**注册表值类型**

*注册表值*可以采用多种格式之一存储数据，例如字符串值或整数值。 在注册表值中存储数据（例如通过调用 [**RegSetValueEx**](https://learn.microsoft.com/zh-cn/windows/win32/api/Winreg/nf-winreg-regsetvalueexa) 函数）时，可以通过指定下表中的一种类型来指示所存储的数据类型。 检索注册表值时， [**RegQueryValueEx**](https://learn.microsoft.com/zh-cn/windows/win32/api/Winreg/nf-winreg-regqueryvalueexa) 等函数使用这些类型来指示检索的数据的类型。

头文件中定义了 [winnt.h](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt) 以下注册表值类型：

展开表

| **值** | **类型** |
| --- | --- |
| REG\_BINARY | 任意格式的二进制数据。 |
| REG\_DWORD | 32 位数字。 |
| REG\_DWORD\_LITTLE\_ENDIAN | 采用 [little-endian 格式](https://learn.microsoft.com/zh-cn/windows/win32/sysinfo/registry-value-types#byte-formats)的 32 位数字。 Windows 设计为在 little-endian 计算机体系结构上运行。 因此，此值在 Windows 头文件中定义为 REG\_DWORD 。 |
| REG\_DWORD\_BIG\_ENDIAN | [采用 big-endian 格式](https://learn.microsoft.com/zh-cn/windows/win32/sysinfo/registry-value-types#byte-formats)的 32 位数字。 某些 UNIX 系统支持 big-endian 体系结构。 |
| REG\_EXPAND\_SZ | 一个以 null 结尾的字符串，其中包含对环境变量（例如 *%PATH%*）的未扩展引用。 它是 Unicode 或 ANSI 字符串，具体取决于是使用 Unicode 函数还是 ANSI 函数。 若要展开环境变量引用，请使用 [**ExpandEnvironmentStrings**](https://learn.microsoft.com/zh-cn/windows/win32/api/processenv/nf-processenv-expandenvironmentstringsa) 函数。 |
| REG\_LINK | 一个以 null 结尾的 Unicode 字符串，其中包含通过调用具有 **REG\_OPTION\_CREATE\_LINK** 的 [**RegCreateKeyEx**](https://learn.microsoft.com/zh-cn/windows/win32/api/Winreg/nf-winreg-regcreatekeyexa) 函数创建的符号链接的目标路径。 |
| REG\_MULTI\_SZ | 以 null 结尾的字符串序列，由空字符串终止， (\0) 。 下面是一个示例： String1\0String2\0String3\0LastString\0\0。 第一个 \0 终止第一个字符串，第二个终止 \0 最后一个字符串，最后一个 \0 终止序列。 请注意，最终终止符必须计入字符串的长度。 |
| REG\_NONE | 没有定义的值类型。 |
| REG\_QWORD | 64 位数字。 |
| REG\_QWORD\_LITTLE\_ENDIAN | 采用 little-endian 格式的 64 位数字。 Windows is designed to run on little-endian computer architectures. Therefore, this value is defined as REG\_QWORD in the Windows header files. |
| REG\_SZ | 以 null 终止的字符串。 It's either a Unicode or an ANSI string, depending on whether you use the Unicode or ANSI functions. |

**字符串值**

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 字符存储。 因此，从注册表读取字符串时，必须确保在使用前正确终止该字符串;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。

将字符串写入注册表时，必须指定字符串的长度，包括终止 null 字符 (\0) 。 一个常见错误是使用 strlen 函数来确定字符串的长度，但忘记仅 strlen 返回字符串中的字符计数，而不计算终止 null。 因此，应使用 计算字符串的长度 strlen(string) + 1

字符串 REG\_MULTI\_SZ 以长度 *为 0* 的字符串结尾。 因此，无法在序列中包含零长度字符串。 空序列的定义如下： \0。

以下示例遍执行字符串 REG\_MULTI\_SZ 操作。

C++复制

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

void SampleSzz(PTSTR pszz)

{

\_tprintf(\_TEXT("\tBegin multi-sz string\n"));

while (\*pszz)

{

\_tprintf(\_TEXT("\t\t%s\n"), pszz);

pszz = pszz + \_tcslen(pszz) + 1;

}

\_tprintf(\_TEXT("\tEnd multi-sz\n"));

}

int \_\_cdecl main(int argc, char \*\*argv)

{

// Because the compiler adds a \0 at the end of quoted strings,

// there are two \0 terminators at the end.

\_tprintf(\_TEXT("Conventional multi-sz string:\n"));

SampleSzz(\_TEXT("String1\0String2\0String3\0LastString\0"));

\_tprintf(\_TEXT("\nTest case with no strings:\n"));

SampleSzz(\_TEXT(""));

return 0;

}

**字节格式**

在 *little-endian 格式*中，多字节值存储在内存中，从最低字节 (*小端*) 到最高字节。 例如，该值 0x12345678 以 little-endian 格式存储 0x78 0x56 0x34 0x12 为 。

在 *big-endian 格式*中，多字节值存储在内存中，从最高字节 (*大端*) 到最低字节。 例如，该值 0x12345678 以 big-endian 格式存储 0x12 0x34 0x56 0x78 为 。