## 学习大纲

|  |
| --- |
|  |

## ServiceMain函数

|  |
| --- |
| 当服务控制程序请求运行新服务时，服务控制管理器 (SCM) 启动服务并将启动请求发送到控制调度程序。 控制调度程序创建一个新线程来为服务执行 [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数。 有关示例，请参阅 [编写 ServiceMain 函数](https://learn.microsoft.com/zh-cn/windows/win32/services/writing-a-servicemain-function)。  [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数应执行以下任务：   1. 初始化所有全局变量。 2. 立即调用 [**RegisterServiceCtrlHandler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-registerservicectrlhandlera) 函数以注册 [**Handler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nc-winsvc-lphandler_function) 函数来处理服务的控制请求。 **RegisterServiceCtrlHandler** 的返回值是服务状态句柄，将在调用中用于通知 SCM 服务状态。 3. 执行初始化。 如果初始化代码的执行时间预计很短 (小于一秒) ，则可以直接在 [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 中执行初始化。   如果初始化时间预计超过 1 秒，则服务应使用以下初始化技术之一：   * + 调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 函数以报告SERVICE\_RUNNING但在初始化完成之前不接受任何控件。 该服务通过调用 **SetServiceStatus** 来执行此操作，其中 **dwCurrentState** 设置为 SERVICE\_RUNNING， **dwControlsAccepted** 在 [**SERVICE\_STATUS**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/ns-winsvc-service_status) 结构中设置为 0。 这可确保 SCM 在服务准备就绪之前不会向服务发送任何控制请求，并释放 SCM 来管理其他服务。 对于性能而言，建议使用初始化方法，尤其是对于自动启动服务。   + 报告SERVICE\_START\_PENDING，不接受任何控件，并指定等待提示。 如果服务的初始化代码执行的任务预计花费的时间超过初始等待提示值，则代码必须定期调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 函数， (可能具有修订的等待提示) 以指示正在取得进展。 请确保仅在初始化正在进行时调用 **SetServiceStatus** 。 否则，SCM 可以等待服务进入SERVICE\_RUNNING状态（假设服务正在取得进展并阻止其他服务启动）。 请勿从单独的线程调用 **SetServiceStatus** ，除非你确信执行初始化的线程确实取得了进展。   使用此方法的服务还可以指定检查点值，并在长时间初始化期间定期递增值。 启动该服务的程序可以调用 [**QueryServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-queryservicestatus) 或 [**QueryServiceStatusEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-queryservicestatusex)，以从 SCM 获取最新的检查点值，并使用该值向用户报告增量进度。   1. 初始化完成后，调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 将服务状态设置为 SERVICE\_RUNNING并指定服务准备接受的控件。 有关控件的列表，请参阅 [**SERVICE\_STATUS**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/ns-winsvc-service_status) 结构。 2. 执行服务任务，或者，如果没有挂起的任务，请将控制权返回给调用方。 服务状态的任何更改都要求调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 来报告新的状态信息。 3. 如果在服务初始化或运行时出错，则服务应调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) ，以将服务状态设置为SERVICE\_STOP\_PENDING如果清理时间较长。 清理完成后，调用 **SetServiceStatus** ，将服务状态设置为从要终止的最后一个线程SERVICE\_STOPPED。 请务必设置[**SERVICE\_STATUS**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/ns-winsvc-service_status)结构的 **dwServiceSpecificExitCode** 和 **dwWin32ExitCode** 成员以识别错误。   以下示例中的 SvcMain 函数是示例服务的 [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数。 SvcMain 可以按照控制台应用程序的 **main** 函数的方式访问服务的命令行参数。 第一个参数包含在第二个参数中传递给服务的参数数。 始终至少有一个参数。 第二个参数是指向字符串指针数组的指针。 数组中的第一项始终是服务名称。  SvcMain 函数首先调用 [**RegisterServiceCtrlHandler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-registerservicectrlhandlera) 函数，以将 SvcCtrlHandler 函数注册为服务的 [**Handler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nc-winsvc-lphandler_function) 函数并开始初始化。 **RegisterServiceCtrlHandler** 应为 [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 中的第一个非失败函数，以便服务可以使用此函数返回的状态句柄在发生错误时调用具有SERVICE\_STOPPED状态的 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 。  接下来，SvcMain 函数调用 ReportSvcStatus 函数以指示其初始状态为SERVICE\_START\_PENDING。 当服务处于此状态时，不会接受任何控件。 为了简化服务的逻辑，建议服务在执行初始化时不要接受任何控件。  最后，SvcMain 函数调用 SvcInit 函数来执行特定于服务的初始化，并开始服务要执行的工作。  示例初始化函数 SvcInit 是一个非常简单的示例;它不会执行更复杂的初始化任务，例如创建其他线程。 它创建一个事件，服务控制处理程序可以发出信号来指示服务应停止，然后调用 ReportSvcStatus 以指示服务已进入SERVICE\_RUNNING状态。 此时，服务已完成其初始化，并已准备好接受控制。 为获得最佳系统性能，应用程序应在 25-100 毫秒内进入运行状态。  由于此示例服务未完成任何实际任务，SvcInit 只需通过调用 [**WaitForSingleObject**](https://learn.microsoft.com/zh-cn/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject) 函数来等待服务停止事件发出信号，调用 ReportSvcStatus 以指示服务已进入SERVICE\_STOPPED状态，并返回 。 (请注意，函数返回而不是调用 [**ExitThread**](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-exitthread) 函数非常重要，因为返回 允许清理为 arguments.) 可以使用 [**RegisterWaitForSingleObject**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-registerwaitforsingleobject) 函数而不是 **WaitForSingleObject** 函数来执行其他清理任务。 运行 [**ServiceMain**](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数的线程终止，但服务本身继续运行。 当服务控制处理程序发出事件信号时，线程池中的线程将执行回调以执行其他清理，包括将状态设置为SERVICE\_STOPPED。  请注意，此示例使用 SvcReportEvent 将错误事件写入事件日志。 有关 SvcReportEvent 的源代码，请参阅 [Svc.cpp](https://learn.microsoft.com/zh-cn/windows/win32/services/svc-cpp)。 有关控件处理程序函数的示例，请参阅 [编写控件处理程序函数](https://learn.microsoft.com/zh-cn/windows/win32/services/writing-a-control-handler-function)。  此示例使用以下全局定义。  C++复制  #define SVCNAME TEXT("SvcName")  SERVICE\_STATUS gSvcStatus;  SERVICE\_STATUS\_HANDLE gSvcStatusHandle;  HANDLE ghSvcStopEvent = NULL;  以下示例片段取自完整的服务示例。  C++复制  //  // Purpose:  // Entry point for the service  //  // Parameters:  // dwArgc - Number of arguments in the lpszArgv array  // lpszArgv - Array of strings. The first string is the name of  // the service and subsequent strings are passed by the process  // that called the StartService function to start the service.  //  // Return value:  // None.  //  VOID WINAPI SvcMain( DWORD dwArgc, LPTSTR \*lpszArgv )  {  // Register the handler function for the service  gSvcStatusHandle = RegisterServiceCtrlHandler(  SVCNAME,  SvcCtrlHandler);  if( !gSvcStatusHandle )  {  SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));  return;  }  // These SERVICE\_STATUS members remain as set here  gSvcStatus.dwServiceType = SERVICE\_WIN32\_OWN\_PROCESS;  gSvcStatus.dwServiceSpecificExitCode = 0;  // Report initial status to the SCM  ReportSvcStatus( SERVICE\_START\_PENDING, NO\_ERROR, 3000 );  // Perform service-specific initialization and work.  SvcInit( dwArgc, lpszArgv );  }  //  // Purpose:  // The service code  //  // Parameters:  // dwArgc - Number of arguments in the lpszArgv array  // lpszArgv - Array of strings. The first string is the name of  // the service and subsequent strings are passed by the process  // that called the StartService function to start the service.  //  // Return value:  // None  //  VOID SvcInit( DWORD dwArgc, LPTSTR \*lpszArgv)  {  // TO\_DO: Declare and set any required variables.  // Be sure to periodically call ReportSvcStatus() with  // SERVICE\_START\_PENDING. If initialization fails, call  // ReportSvcStatus with SERVICE\_STOPPED.  // Create an event. The control handler function, SvcCtrlHandler,  // signals this event when it receives the stop control code.  ghSvcStopEvent = CreateEvent(  NULL, // default security attributes  TRUE, // manual reset event  FALSE, // not signaled  NULL); // no name  if ( ghSvcStopEvent == NULL)  {  ReportSvcStatus( SERVICE\_STOPPED, NO\_ERROR, 0 );  return;  }  // Report running status when initialization is complete.  ReportSvcStatus( SERVICE\_RUNNING, NO\_ERROR, 0 );  // TO\_DO: Perform work until service stops.  while(1)  {  // Check whether to stop the service.  WaitForSingleObject(ghSvcStopEvent, INFINITE);  ReportSvcStatus( SERVICE\_STOPPED, NO\_ERROR, 0 );  return;  }  }  //  // Purpose:  // Sets the current service status and reports it to the SCM.  //  // Parameters:  // dwCurrentState - The current state (see SERVICE\_STATUS)  // dwWin32ExitCode - The system error code  // dwWaitHint - Estimated time for pending operation,  // in milliseconds  //  // Return value:  // None  //  VOID ReportSvcStatus( DWORD dwCurrentState,  DWORD dwWin32ExitCode,  DWORD dwWaitHint)  {  static DWORD dwCheckPoint = 1;  // Fill in the SERVICE\_STATUS structure.  gSvcStatus.dwCurrentState = dwCurrentState;  gSvcStatus.dwWin32ExitCode = dwWin32ExitCode;  gSvcStatus.dwWaitHint = dwWaitHint;  if (dwCurrentState == SERVICE\_START\_PENDING)  gSvcStatus.dwControlsAccepted = 0;  else gSvcStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP;  if ( (dwCurrentState == SERVICE\_RUNNING) ||  (dwCurrentState == SERVICE\_STOPPED) )  gSvcStatus.dwCheckPoint = 0;  else gSvcStatus.dwCheckPoint = dwCheckPoint++;  // Report the status of the service to the SCM.  SetServiceStatus( gSvcStatusHandle, &gSvcStatus );  } |

## 使用服务，参考：[使用服务程序.docx](使用Windows服务程序.docx)

## 完整的服务示例

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 本节中的主题构成了完整的服务示例： [Sample.mc](https://learn.microsoft.com/zh-cn/windows/win32/services/sample-mc) (包含错误消息)  |  | | --- | | Sample.mc  * 项目 * 2023/06/13 * 3 个参与者   反馈  下面是一个示例消息文件，可用于生成在将事件写入事件日志时要与服务示例一起使用的仅限资源的 DLL。  使用以下步骤生成 DLL：   1. **mc -U sample.mc** 2. **rc -r sample.rc** 3. **link -dll -noentry -out:sample.dll sample.res**   syntax复制  MessageIdTypedef=DWORD  SeverityNames=(Success=0x0:STATUS\_SEVERITY\_SUCCESS  Informational=0x1:STATUS\_SEVERITY\_INFORMATIONAL  Warning=0x2:STATUS\_SEVERITY\_WARNING  Error=0x3:STATUS\_SEVERITY\_ERROR  )  FacilityNames=(System=0x0:FACILITY\_SYSTEM  Runtime=0x2:FACILITY\_RUNTIME  Stubs=0x3:FACILITY\_STUBS  Io=0x4:FACILITY\_IO\_ERROR\_CODE  )  LanguageNames=(English=0x409:MSG00409)  ; // The following are message definitions.  MessageId=0x1  Severity=Error  Facility=Runtime  SymbolicName=SVC\_ERROR  Language=English  An error has occurred (%2).  .  ; // A message file must end with a period on its own line  ; // followed by a blank line. |  [Svc.cpp](https://learn.microsoft.com/zh-cn/windows/win32/services/svc-cpp) (包含服务代码)  |  | | --- | | 下面是完整的服务示例。 将此代码用作模板时，只需将代码添加到 前面由 TO\_DO的节。  生成示例时，请务必与 Kernel32.lib 和 Advapi32.lib 链接。 生成仅限资源的 DLL 时生成 Sample.h 文件，Sample.dll。 有关详细信息，请参阅 [Sample.mc](https://learn.microsoft.com/zh-cn/windows/win32/services/sample-mc)。  C++复制  #include <windows.h>  #include <tchar.h>  #include <strsafe.h>  #include "sample.h"  #pragma comment(lib, "advapi32.lib")  #define SVCNAME TEXT("SvcName")  SERVICE\_STATUS gSvcStatus;  SERVICE\_STATUS\_HANDLE gSvcStatusHandle;  HANDLE ghSvcStopEvent = NULL;  VOID SvcInstall(void);  VOID WINAPI SvcCtrlHandler( DWORD );  VOID WINAPI SvcMain( DWORD, LPTSTR \* );  VOID ReportSvcStatus( DWORD, DWORD, DWORD );  VOID SvcInit( DWORD, LPTSTR \* );  VOID SvcReportEvent( LPTSTR );  //  // Purpose:  // Entry point for the process  //  // Parameters:  // None  //  // Return value:  // None, defaults to 0 (zero)  //  int \_\_cdecl \_tmain(int argc, TCHAR \*argv[])  {  // If command-line parameter is "install", install the service.  // Otherwise, the service is probably being started by the SCM.  if( lstrcmpi( argv[1], TEXT("install")) == 0 )  {  SvcInstall();  return 0;  }  // TO\_DO: Add any additional services for the process to this table.  SERVICE\_TABLE\_ENTRY DispatchTable[] =  {  { SVCNAME, (LPSERVICE\_MAIN\_FUNCTION) SvcMain },  { NULL, NULL }  };    // This call returns when the service has stopped.  // The process should simply terminate when the call returns.  if (!StartServiceCtrlDispatcher( DispatchTable ))  {  SvcReportEvent(TEXT("StartServiceCtrlDispatcher"));  }  }  //  // Purpose:  // Installs a service in the SCM database  //  // Parameters:  // None  //  // Return value:  // None  //  VOID SvcInstall()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  TCHAR szUnquotedPath[MAX\_PATH];  if( !GetModuleFileName( NULL, szUnquotedPath, MAX\_PATH ) )  {  printf("Cannot install service (%d)\n", GetLastError());  return;  }  // In case the path contains a space, it must be quoted so that  // it is correctly interpreted. For example,  // "d:\my share\myservice.exe" should be specified as  // ""d:\my share\myservice.exe"".  TCHAR szPath[MAX\_PATH];  StringCbPrintf(szPath, MAX\_PATH, TEXT("\"%s\""), szUnquotedPath);  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Create the service  schService = CreateService(  schSCManager, // SCM database  SVCNAME, // name of service  SVCNAME, // service name to display  SERVICE\_ALL\_ACCESS, // desired access  SERVICE\_WIN32\_OWN\_PROCESS, // service type  SERVICE\_DEMAND\_START, // start type  SERVICE\_ERROR\_NORMAL, // error control type  szPath, // path to service's binary  NULL, // no load ordering group  NULL, // no tag identifier  NULL, // no dependencies  NULL, // LocalSystem account  NULL); // no password    if (schService == NULL)  {  printf("CreateService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  else printf("Service installed successfully\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Entry point for the service  //  // Parameters:  // dwArgc - Number of arguments in the lpszArgv array  // lpszArgv - Array of strings. The first string is the name of  // the service and subsequent strings are passed by the process  // that called the StartService function to start the service.  //  // Return value:  // None.  //  VOID WINAPI SvcMain( DWORD dwArgc, LPTSTR \*lpszArgv )  {  // Register the handler function for the service  gSvcStatusHandle = RegisterServiceCtrlHandler(  SVCNAME,  SvcCtrlHandler);  if( !gSvcStatusHandle )  {  SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));  return;  }  // These SERVICE\_STATUS members remain as set here  gSvcStatus.dwServiceType = SERVICE\_WIN32\_OWN\_PROCESS;  gSvcStatus.dwServiceSpecificExitCode = 0;  // Report initial status to the SCM  ReportSvcStatus( SERVICE\_START\_PENDING, NO\_ERROR, 3000 );  // Perform service-specific initialization and work.  SvcInit( dwArgc, lpszArgv );  }  //  // Purpose:  // The service code  //  // Parameters:  // dwArgc - Number of arguments in the lpszArgv array  // lpszArgv - Array of strings. The first string is the name of  // the service and subsequent strings are passed by the process  // that called the StartService function to start the service.  //  // Return value:  // None  //  VOID SvcInit( DWORD dwArgc, LPTSTR \*lpszArgv)  {  // TO\_DO: Declare and set any required variables.  // Be sure to periodically call ReportSvcStatus() with  // SERVICE\_START\_PENDING. If initialization fails, call  // ReportSvcStatus with SERVICE\_STOPPED.  // Create an event. The control handler function, SvcCtrlHandler,  // signals this event when it receives the stop control code.  ghSvcStopEvent = CreateEvent(  NULL, // default security attributes  TRUE, // manual reset event  FALSE, // not signaled  NULL); // no name  if ( ghSvcStopEvent == NULL)  {  ReportSvcStatus( SERVICE\_STOPPED, GetLastError(), 0 );  return;  }  // Report running status when initialization is complete.  ReportSvcStatus( SERVICE\_RUNNING, NO\_ERROR, 0 );  // TO\_DO: Perform work until service stops.  while(1)  {  // Check whether to stop the service.  WaitForSingleObject(ghSvcStopEvent, INFINITE);  ReportSvcStatus( SERVICE\_STOPPED, NO\_ERROR, 0 );  return;  }  }  //  // Purpose:  // Sets the current service status and reports it to the SCM.  //  // Parameters:  // dwCurrentState - The current state (see SERVICE\_STATUS)  // dwWin32ExitCode - The system error code  // dwWaitHint - Estimated time for pending operation,  // in milliseconds  //  // Return value:  // None  //  VOID ReportSvcStatus( DWORD dwCurrentState,  DWORD dwWin32ExitCode,  DWORD dwWaitHint)  {  static DWORD dwCheckPoint = 1;  // Fill in the SERVICE\_STATUS structure.  gSvcStatus.dwCurrentState = dwCurrentState;  gSvcStatus.dwWin32ExitCode = dwWin32ExitCode;  gSvcStatus.dwWaitHint = dwWaitHint;  if (dwCurrentState == SERVICE\_START\_PENDING)  gSvcStatus.dwControlsAccepted = 0;  else gSvcStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP;  if ( (dwCurrentState == SERVICE\_RUNNING) ||  (dwCurrentState == SERVICE\_STOPPED) )  gSvcStatus.dwCheckPoint = 0;  else gSvcStatus.dwCheckPoint = dwCheckPoint++;  // Report the status of the service to the SCM.  SetServiceStatus( gSvcStatusHandle, &gSvcStatus );  }  //  // Purpose:  // Called by SCM whenever a control code is sent to the service  // using the ControlService function.  //  // Parameters:  // dwCtrl - control code  //  // Return value:  // None  //  VOID WINAPI SvcCtrlHandler( DWORD dwCtrl )  {  // Handle the requested control code.  switch(dwCtrl)  {  case SERVICE\_CONTROL\_STOP:  ReportSvcStatus(SERVICE\_STOP\_PENDING, NO\_ERROR, 0);  // Signal the service to stop.  SetEvent(ghSvcStopEvent);  ReportSvcStatus(gSvcStatus.dwCurrentState, NO\_ERROR, 0);    return;    case SERVICE\_CONTROL\_INTERROGATE:  break;    default:  break;  }    }  //  // Purpose:  // Logs messages to the event log  //  // Parameters:  // szFunction - name of function that failed  //  // Return value:  // None  //  // Remarks:  // The service must have an entry in the Application event log.  //  VOID SvcReportEvent(LPTSTR szFunction)  {  HANDLE hEventSource;  LPCTSTR lpszStrings[2];  TCHAR Buffer[80];  hEventSource = RegisterEventSource(NULL, SVCNAME);  if( NULL != hEventSource )  {  StringCchPrintf(Buffer, 80, TEXT("%s failed with %d"), szFunction, GetLastError());  lpszStrings[0] = SVCNAME;  lpszStrings[1] = Buffer;  ReportEvent(hEventSource, // event log handle  EVENTLOG\_ERROR\_TYPE, // event type  0, // event category  SVC\_ERROR, // event identifier  NULL, // no security identifier  2, // size of lpszStrings array  0, // no binary data  lpszStrings, // array of strings  NULL); // no binary data  DeregisterEventSource(hEventSource);  }  } |  [SvcConfig.cpp](https://learn.microsoft.com/zh-cn/windows/win32/services/svcconfig-cpp) (包含服务配置代码)  |  | | --- | | 生成示例时，请确保与 Kernel32.lib 和 Advapi32.lib 链接。  C++复制  #include <windows.h>  #include <tchar.h>  #include <strsafe.h>  #include <stdio.h>  #pragma comment(lib, "advapi32.lib")  TCHAR szCommand[10];  TCHAR szSvcName[80];  VOID \_\_stdcall DisplayUsage(void);  VOID \_\_stdcall DoQuerySvc(void);  VOID \_\_stdcall DoUpdateSvcDesc(void);  VOID \_\_stdcall DoDisableSvc(void);  VOID \_\_stdcall DoEnableSvc(void);  VOID \_\_stdcall DoDeleteSvc(void);  //  // Purpose:  // Entry point function. Executes specified command from user.  //  // Parameters:  // Command-line syntax is: svcconfig [command] [service\_path]  //  // Return value:  // None, defaults to 0 (zero)  //  int \_\_cdecl \_tmain(int argc, TCHAR \*argv[])  {  printf("\n");  if( argc != 3 )  {  printf("ERROR:\tIncorrect number of arguments\n\n");  DisplayUsage();  return;  }  StringCchCopy(szCommand, 10, argv[1]);  StringCchCopy(szSvcName, 80, argv[2]);  if (lstrcmpi( szCommand, TEXT("query")) == 0 )  DoQuerySvc();  else if (lstrcmpi( szCommand, TEXT("describe")) == 0 )  DoUpdateSvcDesc();  else if (lstrcmpi( szCommand, TEXT("disable")) == 0 )  DoDisableSvc();  else if (lstrcmpi( szCommand, TEXT("enable")) == 0 )  DoEnableSvc();  else if (lstrcmpi( szCommand, TEXT("delete")) == 0 )  DoDeleteSvc();  else  {  \_tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);  DisplayUsage();  }  }  VOID \_\_stdcall DisplayUsage()  {  printf("Description:\n");  printf("\tCommand-line tool that configures a service.\n\n");  printf("Usage:\n");  printf("\tsvcconfig [command] [service\_name]\n\n");  printf("\t[command]\n");  printf("\t query\n");  printf("\t describe\n");  printf("\t disable\n");  printf("\t enable\n");  printf("\t delete\n");  }  //  // Purpose:  // Retrieves and displays the current service configuration.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoQuerySvc()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  LPQUERY\_SERVICE\_CONFIG lpsc;  LPSERVICE\_DESCRIPTION lpsd;  DWORD dwBytesNeeded, cbBufSize, dwError;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_QUERY\_CONFIG); // need query config access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Get the configuration information.    if( !QueryServiceConfig(  schService,  NULL,  0,  &dwBytesNeeded))  {  dwError = GetLastError();  if( ERROR\_INSUFFICIENT\_BUFFER == dwError )  {  cbBufSize = dwBytesNeeded;  lpsc = (LPQUERY\_SERVICE\_CONFIG) LocalAlloc(LMEM\_FIXED, cbBufSize);  }  else  {  printf("QueryServiceConfig failed (%d)", dwError);  goto cleanup;  }  }    if( !QueryServiceConfig(  schService,  lpsc,  cbBufSize,  &dwBytesNeeded) )  {  printf("QueryServiceConfig failed (%d)", GetLastError());  goto cleanup;  }  if( !QueryServiceConfig2(  schService,  SERVICE\_CONFIG\_DESCRIPTION,  NULL,  0,  &dwBytesNeeded))  {  dwError = GetLastError();  if( ERROR\_INSUFFICIENT\_BUFFER == dwError )  {  cbBufSize = dwBytesNeeded;  lpsd = (LPSERVICE\_DESCRIPTION) LocalAlloc(LMEM\_FIXED, cbBufSize);  }  else  {  printf("QueryServiceConfig2 failed (%d)", dwError);  goto cleanup;  }  }    if (! QueryServiceConfig2(  schService,  SERVICE\_CONFIG\_DESCRIPTION,  (LPBYTE) lpsd,  cbBufSize,  &dwBytesNeeded) )  {  printf("QueryServiceConfig2 failed (%d)", GetLastError());  goto cleanup;  }    // Print the configuration information.    \_tprintf(TEXT("%s configuration: \n"), szSvcName);  \_tprintf(TEXT(" Type: 0x%x\n"), lpsc->dwServiceType);  \_tprintf(TEXT(" Start Type: 0x%x\n"), lpsc->dwStartType);  \_tprintf(TEXT(" Error Control: 0x%x\n"), lpsc->dwErrorControl);  \_tprintf(TEXT(" Binary path: %s\n"), lpsc->lpBinaryPathName);  \_tprintf(TEXT(" Account: %s\n"), lpsc->lpServiceStartName);  if (lpsd->lpDescription != NULL && lstrcmp(lpsd->lpDescription, TEXT("")) != 0)  \_tprintf(TEXT(" Description: %s\n"), lpsd->lpDescription);  if (lpsc->lpLoadOrderGroup != NULL && lstrcmp(lpsc->lpLoadOrderGroup, TEXT("")) != 0)  \_tprintf(TEXT(" Load order group: %s\n"), lpsc->lpLoadOrderGroup);  if (lpsc->dwTagId != 0)  \_tprintf(TEXT(" Tag ID: %d\n"), lpsc->dwTagId);  if (lpsc->lpDependencies != NULL && lstrcmp(lpsc->lpDependencies, TEXT("")) != 0)  \_tprintf(TEXT(" Dependencies: %s\n"), lpsc->lpDependencies);    LocalFree(lpsc);  LocalFree(lpsd);  cleanup:  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Disables the service.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoDisableSvc()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_CHANGE\_CONFIG); // need change config access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Change the service start type.  if (! ChangeServiceConfig(  schService, // handle of service  SERVICE\_NO\_CHANGE, // service type: no change  SERVICE\_DISABLED, // service start type  SERVICE\_NO\_CHANGE, // error control: no change  NULL, // binary path: no change  NULL, // load order group: no change  NULL, // tag ID: no change  NULL, // dependencies: no change  NULL, // account name: no change  NULL, // password: no change  NULL) ) // display name: no change  {  printf("ChangeServiceConfig failed (%d)\n", GetLastError());  }  else printf("Service disabled successfully.\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Enables the service.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoEnableSvc()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_CHANGE\_CONFIG); // need change config access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Change the service start type.  if (! ChangeServiceConfig(  schService, // handle of service  SERVICE\_NO\_CHANGE, // service type: no change  SERVICE\_DEMAND\_START, // service start type  SERVICE\_NO\_CHANGE, // error control: no change  NULL, // binary path: no change  NULL, // load order group: no change  NULL, // tag ID: no change  NULL, // dependencies: no change  NULL, // account name: no change  NULL, // password: no change  NULL) ) // display name: no change  {  printf("ChangeServiceConfig failed (%d)\n", GetLastError());  }  else printf("Service enabled successfully.\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Updates the service description to "This is a test description".  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoUpdateSvcDesc()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  SERVICE\_DESCRIPTION sd;  LPTSTR szDesc = TEXT("This is a test description");  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_CHANGE\_CONFIG); // need change config access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Change the service description.  sd.lpDescription = szDesc;  if( !ChangeServiceConfig2(  schService, // handle to service  SERVICE\_CONFIG\_DESCRIPTION, // change: description  &sd) ) // new description  {  printf("ChangeServiceConfig2 failed\n");  }  else printf("Service description updated successfully.\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Deletes a service from the SCM database  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoDeleteSvc()  {  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  DELETE); // need delete access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Delete the service.    if (! DeleteService(schService) )  {  printf("DeleteService failed (%d)\n", GetLastError());  }  else printf("Service deleted successfully\n");    CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  } |  [SvcControl.cpp](https://learn.microsoft.com/zh-cn/windows/win32/services/svccontrol-cpp) (包含服务控制代码)  |  | | --- | | 生成示例时，请务必与 Kernel32.lib 和 Advapi32.lib 链接。  C++复制  #include <windows.h>  #include <tchar.h>  #include <strsafe.h>  #include <aclapi.h>  #include <stdio.h>  #pragma comment(lib, "advapi32.lib")  TCHAR szCommand[10];  TCHAR szSvcName[80];  SC\_HANDLE schSCManager;  SC\_HANDLE schService;  VOID \_\_stdcall DisplayUsage(void);  VOID \_\_stdcall DoStartSvc(void);  VOID \_\_stdcall DoUpdateSvcDacl(void);  VOID \_\_stdcall DoStopSvc(void);  BOOL \_\_stdcall StopDependentServices(void);  //  // Purpose:  // Entry point function. Executes specified command from user.  //  // Parameters:  // Command-line syntax is: svccontrol [command] [service\_name]  //  // Return value:  // None  //  void \_tmain(int argc, TCHAR \*argv[])  {  printf("\n");  if( argc != 3 )  {  printf("ERROR: Incorrect number of arguments\n\n");  DisplayUsage();  return;  }  StringCchCopy(szCommand, 10, argv[1]);  StringCchCopy(szSvcName, 80, argv[2]);  if (lstrcmpi( szCommand, TEXT("start")) == 0 )  DoStartSvc();  else if (lstrcmpi( szCommand, TEXT("dacl")) == 0 )  DoUpdateSvcDacl();  else if (lstrcmpi( szCommand, TEXT("stop")) == 0 )  DoStopSvc();  else  {  \_tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);  DisplayUsage();  }  }  VOID \_\_stdcall DisplayUsage()  {  printf("Description:\n");  printf("\tCommand-line tool that controls a service.\n\n");  printf("Usage:\n");  printf("\tsvccontrol [command] [service\_name]\n\n");  printf("\t[command]\n");  printf("\t start\n");  printf("\t dacl\n");  printf("\t stop\n");  }  //  // Purpose:  // Starts the service if possible.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoStartSvc()  {  SERVICE\_STATUS\_PROCESS ssStatus;  DWORD dwOldCheckPoint;  DWORD dwStartTickCount;  DWORD dwWaitTime;  DWORD dwBytesNeeded;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // servicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_ALL\_ACCESS); // full access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Check the status in case the service is not stopped.  if (!QueryServiceStatusEx(  schService, // handle to service  SC\_STATUS\_PROCESS\_INFO, // information level  (LPBYTE) &ssStatus, // address of structure  sizeof(SERVICE\_STATUS\_PROCESS), // size of structure  &dwBytesNeeded ) ) // size needed if buffer is too small  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }  // Check if the service is already running. It would be possible  // to stop the service here, but for simplicity this example just returns.  if(ssStatus.dwCurrentState != SERVICE\_STOPPED && ssStatus.dwCurrentState != SERVICE\_STOP\_PENDING)  {  printf("Cannot start the service because it is already running\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }  // Save the tick count and initial checkpoint.  dwStartTickCount = GetTickCount();  dwOldCheckPoint = ssStatus.dwCheckPoint;  // Wait for the service to stop before attempting to start it.  while (ssStatus.dwCurrentState == SERVICE\_STOP\_PENDING)  {  // Do not wait longer than the wait hint. A good interval is  // one-tenth of the wait hint but not less than 1 second  // and not more than 10 seconds.    dwWaitTime = ssStatus.dwWaitHint / 10;  if( dwWaitTime < 1000 )  dwWaitTime = 1000;  else if ( dwWaitTime > 10000 )  dwWaitTime = 10000;  Sleep( dwWaitTime );  // Check the status until the service is no longer stop pending.    if (!QueryServiceStatusEx(  schService, // handle to service  SC\_STATUS\_PROCESS\_INFO, // information level  (LPBYTE) &ssStatus, // address of structure  sizeof(SERVICE\_STATUS\_PROCESS), // size of structure  &dwBytesNeeded ) ) // size needed if buffer is too small  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }  if ( ssStatus.dwCheckPoint > dwOldCheckPoint )  {  // Continue to wait and check.  dwStartTickCount = GetTickCount();  dwOldCheckPoint = ssStatus.dwCheckPoint;  }  else  {  if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)  {  printf("Timeout waiting for service to stop\n");  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }  }  }  // Attempt to start the service.  if (!StartService(  schService, // handle to service  0, // number of arguments  NULL) ) // no arguments  {  printf("StartService failed (%d)\n", GetLastError());  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }  else printf("Service start pending...\n");  // Check the status until the service is no longer start pending.    if (!QueryServiceStatusEx(  schService, // handle to service  SC\_STATUS\_PROCESS\_INFO, // info level  (LPBYTE) &ssStatus, // address of structure  sizeof(SERVICE\_STATUS\_PROCESS), // size of structure  &dwBytesNeeded ) ) // if buffer too small  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  return;  }    // Save the tick count and initial checkpoint.  dwStartTickCount = GetTickCount();  dwOldCheckPoint = ssStatus.dwCheckPoint;  while (ssStatus.dwCurrentState == SERVICE\_START\_PENDING)  {  // Do not wait longer than the wait hint. A good interval is  // one-tenth the wait hint, but no less than 1 second and no  // more than 10 seconds.    dwWaitTime = ssStatus.dwWaitHint / 10;  if( dwWaitTime < 1000 )  dwWaitTime = 1000;  else if ( dwWaitTime > 10000 )  dwWaitTime = 10000;  Sleep( dwWaitTime );  // Check the status again.    if (!QueryServiceStatusEx(  schService, // handle to service  SC\_STATUS\_PROCESS\_INFO, // info level  (LPBYTE) &ssStatus, // address of structure  sizeof(SERVICE\_STATUS\_PROCESS), // size of structure  &dwBytesNeeded ) ) // if buffer too small  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  break;  }    if ( ssStatus.dwCheckPoint > dwOldCheckPoint )  {  // Continue to wait and check.  dwStartTickCount = GetTickCount();  dwOldCheckPoint = ssStatus.dwCheckPoint;  }  else  {  if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)  {  // No progress made within the wait hint.  break;  }  }  }  // Determine whether the service is running.  if (ssStatus.dwCurrentState == SERVICE\_RUNNING)  {  printf("Service started successfully.\n");  }  else  {  printf("Service not started. \n");  printf(" Current State: %d\n", ssStatus.dwCurrentState);  printf(" Exit Code: %d\n", ssStatus.dwWin32ExitCode);  printf(" Check Point: %d\n", ssStatus.dwCheckPoint);  printf(" Wait Hint: %d\n", ssStatus.dwWaitHint);  }  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  //  // Purpose:  // Updates the service DACL to grant start, stop, delete, and read  // control access to the Guest account.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoUpdateSvcDacl()  {  EXPLICIT\_ACCESS ea;  SECURITY\_DESCRIPTOR sd;  PSECURITY\_DESCRIPTOR psd = NULL;  PACL pacl = NULL;  PACL pNewAcl = NULL;  BOOL bDaclPresent = FALSE;  BOOL bDaclDefaulted = FALSE;  DWORD dwError = 0;  DWORD dwSize = 0;  DWORD dwBytesNeeded = 0;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service  schService = OpenService(  schSCManager, // SCManager database  szSvcName, // name of service  READ\_CONTROL | WRITE\_DAC); // access    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Get the current security descriptor.  if (!QueryServiceObjectSecurity(schService,  DACL\_SECURITY\_INFORMATION,  &psd, // using NULL does not work on all versions  0,  &dwBytesNeeded))  {  if (GetLastError() == ERROR\_INSUFFICIENT\_BUFFER)  {  dwSize = dwBytesNeeded;  psd = (PSECURITY\_DESCRIPTOR)HeapAlloc(GetProcessHeap(),  HEAP\_ZERO\_MEMORY, dwSize);  if (psd == NULL)  {  // Note: HeapAlloc does not support GetLastError.  printf("HeapAlloc failed\n");  goto dacl\_cleanup;  }    if (!QueryServiceObjectSecurity(schService,  DACL\_SECURITY\_INFORMATION, psd, dwSize, &dwBytesNeeded))  {  printf("QueryServiceObjectSecurity failed (%d)\n", GetLastError());  goto dacl\_cleanup;  }  }  else  {  printf("QueryServiceObjectSecurity failed (%d)\n", GetLastError());  goto dacl\_cleanup;  }  }  // Get the DACL.  if (!GetSecurityDescriptorDacl(psd, &bDaclPresent, &pacl,  &bDaclDefaulted))  {  printf("GetSecurityDescriptorDacl failed(%d)\n", GetLastError());  goto dacl\_cleanup;  }  // Build the ACE.  BuildExplicitAccessWithName(&ea, TEXT("GUEST"),  SERVICE\_START | SERVICE\_STOP | READ\_CONTROL | DELETE,  SET\_ACCESS, NO\_INHERITANCE);  dwError = SetEntriesInAcl(1, &ea, pacl, &pNewAcl);  if (dwError != ERROR\_SUCCESS)  {  printf("SetEntriesInAcl failed(%d)\n", dwError);  goto dacl\_cleanup;  }  // Initialize a new security descriptor.  if (!InitializeSecurityDescriptor(&sd,  SECURITY\_DESCRIPTOR\_REVISION))  {  printf("InitializeSecurityDescriptor failed(%d)\n", GetLastError());  goto dacl\_cleanup;  }  // Set the new DACL in the security descriptor.  if (!SetSecurityDescriptorDacl(&sd, TRUE, pNewAcl, FALSE))  {  printf("SetSecurityDescriptorDacl failed(%d)\n", GetLastError());  goto dacl\_cleanup;  }  // Set the new DACL for the service object.  if (!SetServiceObjectSecurity(schService,  DACL\_SECURITY\_INFORMATION, &sd))  {  printf("SetServiceObjectSecurity failed(%d)\n", GetLastError());  goto dacl\_cleanup;  }  else printf("Service DACL updated successfully\n");  dacl\_cleanup:  CloseServiceHandle(schSCManager);  CloseServiceHandle(schService);  if(NULL != pNewAcl)  LocalFree((HLOCAL)pNewAcl);  if(NULL != psd)  HeapFree(GetProcessHeap(), 0, (LPVOID)psd);  }  //  // Purpose:  // Stops the service.  //  // Parameters:  // None  //  // Return value:  // None  //  VOID \_\_stdcall DoStopSvc()  {  SERVICE\_STATUS\_PROCESS ssp;  DWORD dwStartTime = GetTickCount();  DWORD dwBytesNeeded;  DWORD dwTimeout = 30000; // 30-second time-out  DWORD dwWaitTime;  // Get a handle to the SCM database.    schSCManager = OpenSCManager(  NULL, // local computer  NULL, // ServicesActive database  SC\_MANAGER\_ALL\_ACCESS); // full access rights    if (NULL == schSCManager)  {  printf("OpenSCManager failed (%d)\n", GetLastError());  return;  }  // Get a handle to the service.  schService = OpenService(  schSCManager, // SCM database  szSvcName, // name of service  SERVICE\_STOP |  SERVICE\_QUERY\_STATUS |  SERVICE\_ENUMERATE\_DEPENDENTS);    if (schService == NULL)  {  printf("OpenService failed (%d)\n", GetLastError());  CloseServiceHandle(schSCManager);  return;  }  // Make sure the service is not already stopped.  if ( !QueryServiceStatusEx(  schService,  SC\_STATUS\_PROCESS\_INFO,  (LPBYTE)&ssp,  sizeof(SERVICE\_STATUS\_PROCESS),  &dwBytesNeeded ) )  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  goto stop\_cleanup;  }  if ( ssp.dwCurrentState == SERVICE\_STOPPED )  {  printf("Service is already stopped.\n");  goto stop\_cleanup;  }  // If a stop is pending, wait for it.  while ( ssp.dwCurrentState == SERVICE\_STOP\_PENDING )  {  printf("Service stop pending...\n");  // Do not wait longer than the wait hint. A good interval is  // one-tenth of the wait hint but not less than 1 second  // and not more than 10 seconds.    dwWaitTime = ssp.dwWaitHint / 10;  if( dwWaitTime < 1000 )  dwWaitTime = 1000;  else if ( dwWaitTime > 10000 )  dwWaitTime = 10000;  Sleep( dwWaitTime );  if ( !QueryServiceStatusEx(  schService,  SC\_STATUS\_PROCESS\_INFO,  (LPBYTE)&ssp,  sizeof(SERVICE\_STATUS\_PROCESS),  &dwBytesNeeded ) )  {  printf("QueryServiceStatusEx failed (%d)\n", GetLastError());  goto stop\_cleanup;  }  if ( ssp.dwCurrentState == SERVICE\_STOPPED )  {  printf("Service stopped successfully.\n");  goto stop\_cleanup;  }  if ( GetTickCount() - dwStartTime > dwTimeout )  {  printf("Service stop timed out.\n");  goto stop\_cleanup;  }  }  // If the service is running, dependencies must be stopped first.  StopDependentServices();  // Send a stop code to the service.  if ( !ControlService(  schService,  SERVICE\_CONTROL\_STOP,  (LPSERVICE\_STATUS) &ssp ) )  {  printf( "ControlService failed (%d)\n", GetLastError() );  goto stop\_cleanup;  }  // Wait for the service to stop.  while ( ssp.dwCurrentState != SERVICE\_STOPPED )  {  Sleep( ssp.dwWaitHint );  if ( !QueryServiceStatusEx(  schService,  SC\_STATUS\_PROCESS\_INFO,  (LPBYTE)&ssp,  sizeof(SERVICE\_STATUS\_PROCESS),  &dwBytesNeeded ) )  {  printf( "QueryServiceStatusEx failed (%d)\n", GetLastError() );  goto stop\_cleanup;  }  if ( ssp.dwCurrentState == SERVICE\_STOPPED )  break;  if ( GetTickCount() - dwStartTime > dwTimeout )  {  printf( "Wait timed out\n" );  goto stop\_cleanup;  }  }  printf("Service stopped successfully\n");  stop\_cleanup:  CloseServiceHandle(schService);  CloseServiceHandle(schSCManager);  }  BOOL \_\_stdcall StopDependentServices()  {  DWORD i;  DWORD dwBytesNeeded;  DWORD dwCount;  LPENUM\_SERVICE\_STATUS lpDependencies = NULL;  ENUM\_SERVICE\_STATUS ess;  SC\_HANDLE hDepService;  SERVICE\_STATUS\_PROCESS ssp;  DWORD dwStartTime = GetTickCount();  DWORD dwTimeout = 30000; // 30-second time-out  // Pass a zero-length buffer to get the required buffer size.  if ( EnumDependentServices( schService, SERVICE\_ACTIVE,  lpDependencies, 0, &dwBytesNeeded, &dwCount ) )  {  // If the Enum call succeeds, then there are no dependent  // services, so do nothing.  return TRUE;  }  else  {  if ( GetLastError() != ERROR\_MORE\_DATA )  return FALSE; // Unexpected error  // Allocate a buffer for the dependencies.  lpDependencies = (LPENUM\_SERVICE\_STATUS) HeapAlloc(  GetProcessHeap(), HEAP\_ZERO\_MEMORY, dwBytesNeeded );    if ( !lpDependencies )  return FALSE;  \_\_try {  // Enumerate the dependencies.  if ( !EnumDependentServices( schService, SERVICE\_ACTIVE,  lpDependencies, dwBytesNeeded, &dwBytesNeeded,  &dwCount ) )  return FALSE;  for ( i = 0; i < dwCount; i++ )  {  ess = \*(lpDependencies + i);  // Open the service.  hDepService = OpenService( schSCManager,  ess.lpServiceName,  SERVICE\_STOP | SERVICE\_QUERY\_STATUS );  if ( !hDepService )  return FALSE;  \_\_try {  // Send a stop code.  if ( !ControlService( hDepService,  SERVICE\_CONTROL\_STOP,  (LPSERVICE\_STATUS) &ssp ) )  return FALSE;  // Wait for the service to stop.  while ( ssp.dwCurrentState != SERVICE\_STOPPED )  {  Sleep( ssp.dwWaitHint );  if ( !QueryServiceStatusEx(  hDepService,  SC\_STATUS\_PROCESS\_INFO,  (LPBYTE)&ssp,  sizeof(SERVICE\_STATUS\_PROCESS),  &dwBytesNeeded ) )  return FALSE;  if ( ssp.dwCurrentState == SERVICE\_STOPPED )  break;  if ( GetTickCount() - dwStartTime > dwTimeout )  return FALSE;  }  }  \_\_finally  {  // Always release the service handle.  CloseServiceHandle( hDepService );  }  }  }  \_\_finally  {  // Always free the enumeration buffer.  HeapFree( GetProcessHeap(), 0, lpDependencies );  }  }  return TRUE;  } |  生成服务 以下过程介绍如何生成服务并注册事件消息 DLL。  **生成服务并注册事件消息 DLL**   1. 使用以下步骤从 Sample.mc 生成消息 DLL：    1. **mc -U sample.mc**    2. **rc -r sample.rc**    3. **link -dll -noentry -out:sample.dll sample.res** 2. 分别从 Svc.cpp、SvcConfig.cpp 和 SvcControl.cpp 生成Svc.exe、SvcConfig.exe和SvcControl.exe。 3. **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\SvcName**创建注册表项，并将以下注册表值添加到此注册表项。   展开表   | **值** | **类型** | **说明** | | --- | --- | --- | | **EventMessageFile** = dll\_path | REG\_SZ | 仅限资源的 DLL 的路径，其中包含服务可以写入事件日志的字符串。 | | **TypesSupported** = 0x00000007 | REG\_DWORD | 一个位掩码，指定支持的事件类型。 值0x000000007表示支持所有类型。 |    正在测试服务 以下过程介绍如何测试服务。  **测试服务**   1. 在“控制面板”中，启动**“服务”**应用程序。 (在以下步骤中，在执行修改 **Services** 应用程序中信息的命令后，使用 F5 键刷新显示) 2. 运行以下命令以安装服务：   **svc 安装**  如果操作成功或出现错误消息，服务会将“已成功安装服务”写入控制台。  如果服务安装成功，服务将显示在 **服务** 应用程序中。 请注意， **“名称** ”设置为“SvcName”， **“说明** ”和 **“状态”** 为空， **“启动类型** ”设置为“手动”。   1. 运行以下命令以启动服务：   **svccontrol start SvcName**  如果操作成功，服务控制程序会写入“服务启动挂起...”。然后“服务已成功启动”到控制台。 否则，程序会将错误消息写入控制台。  如果服务成功启动， **则状态** 设置为“已启动”。 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/win32/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数中的代码由 SCM 执行。 如果发生错误，服务会将错误消息写入事件日志。 此消息包括失败的函数的名称和失败时返回的错误代码。   1. 运行以下命令以更新服务说明：   **svcconfig 描述 SvcName**  如果操作成功或出现错误消息，服务配置程序会将“服务说明已成功更新”写入控制台。  如果更新成功， **说明** 将设置为“这是测试说明”。   1. 运行以下命令以查询服务配置：   **svcconfig 查询 SvcName**  如果操作成功或出现错误消息，服务配置程序会将服务配置信息写入控制台。   1. 运行以下命令以更改服务 DACL：   **svccontrol dacl SvcName**  如果操作成功或出现错误消息，服务配置程序会将“服务 DACL 已成功更新”写入控制台。   1. 运行以下命令以禁用该服务：   **svcconfig 禁用 SvcName**  如果操作成功或出现错误消息，服务配置程序会将“已成功禁用服务”写入控制台。  如果服务已成功禁用， **则启动类型** 设置为“已禁用”。   1. 运行以下命令以启用该服务：   **svcconfig 启用 SvcName**  如果操作成功或出现错误消息，服务配置程序会将“已成功启用服务”写入控制台。  如果服务成功启用， **则启动类型** 设置为“手动”。   1. 运行以下命令以停止服务：   **svccontrol stop SvcName**  如果操作成功，服务控制程序将写入“服务停止挂起...”。然后“服务已成功停止”到控制台。 否则，程序会将错误消息写入控制台。  如果服务成功停止， **则“状态”** 为空。  如果服务无法停止，服务控制程序会将错误消息写入事件日志，其中包括失败的函数的名称和失败时返回的错误代码。   1. 运行以下命令以删除服务：   **svcconfig 删除 SvcName**  如果操作成功或出现错误消息，服务配置程序会将“已成功删除服务”写入控制台。  如果服务已成功删除，则服务将不再显示在 **服务** 应用程序中。 (请注意，如果尝试删除未停止的服务，操作会成功，但 **启动类型** 设置为“已禁用”，并且服务条目将在系统重启或使用任务管理器终止服务时删除。) |

## StartServiceCtrlDispatcher函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将服务进程的main线程连接到服务控制管理器，这会使线程成为调用进程的服务控制调度程序线程。 语法 C++复制  BOOL StartServiceCtrlDispatcherA(  [in] const SERVICE\_TABLE\_ENTRYA \*lpServiceStartTable  ); 参数 [in] lpServiceStartTable  指向 [SERVICE\_TABLE\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/ns-winsvc-service_table_entrya) 结构的数组的指针，其中包含可在调用过程中执行的每个服务的一个条目。 表中最后一个条目的成员必须具有 NULL 值才能指定表的末尾。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。  服务控制管理器可以设置以下错误代码。 其他错误代码可由服务控制管理器调用的注册表函数设置。  展开表   |  |  | | --- | --- | | **返回代码** | **说明** | | **ERROR\_FAILED\_SERVICE\_CONTROLLER\_CONNECT** | 如果程序以控制台应用程序而不是服务的形式运行，则返回此错误。  如果程序将作为控制台应用程序运行以用于调试目的，请对其进行构造，以便在返回此错误时不调用特定于服务的代码。 | | **ERROR\_INVALID\_DATA** | 指定的调度表包含格式不当的条目。 | | **ERROR\_SERVICE\_ALREADY\_RUNNING** | 该过程已调用 [StartServiceCtrlDispatcher](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-startservicectrldispatchera)。 每个进程只能调用 **StartServiceCtrlDispatcher** 一次。 |  注解 当服务控制管理器启动服务进程时，它会等待进程调用 **StartServiceCtrlDispatcher** 函数。 服务进程的main线程应在) 30 秒内启动 (后尽快进行此调用。 如果 **StartServiceCtrlDispatcher** 成功，它将调用线程连接到服务控制管理器，并且不会返回，直到进程中的所有正在运行的服务都进入SERVICE\_STOPPED状态。 服务控制管理器使用此连接将控制和服务启动请求发送到服务进程的main线程。 main线程充当调度程序，方法是调用相应的 [HandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function_ex) 函数来处理控制请求，或者在启动新服务时创建新线程以执行相应的 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数。  *lpServiceTable* 参数包含可在调用过程中运行的每个服务的条目。 每个条目指定该服务的 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数。 对于SERVICE\_WIN32\_SHARE\_PROCESS服务，每个条目都必须包含服务的名称。 此名称是安装服务时 [由 CreateService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-createservicea) 函数指定的服务名称。 对于SERVICE\_WIN32\_OWN\_PROCESS服务，将忽略表条目中的服务名称。  如果服务在其自己的进程中运行，则服务进程的main线程应立即调用 **StartServiceCtrlDispatcher**。 启动服务时，所有初始化任务都在服务的 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数中完成。  如果多个服务共享一个进程，并且需要在调用任何 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数之前完成一些常见的进程范围的初始化，则main线程可以在调用 **StartServiceCtrlDispatcher** 之前执行该工作，只要花费的时间少于 30 秒。 否则，必须创建另一个线程来执行进程范围的初始化，而main线程调用 **StartServiceCtrlDispatcher** 并成为服务控制调度程序。 任何特定于服务的初始化仍应在单个服务main函数中完成。  服务不应尝试直接显示用户界面。 有关详细信息，请参阅 [Interactive Services](https://learn.microsoft.com/zh-cn/windows/desktop/Services/interactive-services)。 示例 有关示例，请参阅 [编写服务程序的 Main 函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/writing-a-service-program-s-main-function)。  **备注**  winsvc.h 标头将 StartServiceCtrlDispatcher 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winsvc.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [ControlService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-controlservice)  [HandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function_ex)  [SERVICE\_TABLE\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/ns-winsvc-service_table_entrya)  [服务入口点](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-entry-point)  [服务函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-functions)  [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) |

## ServiceCtrlHandler函数

|  |
| --- |
| 每个服务都有一个控制处理程序， [**即 Handler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nc-winsvc-lphandler_function) 函数，当服务进程收到来自服务控制程序的控制请求时，控制调度程序将调用该处理程序。 因此，此函数在控件调度程序的上下文中执行。 有关示例，请参阅 [编写控件处理程序函数](https://learn.microsoft.com/zh-cn/windows/win32/services/writing-a-control-handler-function)。  服务调用 [**RegisterServiceCtrlHandler**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-registerservicectrlhandlera) 或 [**RegisterServiceCtrlHandlerEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-registerservicectrlhandlerexa) 函数来注册其服务控制处理程序函数。  调用服务控制处理程序时，服务必须调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 函数，以便仅当处理控制代码导致服务状态更改时，才向 SCM 报告其状态。 如果处理控制代码不会导致服务状态更改，则无需调用 **SetServiceStatus**。  服务控制程序可以使用 [**ControlService**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-controlservice) 函数发送控制请求。 所有服务都必须接受并处理 **SERVICE\_CONTROL\_INTERROGATE** 控制代码。 可以通过调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 启用或禁用接受其他控制代码。 若要接收 **SERVICE\_CONTROL\_DEVICEEVENT** 控制代码，必须调用 [**RegisterDeviceNotification**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerdevicenotificationa) 函数。 服务还可以处理用户定义的其他控制代码。  如果服务接受 **SERVICE\_CONTROL\_STOP** 控制代码，则必须在收到后停止，进入 **SERVICE\_STOP\_PENDING** 或 **SERVICE\_STOPPED** 状态。 SCM 发送此控制代码后，不会发送其他控制代码。  **Windowsxp：** 如果服务返回 **NO\_ERROR** 并继续运行，它将继续接收控制代码。 从 Windows Server 2003 和 Windows XP service Pack 2 (SP2) 开始，此行为已更改。  控制处理程序必须在 30 秒内返回，否则 SCM 返回错误。 如果服务在执行控件处理程序时必须执行冗长处理，则它应创建一个辅助线程来执行冗长的处理，然后从控件处理程序返回。 这可以防止服务将控制调度程序绑起来。 例如，处理某个服务需要很长时间的停止请求时，请创建另一个线程来处理停止进程。 控件处理程序应仅使用**SERVICE\_STOP\_PENDING**消息调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 并返回。  当用户关闭系统时，使用**SERVICE\_ACCEPT\_PRESHUTDOWN**控制代码调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 的所有控件处理程序都会接收**SERVICE\_CONTROL\_PRESHUTDOWN**控件代码。 服务控制管理器将等待服务停止或指定的预关闭超时值过期， (可以使用 [**ChangeServiceConfig2**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-changeserviceconfig2a) 函数) 设置此值。 此控制代码应仅在特殊情况下使用，因为处理此通知的服务会阻止系统关闭，直到服务停止或预关闭超时间隔过期。  完成预暂停通知后，使用**SERVICE\_ACCEPT\_SHUTDOWN**控件代码调用 [**SetServiceStatus**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winsvc/nf-winsvc-setservicestatus) 的所有控件处理程序都会接收**SERVICE\_CONTROL\_SHUTDOWN**控件代码。 按照它们在已安装服务的数据库中出现的顺序通知它们。 默认情况下，服务在系统关闭之前有大约 20 秒的时间执行清理任务。 此时间到期后，无论服务关闭是否完成，系统关闭都会继续。 请注意，如果系统保持关闭状态 (未重启或关闭) ，则服务将继续运行。  如果服务需要更多时间进行清理，它会发送 **STOP\_PENDING** 状态消息以及等待提示，以便服务控制器知道在向系统报告服务关闭已完成之前等待的时间。 但是，为了防止服务停止关闭，服务控制器的等待时间有限制。 如果通过服务管理单元关闭服务，则限制为 125 秒或 125，000 毫秒。 如果操作系统正在重新启动，则会在以下注册表项) **WaitToKillServiceTimeout** 值 (指定时间限制（以毫秒为单位）：  **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control**  **重要**  服务不应尝试通过修改此值来增加时间限制。 如果需要手动设置 **WaitToKillServiceTimeout** ，该值应以毫秒为单位。  客户需要快速关闭操作系统。 例如，如果使用 UPS 电源运行的计算机在 UPS 断电之前无法完成关闭，则数据可能会丢失。 因此，服务应尽快完成其清理任务。 最好通过定期保存数据、跟踪保存到磁盘的数据以及仅在关闭时保存未保存的数据，来最大程度地减少未保存的数据。 由于计算机正在关闭，因此不要花时间释放分配的内存或其他系统资源。 如果需要通知服务器你正在退出，请尽量减少等待回复所用的时间，因为网络问题可能会延迟服务的关闭。  请注意，在服务关闭期间，默认情况下，SCM 不会考虑依赖项。 SCM 枚举正在运行的服务的列表，并发送 **SERVICE\_CONTROL\_SHUTDOWN** 命令。 因此，服务可能会失败，因为它所依赖的另一个服务已停止。  若要手动设置服务的关闭顺序，请创建一个多字符串注册表值，该值包含服务名称（按应关闭的顺序），并将其分配给控制键的 **PreshutdownOrder** 值，如下所示：  **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\PreshutdownOrder="Shutdown Order”**  若要设置应用程序中依赖服务的关闭顺序，请使用 [**SetProcessShutdownParameters**](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-setprocessshutdownparameters) 函数。 SCM 使用此函数为其处理程序0x1E0优先级。 当调用其控制处理程序时，SCM 会发送 **SERVICE\_CONTROL\_SHUTDOWN** 通知，并等待服务退出，然后从其控制处理程序返回。 |

## 服务控制常量

|  |
| --- |
| #define SERVICE\_CONTROL\_STOP 0x00000001  #define SERVICE\_CONTROL\_PAUSE 0x00000002  #define SERVICE\_CONTROL\_CONTINUE 0x00000003  #define SERVICE\_CONTROL\_INTERROGATE 0x00000004  #define SERVICE\_CONTROL\_SHUTDOWN 0x00000005  #define SERVICE\_CONTROL\_PARAMCHANGE 0x00000006  #define SERVICE\_CONTROL\_NETBINDADD 0x00000007  #define SERVICE\_CONTROL\_NETBINDREMOVE 0x00000008  #define SERVICE\_CONTROL\_NETBINDENABLE 0x00000009  #define SERVICE\_CONTROL\_NETBINDDISABLE 0x0000000A  #define SERVICE\_CONTROL\_DEVICEEVENT 0x0000000B  #define SERVICE\_CONTROL\_HARDWAREPROFILECHANGE 0x0000000C  #define SERVICE\_CONTROL\_POWEREVENT 0x0000000D  #define SERVICE\_CONTROL\_SESSIONCHANGE 0x0000000E  #define SERVICE\_CONTROL\_PRESHUTDOWN 0x0000000F  #define SERVICE\_CONTROL\_TIMECHANGE 0x00000010  #define SERVICE\_CONTROL\_TRIGGEREVENT 0x00000020 |

## RegisterServiceCtrlHandler函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 注册用于处理服务控制请求的函数。  此函数已被 [RegisterServiceCtrlHandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-registerservicectrlhandlerexa) 函数取代。 服务可以使用任一函数，但新函数支持用户定义的上下文数据，新的处理程序函数支持其他扩展控制代码。 语法 C++复制  SERVICE\_STATUS\_HANDLE RegisterServiceCtrlHandlerA(  [in] LPCSTR lpServiceName,  [in] LPHANDLER\_FUNCTION lpHandlerProc  ); 参数 [in] lpServiceName  由调用线程运行的服务的名称。 这是创建服务时在 [CreateService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-createservicea) 函数中指定的服务控制程序的服务名称。  如果服务类型SERVICE\_WIN32\_OWN\_PROCESS，则函数不会验证指定名称是否有效，因为进程中只有一个已注册的服务。  [in] lpHandlerProc  指向要注册的处理程序函数的指针。 有关详细信息，请参阅 [处理程序](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function)。 返回值 如果函数成功，则返回值为服务状态句柄。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。  服务控制管理器可以设置以下错误代码。  展开表   |  |  | | --- | --- | | **返回代码** | **说明** | | **ERROR\_NOT\_ENOUGH\_MEMORY** | 内存不足，无法将 ANSI 字符串参数转换为 Unicode。 Unicode 字符串参数不会发生此错误。 | | **ERROR\_SERVICE\_NOT\_IN\_EXE** | 进程调用 [StartServiceCtrlDispatcher](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-startservicectrldispatchera) 函数时，未正确指定服务条目。 |  注解 新服务的 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数应立即调用 **RegisterServiceCtrlHandler** 函数，以便向控件调度程序注册控件处理程序函数。 这使控件调度程序能够在收到对此服务的控件请求时调用指定的函数。 有关可能的控制代码的列表，请参阅 [处理程序](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function)。 调用进程的线程可以使用此函数返回的服务状态句柄来标识对 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) 函数的后续调用中的服务。  **RegisterServiceCtrlHandler** 函数必须在第一次 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) 调用之前调用，因为 **RegisterServiceCtrlHandler** 返回服务状态句柄供调用方使用，因此其他服务无法无意中设置此服务状态。 此外，控件处理程序必须在服务指定它通过 **SetServiceStatus** 函数接受的控件时接收控件请求。  使用控件请求调用控件处理程序函数时，服务必须调用 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) ，以便仅当服务状态发生更改时（例如，服务正在处理停止或关闭控件时）向服务控制管理器报告状态。 如果服务状态未更改，则服务不应向服务控制管理器报告状态。  无需关闭服务状态句柄。 示例 有关示例，请参阅 [编写 ServiceMain 函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/writing-a-servicemain-function)。  **备注**  winsvc.h 标头将 RegisterServiceCtrlHandler 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winsvc.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [CreateService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-createservicea)  [处理程序](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function)  [RegisterServiceCtrlHandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-registerservicectrlhandlerexa)  [服务控制处理程序函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-control-handler-function)  [服务函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-functions)  [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona)  [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) |

## RegisterServiceCtrlHandlerEx函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 注册用于处理扩展服务控制请求的函数。 语法 C++复制  SERVICE\_STATUS\_HANDLE RegisterServiceCtrlHandlerExA(  [in] LPCSTR lpServiceName,  [in] LPHANDLER\_FUNCTION\_EX lpHandlerProc,  [in, optional] LPVOID lpContext  ); 参数 [in] lpServiceName  由调用线程运行的服务的名称。 这是创建服务时在 [CreateService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-createservicea) 函数中指定的服务控制程序的服务名称。  [in] lpHandlerProc  指向要注册的处理程序函数的指针。 有关详细信息，请参阅 [HandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function_ex)。  [in, optional] lpContext  任何用户定义的数据。 此参数传递给处理程序函数，当多个服务共享一个进程时，可帮助识别服务。 返回值 如果函数成功，则返回值为服务状态句柄。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。  服务控制管理器可以设置以下错误代码。  展开表   |  |  | | --- | --- | | **返回代码** | **说明** | | **ERROR\_NOT\_ENOUGH\_MEMORY** | 内存不足，无法将 ANSI 字符串参数转换为 Unicode。 Unicode 字符串参数不会发生此错误。 | | **ERROR\_SERVICE\_NOT\_IN\_EXE** | 进程调用 [StartServiceCtrlDispatcher](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-startservicectrldispatchera) 函数时，未正确指定服务条目。 |  注解 新服务的 [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona) 函数应立即调用 **RegisterServiceCtrlHandlerEx** 函数，以便向控件调度程序注册控件处理程序函数。 这使控件调度程序能够在收到对此服务的控件请求时调用指定的函数。 有关可能的控制代码的列表，请参阅 [HandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function_ex)。 调用进程的线程可以使用此函数返回的服务状态句柄来标识对 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) 函数的后续调用中的服务。  **RegisterServiceCtrlHandlerEx** 函数必须在第一次 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) 调用之前调用，因为 **RegisterServiceCtrlHandlerEx** 返回调用方要使用的服务状态句柄，因此其他服务无法无意中设置此服务状态。 此外，控件处理程序必须在服务指定它通过 **SetServiceStatus** 函数接受的控件时接收控件请求。  使用控件请求调用控件处理程序函数时，服务必须调用 [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) ，以便仅当服务状态发生更改时（例如，服务正在处理停止或关闭控件时）向服务控制管理器报告状态。 如果服务状态未更改，则服务不应向服务控制管理器报告状态。  无需关闭服务状态句柄。  **备注**  winsvc.h 标头将 RegisterServiceCtrlHandlerEx 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winsvc.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [CreateService](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-createservicea)  [HandlerEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lphandler_function_ex)  [服务控制处理程序函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-control-handler-function)  [服务函数](https://learn.microsoft.com/zh-cn/windows/desktop/Services/service-functions)  [ServiceMain](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nc-winsvc-lpservice_main_functiona)  [SetServiceStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsvc/nf-winsvc-setservicestatus) |

## 服务类型常量

|  |
| --- |
| #define SERVICE\_KERNEL\_DRIVER 0x00000001  #define SERVICE\_FILE\_SYSTEM\_DRIVER 0x00000002  #define SERVICE\_ADAPTER 0x00000004  #define SERVICE\_RECOGNIZER\_DRIVER 0x00000008  #define SERVICE\_DRIVER (SERVICE\_KERNEL\_DRIVER | \  SERVICE\_FILE\_SYSTEM\_DRIVER | \  SERVICE\_RECOGNIZER\_DRIVER)  #define SERVICE\_WIN32\_OWN\_PROCESS 0x00000010  #define SERVICE\_WIN32\_SHARE\_PROCESS 0x00000020  #define SERVICE\_WIN32 (SERVICE\_WIN32\_OWN\_PROCESS | \  SERVICE\_WIN32\_SHARE\_PROCESS)  #define SERVICE\_INTERACTIVE\_PROCESS 0x00000100  #define SERVICE\_TYPE\_ALL (SERVICE\_WIN32 | \  SERVICE\_ADAPTER | \  SERVICE\_DRIVER | \  SERVICE\_INTERACTIVE\_PROCESS) |

## 可以接受的服务常量

|  |
| --- |
| //  // Controls Accepted (Bit Mask)  //  #define SERVICE\_ACCEPT\_STOP 0x00000001  #define SERVICE\_ACCEPT\_PAUSE\_CONTINUE 0x00000002  #define SERVICE\_ACCEPT\_SHUTDOWN 0x00000004  #define SERVICE\_ACCEPT\_PARAMCHANGE 0x00000008  #define SERVICE\_ACCEPT\_NETBINDCHANGE 0x00000010  #define SERVICE\_ACCEPT\_HARDWAREPROFILECHANGE 0x00000020  #define SERVICE\_ACCEPT\_POWEREVENT 0x00000040  #define SERVICE\_ACCEPT\_SESSIONCHANGE 0x00000080  #define SERVICE\_ACCEPT\_PRESHUTDOWN 0x00000100  #define SERVICE\_ACCEPT\_TIMECHANGE 0x00000200  #define SERVICE\_ACCEPT\_TRIGGEREVENT 0x00000400 |

## 服务状态常量

|  |
| --- |
| //  // Service State -- for CurrentState  //  #define SERVICE\_STOPPED 0x00000001  #define SERVICE\_START\_PENDING 0x00000002  #define SERVICE\_STOP\_PENDING 0x00000003  #define SERVICE\_RUNNING 0x00000004  #define SERVICE\_CONTINUE\_PENDING 0x00000005  #define SERVICE\_PAUSE\_PENDING 0x00000006  #define SERVICE\_PAUSED 0x00000007 |

## GlobalmemoryStatus函数用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C++复制  void GlobalMemoryStatus(  [out] LPMEMORYSTATUS lpBuffer  ); 参数 [out] lpBuffer  指向 [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus) 结构的指针。 **GlobalMemoryStatus** 函数将有关当前内存可用性的信息存储到此结构中。 返回值 无 备注 在内存超过 4 GB 的计算机上， **GlobalMemoryStatus** 函数可能会返回错误的信息，报告值 –1 以指示溢出。 因此，应用程序应改用 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数。  在内存超过 2 GB 且内存小于 4 GB 的 Intel x86 计算机上，**GlobalMemoryStatus** 函数将始终在 [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus) 结构的 **dwTotalPhys** 成员中返回 2 GB。 同样，如果总可用内存在 2 到 4 GB 之间，则 **MEMORYSTATUS** 结构的 **dwAvailPhys** 成员将向下舍入为 2 GB。 如果使用 **/LARGEADDRESSAWARE** 链接器选项链接可执行文件，则 **GlobalMemoryStatus** 函数将在两个成员中返回正确的物理内存量。  **GlobalMemoryStatus** 函数返回的信息是可变的。 不能保证对此函数的两次连续调用将返回相同的信息。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex)  [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85))  [虚拟地址空间和物理存储](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-address-space-and-physical-storage) |

## 写一个服务程序最重要的是要写ServiceMain函数和ServiceCtrlHandler函数，一个服务程序不能直接双击运行，需要安装为服务才能运行。服务程序是命令行程序，不要界面。

# 演练

## 1.新建一个文件夹Lesson50-service，然后在里面新建一叫做servicedemo的常规空项目，然后新建一个testservice.c文件,一个服务程序可以有任意多个服务但是最后一个一定要是NULL，NULL。

|  |
| --- |
|  |

## 2.先写程序的入口函数main，然后在面函数里面注册ServiceMain函数

|  |
| --- |
|  |

## 3.然后我们需要在ServiceMain里面写需要做的事情的代码，例如可以在里面写一个循环，并且我们需要定义一个服务状态全局变量和一个服务状态句柄变量，然后在ServiceMain里面注册一个服务控制处理函数，用服务状态句柄来接收结果，需要先声明一下这个函数，一个服务的启动、暂停、停止、继续等等的操作都需要在服务控制函数里面完成

|  |
| --- |
|  |

## 4.我们来写ServiceMain函数,先在里面初始化服务的初始状态,然后注册服务控制处理函数.无论注册成功与否,都要写日志,然后才真正设置服务状态,我们在服务运行的时候每个10秒钟检查一下电脑可用的物理内存大小，然后写入日志

|  |
| --- |
| void WINAPI ServiceMain(int argc,char\*\* argv)  {  //内存操作需要的变量  MEMORYSTATUS memstatus;  char str[100];  int availMb = 0;//可用内存空间以MB为单位  //初始化服务状态属性  sStatus.dwServiceType = SERVICE\_WIN32;  sStatus.dwCurrentState = SERVICE\_START\_PENDING;//等待状态，因为服务还没有运行  sStatus.dwControlsAccepted = SERVICE\_ACCEPT\_SHUTDOWN|SERVICE\_ACCEPT\_STOP;//在等待状态下是允许关机和停止的  sStatus.dwWin32ExitCode = 0;//Win32退出码  sStatus.dwServiceSpecificExitCode = 0;//指定服务的退出码  sStatus.dwCheckPoint = 0;//一般用的很少设置为0即可  sStatus.dwWaitHint = 0; //一般用的很少设置为0即可  //注册服务控制函数  ssHandle = RegisterServiceCtrlHandler("Test Service",(LPHANDLER\_FUNCTION)ServiceCtrlHandler);  //如果注册失败，写到日志里面  if(ssHandle == 0)//注册失败返回0  {  WriteLog("Register Service Control Handler Failed!!");  }  //如果注册成功，也需要写日志  WriteLog("Register Service Control Handler Succeeded!!");  //注册成功后需要修改服务状态为正在运行  sStatus.dwCurrentState = SERVICE\_RUNNING;  //真正设置服务状态到服务控制句柄中，也就是相当于把状态公司服务控制函数  SetServiceStatus(ssHandle,&sStatus);  running = TRUE;    memset(str,0,100);  while(running)  {  //我们想在这个服务里面去检查电脑可用内存的大小  GlobalMemoryStatus(&memstatus);//获取内存当前状态  availMb = memstatus.dwAvailPhys/1024/1024;//可用物理内存  sprintf\_s(str,100,"可用物理内存为:%dMB\n",availMb);  WriteLog(str);  Sleep(DURATION);//让他睡眠一会，大概10秒钟  }  //循环结束也就是服务结束，此时也要写日志  WriteLog("服务停止了...");  } |

## 5.在ServiceMain函数里面用到了一个自定义函数WriteLog，我们来实现这个函数

|  |
| --- |
|  |
|  |

## 6.然后在控制函数里面停止服务的地方把running赋值为false，服务就会停止，然后需要设置服务当前状态属性为服务停止状态，然后需要在函数退出之前设置到控制函数句柄中

|  |
| --- |
|  |

## 7.编译一下项目,发现成功生成了项目

|  |
| --- |
|  |

### 注意:这是一个服务程序,不是普通的应用程序,不能双击运行,你在cmd窗口运行,不会有任何输出,它需要安装为服务,然后才能运行

## 8.为了方便安装,我们把servicedemo.exe拷贝粘贴到e:\servicedemo文件夹里面

|  |
| --- |
|  |

## 9.安装服务可以使用windows提供的sc.exe工具，我们在e:根目录里面打开一个命令行窗口

|  |
| --- |
|  |

## 10.安装服务的命令如下格式

|  |
| --- |
|  |

### 我的是这样子写的

|  |
| --- |
|  |

## 11.安装服务成功后，在服务面板里面可以看得到

|  |
| --- |
|  |

## 12，此时你可以双击这个服务打开控制窗口然后启动它，也可以把他设置为自动启动，还可以用sc在命令行里面启动它

|  |
| --- |
|  |

## 13.如果服务启动失败，你可以使用下面的命令来查看服务状态

|  |
| --- |
|  |

## 14.删除服务，使用这个命令，删除后需要刷新服务面板界面

|  |
| --- |
|  |

## 15.停止服务可以使用下面的命令

|  |
| --- |
|  |

# 扩展

## 这个日志系统不够完善，因为作为日志系统其实需要有时间信息的，我们以后需要添加这个功能

# 这一节的学习到此为止，完整代码如下

## testservice.c

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  //定义一个睡眠时间常量  #define DURATION 10000  #define FILEPATH "e:\\servicelog.txt"  SERVICE\_STATUS sStatus; //服务状态  SERVICE\_STATUS\_HANDLE ssHandle;  BOOL running = TRUE;//运行标记  void WINAPI ServiceMain(int argc,char\*\* argv);//声明服务入口函数  void WINAPI ServiceCtrlHandler(DWORD);//声明服务控制函数  int WriteLog(char \*str);//写日志函数  void main(void)  {  SERVICE\_TABLE\_ENTRY srvTable[2];//这个数组其实可以很大，也就是说可以有很多服务，但是最后一个必须是NULL，NULL，也就是说这里2其实代表只有一个服务，因为最后一个是NULL  srvTable[0].lpServiceName = "Test Service";//给服务取个名字  srvTable[0].lpServiceProc =(LPSERVICE\_MAIN\_FUNCTION)ServiceMain ;//指定服务的口函数入  srvTable[1].lpServiceName = NULL;  srvTable[1].lpServiceProc = NULL;  //注册服务入口函数  StartServiceCtrlDispatcher(srvTable);//注册成功后服务分派器就会给服务创建一个线程  system("pause");  }  void WINAPI ServiceMain(int argc,char\*\* argv)  {  //内存操作需要的变量  MEMORYSTATUS memstatus;  char str[100];  int availMb = 0;//可用内存空间以MB为单位  //初始化服务状态属性  sStatus.dwServiceType = SERVICE\_WIN32;  sStatus.dwCurrentState = SERVICE\_START\_PENDING;//等待状态，因为服务还没有运行  sStatus.dwControlsAccepted = SERVICE\_ACCEPT\_SHUTDOWN|SERVICE\_ACCEPT\_STOP;//在等待状态下是允许关机和停止的  sStatus.dwWin32ExitCode = 0;//Win32退出码  sStatus.dwServiceSpecificExitCode = 0;//指定服务的退出码  sStatus.dwCheckPoint = 0;//一般用的很少设置为0即可  sStatus.dwWaitHint = 0; //一般用的很少设置为0即可  //注册服务控制函数  ssHandle = RegisterServiceCtrlHandler("Test Service",(LPHANDLER\_FUNCTION)ServiceCtrlHandler);  //如果注册失败，写到日志里面  if(ssHandle == 0)//注册失败返回0  {  WriteLog("Register Service Control Handler Failed!!");  }  //如果注册成功，也需要写日志  WriteLog("Register Service Control Handler Succeeded!!");  //注册成功后需要修改服务状态为正在运行  sStatus.dwCurrentState = SERVICE\_RUNNING;  //真正设置服务状态到服务控制句柄中，也就是相当于把状态公司服务控制函数  SetServiceStatus(ssHandle,&sStatus);  running = TRUE;    memset(str,0,100);  while(running)  {  //我们想在这个服务里面去检查电脑可用内存的大小  GlobalMemoryStatus(&memstatus);//获取内存当前状态  availMb = memstatus.dwAvailPhys/1024/1024;//可用物理内存  sprintf\_s(str,100,"可用物理内存为:%dMB\n",availMb);  WriteLog(str);  Sleep(DURATION);//让他睡眠一会，大概10秒钟  }  //循环结束也就是服务结束，此时也要写日志  WriteLog("服务停止了...");  }  //服务控制程序，控制服务的启动、暂停、停止、继续等等的操作  void WINAPI ServiceCtrlHandler(DWORD dwCtrl)  {  switch(dwCtrl)//注意这里的分支数目应该和上面dwControlsAccepted属性设置的操作数量一致  {  case SERVICE\_CONTROL\_STOP:  running= FALSE;  sStatus.dwCurrentState = SERVICE\_STOPPED;  break;  case SERVICE\_CONTROL\_SHUTDOWN:  running= FALSE;  break;  default:  break;  }  //报告服务状态和控制函数句柄  SetServiceStatus(ssHandle,&sStatus);  }  int WriteLog(char \*str)  {  FILE \*pfile;  fopen\_s(&pfile,FILEPATH,"a+");//打开文件，文件名以常量的形式定义好了  if(pfile == NULL)  {  printf("打开文件失败\n");  return -1;  }  fprintf\_s(pfile,"%s\n",str);  fclose(pfile);  return 0;  } |