# 一．循序渐进全球化

## 导言

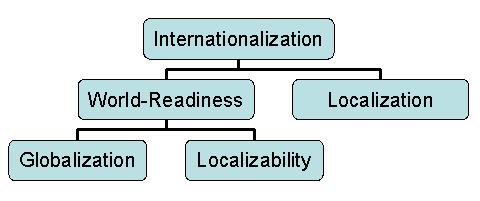
|  |
| --- |
| 本指南旨在帮助应用程序开发人员和其他人员了解全球化应用程序需要执行的步骤。本指南中的每篇文章都包括两部分：   * 概述和说明：围绕学员想要了解的领域对问题进行简介和讨论。它适合项目经理。 * 解决方案和代码示例：与开发和编程相关，适合想要深入了解此领域内容的人员。这部分内容概述了用于 Win32、.NET Framework、Web 或控制台编程的技术。其中包含详细说明、实际示例、解决方案、代码示例、到其他引用的链接或涉及的某些项目的组合。 |

**二．**开发国际化产品是一种通盘兼顾的连续行为。要创建全球通用的单二进制应用程序以供多个不同市场使用，或产品高质量的外语版本，需要处理大量的细节问题，而开发人员及其上司经常对这项工作的强度估计不足，或重视程度不够。如果您是一名开发人员，请确保您的上司了解其中的细节。熟悉出现的问题将有助于您在产品周期开始时做出正确决策，毋容置疑，这会为您节省时间和资金。

开发国际化软件包含两个方面（请参阅下图）：

* **全球通用性。**此方面涉及通用编码和设计问题，其中包括两个主要领域：全球化和可本地化性。
* **本地化。**此方面包括针对某个特定的市场对产品进行翻译和定制。

例如，使用由 Microsoft Win32 应用程序编程接口 (API) 提供的国家/地区语言支持 (NLS) 属于全球通用性中的步骤，而修改用户界面 (UI) 元素、翻译文本和标准化术语则是本地化中的步骤。由于开发人员要编写代码，因此他们趋向于主要关注全球通用性问题。但由于代码和功能设计会影响产品的翻译和定制方式，因此开发人员也必须了解基本的本地化概念。



**术语表**

**全球化：**

开发某个程序核心的过程，其功能和代码设计不只依赖于某个单一的语言或区域设置。其设计开发目标是支持 Unicode 的一组预定义语言脚本和特定区域设置相关数据的输入、显示和输出。

**可本地化性：**

软件代码库和资源的一类设计，使某个程序无需更改任何源代码即可本地化为不同的语言版本。

## 三．概述和说明

创建 Unicode 编码标准是由诸多因素所致，例如，使用混合字节编码所需的复杂编程方法、每次在另一种语言需要计算机支持时创建新代码页所涉及的过程，以及跨不同系统以多种语言混合和共享信息等。Unicode 最初是由 Xerox 和 Apple 联手打造的。随后几家公司形成了临时委员会，其他公司（包括 IBM 和 Microsoft）也迅速加入其中。在 1991 年，此委员会成立了“Unicode Consortium”，该机构的成员现在包括几大信息技术 (IT) 公司。（有关 Unicode 的详细信息，请访问 Unicode Consortium 的站点，网址为[http://www.Unicode.org](http://www.unicode.org/)）。

Unicode 尤其适合于 Internet 时代，因为 Internet 的全球特性要求解决方案可在任何语言下工作。万维网联盟 (W3C) 已认识到这一现实，现正要求所有新 RFC 均使用 Unicode 文本。许多其他产品和标准目前要求或允许使用 Unicode；例如，XML、HTML、Microsoft Jscript、Java、Perl、Microsoft C# 和 Microsoft Visual Basic 7 (VB.NET)。如今，Unicode 被所有主要计算机公司接受的非官方字符编码标准，而 ISO 10646 被所有 ISO 成员国视为全球法定标准。两个标准包括相同的字符库和二进制表示。

Unicode 实际上包含当今在计算机中广泛使用的所有字符。它能够编制 110 多万个码位。该标准包括针对 8 位、16 位和 32 位编码形式所做的规定。16 位编码为其默认编码，超过百万的码位跨 17 个“平面”分布，每个平面可编制 65,000 多个字符。平面 0（或通常称为“基本多文种平面”(BMP)）中的字符用于表示世界上的大部分书面文字、出版中使用的字符、数学和技术符号、几何形状、基本标志（包括所有 100 级 Zapf Dingbat）以及标点符号。除支持流行语言字符以及刚才所提的符号和形状外，Unicode 还包括其他字符，如普及性低一些的中文、日语和韩语 (CJK) 象形文字、阿拉伯语表示形式以及音乐符号。上述许多字符都使用名为“代理项对”的扩展机制在原始平面之外进行映射。Unicode 3.2 已为 95,000 多个码位分配了字符；其余码位留待将来使用。Unicode 还为应用程序提供了有 131,000 多个位置的专用区，供用户定义字符使用（通常是代表人名或地名的稀有象形文字）。

下图以抽象形式显示了 BMP（平面 0）的 Unicode 编码布局。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.f03tm07(zh-cn).jpg |
| **图 1：**BMP（平面 0）的 Unicode 编码布局 |

然而，Unicode 规则在码位分配方面非常严格 – 每个码位都有不同的表示。此外，Unicode 有时刻意不提供码位。现有字符的变体并未被赋予单独的码位，因为这样做会造成同一字符下内容的编码重复。示例是字体变体（例如粗体和斜体）及象形文字变体，它们基本上是相同字符的不同表示方法。

在绝大多数情况下，Unicode 为一个字符定义一种编码，但某些字符可能会组合在一起形成其他字符，如带重音符的字符。最常见的带重音符字符（如法语、德语和许多其他欧洲语言）存在于它们的预构成形式中，而且分配有码位。可通过将基本字符与一个或多个非空格音调符号标记相组合来表示这些相同的字符。例如，后跟非空格重音符标记的 "a" 显示为 "à"。有了非空格重音符标记，就能表达众多带重音符的字符，而不必为其分配不同的码位。这对于表示生涩书面语言（如某些非洲语言）中带重音符的字符很有用。并有利于创造各种数学符号。预构成字符的编码采用基本与其他编码兼容的 Unicode 标准。Unicode 标准包含严格的规则，用于确定预构成字符相对组合字符序列的等效性。Win32 API 函数 FoldStringW 将多个组合字符映射为预构成形式。此外，MultiByteToWideChar 可与 MB\_PRECOMPOSED 或 MB\_COMPOSITE 标志配合使用，以将字符映射为它们的预构成形式或复合形式。

Unicode 标准凭这些优点还不足以成为国际化中的多面手。Unicode 元素的码位位置并不暗示排序顺序，Unicode 也不编码字体信息。定义这些规则的是操作系统，例如在基于 Win32 的应用程序中，需要从操作系统获取排序和字体信息。

另外，基于 Unicode 标准设计软件只是国际化过程中的一步。您仍需编写适应首选地域或语言规则的代码。（有关全球化注意事项的其他详细信息，请参阅 [“区域设置模型”](http://msdn.microsoft.com/zh-cn/goglobal/bb688121)、 [“输入、显示和输出”](http://msdn.microsoft.com/zh-cn/goglobal/bb688132)和 [“多语言用户界面 [MUI]”。](http://msdn.microsoft.com/zh-cn/goglobal/bb688115)）

需要进一步指出的是，并非所有基于 Unicode 的文本处理都是简单的逐字符分析。基于复杂文本的操作（如断字、断行和构成象形文字）需要考虑它们所处的上下文（例如，与周围字符的关系）。这些操作的复杂性取决于语言规则，与作为编码标准的 Unicode 毫无关系。而软件实现应定义更高级别的协议以处理这些操作。

与此相比，有一些不寻常的字符有非常特定的语义规则；这些字符将在 Unicode 标准中详细介绍。一些字符总是允许断行（如大多数空格），而其他字符从不允许此种情况（例如，非空格或非断开字符）。仍有其他字符（包括许多在阿拉伯语和希伯来语中使用的字符）被定义为具有强或弱的文本方向性。Unicode 标准定义一个用于确定双向文本显示顺序的算法，对于不由隐式双向排序规则处理的情况，它还将几个“方向格式化代码”定义为替换值以帮助创建易于理解的双向文本。这些格式化代码允许字符以逻辑顺序存储，但根据其方向性相应地进行显示。中性字符（例如标点符号）对邻近强弱字符的方向性做出假设。格式化代码可用于描述嵌入的文本或指定字符的方向性。（有关显示基于 Unicode 的双向文本的详细信息，请参阅《Unicode Standard》一书。）

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.f03tm08(zh-cn).jpg |
| **图 2：**预构成和复合字符 |

您现已了解 Unicode 提供的一些功能。下面几节更深入地探讨 Unicode 功能，以提供对您使用 Unicode 标准和编码有所帮助的信息。例如，字节顺序标记 (BOM) 的功能是什么？什么是代理项对，它们如何使您能够从编码 65,000 个字符变为编码 100 多万个其他字符？这些问题及其他问题将在随后几节中加以探讨。

**Unicode 码位的转换**

以二进制格式表示每个 Unicode 码位的方法有多种。下列每项技术使用不同的映射表示每个 Unicode 字符。Unicode 编码包括：

* **UTF-8：**为满足面向字节和基于 ASCII 系统的要求，Unicode 标准定义了 UTF-8。采用 UTF-8 的每个字符最多表示为 4 个字节，其中，第一个字节指示多字节序列中的字节数，从而允许更好地解析字符串。UTF-8 通常用在使用 Internet 协议的传输中以及 Web 内容中。
* **UTF-16：**这是 Unicode 标准的 16 位编码形式，在该形式中，除了由代理项对编码的字符（由一个 16 位值对组成）外，其他字符均被分配一个唯一的 16 位值。Unicode 16 位编码形式与国际标准化组织/国际电工委员会 (ISO/IEC) UTF-16 传输格式相同。在 UTF-16 中，映射值不高于 65,535 的所有字符被编码为一个 16 位值；映射值高于 65,535 的字符被编码为 16 位值对。（有关代理项对的详细信息，请参阅本章后面的“代理项对”。）UTF-16 little-endian 是 Microsoft（以及 Windows 操作系统中）的编码标准。
* **UTF-32：**每个字符均被表示为一个 32 位的整数。

下图显示了使用 UTF-16 和 UTF-8 在代码页与 Unicode 中编码的两个字符。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.f03tm09(zh-cn).jpg |
| **图 3：**使用 UTF-16 和 UTF-8 在代码页与 Unicode 中编码的字符 "A" 和日本汉字字符。 |

由于 UTF-8 常用在 Web 内容中，因此它有助于理解如何将 Unicode 码位映射到此编码中，省去了使用 MBCS 字符的麻烦。表 1 显示了 Unicode 码位和 UTF-8 编码字符之间的关系。UTF-8 编码字符中字节链的起始字节表明使用了多少个字节编码此字符。所有后续字节均以 "10" 开头，连续的 x 指示给定范围内编码的二进制表示。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.utf8(zh-cn).jpg |
| **表 1：**Unicode 码位和 UTF-8 编码字符之间的关系。在 UTF-8 中，第一个字节指示多字节编码序列中的字节数。 |

**字节顺序标记**

您在使用 Unicode 时另一个常出现的概念是字节顺序标记。BOM 用于指示处理器如何将连续文本置于字节序列中。如果将最不重要的字节置于初始位置，这称为 "little-endian"；如果将最重要的字节置于初始位置，此方法称为 "big-endian"。BOM 也可用作识别文本文件编码的参考。例如，记事本根据保存文件时所用的编码，将 BOM 添加到每个文件的开头。此签名将允许记事本在稍后重新打开该文件。表 2 显示了各种编码的字节顺序标记。UTF-8 BOM 标识编码格式而非文档的 BOM – 因为每个字符由一个字节序列表示。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.bom(zh-cn).jpg |
| **表 2：**特定编码的字节顺序标记 (U+FEFF) 的二进制表示。 |

**代理项对**

使用 Unicode 16 位编码系统，可编码 65,000 多个字符 (2^16 = 65536)。然而，需要编码的字符总数实际上已超过了此限制（主要是为了适应字符在 CJK 方面的扩展）。为了替新字符找到位置，Unicode 标准开发人员决定引入代理项对这一概念。使用代理项对后，Unicode 码位的范围变为两类：从 U+D800 到 U+DBFF 称为“高代理项”；从 U+DC00 到 U+DFFF 称为“低代理项”，两者组合生成一个全新的字符，从而可以再编码 100 多万个字符。与 MBCS 字符不同，如高低代理项不是出现在代理项对里，无法对其进行译码（MBCS 文本前导字节和拖尾字节处理的主要挑战之一）。

在 Unicode 3.01 中，这是字符首次在原始 16 位代码空间或 BMP（平面 0）之外编码。在 U+10000 或更高代码位置编码的这些新字符与国际化标准 ISO/IEC 10646-2 同步。除了两个专用区平面 15 (U+F0000 - U+FFFFD) 和平面 16 (U+100000 - U+10FFFD) 外，Unicode 3.1 和 10646-2 还定义三个新的辅助平面：

* 第一辅助平面（Supplementary Multilingual Plane，简称 SMP）– 代码位置从 U+10000 到 U+1FFFF
* 第二辅助平面（Supplementary Ideographic Plane，简称 SIP）– 代码位置从 U+20000 到 U+2FFFF
* 第十四辅助平面（Supplementary Special-purpose Plane，简称 SSP）– 代码位置从 (SSP) U+E0000 到 U+EFFFF

SMP（或平面 1）包含若干古老文字和若干符号组：古意大利文、哥特文、犹太文、东正教音乐符号、（西方）音乐符号和数学字母数字符号。它们一起组成 1,594 个新编码字符。SIP（或平面 2）额外收集了大量统一汉字象形文字（称为“CJK 扩展 B 区”），组成了 42,711 个字符和 542 个 CJK 兼容象形文字。SSP（或平面 14）包含一组标记字符（97 个）。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688113#top)

**创建 Win32 Unicode 应用程序**

除了深入地研究了 Unicode 的功能性之外，您现在还更详细地了解了它所提供的优点和功能。您可能还想知道，Windows 对 Unicode 功能支持到什么程度。Microsoft Windows NT 3.1 是第一个支持 Unicode 的主要操作系统，自那以后，Microsoft Windows NT 4、Microsoft Windows 2000 和 Microsoft Windows XP 扩大了这一支持，使用 Unicode 作为它们的本机编码。实际上，当您在这些操作系统中运行非 Unicode 应用程序时，它们会先在内部将应用程序文本转换为 Unicode，然后再做处理。操作系统随后将文本转换回预期的代码页编码，再将信息传回应用程序。

此外，Windows XP 的字体、键盘驱动程序和其他系统文件（用所有支持语言输入和显示内容所必需的）支持大多数 Unicode 码位。再次重申，在基于 Windows NT 的操作系统中，文本的基本表示是 UTF-16，WCHAR 数据类型是 UTF-16 编码单元。Windows 确实为其他编码提供了接口以便向后兼容，但它会在内部将此类文本转换为 UTF-16。此外，系统还提供了多个接口，以在 UTF-16 和 UTF-8 之间进行转换，并查询 UTF-16 码位的基本属性（例如，它是字母、数字还是标点符号）。由于 Microsoft Windows 95、Microsoft Windows 98 和 Windows Me 并不以 Unicode 为构建基础，因此，与基于 Windows NT 的 Windows 版本相比，它们所提供的 Unicode 支持仅是其中的一小部分。因此，通过使用 Unicode 和基于 Windows NT 的操作系统，您距创建全球通用应用程序这一目标更近了一步。剩余几节将向您展示创建 Win32 Unicode 应用程序的实用技巧和示例，以及在 .NET Framework、控制台或文本模式编程中将编码用于网页的技巧。

将现有基于代码页的应用程序移植到 Unicode 比您想像的要容易。实际上，Unicode 是以这样一种方式实现的：使编写 Unicode 应用程序对开发人员几乎是透明的。Unicode 还需要以另外一种方式实现：即确保无论何时在纯 Unicode 平台中运行非 Unicode 应用程序，它均能正常工作。要适应这些需求，Unicode 实现需要在两个主要领域做出改变：

* 创建处理 16 位字符的数据类型变量 (WCHAR)
* 创建一组 API，接受采用 16 位字符编码的字符串参数

**WCHAR，一个 16 位数据类型**

Unicode 大多数字符串操作的编码逻辑与 Windows 字符集的处理逻辑相同。不同之处在于：操作的基本单元是 16 位量，而非 8 位量。头文件提供了许多类型定义，便于为 Unicode 或 Windows 字符集创建可编译的资源。

For 8-bit (ANSI) and double-byte characters:  
typedef char CHAR; // 8-bit character  
typedef char \*LPSTR; // pointer to 8-bit string  
  
For Unicode (wide) characters:  
typedef unsigned short WCHAR; // 16-bit character  
typedef WCHAR \*LPWSTR; // pointer to 16-bit string

下图显示了 Win32 头文件定义三组类型所依据的方法：

* 一组通用类型定义（TCHAR、LPTSTR），它取决于 \_UNICODE 明示常量的状态。
* 两组显式类型定义（一组用于基于代码页或 ANSI 的类型，另一种用于 Unicode）。

使用通用声明可保持一组源文件并对其进行编译，以支持 Unicode 或 ANSI。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.f03tm12(zh-cn).jpg |
| **图 4：**新数据类型 WCHAR。 |

**Win32 API 的 W 函数原型**

所有采用文本参数作为输入或输出变量的 Win32 API 都配备了一个通用函数原型和两个定义：一个基于代码页或 ANSI 的版本（称为 "A"），用于处理基于代码页的文本参数和一个宽版本（称为 "W "），用于处理 Unicode。通用函数原型包括宏形式的标准 API 函数名称。通用原型解析为一种显式函数原型（"A" 或 "W"），具体取决于编译时明示常量 UNICODE 是否是在 #define 语句中定义。在显式函数原型中，字母 "W" 或 "A" 添加在 API 函数名称的结尾处。

// windows.h  
#ifdef UNICODE  
#define SetWindowText SetWindowTextW  
#else  
#define SetWindowText SetWindowTextA  
#endif // UNICODE

利用此机制，应用程序可使用通用 API 函数来以透明方式处理 Unicode，具体视 #define UNICODE 明示常量而定。通过使用带有 "W" 或 "A" 的显式函数名称，它还可以进行混合调用。

在此双重编译设计中，RegisterClass（以及 RegisterClassEx）是极重要的一个函数。窗口类由窗口过程来实现。可使用 RegisterClassA 或 RegisterClassW 函数来注册窗口过程。通过使用函数的 "A" 版本，程序告知系统所创建类的窗口过程请求具有文本或参数（基于代码页）的消息；创建与该窗口相关联的其他对象时使用 Windows 代码页作为文本编码。通过调用宽字符函数来注册窗口类，程序可请求系统按 Unicode 传递消息的文本参数。IsWindowUnicode 函数允许程序查询每个窗口的特性。

在 Windows NT 4、Windows 2000 和 Windows XP 上，"A" 例程为包装程序，用于将基于代码页或 ANSI 的文本转换为 Unicode（使用系统区域设置代码页），然后调用相应的 "W" 例程。在 Windows 95、Windows 98 和 Windows Me 上，"A" 例程为本机例程，而大多数 "W" 例程都无法实现。如果已调用 "W" 例程但其尚未实现，则会返回 ERROR\_CALL\_NOT\_ IMPLEMENTED 错误消息。（有关如何为非 Unicode 平台编写基于 Unicode 的应用程序的详细信息，请参阅 " [Microsoft Layer for Unicode (MSLU)](http://msdn.microsoft.com/magazine/cc301794.aspx)"）。

**Unicode 文本宏**

Visual C++ 让您使用 "L" 为文字加前缀以指示它是 Unicode 字符串，如下所示：

LPWSTR str = L"This is a Unicode string";

在源文件中，该字符串以编辑器或编译器可以理解的代码页表示。在编译时，字符被转换为 Unicode。Win32 SDK 资源编译器也支持 "L" 前缀表示法，尽管它可以直接解释 Unicode 源文件。WINDOWS.H 定义名为 TEXT() 的宏，它根据 UNICODE 编译标志设置，将字符串文字标记为 Unicode。

#ifdef UNICODE  
#define TEXT(string) L#string  
#else  
#define TEXT(string) string  
#endif // UNICODE

因此，字符字符串的通用版本应是：

LPTSTR str = TEXT("This is a generic string");

**C 运行时扩展**

在 ANSI C 中，Unicode 数据类型与宽字符数据类型 wchar\_t 兼容，因而允许访问宽字符字符串函数。大多数 C 运行时 (CRT) 库都包含 strxxx 字符串函数的宽字符版本。函数的宽字符版本均以 *wcs* 开头。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.crt(zh-cn).jpg |
| **表 3：**用于字符串操作的 C 运行时库例程示例。 |

C 运行时库还提供诸如 *mbtowc* 和 *wctomb* 等函数，它们可在 C 字符集与 Unicode 之间相互转换。Win32 API 更为常规的一组函数可执行与 C 运行时库相同的功能，包括 Unicode、Windows 字符集和 MS-DOS 代码页之间的转换。在 Windows 编程中，强烈建议您使用 Win32 API 代替 CRT 库，以利用系统所提供的区域设置识别功能，如 [使用区域设置模型](http://msdn.microsoft.com/zh-cn/goglobal/bb688121)中所述。

|  |
| --- |
| http://i.msdn.microsoft.com/bb688113.w32crt(zh-cn).jpg |
| **表 4：**C 运行时库例程的等效 Win32 API 函数。 |

**代码页和 Unicode 之间的转换函数**

由于大量应用程序仍以代码页为基础，而且您可能希望在内部支持 Unicode，因此在许多情况下，仍需要在代码页编码和 Unicode 之间进行转换。Win32 API 对（MultiByteToWideChar 和 WideCharToMultiByte）能完成代码页编码与 Unicode 的相互转换。每个上述 API 都将要用于此转换的代码页值作为一个参数。因此，您可以指定给定代码页的值（示例：1256 用于阿拉伯语）或使用预定义标志，例如：

* CP\_ACP：用于当前选定的系统 Windows 代码页
* CP\_OEMCP：用于当前选定的系统 OEM 代码页
* CP\_UTF8：用于 UTF-16 和 UTF-8 之间的转换

（有关详细信息，请参阅 Microsoft 开发人员网络 [MSDN] 文档，网址为 [http://msdn.microsoft.com](http://msdn2.microsoft.com/)）。

通过顺次使用 *MultiByteToWideChar* 和 *WideCharToMultiByte*，并利用同一代码页信息，可执行“往复转换”。如果此编码转换与原始字符串编码中的代码页编号相同，则往复转换应允许您检索初始字符字符串。

**在 Visual C++ 中编译 Unicode 应用程序**

通过使用通用数据类型和函数原型，您可以随意地创建非 Unicode 应用程序或将您的软件编译为 Unicode。要在 Visual C/C++ 中将应用程序编译为 Unicode，请转到 Project/Settings/C/C++ /General 并将 UNICODE 和 \_UNICODE 包括在 Preprocessor Definitions 内。UNICODE 标志是所有 Win32 API 和数据类型的预处理器定义，而 \_UNICODE 是 C 运行时函数的预处理器定义。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688113#top)

**迁移至 Unicode**

基于 Unicode 创建新程序相当容易。Unicode 有几个功能需要特殊处理，但您可在代码中将它们隔离。将使用代码页编码的现有程序转换为使用 Unicode 或通用声明的程序也非常简单。下面是需要执行的步骤：

1. **修改您的代码以使用通用数据类型。**确定被声明为 *char* 或 *char\** 的变量是文本，而不是指向缓冲区或二进制字节数组的指针。将这些类型更改为 TCHAR 和 TCHAR\*（如 Win32 文件 WINDOWS.H 中所定义的），或更改为 \_TCHAR（如 Visual C++ 文件 TCHAR.H 中所定义的）。使用 LPTSTR 和 LPTCH 替换 LPSTR 和 LPCH 的实例。确保检查所有局部变量并返回类型。使用通用数据类型是一个很好的转换策略，因为您可以编译您程序的 ANSI 和 Unicode 版本，而不影响代码的可读性。不过，对于始终为 Unicode 或始终保持在给定代码页中的数据，请勿使用通用数据类型。例如，*MultiByteToWideChar* 和 *WideCharToMultiByte* 的字符串参数之一应始终为基于代码页的数据类型，而其他参数应始终为 Unicode 数据类型。
2. **修改您的代码以使用通用函数原型。**例如，使用 C run-time call \_*tcslen* 代替 *strlen*，使用 Win32 API *SetWindowText* 代替 *SetWindowTextA*。此规则适用于处理文本参数的所有 API 和 C 函数。
3. **使用 TEXT 宏修饰任意字符或字符串文字。**TEXT 宏有条件地将 "L" 置于字符文字或字符串文字定义的前面。请注意转义序列。例如，Win32 资源编译器将 L/" 解释为指定 16 位 Unicode 双引号字符的转义序列，而非 Unicode 字符串的开头。
4. **创建通用数据结构。**结构中字符串或字符字段的类型定义应根据 UNICODE 编译时标志正确地进行解析。如果您编写自己的字符串处理和字符处理函数或采用字符串作为参数的函数，请创建它们的 Unicode 版本并为其定义通用原型。
5. **更改您的构建过程。**如果您希望构建 Unicode 版本的应用程序，必须同时定义 Win32 编译时标志 -DUNICODE 和 C 运行时编译时标志 -D\_UNICODE。
6. **调整指针算法。**减去 char\* 值会产生针对字节的答案；减去 wchar\_t\* 值会产生针对 16 位块的答案。确定字节数时（例如，为字符串分配内存时），将符号中字符串的长度乘以 *sizeof(TCHAR)*。根据字节数确定字符数时，除以 *sizeof(TCHAR)*。您还可以为这两个操作创建宏。C 可确保 ++ 和 -- 运算符按数据类型的大小递增和递减。如果使用 Win32 API *CharNext* 和 *CharPrev*，效果会更好。
7. **检查假定字符始终为 1 字节长的任何代码。**必须更改假定字符值始终小于 256 的代码（例如，将字符值用作大小为 256 的表格索引的代码）。确保您的 NULL 定义为 16 位长。
8. **添加代码以支持特殊的 Unicode 字符。**此类字符包括兼容区域中的 Unicode 字符、专用区中的字符、组合字符以及具有方向性的字符。其他特殊字符包括专用区非字符 U+FFFF（可用作占位符）以及字节顺序标记 U+FEFF 和 U+FFFE（Unicode 存储文件的指示标志）。字节顺序标记用于指示文本流是 little-endian 还是 big-endian。在纯文本中，行分隔符 U+2028 标记行的无条件结尾。在段落之间插入段落分隔符 U+2029 可更轻松地用不同行宽布置文本。
9. **通过启用您编译器的类型检查来调试移植。**无论是否定义了 UNICODE 标志均请执行此操作。某些在基于代码页的环境中可忽略的警告将导致 Unicode 出现问题。如果您的原始代码在启用类型检查时编译清晰，则它更易于移植。警告将帮助您确保不将错误的数据类型传递到要求宽字符数据类型的代码。使用 Win32 国家/地区语言支持 API (NLS API) 或等效 C 运行时调用来获取字符键入和排序信息。请勿试图编写您自己的逻辑来处理特定于区域设置的类型检查 – 您的应用程序将生成大的惊人的表格！

在以下示例中，字符串从资源加载并用在两种方案中：

* 作为消息框的正文
* 在运行时于给定窗口中书写

为了简化，此示例将忽略定义无关变量的位置和方法。假设您要将下列基于代码页的代码迁移到 Unicode：

char g\_szTemp[MAX\_STR]; // Definition of a char data type  
  
// Loading IDS\_SAMPLE from the resources in our char variable  
LoadString(g\_hInst, IDS\_SAMPLE, g\_szTemp, MAX\_STR);  
  
// Using the loaded string as the body of the message box  
MessageBox(NULL, g\_szTemp, "This is an ANSI message box!", MB\_OK);  
  
// Using the loaded string in a call to TextOut for drawing at  
// run time  
ExtTextOut(hDC, 10, 10, ETO\_CLIPPED , NULL, g\_szTemp,  
strlen(g\_szTemp), NULL);

将此代码迁移到 Unicode 很容易，就象采用通用编码惯例并正确替换数据类型、Win32 API 和 C 运行时 API 定义一样。您可以看到粗体字样的更改。

#include  
// Include wchar specific header file  
TCHAR g\_szTemp[MAX\_STR]; // Definition of the data type as a  
// generic variable  
  
// Calling the generic LoadString and not W or A versions explicitly  
LoadString(g\_hInst, IDS\_SAMPLE, g\_szTemp, MAX\_STR);  
  
// Using the appropriate text macro for the title of our message box  
MessageBox(NULL, g\_szTemp, TEXT("This is a Unicode message box."),  
MB\_OK);  
  
// Using the generic run-time version of strlen  
ExtTextOut(hDC, 10, 10, ETO\_CLIPPED , NULL, g\_szTemp,  
\_tcslen(g\_szTemp), NULL);

完成这些简单步骤后，只需定义编译标志 UNICODE 和 \_UNICODE，将您的代码编译为 Unicode，即可创建 Unicode 应用程序。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688113#top)

**迁移至 Unicode 的选项**

根据您的需求和目标操作系统，可选择几个选项从基于代码页的应用程序进行迁移，或迁移到基于 Unicode 的应用程序。不过，一些这样的选项确实有需要注意的具体问题。

* 创建两个二进制：Windows 95、Windows 98 和 Windows Me 的默认编译，以及 Windows NT、Windows 2000 和 Windows XP 的默认编译。  
  缺点：保持两个版本的软件有些杂乱，而且违背单一全球二进制的原则。
* 始终注册为非 Unicode 应用程序，必要时与 Unicode 相互转换。  
  缺点：由于 Windows 不支持自定义代码页的创建，因此您不能使用仅通过 Unicode 支持的文字（如印度语言系列、亚美尼亚语和格鲁吉亚语中的文字）。另外，此选项使多语言计算变为不可能，因为在显示时，您的应用程序始终受限于系统的代码页。
* 创建纯 Unicode 应用程序。  
  缺点：它仅对 Windows NT、Windows 2000 和 Windows XP 有效，因为旧平台上所提供的 Unicode 支持有限。如果您仅以 Unicode 平台为目标，此为首选方法。
* 使用 Microsoft Layer for Unicode (MSLU)。在这个便捷的方法中，您只是创建一个纯 Unicode 应用程序，然后将 SDK 平台提供的 Unicows.lib 文件链接至您的项目即可。您还需要提供 Unicows.dll 文件和可交付内容。MSLU 实质上是在运行时将您代码中调用的所有显式 "W" 版本封装为 "A" 版本（如果在运行时检测到非 Unicode 平台）。对于向 Unicode 迁移并确保向后兼容性而言，此方法是目前最好的解决方案。（有关详细信息，请参阅 " [MSLU](http://msdn.microsoft.com/magazine/cc301794.aspx)"）。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688113#top)

**最佳实践**

编写 Unicode 代码时，有许多方面要考虑，例如何时使用 UTF-16、何时使用 UTF-8、使用什么进行压缩等等。以下为建议使用的实践方法，有助于确保您根据所处环境选择最佳方法。

* 选择 UTF-16 作为您应用程序中文本的基本表示。UTF-8 应仅用于应用程序互操作性（例如，用于发送给不支持 Unicode 的浏览器进行显示的内容，或通过不支持 Unicode 的网络和服务器发送的内容）。避免逐字节处理并尽可能地使用现有 WCHAR 系统接口和资源。某些语言中字符之间的交互需要有关这些语言的专业知识。Microsoft 已使用由 Unicode 表示的大多数语言开发和测试了系统接口 – 除非您是多语言专家，否则重现此支持将非常困难。
* 如果您的应用程序必须在 Windows 95、Windows 98 或 Windows Me 上运行，请主要使用 UTF-16 表示基本文本并在这些操作系统上使用 MSLU。如果您必须支持非 Unicode 文本，请以 UTF-16 将数据保持在内部，并通过网关转换为其他编码。必要时，使用系统接口（如 MultiByteToWideChar）进行转换。确保您的应用程序支持需要两个 UTF-16 码位（代理项对）的 Unicode 字符。如果您使用现有系统接口，这一点应能自动实现；但是，如果您不使用现有系统接口，则应仔细地进行开发和测试。避免误将代理项对视为旧式东亚双字节编码 (DBCS)。将所需的字符串操作集中在几个子例程中。这些子例程应该考虑到代理项对，还应处理组合字符和其他需要特殊处理的字符。编写良好的应用程序可以将代理项处理限制在为数不多的几个这样的例程中。请勿使用 UTF-8 进行压缩 – 对于大多数语言而言，它实际上是扩展数据的大小。如果您需要 Unicode 的实际压缩算法，请参阅 Unicode Consortium 在其站点上提供的技术标准“Unicode 技术标准 6：Unicode 的标准压缩方案”，网址为 [http://www.unicode.org](http://www.unicode.org/)。
* 不要为了避免处理代理项而只选择 UTF-32。那样，数据大小将加倍，而且处理优势难以实现。如果您遵循先前在代理项处理方面的建议，则 UTF-16 应足以满足要求。
* 使用不相关的语言（例如阿拉伯语、印地语和韩语）混合来测试您的 Unicode 支持。对于编写良好的 Unicode 应用程序，系统区域设置应不受影响，通过测试验证这一事实。

您现已了解创建 Win32 Unicode 应用程序的技巧和代码示例。Unicode 对于在全球环境和市场中处理 Web 内容也极其有用。一旦知道如何在网页中处理编码，就相当于在当今 Web 内容中所使用的多种语言之间架起了一座桥梁。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688113#top)

**引用**

* MSDN 对应用程序中 [Unicode 使用方法](http://msdn.microsoft.com/library/ms776440.aspx) 的说明。
* 可通过下列网址在线访问 Unicode Consortium： [http://www.unicode.org](http://www.unicode.org/)
* [将 Microsoft Layer for Unicode 用于 Win 9x/ME](http://msdn.microsoft.com/magazine/cc301794.aspx)

**四.概述和说明**

代码页是一个选定字符代码的列表，字符表示为以某个特定的顺序排列的码位。代码页通常会定义为支持特定的语言或那些共享公用书写系统的语言组。所有的 Window 代码页只能包含 256 个码位。大多数前 127 个码位都表示相同的字符。这是为保证连续性和旧代码预留的码位。代码页在后面的 128 个码位 128-255（以 0 为基数）有着显著的不同。

例如，代码页 1253 为希腊语书写系统提供所需的字符代码，而代码页 1250 为拉丁语书写系统（包括英语、德语和法语）提供字符。后 128 个码位中包含重音字符或希腊语字符。因此不能将希腊语和德语存储在同一代码流中，除非您放置某些类型的标识符来指明所引用的代码页。

在处理亚洲字符集时情况更为复杂。由于中文、日语和韩语包含的字符超过 256 个,，因而需要开发一种不同的方案，但该方案必须以 256 个字符的代码页为基础。因此 DBCS（双字节字符集）应运而生。

每个亚洲字符都由一对码位表示（因此为双字节）。为支持编程，将留出一组码位表示该集的第一个字节但不赋值（除非后面紧跟已定义的第二个字节）。DBCS 意味着您必须编写可将这些码位对视为一个码位的代码，但这样仍无法在同一数据流中组合日语和中文，因为在不同代码页中，相同的双字节码位代表不同的语言中的不同字符。

为了在同一数据流中存储不同的语言，需要创建 [Unicode](http://msdn.microsoft.com/zh-cn/goglobal/bb688113)。该“代码页”可表示 64000 多个字符，现在由于引入了代理项，它可以表示 1,000,000,000 多个字符。Windows 2000 中使用了 Unicode 后可轻松地创建全球通用的代码，因为您无需再担心寻址哪个代码页，也不必再担心是否需要归组字符码位来表示一个字符。

请注意，在为 Win95/98/ME 编写 Unicode 应用程序时，您仍需要将 Unicode 码位转换回 Window 代码页。这是因为 Win95/98/ME GDI 仍然是基于 ANSI 的。但可使用 **WideChartoMultiByte** 和**MultiByteToWideChar** 函数简化此操作。请参阅 MSDN 上的“ [Unicode 和字符集](http://msdn.microsoft.com/library/ms776440.aspx)”。

有关网页中编码的信息，请参阅 MSDN 上的 [**MLang**](http://msdn.microsoft.com/library/aa767865.aspx)。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688114.aspx#top)

**Win32 中的编码**

请参阅 MSDN 上的 [Unicode 和字符集](http://msdn.microsoft.com/library/ms776440.aspx)。

**.NET Framework 中的编码**

.NET Framework 是一个用于构建、部署和运行 Web 服务及应用程序的平台，它提供了基于标准的高效多语言环境，可将现有或原有的投资与下一代应用程序和服务集成到一起。.NET Framework 用 Unicode UTF-16 表示字符，但在某些情况下其内部使用 UTF-8。System.Text 命名空间提供可用于编码和解码字符的类，并包括含以下编码的支持：

* Unicode UTF-16 编码。使用 UnicodeEncoding 类在字符与 UTF-16 编码之间进行转换。
* Unicode UTF-8 编码。使用 UTF8Encoding 类在字符与 UTF-8 编码之间进行转换。
* ASCII 编码。ASCII 将拉丁字母字符集编码为 7 位字符。由于此编码仅支持从 U+0000 到 U+007F 的字符值，因此大多数情况下不适用于国际化应用程序。当您需要与旧编码和系统进行互动时，可以使用 ASCIIEncoding 类在字符和 ASCII 编码之间进行转换。
* Windows/ISO 编码。**System.Text.Encoding** 类提供对多种 Windows/ISO 编码的支持。

.NET Framework 支持使用代码页编码的数据。您可以使用 **Encoding.GetEncoding** 方法 (Int32) 为指定的代码页创建目标编码对象。将某个代码页数码指定为 Int32 参数。以下代码示例为代码页 1252 创建了一个编码对象 enc。

[Visual Basic]  
Encoding enc = Encoding.GetEncoding(1252)[C#]  
Encoding enc = Encoding.GetEncoding(1252);

在创建了与某个指定代码页对应的编码对象后，可使用该对象执行 System.Text.Encoding 类支持的其他操作。

ASP.NET 引入的另一项支持是可清楚地区别文件、请求和响应编码。要在 ASP.NET 中为代码、页面指令和配置文件设置编码，您需要执行以下操作。

在代码中：

Response.ContentEncoding=<value>Request.ContentEncoding=<value>File.ContentEncoding=<value>

在页面指令中：

<%@Page ResponseEncoding=<value>%>  
<%@Page RequestEncoding=<value>%>  
<%@Page FileEncoding=<value>%>

在配置文件中：

<configuration>  
 <globalization  
 fileEncoding=<value>  
 requestEncoding=<value>  
 responseEncoding=<value>  
 />  
</configuration>

以下的 C# 代码示例使用 Encoding.GetEncoding 方法为指定代码页创建目标编码对象。对目标编码对象调用 Encoding.-GetBytes 方法，将 Unicode 字符串转换为其在目标编码中字节表示。随即显示指定代码页中这些字符串的字节表示。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688114.aspx#top)

using System;  
using System.IO;  
using System.Globalization;  
using System.Text;

public class Encoding\_UnicodeToCP  
 {  
 public static void Main()  
 {  
 // Convert ASCII characters to bytes.  
 // Display the string's byte representation in the  
 // specified code page.  
 // Code page 1252 represents Latin characters.  
 PrintCPBytes("Hello, World!",1252);  
 // Code page 932 represents Japanese characters.  
 PrintCPBytes("Hello, World!",932);

// Convert Japanese characters to bytes.  
 PrintCPBytes("\u307b,\u308b,\u305a,\u3042,\u306d",1252);  
 PrintCPBytes("\u307b,\u308b,\u305a,\u3042,\u306d",932);  
 }

public static void PrintCPBytes(string str, int codePage)  
 {  
 Encoding targetEncoding;  
 byte[] encodedChars;  
 // Get the encoding for the specified code page.  
 targetEncoding = Encoding.GetEncoding(codePage);

// Get the byte representation of the specified string.  
 encodedChars = targetEncoding.GetBytes(str);

// Print the bytes.  
 Console.WriteLine  
 ("Byte representation of '{0}' in Code Page '{1}':", str, codePage);  
 for (int i = 0; i < encodedChars.Length; i++)  
 Console.WriteLine("Byte {0}:{1}", i, encodedChars[i]);  
 }  
 }

要确定用于 .NET Framework (ASP.NET) 应用程序“动态服务器页”中响应字符的编码，将 HttpResponse.ContentEncoding 属性的值设置为相应方法的返回值。以下代码示例阐明了如何设置 HttpResponse.ContentEncoding。

// Explicitly set the encoding to UTF-8.  
Response.ContentEncoding = Encoding.UTF8;

// Set ContentEncoding using the name of an encoding.  
Response.ContentEncoding = Encoding.GetEncoding(name);

// Set ContentEncoding using a code page number.  
Response.ContentEncoding = Encoding.GetEncoding(codepageNumber);

最后主要讨论控制台或文本模式编程中的编码。在以下内容中，您将了解到有关使用 Win32 API 和 C 运行时 (CRT) 库函数、CRT 控制台输入/输出 (I/O) 以及 Win32 文本模式 I/O 的信息，帮您处理这类应用程序。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688114.aspx#top)

**网页中的编码**

设置 Web 页面的字符集或编码一般有四种方法。

* 使用此方法时，您可以从支持的代码页列表中进行选择以创建您自己的 Web 内容。这种方法的弊端是您被限定于所选字符集所包括的语言，无法实现真正的多语言 Web 内容。您只能处理单一脚本的 Web 页面。
* 数字实体可用于表示当前选定代码页或编码范围之外的一些符号。例如，假设您决定利用前一种方法和拉丁语 ISO 字符集 8859-1 创建一个 Web 页面。现在您想在某个算式等式中显示一些希腊语字符，但拉丁语代码页中并不包含希腊语字符。例如，选用希腊语字符 Φ（具有 Unicode 码位 U+03A6）。在此码位的十进位制数实体前放置 &#，则该字符的输出如下：This is my text with a Greek Phi:Φ。而输出为：This is my text with a Greek Phi:Φ。遗憾的是此方法无法编排大量文本，使编辑您的 Web 内容变得非常困难。
* 在 Win32 应用程序中，UTF-16 是迄今为止最好的方法，但对于 Web 内容，只能在完全支持 Unicode 的 Windows NT 网络中安全使用 UTF-16。因此，对于那些不清楚是否有客户端 Web 浏览器和网络 Unicode 支持的 Internet 站点，建议不使用此编码。
* 对于多语言 Web 页面，Unicode 编码是最好且最安全的方法。使用此方法可编码整套 Unicode 字符。此外，Internet Explorer 4 和 Netscape 4 以后的所有版本都将支持此编码，此功能不受网络或线路功能的限制。利用 UTF-8 编码无需根据目标语言更改编码即可创建多语言 Web 内容。

|  |
| --- |
| 图 1：用 UTF-8 编码的多语言 Web 页面示例。  **图 1：**用 UTF-8 编码的多语言 Web 页面示例。 |

**设置和处理编码**

由于当前 Web 内容是基于 Windows 或其他编码方案的，因此您需要了解如何设置和处理编码。以下内容说明了如何在 HTML 页面、动态服务器页 (ASP) 和 XML 页面中执行这些操作。

**HTML 页面**

Internet Explorer 使用针对文档指定的字符集来确定如何将文档中的字节转译为屏幕或纸张上的字符。默认情况下，Internet Explorer 使用由服务器返回的 HTTP 内容类型中指定的字符集来确定此转译过程。如果未给定此参数，Internet Explorer 将使用文档中的 meta 元素所指定的字符集，如果未指定任何 meta 元素，则考虑使用用户的首选项。要将某个字符集应用到整个文档中，必须在 body 元素前插入 meta 元素。为清晰起见，它应作为 head 后的第一个元素出现，这样所有的浏览器都可在解析文档前转译该 meta 元素。从而将 meta 应用于其所在的整个文档。这意味着复合文档（由一组框架中的两个或多个文档组成的文档）可在不同的框架中使用不同的字符集。下面介绍它的工作原理：

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=<value>">

您可以替换为任何支持的字符集名称（如 UTF-8）或任何代码页名称（如 Windows 1251）。（有关详细信息，请参阅 MSDN 上的 [字符集识别](http://msdn.microsoft.com/library/aa752010.aspx)）。

**ASP 页面**

在内部，ASP 与其调用的语言引擎（如 Microsoft Visual Basic Scripting Edition (VBScript)、Jscript 等）都是用 Unicode 字符串进行通信的。但是，除了 Unicode 以外，当前组成 Web 页面的内容还可以采用 Windows 或其他字符编码方案。因此，当表单或查询字符串值来自浏览器中的 HTTP 请求时，它们必须从浏览器使用的字符集转换为 Unicode 以便 ASP 脚本进行处理。类似地，当输出发送回浏览器时，由脚本返回的字符串必须从 Unicode 转换为客户端使用的代码页。在 ASP 中，这些内部转换通过 Web 服务器的默认代码页来完成。当用户和服务都使用同一种语言或脚本（更确切地说，如果它们使用同一代码页）时，该过程运行正常。但如果是日语客户端与英语服务器连接，则刚才提到的代码页转译就无法进行，因为 ASP 会尝试将日语字符作为英语字符处理。

解决方案是设置 ASP 转译这些入站和出站字符串时所使用的代码页。设置代码页的机制有两种：

* 在设计时按页面设置。例如，<% @ LANGUAGE=VBScript CODEPAGE=1252 %>
* 运行时在脚本代码中设置：Session.CodePage 属性可设置当前会话的字符串转译所使用的代码页。在 Microsoft Internet 信息服务 (IIS) 5.1 及更高版本中，Response.Code-Page 属性可定义客户端的响应代码页。在显式设置后，响应代码页将覆盖会话代码页，而会话代码页将覆盖 @CODEPAGE 设置。例如，<% @ LANGUAGE=VBScript CODEPAGE=65001 %> <% Response.Write (Session.CodePage) Session.CodePage = 1252 %>

应用这些代码页设置的效果如下：首先，不会影响到 .asp 文件中的任何静态内容 (HTML)，这些返回内容与编写内容完全相同。将根据 .asp 文件中的 CODEPAGE 设置转换脚本代码中的所有静态字符串（实际上是脚本代码本身）。可以这样来看 CODEPAGE，作者（或者更确切地说是自动放置在 .asp 文件中的编写工具）告诉 ASP 编写 .asp 文件时所使用的代码页。

任何动态内容（如 Response.Write(x) 调用，其中 x 为一个变量）都使用 Response.CodePage（默认为 CODEPAGE 设置但可被覆盖）的值进行转换。您必须执行此覆盖，因为编写脚本所使用的代码页可能与您将输出发送给特定客户端所使用的代码页不同。例如，作者编写 ASP 页面时，可能使用生成以 JIS 编码的文本的工具，而终端用户的浏览器可能使用的是 UTF-8。现在，ASP 可利用此代码页控制功能来正确处理代码页的转换。

服务器端脚本中 meta 标记（如前文所述）设置的浏览器行为可通过设置 Response.Charset 属性来实现。设置此属性会指示浏览器如何解释传入流的编码。通常，此值应始终与该会话的代码页值保持一致。

例如，如果某个 ASP 页面不包含 Response.Charset 属性，则 content-type 标头将为：

content-type:text/html

如果包括同一 .asp 文件：

<% Response.Charset= "ISO-LATIN-7" %>

content-type 标头将为：

content-type:text/html; charset=ISO-LATIN-7

**XML 页面**

所有 XML 处理器都必须理解 Unicode 字符编码的两种转换：UTF-8（默认编码）和 UTF-16。Microsoft XML Parser (MSXML) 支持更多编码，但 XML 文档中的所有文本在内部都被视为 Unicode UTF-16 字符编码。

编码声明用于标识文档中使用了哪种编码来表示字符。尽管 XML 分析器可自动判断文档中是使用了 UTF-8 还是 UTF-16 Unicode 编码，但此声明应在支持其他编码的文档中使用。

例如，以下是某个使用了 ISO 8859-1 编码 (Latin 1) 的文档的编码声明：

< xml version="1.0" encoding="ISO-8859-1" >

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688114.aspx#top)

**控制台中的编码**

程序员在对控制台或“文本模式”应用程序编程时可同时使用 Unicode 和 SBCS 或 DBCS 编码。出于承袭方面的考虑，非 Unicode 控制台 I/O 函数使用控制台代码页（默认为 OEM 代码页）。Windows 中的所有其他非 Unicode 函数都使用 Windows 代码页。这意味着其他函数可能无法正确地处理由控制台函数返回的字符串，反之亦然。例如，如果 FindFirstFileA 返回了一个包含某个非 ASCII 字符的字符串，WriteConsoleA 将无法正常地显示该字符串。

要一直跟踪哪些函数需要哪些编码并正确转换文本参数的编码非常困难。函数 SetFileApisToOEM、SetFileApisToANSI 和帮助程序函数 AreFileApisANSI 的引入极大地简化了此项工作。前两个函数可接受或返回文件名称，对 KERNEL32.dll 导出的非 Unicode 函数有效果。顾名思义，SetFileApisToOEM 将这些函数设置为接受或返回与当前系统区域设置对应的 OEM 字符集中的文件名称，而 SetFileApisToANSI 用于为这些名称恢复默认值（Windows ANSI 编码）。使用 AreFileApisANSI 可查询到当前选定的编码。

利用 SetFileApisToOEM 可轻松地解决 WindFirstFileA（或 GetCurrentDirectoryA，或 Win32 API 的任何文件处理函数）的结果无法直接传送给 WriteConsoleA 这一问题：在调用 SetFileApisToOEM 后，WindFirstFileA 将返回以 OEM（而不是以 Windows ANSI 字符集）编码的文本。但此解决方案并非应对所有 Windows ANSI 与 OEM 不兼容情况的万能良方。设想您需要从某个文件处理函数中获取文本，然后将其输出到控制台，接着用其他函数（不受SetFileApisToOEM 的影响）处理该文本。这种绝对理想的情况需要更改编码。否则，您将需要调用 SetFileApisToOEM 以获取用于控制台输出的数据，然后调用SetFileApisToANSI 得到同一文本（使用另一种编码）进行内部处理。另一种 SetFileApisToOEM 无法发挥作用的情况对命令行参数的处理：当您的应用程序的入口点为 main（而不是 wmain）时，参数始终作为 Windows ANSI 字符串数组进行传递。这一切显然使编写非 Unicode 控制台应用程序的程序员的工作变得更复杂了。

更为复杂的是为控制台编写的 8 位代码需要处理两种不同类型的区域设置。在编写代码时，您可以使用 Win32 API 或 C 运行时库函数。Win32 API 的 ANSI 函数假定文本是针对当前控制台代码页（系统区域设置默认定义的）编码的。SetConsoleCP 和 SetConsoleOutputCP 函数可更改这些操作中使用的代码页。用户可在命令提示符下调用 chcp or mode con cp select= 命令，这将会更改当前控制台的代码页。设置固定的控制台代码页的另一种方法是用默认的代码页集创建控制台快捷方式（仅适用于东亚本地化版本的操作系统）。应用程序应能够响应用户的操作。

C 运行时库（CRT 函数）中区分区域设置函数可依照 (\_w)setlocale 调用所定义的设置处理文本。如果代码中未调用 (\_w)setlocale，则 CRT 函数对这些操作使用 ANSI "C" 语言不变区域设置，这样就丢失了特定于语言的功能。

该函数的声明为：

setlocale( int category, const char \*locale)

或

\_wsetlocale( int category, const wchar\_t \*locale)

其中 "category" 用于定义受影响的特定于区域的设置（如果指定了 LC\_ALL，则全部定义）。可变区域设置可以是显式区域设置名称或下面的某一项：

* ".OCP" 指与当前用户区域设置对应的 OEM 代码页
* ".ACP" 或 "" 指与当前用户区域设置对应的 Windows 代码页

".OCP" 和 ".ACP" 参数始终引用用户区域设置（而不是系统区域设置）的设置值。因此不能将它们用于设置 LC\_CTYPE。此 "category" 定义了 Unicode 向 8 位转换的规则和其他文本处理程序，必须遵循控制台的设置（可使用 GetConsoleCP 和 GetConsoleOutputCP 访问）。

对控制台应用程序最佳的长期解决方案是使用 Unicode，因为 Unicode 接口是针对 Win32 API 和 C 运行时库定义的。今后的编程模型仍需要您显式地设置区域设置，但至少可肯定通过 Win32 和 CRT 看到的文本不再需要进行代码转换。

[返回页首返回页首](http://msdn.microsoft.com/zh-cn/goglobal/bb688114.aspx#top)

**引用**

* [Unicode 和字符集](http://msdn.microsoft.com/library/ms776440.aspx)
* [字符集和代码页](http://www.microsoft.com/typography/unicode/cscp.htm)
* [MLang](http://msdn.microsoft.com/library/aa767865.aspx)