**[笔记]Windows系统编程《九》Windows服务编程**

**前言**

**一、Windows服务的概念和管理**

**1.1 管理windows服务**

打开Windows服务管理器窗口

ctrl+R + 命令：

services.msc

**sc.exe管理Windows服务**

sc <server>[ command] [service name] <option1><option2> ...

**1.2 服务控制器**

服务控制器（Service Control Manager，SCM）是Windows NT家族操作系统中的一个系统进程，它可以启动、停止Windows服务，并与 Windows服务交流。

服务控制器对应的可执行文件是%SystemRoot%\services.exe，它以 Windows控制台程序的形式运行。在系统启动时，由Wininit进程加载。

SCM 的主函数是SvcCtrlMain()，它将加载被配置为自动启动的服务列表。

Windows服务的信息保存在注册表的如下位置中：

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services

**二、Windows服务编程**

在Visual C++程序中，可以对Windows服务进行控制，例如：

   创建服务

   启动服务

   停止服务

   查询服务的状态

   枚举服务列表等

**2.1 与SCM建立连接**

要在程序中访问Windows服务，首先需要与服务控制器( SCM)建立连接。

**OpenSCManager**

SC\_HANDLE OpenSCManager(

[in, optional] LPCSTR lpMachineName,

[in, optional] LPCSTR lpDatabaseName,

[in] DWORD dwDesiredAccess

);

lpMachineName：目标计算机的名称。如果指针是NULL或空字符串,函数连接到本地计算机上的服务控制管理器。

lpDatabaseName：服务控制管理器数据库的名称。这个参数应该设置为SERVICES\_ACTIVE\_DATABASE。如果是NULL,

SERVICES\_ACTIVE\_DATABASE数据库默认打开。

dwDesiredAccess：对服务的访问控制管理。

相关权限查看访问

**2.2 创建服务**

CreateService

调用CreateService()函数可以创建一个服务对象，并将其添加到数据库中，函数原型如下：

SC\_HANDLE CreateService(

[in] SC\_HANDLE hSCManager,

[in] LPCSTR lpServiceName,

[in, optional] LPCSTR lpDisplayName,

[in] DWORD dwDesiredAccess,

[in] DWORD dwServiceType,

[in] DWORD dwStartType,

[in] DWORD dwErrorControl,

[in, optional] LPCSTR lpBinaryPathName,

[in, optional] LPCSTR lpLoadOrderGroup,

[out, optional] LPDWORD lpdwTagId,

[in, optional] LPCSTR lpDependencies,

[in, optional] LPCSTR lpServiceStartName,

[in, optional] LPCSTR lpPassword

);

**2.3 打开服务/关闭服务**

在对服务进行操作之前 需要调用OpenService函数打开指定服务对象，并返回服务句柄。

SC\_HANDLE OpenService(

[in] SC\_HANDLE hSCManager,

[in] LPCSTR lpServiceName,

[in] DWORD dwDesiredAccess

);

hSCManager：调用OpenSCManager()函数返回的服务控制器句柄。

lpServiceName：要打开的服务名

dwDesiredAccess：对服务的访问权限。

返回值：

关闭服务

BOOL CloseServiceHandle(

[in] SC\_HANDLE hSCObject

);

hSCObject：关闭的服务句柄。

**2.4 枚举服务列表**

枚举服务控制器数据库中服务列表。

BOOL EnumServicesStatusEx(

[in] SC\_HANDLE hSCManager,

[in] SC\_ENUM\_TYPE InfoLevel,

[in] DWORD dwServiceType,

[in] DWORD dwServiceState,

[out, optional] LPBYTE lpServices,

[in] DWORD cbBufSize,

[out] LPDWORD pcbBytesNeeded,

[out] LPDWORD lpServicesReturned,

[in, out, optional] LPDWORD lpResumeHandle,

[in, optional] LPCSTR pszGroupName

);

hSCManager：调用OpenSCManager()函数返回的服务控制器句柄。

InfoLevel：指定返回得服务属性。

dwServiceType：指定美剧服务的类型。

dwServiceStatus：指定枚举服务的状态。

lpServices：用于接收服务信息的缓冲区。

cbBufferSize：lpService指定缓冲区大小。

pcbBytesNeeded：如果缓冲区太小，则指定剩余服务记录的数量。

**实例：枚举系统服务并打印**

#include <Windows.h>

#include <stdio.h>

int main(int argc, char\* argv[])

{

LONG lRet = 0;

BOOL bRet = FALSE;

SC\_HANDLE hSCM = NULL; // 服务数据库句柄

char\* pBuf = NULL; // 缓冲区指针

DWORD dwBufSize = 0; // 传入的缓冲长度

DWORD dwBufNeed = 0; // 需要的缓冲长度

DWORD dwNumberOfService = 0; // 返回的服务个数

ENUM\_SERVICE\_STATUS\_PROCESS\* pServiceInfo = NULL; // 服务信息

// 建立了一个到服务控制管理器的连接，并打开指定的数据库

hSCM = OpenSCManager(NULL, NULL, SC\_MANAGER\_ENUMERATE\_SERVICE | SC\_MANAGER\_CONNECT);

if (NULL == hSCM)

{

printf("OpenSCManager error.\n");

return -1;

}

// 获取需要的缓冲区大小

if (EnumServicesStatusEx(hSCM, SC\_ENUM\_PROCESS\_INFO, SERVICE\_WIN32, SERVICE\_STATE\_ALL,

NULL, dwBufSize, &dwBufNeed, &dwNumberOfService, NULL, NULL))

{

printf("get buffer failed! \n");

return -1;

}

// 多设置存放1个服务信息的长度

dwBufSize = dwBufNeed + sizeof(ENUM\_SERVICE\_STATUS\_PROCESS);

pBuf = (char\*)malloc(dwBufSize);

if (NULL == pBuf)

{

printf("malloc error.\n");

return -1;

}

memset(pBuf, 0, dwBufSize);

// 获取服务信息

bRet = EnumServicesStatusEx(hSCM, SC\_ENUM\_PROCESS\_INFO, SERVICE\_WIN32, SERVICE\_STATE\_ALL,

(LPBYTE)pBuf, dwBufSize, &dwBufNeed, &dwNumberOfService, NULL, NULL);

if (bRet == FALSE)

{

printf("EnumServicesStatusEx error.\n");

::CloseServiceHandle(hSCM);

free(pBuf);

return -1;

}

// 关闭打开的服务句柄

bRet = ::CloseServiceHandle(hSCM);

if (bRet == FALSE)

{

printf("CloseServiceHandle error.\n");

}

printf("Service Num:%d\n", dwNumberOfService);

pServiceInfo = (LPENUM\_SERVICE\_STATUS\_PROCESS)pBuf;

// 打印取得的服务信息

for (unsigned int i = 0; i < dwNumberOfService; i++)

{

printf("----------%d----------\n", i);

printf("DisplayName \t\t : %s \n", pServiceInfo[i].lpDisplayName);

printf("ServiceName \t\t : %s \n", pServiceInfo[i].lpServiceName);

printf("ServiceType \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwServiceType);

printf("CurrentState \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwCurrentState);

printf("ControlsAccepted \t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwControlsAccepted);

printf("Win32ExitCode \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwWin32ExitCode);

printf("ServiceSpecificExitCode : %d \n", pServiceInfo[i].ServiceStatusProcess.dwServiceSpecificExitCode);

printf("CheckPoint \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwCheckPoint);

printf("WaitHint \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwWaitHint);

printf("Process Id \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwProcessId);

printf("ServiceFlags \t\t : %d \n", pServiceInfo[i].ServiceStatusProcess.dwServiceFlags);

}

free(pBuf);

system("PAUSE");

return 0;

}

**2.5 启动服务**

**2.6 停止服务**

**2.7 查询服务的状态**

**2.8 修改服务的配置参数**

**2.9 完整创建Windows服务**

<https://docs.microsoft.com/en-us/windows/win32/services/svc-cpp>

#include <windows.h>

#include <tchar.h>

#include <strsafe.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(LPCTSTR);

#define SVC\_ERROR -11

//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;

// }

// TCHAR svcname[] = SVCNAME;

// // 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"));

// }

// return 0;

//}

VOID SvcInstall()

{

SC\_HANDLE schSCManager;

SC\_HANDLE schService;

TCHAR szPath[MAX\_PATH];

if (!GetModuleFileName(NULL, szPath, MAX\_PATH))

{

printf("Cannot install service (%d)\n", GetLastError());

return;

}

// 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);

}

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);

}

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;

}

}

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);

}

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;

}

}

VOID SvcReportEvent(LPCTSTR 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);

}

}

**三、开发Windows服务程序**

**前言**

ATL，Active Template Library活动(动态)模板库，是一种微软程序库，支持利用C++语言编写ASP代码以及其它ActiveX程序。通过活动模板库，可以建立COM组件，然后通过ASP页面中的脚本对COM对象进行调用。这种COM组件可以包含属性页、对话框等控件。

ATL中所使用的基本技术包括以下几个方面：

   COM技术

   C++模板类技术（Template)

   C++多继承技术（Multi-Inheritance)

**3.1 创建ALT服务应用程序**

文件-》新建-》项目->选择ALT项目->exe服务程序

MyService：用于生成ALT服务对应的可执行文件。

MyServicePS：动态库DLL类型，是·1服务程序与客户端进行通信的代理，御用负责服务程序与客户端的通信。

   注意：

   当你开发的组件需要用到代理/存根（即IPC（LPC+RPC）），就需要MyServicePS，否则MyServicePS无用

MyService.cpp是MyService项目的主文件。其中，CMyServiceModule是实现服务的类，也是从CAltServiceModuleT派生的，IDS\_SERVICENAME是服务的资源表示符。\_ALTModule是CMyServiceModule的对象，也是运行的服务实例。\_tWinMain()是服务程序的入口函数，它是AtlModule.WinModule.WinMain()函数来初始化和加载服务。

可以运行一下。

   注意：服务程序都是需要管理权限，所以exe需要配置默认以管理员运行。

   右键属性-》配置属性-》链接器-》清单文件-》UAC执行级别

   设置为 requireAdministrator (/level=‘requireAdministrator’)

**3.2 安装和卸载ALT服务**

**安装**

生成09.3.1MyService.exe，使用命令行安装：

09.3.1MyService.exe /Service

从任务管理服务查看已安装的服务

**卸载**

卸载服务命令

09.3.1MyService.exe /unRegService

我竟然没卸载成功

用sc命令卸载的

sc Delete 09.3.1MyService

**3.3 设置服务的属性**

**设置服务的名称**

打开09.3.1MyService.rc

IDS\_SERVICENAME：此处为服务名，修改即可修改服务名。

**设置服务描述和启动方式**

inline HRESULT RegisterAppId(\_In\_ bool bService = false) throw()

CAtlServiceModuleT::RegisterAppId()函数用于在注册表中注册服务，在CMyServiceModule类中重载RegisterAppId函数，可以在注册服务时设置服务的描述和启动方式。

以下代码实现修改服务描述和启动方式：

HRESULT CMyServiceModule::RegisterAppId(bool bService = false) throw()

{

HRESULT hr = S\_OK;

BOOL res = \_\_super::RegisterAppId(bService); // 调用父类的RegisterAppId()函数

if (bService) {

if(IsInstalled()) // 如果服务安装成功

{

// 以修改配置的权限打开服务控制器

SC\_HANDLE hSCM = ::OpenSCManager(NULL, NULL,SERVICE\_CHANGE\_CONFIG);

SC\_HANDLE hService = NULL;

if (hSCM == NULL)

hr = AtlHresultFromLastError();

else

{

// 以修改配置的权限打开安装的服务

hService = ::OpenService(hSCM, m\_szServiceName, SERVICE\_CHANGE\_CONFIG);

if (hService != NULL)

{

::ChangeServiceConfig(hService, SERVICE\_NO\_CHANGE,

SERVICE\_AUTO\_START,// 修改服务为自动启动

NULL, NULL, NULL, NULL, NULL, NULL, NULL,

m\_szServiceName); // 通过修改资源IDS\_SERVICENAME 修改服务的显示名字

// 设置服务描述信息

SERVICE\_DESCRIPTION Description;

TCHAR szDescription[1024];

ZeroMemory(szDescription, 1024);

ZeroMemory(&Description, sizeof(SERVICE\_DESCRIPTION));

lstrcpy(szDescription, \_T("我的第1个服务"));

Description.lpDescription = szDescription;

::ChangeServiceConfig2(hService, SERVICE\_CONFIG\_DESCRIPTION, &Description);

::CloseServiceHandle(hService);

}

else

hr = AtlHresultFromLastError();

::CloseServiceHandle(hSCM);

}

}

}

return hr;

}

如果服务安装成功，则程序调用ChangeServiceConfig函数和ChangeServiceConfig2函数设置服务的启动方式和描述。

**四、增加和使用组件**

如何在ATL服务中增加组件类以及在客户端程序中使用组件类的方法。

**4.1 增加组件**

右键项目 添加->新建项->Vistual C++ ATL ->ATL 简单对象

MyMath

注意：

* 添加组件类时会同时添加一个\*.regs文件，作用是注册组件类，以使在客户端程序中可以调用组件类。
* \*.regs文件中添加AppID时，以指定服务程序的ID，如果不添加，则客户端创建组件时会超时报错并报错误。

MyMath.regs

HKCR

{

//需要添加的AppID代码部分

NoRemove AppID

{

'%APPID%' = s 'MyService'

'MyService.EXE'

{

val AppID = s '%APPID%'

}

}

NoRemove CLSID

{

ForceRemove {0973bc39-d603-48ec-b784-41d7ca69c58f} = s 'MyMath class'

{

ForceRemove Programmable

LocalServer32 = s '%MODULE%'

{

val ServerExecutable = s '%MODULE\_RAW%'

}

TypeLib = s '{15f0d8bf-d610-45b6-b643-7ab15f7b31b3}'

Version = s '1.0'

}

}

}

**组件类中添加函数**

打开vs 类视图，可以找到IMyMath

选中MyMath右键-》添加方法，然后按如图设置参数和返回参数

查看

MyMath.cpp 添加Sum实现：

STDMETHODIMP CMyMath::Sum(LONG a, LONG b, LONG\* s)

{

// TODO: 在此处添加实现代码

\*s = a + b;

return S\_OK;

}

MyService.idl中添加

注意：最后要生成一个tbl文件，用于客户端访问。

**4.2 在客户端程序中使用组件类**

**五、Windows服务状态监视器实例**

**5.1 设计程序界面**

**5.2 设计自定义类CService**

**总结**