第31章　宽字符函数

1995年，标准C中添加了一些宽字符函数，随后被标准C++所采用。宽字符函数对wchar_t类型（16位）的字符进行操作。大多数情况下，这些函数与它们的char对应物并行存在。例如，函数iswspace( )是isspace( )的宽字符版本。一般来讲，宽字符函数使用的名称与它们的char对应物相同，只是增加了“w”。

宽字符函数使用两个头文件：<cwchar>和<cwctype>。也支持C语言的头文件wchar.h和wctype.h。

头文件<cwctype>定义了类型wint_t, wctrans_t和wctype_t。许多宽字符函数都接受一个宽字符作为参数，这种参数是wint_t，它能够容纳一个宽字符。在宽字符函数中使用wint_t与在基于char的函数中使用int相同。wctrans_t和wctype_t是对象的类型，分别用于表示一个字符映射（即，字符翻译）和一个字符分类，宽字符的EOF标记被定义为WEOF。

除了定义win_t外，头文件<cwchar>定义了类型wchar_t, size_t和mbstate_t。wchar_t创建一个宽字符对象，size_t是由sizeof返回的值类型。mbstate_t描述了一个对象，该对象保存一个从多字节到宽字符转换的状态。<cwchar>头文件也定义了宏NULL, WEOF, WCHAR_MAX和WCHAR_MIN。最后两个定义了可以保存在wchar_t类型的对象中的最大和最小值。

尽管标准函数库对宽字符的支持是相当全面的，但并不经常使用这些函数。其中的一个理由是标准C++的I/O系统和类库通过使用模板类提供了对一般的字符和宽字符的支持，还有，对宽字符兼容的程序的兴趣也比预期的少。当然，这种情况可能会改变。

因为大多数宽字符函数只是简单地对应于它们的char等价物，并不被大多数C++程序员经常使用，所以我们只简单地描述一下这些函数。

31.1　宽字符分类函数

头文件<cwctype>提供了支持字符分类的宽字符函数的原型。这些函数把宽字符按其类型进行分类，或者转换字符的大小写。表31.1列出了这些函数和它们的char对应物，char函数已在第26章讲述过。

除了表31.1中显示的函数外，<cwctype>定义了下面的函数，这些函数提供了分类字符的开放（open-ended）方法。

```
wctype_t wctype(const char *attr);
int iswctype(wint_t ch, wctype_t attr_ob);
```

函数wctype( )返回一个值，该值可以被传递给iswctype( )所带的attr_ob参数。attr所指的字符串规定了字符必须具有的属性。使用attr_ob中的值来决定是否ch是一个具有那个属性的字符。如果是，iswctype( )返回非0值，否则，返回0。下面的属性字符串是为所有执行环境定义的：

alnum	alpha	cntrl	digit
graph	lower	print	punct
space	upper	xdigit	

表 31.1　宽字符分类函数

函数	char 对应物
int iswalnum(wint_t *ch*)	isalnum( )
int iswalpha(wint_t *ch*)	isalpha( )
int iswcntrl(wint_t *ch*)	iscntrl( )
int iswdigit(wint_t *ch*)	isdigit( )
int iswgraph(wint_t *ch*)	isgraph( )
int iswlower(wint_t *ch*)	islower( )
int iswprint(wint_t *ch*)	isprint( )
int iswpunct(wint_t *c*)	ispunct( )
int iswspace(wint_t *ch*)	isspace( )
int iswupper(wint_t *ch*)	isupper( )
int iswxdigit(wint_t *ch*)	isxdigit( )
wint_t tolower(wint_t *ch*)	tolower( )
wint_t toupper(wint_t *ch*)	toupper( )

下面的程序演示了 wctype( )和 iswctype( )函数。

```
#include <iostream>
#include <cwctype>
using namespace std;
int main()
{
wctype_t x;
x = wctype("space");
if(iswctype(L' ', x))
   cout << "Is a space.\n";
return 0;
}
```

这个程序显示“Is a space”。

函数 wctrans( )和 towctrans( )也是在<cwctype>中定义的，如下所示：

```
wctrans_t wctrans(const char *mapping);
wint_t towctrans(wint_t ch, wctrans_t mapping_ob);
```

函数 wctrans( )返回一个值，该值可以被传递给 towctrans( )所带的 mapping_ob 参数。其中，mapping 所指的字符串规定了一个字符到另一个字符的映射。然后 iswctrans( )可以使用这个值来映射 ch，返回所映射的值。下面的映射字符串在所有的执行环境下都得到了支持。

```
tolower          toupper
```

下面是一个演示 wctrans( )和 towctrans( )的小例子。

```
#include <iostream>
#include <cwctype>
using namespace std;
```

```
int main()
{
  wctrans_t x;

  x = wctrans("tolower");
  wchar_t ch = towctrans(L'W', x);
  cout << (char) ch;

  return 0;
}
```

这个程序显示了一个小写的“w”。

31.2 宽字符I/O函数

在第25章描述的几个I/O函数都有其宽字符的实现，这些函数展示于表31.2中。宽字符I/O函数使用头文件<cwchar>。注意swprintf( )和vswprintf( )要求一个附加参数，它们的char对应物不需要这个附加参数。

除了表中显示的那些外，添加了下面的宽字符I/O函数：

```
int fwide(FILE *stream, int how);
```

如果how是一个正数，fwide( )使stream成为一个宽字符流。如果how是负数，fwide( )使stream成为一个字符流。如果how是0，该流不受影响。如果这个流已经被定位为宽字符或一般的字符，它将不会改变。如果流使用宽字符，函数返回正数，如果流使用char，函数返回负数。如果流还没有定位好，则返回0。流的定位也是在它第一次使用时决定的。

表31.2 宽字符I/O函数

函数	char对应物
win_t fgetwc(FILE *stream)	fgetc( )
wchar_t *fgetws(wchar_t *str, int num, FILE *stream)	fgets( )
wint_t fputwc(wchar_t ch, FILE *stream)	fputc( )
int fputws(const wchar_t *str, FILE *stream)	fputs( )
int fwprintf(FILE *stream, const wchar_t fmt, ...)	fprintf( )
int fwscanf(FILE *stream, const wchar_t fmt, ...)	fscanf( )
wint_t getwc(FILE *stream)	getc( )
wint_t getwchar( )	getchar( )
wint_t putwc(wchar_t ch, FILE *stream)	putc( )
wint_t putwchar(wchar_t ch)	putchar( )
int swprintf(wchar_t *str, size_t num, const wchar_t *fmt, ...)	sprintf( )，注意增加了参数num，它限制写到str中的字符数
int swscanf(const wchar_t *str, const wchar_t *fmt, ...)	sscanf( )
wint_t ungetwc(wint_t ch, FILE *stream)	ungetc( )
int vfwprintf(FILE *stream, const wchar_t fmt, va_list arg)	vfprintf( )

（续表）

函数	char 对应物
int vswprintf(wchar_t *str, size_t num, const wchar_t *fmt, va_list arg)	vsprintf( )，注意增加了参数num，它限制写到str中的字符数
int vwprintf(const wchar_t *fmt, va_list arg)	vprintf( )
int wprintf(const wchar_t *fmt, ...)	printf( )
int wscanf(const wchar_t *fmt, ...)	scanf( )

31.3　宽字符串函数

在第 26 章中讲述的字符串操作函数也有宽字符版本，示于表 31.3 中。它们使用头文件 <cwchar>。注意 wcstok( )要求一个不被它的 char 对应物使用的附加参数。

表 31.3　宽字符串函数

函数	char 对应物
wchar_t *wcscat(wchar_t *str1, const wchar_t *str2)	strcat( )
wchar_t *wcschr(const wchar_t *str, wchar_t ch)	strchr( )
int wcscmp(const wchar_t *str1, const wchar_t *str2)	strcmp( )
int wcscoll(const wchar_t *str1, const wchar_t *str2)	strcoll( )
size_t wcscspn(const wchar_t *str1, const wchar_t *str2)	strcspn( )
wchar_t *wcscpy(wchar_t *str1, const wchar_t *str2)	strcpy( )
size_t wcslen(const wchar_t *str)	strlen( )
wchar_t *wcsncpy(wchar_t *str1, const wchar_t str2, size_t num)	strncpy( )
wchar_t *wcsncat(wchar_t *str1, const wchar_t str2, size_t num)	strncat( )
int wcsncmp(const wchar_t *str1, const wchar_t *str2, size_t num)	strncmp( )
wchar_t *wcspbrk(const wchar_t *str1, const wchar_t *str2)	strpbrk( )
wchar_t *wcsrchr(const wchar_t *str, wchar_t ch)	strrchr( )
size_t wcsspn(const wchar_t *str1, const wchar_t str2)	strspn( )
wchar_t *wcstok(wchar_t *str1, const wchar_t *str2, wchar_t **endptr)	strtok( )，其中，endptr是一个指针，包含继续标志化过程所需的信息
wchar_t *wcsstr(const wchar_t *str1, const wchar_t *str2)	strstr( )
size_t wcsxfrm(wchar_t *str1, const wchar_t *str2, size_t num)	strxfrm( )

31.4　宽字符串转换函数

表31.4中显示的函数提供了标准数字和时间转换函数的宽字符版本。这些函数使用头文件 <cwchar>。

表31.4 宽字符转换函数

函数	char对应物
size_t wcsftime(wchar_t *str, size_t max, const wchar_t *fmt, const struct tm *ptr)	strftime( )
double wcstod(const wchar_t *start, wchar_t **end);	strtod( )
long wcstol(const wchar_t *start, wchar_t **end, int radix)	strtol( )
unsigned long wcstoul(const wchar_t *start, wchar_t **end, int radix)	strtoul( )

31.5 宽字符数组函数

标准字符数组操作函数，如memcpy( )，也有其宽字符对应物，如表31.5所示。这些函数使用头文件<cwchar>。

表31.5 宽字符数组函数

函数	char对应物
wchar_t *wmemchr(const wchar_t *str, wchar_t ch, size_t num)	memchr( )
int wmemcmp(const wchar_t *str1, const wchar_t *str2, size_t num)	memcmp( )
wchar_t *wmemcpy(wchar_t *str1, const wchar_t *str2, size_t num)	memcpy( )
wchar_t *wmemmove(wchar_t *str1, const wchar_t *str2, size_t num)	memmove( )
wchar_t *wmemset(wchar_t *str, wchar_t ch, size_t num)	memset( )

31.6 多字节/宽字符转换函数

标准C++函数库提供了各种支持在多字节和宽字符之间进行转换的函数。这些函数使用头文件<cwchar>，如表31.6所示。它们中的许多都是正常多字节函数的可重启（restartable）版本。这些可重启版本利用在mbstate_t参数中传递给它的状态信息。如果这个参数为空，函数将提供它自己的mbstate_t对象。

表31.6 宽字符/多字节转换函数

函数	描述
win_t btowc(int ch)	转换ch成为它的宽字符对应物并返回结果。如果ch不是一个一字节、多字节字符，在错误时返回WEOF
size_t mbrlen(const char *str, size_t num, mbstate_t *state)	mblen( )的可重启版本，如state中所述。返回一个表示下一个多字节字符长度的正数。如果下一个字符为空，返回0。如果出现一个错误，那么返回一个负数

（续表）

函数	描述
size_t mbrtowc(wchar_t *out, const char *in, size_t num, mbstate_t *state)	mbtowc( )的可重启版本，就像state所描述的那样。返回一个指示下一个多字节字符长度的正数。如果下一个字符为空，返回0。如果出现一个错误，返回一个负数。如果出现一个错误，把宏EILSEQ赋给errno
int mbsinit(const mbstate_t *state)	如果state表示一个初始转换状态，返回true
size_t mbsrtowcs(wchar_t *out, const char **in, size_t num, mbstate_t state)	mbstowcs( )的可重启版本，正像state所描述的那样。此外，mbsrtowcs( )不同于mbstowcs( )，区别是：in是一个指向源数组的间接指针。如果出现一个错误，把宏EILSEQ赋给errno
size_t wcrtomb(char *out, wchar_t ch, mbstate_t *state)	wctomb( )的可重启版本，正像state所描述的那样。如果出现一个错误，把宏EILSEQ赋给errno
size_t wcsrtombs(char *out, const wchar_t **in, size_t num, mbstate_t *state)	wcstombs( )的可重启版本，正像state所描述的那样。此外，wcsrtombs( )不同于wcstombs( )，区别是：in是一个指向源数组的间接指针。如果出现一个错误，把宏EILSEQ赋给errno
int wctob(wint_t ch)	把ch转换成它的一字节、多字节对应物。失败时，返回EOF