符合指定之索引鍵的元素。

語法

```
iterator find(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

如果受控制序列中至少有一個專案具有與索引鍵相等的排序，則成員函式會傳回反覆運算器，指定其中一個元素；否則，它會傳回`set::end (STL/CLR)`。您可以使用它來找出目前在受控制序列中且符合指定索引鍵的元素。

範例

```
// cliext_set_find.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("find {0} = {1}",
        L'A', c1.find(L'A') != c1.end());
    System::Console::WriteLine("find {0} = {1}",
        L'b', *c1.find(L'b'));
    System::Console::WriteLine("find {0} = {1}",
        L'C', c1.find(L'C') != c1.end());
    return (0);
}
```

```
a b c
find A = False
find b = b
find C = False
```

set::generic_container (STL/CLR)

容器的泛型介面型別。

語法

```
typedef Microsoft::VisualC::StlClr::
    ITree<GKey, GValue>
generic_container;
```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_set_generic_container.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.insert(L'e');
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

set:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerBidirectionalIterator<generic_value>
generic_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```

// cliext_set_generic_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_iterator gcit = gc1->begin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

set::generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value>
generic_reverse_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```

// cliext_set_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_reverse_iterator gcit = gc1->rbegin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
c

```

set:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_set_generic_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    Myset::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get an element and display it
    Myset::generic_iterator gcit = gc1->begin();
    Myset::generic_value gcval = *gcit;
    System::Console::WriteLine("{0} ", gcval);
    return (0);
}

```

```

a b c
a b c
a

```

set:: insert (STL/CLR)

加入項目。

語法

```

cliext::pair<iterator, bool> insert(value_type val);
iterator insert(iterator where, value_type val);
template<typename InIter>
void insert(InIter first, InIter last);
void insert(System::Collections::Generic::IEnumerable<value_type>^ right);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的索引鍵值。

where

在容器中插入 (提示僅)。

備註

每個成員函式都會插入其餘運算元所指定的序列。

第一個成員函式會致力於插入具有值 *va* 的元素，並傳回一對值 *x*。如果 *x.second* 為 true，會 *x.first* 指定新插入的專案，否則會 *x.first* 指定具有對等順序的專案，而該專案已存在，且不會插入新的元素。您可以使用它來插入單一元素。

第二個成員函式會插入具有值 *va* 的元素，並使用 *where* 作為提示 (來改善效能)，並傳回反覆運算器，以指定新插入的元素。您可以使用它來插入單一元素，這可能與您知道的元素相鄰。

第三個成員函式會將序列 [*first* , *last*) 插入。您可以使用它來插入從另一個序列複製的零或多個元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

每個插入的專案都需要時間與受控制序列中專案數目的對數成正比。但是，如果指定的提示指定插入點連續的元素，則可能會在分攤的常數時間內進行插入。

範例

```

// cliext_set_insert.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
typedef Myset::pair_iter_bool Pairib;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value, unique and duplicate
    Pairib pair1 = c1.insert(L'x');
    System::Console::WriteLine("insert(L'x') = [{0} {1}]",
        *pair1.first, pair1.second);

    pair1 = c1.insert(L'b');
    System::Console::WriteLine("insert(L'b') = [{0} {1}]",
        *pair1.first, pair1.second);

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value with hint
    System::Console::WriteLine("insert(begin(), L'y') = {0}",
        *c1.insert(c1.begin(), L'y'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    Myset c2;
    Myset::iterator it = c1.end();
    c2.insert(c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    Myset c3;
    c3.insert( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(L'x') = [x True]
insert(L'b') = [b False]
a b c x
insert(begin(), L'y') = y
a b c x y
a b c x
a b c x y

```

set:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的雙向反覆運算器。

範例

```
// cliext_set_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    Myset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

set:: key_comp (STL/CLR)

複製兩個索引鍵的排序委派。

語法

```
key_compare^key_comp();
```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您會用它來比較兩個索引鍵。

範例

```

// cliext_set_key_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

set:: key_compare (STL/CLR)

兩個索引鍵的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<GKey, GKey, bool>
key_compare;

```

備註

此類型是委派的同義字，可決定其索引鍵引數的順序。

範例

```

// cliext_set_key_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::key_compare^ kcomp = c1.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    System::Console::WriteLine();

    // test a different ordering rule
    Myset c2 = cliext::greater<wchar_t>();
    kcomp = c2.key_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a'));
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

compare(L'a', L'a') = False
compare(L'a', L'b') = False
compare(L'b', L'a') = True

```

set::key_type (STL/CLR)

排序索引鍵的類型。

語法

```
typedef Key key_type;
```

備註

此類型是樣板參數索引 鍵的同義字。

範例

```
// cliext_set_key_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using key_type
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in key_type object
        Myset::key_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

set::lower_bound (STL/CLR)

尋找符合指定索引鍵的範圍開頭。

語法

```
iterator lower_bound(key_type key);
```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的第一個專案，其對索引 *x* 鍵具有對等的排序。如果沒有這類元素，則會傳回 `set::end (STL/CLR) ()`；否則會傳回指定的 iterator *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的一連串元素。

範例

```

// cliext_set_lower_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("lower_bound(L'x') == end() = {0}",
        c1.lower_bound(L'x') == c1.end());

    System::Console::WriteLine("*lower_bound(L'a') = {0}",
        *c1.lower_bound(L'a'));
    System::Console::WriteLine("*lower_bound(L'b') = {0}",
        *c1.lower_bound(L'b'));
    return (0);
}

```

```

a b c
lower_bound(L'x') == end() = True
*lower_bound(L'a') = a
*lower_bound(L'b') = b

```

set:: make_value (STL/CLR)

結構值物件。

語法

```
static value_type make_value(key_type key);
```

參數

key

要使用的索引鍵值。

備註

成員函式會傳回 `value_type` 其索引鍵為 索引鍵的物件。您可以使用它來撰寫一個適合與其他數個成員函式搭配使用的物件。

範例

```
// cliext_set_make_value.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(Myset::make_value(L'a'));
    c1.insert(Myset::make_value(L'b'));
    c1.insert(Myset::make_value(L'c'));

    // display contents " a b c"
    for each (Myset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

set:: operator = (STL/CLR)

取代受控制的序列。

語法

```
set<Key>% operator=(set<Key>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_set_operator_as.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (Myset::value_type elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2 = c1;

    // display contents " a b c"
    for each (Myset::value_type elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

set:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 `Iterator`，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_set_rbegin.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    Myset::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
```

set:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_set_reference.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    Myset::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        {   // get a reference to an element
        Myset::reference ref = *it;
        System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

set::rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 `Iterator` 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```
// cliext_set_rend.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    Myset::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("--- --rend() = {0}", *--rit);
    System::Console::WriteLine("---rend() = {0}", *++rit);
    return (0);
}
```

```
a b c
--- --rend() = b
---rend() = a
```

set:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```

// cliext_set_reverse_iterator.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" reversed
    Myset::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}

```

c b a

set::set (STL/CLR)

建構容器物件。

語法

```

set();
explicit set(key_compare^ pred);
set(set<Key>% right);
set(set<Key>^ right);
template<typename InIter>
    setset(InIter first, InIter last);
template<typename InIter>
    set(InIter first, InIter last,
        key_compare^ pred);
set(System::Collections::Generic::IEnumerable<GValue>^ right);
set(System::Collections::Generic::IEnumerable<GValue>^ right,
    key_compare^ pred);

```

參數

first

要插入的範圍開頭。

last

要插入的範圍結尾。

Pred

受控制序列的順序述詞。

對

要插入的物件或範圍。

備註

函數：

`set();`

使用預設順序述詞，初始化沒有元素的受控制序列 `key_compare()`。您可以使用它來指定空的初始受控制序列，

以及預設順序述詞。

函數：

```
explicit set(key_compare^ pred);
```

使用順序述詞 *pred*, 初始化沒有元素的受控制序列。您可以使用它來指定空的初始受控制序列, 以及指定的順序述詞。

函數：

```
set(set<Key>% right);
```

使用順序 [*right.begin()* , *right.end()*], 以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列, 這是由 *set* 物件 許可權所控制之序列的複本與預設順序述詞。

函數：

```
set(set<Key>^ right);
```

使用順序 [*right->begin()* , *right->end()*], 以預設順序述詞初始化受控制的序列。您可以使用它來指定初始受控制序列, 這是由 *set* 物件 許可權所控制之序列的複本與預設順序述詞。

函數：

```
template<typename InIter> set(InIter first, InIter last);
```

使用順序 [*first* , *last*], 以預設順序述詞初始化受控制的序列。您可以使用它, 以預設順序述詞, 讓受控制的序列成為另一個序列的複本。

函數：

```
template<typename InIter> set(InIter first, InIter last, key_compare^ pred);
```

使用序列 [*first* , *last*], 以順序述詞 *pred* 初始化受控制的序列。您可以使用它, 利用指定的順序述詞, 讓受控制的序列成為另一個序列的複本。

函數：

```
set(System::Collections::Generic::IEnumerable<Key>^ right);
```

以列舉值 右邊指定的順序, 使用預設順序述詞, 初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個順序的複本, 以及預設順序述詞。

函數：

```
set(System::Collections::Generic::IEnumerable<Key>^ right, key_compare^ pred);
```

使用由列舉值 右邊指定的序列, 並搭配順序述詞 *pred*, 初始化受控制的序列。您可以使用它, 透過指定的順序述詞, 讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```
// cliext_set_construct.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
// construct an empty container
    Myset c1;
    System::Console::WriteLine("size() = {0}", c1.size());
```

```

c1.insert(L'a');
c1.insert(L'b');
c1.insert(L'c');
for each (wchar_t elem in c1)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an ordering rule
Myset c2 = cliext::greater_equal<wchar_t>();
System::Console::WriteLine("size() = {0}", c2.size());

c2.insert(c1.begin(), c1.end());
for each (wchar_t elem in c2)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range
Myset c3(c1.begin(), c1.end());
for each (wchar_t elem in c3)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an iterator range and an ordering rule
Myset c4(c1.begin(), c1.end(),
         cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c4)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration
Myset c5( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
for each (wchar_t elem in c5)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct with an enumeration and an ordering rule
Myset c6( // NOTE: cast is not needed
    (System::Collections::Generic::IEnumerable<wchar_t>^)%c3,
    cliext::greater_equal<wchar_t>());
for each (wchar_t elem in c6)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct from a generic container
Myset c7(c4);
for each (wchar_t elem in c7)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();

// construct by copying another container
Myset c8(%c3);
for each (wchar_t elem in c8)
    System::Console::Write("{0} ", elem);
System::Console::WriteLine();
return (0);
}

```

```
size() = 0
a b c
size() = 0
c b a
a b c
c b a
a b c
c b a
c b a
a b c
```

set:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[set:: empty \(STL/CLR\) \(\)](#)。

範例

```
// cliext_set_size.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.insert(L'a');
    c1.insert(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}
```

```
a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2
```

set:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```
// cliext_set_size_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Myset::size_type diff = 0;
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}
```

```
a b c
end()-begin() = 3
```

set:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(set<Key>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_set_swap.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    Myset c2;
    c2.insert(L'd');
    c2.insert(L'e');
    c2.insert(L'f');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
d e f
d e f
a b c

```

set:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<value_type>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```

// cliext_set_to_array.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.insert(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c d
a b c

```

set::upper_bound (STL/CLR)

尋找符合指定索引鍵的範圍結尾。

語法

```

iterator upper_bound(key_type key);

```

參數

key

要搜尋的索引鍵值。

備註

成員函式會決定受控制序列中的最後一個專案，其對索引 *x* 鍵具有對等的順序。如果不存在這類專案，或 *x* 為受控制序列中的最後一個專案，則會傳回 `set::END (STL/CLR) ()`；否則會傳回指定第一個元素的反覆運算器 *x*。您可以使用它來找出目前在受控制序列中，符合指定索引鍵的專案序列結尾。

範例

```

// cliext_set_upper_bound.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("upper_bound(L'x') == end() = {0}",
        c1.upper_bound(L'x') == c1.end());

    System::Console::WriteLine("*upper_bound(L'a') = {0}",
        *c1.upper_bound(L'a'));
    System::Console::WriteLine("*upper_bound(L'b') = {0}",
        *c1.upper_bound(L'b'));
    return (0);
}

```

```

a b c
upper_bound(L'x') == end() = True
*upper_bound(L'a') = b
*upper_bound(L'b') = c

```

set:: value_comp (STL/CLR)

針對兩個元素值複製順序委派。

語法

```

value_compare^ value_comp();

```

備註

成員函式會傳回排序委派，用來排序受控制的序列。您可以使用它來比較兩個元素值。

範例

```

// cliext_set_value_comp.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a')));
    System::Console::WriteLine();
    return (0);
}

```

```

compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False

```

set:: value_compare (STL/CLR)

兩個元素值的排序委派。

語法

```

Microsoft::VisualC::StlClr::BinaryDelegate<generic_value, generic_value, bool>
    value_compare;

```

備註

此類型是委派的同義字，可決定其值引數的順序。

範例

```

// cliext_set_value_compare.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    Myset::value_compare^ kcomp = c1.value_comp();

    System::Console::WriteLine("compare(L'a', L'a') = {0}",
        kcomp(L'a', L'a'));
    System::Console::WriteLine("compare(L'a', L'b') = {0}",
        kcomp(L'a', L'b'));
    System::Console::WriteLine("compare(L'b', L'a') = {0}",
        kcomp(L'b', L'a')));
    System::Console::WriteLine();
    return (0);
}

```

```
compare(L'a', L'a') = False
compare(L'a', L'b') = True
compare(L'b', L'a') = False
```

set:: value_type (STL/CLR)

項目的類型。

語法

```
typedef generic_value value_type;
```

備註

這個類型與 `generic_value` 同義。

範例

```
// cliext_set_value_type.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c" using value_type
    for (Myset::iterator it = c1.begin(); it != c1.end(); ++it)
        { // store element in value_type object
        Myset::value_type val = *it;

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

operator != (設定) (STL/CLR)

清單不等於比較。

語法

```
template<typename Key>
bool operator!=(set<Key>% left,
                set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個集合是依元素進行比較時，左邊是否未以正確的順序排序。

範例

```
// cliext_set_operator_ne.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

運算子 < (設定) (STL/CLR)

清單小於比較。

語法

```
template<typename Key>
bool operator<(set<Key>% left,
                 set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 `i !(right[i] < left[i])` 而言也是 true `left[i] < right[i]`。否則，它會傳回 `left->size() < right->size()` 您使用它來測試當兩個集合 `right` 是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_set_operator_lt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (設定) (STL/CLR)

清單小於或等於比較。

語法

```
template<typename Key>
bool operator<=(set<Key>% left,
    set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個集合是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_set_operator_le.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (設定) (STL/CLR)

列出相等的比較。

語法

```
template<typename Key>
bool operator==(set<Key>% left,
                  set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true `i` `left[i] == right[i]`。您可以使用它來測試當兩個集合是依元素進行比較時，左邊是否以相同的順序排序。

範例

```
// cliext_set_operator_eq.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
                           c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
                           c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

運算子 > (設定) (STL/CLR)

清單大於比較。

語法

```
template<typename Key>
bool operator>(set<Key>% left,
                  set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個集合是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_set_operator_gt.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                           c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                           c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (設定) (STL/CLR)

列出大於或等於比較。

語法

```
template<typename Key>
bool operator>=(set<Key>% left,
                 set<Key>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個集合是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_set_operator_ge.cpp
// compile with: /clr
#include <cliext/set>

typedef cliext::set<wchar_t> Myset;
int main()
{
    Myset c1;
    c1.insert(L'a');
    c1.insert(L'b');
    c1.insert(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Myset c2;
    c2.insert(L'a');
    c2.insert(L'b');
    c2.insert(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

stack (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有後進先出存取權之不同長度的元素序列。您可以使用容器介面卡 `stack`，以向下推堆疊的方式管理基礎容器。

在下列描述中，與 `GValue` 值相同，除非後者是 ref 型別，在這種情況下就是 `Value^`。同樣地，與 `GContainer` 容器相同，除非後者是 ref 類型，在此情況下為 `Container^`。

語法

```
template<typename Value,
         typename Container>
ref class stack
    : public
        System::ICloneable,
        Microsoft::VisualC::StlClr::IStack<GValue, GContainer>
{ .... };
```

參數

值

受控制序列中項目的類型。

容器

基礎容器的類型。

需求

標頭: <cliext/stack>

命名空間: cliext

宣告

宣告	說明
<code>stack::const_reference (STL/CLR)</code>	項目的常數參考類型。
<code>stack::container_type (STL/CLR)</code>	基礎容器的類型。
<code>stack::difference_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。
<code>stack::generic_container (STL/CLR)</code>	容器介面卡的泛型介面型別。
<code>stack::generic_value (STL/CLR)</code>	容器介面卡泛型介面的元素類型。
<code>stack::reference (STL/CLR)</code>	項目的參考類型。
<code>stack::size_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。

<p> </p> <p><code>stack::value_type (STL/CLR)</code></p>	<p> </p> <p>項目的類型。</p>
<p> </p> <p><code>stack::assign (STL/CLR)</code></p>	<p> </p> <p>取代所有項目。</p>
<p> </p> <p><code>stack::empty (STL/CLR)</code></p>	<p> </p> <p>測試項目是否不存在。</p>
<p> </p> <p><code>stack::get_container (STL/CLR)</code></p>	<p> </p> <p>存取基礎容器。</p>
<p> </p> <p><code>stack::pop (STL/CLR)</code></p>	<p> </p> <p>移除最後一個元素。</p>
<p> </p> <p><code>stack::push (STL/CLR)</code></p>	<p> </p> <p>加入新的最後一個元素。</p>
<p> </p> <p><code>stack::size (STL/CLR)</code></p>	<p> </p> <p>計算元素的數目。</p>
<p> </p> <p><code>stack::stack (STL/CLR)</code></p>	<p> </p> <p>建構容器物件。</p>
<p> </p> <p><code>stack::top (STL/CLR)</code></p>	<p> </p> <p>存取最後一個項目。</p>
<p> </p> <p><code>stack::to_array (STL/CLR)</code></p>	<p> </p> <p>將受控制序列複製到新的陣列。</p>
<p> </p> <p><code>stack::top_item (STL/CLR)</code></p>	<p> </p> <p>存取最後一個項目。</p>
<p> </p> <p><code>stack::operator= (STL/CLR)</code></p>	<p> </p> <p>取代受控制的序列。</p>
<p> </p> <p><code>operator ! = (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷物件是否 <code>stack</code> 不等於另一個 <code>stack</code> 物件。</p>
<p> </p> <p><code>operator< (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷 <code>stack</code> 物件是否小於另一個 <code>stack</code> 物件。</p>
<p> </p> <p><code>operator<= (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷 <code>stack</code> 物件是否小於或等於另一個 <code>stack</code> 物件。</p>
<p> </p> <p><code>operator== (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷 <code>stack</code> 物件是否等於另一個 <code>stack</code> 物件。</p>
<p> </p> <p><code>operator> (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷 <code>stack</code> 物件是否大於另一個 <code>stack</code> 物件。</p>
<p> </p> <p><code>operator>= (stack) (STL/CLR)</code></p>	<p> </p> <p>判斷 <code>stack</code> 物件是否大於或等於另一個 <code>stack</code> 物件。</p>

介面

<p> </p> <p><code>ICloneable</code></p>	<p> </p> <p>複製物件。</p>
--	------------------------

II	II
IStack<Value, Container>	維護一般容器介面卡。

備註

物件會透過儲存值元素並隨需成長的基礎容器，配置並釋放它所控制之序列的儲存體。物件會限制只推送和取出最後一個專案的存取權，以執行後進先出佇列（也稱為 LIFO 佇列或堆疊）。

成員

stack:: assign (STL/CLR)

取代所有項目。

語法

```
void assign(stack<Value, Container>% right);
```

參數

對

要插入的容器介面卡。

備註

成員函式會指派 `right.get_container()` 給基礎容器。您可以使用它來變更堆疊的整個內容。

範例

```
// cliext_stack_assign.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign a repetition of values
    Mystack c2;
    c2.assign(c1);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

stack:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```
// cliext_stack_const_reference.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " c b a"
    for ( ; !c1.empty(); c1.pop())
        {   // get a const reference to an element
            Mystack::const_reference cref = c1.top();
            System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

stack:: container_type (STL/CLR)

基礎容器的類型。

語法

```
typedef Container value_type;
```

備註

此類型與範本參數 *Container* 同義。

範例

```
// cliext_stack_container_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mystack::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

stack::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型描述可能的負元素計數。

範例

```

// cliext_stack_difference_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute negative difference
    Mystack::difference_type diff = c1.size();
    c1.push(L'd');
    c1.push(L'e');
    diff -= c1.size();
    System::Console::WriteLine("pushing 2 = {0}", diff);

    // compute positive difference
    diff = c1.size();
    c1.pop();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("popping 3 = {0}", diff);
    return (0);
}

```

```

a b c
pushing 2 = -2
popping 3 = 3

```

stack::empty (STL/CLR)

測試項目是否存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[stack::size \(STL/CLR\) \(\) == 0](#)。您可以使用它來測試堆疊是否為空的。

範例

```

// cliext_stack_empty.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.pop();
    c1.pop();
    c1.pop();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

stack:: generic_container (STL/CLR)

容器介面卡的泛型介面型別。

語法

```

typedef Microsoft::VisualC::StlClr::IStack<Value>
generic_container;

```

備註

此類型描述此範本容器介面卡類別的泛型介面。

範例

```

// cliext_stack_generic_container.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mystack::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push(L'e');
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

stack:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```

typedef GValue generic_value;

```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。（`GValue` 是，`value_type` 或者 `value_type^` `value_type` 是 ref 類型。）

範例

```

// cliext_stack_generic_value.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // get interface to container
    Mystack::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1->get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display in reverse using generic_value
    for (; !gc1->empty(); gc1->pop())
    {
        Mystack::generic_value elem = gc1->top();

        System::Console::Write("{0} ", elem);
    }
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
c b a

```

stack:: get_container (STL/CLR)

存取基礎容器。

語法

```

container_type^ get_container();

```

備註

成員函式會傳回基礎容器的控制碼。您可以使用它來略過容器包裝函式所加諸的限制。

範例

```
// cliext_stack_get_container.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c" using container_type
    Mystack::container_type wc1 = c1.get_container();
    for each (wchar_t elem in wc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

stack:: operator = (STL/CLR)

取代受控制的序列。

語法

```
stack <Value, Container>% operator=(stack <Value, Container>% right);
```

參數

對

要複製的容器介面卡。

備註

成員運算子會將 *右* 移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 *right* 中受控制序列的複本。

範例

```
// cliext_stack_operator_as.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2 = c1;
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

stack::pop (STL/CLR)

移除最後一個元素。

語法

```
void pop();
```

備註

成員函式會移除受控制序列的最後一個專案，其必須為非空白。您可以使用它來將堆疊縮短一回一個元素。

範例

```

// cliext_stack_pop.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop();
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b

```

stack::push (STL/CLR)

加入新的最後一個元素。

語法

```
void push(value_type val);
```

備註

成員函式會 `val` 在受控制序列的結尾插入具有值的元素。您可以使用它將另一個元素附加至堆疊。

範例

```

// cliext_stack_push.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

stack:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_stack_reference.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify top of stack and redisplay
    Mystack::reference ref = c1.top();
    ref = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b x
```

stack:: size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[stack:: empty \(STL/CLR\) \(\)](#)。

範例

```
// cliext_stack_size.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // pop an item and reinspect
    c1.pop();
    System::Console::WriteLine("size() = {0} after popping", c1.size());

    // add two elements and reinspect
    c1.push(L'a');
    c1.push(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}
```

```
a b c
size() = 3 starting with 3
size() = 2 after popping
size() = 4 after adding 2
```

stack:: size_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_stack_size_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    Mystack::size_type diff = c1.size();
    c1.pop();
    c1.pop();
    diff -= c1.size();
    System::Console::WriteLine("size difference = {0}", diff);
    return (0);
}

```

```

a b c
size difference = 2

```

stack:: stack (STL/CLR)

構造容器介面卡物件。

語法

```

stack();
stack(stack<Value, Container>% right);
stack(stack<Value, Container>^ right);
explicit stack(container_type% wrapped);

```

參數

對

要複製的物件。

包裹

要使用的包裝容器。

備註

函數:

```
stack();
```

建立空的包裝容器。您可以使用它來指定空的初始受控制序列。

函數:

```
stack(stack<Value, Container>% right);
```

建立包裝的容器，該容器為的複本 `right.get_container()`。您可以使用它來指定初始受控制序列，這是由 `stack` 物件 `許可權` 所控制之序列的複本。

函數：

```
stack(stack<Value, Container>^ right);
```

建立包裝的容器，該容器為的複本 `right->get_container()`。您可以使用它來指定初始受控制序列，其為堆疊物件所控制之序列的複本 `*right`。

函數：

```
explicit stack(container_type% wrapped);
```

使用 包裝 為包裝容器的現有容器。您可以使用它來從現有的容器中建立堆疊。

範例

```
// cliext_stack_construct.cpp
// compile with: /clr
#include <cliext/stack>
#include <cliext/vector>

typedef cliext::stack<wchar_t> Mystack;
typedef cliext::vector<wchar_t> Myvector;
typedef cliext::stack<wchar_t, Myvector> Mystack_vec;
int main()
{
    //
    // construct an empty container
    Mystack c1;
    System::Console::WriteLine("size() = {0}", c1.size());

    // construct from an underlying container
    Myvector v2(5, L'x');
    Mystack_vec c2(v2);
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container
    Mystack_vec c3(c2);
    for each (wchar_t elem in c3.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct by copying another container through handle
    Mystack_vec c4(%c2);
    for each (wchar_t elem in c4.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
x x x x x
x x x x x
x x x x x
```

stack:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```
cli::array<Value>^ to_array();
```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```
// cliext_stack_to_array.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push(L'd');
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

stack:: top (STL/CLR)

存取最後一個項目。

語法

```
reference top();
```

備註

成員函式會傳回受控制序列之最後一個元素的參考，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來存取最後一個專案。

範例

```

// cliext_stack_top.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top() = {0}", c1.top());

    // alter last item and reinspect
    c1.top() = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
top() = c
a b x

```

stack:: top_item (STL/CLR)

存取最後一個項目。

語法

```
property value_type top_item;
```

備註

屬性會存取受控制序列的最後一個專案，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來讀取或寫入最後一個專案。

範例

```
// cliext_stack_top_item.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("top_item = {0}", c1.top_item);

    // alter last item and reinspect
    c1.top_item = L'x';
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
top_item = c
a b x
```

stack::value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數 *值* 同義。

範例

```

// cliext_stack_value_type.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display reversed contents " a b c" using value_type
    for ( ; !c1.empty(); c1.pop())
        { // store element in value_type object
        Mystack::value_type val = c1.top();

        System::Console::Write("{0} ", val);
        }
    System::Console::WriteLine();
    return (0);
}

```

c b a

operator != (stack) (STL/CLR)

堆疊不等於比較。

語法

```

template<typename Value,
         typename Container>
bool operator!=(stack<Value, Container>% left,
                  stack<Value, Container>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個堆疊是依元素進行比較時，左邊是否未以正確的順序排序。

範例

```

// cliext_stack_operator_ne.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
        c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
        c1 != c2);
    return (0);
}

```

```

a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True

```

operator < (stack) (STL/CLR)

堆疊小於比較。

語法

```

template<typename Value,
         typename Container>
bool operator<(stack<Value, Container>% left,
                  stack<Value, Container>% right);

```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 i $!(right[i] < left[i])$ 而言也是 true $left[i] < right[i]$ 。否

則，它會傳回 `left->stack::size(STL/CLR) () < right->size()` 您使用它來測試當兩個堆疊是依元素進行比較時，是否要在`right`之前排序左。

範例

```
// cliext_stack_operator_lt.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
        c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
        c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (stack) (STL/CLR)

堆疊小於或等於比較。

語法

```
template<typename Value,
         typename Container>
bool operator<=(stack<Value, Container>% left,
                 stack<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個堆疊是依元素進行比較時，是否不會將左方排序。

範例

```
// cliext_stack_operator_le.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
        c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (stack) (STL/CLR)

堆疊相等比較。

語法

```
template<typename Value,
         typename Container>
bool operator==(stack<Value, Container>% left,
                  stack<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true `i` `left[i] == right[i]`。您可以使用它來測試當兩個堆疊是依元素進行比較時，左邊是否以相同的順序排序。

範例

```

// cliext_stack_operator_eq.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}", 
        c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}", 
        c1 == c2);
    return (0);
}

```

```
a b c  
a b d  
[a b c] == [a b c] is True  
[a b c] == [a b d] is False
```

operator > (stack) (STL/CLR)

堆疊大於比較。

語法

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個堆疊是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_stack_operator_gt.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
        c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
        c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator > = (stack) (STL/CLR)

堆疊大於或等於比較。

語法

```
template<typename Value,
         typename Container>
bool operator>=(stack<Value, Container>% left,
                  stack<Value, Container>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個堆疊是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_stack_operator_ge.cpp
// compile with: /clr
#include <cliext/stack>

typedef cliext::stack<wchar_t> Mystack;
int main()
{
    Mystack c1;
    c1.push(L'a');
    c1.push(L'b');
    c1.push(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    Mystack c2;
    c2.push(L'a');
    c2.push(L'b');
    c2.push(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2.get_container())
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```

utility (STL/CLR)

2020/11/2 • [Edit Online](#)

包含 STL/CLR 標頭 `<cliext/utility>` 來定義範本類別 `pair` 以及數個支援的範本函數。

語法

```
#include <utility>
```

需求

標頭 : `<cliext/utility>`

命名空間 : `cliext`

宣告

<code>pair (STL/CLR)</code>	包裝成對的元素。
<code>operator== (pair) (STL/CLR)</code>	配對相等比較。
<code>operator != (配對) (STL/CLR)</code>	配對不等於比較。
<code>運算子< (組) (STL/CLR)</code>	配對小於比較。
<code>operator <= (對) (STL/CLR)</code>	配對小於或等於比較。
<code>運算子> (組) (STL/CLR)</code>	配對大於比較。
<code>operator>= (組) (STL/CLR)</code>	配對大於或等於比較。
<code>make_pair (STL/CLR)</code>	從一對值進行配對。

(STL/CLR) 的配對

此範本類別描述包裝一組值的物件。

語法

```
template<typename Value1,
         typename Value2>
ref class pair;
```

參數

Value1

第一個包裝值的型別。

Value2

第二個包裝值的型別。

成員

MEMBER	描述
pair::first_type (STL/CLR)	第一個包裝值的型別。
pair::second_type (STL/CLR)	第二個包裝值的型別。
pair::first (STL/CLR)	第一個儲存的值。
pair::second (STL/CLR)	第二個儲存的值。
pair::pair (STL/CLR)	建立成對的物件。
pair::swap (STL/CLR)	交換兩組的內容。
pair::operator= (STL/CLR)	取代預存值的配對。

備註

物件會儲存一組值。您可以使用此範本類別將兩個值合併成單一物件。此外，`cliext::pair` 此處所述的物件 () 只會儲存 managed 類型；若要儲存一組非受控型別 `std::pair`，請使用在中宣告的類型 `<utility>`。

配對:: first (STL/CLR)

第一個包裝的值。

語法

```
Value1 first;
```

備註

物件會儲存第一個包裝的值。

範例

```
// cliext_pair_first.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

配對:: first_type (STL/CLR)

第一個包裝值的型別。

語法

```
typedef Value1 first_type;
```

備註

此類型與範本參數 *Value1* 同義。

範例

```
// cliext_pair_first_type.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

配對:: operator = (STL/CLR)

取代預存值的配對。

語法

```
pair<Value1, Value2>% operator=(pair<Value1, Value2>% right);
```

參數

對

要複製的配對。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它來取代預存值的配對，並在右邊儲存一組值的複本。

範例

```
// cliext_pair_operator_as.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    // assign to a new pair
    cliext::pair<wchar_t, int> c2;
    c2 = c1;
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);
    return (0);
}
```

```
[x, 3]
[x, 3]
```

配對 ::pair (STL/CLR)

建立成對的物件。

語法

```
pair();
pair(pair<Coll>% right);
pair(pair<Coll>^ right);
pair(Value1 val1, Value2 val2);
```

參數

對

要儲存的配對。

val1

要儲存的第一個值。

val2

要儲存的第二個值。

備註

函數：

```
pair();
```

使用預設的結構值，初始化預存的配對。

函數：

```
pair(pair<Value1, Value2>% right);
```

使用 `right.` 對 `:: FIRST (STL/clr)` 和組 `right.` :: second (stl/clr) , 初始化預存的配對。

```
pair(pair<Value1, Value2>^ right);
```

使用 `right->` 對 `:: FIRST (STL/clr)` 和組 `right>` :: second (stl/clr) , 初始化預存的配對。

函數:

```
pair(Value1 val1, Value2 val2);
```

使用 `val1` 和 `val2`, 初始化預存配對。

範例

```
// cliext_pair_construct.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
// construct an empty container
    cliext::pair<wchar_t, int> c1;
    System::Console::WriteLine("[{0}, {1}]",
        c1.first == L'\0' ? "\\\0" : "??", c1.second);

// construct with a pair of values
    cliext::pair<wchar_t, int> c2(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

// construct by copying another pair
    cliext::pair<wchar_t, int> c3(c2);
    System::Console::WriteLine("[{0}, {1}]", c3.first, c3.second);

// construct by copying a pair handle
    cliext::pair<wchar_t, int> c4(%c3);
    System::Console::WriteLine("[{0}, {1}]", c4.first, c4.second);

    return (0);
}
```

```
[\0, 0]
[x, 3]
[x, 3]
[x, 3]
```

成對 :: second (STL/CLR)

第二個包裝值。

語法

```
Value2 second;
```

備註

物件會儲存第二個包裝值。

範例

```
// cliext_pair_second.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

配對:: second_type (STL/CLR)

第二個包裝值的型別。

語法

```
typedef Value2 second_type;
```

備註

此類型是樣板參數 *Value2*的同義字。

範例

```
// cliext_pair_second_type.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    cliext::pair<wchar_t, int>::first_type first_val = c1.first;
    cliext::pair<wchar_t, int>::second_type second_val = c1.second;
    System::Console::WriteLine("[{0}, {1}]", first_val, second_val);
    return (0);
}
```

```
[x, 3]
```

配對:: swap (STL/CLR)

交換兩組的內容。

語法

```
void swap(pair<Value1, Value2>% right);
```

參數

對

與交換內容的配對。

備註

成員函式會將預存值的成對值 `*this` 與 右值交換。

範例

```
// cliext_pair_swap.cpp
// compile with: /clr
#include <cliext/adapter>
#include <cliext/deque>

typedef cliext::collection_adapter<
    System::Collections::ICollection> Mycoll;
int main()
{
    cliext::deque<wchar_t> d1;
    d1.push_back(L'a');
    d1.push_back(L'b');
    d1.push_back(L'c');
    Mycoll c1(%d1);

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::deque<wchar_t> d2(5, L'x');
    Mycoll c2(%d2);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x x x x x
x x x x x
a b c
```

make_pair (STL/CLR)

`pair` 從一對值進行。

語法

```
template<typename Value1,
         typename Value2>
pair<Value1, Value2> make_pair(Value1 first, Value2 second);
```

參數

Value1

第一個包裝值的型別。

Value2

第二個包裝值的型別。

first

要換行的第一個值。

second

要換行的第二個值。

備註

此範本函式會傳回 `pair<Value1, Value2>(first, second)`。您可以使用它來 `pair<Value1, Value2>` 從一對值中建立物件。

範例

```
// cliext_make_pair.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);

    c1 = cliext::make_pair(L'y', 4);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    return (0);
}
```

```
[x, 3]
[y, 4]
```

operator != (配對) (STL/CLR)

配對不等於比較。

語法

```
template<typename Value1,
         typename Value2>
bool operator!=(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);
```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試當兩個配對是依元素進行比較時，左邊是否未以正確的順序排序。

範例

```
// cliext_pair_operator_ne.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] != [x 3] is {0}",
        c1 != c1);
    System::Console::WriteLine("[x 3] != [x 4] is {0}",
        c1 != c2);
    return (0);
}
```

```
[x, 3]
[x, 4]
[x 3] != [x 3] is False
[x 3] != [x 4] is True
```

運算子 < (配對) (STL/CLR)

配對小於比較。

語法

```
template<typename Value1,
         typename Value2>
bool operator<(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);
```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `left.first < right.first || !(right.first < left.first && left.second < right.second)`。您可以使用它來測試 是否要在兩個配對是依專案進行比較時，于 右邊排序。

範例

```

// cliext_pair_operator_lt.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] < [x 3] is {0}",
        c1 < c1);
    System::Console::WriteLine("[x 3] < [x 4] is {0}",
        c1 < c2);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] < [x 3] is False
[x 3] < [x 4] is True

```

operator < = (對) (STL/CLR)

配對小於或等於比較。

語法

```

template<typename Value1,
         typename Value2>
bool operator<=(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個配對是依元素進行比較時，是否不會排序 `left < right`。

範例

```

// cliext_pair_operator_le.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] <= [x 3] is {0}",
        c1 <= c1);
    System::Console::WriteLine("[x 4] <= [x 3] is {0}",
        c2 <= c1);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] <= [x 3] is True
[x 4] <= [x 3] is False

```

operator == (對) (STL/CLR)

配對相等比較。

語法

```

template<typename Value1,
         typename Value2>
bool operator==(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `left.first == right.first && left.second == right.second`。您可以使用它來測試當兩個配對是依元素進行比較時，是否將 左 的順序與 右方相同。

範例

```

// cliext_pair_operator_eq.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] == [x 3] is {0}",
        c1 == c1);
    System::Console::WriteLine("[x 3] == [x 4] is {0}",
        c1 == c2);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] == [x 3] is True
[x 3] == [x 4] is False

```

運算子 > (配對) (STL/CLR)

配對大於比較。

語法

```

template<typename Value1,
         typename Value2>
bool operator>(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個配對是依元素進行比較時，是否要將左方排序。

範例

```

// cliext_pair_operator_gt.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] > [x 3] is {0}",
        c1 > c1);
    System::Console::WriteLine("[x 4] > [x 3] is {0}",
        c2 > c1);
    return (0);
}

```

```

[x, 3]
[x, 4]
[x 3] > [x 3] is False
[x 4] > [x 3] is True

```

operator > = (對) (STL/CLR)

配對大於或等於比較。

語法

```

template<typename Value1,
         typename Value2>
bool operator>=(pair<Value1, Value2>% left,
                  pair<Value1, Value2>% right);

```

參數

離開

要比較的左配對。

對

要比較的右側配對。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個配對是依專案進行比較時，是否要將左方排序。

範例

```
// cliext_pair_operator_ge.cpp
// compile with: /clr
#include <cliext/utility>

int main()
{
    cliext::pair<wchar_t, int> c1(L'x', 3);
    System::Console::WriteLine("[{0}, {1}]", c1.first, c1.second);
    cliext::pair<wchar_t, int> c2(L'x', 4);
    System::Console::WriteLine("[{0}, {1}]", c2.first, c2.second);

    System::Console::WriteLine("[x 3] >= [x 3] is {0}",
        c1 >= c1);
    System::Console::WriteLine("[x 3] >= [x 4] is {0}",
        c1 >= c2);
    return (0);
}
```

```
[x, 3]
[x, 4]
[x 3] >= [x 3] is True
[x 3] >= [x 4] is False
```

vector (STL/CLR)

2020/11/2 • [Edit Online](#)

此樣板類別描述一個物件，該物件可控制具有隨機存取之元素的不同長度序列。您可以使用容器 `vector`，將一連串的元素管理為連續的儲存區塊。區塊會實作為隨需成長的陣列。

在下列描述中，與 `GValue` 值相同，除非後者是 ref 型別，在這種情況下就是 `Value^`。

語法

```
template<typename Value>
ref class vector
    : public
    System::ICloneable,
    System::Collections::IEnumerable,
    System::Collections::ICollection,
    System::Collections::Generic::IEnumerable<GValue>,
    System::Collections::Generic::ICollection<GValue>,
    System::Collections::Generic::IList<GValue>,
    Microsoft::VisualC::StlClr::IVector<GValue>
{ .....
```

參數

值

受控制序列中項目的類型。

需求

標頭: <cliext/vector>

命名空間: cliext

宣告

III	II
vector::const_iterator (STL/CLR)	用於受控制序列的常數迭代器類型。
vector::const_reference (STL/CLR)	項目的常數參考類型。
vector::const_reverse_iterator (STL/CLR)	用於受控制序列的常數反向迭代器類型。
vector::difference_type (STL/CLR)	兩個項目之間帶正負號距離的類型。
vector::generic_container (STL/CLR)	容器的泛型介面型別。
vector::generic_iterator (STL/CLR)	容器之泛型介面的反覆運算器類型。
vector::generic_reverse_iterator (STL/CLR)	容器的泛型介面之反向反覆運算器的類型。
vector::generic_value (STL/CLR)	容器之泛型介面的元素類型。

vector	vector
<code>vector::iterator (STL/CLR)</code>	受控制序列之迭代器的類型。
<code>vector::reference (STL/CLR)</code>	項目的參考類型。
<code>vector::reverse_iterator (STL/CLR)</code>	受控制序列的反向迭代器類型。
<code>vector::size_type (STL/CLR)</code>	兩個項目之間帶正負號距離的類型。
<code>vector::value_type (STL/CLR)</code>	項目的類型。
vector	vector
<code>vector::assign (STL/CLR)</code>	取代所有項目。
<code>vector::at (STL/CLR)</code>	存取指定位置的項目。
<code>vector::back (STL/CLR)</code>	存取最後一個項目。
<code>vector::begin (STL/CLR)</code>	指定受控制序列的開頭。
<code>vector::capacity (STL/CLR)</code>	報告容器配置儲存區的大小。
<code>vector::clear (STL/CLR)</code>	移除所有項目。
<code>vector::empty (STL/CLR)</code>	測試項目是否不存在。
<code>vector::end (STL/CLR)</code>	指定受控制序列的結尾。
<code>vector::erase (STL/CLR)</code>	移除位於指定位置的項目。
<code>vector::front (STL/CLR)</code>	存取第一個項目。
<code>vector::insert (STL/CLR)</code>	將專案加入至指定的位置。
<code>vector::pop_back (STL/CLR)</code>	移除最後一個元素。
<code>vector::push_back (STL/CLR)</code>	加入新的最後一個元素。
<code>vector::rbegin (STL/CLR)</code>	指定反向受控制序列的開頭。
<code>vector::rend (STL/CLR)</code>	指定反向受控制序列的結尾。
<code>vector::reserve (STL/CLR)</code>	確保容器的成長容量下限。
<code>vector::resize (STL/CLR)</code>	變更項目的數目。
<code>vector::size (STL/CLR)</code>	計算元素的數目。
<code>vector::swap (STL/CLR)</code>	交換兩個容器的內容。

vector	
<code>vector::to_array (STL/CLR)</code>	將受控制序列複製到新的陣列。
<code>vector::vector (STL/CLR)</code>	建構容器物件。
vector	
<code>vector::back_item (STL/CLR)</code>	存取最後一個項目。
<code>vector::front_item (STL/CLR)</code>	存取第一個項目。
vector	
<code>vector::operator= (STL/CLR)</code>	取代受控制的序列。
<code>vector::operator (STL/CLR)</code>	存取指定位置的項目。
<code>operator != (vector) (STL/CLR)</code>	判斷物件是否 <code>vector</code> 不等於另一個 <code>vector</code> 物件。
<code>運算子< (vector) (STL/CLR)</code>	判斷 <code>vector</code> 物件是否小於另一個 <code>vector</code> 物件。
<code>operator<= (向量) (STL/CLR)</code>	判斷 <code>vector</code> 物件是否小於或等於另一個 <code>vector</code> 物件。
<code>operator == (vector) (STL/CLR)</code>	判斷 <code>vector</code> 物件是否等於另一個 <code>vector</code> 物件。
<code>operator> (vector) (STL/CLR)</code>	判斷 <code>vector</code> 物件是否大於另一個 <code>vector</code> 物件。
<code>operator>= (向量) (STL/CLR)</code>	判斷 <code>vector</code> 物件是否大於或等於另一個 <code>vector</code> 物件。

介面

ICloneable	複製物件。
IEnumerable	排序元素。
ICollection	維護元素群組。
IEnumerable<T>	透過具類型的元素排序。
ICollection<T>	維護具類型的元素群組。
IList<T>	維護具類型專案的已排序群組。
IVector<值>	維護泛型容器。

備註

物件會透過儲存的 `值` 元素陣列，配置並釋放它所控制之序列的儲存體，而這些專案會隨需求成長。成長的發生方

式，是將新元素附加的成本分攤為固定的时间。換句話說，在最後新增專案的成本不會增加，因為受控制序列的長度會變大。因此，向量是樣板類別 [堆疊 \(STL/CLR\)](#) 的基礎容器的絕佳候選。

`vector` 支援隨機存取反覆運算器，這表示您可以直接參考專案的數值位置，從零算起的第一個 (front) 元素，直到 `size() - 1` 最後一個 (的) 專案為止。這也表示向量是樣板類別 [priority_queue \(STL/CLR\)](#) 的基礎容器的絕佳候選。

向量反覆運算器會將控制碼儲存至其相關聯的向量物件，以及它所指定之元素的偏差。您只能使用反覆運算器與其相關聯的容器物件。Vector 元素的偏差與其位置相同。

插入或清除元素可以變更儲存在指定位置的專案值，因此 iterator 所指定的值也可以變更。(容器可能必須將專案向上或向下複製，以在插入或在清除後填滿洞之前建立洞。) 不過，只要向量反覆運算器在範圍內，就會維持 `[0, size()]` 有效。此外，有效的 iterator 仍維持 dereferencable--您可以使用它來存取或修改它所指定的元素值，只要其偏差不等於 `size()`。

清除或移除專案會呼叫其預存值的函式。終結容器會清除所有元素。因此，其元素類型為 ref 類別的容器可確保沒有任何專案存留時間容器。不過請注意，控制碼的容器不會摧毀其元素。

成員

`vector:: assign (STL/CLR)`

取代所有項目。

語法

```
void assign(size_type count, value_type val);
template<typename InIt>
    void assign(InIt first, InIt last);
void assign(System::Collections::Generic::IEnumarable<Value>^ right);
```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的元素值。

備註

第一個成員函式會以值 *val* 的 *count* 元素重複來取代受控制的序列。您可以使用它來填滿具有相同值之元素的容器。

如果 `InIt` 是整數類型，則第二個成員函式的行為會與相同 `assign((size_type)first, (value_type)last)` 。否則，它會以順序 `[,)` 取代受控制的序列 `first last`。您可以使用它來讓受控制序列的複製另一個序列。

第三個成員函式會將受控制序列取代為列舉值 右邊所指定的序列。您可以使用它來讓受控制的序列成為列舉值所描述之序列的複本。

範例

```

// cliext_vector_assign.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // assign a repetition of values
    cliext::vector<wchar_t> c2;
    c2.assign(6, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an iterator range
    c2.assign(c1.begin(), c1.end() - 1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign an enumeration
    c2.assign( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

x x x x x x
a b
a b c

```

vector:: at (STL/CLR)

存取指定位置的項目。

語法

```
reference at(size_type pos);
```

參數

pos

要存取的項目的位置。

備註

成員函式會傳回位置 *pos* 之受控制序列之元素的參考。您可以使用它來讀取或寫入您知道其位置的元素。

範例

```
// cliext_vector_at.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using at
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // change an entry and redisplay
    c1.at(1) = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a x c
```

vector:: back (STL/CLR)

存取最後一個項目。

語法

```
reference back();
```

備註

成員函式會傳回受控制序列之最後一個元素的參考，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來存取最後一個專案。

範例

```
// cliext_vector_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back() = {0}", c1.back());

    // alter last item and reinspect
    c1.back() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
back() = c
a b x
```

vector:: back_item (STL/CLR)

存取最後一個項目。

語法

```
property value_type back_item;
```

備註

屬性會存取受控制序列的最後一個專案，其必須為非空白。當您知道最後一個元素存在時，您可以使用它來讀取或寫入最後一個專案。

範例

```

// cliext_vector_back_item.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last item
    System::Console::WriteLine("back_item = {0}", c1.back_item);

    // alter last item and reinspect
    c1.back_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
back_item = c
a b x

```

vector::begin (STL/CLR)

指定受控制序列的開頭。

語法

```
iterator begin();
```

備註

成員函式會傳回隨機存取反覆運算器，此反覆運算器會指定受控制序列的第一個專案，或在空序列結尾以外的第一個元素。您要用它來取得的 Iterator 可指定受控制序列之 `current` 開頭，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_vector_begin.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::vector<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("*begin() = {0}", *it);
    System::Console::WriteLine("*++begin() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*begin() = a
*++begin() = b
x y c

```

vector::容量 (STL/CLR)

報告容器配置儲存區的大小。

語法

```
size_type capacity();
```

備註

成員函式會傳回目前配置用來保存受控制序列的儲存體，此值至少與[vector:: size \(STL/CLR\)](#) 相同 `()`。您可以使用它來判斷容器在必須為受控制序列重新配置儲存體之前，可成長的程度。

範例

```
// cliext_vector_capacity.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // increase capacity
    cliext::vector<wchar_t>::size_type cap = c1.capacity();
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        cap, c1.size() <= cap);
    c1.reserve(cap + 5);
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        c1.capacity(), cap + 5 <= c1.capacity());
    return (0);
}
```

```
a b c
capacity() = 4, ok = True
capacity() = 9, ok = True
```

vector:: clear (STL/CLR)

移除所有項目。

語法

```
void clear();
```

備註

成員函式會有效地呼叫[vector:: erase \(stl/clr\) \(\)](#) [vector:: begin \(stl/clr\) \(\)](#), [vector:: end \(STL/clr\) \(\)](#)。您可以使用它來確保受控制的序列是空的。

範例

```

// cliext_vector_clear.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');

    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}

```

```

a b c
size() = 0
a b
size() = 0

```

vector:: const_iterator (STL/CLR)

用於受控制序列的常數迭代器類型。

語法

```

typedef T2 const_iterator;

```

備註

型別描述未指定類型的物件 `T2`，可作為受控制序列的常數隨機存取反覆運算器。

範例

```
// cliext_vector_const_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::vector<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        System::Console::Write("{0} ", *cit);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

vector:: const_reference (STL/CLR)

項目的常數參考類型。

語法

```
typedef value_type% const_reference;
```

備註

型別描述元素的常數參考。

範例

```
// cliext_vector_const_reference.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::vector<wchar_t>::const_iterator cit = c1.begin();
    for (; cit != c1.end(); ++cit)
        { // get a const reference to an element
        cliext::vector<wchar_t>::const_reference cref = *cit;
        System::Console::Write("{0} ", cref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
```

vector::const_reverse_iterator (STL/CLR)

受控制序列的常數反向反覆運算器類型。

語法

```
typedef T4 const_reverse_iterator;
```

備註

型別描述未指定類型的物件 `T4`，可作為受控制序列的常數反向反覆運算器。

範例

```
// cliext_vector_const_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::vector<wchar_t>::const_reverse_iterator crit = c1.rbegin();
    cliext::vector<wchar_t>::const_reverse_iterator crend = c1.rend();
    for (; crit != crend; ++crit)
        System::Console::Write("{0} ", *crit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
```

vector::difference_type (STL/CLR)

兩個元素之間的帶正負號距離類型。

語法

```
typedef int difference_type;
```

備註

此類型會描述已簽署的元素計數。

範例

```

// cliext_vector_difference_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::vector<wchar_t>::difference_type diff = 0;
    for (cliext::vector<wchar_t>::iterator it = c1.begin();
         it != c1.end(); ++it) ++diff;
    System::Console::WriteLine("end()-begin() = {0}", diff);

    // compute negative difference
    diff = 0;
    for (cliext::vector<wchar_t>::iterator it = c1.end();
         it != c1.begin(); --it) --diff;
    System::Console::WriteLine("begin()-end() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3
begin()-end() = -3

```

vector::empty (STL/CLR)

測試項目是否不存在。

語法

```
bool empty();
```

備註

成員函式會對空的受控制序列傳回 true。它相當於[vector::size \(STL/CLR\)](#) `() == 0`。您可以使用它來測試向量是否為空白。

範例

```

// cliext_vector_empty.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0}", c1.size());
    System::Console::WriteLine("empty() = {0}", c1.empty());
    return (0);
}

```

```

a b c
size() = 3
empty() = False
size() = 0
empty() = True

```

vector::end (STL/CLR)

指定受控制序列的結尾。

語法

```
iterator end();
```

備註

成員函式會傳回隨機存取反覆運算器，其指向受控制序列的結尾以外的位置。您會用它來取得指定受控制序列之 **current** 結尾的 Iterator，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```

// cliext_vector_end.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect last two items
    cliext::vector<wchar_t>::iterator it = c1.end();
    --it;
    System::Console::WriteLine("*-- --end() = {0}", *--it);
    System::Console::WriteLine("*--end() = {0}", *++it);

    // alter first two items and reinspect
    *--it = L'x';
    *++it = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*-- --end() = b
*--end() = c
a x y

```

vector:: erase (STL/CLR)

移除位於指定位置的項目。

語法

```

iterator erase(iterator where);
iterator erase(iterator first, iterator last);

```

參數

first

要清除的範圍開頭。

last

要清除的範圍結尾。

where

要清除的元素。

備註

第一個成員函式會移除由 *where*所指向之受控制序列的元素。您可以使用它來移除單一專案。

第二個成員函式會移除 [*first* , *last*) 範圍中受控制序列中的元素。您可以使用它來移除零個或多個連續元素。

這兩個成員函式會傳回反覆運算器，此反覆運算器會指定任何移除的元素之外的第一個元素，如果沒有這類專案存在，則[vector:: end \(STL/CLR\)](#) ()。

清除專案時，專案複本的數目會是抹除結尾和序列最接近端之間的元素數目的線性。(清除序列結尾的一或多個元素時，不會複製任何元素。)

範例

```
// cliext_vector_erase.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase an element and reinspect
    System::Console::WriteLine("erase(begin()) = {0}",
        *c1.erase(c1.begin()));

    // add elements and display " b c d e"
    c1.push_back(L'd');
    c1.push_back(L'e');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // erase all but end
    cliext::vector<wchar_t>::iterator it = c1.end();
    System::Console::WriteLine("erase(begin(), end()-1) = {0}",
        *c1.erase(c1.begin(), --it));
    System::Console::WriteLine("size() = {0}", c1.size());
    return (0);
}
```

```
a b c
erase(begin()) = b
b c d e
erase(begin(), end()-1) = e
size() = 1
```

vector:: front (STL/CLR)

存取第一個項目。

語法

```
reference front();
```

備註

成員函式會傳回受控制序列的第一個元素的參考，而該專案必須為非空白。當您知道第一個元素存在時，您可以使用它來讀取或寫入第一個專案。

範例

```
// cliext_vector_front.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front() = {0}", c1.front());

    // alter first item and reinspect
    c1.front() = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
front() = a
x b c
```

vector::front_item (STL/CLR)

存取第一個項目。

語法

```
property value_type front_item;
```

備註

屬性會存取受控制序列的第一個專案，其必須為非空白。當您知道第一個元素存在時，您可以使用它來讀取或寫入第一個專案。

範例

```

// cliext_vector_front_item.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first item
    System::Console::WriteLine("front_item = {0}", c1.front_item);

    // alter first item and reinspect
    c1.front_item = L'x';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
front_item = a
x b c

```

vector:: generic_container (STL/CLR)

容器的泛型介面型別。

語法

```

typedef Microsoft::VisualC::StlClr::
    IVector<generic_value>
generic_container;

```

備註

此類型描述此範本容器類別的泛型介面。

範例

```

// cliext_vector_generic_container.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    gc1->insert(gc1->end(), L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify original and display generic
    c1.push_back(L'e');

    System::Collections::IEnumerator^ enum1 =
        gc1->GetEnumerator();
    while (enum1->MoveNext())
        System::Console::Write("{0} ", enum1->Current);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a b c d
a b c d e

```

vector:: generic_iterator (STL/CLR)

反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ContainerRandomAccessIterator<generic_value>
generic_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反覆運算器。

範例

```

// cliext_vector_generic_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

vector:: generic_reverse_iterator (STL/CLR)

反向反覆運算器的類型，用於容器的泛型介面。

語法

```

typedef Microsoft::VisualC::StlClr::Generic::
    ReverseRandomAccessIterator<generic_value> generic_reverse_iterator;

```

備註

此類型描述可搭配此樣板容器類別的泛型介面使用的泛型反向反覆運算器。

範例

```

// cliext_vector_generic_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_reverse_iterator gcit = gc1->rbegin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a c c

```

vector:: generic_value (STL/CLR)

要搭配容器的泛型介面使用的元素類型。

語法

```
typedef GValue generic_value;
```

備註

型別描述型別的物件，此物件 `GValue` 描述與這個樣板容器類別的泛型介面搭配使用的預存專案值。

範例

```

// cliext_vector_generic_value.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct a generic container
    cliext::vector<wchar_t>::generic_container^ gc1 = %c1;
    for each (wchar_t elem in gc1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // modify generic and display original
    cliext::vector<wchar_t>::generic_iterator gcit = gc1->begin();
    cliext::vector<wchar_t>::generic_value gcval = *gcit;
    *++gcit = gcval;
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b c
a a c

```

vector:: insert (STL/CLR)

將專案加入至指定的位置。

語法

```

iterator insert(iterator where, value_type val);
void insert(iterator where, size_type count, value_type val);
template<typename InIt>
    void insert(iterator where, InIt first, InIt last);
void insert(iterator where,
    System::Collections::Generic::IEnumerable<Value>^ right);

```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的列舉。

瓦爾

要插入的元素值。

where

在容器中要插入的位置。

備註

每個成員函式會在受控制序列中的 *位置* 所指向的專案之前，插入由其餘運算元所指定的序列。

第一個成員函式會插入具有值 *val* 的元素，並傳回指定新插入之元素的反覆運算器。您可以使用它，在反覆運算器所指定的位置之前插入單一元素。

第二個成員函式會插入值 *val* 的 *count* 元素重複。您可以使用它來插入零個或多個連續的元素，這些都是相同值的所有複本。

如果 `Init` 是整數類型，第三個成員函式的行為即與 `insert(where, (size_type)first, (value_type)last)` 相同。否則，它會插入順序 [`first` , `last`)。您可以使用它來插入零個或多個從另一個序列複製的連續元素。

第四個成員函式會插入 右邊指定的順序。您可以使用它來插入列舉值所描述的序列。

插入單一專案時，專案複本的數目是在插入點與序列最接近端之間的專案數中的線性。(在序列的任一結尾插入一或多個專案時，不會複製任何專案。) 如果 `Init` 是輸入反覆運算器，則第三個成員函式會針對序列中的每個元素，有效地執行單一插入。否則，插入專案時 `N`，專案複本的數目會是線性，`N` 加上插入點與序列最接近端之間的元素數目。

範例

```

// cliext_vector_insert.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a single value using iterator
    cliext::vector<wchar_t>::iterator it = c1.begin();
    System::Console::WriteLine("insert(begin())+1, L'x') = {0}",
        *c1.insert(++it, L'x'));
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert a repetition of values
    cliext::vector<wchar_t> c2;
    c2.insert(c2.begin(), 2, L'y');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an iterator range
    it = c1.end();
    c2.insert(c2.end(), c1.begin(), --it);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // insert an enumeration
    c2.insert(c2.begin(),   // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c1);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
insert(begin())+1, L'x') = x
a x b c
y y
y y a x b
a x b c y y a x b

```

vector:: iterator (STL/CLR)

受控制序列之迭代器的類型。

語法

```
typedef T1 iterator;
```

備註

型別描述未指定類型的物件 `T1`，可作為受控制序列的隨機存取反覆運算器。

範例

```
// cliext_vector_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    cliext::vector<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();

    // alter first element and redisplay
    it = c1.begin();
    *it = L'x';
    for (; it != c1.end(); ++it)
        System::Console::Write("{0} ", *it);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
x b c
```

vector:: operator = (STL/CLR)

取代受控制的序列。

語法

```
vector<Value>% operator=(vector<Value>% right);
```

參數

對

要複製的容器。

備註

成員運算子會將右移至物件，然後傳回 `*this`。您可以使用它，將受控制序列取代為 `right` 中受控制序列的複本。

範例

```
// cliext_vector_operator_as.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2 = c1;
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a b c
```

vector:: operator (STL/CLR)

存取指定位置的項目。

語法

```
reference operator[](size_type pos);
```

參數

pos

要存取的項目的位置。

備註

成員運算子會將 referene 傳回至位置 *pos*的元素。您可以使用它來存取您知道其位置的元素。

範例

```
// cliext_vector_operator_sub.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using subscripting
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();

    // change an entry and redisplay
    c1[1] = L'x';
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1[i]);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
a x c
```

vector::p op_back (STL/CLR)

移除最後一個元素。

語法

```
void pop_back();
```

備註

成員函式會移除受控制序列的最後一個專案，其必須為非空白。您可以使用它來將向量減少一個元素的後面。

範例

```

// cliext_vector_pop_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // pop an element and redisplay
    c1.pop_back();
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
a b

```

vector::push_back (STL/CLR)

加入新的最後一個元素。

語法

```
void push_back(value_type val);
```

備註

成員函式會 `val` 在受控制序列的結尾插入具有值的元素。您可以使用它來將另一個元素附加至向量。

範例

```

// cliext_vector_push_back.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```
a b c
```

vector:: rbegin (STL/CLR)

指定反向受控制序列的開頭。

語法

```
reverse_iterator rbegin();
```

備註

成員函式會傳回反向反覆運算器，此反覆運算器會指定受控制序列的最後一個專案，或在空白序列的開頭之外。因此，它會指定反向序列的 `beginning`。您會用它來取得指定以反向順序顯示之受控制序列 `current` 開頭的 Iterator，但是如果受控制序列的長度變更，它的狀態也會變更。

範例

```
// cliext_vector_rbegin.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items in reversed sequence
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rbegin();
    System::Console::WriteLine("*rbegin() = {0}", *rit);
    System::Console::WriteLine("*++rbegin() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
*rbegin() = c
*++rbegin() = b
a y x
```

vector:: reference (STL/CLR)

項目的參考類型。

語法

```
typedef value_type% reference;
```

備註

型別描述對元素的參考。

範例

```
// cliext_vector_reference.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    cliext::vector<wchar_t>::iterator it = c1.begin();
    for (; it != c1.end(); ++it)
        {   // get a reference to an element
            cliext::vector<wchar_t>::reference ref = *it;
            System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();

    // modify contents " a b c"
    for (it = c1.begin(); it != c1.end(); ++it)
        {   // get a reference to an element
            cliext::vector<wchar_t>::reference ref = *it;

            ref += (wchar_t)(L'A' - L'a');
            System::Console::Write("{0} ", ref);
        }
    System::Console::WriteLine();
    return (0);
}
```

```
a b c
A B C
```

vector:: rend (STL/CLR)

指定反向受控制序列的結尾。

語法

```
reverse_iterator rend();
```

備註

成員函式會傳回指向受控制序列開頭以外的反向反覆運算器。因此，它會指定反向序列的 `end`。您要用它來取得的 Iterator 可指定以相反順序顯示的受控制序列之 `current` 結尾，但是，如果受控制序列的長度變更，它的狀態也可以變更。

範例

```

// cliext_vector_rend.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // inspect first two items
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rend();
    --rit;
    System::Console::WriteLine("*-- --rend() = {0}", *--rit);
    System::Console::WriteLine("*--rend() = {0}", *++rit);

    // alter first two items and reinspect
    *--rit = L'x';
    *++rit = L'y';
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
*-- --rend() = b
*--rend() = a
y x c

```

vector:: reserve (STL/CLR)

確保容器的成長容量下限。

語法

```
void reserve(size_type count);
```

參數

計數

容器的新最小容量。

備註

成員函式可確保 `capacity()` 因而需要至少會傳回 計數。您可以使用它來確保容器不需要重新配置受控制序列的儲存體，直到其成長到指定的大小。

範例

```

// cliext_vector_reserve.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for (int i = 0; i < c1.size(); ++i)
        System::Console::Write("{0} ", c1.at(i));
    System::Console::WriteLine();

    // increase capacity
    cliext::vector<wchar_t>::size_type cap = c1.capacity();
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        cap, c1.size() <= cap);
    c1.reserve(cap + 5);
    System::Console::WriteLine("capacity() = {0}, ok = {1}",
        c1.capacity(), cap + 5 <= c1.capacity());
    return (0);
}

```

```

a b c
capacity() = 4, ok = True
capacity() = 9, ok = True

```

vector:: resize (STL/CLR)

變更項目的數目。

語法

```

void resize(size_type new_size);
void resize(size_type new_size, value_type val);

```

參數

new_size

受控制序列的新大小。

瓦爾

填補元素的值。

備註

成員函式可確保`vector:: size (STL/CLR) ()`因而需要會傳回*new_size*。如果它必須讓受控制序列的時間變長，第一個成員函式會附加具有值的元素`value_type()`，而第二個成員函式會附加具有值 *va*的元素。為了讓受控制的序列變得更短，這兩個成員函式會有效地清除最後一個元素`vector:: size (STL/CLR) () - new_size`時間。您可以使用它來確保受控制序列的大小 *new_size*，方法是修剪或填補目前的受控制序列。

範例

```
// cliext_vector_resize.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
// construct an empty container and pad with default values
    cliext::vector<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());
    c1.resize(4);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// resize to empty
    c1.resize(0);
    System::Console::WriteLine("size() = {0}", c1.size());

// resize and pad
    c1.resize(5, L'x');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
size() = 0
0 0 0 0
size() = 0
x x x x x
```

vector:: reverse_iterator (STL/CLR)

受控制序列的反向迭代器類型。

語法

```
typedef T3 reverse_iterator;
```

備註

此類型描述未指定類型 `T3` 的物件，其可用作受控制序列的反向迭代器。

範例

```
// cliext_vector_reverse_iterator.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" reversed
    cliext::vector<wchar_t>::reverse_iterator rit = c1.rbegin();
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();

    // alter first element and redisplay
    rit = c1.rbegin();
    *rit = L'x';
    for (; rit != c1.rend(); ++rit)
        System::Console::Write("{0} ", *rit);
    System::Console::WriteLine();
    return (0);
}
```

```
c b a
x b a
```

vector::size (STL/CLR)

計算元素的數目。

語法

```
size_type size();
```

備註

成員函式會傳回受控制序列的長度。您可以使用它來判斷目前在受控制序列中的元素數目。如果您只在意順序是否有非零的大小，請參閱[vector::empty \(STL/CLR\) \(\)](#)。

範例

```

// cliext_vector_size.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    System::Console::WriteLine("size() = {0} starting with 3", c1.size());

    // clear the container and reinspect
    c1.clear();
    System::Console::WriteLine("size() = {0} after clearing", c1.size());

    // add elements and clear again
    c1.push_back(L'a');
    c1.push_back(L'b');
    System::Console::WriteLine("size() = {0} after adding 2", c1.size());
    return (0);
}

```

```

a b c
size() = 3 starting with 3
size() = 0 after clearing
size() = 2 after adding 2

```

vector::size_type (STL/CLR)

兩個項目之間帶正負號距離的類型。

語法

```
typedef int size_type;
```

備註

型別描述非負的元素計數。

範例

```

// cliext_vector_size_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // compute positive difference
    cliext::vector<wchar_t>::size_type diff = c1.end() - c1.begin();
    System::Console::WriteLine("end()-begin() = {0}", diff);
    return (0);
}

```

```

a b c
end()-begin() = 3

```

vector:: swap (STL/CLR)

交換兩個容器的內容。

語法

```
void swap(vector<Value>% right);
```

參數

對

要交換內容的容器。

備註

成員函式會交換和右邊的受控制序列 `*this`。`right` 它會以常數時間來執行，且不會擲回任何例外狀況。您可以使用它來快速交換兩個容器的內容。

範例

```

// cliext_vector_swap.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display initial contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // construct another container with repetition of values
    cliext::vector<wchar_t> c2(5, L'x');
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // swap and redisplay
    c1.swap(c2);
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}

```

```

a b c
x x x x x
x x x x x
a b c

```

vector:: to_array (STL/CLR)

將受控制序列複製到新的陣列。

語法

```

cli::array<Value>^ to_array();

```

備註

成員函式會傳回陣列，其中包含受控制的序列。您可以使用它，以陣列形式取得受控制序列的複本。

範例

```
// cliext_vector_to_array.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // copy the container and modify it
    cli::array<wchar_t>^ a1 = c1.to_array();

    c1.push_back(L'd');
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // display the earlier array copy
    for each (wchar_t elem in a1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();
    return (0);
}
```

```
a b c d
a b c
```

vector::value_type (STL/CLR)

項目的類型。

語法

```
typedef Value value_type;
```

備註

此類型與範本參數 `Value` 同義。

範例

```

// cliext_vector_value_type.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c" using value_type
    for (cliext::vector<wchar_t>::iterator it = c1.begin();
        it != c1.end(); ++it)
    {
        // store element in value_type object
        cliext::vector<wchar_t>::value_type val = *it;

        System::Console::Write("{0} ", val);
    }
    System::Console::WriteLine();
    return (0);
}

```

a b c

vector:: vector (STL/CLR)

建構容器物件。

語法

```

vector();
vector(vector<Value>% right);
vector(vector<Value>^ right);
explicit vector(size_type count);
vector(size_type count, value_type val);
template<typename InIt>
    vector(InIt first, InIt last);
vector(System::Collections::Generic::IEnumerable<Value>^ right);

```

參數

計數

要插入的元素數目。

first

要插入的範圍開頭。

last

要插入的範圍結尾。

對

要插入的物件或範圍。

瓦爾

要插入的元素值。

備註

函數：

```
vector();
```

使用沒有元素的方式，初始化受控制的序列。您可以使用它來指定空的初始受控制序列。

函數：

```
vector(vector<Value>% right);
```

使用序列 [,) 初始化受控制的序列 `right.begin()` `right.end()`。您可以使用它來指定初始受控制序列，這是由 `vector` 物件 右邊所控制之序列的複本。

函數：

```
vector(vector<Value>^ right);
```

使用序列 [,) 初始化受控制的序列 `right->begin()` `right->end()`。您可以使用它來指定初始受控制序列，其為控制碼 正確的向量物件所控制的序列複本。

函數：

```
explicit vector(size_type count);
```

使用 `count` 元素(具有值)初始化受控制的序列 `value_type()`。您可以使用它來填滿具有預設值的元素的容器。

函數：

```
vector(size_type count, value_type val);
```

使用具有值 `val` 的 `count` 元素，初始化受控制的序列。您可以使用它來填滿具有相同值之元素的容器。

函數：

```
template<typename InIt>
```

```
vector(InIt first, InIt last);
```

使用序列 [,) 初始化受控制的序列 `first` `last`。您可以使用它來讓受控制的序列成為另一個序列的複本。

函數：

```
vector(System::Collections::Generic::IEnumerable<Value>^ right);
```

使用列舉值 右邊指定的順序，初始化受控制的序列。您可以使用它來讓受控制的序列成為列舉值所描述之另一個序列的複本。

範例

```

// cliext_vector_construct.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
// construct an empty container
    cliext::vector<wchar_t> c1;
    System::Console::WriteLine("size() = {0}", c1.size());

// construct with a repetition of default values
    cliext::vector<wchar_t> c2(3);
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", (int)elem);
    System::Console::WriteLine();

// construct with a repetition of values
    cliext::vector<wchar_t> c3(6, L'x');
    for each (wchar_t elem in c3)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an iterator range
    cliext::vector<wchar_t>::iterator it = c3.end();
    cliext::vector<wchar_t> c4(c3.begin(), --it);
    for each (wchar_t elem in c4)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct with an enumeration
    cliext::vector<wchar_t> c5( // NOTE: cast is not needed
        (System::Collections::Generic::IEnumerable<wchar_t>^)%c3);
    for each (wchar_t elem in c5)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying another container
    cliext::vector<wchar_t> c7(c3);
    for each (wchar_t elem in c7)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

// construct by copying a container handle
    cliext::vector<wchar_t> c8(%c3);
    for each (wchar_t elem in c8)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    return (0);
}

```

```

size() = 0
0 0 0
x x x x x
x x x x x
x x x x x x
x x x x x x
x x x x x x

```

operator != (vector) (STL/CLR)

向量不等於比較。

語法

```
template<typename Value>
bool operator!=(vector<Value>% left,
                 vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left == right)`。您可以使用它來測試兩個向量是依元素進行比較時，左邊是否未以正確的順序排序。

範例

```
// cliext_vector_operator_ne.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] != [a b c] is {0}",
                             c1 != c1);
    System::Console::WriteLine("[a b c] != [a b d] is {0}",
                             c1 != c2);
    return (0);
}
```

```
a b c
a b d
[a b c] != [a b c] is False
[a b c] != [a b d] is True
```

運算子 < (vector) (STL/CLR)

向量小於比較。

語法

```
template<typename Value>
bool operator<(vector<Value>% left,
                 vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

如果是，運算子函式會傳回 true，如果是，則對 `i !(right[i] < left[i])` 而言也是 true `left[i] < right[i]`。否則，它會傳回 `left->size() < right->size()` 您使用它來測試當兩個向量 `right` 是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_vector_operator_lt.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] < [a b c] is {0}",
                             c1 < c1);
    System::Console::WriteLine("[a b c] < [a b d] is {0}",
                             c1 < c2);
    return (0);
}
```

```
a b c
a b d
[a b c] < [a b c] is False
[a b c] < [a b d] is True
```

operator < = (vector) (STL/CLR)

向量小於或等於比較。

語法

```
template<typename Value>
bool operator<=(vector<Value>% left,
                 vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(right < left)`。您可以使用它來測試當兩個向量是依元素進行比較時，左邊是否未排序。

範例

```
// cliext_vector_operator_le.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] <= [a b c] is {0}",
                             c1 <= c1);
    System::Console::WriteLine("[a b d] <= [a b c] is {0}",
                             c2 <= c1);
    return (0);
}
```

```
a b c
a b d
[a b c] <= [a b c] is True
[a b d] <= [a b c] is False
```

operator == (vector) (STL/CLR)

向量相等比較。

語法

```
template<typename Value>
bool operator==(vector<Value>% left,
                  vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

只有在由左至右控制的序列和每個位置都有相同的長度和時，運算子函式才會傳回 true `i` `left[i] == right[i]`。您可以使用它來測試當兩個向量是依元素進行比較時，左邊是否以相同的順序排序。

範例

```
// cliext_vector_operator_eq.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] == [a b c] is {0}",
                             c1 == c1);
    System::Console::WriteLine("[a b c] == [a b d] is {0}",
                             c1 == c2);
    return (0);
}
```

```
a b c
a b d
[a b c] == [a b c] is True
[a b c] == [a b d] is False
```

運算子 > (vector) (STL/CLR)

向量大於比較。

語法

```
template<typename Value>
bool operator>(vector<Value>% left,
                  vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `right < left`。您可以使用它來測試當兩個向量是依元素進行比較時，是否要將左方排序。

範例

```
// cliext_vector_operator_gt.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] > [a b c] is {0}",
                           c1 > c1);
    System::Console::WriteLine("[a b d] > [a b c] is {0}",
                           c2 > c1);
    return (0);
}
```

```
a b c
a b d
[a b c] > [a b c] is False
[a b d] > [a b c] is True
```

operator >= (vector) (STL/CLR)

向量大於或等於比較。

語法

```
template<typename Value>
bool operator>=(vector<Value>% left,
                 vector<Value>% right);
```

參數

離開

要比較的左容器。

對

要比較的右容器。

備註

Operator 函數會傳回 `!(left < right)`。您可以使用它來測試當兩個向量是依專案進行比較時，左邊是否未排序。

範例

```
// cliext_vector_operator_ge.cpp
// compile with: /clr
#include <cliext/vector>

int main()
{
    cliext::vector<wchar_t> c1;
    c1.push_back(L'a');
    c1.push_back(L'b');
    c1.push_back(L'c');

    // display contents " a b c"
    for each (wchar_t elem in c1)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    // assign to a new container
    cliext::vector<wchar_t> c2;
    c2.push_back(L'a');
    c2.push_back(L'b');
    c2.push_back(L'd');

    // display contents " a b d"
    for each (wchar_t elem in c2)
        System::Console::Write("{0} ", elem);
    System::Console::WriteLine();

    System::Console::WriteLine("[a b c] >= [a b c] is {0}",
                             c1 >= c1);
    System::Console::WriteLine("[a b c] >= [a b d] is {0}",
                             c1 >= c2);
    return (0);
}
```

```
a b c
a b d
[a b c] >= [a b c] is True
[a b c] >= [a b d] is False
```


C++ 支援程式庫

2020/3/25 • [Edit Online](#)

C++支援程式庫提供的類別可支援中C++的 managed 程式設計。

本節內容

[C++ 中封送處理的概觀](#)

[資源管理類別](#)

[同步處理 \(lock 類別\)](#)

[呼叫特定應用程式定義域的函式](#)

[com::ptr](#)

C++/CLI 中封送的概述

2020/4/15 • [Edit Online](#)

在混合模式下，有時必須在本機類型和託管類型之間封送數據。封送庫可說明您以簡單方式封送和轉換數據。封送庫由一組函數和一個 `marshal_context` 類組成，這些函數和類執行公共類型的封送。函式庫在 Visual Studio 版的「包括/msclr」目錄中的這些標頭中定義：

II	II
元帥.h	<code>marshal_context</code> 類別和無上下文封送函數
marshal_atl.h	封送 ATL 類型的函數
marshal_cppstd.h	封送標準C++類型的函數
marshal_windows.h	封送 Windows 型態的函式

msclr 資料夾的預設路徑如下所示，具體取決於您擁有的版本和內部版本號：

```
C:\Program Files (x86)\Microsoft Visual  
Studio\Preview\Enterprise\VC\Tools\MSVC\14.15.26528\include\msclr
```

您可以使用帶或不帶 `marshal_context` 類的封送庫。某些轉換需要上下文。可以使用 `marshal_as` 函數實現其他轉換。下表列出了支援的目前轉換、是否需要上下文以及必須包括哪些封送檔：

III	II		
系統:字串*	康斯特字元*	<code>marshal_context</code>	元帥.h
康斯特字元*	系統:字串*	<code>marshal_as</code>	元帥.h
字元*	系統:字串*	<code>marshal_as</code>	元帥.h
系統:字串*	康斯特wchar_t*	<code>marshal_context</code>	元帥.h
康斯特wchar_t*	系統:字串*	<code>marshal_as</code>	元帥.h
wchar_t *	系統:字串*	<code>marshal_as</code>	元帥.h
系統::IntPtr	HANDLE	<code>marshal_as</code>	marshal_windows.h
HANDLE	系統::IntPtr	<code>marshal_as</code>	marshal_windows.h
系統:字串*	BSTR	<code>marshal_context</code>	marshal_windows.h
BSTR	系統:字串*	<code>marshal_as</code>	元帥.h
系統:字串*	bstr_t	<code>marshal_as</code>	marshal_windows.h

封送	封送	封送	封送
bstr_t	系統:字串*	marshal_as	marshal_windows.h
系統:字串*	std::字串	marshal_as	marshal_cppstd.h
std::字串	系統:字串*	marshal_as	marshal_cppstd.h
系統:字串*	std::wstring	marshal_as	marshal_cppstd.h
std::wstring	系統:字串*	marshal_as	marshal_cppstd.h
系統:字串*	CStringT<字元>	marshal_as	marshal_atl.h
CStringT<字元>	系統:字串*	marshal_as	marshal_atl.h
系統:字串*	CStringt<wchar_t>	marshal_as	marshal_atl.h
CStringt<wchar_t>	系統:字串*	marshal_as	marshal_atl.h
系統:字串*	CComBSTR	marshal_as	marshal_atl.h
CComBSTR	系統:字串*	marshal_as	marshal_atl.h

封送僅在從託管數據類型封送到本機數據類型且要轉換為的本機類型沒有用於自動清理的析構函數時,才需要上下文。封送上下文銷毀其析構函數中分配的本機數據類型。因此,需要上下文的轉換僅在刪除上下文之前才有效。要保存任何封送值,必須將值複製到自己的變數。

NOTE

如果字串中嵌入 `NULL` 了 s, 則不能保證對字串進行封送的結果。嵌入 `NULL` 的 s 可能導致字串被截斷, 或者可能保留它們。

此範例展示如何在包含標頭宣告中包括 msclr 目錄:

```
#include "msclr\marshal_cppstd.h"
```

封送庫是可擴展的,因此您可以添加自己的封送類型。有關擴展封送庫的詳細資訊,請參閱[如何:擴展封送庫](#)。

另請參閱

[C++ 支援程式庫](#)

[如何:擴充封送處理程式庫](#)

marshal_as

2019/12/10 • [Edit Online](#)

這個方法會在原生和 managed 環境之間轉換資料。

語法

```
To_Type marshal_as<To_Type>(
    From_Type input
);
```

參數

input

在要封送處理至 `To_Type` 變數的值。

傳回值

`To_Type` 類型的變數，這是 `input` 的轉換值。

備註

這個方法是在原生和 managed 類型之間轉換資料的簡化方式。若要判斷支援哪些資料類型，請參閱[中C++的封送處理總覽](#)。某些資料轉換需要內容。您可以使用[Marshal_Context 類別](#)來轉換這些資料類型。

如果您嘗試封送處理一對不支援的資料類型，`marshal_as` 會在編譯時期產生錯誤[C4996](#)。如需詳細資訊，請參閱此錯誤所提供的訊息。不只是已被取代的函式，也可以產生 `C4996` 錯誤。其中一個範例是嘗試封送處理一對不支援的資料類型。

封送處理程式庫是由數個標頭檔所組成。任何轉換只需要一個檔案，但如果需要進行其他轉換，則可以包含其他檔案。若要查看哪些轉換與哪些檔案相關聯，請查看 [Marshaling Overview](#) 中的表格。不論您想要進行何種轉換，命名空間需求一律有效。

如果輸入參數為 `null`，則會擲回 `System::ArgumentNullException(_EXCEPTION_NULLPTR)`。

範例

這個範例會從 `const char*` 封送處理至 `System::String` 變數類型。

```
// marshal_as_test.cpp
// compile with: /clr
#include <stdlib.h>
#include <string.h>
#include <msclr\marshal.h>

using namespace System;
using namespace msclr::interop;

int main() {
    const char* message = "Test String to Marshal";
    String^ result;
    result = marshal_as<String^>( message );
    return 0;
}
```

需求

標頭檔：`<msclr\marshal.h>`、`<msclr\marshal_windows.h>`、`<msclr\marshal_cppstd.h>` 或 `<msclr\marshal_atl.h>`

命名空間：`msclr::interop`

請參閱

[C++ 中封送處理的概觀](#)

[marshal_context 類別](#)

marshal_context 類別

2020/11/2 • [Edit Online](#)

這個類別會在原生和 Managed 環境之間轉換。

語法

```
class marshal_context
```

備註

為需要內容的資料轉換使用 `marshal_context` 類別。如需有關需要內容的轉換以及必須包含哪些封送處理檔案的詳細資訊，請參閱 [c++ 中的封送處理總覽](#)。在您使用內容時的封送處理結果的有效性只到 `marshal_context` 物件終結時。若要保存結果，您必須複製資料。

`marshal_context` 也可以用於許多資料轉換。以這種方式重複使用內容，並不會影響先前封送處理呼叫的結果。

成員

公用建構函式

II	II
<code>marshal_context::marshal_context</code>	<code>marshal_context</code> 對 managed 和原生資料類型之間的資料轉換，建立要使用的物件。
<code>marshal_context::~marshal_context</code>	終結 <code>marshal_context</code> 物件。

公用方法

II	II
<code>marshal_context::marshal_as</code>	在特定資料物件上執行封送處理，以便在 Managed 資料類型和原生資料類型之間進行轉換。

需求

標頭檔：`<msclr\marshal.h>`、`<msclr\marshal_windows.h>`、`<msclr\marshal_cppstd.h>` 或
`<msclr\marshal_atl.h>`

命名空間：`msclr::interop`

marshal_context::marshal_context

`marshal_context` 對 managed 和原生資料類型之間的資料轉換，建立要使用的物件。

```
marshal_context();
```

備註

某些資料轉換需要封送處理內容。如需有關哪些翻譯需要內容以及您必須在應用程式中包含哪些封送處理檔案的詳細資訊，請參閱 [c++ 中的封送處理總覽](#)。

範例

請參閱 [marshal_context::marshal_as](#) 的範例。

marshal_context:: ~marshal_context

終結 `marshal_context` 物件。

```
~marshal_context();
```

備註

某些資料轉換需要封送處理內容。請參閱 [c++ 中的封送處理總覽](#)，以取得哪些翻譯需要內容，以及哪些封送處理檔案必須包含在應用程式中的詳細資訊。

刪除 `marshal_context` 物件將會使要由該內容轉換的資料失效。如果想要在終結 `marshal_context` 物件之後保留資料，您必須手動將資料複製至會保存的變數。

marshal_context:: marshal_as

在特定資料物件上執行封送處理，以便在 Managed 資料類型和原生資料類型之間進行轉換。

```
To_Type marshal_as<To_Type>(
    From_Type input
);
```

參數

input

在要封送處理至變數的值 `To_Type`。

傳回值

型別的變數 `To_Type`，這是的轉換值 `input`。

備註

這個函式會在特定資料物件上執行封送處理。此函數僅適用於在 [c++ 中的封送處理總覽](#) 中，資料表所指出的轉換。

如果您嘗試封送處理一對不支援的資料類型，`marshal_as` 將會在編譯時期產生錯誤 [C4996](#)。如需詳細資訊，請閱讀此錯誤所提供的訊息。[C4996](#) 除了被取代的函式之外，可能會產生錯誤。產生此錯誤的兩個條件是嘗試封送處理一組不支援的資料類型，並嘗試將其用於 `marshal_as` 需要內容的轉換。

封送處理程式庫包含數個標頭檔。任何轉換都只需要一個檔案，但如果需要進行其他轉換，則可以包含其他檔案。[Marshaling Overview in C++](#) 中的表格所指出的封送處理檔案應該包含在每次轉換中。

範例

這個範例會建立從 `System::String` 到 `const char *` 變數類型的封送處理的內容。已轉換的資料在刪除內容的那一行之後將無法有效。

```
// marshal_context_test.cpp
// compile with: /clr
#include <stdlib.h>
#include <string.h>
#include <msclr\marshal.h>

using namespace System;
using namespace msclr::interop;

int main() {
    marshal_context^ context = gcnew marshal_context();
    String^ message = gcnew String("Test String to Marshal");
    const char* result;
    result = context->marshal_as<const char*>( message );
    delete context;
    return 0;
}
```

msclr 命名空間

2019/12/2 • [Edit Online](#)

msclr 命名空間包含 C++ 支援程式庫的所有類別。如需有關這些類別的詳細資訊，請參閱 < [C++ 支援程式庫](#)。

另請參閱

[C++ 支援程式庫](#)

資源管理類別

2020/11/2 • • [Edit Online](#)

這些類別提供 Managed 類別的自動管理。

本節內容

[auto_gcroot](#)

在原生類型中嵌入虛擬控制代碼。

[auto_handle](#)

在 Managed 類型中嵌入虛擬控制代碼。

另請參閱

[C++ 支援程式庫](#)

auto_gcroot

2019/12/2 • [Edit Online](#)

定義 `auto_gcroot` 類別和 `swap` 函式。

語法

```
#include <msclr\auto_gcroot.h>
```

備註

在此標頭檔中：

[auto_gcroot 類別](#)

[swap 函式 \(auto_gcroot\)](#)

另請參閱

[C++ 支援程式庫](#)

auto_gcroot 類別

2020/11/2 • [Edit Online](#)

自動資源管理(例如[Auto_ptr 類別](#))，可以用來將虛擬控制碼內嵌至原生類型。

語法

```
template<typename _element_type>
class auto_gcroot;
```

參數

_element_type

要內嵌的 managed 類型。

成員

公用建構函式

名稱	說明
<code>auto_gcroot:: auto_gcroot</code>	<code>auto_gcroot</code> 函數。
<code>auto_gcroot::~auto_gcroot</code>	<code>auto_gcroot</code> 析構函式。

公用方法

名稱	說明
<code>auto_gcroot::attach</code>	附加 <code>auto_gcroot</code> 至物件。
<code>auto_gcroot::get</code>	取得包含的物件。
<code>auto_gcroot::release</code>	從管理釋放物件 <code>auto_gcroot</code> 。
<code>auto_gcroot::reset</code>	終結目前擁有的物件，並選擇性地擁有新的物件。
<code>auto_gcroot::swap</code>	交換物件與另一個物件 <code>auto_gcroot</code> 。

公用運算子

名稱	說明
<code>auto_gcroot:: operator-></code>	成員存取運算子。
<code>auto_gcroot::operator=</code>	指派運算子。
<code>auto_gcroot:: operator auto_gcroot</code>	與相容類型之間的類型轉換運算子 <code>auto_gcroot</code> 。

II

II

auto_gcroot:: operator bool

auto_gcroot 在條件運算式中使用的運算子。

auto_gcroot:: operator !

auto_gcroot 在條件運算式中使用的運算子。

需求

標頭檔 <msclr\auto_gcroot.h>

命名空間 msclr

auto_gcroot:: auto_gcroot

auto_gcroot 函數。

```
auto_gcroot(
    _element_type _ptr = nullptr
);
auto_gcroot(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot(
    auto_gcroot<_other_type> & _right
);
```

參數

_ptr

要擁有的物件。

_right

現有的 auto_gcroot。

備註

從現有的建立時 auto_gcroot auto_gcroot，現有的會在將物件的 auto_gcroot 擁有權轉移給新的之前釋放其物件 auto_gcroot。

範例

```
// msl_auto_gcroot_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class RefClassA {
protected:
    String^ m_s;
public:
    RefClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in RefClassA constructor: " + m_s );
    }
    ~RefClassA() {
        Console::WriteLine( "in RefClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
```

```

        Console::WriteLine( "Hello from {0} A!", m_s );
    }

};

ref class RefClassB : RefClassA {
public:
    RefClassB( String^ s ) : RefClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

class ClassA { //unmanaged class
private:
    auto_gcroot<RefClassA^> m_a;

public:
    ClassA() : m_a( gcnew RefClassA( "unmanaged" ) ) {}
    ~ClassA() {} //no need to delete m_a

    void DoSomething() {
        m_a->PrintHello();
    }
};

int main()
{
{
    ClassA a;
    a.DoSomething();
} // a.m_a is automatically destroyed as a goes out of scope

{
    auto_gcroot<RefClassA^> a(gcnew RefClassA( "first" ) );
    a->PrintHello();
}

{
    auto_gcroot<RefClassB^> b(gcnew RefClassB( "second" ) );
    b->PrintHello();
    auto_gcroot<RefClassA^> a(b); //construct from derived type
    a->PrintHello();
    auto_gcroot<RefClassA^> a2(a); //construct from same type
    a2->PrintHello();
}

Console::WriteLine("done");
}

```

```

in RefClassA constructor: unmanaged
Hello from unmanaged A!
in RefClassA destructor: unmanaged
in RefClassA constructor: first
Hello from first A!
in RefClassA destructor: first
in RefClassA constructor: second
Hello from second B!
Hello from second A!
Hello from second A!
in RefClassA destructor: second
done

```

auto_gcroot:: ~ auto_gcroot

auto_gcroot 析構函式。

```
~auto_gcroot();
```

備註

此析構函式也會 destructs 所擁有的物件。

範例

```
// msl_auto_gcroot_dtor.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
public:
    ClassA() { Console::WriteLine( "ClassA constructor" ); }
    ~ClassA() { Console::WriteLine( "ClassA destructor" ); }
};

int main()
{
    // create a new scope for a:
    {
        auto_gcroot<ClassA^> a = gcnew ClassA;
    }
    // a goes out of scope here, invoking its destructor
    // which in turns destructs the ClassA object.

    Console::WriteLine( "done" );
}
```

```
ClassA constructor
ClassA destructor
done
```

auto_gcroot:: attach

附加 `auto_gcroot` 至物件。

```
auto_gcroot<_element_type> & attach(
    _element_type _right
);
auto_gcroot<_element_type> & attach(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot<_element_type> & attach(
    auto_gcroot<_other_type> & _right
);
```

參數

`_right`

要附加的物件，或 `auto_gcroot` 包含要附加之物件的。

傳回值

目前的 `auto_gcroot`。

備註

如果 `_right` 是 `auto_gcroot`，它會先釋放其物件的擁有權，然後物件才會附加至目前的 `auto_gcroot`。

範例

```
// msl_auto_gcroot_attach.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_gcroot<ClassA^> a( gcnew ClassA( "first" ) );
    a->PrintHello();
    a.attach( gcnew ClassA( "second" ) ); // attach same type
    a->PrintHello();
    ClassA^ ha = gcnew ClassA( "third" );
    a.attach( ha ); // attach raw handle
    a->PrintHello();
    auto_gcroot<ClassB^> b( gcnew ClassB("fourth") );
    b->PrintHello();
    a.attach( b ); // attach derived type
    a->PrintHello();
}
```

```
in ClassA constructor:first
Hello from first A!
in ClassA constructor:second
in ClassA destructor:first
Hello from second A!
in ClassA constructor:third
in ClassA destructor:second
Hello from third A!
in ClassA constructor:fourth
Hello from fourth B!
in ClassA destructor:third
Hello from fourth A!
in ClassA destructor:fourth
```

auto_gcroot:: get

取得包含的物件。

```
_element_type get() const;
```

傳回值

包含的物件。

範例

```
// ms1_auto_gcroot_get.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ){
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

void PrintA( ClassA^ a ) {
    a->PrintHello();
}

int main() {
    auto_gcroot<ClassA^> a = gcnew ClassA( "first" );
    a->PrintHello();

    ClassA^ a2 = a.get();
    a2->PrintHello();

    PrintA( a.get() );
}
```

```
in ClassA constructor:first
Hello from first A!
Hello from first A!
Hello from first A!
in ClassA destructor:first
```

auto_gcroot:: release

從管理釋放物件 `auto_gcroot`。

```
_element_type release();
```

傳回值

已釋放的物件。

範例

```
// ms1_auto_gcroot_release.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    ClassA^ a;

    // create a new scope:
    {
        auto_gcroot<ClassA^> agc1 = gcnew ClassA( "first" );
        auto_gcroot<ClassA^> agc2 = gcnew ClassA( "second" );
        a = agc1.release();
    }
    // agc1 and agc2 go out of scope here

    a->PrintHello();

    Console::WriteLine( "done" );
}
```

```
ClassA constructor: first
ClassA constructor: second
ClassA destructor: second
Hello from first A!
done
```

auto_gcroot:: reset

終結目前擁有的物件，並選擇性地擁有新的物件。

```
void reset(
    _element_type _new_ptr = nullptr
);
```

參數

_new_ptr

選擇性新的物件。

範例

```
// msl_auto_gcroot_reset.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    auto_gcroot<ClassA^> agc1 = gcnew ClassA( "first" );
    agc1->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    agc1.reset( ha ); // release first object, reference second
    agc1->PrintHello();

    agc1.reset(); // release second object, set to nullptr

    Console::WriteLine( "done" );
}
```

```
ClassA constructor: first
Hello from first A!
ClassA constructor: second
ClassA destructor: first
Hello from second A!
ClassA destructor: second
done
```

auto_gcroot::swap

交換物件與另一個物件 `auto_gcroot`。

```
void swap(
    auto_gcroot<_element_type> & _right
);
```

參數

`_right`

`auto_gcroot` 用來交換物件的。

範例

```

// msl_auto_gcroot_swap.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s1 = "string one";
    auto_gcroot<String^> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'", 
        s1->ToString(), s2->ToString() );
    s1.swap( s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}

```

```

s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'

```

auto_gcroot:: operator->

成員存取運算子。

```

_element_type operator->() const;

```

傳回值

包裝的物件 `auto_gcroot`。

範例

```

// msl_auto_gcroot_op_arrow.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }

    int m_i;
};

int main() {
    auto_gcroot<ClassA^> a( gcnew ClassA( "first" ) );
    a->PrintHello();

    a->m_i = 5;
    Console::WriteLine( "a->m_i = {0}", a->m_i );
}

```

```
Hello from first A!
a->m_i = 5
```

auto_gcroot:: operator =

指派運算子。

```
auto_gcroot<_element_type> & operator=(
    _element_type _right
);
auto_gcroot<_element_type> & operator=(
    auto_gcroot<_element_type> & _right
);
template<typename _other_type>
auto_gcroot<_element_type> & operator=(
    auto_gcroot<_other_type> & _right
);
```

參數

_right

要 `auto_gcroot` 指派給目前的物件 `auto_gcroot` 。

傳回值

目前的 `auto_gcroot`，現在擁有 `_right` 。

範例

```

// msclr_auto_gcroot_operator_equals.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main()
{
    auto_gcroot<ClassA^> a;
    auto_gcroot<ClassA^> a2(gcnew ClassA( "first" ) );
    a = a2; // assign from same type
    a->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    a = ha; // assign from raw handle

    auto_gcroot<ClassB^> b(gcnew ClassB( "third" ) );
    b->PrintHello();
    a = b; // assign from derived type
    a->PrintHello();

    Console::WriteLine("done");
}

```

```

in ClassA constructor: first
Hello from first A!
in ClassA constructor: second
in ClassA destructor: first
in ClassA constructor: third
Hello from third B!
in ClassA destructor: second
Hello from third A!
done
in ClassA destructor: third

```

auto_gcroot:: operator auto_gcroot

與相容類型之間的類型轉換運算子 `auto_gcroot`。

```
template<typename _other_type>
operator auto_gcroot<_other_type>();
```

傳回值

目前 `auto_gcroot` 轉換成的 `auto_gcroot<_other_type>`。

範例

```
// msl_auto_gcroot_op_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_gcroot<ClassB^> b = gcnew ClassB("first");
    b->PrintHello();
    auto_gcroot<ClassA^> a = (auto_gcroot<ClassA^>)b;
    a->PrintHello();
}
```

```
Hello from first B!
Hello from first A!
```

auto_gcroot:: operator bool

`auto_gcroot` 在條件運算式中使用的運算子。

```
operator bool() const;
```

傳回值

`true` 如果包裝的物件有效，則為，`false` 否則為。

備註

這個運算子會實際將轉換成 `_detail_class::_safe_bool`，這比更安全，`bool` 因為它無法轉換成整數類資料類型。

範例

```
// msl_auto_gcroot_operator_bool.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s;
    if ( s ) Console::WriteLine( "s is valid" );
    if ( !s ) Console::WriteLine( "s is invalid" );
    s = "something";
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
    s.reset();
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
}
```

```
s is invalid
now s is valid
now s is invalid
```

auto_gcroot:: operator !

auto_gcroot 在條件運算式中使用的運算子。

```
bool operator!() const;
```

傳回值

true 如果包裝的物件無效，則為，false 否則為。

範例

```
// msl_auto_gcroot_operator_not.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s;
    if ( s ) Console::WriteLine( "s is valid" );
    if ( !s ) Console::WriteLine( "s is invalid" );
    s = "something";
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
    s.reset();
    if ( s ) Console::WriteLine( "now s is valid" );
    if ( !s ) Console::WriteLine( "now s is invalid" );
}
```

```
s is invalid
now s is valid
now s is invalid
```

swap 函式 (auto_gcroot)

2020/11/2 • [Edit Online](#)

交換一個物件 `auto_gcroot` 與另一個物件。

語法

```
template<typename _element_type>
void swap(
    auto_gcroot<_element_type> & _left,
    auto_gcroot<_element_type> & _right
);
```

參數

`_left`

另一個 `auto_gcroot`。

`_right`

另一個 `auto_gcroot`。

範例

```
// ms1_swap_auto_gcroot.cpp
// compile with: /clr
#include <msclr\auto_gcroot.h>

using namespace System;
using namespace msclr;

int main() {
    auto_gcroot<String^> s1 = "string one";
    auto_gcroot<String^> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
    swap( s1, s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}
```

```
s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'
```

需求

標頭檔 `<msclr\auto_gcroot.h>`

命名空間 `msclr`

另請參閱

[auto_gcroot](#)

auto_gcroot::swap

auto_handle

2019/12/2 • [Edit Online](#)

定義 `auto_handle` 類別和 `swap` 函式。

語法

```
#include <msclr\auto_handle.h>
```

備註

在此標頭檔中：

[auto_handle 類別](#)

[swap 函式 \(auto_handle\)](#)

另請參閱

[C++ 支援程式庫](#)

auto_handle 類別

2020/11/2 • [Edit Online](#)

自動資源管理，可以用來將虛擬控制碼內嵌至 managed 類型。

語法

```
template<typename _element_type>
ref class auto_handle;
```

參數

_element_type

要內嵌的 managed 類型。

屬於

公用建構函式

名稱	說明
auto_handle::auto_handle	<code>auto_handle</code> 函數。
auto_handle:: ~auto_handle	<code>auto_handle</code> 析構函式。

公用方法

名稱	說明
auto_handle::get	取得包含的物件。
auto_handle::release	從管理釋放物件 <code>auto_handle</code> 。
auto_handle::reset	終結目前擁有的物件，並選擇性地擁有新的物件。
auto_handle::swap	交換物件與另一個物件 <code>auto_handle</code> 。

公用運算子

名稱	說明
auto_handle:: operator->	成員存取運算子。
auto_handle::operator=	指派運算子。
auto_handle::operator auto_handle	與相容類型之間的類型轉換運算子 <code>auto_handle</code> 。
auto_handle::operator bool	<code>auto_handle</code> 在條件運算式中使用的運算子。

II

II

auto_handle:: operator !

auto_handle 在條件運算式中使用的運算子。

需求

標頭檔 <msclr\auto_handle.h>

命名空間 msclr

auto_handle:: auto_handle

auto_handle 函數。

```
auto_handle();
auto_handle(
    _element_type ^ _ptr
);
auto_handle(
    auto_handle<_element_type> % _right
);
template<typename _other_type>
auto_handle(
    auto_handle<_other_type> % _right
);
```

參數

_ptr

要擁有的物件。

_right

現有的 auto_handle。

範例

```

// ms1_auto_handle_auto_handle.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;
ref class RefClassA {
protected:
    String^ m_s;
public:
    RefClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in RefClassA constructor: " + m_s );
    }
    ~RefClassA() {
        Console::WriteLine( "in RefClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!" , m_s );
    }
};

ref class RefClassB : RefClassA {
public:
    RefClassB( String^ s ) : RefClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!" , m_s );
    }
};

int main()
{
{
    auto_handle<RefClassA> a(gcnew RefClassA( "first" ) );
    a->PrintHello();
}

{
    auto_handle<RefClassB> b(gcnew RefClassB( "second" ) );
    b->PrintHello();
    auto_handle<RefClassA> a(b); //construct from derived type
    a->PrintHello();
    auto_handle<RefClassA> a2(a); //construct from same type
    a2->PrintHello();
}

    Console::WriteLine("done");
}

```

```

in RefClassA constructor: first
Hello from first A!
in RefClassA destructor: first
in RefClassA constructor: second
Hello from second B!
Hello from second A!
Hello from second A!
in RefClassA destructor: second
done

```

auto_handle:: ~ auto_handle

auto_handle 析構函式。

```
~auto_handle();
```

備註

此析構函式也會 destructs 所擁有的物件。

範例

```
// msclr_auto_handle_dtor.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;

ref class ClassA {
public:
    ClassA() { Console::WriteLine( "ClassA constructor" ); }
    ~ClassA() { Console::WriteLine( "ClassA destructor" ); }
};

int main()
{
    // create a new scope for a:
    {
        auto_handle<ClassA> a = gcnew ClassA;
    }
    // a goes out of scope here, invoking its destructor
    // which in turns destructs the ClassA object.

    Console::WriteLine( "done" );
}
```

```
ClassA constructor
ClassA destructor
done
```

auto_handle:: get

取得包含的物件。

```
_element_type ^ get();
```

傳回值

包含的物件。

範例

```

// msclr_auto_handle_get.cpp
// compile with: /clr
#include "msclr\auto_handle.h"

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ){
        Console::WriteLine( "in ClassA constructor:" + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor:" + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

void PrintA( ClassA^ a ) {
    a->PrintHello();
}

int main() {
    auto_handle<ClassA> a = gcnew ClassA( "first" );
    a->PrintHello();

    ClassA^ a2 = a.get();
    a2->PrintHello();

    PrintA( a.get() );
}

```

```

in ClassA constructor:first
Hello from first A!
Hello from first A!
Hello from first A!
in ClassA destructor:first

```

auto_handle::release

從管理釋放物件 `auto_handle`。

```
_element_type ^ release();
```

傳回值

已釋放的物件。

範例

```

// msclr_auto_handle_release.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    ClassA^ a;

    // create a new scope:
    {
        auto_handle<ClassA> agc1 = gcnew ClassA( "first" );
        auto_handle<ClassA> agc2 = gcnew ClassA( "second" );
        a = agc1.release();
    }
    // agc1 and agc2 go out of scope here

    a->PrintHello();

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
ClassA constructor: second
ClassA destructor: second
Hello from first A!
done

```

auto_handle::reset

終結目前擁有的物件，並選擇性地擁有新的物件。

```

void reset(
    _element_type ^ _new_ptr
);
void reset();

```

參數

_new_ptr

選擇性新的物件。

範例

```

// msclr_auto_handle_reset.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {
        Console::WriteLine( "ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "ClassA destructor: " + m_s );
    }

    void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

int main()
{
    auto_handle<ClassA> agc1 = gcnew ClassA( "first" );
    agc1->PrintHello();

    ClassA^ ha = gcnew ClassA( "second" );
    agc1.reset( ha ); // release first object, reference second
    agc1->PrintHello();

    agc1.reset(); // release second object, set to nullptr

    Console::WriteLine( "done" );
}

```

```

ClassA constructor: first
Hello from first A!
ClassA constructor: second
ClassA destructor: first
Hello from second A!
ClassA destructor: second
done

```

auto_handle::swap

交換物件與另一個物件 `auto_handle`。

```

void swap(
    auto_handle<_element_type> % _right
);

```

參數

`_right`

`auto_handle` 用來交換物件的。

範例

```

// msl_auto_handle_swap.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1 = "string one";
    auto_handle<String> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'", 
        s1->ToString(), s2->ToString() );
    s1.swap( s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}

```

```

s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'

```

auto_handle:: operator->

成員存取運算子。

```

_element_type ^ operator->();

```

傳回值

包裝的物件 `auto_handle`。

範例

```

// msl_auto_handle_op_arrow.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }

    int m_i;
};

int main() {
    auto_handle<ClassA> a( gcnew ClassA( "first" ) );
    a->PrintHello();

    a->m_i = 5;
    Console::WriteLine( "a->m_i = {0}", a->m_i );
}

```

```
Hello from first A!
a->m_i = 5
```

auto_handle:: operator =

指派運算子。

```
auto_handle<_element_type> % operator=(  
    auto_handle<_element_type> % _right  
>;  
template<typename _other_type>  
auto_handle<_element_type> % operator=(  
    auto_handle<_other_type> % _right  
>;
```

參數

_right

目前的 `auto_handle` 要指派給目前的 `auto_handle`。

傳回值

目前的 `auto_handle`，現在擁有 `_right`。

範例

```

// msclr_auto_handle_op_assign.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA(String^ s) : m_s(s) {
        Console::WriteLine( "in ClassA constructor: " + m_s );
    }
    ~ClassA() {
        Console::WriteLine( "in ClassA destructor: " + m_s );
    }

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main()
{
    auto_handle<ClassA> a;
    auto_handle<ClassA> a2(gcnew ClassA( "first" ) );
    a = a2; // assign from same type
    a->PrintHello();

    auto_handle<ClassB> b(gcnew ClassB( "second" ) );
    b->PrintHello();
    a = b; // assign from derived type
    a->PrintHello();

    Console::WriteLine("done");
}

```

```

in ClassA constructor: first
Hello from first A!
in ClassA constructor: second
Hello from second B!
in ClassA destructor: first
Hello from second A!
done
in ClassA destructor: second

```

auto_handle:: operator auto_handle

與相容類型之間的類型轉換運算子 `auto_handle`。

```

template<typename _other_type>
operator auto_handle<_other_type>();

```

傳回值

目前 `auto_handle` 轉換成的 `auto_handle<_other_type>`。

範例

```
// msl_auto_handle_op_auto_handle.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

ref class ClassA {
protected:
    String^ m_s;
public:
    ClassA( String^ s ) : m_s( s ) {}

    virtual void PrintHello() {
        Console::WriteLine( "Hello from {0} A!", m_s );
    }
};

ref class ClassB : ClassA {
public:
    ClassB( String ^ s ) : ClassA( s ) {}
    virtual void PrintHello() new {
        Console::WriteLine( "Hello from {0} B!", m_s );
    }
};

int main() {
    auto_handle<ClassB> b = gcnew ClassB("first");
    b->PrintHello();
    auto_handle<ClassA> a = (auto_handle<ClassA>)b;
    a->PrintHello();
}
```

```
Hello from first B!
Hello from first A!
```

auto_handle:: operator bool

`auto_handle` 在條件運算式中使用的運算子。

```
operator bool();
```

傳回值

`true` 如果包裝的物件有效，則為，`false` 否則為。

備註

這個運算子會實際轉換為，`_detail_class::_safe_bool` `bool` 因為它無法轉換成整數類資料類型。

範例

```
// msl_auto_handle_operator_bool.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1;
    auto_handle<String> s2 = "hi";
    if ( s1 ) Console::WriteLine( "s1 is valid" );
    if ( !s1 ) Console::WriteLine( "s1 is invalid" );
    if ( s2 ) Console::WriteLine( "s2 is valid" );
    if ( !s2 ) Console::WriteLine( "s2 is invalid" );
    s2.reset();
    if ( s2 ) Console::WriteLine( "s2 is now valid" );
    if ( !s2 ) Console::WriteLine( "s2 is now invalid" );
}
```

```
s1 is invalid
s2 is valid
s2 is now invalid
```

auto_handle:: operator !

`auto_handle` 在條件運算式中使用的運算子。

```
bool operator!();
```

傳回值

`true` 如果包裝的物件無效，則為，`false` 否則為。

範例

```
// msl_auto_handle_operator_not.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1;
    auto_handle<String> s2 = "something";
    if ( s1 ) Console::WriteLine( "s1 is valid" );
    if ( !s1 ) Console::WriteLine( "s1 is invalid" );
    if ( s2 ) Console::WriteLine( "s2 is valid" );
    if ( !s2 ) Console::WriteLine( "s2 is invalid" );
    s2.reset();
    if ( s2 ) Console::WriteLine( "s2 is now valid" );
    if ( !s2 ) Console::WriteLine( "s2 is now invalid" );
}
```

```
s1 is invalid
s2 is valid
s2 is now invalid
```

swap 函式 (auto_handle)

2020/11/2 • • [Edit Online](#)

交換一個物件 `auto_handle` 與另一個物件。

語法

```
template<typename _element_type>
void swap(
    auto_handle<_element_type>% _left,
    auto_handle<_element_type>% _right
);
```

參數

`_left`

另一個 `auto_handle`。

`_right`

另一個 `auto_handle`。

範例

```
// ms1_swap_auto_handle.cpp
// compile with: /clr
#include <msclr\auto_handle.h>

using namespace System;
using namespace msclr;

int main() {
    auto_handle<String> s1 = "string one";
    auto_handle<String> s2 = "string two";

    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
    swap( s1, s2 );
    Console::WriteLine( "s1 = '{0}', s2 = '{1}'",
        s1->ToString(), s2->ToString() );
}
```

```
s1 = 'string one', s2 = 'string two'
s1 = 'string two', s2 = 'string one'
```

需求

標頭檔 `<msclr\auto_handle.h>`

命名空間 `msclr`

另請參閱

[auto_handle](#)

auto_handle::swap

同步處理 (lock 類別)

2019/12/2 • [Edit Online](#)

提供一種機制，可自動化取得物件的同步存取鎖定。

本節內容

[lock](#)

另請參閱

[C++ 支援程式庫](#)

鎖定

2020/3/18 • [Edit Online](#)

定義 `lock` 類別，以自動同步處理對物件的存取。

語法

```
#include <msclr\lock.h>
```

備註

在此標頭檔中：

`lock` 類別

`lock_when` 列舉

另請參閱

[C++ 支援程式庫](#)

lock 類別

2020/11/2 • • [Edit Online](#)

這個類別會自動進行鎖定，以同步處理從數個執行緒對物件的存取。當建立時，它會取得鎖定，而當終結時，會釋放鎖定。

語法

```
ref class lock;
```

備註

`lock` 僅適用於 CLR 物件，且只能在 CLR 程式碼中使用。

就內部而言，鎖定類別會使用 [Monitor](#) 來同步存取。如需詳細資訊，請參閱參考的文章。

成員

公用建構函式

■	■
<code>lock::lock</code>	<code>lock</code> 會建立物件，並選擇性地在指定的一段時間內（或完全不會）等待取得鎖定。
<code>lock:: ~ lock</code>	Destructs <code>lock</code> 物件。

公用方法

■	■
<code>lock::acquire</code>	取得物件的鎖定，選擇性地等候一段指定的時間，或完全不取得鎖定。
<code>lock::is_locked</code>	指出是否持有鎖定。
<code>lock::release</code>	釋放鎖定。
<code>lock::try_acquire</code>	取得物件的鎖定，等候指定的時間量，並傳回 <code>bool</code> 來回報取得成功，而不是擲回例外狀況。

公用運算子

■	■
<code>lock:: operator bool</code>	<code>lock</code> 在條件運算式中使用的運算子。
<code>lock::operator==</code>	等號比較運算子。

lock:: operator !=	不等比較運算子。
--------------------	----------

需求

標頭檔 <msclr\lock.h>

命名空間 msclr

lock:: lock

`lock` 會建立物件，並選擇性地在指定的一段時間內(或完全不會)等待取得鎖定。

```
template<class T> lock(
    T ^ _object
);
template<class T> lock(
    T ^ _object,
    int _timeout
);
template<class T> lock(
    T ^ _object,
    System::TimeSpan _timeout
);
template<class T> lock(
    T ^ _object,
    lock_later
);
```

參數

`_object`

要鎖定的物件。

`_timeout`

超時值(以毫秒為單位)或為 `TimeSpan`。

例外狀況

`ApplicationException`如果在超時之前未發生鎖定取得，則擲回。

備註

第三種形式的函式會嘗試在 `_object` 指定的超時時間內取得鎖定(如果沒有 `Infinite` 指定)，則為。

第四種形式的函式不會取得鎖定 `_object`。`lock_later` 這是 `lock_when` 列舉的成員。在此情況下，請使用 `lock::取得` 或 `lock:: try_acquire` 取得鎖定。

呼叫函式時，會自動釋放鎖定。

`_object` 不可為 `ReaderWriterLock`。如果是，就會產生編譯器錯誤。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。然後，主應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```
// msl_lock_lock.cpp
// compile with: /clr
#include <msclr\lock.h>
```

```

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```
In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.
```

lock:: ~ lock

Destructs `lock` 物件。

```
~lock();
```

備註

此呼叫函式會呼叫 `lock:: release`。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。然後，主應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```
// msclr_lock_dtor.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
    }
}
```

```

        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }

        ThreadCount--;
    }
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

lock::取得

取得物件的鎖定，選擇性地等候一段指定的時間，或完全不取得鎖定。

```

void acquire();
void acquire(
    int _timeout
);
void acquire(
    System::TimeSpan _timeout
);

```

參數

_timeout

Timeout 值(以毫秒為單位)或 [TimeSpan](#)。

例外狀況

[ApplicationException](#)如果在超時之前未發生鎖定取得，則擲回。

備註

如果未提供超時值，預設的超時值為 [Infinite](#)。

如果已經取得鎖定，此函式不會執行任何動作。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。然後，主應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```
// ms1_lock_acquire.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                               Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }

        ThreadCount--;
    }
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }
}
```

```

}

// keep our main thread alive until all worker threads have completed
lock l(cc, lock_later); // don't lock now, just create the object
while (true) {
    if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
        if (0 == cc->ThreadCount) {
            Console::WriteLine("All threads completed.");
            break; // all threads are gone, exit while
        }
        else {
            Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
            l.release(); // some threads exist, let them do their work
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

lock::is_locked

指出是否持有鎖定。

```
bool is_locked();
```

傳回值

`true` 如果保留鎖定，則為，`false` 否則為。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在，並等候結束，直到所有背景工作執行緒都完成其工作為止。

```

// msl_lock_is_locked.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
}
```

```

void UseCounter() {
    try {
        lock l(this); // wait infinitely

        Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                           Counter);

        for (int i = 0; i < 10; i++) {
            Counter++;
            Thread::Sleep(10);
        }

        Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                           Counter);

        Counter = 0;
        // lock is automatically released when it goes out of scope and its destructor is called
    }
    catch (...) {
        Console::WriteLine("Couldn't acquire lock!");
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        l.try_acquire(50); // try to acquire lock, don't throw an exception if can't
        if (l.is_locked()) { // check if we got the lock
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

lock::operator bool

lock 在條件運算式中使用的運算子。

```
operator bool();
```

傳回值

true 如果保留鎖定，則為，false 否則為。

備註

這個運算子實際上會轉換成 `_detail_class::_safe_bool` 更安全的，`bool` 因為它無法轉換成整數類資料類型。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。主要應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```
// ms1_lock_op_bool.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
```

```

array<Thread^>^ tarr = gcnew array<Thread^>(5);
ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
for (int i = 0; i < tarr->Length; i++) {
    tarr[i] = gcnew Thread(startDelegate);
    cc->ThreadCount++;
    tarr[i]->Start();
}

// keep our main thread alive until all worker threads have completed
lock l(cc, lock_later); // don't lock now, just create the object
while (true) {
    l.try_acquire(50); // try to acquire lock, don't throw an exception if can't
    if (l) { // use bool operator to check for lock
        if (0 == cc->ThreadCount) {
            Console::WriteLine("All threads completed.");
            break; // all threads are gone, exit while
        }
        else {
            Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
            l.release(); // some threads exist, let them do their work
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

lock:: release

釋放鎖定。

```
void release();
```

備註

如果未持有鎖定，就 `release` 不會執行任何動作。

您不需要明確呼叫此函數。當 `lock` 物件超出範圍時，它的函式會 `release` 呼叫。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。然後，主應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```

// msl_lock_release.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

```

```

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {
            Console::WriteLine("Couldn't acquire lock!");
        }
    }

    ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```
In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.
```

lock::try_acquire

取得物件的鎖定，等候指定的時間量，並傳回 `bool` 來回報取得成功，而不是擲回例外狀況。

```
bool try_acquire(
    int _timeout_ms
);
bool try_acquire(
    System::TimeSpan _timeout
);
```

參數

`_timeout`

Timeout 值(以毫秒為單位)或 [TimeSpan](#)。

傳回值

`true` 如果已取得鎖定，則 `false` 為，否則為。

備註

如果已經取得鎖定，此函式不會執行任何動作。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保對每個執行緒而言，其內部資料的存取都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在。然後，主應用程式會等到所有工作者執行緒都完成其工作之後，才會結束。

```
// msl_lock_try_acquire.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely
            Console::WriteLine("In thread {0} Counter = {1}", Thread::CurrentThread->ManagedThreadId, Counter);
        }
    }
}
```

```

        Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                           Counter);

    for (int i = 0; i < 10; i++) {
        Counter++;
        Thread::Sleep(10);
    }

    Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                           Counter);

    Counter = 0;
    // lock is automatically released when it goes out of scope and its destructor is called
}
catch (...) {
    Console::WriteLine("Couldn't acquire lock!");
}

ThreadCount--;
}
};

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

lock:: operator ==

等號比較運算子。

```
template<class T> bool operator==(  
    T t  
)
```

參數

t

要比較是否相等的物件。

傳回值

`true` 如果與 `t` 鎖定的物件相同，則傳回，`false` 否則傳回。

範例

```
// msl_lock_op_eq.cpp  
// compile with: /clr  
#include <msclr/lock.h>  
  
using namespace System;  
using namespace System::Threading;  
using namespace msclr;  
  
int main () {  
    Object^ o1 = gcnew Object;  
    lock l1(o1);  
    if (l1 == o1) {  
        Console::WriteLine("Equal!");  
    }  
}
```

Equal!

lock::operator !=

不等比較運算子。

```
template<class T> bool operator!=(  
    T t  
)
```

參數

t

要比較是否不相等的物件。

傳回值

`true` 如果 `t` 與鎖定的物件不同，則傳回，`false` 否則傳回。

範例

```
// msclr_lock_op_ineq.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

int main () {
    Object^ o1 = gcnew Object;
    Object^ o2 = gcnew Object;
    lock l1(o1);
    if (l1 != o2) {
        Console::WriteLine("Inequal!");
    }
}
```

```
Inequal!
```

lock_when 列舉

2020/11/2 • [Edit Online](#)

指定延遲的鎖定。

語法

```
enum lock_when {
    lock_later
};
```

備註

傳遞至 `lock::lock` 時，`lock_later` 指定目前不會執行鎖定。

範例

此範例會在多個執行緒中使用類別的單一實例。類別會對本身使用鎖定，以確保其內部資料的存取對每個執行緒都是一致的。主應用程式執行緒會在類別的相同實例上使用鎖定，以定期檢查是否有任何背景工作執行緒仍然存在，並等候結束，直到所有背景工作執行緒都完成其工作為止。

```
// msl_lock_lock_when.cpp
// compile with: /clr
#include <msclr/lock.h>

using namespace System;
using namespace System::Threading;
using namespace msclr;

ref class CounterClass {
private:
    int Counter;

public:
    property int ThreadCount;

    // function called by multiple threads, use lock to keep Counter consistent
    // for each thread
    void UseCounter() {
        try {
            lock l(this); // wait infinitely

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            for (int i = 0; i < 10; i++) {
                Counter++;
                Thread::Sleep(10);
            }

            Console::WriteLine("In thread {0}, Counter = {1}", Thread::CurrentThread->ManagedThreadId,
                Counter);

            Counter = 0;
            // lock is automatically released when it goes out of scope and its destructor is called
        }
        catch (...) {

```

```

        ...
        Console::WriteLine("Couldn't acquire lock!");
    }

    ThreadCount--;
}

int main() {
    // create a few threads to contend for access to the shared data
    CounterClass^ cc = gcnew CounterClass;
    array<Thread^>^ tarr = gcnew array<Thread^>(5);
    ThreadStart^ startDelegate = gcnew ThreadStart(cc, &CounterClass::UseCounter);
    for (int i = 0; i < tarr->Length; i++) {
        tarr[i] = gcnew Thread(startDelegate);
        cc->ThreadCount++;
        tarr[i]->Start();
    }

    // keep our main thread alive until all worker threads have completed
    lock l(cc, lock_later); // don't lock now, just create the object
    while (true) {
        if (l.try_acquire(50)) { // try to acquire lock, don't throw an exception if can't
            if (0 == cc->ThreadCount) {
                Console::WriteLine("All threads completed.");
                break; // all threads are gone, exit while
            }
            else {
                Console::WriteLine("{0} threads exist, continue waiting...", cc->ThreadCount);
                l.release(); // some threads exist, let them do their work
            }
        }
    }
}
}

```

```

In thread 3, Counter = 0
In thread 3, Counter = 10
In thread 5, Counter = 0
In thread 5, Counter = 10
In thread 7, Counter = 0
In thread 7, Counter = 10
In thread 4, Counter = 0
In thread 4, Counter = 10
In thread 6, Counter = 0
In thread 6, Counter = 10
All threads completed.

```

需求

標頭檔 <msclr\lock.h>

命名空間 msclr

另請參閱

鎖

呼叫特定應用程式定義域的函式

2019/12/2 • [Edit Online](#)

特定的應用程式定義域中呼叫函式的支援。

本節內容

[call_in_appdomain 函式](#)

另請參閱

[C++ 支援程式庫](#)

call_in_appdomain 函式

2020/11/2 • [Edit Online](#)

在指定的應用程式域中執行函數。

語法

```
template <typename ArgType1, ...typename ArgTypeN>
void call_in_appdomain(
    DWORD appdomainId,
    void (*voidFunc)(ArgType1, ...ArgTypeN) [ ,
        ArgType1 arg1,
        ...
        ArgTypeN argN ]
);
template <typename RetType, typename ArgType1, ...typename ArgTypeN>
RetType call_in_appdomain(
    DWORD appdomainId,
    RetType (*nonvoidFunc)(ArgType1, ...ArgTypeN) [ ,
        ArgType1 arg1,
        ...
        ArgTypeN argN ]
);
```

參數

appdomainId

要在其中呼叫函數的 appdomain。

voidFunc

`void` 接受 N 個參數($0 \leq N \leq 15$)之函式的指標。

nonvoidFunc

非函式的指標 `void`，接受 N 個參數($0 \leq N \leq 15$)。

arg1 .. Argn

要傳遞至 `voidFunc` 或 `nonvoidFunc` 其他 appdomain 中的零到15個參數。

傳回值

`voidFunc` `nonvoidFunc` 在指定的應用程式域中執行或的結果。

備註

傳遞至之函式的引數 `call_in_appdomain` 不得為 CLR 類型。

範例

```

// msl_call_in_appdomain.cpp
// compile with: /clr

// Defines two functions: one takes a parameter and returns nothing,
// the other takes no parameters and returns an int. Calls both
// functions in the default appdomain and in a newly-created
// application domain using call_in_appdomain.

#include <msclr\appdomain.h>

using namespace System;
using namespace msclr;

void PrintCurrentDomainName( char* format )
{
    String^ s = gcnew String(format);
    Console::WriteLine( s, AppDomain::CurrentDomain->FriendlyName );
}

int GetDomainId()
{
    return AppDomain::CurrentDomain->Id;
}

int main()
{
    AppDomain^ appDomain1 = AppDomain::CreateDomain( "First Domain" );

    call_in_appdomain( AppDomain::CurrentDomain->Id,
                       &PrintCurrentDomainName,
                       (char*)"default appdomain: {0}" );
    call_in_appdomain( appDomain1->Id,
                       &PrintCurrentDomainName,
                       (char*)"in appDomain1: {0}" );

    int id;
    id = call_in_appdomain( AppDomain::CurrentDomain->Id, &GetDomainId );
    Console::WriteLine( "default appdomain id = {0}", id );
    id = call_in_appdomain( appDomain1->Id, &GetDomainId );
    Console::WriteLine( "appDomain1 id = {0}", id );
}

```

輸出

```

default appdomain: msl_call_in_appdomain.exe
in appDomain1: First Domain
default appdomain id = 1
appDomain1 id = 2

```

需求

標頭檔 <msclr\appdomain.h>

命名空間msclr

com::ptr

2020/3/18 • • [Edit Online](#)

可當做 CLR 類別成員使用之 COM 物件的包裝函式。包裝函式也會自動執行 COM 物件的存留期管理，並在呼叫其析構器時釋放物件上擁有的參考。類似于[CComPtr 類別](#)。

語法

```
#include <msclr\com\ptr.h>
```

備註

[com::p Tr 類別](#)是在 <msclr\com\ptr.h> 檔中定義。

另請參閱

[C++ 支援程式庫](#)

com::ptr 類別

2020/11/2 • [Edit Online](#)

可當做 CLR 類別成員使用之 COM 物件的包裝函式。包裝函式也會將 COM 物件的存留期管理自動化，在呼叫其解構函式時，釋出物件上所有已擁有的參考。類似于[CComPtr 類別](#)。

語法

```
template<class _interface_type>
ref class ptr;
```

參數

_interface_type

COM 介面。

備註

`com::ptr` 也可以當做本機函式變數使用，以簡化各種 COM 工作並且將存留期管理自動化。

無法直接當做函式 `com::ptr` 參數使用，請改用追蹤參考運算子或物件運算子的控制碼(^)。

`com::ptr` 無法直接從函式傳回；請改用控制碼。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。呼叫類別的公用方法會造成對包含的 `IXMLDOMDocument` 物件進行呼叫。這個範例會建立 XML 文件的執行個體，填滿一些簡單的 XML，然後在剖析的文件樹狀進行節點的簡化查核行程，以將 XML 列印到主控台。

```
// comptr.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    void LoadXml(String^ xml) {
        pin_ptr<const wchar_t> pinnedXml = PtrToStringChars(xml);
        BSTR bstr = NULL;

        try {

```

```

// load some XML into the document
bstr = ::SysAllocString(pinnedXml);
if (NULL == bstr) {
    throw gcnew OutOfMemoryException;
}
VARIANT_BOOL bIsSuccessful = false;
// use operator -> to call IXMDOMDocument member function
Marshal::ThrowExceptionForHR(m_ptrDoc->loadXML(bstr, &bIsSuccessful));
}

finally {
    ::SysFreeString(bstr);
}

}

// simplified function to write just the first xml node to the console
void WriteXml() {
    IXMLDOMNode* pNode = NULL;

    try {
        // the first child of the document is the first real xml node
        Marshal::ThrowExceptionForHR(m_ptrDoc->get_firstChild(&pNode));
        if (NULL != pNode) {
            WriteNode(pNode);
        }
    }
    finally {
        if (NULL != pNode) {
            pNode->Release();
        }
    }
}

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    // simplified function that only writes the node
    void WriteNode(IXMLDOMNode* pNode) {
        BSTR bstr = NULL;

        try {
            // write out the name and text properties
            Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
            String^ strName = gcnew String(bstr);
            Console::Write("<{0}>", strName);
            ::SysFreeString(bstr);
            bstr = NULL;

            Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
            Console::Write(gcnew String(bstr));
            ::SysFreeString(bstr);
            bstr = NULL;

            Console::WriteLine("</{0}>", strName);
        }
        finally {
            ::SysFreeString(bstr);
        }
    }

    com::ptr<IXMDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

```

```

    // stream some xml into the document
    doc.LoadXml("<word>persnickety</word>");

    // write the document to the console
    doc.WriteXml();
}
catch (Exception^ e) {
    Console::WriteLine(e);
}
}

```

```
<word>persnickety</word>
```

成員

公用建構函式

ptr::ptr	結構 <code>com::ptr</code> 來包裝 COM 物件。
ptr::~ptr	Destructs <code>com::ptr</code> 。

公用方法

ptr::Attach	將 COM 物件附加至 <code>com::ptr</code> 。
ptr::CreateInstance	在內建立 COM 物件的實例 <code>com::ptr</code> 。
ptr::Detach	提供 COM 物件的擁有權，並傳回物件的指標。
ptr::GetInterface	在內建立 COM 物件的實例 <code>com::ptr</code> 。
ptr::QueryInterface	查詢介面擁有的 COM 物件，並將結果附加至另一個 <code>com::ptr</code> 。
ptr::Release	釋放 COM 物件上所有擁有的參考。

公用運算子

ptr::operator->	成員存取運算子，用來呼叫所擁有 COM 物件上的方法。
ptr::operator =	將 COM 物件附加至 <code>com::ptr</code> 。
ptr::operator bool	<code>com::ptr</code> 在條件運算式中使用的運算子。
ptr::operator!	用來判斷擁有的 COM 物件是否不正確運算子。

需求

標頭檔 <msclr\com\ptr.h>

命名空間 msclr:: com

ptr::p tr

傳回所擁有 COM 物件的指標。

```
ptr();
ptr(
    _interface_type * p
);
```

參數

P

COM 的介面指標。

備註

無引數的函式會指派 `nullptr` 給基礎物件控制碼。未來對的呼叫 `com::ptr` 將會驗證內建物件，並以無訊息模式失敗，直到建立或附加物件為止。

單一引數的函式會加入 COM 物件的參考，但不會釋放呼叫端的參考，因此呼叫端必須 `Release` 在 COM 物件上呼叫，才能真正授與控制權。`com::ptr` 呼叫的函式時，它會自動釋放其對 COM 物件的參考。

傳遞 `NULL` 至這個函式的方式與呼叫無引數版本相同。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。它會示範兩個版本的函式的使用方式。

```

// comprtr_ptr.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // construct the internal com::ptr with a COM object
    XmlDocument(IXMLDOMDocument* pDoc) : m_ptrDoc(pDoc) {}

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create an XML DOM document object
        Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
            CLSCTX_ALL, IID_IXMLDOMDocument, (void**)&pDoc));
        // construct the ref class with the COM object
        XmlDocument doc1(pDoc);

        // or create the class from a progid string
        XmlDocument doc2("Msxml2.DOMDocument.3.0");
    }
    // doc1 and doc2 destructors are called when they go out of scope
    // and the internal com::ptr releases its reference to the COM object
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
}

```

ptr:: ~ptr

Destructs `com::ptr` .

```
~ptr();
```

備註

在銷毀時，`com::ptr` 會將它擁有的所有參考發行至其 COM 物件。假設沒有任何其他參考保存至 COM 物件，則會刪除 COM 物件，並釋放其記憶體。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。在函式中 `main`，當兩個 `Xm1Document` 物件的析構函式超出區塊的範圍時，將會呼叫它，`try` 因此會呼叫基礎的 `com::ptr` 析構函數，釋放 COM 物件所擁有的所有參考。

```

// comprtr_dtor.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // construct the internal com::ptr with a COM object
    XmlDocument(IXMLDOMDocument* pDoc) : m_ptrDoc(pDoc) {}

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create an XML DOM document object
        Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
            CLSCTX_ALL, IID_IXMLDOMDocument, (void**)&pDoc));
        // construct the ref class with the COM object
        XmlDocument doc1(pDoc);

        // or create the class from a progid string
        XmlDocument doc2("Msxml2.DOMDocument.3.0");
    }
    // doc1 and doc2 destructors are called when they go out of scope
    // and the internal com::ptr releases its reference to the COM object
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
}

```

ptr:: Attach

將 COM 物件附加至 `com::ptr`。

```
void Attach(
    _interface_type * _right
);
```

參數

_right

要附加的 COM 介面指標。

例外狀況

如果 `com::ptr` 已經擁有 COM 物件的參考，則會擲 `Attach` 回 `InvalidOperationException`。

備註

對的呼叫 `Attach` 會參考 COM 物件，但不會釋放呼叫者的參考。

傳遞 `NULL` 至會 `Attach` 導致未採取任何動作。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式會 `ReplaceDocument` 先呼叫 `Release` 任何先前擁有的物件，然後呼叫 `Attach` 以附加新的檔物件。

```
// comptr_attach.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // replace currently held COM object with another one
    void ReplaceDocument(IXMLDOMDocument* pDoc) {
        // release current document object
        m_ptrDoc.Release();
        // attach the new document object
        m_ptrDoc.Attach(pDoc);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object
private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that creates a raw XML DOM Document object
IXMLDOMDocument* CreateDocument() {
    IXMLDOMDocument* pDoc = NULL;
    Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
        CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
    return pDoc;
}
```

```
// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by our ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
```

ptr::CreateInstance

在內建立 COM 物件的實例 `com::ptr`。

```
void CreateInstance(
    System::String ^ progid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    System::String ^ progid,
    LPUNKNOWN pouter
);
void CreateInstance(
    System::String ^ progid
);
void CreateInstance(
    const wchar_t * progid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    const wchar_t * progid,
    LPUNKNOWN pouter
);
void CreateInstance(
    const wchar_t * progid
);
void CreateInstance(
    REFCLSID rclsid,
    LPUNKNOWN pouter,
    DWORD cls_context
);
void CreateInstance(
    REFCLSID rclsid,
    LPUNKNOWN pouter
);
void CreateInstance(
    REFCLSID rclsid
);
```

參數

進程

ProgID 字串。

pouter

匯總物件的 IUnknown 介面的指標(控制 IUnknown)。如果 **pouter** 未指定, **NULL** 則會使用。

cls_Context

用來管理新建立之物件的程式碼會在其中執行的內容。值取自 **CLSTX** 列舉。如果 **cls_Context** 未指定, 則會使用 **CLSTX_ALL** 的值。

rclsid

CLSID 與將用來建立物件的資料和程式碼相關聯。

例外狀況

如果 **com::ptr** 已經擁有 COM 物件的參考, 則會擲 **CreateInstance** 回 [InvalidOperationException](#)。

此函式會呼叫 **CoCreateInstance**, 並使用將 **ThrowExceptionForHR** 任何錯誤轉換 **HRESULT** 為適當的例外狀況。

備註

CreateInstance 會使用 **CoCreateInstance** 來建立指定物件的新實例, 識別其從 ProgID 或 CLSID。**com::ptr** 會參考新建立的物件, 而且會在銷毀時自動釋放所有擁有的參考。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。類別的函式會使用兩種不同的形式 `CreateInstance`，從 ProgID 或從 CLSID 加上 CLSCTX 來建立檔物件。

```
// comptr_createinstance.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }
    XmlDocument(REFCLSID clsid, DWORD clsctx) {
        m_ptrDoc.CreateInstance(clsid, NULL, clsctx);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc1("Msxml2.DOMDocument.3.0");

        // or from a clsid with specific CLSCTX
        XmlDocument doc2(CLSID_DOMDocument30, CLSCTX_INPROC_SERVER);
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
```

ptr::Detach

提供 COM 物件的擁有權，並傳回物件的指標。

```
_interface_type * Detach();
```

傳回值

COM 物件的指標。

如果未擁有任何物件，則會傳回 Null。

例外狀況

在內部，`QueryInterface` 會在擁有的 COM 物件上呼叫，而且任何錯誤 `HRESULT` 都會由轉換成例外狀況

ThrowExceptionForHR。

備註

`Detach` 首先，代表呼叫者加入 COM 物件的參考，然後釋放所擁有的所有參考 `com::ptr`。呼叫端最後必須釋放傳回的物件來摧毀它。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式會 `DetachDocument` 呼叫 `Detach` 來提供 COM 物件的擁有權，並將指標傳回給呼叫者。

```
// comptr_detach.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // detach the COM object and return it
    // this releases the internal reference to the object
    IXMLDOMDocument* DetachDocument() {
        return m_ptrDoc.Detach();
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        bstrXml = ::SysAllocString(L"<word>persnickety</word>");
        if (NULL == bstrXml) {

```

```

        throw gcnew OutOfMemoryException("bstrXml");
    }
    // detach the document object from the ref class
    pDoc = doc.DetachDocument();
    // use unmanaged function and raw object to load xml
    Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
    // release document object as the ref class no longer owns it
    pDoc->Release();
    pDoc = NULL;
}
catch (Exception^ e) {
    Console::WriteLine(e);
}
finally {
    if (NULL != pDoc) {
        pDoc->Release();
    }
}
}

```

ptr:: GetInterface

傳回所擁有 COM 物件的指標。

```
_interface_type * GetInterface();
```

傳回值

所擁有 COM 物件的指標。

例外狀況

在內部，`QueryInterface` 會在擁有的 COM 物件上呼叫，而且任何錯誤 `HRESULT` 都會由轉換成例外狀況 `ThrowExceptionForHR`。

備註

會 `com::ptr` 代表呼叫端加入 com 物件的參考，也會在 com 物件上保留自己的參考。呼叫端最後必須釋放傳回之物件上的參考，否則永遠不會終結。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式會 `GetDocument` 使用 `GetInterface` 來傳回 COM 物件的指標。

```

// comprtr_getinterface.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {

```

```

        m_ptrDoc.CreateInstance(progid);
    }

    // add a reference to and return the COM object
    // but keep an internal reference to the object
    IXMLDOMDocument* GetDocument() {
        return m_ptrDoc.GetInterface();
    }

    // simplified function that only writes the first node
    void WriteDocument() {
        IXMLDOMNode* pNode = NULL;
        BSTR bstr = NULL;

        try {
            // use operator -> to call XML Doc member
            Marshal::ThrowExceptionForHR(m_ptrDoc->get.firstChild(&pNode));
            if (NULL != pNode) {
                // write out the xml
                Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
                String^ strName = gcnew String(bstr);
                Console::Write("<{0}>", strName);
                ::SysFreeString(bstr);
                bstr = NULL;

                Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
                Console::Write(gcnew String(bstr));
                ::SysFreeString(bstr);
                bstr = NULL;

                Console::WriteLine("</{0}>", strName);
            }
        }
        finally {
            if (NULL != pNode) {
                pNode->Release();
            }
            ::SysFreeString(bstr);
        }
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        bstrXml = ::SysAllocString(L"<word>persnickety</word>");
    }
}

```

```

    if (NULL == bstrXml) {
        throw gcnew OutOfMemoryException("bstrXml");
    }
    // detach the document object from the ref class
    pDoc = doc.GetDocument();
    // use unmanaged function and raw object to load xml
    Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
    // release reference to document object (but ref class still references it)
    pDoc->Release();
    pDoc = NULL;

    // call another function on the ref class
    doc.WriteDocument();
}
catch (Exception^ e) {
    Console::WriteLine(e);
}
finally {
    if (NULL != pDoc) {
        pDoc->Release();
    }
}

}

```

<word>persnickety</word>

ptr:: QueryInterface

查詢介面擁有的 COM 物件，並將結果附加至另一個 `com::ptr`。

```

template<class _other_type>
void QueryInterface(
    ptr<_other_type> % other
);

```

參數

另一方面

`com::ptr` 將取得介面的。

例外狀況

在內部，`QueryInterface` 會在擁有的 COM 物件上呼叫，而且任何錯誤 `HRESULT` 都會由轉換成例外狀況 `ThrowExceptionForHR`。

備註

您可以使用這個方法，針對目前包裝函式所擁有之 COM 物件的不同介面，建立 COM 包裝函式。這個方法會 `QueryInterface` 透過所擁有的 com 物件來呼叫，以要求 COM 物件的特定介面指標，並將傳回的介面指標附加至傳入的 `com::ptr`。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。此 `WriteTopLevelNode` 成員函式會使用，在 `QueryInterface` 本機填入 `com::ptr IXMLDOMNode`，然後將(藉 `com::ptr` 由追蹤參考)傳遞至將節點的名稱和文字屬性寫入主控台的私用成員函式。

```

// comptr_queryinterface.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

```

```

#include <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    void LoadXml(String^ xml) {
        pin_ptr<const wchar_t> pinnedXml = PtrToStringChars(xml);
        BSTR bstr = NULL;

        try {
            // load some XML into our document
            bstr = ::SysAllocString(pinnedXml);
            if (NULL == bstr) {
                throw gcnew OutOfMemoryException;
            }
            VARIANT_BOOL bIsSuccessful = false;
            // use operator -> to call IXMODOMDocument member function
            Marshal::ThrowExceptionForHR(m_ptrDoc->loadXML(bstr, &bIsSuccessful));
        }
        finally {
            ::SysFreeString(bstr);
        }
    }

    // write the top level node to the console
    void WriteTopLevelNode() {
        com::ptr<IXMLDOMNode> ptrNode;

        // query for the top level node interface
        m_ptrDoc.QueryInterface(ptrNode);
        WriteNode(ptrNode);
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object

private:
    // simplified function that only writes the node
    void WriteNode(com::ptr<IXMLDOMNode> % node) {
        BSTR bstr = NULL;

        try {
            // write out the name and text properties
            Marshal::ThrowExceptionForHR(node->get_nodeName(&bstr));
            String^ strName = gcnew String(bstr);
            Console::Write("<{0}>", strName);
            ::SysFreeString(bstr);
            bstr = NULL;

            Marshal::ThrowExceptionForHR(node->get_text(&bstr));
            Console::Write(gcnew String(bstr));
            ::SysFreeString(bstr);
            bstr = NULL;

            Console::WriteLine("</{0}>", strName);
        }
        finally {

```

```

    finally {
        ::SysFreeString(bstr);
    }
}

com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // stream some xml into the document
        doc.LoadXml("<word>persnickety</word>");

        // write the document to the console
        doc.WriteTopLevelNode();
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
}

```

```
<#document>persnickety</#document>
```

ptr:: Release

釋放 COM 物件上所有擁有的參考。

```
void Release();
```

備註

呼叫這個函式會釋放 COM 物件上所有擁有的參考，並將 COM 物件的內部控制碼設定為 `nullptr`。如果 COM 物件上沒有任何其他參考存在，則會終結它。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式會 `ReplaceDocument` 使用 `Release`，在附加新檔之前釋放任何先前的檔案物件。

```

// comprtr_release.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }
}

```

```

// replace currently held COM object with another one
void ReplaceDocument(IXMLDOMDocument* pDoc) {
    // release current document object
    m_ptrDoc.Release();
    // attach the new document object
    m_ptrDoc.Attach(pDoc);
}

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that creates a raw XML DOM Document object
IXMLDOMDocument* CreateDocument() {
    IXMLDOMDocument* pDoc = NULL;
    Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
        CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
    return pDoc;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by our ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}
}

```

ptr:: operator->

成員存取運算子，用來呼叫所擁有 COM 物件上的方法。

```
_detail::smart_com_ptr<_interface_type> operator->();
```

傳回值

`smart_com_ptr` COM 物件的。

例外狀況

在內部，`QueryInterface` 會在擁有的 COM 物件上呼叫，而且任何錯誤 `HRESULT` 都會由轉換成例外狀況 `ThrowExceptionForHR`。

備註

這個運算子可讓您呼叫所擁有 COM 物件的方法。它會傳回一個暫存 `smart_com_ptr`，它會自動處理自己的 `AddRef` 和 `Release`。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。函式會 `WriteDocument` 使用 `operator->` 來呼叫 `get.firstChild` `document` 物件的成員。

```
// comprtr_op_member.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    // construct the internal com::ptr with a null interface
    // and use CreateInstance to fill it
    XmlDocument(String^ progid) {
        m_ptrDoc.CreateInstance(progid);
    }

    // add a reference to and return the COM object
    // but keep an internal reference to the object
    IXMLDOMDocument* GetDocument() {
        return m_ptrDoc.GetInterface();
    }

    // simplified function that only writes the first node
    void WriteDocument() {
        IXMLElementNode* pNode = NULL;
        BSTR bstr = NULL;

        try {
            // use operator -> to call XML Doc member
            Marshal::ThrowExceptionForHR(m_ptrDoc->get.firstChild(&pNode));
            if (NULL != pNode) {
                // write out the xml
                Marshal::ThrowExceptionForHR(pNode->get_nodeName(&bstr));
                String^ strName = gcnew String(bstr);
                Console::Write("<{0}>", strName);
                ::SysFreeString(bstr);
                bstr = NULL;

                Marshal::ThrowExceptionForHR(pNode->get_text(&bstr));
                Console::Write(gcnew String(bstr));
                ::SysFreeString(bstr);
                bstr = NULL;

                Console::WriteLine("</{0}>", strName);
            }
        }
        finally {
            if (NULL != pNode) {
                pNode->Release();
            }
            ::SysFreeString(bstr);
        }
    }
}
```

```

// note that the destructor will call the com::ptr destructor
// and automatically release the reference to the COM object

private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// unmanaged function that loads XML into a raw XML DOM Document object
HRESULT LoadXml(IXMLDOMDocument* pDoc, BSTR bstrXml) {
    HRESULT hr = S_OK;
    VARIANT_BOOL bSuccess;
    hr = pDoc->loadXML(bstrXml, &bSuccess);
    if (S_OK == hr && !bSuccess) {
        hr = E_FAIL;
    }
    return hr;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;
    BSTR bstrXml = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        bstrXml = ::SysAllocString(L"<word>persnickety</word>");
        if (NULL == bstrXml) {
            throw gcnew OutOfMemoryException("bstrXml");
        }
        // detach the document object from the ref class
        pDoc = doc.GetDocument();
        // use unmanaged function and raw object to load xml
        Marshal::ThrowExceptionForHR(LoadXml(pDoc, bstrXml));
        // release reference to document object (but ref class still references it)
        pDoc->Release();
        pDoc = NULL;

        // call another function on the ref class
        doc.WriteDocument();
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}

```

```
<word>persnickety</word>
```

ptr:: operator =

將 COM 物件附加至 `com::ptr` 。

```
ptr<_interface_type> % operator=(  
    _interface_type * _right  
)
```

參數

_right

要附加的 COM 介面指標。

傳回值

上的追蹤參考 `com::ptr`。

例外狀況

如果 `com::ptr` 已經擁有 COM 物件的參考，則會擲 `operator=` 回 `InvalidOperationException`。

備註

將 COM 物件指派給會 `com::ptr` 參考 `com` 物件，但不會釋放呼叫者的參考。

這個運算子與具有相同的效果 `Attach`。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式會 `ReplaceDocument` 先 `Release` 在任何先前擁有的物件上呼叫，然後使用 `operator=` 來附加新的檔案。

```
// comptr_op_assign.cpp  
// compile with: /clr /link msxml2.lib  
#include <msxml2.h>  
#include <msclr\com\ptr.h>  
  
#import <msxml3.dll> raw_interfaces_only  
  
using namespace System;  
using namespace System::Runtime::InteropServices;  
using namespace msclr;  
  
// a ref class that uses a com::ptr to contain an  
// IXMLDOMDocument object  
ref class XmlDocument {  
public:  
    // construct the internal com::ptr with a null interface  
    // and use CreateInstance to fill it  
    XmlDocument(String^ progid) {  
        m_ptrDoc.CreateInstance(progid);  
    }  
  
    // replace currently held COM object with another one  
    void ReplaceDocument(IXMLDOMDocument* pDoc) {  
        // release current document object  
        m_ptrDoc.Release();  
        // attach the new document object  
        m_ptrDoc = pDoc;  
    }  
  
    // note that the destructor will call the com::ptr destructor  
    // and automatically release the reference to the COM object  
  
private:  
    com::ptr<IXMLDOMDocument> m_ptrDoc;  
};  
  
// unmanaged function that creates a raw XML DOM Document object  
IXMLDOMDocument* CreateDocument() {  
    IXMLDOMDocument* pDoc = NULL;
```

```

Marshal::ThrowExceptionForHR(CoCreateInstance(CLSID_DOMDocument30, NULL,
    CLSCTX_INPROC_SERVER, IID_IXMLDOMDocument, (void**)&pDoc));
return pDoc;
}

// use the ref class to handle an XML DOM Document object
int main() {
    IXMLDOMDocument* pDoc = NULL;

    try {
        // create the class from a progid string
        XmlDocument doc("Msxml2.DOMDocument.3.0");

        // get another document object from unmanaged function and
        // store it in place of the one held by the ref class
        pDoc = CreateDocument();
        doc.ReplaceDocument(pDoc);
        // no further need for raw object reference
        pDoc->Release();
        pDoc = NULL;
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
    finally {
        if (NULL != pDoc) {
            pDoc->Release();
        }
    }
}

```

ptr:: operator bool

`com::ptr` 在條件運算式中使用的運算子。

```
operator bool();
```

傳回值

`true` 如果擁有的 COM 物件有效，則為，`false` 否則為。

備註

擁有的 COM 物件無效(如果不是的話) `nullptr`。

這個運算子會將轉換成 `_detail_class::_safe_bool` 較安全的，`bool` 因為它無法轉換成整數類資料類型。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式 `CreateInstance` `operator bool` 會在建立新的檔物件之後使用，以判斷它是否有效，如果是，則會寫入主控台。

```

// comprtr_op_bool.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    void CreateInstance(String^ progid) {
        if (!m_ptrDoc) {
            m_ptrDoc.CreateInstance(progid);
            if (m_ptrDoc) { // uses operator bool
                Console::WriteLine("DOM Document created.");
            }
        }
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object
private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        XmlDocument doc;
        // create the instance from a progid string
        doc.CreateInstance("Msxml2.DOMDocument.3.0");
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}

```

DOM Document created.

ptr:: operator !

用來判斷擁有的 COM 物件是否不正確運算子。

```
bool operator!();
```

傳回值

`true` 如果擁有的 COM 物件無效，則為，`false` 否則為。

備註

擁有的 COM 物件無效(如果不是的話) `nullptr`。

範例

這個範例實作 CLR 類別，此類別使用 `com::ptr` 來包裝其私用成員 `IXMLDOMDocument` 物件。成員函式

CreateInstance operator! 會使用來判斷檔案物件是否已擁有，而且只有在物件無效時，才會建立新的實例。

```
// comptr_op_not.cpp
// compile with: /clr /link msxml2.lib
#include <msxml2.h>
#include <msclr\com\ptr.h>

#import <msxml3.dll> raw_interfaces_only

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace msclr;

// a ref class that uses a com::ptr to contain an
// IXMLDOMDocument object
ref class XmlDocument {
public:
    void CreateInstance(String^ progid) {
        if (!m_ptrDoc) {
            m_ptrDoc.CreateInstance(progid);
            if (m_ptrDoc) {
                Console::WriteLine("DOM Document created.");
            }
        }
    }

    // note that the destructor will call the com::ptr destructor
    // and automatically release the reference to the COM object
private:
    com::ptr<IXMLDOMDocument> m_ptrDoc;
};

// use the ref class to handle an XML DOM Document object
int main() {
    try {
        XmlDocument doc;
        // create the instance from a progid string
        doc.CreateInstance("Msxml2.DOMDocument.3.0");
    }
    catch (Exception^ e) {
        Console::WriteLine(e);
    }
}
```

DOM Document created.

C++/CLI 中的例外狀況

2020/11/2 • [Edit Online](#)

本檔的這一節中的文章說明 c + +/CLI 中的例外狀況處理，以及它與標準例外狀況處理有何不同。

相關文章

II	II
使用 Managed 例外狀況的基本概念	討論在 Managed 應用程式中的例外狀況處理。
/CLR 底下的例外狀況處理行為差異	討論標準例外狀況處理和 c + +/CLI 中的例外狀況處理之間的差異
最後	描述在 <code>finally</code> 發生例外狀況之後，用來清除剩餘資源的區塊。
如何：攔截從 MSIL 擲回之機器碼中的例外狀況	示範如何使用 <code>__try</code> 和 <code>__except</code> 來攔截從 MSIL 擲回之機器碼中的例外狀況。
如何：定義和安裝全域例外狀況處理常式	示範如何擷取未處理的例外狀況。
使用 c + +/CLI 進行 .NET 程式設計 (Visual C++)	在 Visual C++ 文件中關於 .NET 程式設計的最上層文件。

使用 Managed 例外狀況中的基本概念

2020/11/2 • [Edit Online](#)

本主題討論 managed 應用程式中的例外狀況處理。也就是使用 /clr 編譯器選項編譯的應用程式。

本主題內容

- [在 /clr 下擲回例外狀況](#)
- [CLR 延伸模組的 Try/Catch 區塊](#)

備註

如果您使用 /clr 選項進行編譯，則可以處理 clr 例外狀況，而標準 [Exception](#) 類別則提供許多有用的方法來處理 clr 例外狀況，而且建議做為使用者定義例外狀況類別的基類。

/CLR 不支援攔截衍生自介面的例外狀況類型。此外，common language runtime 不允許您捕捉堆疊溢位例外狀況；堆疊溢位例外狀況會終止進程。

如需受控和未受管理的應用程式中例外狀況處理差異的詳細資訊，請參閱[Managed Extensions for C++ 之下例外狀況處理行為的差異](#)。

在 /clr 下擲回例外狀況

C++ throw 運算式已擴充，以擲回 CLR 類型的控制碼。下列範例會建立自訂例外狀況類型，然後擲回該類型的實例：

```
// clr_exception_handling.cpp
// compile with: /clr /c
ref struct MyStruct: public System::Exception {
public:
    int i;
};

void GlobalFunction() {
    MyStruct^ pMyStruct = gcnew MyStruct;
    throw pMyStruct;
}
```

實值型別必須先進行裝箱，才會擲回：

```
// clr_exception_handling_2.cpp
// compile with: /clr /c
value struct MyValueStruct {
    int i;
};

void GlobalFunction() {
    MyValueStruct v = {11};
    throw (MyValueStruct ^)v;
}
```

CLR 延伸模組的 Try/Catch 區塊

相同的 `try` / `catch` 區塊結構可以用來捕捉 CLR 和原生例外狀況：

```
// clr_exception_handling_3.cpp
// compile with: /clr
using namespace System;
ref struct MyStruct : public Exception {
public:
    int i;
};

struct CMyClass {
public:
    double d;
};

void GlobalFunction() {
    MyStruct^ pMyStruct = gcnew MyStruct;
    pMyStruct->i = 11;
    throw pMyStruct;
}

void GlobalFunction2() {
    CMyClass c = {2.0};
    throw c;
}

int main() {
    for ( int i = 1; i >= 0; --i ) {
        try {
            if ( i == 1 )
                GlobalFunction2();
            if ( i == 0 )
                GlobalFunction();
        }
        catch ( CMyClass& catchC ) {
            Console::WriteLine( "In 'catch(CMyClass& catchC)' " );
            Console::WriteLine( catchC.d );
        }
        catch ( MyStruct^ catchException ) {
            Console::WriteLine( "In 'catch(MyStruct^ catchException)' " );
            Console::WriteLine( catchException->i );
        }
    }
}
```

輸出

```
In 'catch(CMyClass& catchC)'
2
In 'catch(MyStruct^ catchException)'
11
```

C++ 物件的回溯順序

具有析構函式之任何 C++ 物件的回溯會發生在擲回函數和處理函數之間的執行時間堆疊上。因為 CLR 類型是在堆積上配置，所以回溯不適用。

所擲回例外狀況的事件順序如下所示：

1. 執行時間會引導堆疊尋找適當的 `catch` 子句，或在 SEH 的情況下，使用 `seh 的 except 篩選準則` 來攔截例外狀況。Catch 子句會先以詞法順序搜尋，然後再動態地向下呼叫堆疊。
2. 一旦找到正確的處理常式，就會將堆疊展開到該點。對於堆疊上的每個函式呼叫，其本機物件都是解構的，而且會從大部分的對外向外執行 `__finally` 區塊。

3. 展開堆疊之後，就會執行 catch 子句。

攔截非受控類型

當擲回非受控物件類型時，它會包裝為類型的例外狀況 [SEHException](#)。搜尋適當的子句時 `catch`，有兩種可能性。

- 如果遇到原生 C++ 型別，則例外狀況會解除包裝並與所遇到的類型相比較。這項比較可讓原生 C++ 類型以正常方式攔截。
- 不過，如果 `catch` 先檢查 [SEHException](#) 類型的子句或其任何基類，則子句會攔截例外狀況。因此，您應該在 CLR 類型的任何 `catch` 子句之前，先放置攔截原生 C++ 類型的所有 `catch` 子句。

請注意

```
catch(Object^)
```

和

```
catch(...)
```

會攔截任何擲回的型別，包括 SEH 例外狀況。

如果 `catch (Object ^)` 攜截到非受控型別，它就不會損毀所擲回的物件。

當擲回或攔截未受管理的例外狀況時，建議您使用[/ehsc](#)編譯器選項，而不要使用 [/ehs](#) 或 [/eha](#)。

另請參閱

[例外狀況處理](#)

[safe_cast](#)

[例外狀況處理](#)

在 /CLR 之下例外狀況處理行為的差異

2020/4/15 • [Edit Online](#)

使用託管異常的基本概念討論託管應用程式中的異常處理。在本主題中會詳細討論例外狀況處理之標準行為差異和一些限制。有關詳細資訊,請參閱[_set_se_translator 函數](#)。

跳出最後一塊

在本機 C/C++ 代碼中,允許使用結構化異常處理 (SEH) 從 `__finally` 最終阻止跳出,儘管它會產生警告。在`/clr`下,跳出最後一個塊會導致錯誤:

```
// clr_exception_handling_4.cpp
// compile with: /clr
int main() {
    try {}
    finally {
        return 0;    // also fails with goto, break, continue
    }
} // C3276
```

在例外篩選器中引發異常

在處理託管代碼中的異常篩選器期間引發異常時,將捕獲該異常並將其視為篩選器返回 0。

這與引發嵌套異常的本機代碼中的行為相反,EXCEPTION_RECORD結構中的****異常記錄欄位(Getexception資訊返回)設置,異常標誌欄位設置 0x10 位元。以下範例說明行為中的差異:

```

// clr_exception_handling_5.cpp
#include <windows.h>
#include <stdio.h>
#include <assert.h>

#ifndef false
#define false 0
#endif

int *p;

int filter(PEXCEPTION_POINTERS ExceptionPointers) {
    PEXCEPTION_RECORD ExceptionRecord =
        ExceptionPointers->ExceptionRecord;

    if ((ExceptionRecord->ExceptionFlags & 0x10) == 0) {
        // not a nested exception, throw one
        *p = 0; // throw another AV
    }
    else {
        printf("Caught a nested exception\n");
        return 1;
    }

    assert(false);

    return 0;
}

void f(void) {
    __try {
        *p = 0; // throw an AV
    }
    __except(filter(GetExceptionInformation())) {
        printf_s("We should execute this handler if "
                "compiled to native\n");
    }
}

int main() {
    __try {
        f();
    }
    __except(1) {
        printf_s("The handler in main caught the "
                "exception\n");
    }
}

```

輸出

```

Caught a nested exception
We should execute this handler if compiled to native

```

取消關聯重新引發

/clr不支援在 catch 處理程式之外重新引發異常(稱為取消關聯的重新引發)。這個類型的例外狀況會視為標準 C++ 重新擲回。若在發生作用中 Managed 例外狀況時遇到取消關聯重新擲回，例外狀況會包裝為 C++ 例外狀況，然後重新擲回。此類型的異常只能作為類型的SEHException異常捕獲。

下列範例示範 Managed 例外狀況重新擲回為 C++. 例外狀況：

```

// clr_exception_handling_6.cpp
// compile with: /clr
using namespace System;
#include <assert.h>
#include <stdio.h>

void rethrow( void ) {
    // This rethrow is a dissociated rethrow.
    // The exception would be masked as SEHException.
    throw;
}

int main() {
    try {
        try {
            throw gcnew ApplicationException();
        }
        catch ( ApplicationException^ ) {
            rethrow();
            // If the call to rethrow() is replaced with
            // a throw statement within the catch handler,
            // the rethrow would be a managed rethrow and
            // the exception type would remain
            // System::ApplicationException
        }
    }
    catch ( ApplicationException^ ) {
        assert( false );

        // This will not be executed since the exception
        // will be masked as SEHException.
    }
    catch ( Runtime::InteropServices::SEHException^ ) {
        printf_s("caught an SEH Exception\n" );
    }
}

```

輸出

```
caught an SEH Exception
```

例外篩選器和EXCEPTION_CONTINUE_EXECUTION

如果篩選條件在 Managed 應用程式中傳回 `EXCEPTION_CONTINUE_EXECUTION`，則會視為篩選條件已傳回 `EXCEPTION_CONTINUE_SEARCH`。有關這些常量的詳細資訊，請參閱 [嘗試除語句](#)。

下面的範例展示此差異：

```

// clr_exception_handling_7.cpp
#include <windows.h>
#include <stdio.h>
#include <assert.h>

int main() {
    int Counter = 0;
    __try {
        __try {
            Counter -= 1;
            RaiseException (0xe0000000|'seh',
                            0, 0, 0);
            Counter -= 2;
        }
        __except (Counter) {
            // Counter is negative,
            // indicating "CONTINUE EXECUTE"
            Counter -= 1;
        }
    }
    __except(1) {
        Counter -= 100;
    }

    printf_s("Counter=%d\n", Counter);
}

```

輸出

```
Counter=-3
```

_set_se_translator功能

由呼叫 `_set_se_translator` 設定的翻譯工具函式，只影響 Unmanaged 程式碼中的攔截。下列範例示範此限制：

```

// clr_exception_handling_8.cpp
// compile with: /clr /EHa
#include <iostream>
#include <windows.h>
#include <eh.h>
#pragma warning (disable: 4101)
using namespace std;
using namespace System;

#define MYEXCEPTION_CODE 0xe0000101

class CMyException {
public:
    unsigned int m_ErrorCode;
    EXCEPTION_POINTERS * m_pExp;

    CMyException() : m_ErrorCode( 0 ), m_pExp( NULL ) {}

    CMyException( unsigned int i, EXCEPTION_POINTERS * pExp )
        : m_ErrorCode( i ), m_pExp( pExp ) {}

    CMyException( CMyException& c ) : m_ErrorCode( c.m_ErrorCode ),
                                    m_pExp( c.m_pExp ) {}

    friend ostream& operator <<
        ( ostream& out, const CMyException& inst ) {
        return out << "CMyException[\n" <<
                    "Error Code: " << inst.m_ErrorCode << "]";
    }
}

```

```

}

#pragma unmanaged
void my_trans_func( unsigned int u, PEXCEPTION_POINTERS pExp ) {
    cout << "In my_trans_func.\n";
    throw CMyException( u, pExp );
}

#pragma managed
void managed_func() {
    try {
        RaiseException( MYEXCEPTION_CODE, 0, 0, 0 );
    }
    catch ( CMyException x ) {}
    catch ( ... ) {
        printf_s("This is invoked since "
            "_set_se_translator is not "
            "supported when /clr is used\n" );
    }
}

#pragma unmanaged
void unmanaged_func() {
    try {
        RaiseException( MYEXCEPTION_CODE,
                        0, 0, 0 );
    }
    catch ( CMyException x ) {
        printf("Caught an SEH exception with "
            "exception code: %x\n", x.m_ErrorCode );
    }
    catch ( ... ) {}
}

// #pragma managed
int main( int argc, char ** argv ) {
    _set_se_translator( my_trans_func );

    // It does not matter whether the translator function
    // is registered in managed or unmanaged code
    managed_func();
    unmanaged_func();
}

```

輸出

```

This is invoked since _set_se_translator is not supported when /clr is used
In my_trans_func.
Caught an SEH exception with exception code: e0000101

```

另請參閱

[例外狀況處理](#)

[safe_cast](#)

[MSVC 中的例外處理](#)

finally

2020/11/2 • • [Edit Online](#)

除了 `try` 和子句之外 `catch`，CLR 例外狀況處理也支援 `finally` 子句。此語義與 `__finally` 結構化例外狀況處理(SEH)中的區塊相同。`__finally` 區塊可以在 `try` 或區塊之後 `catch`。

備註

區塊的目的 `finally` 是要清除例外狀況發生後剩餘的任何資源。請注意，`finally` 即使沒有擲回例外狀況，也一定會執行區塊。`catch` 只有在相關聯的區塊內擲回 managed 例外狀況時，才會執行區塊 `try`。

`finally` 是內容相關的關鍵字；如需詳細資訊，請參閱[內容相關關鍵字](#)。

範例

下列範例示範簡單的 `finally` 區塊：

```
// keyword__finally.cpp
// compile with: /clr
using namespace System;

ref class MyException: public System::Exception{};

void ThrowMyException() {
    throw gcnew MyException;
}

int main() {
    try {
        ThrowMyException();
    }
    catch ( MyException^ e ) {
        Console::WriteLine( "in catch" );
        Console::WriteLine( e->GetType() );
    }
    finally {
        Console::WriteLine( "in finally" );
    }
}
```

```
in catch
MyException
in finally
```

另請參閱

[例外狀況處理](#)

作法：攔截 MSIL 所擲回機器碼的例外狀況

2020/11/2 • [Edit Online](#)

在機器碼中，您可以從 MSIL 攜截原生 C++ 例外狀況。您可以使用和來捕捉 CLR 例外狀況 `__try` `__except`。

如需詳細資訊，請參閱 [結構化例外狀況處理 \(C/c++\)](#) 和 [新式 C++ 的例外狀況和錯誤處理最佳做法](#)。

範例 1

下列範例會定義具有兩個函式的模組，一個會擲回原生例外狀況，另一個則擲回 MSIL 例外狀況。

```
// catch_MSIL_in_native.cpp
// compile with: /clr /c
void Test() {
    throw ("error");
}

void Test2() {
    throw (gcnew System::Exception("error2"));
}
```

範例 2

下列範例會定義攔截原生和 MSIL 例外狀況的模組。

```
// catch_MSIL_in_native_2.cpp
// compile with: /clr catch_MSIL_in_native.obj
#include <iostream>
using namespace std;
void Test();
void Test2();

void Func() {
    // catch any exception from MSIL
    // should not catch Visual C++ exceptions like this
    // runtime may not destroy the object thrown
    __try {
        Test2();
    }
    __except(1) {
        cout << "caught an exception" << endl;
    }
}

int main() {
    // catch native C++ exception from MSIL
    try {
        Test();
    }
    catch(char * s) {
        cout << s << endl;
    }
    Func();
}
```

```
error  
caught an exception
```

請參閱

[例外狀況處理](#)

如何：定義與安裝全域例外狀況處理常式

2019/12/10 • [Edit Online](#)

下列程式碼範例會示範如何捕捉未處理的例外狀況。範例表單包含一個按鈕，當按下時，會執行 null 參考，導致擲回例外狀況。這種功能代表一般的程式碼失敗。主要函式所安裝的整個應用程式例外狀況處理常式會攔截產生的例外狀況。

這是藉由將委派系結至 `ThreadException` 事件來完成。在此情況下，後續的例外狀況就會傳送至 `App::OnUnhandled` 方法。

範例

```
// global_exception_handler.cpp
// compile with: /clr
#using <system.dll>
#using <system.drawing.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Threading;
using namespace System::Drawing;
using namespace System::Windows::Forms;

ref class MyForm : public Form
{
    Button^ b;
public:
    MyForm( )
    {
        b = gcnew Button( );
        b->Text = "Do Null Access";
        b->Size = Drawing::Size(150, 30);
        b->Click += gcnew EventHandler(this, &MyForm::OnClick);
        Controls->Add(b);
    }
    void OnClick(Object^ sender, EventArgs^ args)
    {
        // do something illegal, like call through a null pointer...
        Object^ o = nullptr;
        o->ToString( );
    }
};

ref class App
{
public:
    static void OnUnhandled(Object^ sender, ThreadExceptionEventArgs^ e)
    {
        MessageBox::Show(e->Exception->Message, "Global Exception");
        Application::ExitThread( );
    }
};

int main()
{
    Application::ThreadException += gcnew
        ThreadExceptionHandler(App::OnUnhandled);

    MyForm^ form = gcnew MyForm( );
    Application::Run(form);
}
```

請參閱

[例外狀況處理](#)

Boxing (C++/CLI)

2020/11/2 • [Edit Online](#)

「裝箱」是將實值型別轉換成型別或實值型別所實作為 `object` 任何介面型別的程式。當 common language runtime (CLR) 方塊為實值型別時，它會將值包裝在中，`System.Object` 並將它儲存在 managed 堆積上。Unbox 處理則會從物件中擷取實值類型。Boxing 是隱含處理，unboxing 則是明確處理。

相關文章

II	II
如何 : 明確要求裝箱	描述如何在變數上明確要求裝箱。
如何 : 使用 gcnew 建立實數值型別及使用隱含的裝箱	示範如何使用 <code>gcnew</code> 來建立可放在 managed 垃圾收集堆積上的已裝箱實數值型別。
作法 : Unbox	顯示如何進行 Unbox 處理和修改值。
標準轉換和隱含的裝箱	顯示編譯器在需要進行裝箱的轉換時選擇了標準轉換。
使用 c + +/CLI 進行 .NET 程式設計 (Visual C++)	在 Visual C++ 文件中關於 .NET 程式設計的最上層文件。

如何：明確要求 Boxing

2019/12/10 • [Edit Online](#)

您可以藉由將變數指派給 `object` 類型的變數，明確地要求裝箱。

範例

```
// vcmlcppv2_explicit_boxing3.cpp
// compile with: /clr
using namespace System;

void f(int i) {
    Console::WriteLine("f(int i)");
}

void f(Object ^o) {
    Console::WriteLine("f(Object^ o)");
}

int main() {
    int i = 5;
    Object ^ O = i;    // forces i to be boxed
    f(i);
    f(O);
    f( (Object^)i ); // boxes i
}
```

```
f(int i)
f(Object^ o)
f(Object^ o)
```

請參閱

[Boxing](#)

作法：使用 gcnew 建立實值類型及使用隱含 Boxing

2019/12/10 • [Edit Online](#)

在實值型別上使用gcnew將會建立一個已裝箱的實值型別，然後可以將它放在受管理的垃圾收集堆積上。

範例

```
// vcmcppv2_explicit_boxing4.cpp
// compile with: /clr
public value class V {
public:
    int m_i;
    V(int i) : m_i(i) {}
};

public ref struct TC {
    void do_test(V^ v) {
        if (v != nullptr)
            ;
        else
            ;
    }
};

int main() {
    V^ v = gcnew V(42);
    TC^ tc = gcnew TC;
    tc->do_test(v);
}
```

請參閱

[Boxing](#)

如何 : Unbox

2019/12/10 • [Edit Online](#)

顯示如何進行 Unbox 處理和修改值。

範例

```
// vcmcppv2_unboxing.cpp
// compile with: /clr
using namespace System;

int main() {
    int ^ i = gcnew int(13);
    int j;
    Console::WriteLine(*i);    // unboxing
    *i = 14;    // unboxing and assignment
    Console::WriteLine(*i);
    j = safe_cast<int>(i);    // unboxing and assignment
    Console::WriteLine(j);
}
```

```
13
14
14
```

請參閱

[Boxing](#)

標準轉換和隱含 Boxing

2019/12/10 • [Edit Online](#)

在需要進行 Boxing 的轉換時，編譯器會選擇進行標準轉換。

範例

```
// clr_implicit_boxing_Std_conversion.cpp
// compile with: /clr
int f3(int ^ i) {    // requires boxing
    return 1;
}

int f3(char c) {    // no boxing required, standard conversion
    return 2;
}

int main() {
    int i = 5;
    System::Console::WriteLine(f3(i));
}
```

2

請參閱

[Boxing](#)