# 这一节，我们来学习注册表操作

# 学习大纲

|  |
| --- |
|  |

## RegCreateKey函数的语法

|  |
| --- |
| LSTATUS RegCreateKeyA(  [in] HKEY hKey,  [in, optional] LPCSTR lpSubKey,  [out] PHKEY phkResult  ); 参数 [in] hKey  打开的注册表项的句柄。 调用进程必须具有对密钥KEY\_CREATE\_SUB\_KEY访问权限。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  针对注册表项的安全描述符（而不是获取句柄时指定的访问掩码）检查密钥创建的访问权限。 因此，即使 *hKey* 是使用 *KEY\_READ 的 samDesir* 打开的，它也可用于创建密钥的操作（如果安全描述符允许）。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 或 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  [in, optional] lpSubKey  此函数打开或创建的密钥的名称。 此键必须是 *由 hKey* 参数标识的密钥的子项。  有关键名称的详细信息，请参阅 [注册表的结构](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/structure-of-the-registry)。  如果 *hKey* 是预定义键之一， *lpSubKey* 可能为 **NULL**。 在这种情况下， *phkResult* 接收传递给函数的相同 *hKey* 句柄。  [out] phkResult  一个变量的指针，此变量指向已打开或已创建的项的句柄。 如果该键不是预定义的注册表项之一，请在使用完句柄后调用 [RegCloseKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regclosekey) 函数。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用，以获取错误的泛型说明。 注解 应用程序无法创建作为 **HKEY\_USERS** 或 **HKEY\_LOCAL\_MACHINE**的直接子级的密钥。 应用程序可以在 **较低级别的HKEY\_USERS** 或 **HKEY\_LOCAL\_MACHINE** 树中创建子项。  如果服务或应用程序模拟不同的用户，请不要将此函数与 **HKEY\_CURRENT\_USER**一起使用。 请改为调用 [RegOpenCurrentUser](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopencurrentuser) 函数。  **RegCreateKey** 函数在指定路径中创建所有缺失的键。 应用程序可以利用此行为一次创建多个密钥。 例如，应用程序可以通过为 *lpSubKey* 参数指定以下格式的字符串，创建与前面的三个子项同时深四个级别的子项：  *subkey1\subkey2\subkey3\subkey4*  请注意，如果路径中的现有键拼写错误，则此行为将导致创建不需要的密钥。  **备注**  winreg.h 标头将 RegCreateKey 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)。 |

## RegCreateKeyEx函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LSTATUS RegCreateKeyExA(  [in] HKEY hKey,  [in] LPCSTR lpSubKey,  DWORD Reserved,  [in, optional] LPSTR lpClass,  [in] DWORD dwOptions,  [in] REGSAM samDesired,  [in, optional] const LPSECURITY\_ATTRIBUTES lpSecurityAttributes,  [out] PHKEY phkResult,  [out, optional] LPDWORD lpdwDisposition  ); 参数 [in] hKey  打开的注册表项的句柄。 调用进程必须具有对密钥KEY\_CREATE\_SUB\_KEY访问权限。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  针对注册表项的安全描述符（而不是获取句柄时指定的访问掩码）检查密钥创建的访问权限。 因此，即使 *hKey* 是使用 *samDesired* KEY\_READ 打开的，它也可用于修改注册表的操作（如果其安全描述符允许）。  此句柄由 **RegCreateKeyEx** 或 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  [in] lpSubKey  此函数打开或创建的子项的名称。 指定的子项必须是 *由 hKey* 参数标识的键的子项;它在注册表树中最多可以有 32 个级别。 有关键名称的详细信息，请参阅 [注册表的结构](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/structure-of-the-registry)。  如果 *lpSubKey* 是指向空字符串的指针， *则 phkResult* 将接收 *由 hKey* 指定的键的新句柄。  此参数不能为 **NULL**。  Reserved  此参数是保留的，必须为零。  [in, optional] lpClass  此键的用户定义类类型。 此参数可以忽略。 此参数可以为 NULL。  [in] dwOptions  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **REG\_OPTION\_BACKUP\_RESTORE**  0x00000004L | 如果设置了此标志，函数将忽略 *samDesired* 参数，并尝试使用备份或还原密钥所需的访问权限打开密钥。 如果调用线程启用了SE\_BACKUP\_NAME特权，则使用ACCESS\_SYSTEM\_SECURITY打开密钥，KEY\_READ访问权限。 如果调用线程启用了SE\_RESTORE\_NAME特权（从 Windows Vista 开始），则会使用ACCESS\_SYSTEM\_SECURITY、DELETE 和KEY\_WRITE访问权限打开密钥。 如果两个特权都已启用，则密钥具有这两种权限的组合访问权限。 有关详细信息，请参阅[使用特殊特权运行](https://learn.microsoft.com/zh-cn/windows/desktop/SecBP/running-with-special-privileges)。 | | **REG\_OPTION\_CREATE\_LINK**  0x00000002L | **注意** 注册表符号链接应仅在 绝对 必要时用于应用程序兼容性。    此键是符号链接。 目标路径分配给键的 L“SymbolicLinkValue”值。 目标路径必须是绝对注册表路径。 | | **REG\_OPTION\_NON\_VOLATILE**  0x00000000L | 此键不可变;这是默认值。 信息存储在文件中，并在系统重启时保留。 [RegSaveKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsavekeya) 函数保存不可变的键。 | | **REG\_OPTION\_VOLATILE**  0x00000001L | 函数创建的所有键都是可变的。 此信息存储在内存中，并且在卸载相应的注册表配置单元时不保留此信息。 对于 **HKEY\_LOCAL\_MACHINE**，仅当系统启动完全关闭时，才会发生这种情况。 对于 [RegLoadKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regloadkeya) 函数加载的注册表项，在执行相应的 [RegUnLoadKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regunloadkeya) 时会发生这种情况。 [RegSaveKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsavekeya) 函数不保存易失性密钥。 对于已存在的键，将忽略此标志。  **注意** 在用户选择的关机时，快速启动关闭是系统的默认行为。 |   [in] samDesired  一个掩码，指定要创建的密钥的访问权限。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  [in, optional] lpSecurityAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定是否可由子进程继承返回的句柄。 如果 *lpSecurityAttributes* 为 **NULL**，则无法继承句柄。  结构的 **lpSecurityDescriptor** 成员为新密钥指定安全描述符。 如果 *lpSecurityAttributes* 为 **NULL**，则密钥将获取默认的安全描述符。 密钥的默认安全描述符中的 ACL 继承自其直接父密钥。  [out] phkResult  一个变量的指针，此变量指向已打开或已创建的项的句柄。 如果该键不是预定义的注册表项之一，请在使用完句柄后调用 [RegCloseKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regclosekey) 函数。  [out, optional] lpdwDisposition  指向接收以下处置值之一的变量的指针。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **REG\_CREATED\_NEW\_KEY**  0x00000001L | 密钥不存在且已创建。 | | **REG\_OPENED\_EXISTING\_KEY**  0x00000002L | 密钥存在，只是打开而不更改。 |     如果 *lpdwDisposition* 为 **NULL**，则不返回任何处置信息。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用，以获取错误的泛型说明。 注解 **RegCreateKeyEx** 函数创建的键没有值。 应用程序可以使用 [RegSetValueEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsetvalueexa) 函数来设置键值。  **RegCreateKeyEx** 函数在指定路径中创建所有缺失的键。 应用程序可以利用此行为一次创建多个密钥。 例如，应用程序可以通过为 *lpSubKey* 参数指定以下格式的字符串，创建与前面的三个子项同时深四个级别的子项：  *subkey1\subkey2\subkey3\subkey4*  请注意，如果路径中的现有键拼写错误，则此行为将导致创建不需要的密钥。  应用程序无法创建作为 **HKEY\_USERS** 或 **HKEY\_LOCAL\_MACHINE**的直接子级的密钥。 应用程序可以在 **较低级别的HKEY\_USERS** 或 **HKEY\_LOCAL\_MACHINE** 树中创建子项。  如果服务或应用程序模拟不同的用户，请不要将此函数与 **HKEY\_CURRENT\_USER**一起使用。 请改为调用 [RegOpenCurrentUser](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopencurrentuser) 函数。  请注意，将重定向访问某些注册表项的操作。 有关详细信息，请参阅 [注册表虚拟化](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-virtualization) 和 [注册表中的 32 位和 64 位应用程序数据](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/32-bit-and-64-bit-application-data-in-the-registry)。  **备注**  winreg.h 标头将 RegCreateKeyEx 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)。 |

## RegOpenKey函数的语法

|  |
| --- |
| LSTATUS RegOpenKeyA(  [in] HKEY hKey,  [in, optional] LPCSTR lpSubKey,  [out] PHKEY phkResult  ); 参数 [in] hKey  打开的注册表项的句柄。 此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 或 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：   * **HKEY\_CLASSES\_ROOT** * **HKEY\_CURRENT\_CONFIG** * **HKEY\_CURRENT\_USER** * HKEY\_LOCAL\_MACHINE * **HKEY\_USERS**   [in, optional] lpSubKey  要打开的注册表项的名称。 此键必须是 *由 hKey* 参数标识的密钥的子项。  键名称不区分大小写。  如果此参数为 **NULL** 或指向空字符串的指针，则该函数将返回传入的同一句柄。  有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  [out] phkResult  一个变量的指针，此变量指向已打开键的句柄。 如果该键不是预定义的注册表项之一，请在使用完句柄后调用 [RegCloseKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regclosekey) 函数。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用，以获取错误的泛型说明。 注解 **RegOpenKey** 函数使用默认的安全访问掩码打开密钥。 如果打开密钥需要其他访问权限，则函数会失败，返回ERROR\_ACCESS\_DENIED。 在这种情况下，应用程序应使用 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数来指定访问掩码。  如果数据库中不存在指定键，**则 RegOpenKey** 不会创建指定键。  如果服务或应用程序模拟不同的用户，请不要将此函数与 **HKEY\_CURRENT\_USER**一起使用。 请改为调用 [RegOpenCurrentUser](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopencurrentuser) 函数。  **备注**  winreg.h 标头将 RegOpenKey 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)。 |

## RegOpenKeyEx函数的语法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LSTATUS RegOpenKeyExA(  [in] HKEY hKey,  [in, optional] LPCSTR lpSubKey,  [in] DWORD ulOptions,  [in] REGSAM samDesired,  [out] PHKEY phkResult  ); 参数 [in] hKey  打开的注册表项的句柄。 此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 或 **RegOpenKeyEx** 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOTHKEY\_CURRENT\_CONFIGHKEY\_CURRENT\_USERHKEY\_LOCAL\_MACHINEHKEY\_USERS**  [in, optional] lpSubKey  要打开的注册表子项的名称。  键名称不区分大小写。  如果 *lpSubKey* 参数为 **NULL** 或指向空字符串的指针，并且 *hKey* 是预定义键，则系统会刷新预定义的键，并且 *phkResult* 接收传递到函数中的同一 *hKey* 句柄。 否则， *phkResult* 将接收打开的密钥的新句柄。  有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  [in] ulOptions  指定要在打开密钥时应用的选项。 将此参数设置为零或以下参数：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **REG\_OPTION\_OPEN\_LINK** | 键是符号链接。 仅当绝对必要时才应使用注册表符号链接。 |   [in] samDesired  一个掩码，指定要打开的密钥的所需访问权限。 如果密钥的安全描述符不允许对调用进程进行请求的访问，则函数将失败。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  [out] phkResult  一个变量的指针，此变量指向已打开键的句柄。 如果该键不是预定义的注册表项之一，请在使用完句柄后调用 [RegCloseKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regclosekey) 函数。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用，以获取错误的泛型说明。  **备注**  在旧版 Windows 上，此 API 也由 kernel32.dll 公开。 注解 与 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 函数不同，如果注册表中不存在指定键， **RegOpenKeyEx** 函数不会创建指定键。  某些注册表操作对密钥的安全描述符（而不是获取密钥句柄时指定的访问掩码）执行访问检查。 例如，即使某个项以KEY\_READ的 *samDesir* 打开，也可用于创建注册表项（如果密钥的安全描述符允许）。 相比之下， [RegSetValueEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsetvalueexa) 函数专门要求使用KEY\_SET\_VALUE访问权限打开密钥。  如果服务或应用程序模拟不同的用户，请不要将此函数与 **HKEY\_CURRENT\_USER**一起使用。 请改为调用 [RegOpenCurrentUser](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopencurrentuser) 函数。  请注意，将重定向访问某些注册表项的操作。 有关详细信息，请参阅 [注册表虚拟化](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-virtualization) 和 [注册表中的 32 位和 64 位应用程序数据](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/32-bit-and-64-bit-application-data-in-the-registry)。 示例 C++复制  lResult = RegOpenKeyEx (hKeyRoot, lpSubKey, 0, KEY\_READ, &hKey);  if (lResult != ERROR\_SUCCESS)  {  if (lResult == ERROR\_FILE\_NOT\_FOUND) {  printf("Key not found.\n");  return TRUE;  }  else {  printf("Error opening key.\n");  return FALSE;  }  }  若要在上下文中查看此示例，请参阅 [使用子项删除密钥](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/deleting-a-key-with-subkeys)。  **备注**  winreg.h 标头将 RegOpenKeyEx 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)。 |

## RegQueryValue函数的语法

|  |
| --- |
| LSTATUS RegQueryValueExA(  [in] HKEY hKey,  [in, optional] LPCSTR lpValueName,  LPDWORD lpReserved,  [out, optional] LPDWORD lpType,  [out, optional] LPBYTE lpData,  [in, out, optional] LPDWORD lpcbData  ); 参数 [in] hKey  打开的注册表项的句柄。 密钥必须已使用KEY\_QUERY\_VALUE访问权限打开。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa)、 [RegCreateKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeytransacteda)、 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 或 [RegOpenKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeytransacteda) 函数返回。 它也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_PERFORMANCE\_DATA**  **HKEY\_PERFORMANCE\_NLSTEXT**  **HKEY\_PERFORMANCE\_TEXT**  **HKEY\_USERS**  [in, optional] lpValueName  注册表值的名称。  如果 *lpValueName* 为 **NULL** 或空字符串“”，则函数将检索键的未命名值或默认值的类型和数据（如果有）。  如果 *lpValueName* 指定的值不在注册表中，则函数将返回ERROR\_FILE\_NOT\_FOUND。  键不会自动具有未命名值或默认值。 未命名的值可以是任何类型的值。 有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  lpReserved  此参数是保留的，必须为 **NULL**。  [out, optional] lpType  指向变量的指针，该变量接收指示存储在指定值中的数据类型的代码。 有关可能的类型代码的列表，请参阅 [注册表值类型](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-value-types)。 如果不需要类型代码， *则 lpType* 参数可以为 **NULL** 。  [out, optional] lpData  指向接收值数据的缓冲区的指针。 如果不需要数据，此参数可以为 **NULL** 。  [in, out, optional] lpcbData  指向变量的指针，该变量指定 *lpData* 参数指向的缓冲区的大小（以字节为单位）。 当函数返回时，此变量包含复制到 *lpData 的数据*的大小。  仅当 *lpData* 为 **NULL** 时*，lData* 参数才能为 **NULL**。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，则此大小包括任何终止 **null** 字符或字符，除非数据存储时没有它们。 有关详细信息，请参阅“备注”。  如果 *lpData* 参数指定的缓冲区不够大，无法保存数据，则函数将返回ERROR\_MORE\_DATA并将所需的缓冲区大小存储在 *l ERROR\_MORE\_DATA Data* 指向的变量中。 在这种情况下， *lpData* 缓冲区的内容未定义。  如果 *lpData* 为 **NULL**，且 *lData* 为非 **NULL**，则该函数返回ERROR\_SUCCESS，并将数据的大小（以字节为单位）存储在 *l ERROR\_SUCCESS Data* 指向的变量中。 这使应用程序能够确定为值的数据分配缓冲区的最佳方式。  如果 *hKey* 指定 **HKEY\_PERFORMANCE\_DATA** 并且 *lpData* 缓冲区不够大，无法包含所有返回的数据， **则 RegQueryValueEx** 将返回ERROR\_MORE\_DATA，并且通过 *lusbData* 参数返回的值未定义。 这是因为性能数据的大小可能会从一次调用更改为下一个调用。 在这种情况下，必须增加缓冲区大小，并再次调用 **RegQueryValueEx** ，并在 *lusbData* 参数中传递更新的缓冲区大小。 重复此操作，直到函数成功。 你需要保留一个单独的变量来跟踪缓冲区大小，因为 *lData* 返回的值不可预知。  如果 *lpValueName* 注册表值不存在， **则 RegQueryValueEx** 将返回ERROR\_FILE\_NOT\_FOUND，并且通过 *lusbData* 参数返回的值未定义。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 [系统错误代码](https://learn.microsoft.com/zh-cn/windows/desktop/Debug/system-error-codes)。  如果 *lpData* 缓冲区太小而无法接收数据，则函数将返回ERROR\_MORE\_DATA。  如果 *lpValueName* 注册表值不存在，该函数将返回ERROR\_FILE\_NOT\_FOUND。 注解 应用程序通常调用 [RegEnumValue](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regenumvaluea) 来确定值名称，然后 **调用 RegQueryValueEx** 来检索名称的数据。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，则字符串可能未使用正确的终止 **null** 字符进行存储。 因此，即使函数返回ERROR\_SUCCESS，应用程序也应该确保字符串在使用前正确终止;否则，它可能会覆盖缓冲区。 (请注意，REG\_MULTI\_SZ字符串应具有两个终止 **null** 字符。) 应用程序可以确保字符串正确终止的一种方法是使用 [RegGetValue](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-reggetvaluea)，这将根据需要添加终止 **null** 字符。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，并且此函数的 ANSI 版本通过显式调用 **RegQueryValueExA** 或在将 Windows.h 文件) 之前未定义 UNICODE 来 (使用，则此函数会将存储的 Unicode 字符串转换为 ANSI 字符串，然后再将其复制到 *lpData* 指向的缓冲区。  调用 **RegQueryValueEx** 函数时，将 *hKey* 设置为 **HKEY\_PERFORMANCE\_DATA** 句柄和指定对象的值字符串时，返回的数据结构有时具有未请求的对象。 不要感到惊讶：这是正常行为。 调用 **RegQueryValueEx** 函数时，应始终希望遍历返回的数据结构以查找请求的对象。  请注意，将重定向访问某些注册表项的操作。 有关详细信息，请参阅 [注册表虚拟化](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-virtualization) 和 [注册表中的 32 位和 64 位应用程序数据](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/32-bit-and-64-bit-application-data-in-the-registry)。 示例 请确保在每次调用此函数时，重新初始化 *由 l++Data* 参数指向的值。 在循环中调用此函数时，这一点非常重要，如下面的代码示例所示。  C++复制  #include <windows.h>  #include <malloc.h>  #include <stdio.h>  #define TOTALBYTES 8192  #define BYTEINCREMENT 4096  void main()  {  DWORD BufferSize = TOTALBYTES;  DWORD cbData;  DWORD dwRet;  PPERF\_DATA\_BLOCK PerfData = (PPERF\_DATA\_BLOCK) malloc( BufferSize );  cbData = BufferSize;  printf("\nRetrieving the data...");  dwRet = RegQueryValueEx( HKEY\_PERFORMANCE\_DATA,  TEXT("Global"),  NULL,  NULL,  (LPBYTE) PerfData,  &cbData );  while( dwRet == ERROR\_MORE\_DATA )  {  // Get a buffer that is big enough.  BufferSize += BYTEINCREMENT;  PerfData = (PPERF\_DATA\_BLOCK) realloc( PerfData, BufferSize );  cbData = BufferSize;  printf(".");  dwRet = RegQueryValueEx( HKEY\_PERFORMANCE\_DATA,  TEXT("Global"),  NULL,  NULL,  (LPBYTE) PerfData,  &cbData );  }  if( dwRet == ERROR\_SUCCESS )  printf("\n\nFinal buffer size is %d\n", BufferSize);  else printf("\nRegQueryValueEx failed (%d)\n", dwRet);  }  **备注**  winreg.h 标头将 RegQueryValueEx 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)。 |

## RegQueryValueEx函数的语法

|  |
| --- |
| LSTATUS RegQueryValueExA(  [in] HKEY hKey,  [in, optional] LPCSTR lpValueName,  LPDWORD lpReserved,  [out, optional] LPDWORD lpType,  [out, optional] LPBYTE lpData,  [in, out, optional] LPDWORD lpcbData  ); 参数 [in] hKey  打开的注册表项的句柄。 密钥必须已使用KEY\_QUERY\_VALUE访问权限打开。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa)、 [RegCreateKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeytransacteda)、 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 或 [RegOpenKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeytransacteda) 函数返回。 它也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_PERFORMANCE\_DATA**  **HKEY\_PERFORMANCE\_NLSTEXT**  **HKEY\_PERFORMANCE\_TEXT**  **HKEY\_USERS**  [in, optional] lpValueName  注册表值的名称。  如果 *lpValueName* 为 **NULL** 或空字符串“”，则函数将检索键的未命名值或默认值的类型和数据（如果有）。  如果 *lpValueName* 指定的值不在注册表中，则函数将返回ERROR\_FILE\_NOT\_FOUND。  键不会自动具有未命名值或默认值。 未命名的值可以是任何类型的值。 有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  lpReserved  此参数是保留的，必须为 **NULL**。  [out, optional] lpType  指向变量的指针，该变量接收指示存储在指定值中的数据类型的代码。 有关可能的类型代码的列表，请参阅 [注册表值类型](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-value-types)。 如果不需要类型代码， *则 lpType* 参数可以为 **NULL** 。  [out, optional] lpData  指向接收值数据的缓冲区的指针。 如果不需要数据，此参数可以为 **NULL** 。  [in, out, optional] lpcbData  指向变量的指针，该变量指定 *lpData* 参数指向的缓冲区的大小（以字节为单位）。 当函数返回时，此变量包含复制到 *lpData 的数据*的大小。  仅当 *lpData* 为 **NULL** 时*，lData* 参数才能为 **NULL**。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，则此大小包括任何终止 **null** 字符或字符，除非数据存储时没有它们。 有关详细信息，请参阅“备注”。  如果 *lpData* 参数指定的缓冲区不够大，无法保存数据，则函数将返回ERROR\_MORE\_DATA并将所需的缓冲区大小存储在 *l ERROR\_MORE\_DATA Data* 指向的变量中。 在这种情况下， *lpData* 缓冲区的内容未定义。  如果 *lpData* 为 **NULL**，且 *lData* 为非 **NULL**，则该函数返回ERROR\_SUCCESS，并将数据的大小（以字节为单位）存储在 *l ERROR\_SUCCESS Data* 指向的变量中。 这使应用程序能够确定为值的数据分配缓冲区的最佳方式。  如果 *hKey* 指定 **HKEY\_PERFORMANCE\_DATA** 并且 *lpData* 缓冲区不够大，无法包含所有返回的数据， **则 RegQueryValueEx** 将返回ERROR\_MORE\_DATA，并且通过 *lusbData* 参数返回的值未定义。 这是因为性能数据的大小可能会从一次调用更改为下一个调用。 在这种情况下，必须增加缓冲区大小，并再次调用 **RegQueryValueEx** ，并在 *lusbData* 参数中传递更新的缓冲区大小。 重复此操作，直到函数成功。 你需要保留一个单独的变量来跟踪缓冲区大小，因为 *lData* 返回的值不可预知。  如果 *lpValueName* 注册表值不存在， **则 RegQueryValueEx** 将返回ERROR\_FILE\_NOT\_FOUND，并且通过 *lusbData* 参数返回的值未定义。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 [系统错误代码](https://learn.microsoft.com/zh-cn/windows/desktop/Debug/system-error-codes)。  如果 *lpData* 缓冲区太小而无法接收数据，则函数将返回ERROR\_MORE\_DATA。  如果 *lpValueName* 注册表值不存在，该函数将返回ERROR\_FILE\_NOT\_FOUND。 注解 应用程序通常调用 [RegEnumValue](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regenumvaluea) 来确定值名称，然后 **调用 RegQueryValueEx** 来检索名称的数据。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，则字符串可能未使用正确的终止 **null** 字符进行存储。 因此，即使函数返回ERROR\_SUCCESS，应用程序也应该确保字符串在使用前正确终止;否则，它可能会覆盖缓冲区。 (请注意，REG\_MULTI\_SZ字符串应具有两个终止 **null** 字符。) 应用程序可以确保字符串正确终止的一种方法是使用 [RegGetValue](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-reggetvaluea)，这将根据需要添加终止 **null** 字符。  如果数据具有REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，并且此函数的 ANSI 版本通过显式调用 **RegQueryValueExA** 或在将 Windows.h 文件) 之前未定义 UNICODE 来 (使用，则此函数会将存储的 Unicode 字符串转换为 ANSI 字符串，然后再将其复制到 *lpData* 指向的缓冲区。  调用 **RegQueryValueEx** 函数时，将 *hKey* 设置为 **HKEY\_PERFORMANCE\_DATA** 句柄和指定对象的值字符串时，返回的数据结构有时具有未请求的对象。 不要感到惊讶：这是正常行为。 调用 **RegQueryValueEx** 函数时，应始终希望遍历返回的数据结构以查找请求的对象。  请注意，将重定向访问某些注册表项的操作。 有关详细信息，请参阅 [注册表虚拟化](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-virtualization) 和 [注册表中的 32 位和 64 位应用程序数据](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/32-bit-and-64-bit-application-data-in-the-registry)。 示例 请确保在每次调用此函数时，重新初始化 *由 l++Data* 参数指向的值。 在循环中调用此函数时，这一点非常重要，如下面的代码示例所示。  C++复制  #include <windows.h>  #include <malloc.h>  #include <stdio.h>  #define TOTALBYTES 8192  #define BYTEINCREMENT 4096  void main()  {  DWORD BufferSize = TOTALBYTES;  DWORD cbData;  DWORD dwRet;  PPERF\_DATA\_BLOCK PerfData = (PPERF\_DATA\_BLOCK) malloc( BufferSize );  cbData = BufferSize;  printf("\nRetrieving the data...");  dwRet = RegQueryValueEx( HKEY\_PERFORMANCE\_DATA,  TEXT("Global"),  NULL,  NULL,  (LPBYTE) PerfData,  &cbData );  while( dwRet == ERROR\_MORE\_DATA )  {  // Get a buffer that is big enough.  BufferSize += BYTEINCREMENT;  PerfData = (PPERF\_DATA\_BLOCK) realloc( PerfData, BufferSize );  cbData = BufferSize;  printf(".");  dwRet = RegQueryValueEx( HKEY\_PERFORMANCE\_DATA,  TEXT("Global"),  NULL,  NULL,  (LPBYTE) PerfData,  &cbData );  }  if( dwRet == ERROR\_SUCCESS )  printf("\n\nFinal buffer size is %d\n", BufferSize);  else printf("\nRegQueryValueEx failed (%d)\n", dwRet);  }  **备注**  winreg.h 标头将 RegQueryValueEx 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)。 |

## RegDeleteKey函数的语法

|  |
| --- |
| LSTATUS RegDeleteKeyA(  [in] HKEY hKey,  [in] LPCSTR lpSubKey  ); 参数 [in] hKey  打开的注册表项的句柄。 此密钥的访问权限不会影响删除操作。 有关访问权限的详细信息，请参阅 [注册表项安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 或 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  [in] lpSubKey  要删除的文件的名称。 它必须是 *hKey* 标识的项的子项，但它不能具有子项。 此参数不能为 **NULL**。  函数使用 DELETE 访问权限打开子项。  键名称不区分大小写。  有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 若要获取错误的一般说明，可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用。 注解 在关闭已删除密钥的最后一个句柄之前，不会删除该密钥。  要删除的子项不得具有子项。 若要删除某个键及其所有子项，需要枚举子项并单独删除它们。 若要以递归方式删除密钥，请使用 [RegDeleteTree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regdeletetreea) 或 [SHDeleteKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlwapi/nf-shlwapi-shdeletekeya) 函数。 示例 有关使用此函数的示例，请参阅 [使用子项删除密钥](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/deleting-a-key-with-subkeys)。  **备注**  在旧版 Windows 上，此 API 也由 kernel32.dll 公开。  **备注**  winreg.h 标头将 RegDeleteKey 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)。 |

## RegDeleteKeyEx函数的语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LSTATUS RegDeleteKeyExA(  [in] HKEY hKey,  [in] LPCSTR lpSubKey,  [in] REGSAM samDesired,  DWORD Reserved  ); 参数 [in] hKey  打开的注册表项的句柄。 此密钥的访问权限不会影响删除操作。 有关访问权限的详细信息，请参阅 [注册表项安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa) 或 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 函数返回，也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  [in] lpSubKey  要删除的文件的名称。 此键必须是 *由 hKey* 参数的值指定的键的子项。  函数使用 DELETE 访问权限打开子项。  键名称不区分大小写。  此参数的值不能为 **NULL**。  [in] samDesired  访问掩码 指定注册表的特定于平台的视图。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **KEY\_WOW64\_32KEY**  0x0200 | 从 32 位注册表视图中删除项。 | | **KEY\_WOW64\_64KEY**  0x0100 | 从 64 位注册表视图中删除项。 |   Reserved  此参数是保留的，必须为零。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志结合使用来获取错误的常规说明。 注解 在关闭已删除密钥的最后一个句柄之前，不会删除该密钥。  在 WOW64 上，32 位应用程序查看与 64 位应用程序视图的注册表树分开的注册表树。 此函数使应用程序能够删除备用注册表视图中的条目。  要删除的子项不得具有子项。 若要删除某个键及其所有子项，需要枚举子项并单独删除它们。 若要以递归方式删除密钥，请使用 [RegDeleteTree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regdeletetreea) 或 [SHDeleteKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlwapi/nf-shlwapi-shdeletekeya) 函数。  如果函数成功， **RegDeleteKeyEx** 将从注册表中删除指定的项。 将删除整个键，包括其所有值。  **备注**  在旧版 Windows 上，此 API 也由 kernel32.dll 公开。  **备注**  winreg.h 标头将 RegDeleteKeyEx 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)。 |

## RegSetValue函数用法

|  |
| --- |
| LSTATUS RegSetValueA(  [in] HKEY hKey,  [in, optional] LPCSTR lpSubKey,  [in] DWORD dwType, //必须是REG\_SZ  [in] LPCSTR lpData,  [in] DWORD cbData  ); 参数 [in] hKey  打开的注册表项的句柄。 密钥必须已使用KEY\_SET\_VALUE访问权限打开。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa)、 [RegCreateKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeytransacteda)、 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 或 [RegOpenKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeytransacteda) 函数返回。 也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  [in, optional] lpSubKey  *hKey* 参数的子项的名称。 函数设置指定子项的默认值。 如果 *lpSubKey* 不存在，该函数将创建它。  键名称不区分大小写。  如果此参数为 **NULL** 或指向空字符串，则该函数将设置 *由 hKey* 标识的键的默认值。  有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  [in] dwType  要存储的信息的类型。 此参数必须是REG\_SZ类型。 若要存储其他数据类型，请使用 [RegSetValueEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsetvalueexa) 函数。  [in] lpData  要存储的数据。 此参数不能为 **NULL**。  [in] cbData  忽略此参数。 函数根据 *lpData* 参数中的数据大小计算此值。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志结合使用来获取错误的常规说明。 注解 如果 *lpSubKey* 参数指定的键不存在， **则 RegSetValue** 函数将创建它。  如果通过显式调用 **RegSetValueA** 或在将 Windows.h 文件) 之前未定义 UNICODE 来 (使用此函数的 ANSI 版本，则 *lpData* 参数必须是 ANSI 字符串。 字符串在存储到注册表中之前会转换为 Unicode。  **备注**  winreg.h 标头将 RegSetValue 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)。 |

## RegSetValueEx的用法

|  |
| --- |
| LSTATUS RegSetValueExA(  [in] HKEY hKey,  [in, optional] LPCSTR lpValueName,  DWORD Reserved,  [in] DWORD dwType,  [in] const BYTE \*lpData,  [in] DWORD cbData  ); 参数 [in] hKey  打开的注册表项的句柄。 密钥必须已使用KEY\_SET\_VALUE访问权限打开。 有关详细信息，请参阅 [注册表项安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-key-security-and-access-rights)。  此句柄由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa)、 [RegCreateKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeytransacteda)、 [RegOpenKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 或 [RegOpenKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeytransacteda) 函数返回。 也可以是以下 [预定义键](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/predefined-keys)之一：  **HKEY\_CLASSES\_ROOT**  **HKEY\_CURRENT\_CONFIG**  **HKEY\_CURRENT\_USER**  HKEY\_LOCAL\_MACHINE  **HKEY\_USERS**  此函数的 Unicode 版本支持以下附加预定义键：   * **HKEY\_PERFORMANCE\_TEXT** * **HKEY\_PERFORMANCE\_NLSTEXT**   [in, optional] lpValueName  要设置的值的名称。 如果键中尚不存在具有此名称的值，则函数会将其添加到键中。  如果 *lpValueName* 为 **NULL** 或空字符串“”，则该函数将设置键的未命名值或默认值的类型和数据。  有关详细信息，请参阅 [注册表元素大小限制](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-element-size-limits)。  注册表项没有默认值，但它们可以有一个未命名的值，该值可以是任何类型的。  Reserved  此参数是保留的，必须为零。  [in] dwType  *lpData* 参数指向的数据类型。 有关可能类型的列表，请参阅 [注册表值类型](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-value-types)。  [in] lpData  要存储的数据。  对于基于字符串的类型（如 REG\_SZ），字符串必须以 **null** 结尾。 对于 REG\_MULTI\_SZ 数据类型，字符串必须以两个 **null** 字符结尾。  **注意** lpData 指示 **null** 值有效，但是，如果是这种情况， *cbData* 必须设置为“0”。    [in] cbData  *lpData* 参数指向的信息的大小（以字节为单位）。 如果数据的类型为 REG\_SZ、REG\_EXPAND\_SZ 或 REG\_MULTI\_SZ， *则 cbData* 必须包含终止 **null** 字符的大小。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志结合使用来获取错误的常规说明。 注解 值大小受可用内存的限制。 但是，在注册表中存储较大的值可能会影响其性能。 (超过 2,048 字节的长值) 应存储为文件，文件的位置存储在注册表中。  应用程序元素（如图标、位图和可执行文件）应存储为文件，而不是放置在注册表中。  如果 *dwType* 是REG\_SZ、REG\_MULTI\_SZ或REG\_EXPAND\_SZ类型，并且此函数的 ANSI 版本通过显式调用 **RegSetValueExA** 或未在将 Windows.h 文件) 之前未定义 UNICODE 来 (使用，则 *lpData* 参数指向的数据必须是 ANSI 字符串。 字符串在存储到注册表中之前会转换为 Unicode。  请注意，将重定向访问某些注册表项的操作。 有关详细信息，请参阅 [注册表虚拟化](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/registry-virtualization) 和 [注册表中的 32 位和 64 位应用程序数据](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/32-bit-and-64-bit-application-data-in-the-registry)。  请考虑使用 [RegSetKeyValue](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regsetkeyvaluea) 函数，该函数提供了一种更方便的方式来设置注册表项的值。  **备注**  winreg.h 标头将 RegSetValueEx 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)。 |

## RegCloseKey函数的语法

|  |
| --- |
| LSTATUS RegCloseKey(  [in] HKEY hKey  ); 参数 [in] hKey  要关闭的打开键的句柄。 该句柄必须已由 [RegCreateKeyEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeyexa)、[RegCreateKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regcreatekeytransacteda)、[RegOpenKeyEx、RegOpenKeyTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regopenkeyexa) 或 [RegConnectRegistry](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regconnectregistrya) 函数打开。 返回值 如果函数成功，则返回值为 ERROR\_SUCCESS。  如果函数失败，则返回值为 Winerror.h 中定义的非零错误代码。 可以将 [FormatMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-formatmessage) 函数与 FORMAT\_MESSAGE\_FROM\_SYSTEM 标志一起使用，以获取错误的泛型说明。 注解 指定键的句柄在关闭后不应使用，因为它将不再有效。 键句柄的打开时间不应超过必要时间。  **RegCloseKey** 函数在返回之前不一定将信息写入注册表;缓存刷新到硬盘可能需要几秒钟的时间。 如果应用程序必须将注册表信息显式写入硬盘，则可以使用 [RegFlushKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regflushkey) 函数。 但是，**RegFlushKey** 使用许多系统资源，仅在必要时才应调用。 |

# 演练

## 1.新建一个mfc基于对话框的应用程序，取名Lesson26-registry-op，把取消按钮删除，把确定按钮文本改为退出程序

|  |
| --- |
|  |

### 注意：凡是对话框程序都有一个问题，就是按下回车会退出，需要重新OnButtonClickedOK函数

|  |
| --- |
|  |

## 2.我们先给对话框添加一个组框，标题是读取CPU信息，然后在组框内给对话框添加一个按钮，改名CPU INFO，用它来读取注册表中CPU的信息，对应的注册表键是：HKEY\_LOCAL\_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor，然后在按钮的下面添加一个静态文本，把它的ID改为：IDC\_CPU\_INFO。

|  |
| --- |
|  |

## 3.双击按钮进入点击事件处理代码，我们来写下面的代码

|  |
| --- |
| void CLesson26egistryopDlg::OnBnClickedBtnCpu()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64];  DWORD dwSize;  //有坑， RegQueryValueEx最后一个参数需要先初始化  dwSize = sizeof(szData);  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0"),  0,KEY\_QUERY\_VALUE, &hKey);  if(lRet == ERROR\_SUCCESS)  {  //获取对应的键的值  if(ERROR\_SUCCESS == RegQueryValueEx(hKey,\_T("ProcessorNameString"),NULL,NULL,(LPBYTE)szData,&dwSize))  {  SetDlgItemText(IDC\_CPU\_INFO,szData);  }  else  {  MessageBox(\_T("获取键值失败"));  }  }  else  {  MessageBox(\_T("查询注册表键失败"));  }  RegCloseKey(hKey);//查关闭句柄  } |

### 效果

|  |
| --- |
|  |

## 4.然后我们再添加一个组框，取名：创建注册表键，增加一个按钮，取名：创建键然后我们添加一个提示文本，然后再添加一个静态文本，ID为: IDC\_STATIC\_RESULT

|  |
| --- |
|  |

### 我们想实现的功能是，当我们创建一个键成功了，我们就把它显示在下面的静态文本控件，代码如下

|  |
| --- |
| void CLesson26egistryopDlg::OnBnClickedBtnCreate()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64],szSubKey[64],szValue[64];  DWORD dwSize,dwValueSize;  dwSize = sizeof(szData);//有坑，需要用sizeof初始化  dwValueSize = sizeof(szValue);//有坑，需要用sizeof初始化  lstrcpy(szSubKey,\_T("SOFTWARE\\KENNYCAI\\MyApp"));  lstrcpy(szValue,\_T("Smart Software,version 1.0"));  //我们用HKEY\_CURRENT\_USER来创建注册表键  //用HKEY\_LOCAL\_MACHINE也ok，不过设置在HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node\下面  //if(ERROR\_SUCCESS == RegCreateKey(HKEY\_CURRENT\_USER,szSubKey,&hKey))  if(ERROR\_SUCCESS == RegCreateKey(HKEY\_LOCAL\_MACHINE,szSubKey,&hKey))  {  //MessageBox(\_T("创建注册表键成功"));  //创建注册表key后需要设置value，使用RegSetValue函数来实现  if(ERROR\_SUCCESS == RegSetValueEx(hKey,\_T("SoftwareName"),NULL,REG\_SZ,  (LPBYTE)szValue,dwValueSize))  {  if(ERROR\_SUCCESS == RegQueryValueEx(hKey,\_T("SoftwareName"),NULL,NULL,(LPBYTE)szData,&dwSize))  {  SetDlgItemText(IDC\_STATIC\_RESULT,szData);  }  }  else  {  MessageBox(\_T("设置注册表键值失败"));  }  }  else  {  MessageBox(\_T("创建注册表键失败"));  }  RegCloseKey(hKey);  } |

### 注意：有大坑

|  |
| --- |
| 在64位系统中，使用VC++的RegQueryValueEx读取HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run时遇到失败。问题在于64位系统的注册表结构与32位不同，部分路径被映射到Wow6432Node下。为解决此问题，需在RegOpenKeyEx函数中添加KEY\_WOW64\_64KEY参数以正确读取32位注册表路径。你对HKEY\_LOCAL\_MACHINE\SOFTWARE下面的然后子键做的操作都会被映射到HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node下面 |

## 5.然后我们可以添加一个按钮，文本为WRITE,ID为IDC\_BTN\_WRITE,然后我们可以添加下面的代码，在HKEY\_LOCAL\_MACHINE\SOFTWARE\GoodZip里面添加一个名叫TestKey的字符串，Value是：This is a test Value.代码如下

|  |
| --- |
| void CLesson26egistryopDlg::OnBnClickedBtnWrite()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64],szValue[64];  DWORD dwSize,dwValueSize;;  //有坑， RegQueryValueEx最后一个参数需要先初始化  dwSize = sizeof(szData);  dwValueSize = sizeof(szValue);//有坑，需要用sizeof初始化  lstrcpy(szValue,\_T("This is a test Value."));  /\*lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\WOW6432Node\\KENNYCAI"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);\*/  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\GoodZip"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);  if(lRet == ERROR\_SUCCESS)  {    lRet = RegSetValueEx(hKey,\_T("TestKey"),NULL,REG\_SZ,(LPBYTE)szValue,dwValueSize);  if(lRet == ERROR\_SUCCESS)  {  MessageBox(\_T("设置注册表键成功"));  }  else  {  MessageBox(\_T("设置注册表键失败"));    }  }  else  {  MessageBox(\_T("打开注册表键失败"));  }  RegCloseKey(hKey);  } |

### 运行程序后，发现在HKEY\_LOCAL\_MACHINE\SOFTWARE\GoodZip里面没有我们新建的字符串，

|  |
| --- |
|  |

### 但是在HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node\GoodZip里面有这个字符串

|  |
| --- |
|  |

### 注意，

### 1.在RegSetValueEx中，如果需要为一个key设置默认值，需要把第二个参数设置为NULL

### 2.除了可以设置字符串以外，还可以设置DWORD值

|  |
| --- |
|  |

## 6、我们也来给他设置一个DWORD值Age，代码如下

|  |
| --- |
| void CLesson26egistryopDlg::OnBnClickedBtnWrite()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64],szValue[64];  DWORD dwSize,dwValueSize;;  //有坑， RegQueryValueEx最后一个参数需要先初始化  dwSize = sizeof(szData);  dwValueSize = sizeof(szValue);//有坑，需要用sizeof初始化  lstrcpy(szValue,\_T("This is a test Value."));  /\*lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\WOW6432Node\\KENNYCAI"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);\*/  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\GoodZip"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);  if(lRet == ERROR\_SUCCESS)  {  //下面的函数，如果需要为一个key设置默认值，需要把第二个参数也设置为NULL  lRet = RegSetValueEx(hKey,\_T("TestKey"),NULL,REG\_SZ,(LPBYTE)szValue,dwValueSize);  if(lRet == ERROR\_SUCCESS)  {  MessageBox(\_T("设置注册表键成功"));  }  else  {  MessageBox(\_T("设置注册表键失败"));    }  //还可以设置DWORD值,第5个参数是用指针的，如果不是，设置失败  DWORD dwAge = 40;  RegSetValueEx(hKey,\_T("Age"),NULL,REG\_DWORD,(LPBYTE)&dwAge,4);  }  else  {  MessageBox(\_T("打开注册表键失败"));  }  RegCloseKey(hKey);  } |

### 效果：

|  |
| --- |
|  |

### 不用先打开注册表键，也能够读取他的值？

|  |
| --- |
|  |

## 7.我们也来添加一个READ按钮，点击按钮读取键值放到静态文本框中

|  |
| --- |
|  |
| void CLesson26egistryopDlg::OnBnClickedBtnRead()  {  // TODO: 在此添加控件通知处理程序代码  LONG lValue;  RegQueryValue(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),NULL,&lValue);  TCHAR \*buf = new TCHAR[lValue] ;  RegQueryValue(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),buf,&lValue);  SetDlgItemText(IDC\_STATIC\_RESULT,buf);  } |

### 效果

|  |
| --- |
|  |

## 8.上面是读取字符串，下面我们来读取DWORD值

|  |
| --- |
| void CLesson26egistryopDlg::OnBnClickedBtnReadAge()  {  // TODO: 在此添加控件通知处理程序代码  HKEY hKey;  RegOpenKey(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),&hKey);  DWORD dwType,dwValue,dwAge;  RegQueryValueEx(hKey,\_T("Age"),0,&dwType,(LPBYTE)&dwAge,&dwValue);  SetDlgItemInt(IDC\_STATIC\_RESULT,dwAge);  RegCloseKey(hKey);  } |

### 效果：

|  |
| --- |
|  |

# Lesson26-egistry-opDlg.cpp的完整代码如下

## Lesson26-egistry-opDlg.cpp

|  |
| --- |
| // Lesson26-egistry-opDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson26-egistry-op.h"  #include "Lesson26-egistry-opDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CLesson26egistryopDlg 对话框  CLesson26egistryopDlg::CLesson26egistryopDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CLesson26egistryopDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CLesson26egistryopDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CLesson26egistryopDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDOK, &CLesson26egistryopDlg::OnBnClickedOk)  ON\_BN\_CLICKED(IDC\_BTN\_CPU, &CLesson26egistryopDlg::OnBnClickedBtnCpu)  ON\_BN\_CLICKED(IDC\_BTN\_CREATE, &CLesson26egistryopDlg::OnBnClickedBtnCreate)  ON\_BN\_CLICKED(IDC\_BTN\_WRITE, &CLesson26egistryopDlg::OnBnClickedBtnWrite)  ON\_BN\_CLICKED(IDC\_BTN\_READ, &CLesson26egistryopDlg::OnBnClickedBtnRead)  ON\_BN\_CLICKED(IDC\_BTN\_READ\_AGE, &CLesson26egistryopDlg::OnBnClickedBtnReadAge)  END\_MESSAGE\_MAP()  // CLesson26egistryopDlg 消息处理程序  BOOL CLesson26egistryopDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CLesson26egistryopDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CLesson26egistryopDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CLesson26egistryopDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CLesson26egistryopDlg::OnBnClickedOk()  {  // TODO: 在此添加控件通知处理程序代码  if(IDOK == ::MessageBox(this->m\_hWnd,\_T("退出程序？"),\_T("退出提示"),MB\_OKCANCEL)){  CDialogEx::OnOK();  }  }  void CLesson26egistryopDlg::OnBnClickedBtnCpu()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64];  DWORD dwSize;  //有坑， RegQueryValueEx最后一个参数需要先初始化  dwSize = sizeof(szData);  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0"),  0,KEY\_QUERY\_VALUE, &hKey);  if(lRet == ERROR\_SUCCESS)  {  //获取对应的键的值  if(ERROR\_SUCCESS == RegQueryValueEx(hKey,\_T("ProcessorNameString"),NULL,NULL,(LPBYTE)szData,&dwSize))  {  SetDlgItemText(IDC\_CPU\_INFO,szData);  }  else  {  MessageBox(\_T("获取键值失败"));  }  }  else  {  MessageBox(\_T("查询注册表键失败"));  }  RegCloseKey(hKey);//查关闭句柄  }  void CLesson26egistryopDlg::OnBnClickedBtnCreate()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64],szSubKey[64],szValue[64];  DWORD dwSize,dwValueSize;  dwSize = sizeof(szData);//有坑，需要用sizeof初始化  dwValueSize = sizeof(szValue);//有坑，需要用sizeof初始化  lstrcpy(szSubKey,\_T("SOFTWARE\\KENNYCAI\\MyApp"));  lstrcpy(szValue,\_T("Smart Software,version 1.0"));  //我们用HKEY\_CURRENT\_USER来创建注册表键  //用HKEY\_LOCAL\_MACHINE也ok，不过设置在HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node\下面  //if(ERROR\_SUCCESS == RegCreateKey(HKEY\_CURRENT\_USER,szSubKey,&hKey))  if(ERROR\_SUCCESS == RegCreateKey(HKEY\_LOCAL\_MACHINE,szSubKey,&hKey))  {  //MessageBox(\_T("创建注册表键成功"));  //创建注册表key后需要设置value，使用RegSetValue函数来实现  if(ERROR\_SUCCESS == RegSetValueEx(hKey,\_T("SoftwareName"),NULL,REG\_SZ,  (LPBYTE)szValue,dwValueSize))  {  if(ERROR\_SUCCESS == RegQueryValueEx(hKey,\_T("SoftwareName"),NULL,NULL,(LPBYTE)szData,&dwSize))  {  SetDlgItemText(IDC\_STATIC\_RESULT,szData);  }  }  else  {  MessageBox(\_T("设置注册表键值失败"));  }  }  else  {  MessageBox(\_T("创建注册表键失败"));  }  RegCloseKey(hKey);  }  void CLesson26egistryopDlg::OnBnClickedBtnWrite()  {  // TODO: 在此添加控件通知处理程序代码  long lRet;  HKEY hKey;  TCHAR szData[64],szValue[64];  DWORD dwSize,dwValueSize;;  //有坑， RegQueryValueEx最后一个参数需要先初始化  dwSize = sizeof(szData);  dwValueSize = sizeof(szValue);//有坑，需要用sizeof初始化  lstrcpy(szValue,\_T("This is a test Value."));  /\*lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\WOW6432Node\\KENNYCAI"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);\*/  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  \_T("SOFTWARE\\GoodZip"),  0,KEY\_QUERY\_VALUE|KEY\_WRITE,&hKey);  if(lRet == ERROR\_SUCCESS)  {  //下面的函数，如果需要为一个key设置默认值，需要把第二个参数也设置为NULL  lRet = RegSetValueEx(hKey,\_T("TestKey"),NULL,REG\_SZ,(LPBYTE)szValue,dwValueSize);  if(lRet == ERROR\_SUCCESS)  {  MessageBox(\_T("设置注册表键成功"));  }  else  {  MessageBox(\_T("设置注册表键失败"));    }  //还可以设置DWORD值,第5个参数是用指针的，如果不是，设置失败  DWORD dwAge = 40;  RegSetValueEx(hKey,\_T("Age"),NULL,REG\_DWORD,(LPBYTE)&dwAge,4);  }  else  {  MessageBox(\_T("打开注册表键失败"));  }  RegCloseKey(hKey);  }  void CLesson26egistryopDlg::OnBnClickedBtnRead()  {  // TODO: 在此添加控件通知处理程序代码  LONG lValue;  RegQueryValue(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),NULL,&lValue);  TCHAR \*buf = new TCHAR[lValue] ;  RegQueryValue(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),buf,&lValue);  SetDlgItemText(IDC\_STATIC\_RESULT,buf);  }  void CLesson26egistryopDlg::OnBnClickedBtnReadAge()  {  // TODO: 在此添加控件通知处理程序代码  HKEY hKey;  RegOpenKey(HKEY\_LOCAL\_MACHINE,\_T("SOFTWARE\\GoodZip"),&hKey);  DWORD dwType,dwValue,dwAge;  RegQueryValueEx(hKey,\_T("Age"),0,&dwType,(LPBYTE)&dwAge,&dwValue);  SetDlgItemInt(IDC\_STATIC\_RESULT,dwAge);  RegCloseKey(hKey);  } |

# 扩展:宽字符命令行版本

|  |
| --- |
|  |

## regdemo.cpp

|  |
| --- |
| #include<Windows.h>  #include<iostream>  #include<locale.h>  void testCreateKey(HKEY hkey, const WCHAR\* szsubkey, const WCHAR\* szsubkeyValue)  {  HKEY hKey;  WCHAR szSubKey[64];  DWORD dwSize, dwValueSize;  WCHAR szValue[64];  dwSize = sizeof(szSubKey);  dwValueSize = sizeof(szValue);  lstrcpy(szSubKey, szsubkey);  lstrcpy(szValue, szsubkeyValue);  if (ERROR\_SUCCESS ==RegCreateKey(HKEY\_CURRENT\_USER, szSubKey, &hKey))  {  wprintf\_s(L"创建子键:%s成功\n", szSubKey);  if (ERROR\_SUCCESS == RegSetValueEx(hKey, L"address",0, REG\_SZ,(LPBYTE)szValue, dwValueSize))  {  wprintf\_s(L"设置子键值成功\n");  }  else  {  wprintf\_s(L"设置子键值失败:%d\n", GetLastError());  return;  }  }  else  {  wprintf\_s(L"创建子键失败:%d\n", GetLastError());  return;  }  }  void ReadCpuInfoFromReg()  {  long lRet;  HKEY hKey;  TCHAR szData[64];  DWORD dwSize;  dwSize = sizeof(szData);  //打开HKEY  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",0,KEY\_QUERY\_VALUE,&hKey);  if (lRet == ERROR\_SUCCESS)  {  if (ERROR\_SUCCESS == RegQueryValueEx(hKey,L"ProcessorNameString",0,NULL,(LPBYTE)&szData,&dwSize))  {  wprintf\_s(L"Processor Name: %s\n", szData);  }  else  {  wprintf\_s(L"获取键值失败\n");  }  }  else  {  wprintf\_s(L"打开键失败\n");  }  }  void RegReadValue(HKEY hkey,const WCHAR\* szsubkey) {  long lvalue;  WCHAR\* lpBuf;  RegQueryValue(hkey, szsubkey, NULL, &lvalue);  lpBuf = new WCHAR[lvalue];  if (ERROR\_SUCCESS == RegQueryValue(hkey, szsubkey, lpBuf, &lvalue))  {  wprintf\_s(L"读取的值是: %s\n",lpBuf);  }  else  {  wprintf\_s(L"读取键值失败:%d\n",GetLastError());  delete[] lpBuf;  return;  }  delete[] lpBuf;  }  void TestDeleteKey(HKEY hkey, const WCHAR\* szsubkey)  {  if (ERROR\_SUCCESS == RegDeleteKey(hkey, szsubkey))  {  wprintf\_s(L"删除子键成功\n");  }  else  {  wprintf\_s(L"删除子键失败:%d\n", GetLastError());  }  }  int main()  {  setlocale(LC\_ALL, "");  //ReadCpuInfoFromReg();  testCreateKey(HKEY\_CURRENT\_USER, L"SOFTWARE\\KENNYCAI\\Website",L"https://www.kennysoft.com/index");  //RegReadValue(HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\GoodZip");  //TestDeleteKey(HKEY\_CURRENT\_USER, L"SOFTWARE\\KENNYCAI\\Website");  //TestDeleteKey(HKEY\_CURRENT\_USER, L"SOFTWARE\\KENNYCAI\\MyApp");  system("pause");  return 0;  } |

# 这一节的学习到此为止，下一次复习有想法可以继续更新