# 第36章 字符串类

本章描述标准C++的字符串类。C++支持两种字符串，第一种是以null结束的字符数组，有时也称为C串，第二种是basic_string类型的类对象。有两种basic_string规范：支持char串的string，和支持wchar_t(宽字符)串的wstring。我们更经常的使用string类型的串对象。

本质上，basic_string类是一个容器。这意味着迭代器和STL算法可以对字符串进行操作。然而，字符串还有其他功能。

basic_string所用的类是char_traits，它定义了组成字符串的字符的几个属性。重要的是要理解：虽然大多数常见的字符串是由char或wchar_t字符组成的，basic_string可以对任何用来表示一个文本字符的对象进行操作。下面描述basic_string和char_traits。

**注意**：关于使用字符串类的概述，请参见第24章。

## 36.1 basic_string类

basic_string的模板规范是：

```
template <class CharType, class Attr = char_traits<CharType>,
          class Allocator = allocator<T> > class basic_string
```

其中，CharType是所用的字符类型，Attr是描述字符性质的类，Allocator指定分配器。basic_string具有下面的构造函数：

```
explicit basic_string(const Allocator &a = Allocator( ));
basic_string(size_type len, CharType ch ,
          const Allocator &a = Allocator( ));
basic_string(const CharType *str, const Allocator &a = Allocator( ));
basic_string(const CharType *str, size_type len,
          const Allocator &a = Allocator( ));
basic_string(const basic_string &str, size_type indx = 0,
          size_type len=npos, const Allocator &a = Allocator( ));
template <class InIter> basic_string(InIter start, InIter end,
          const Allocator &a = Allocator( ));
```

第一种形式构造一个空字符串。第二种形式构造一个具有len个字符、值为ch的字符串。第三种形式构造一个包含与str同样元素的字符串。第四种形式构造一个字符串，该串包含一个以0开始、长度为len个字符的str的子串。第五种形式使用以indx表示的元素开始、长度为len个字符的子串从另一个basic_string中构造一个字符串。第六种形式构造一个字符串，该串包含由start和end指定的范围内的元素。

下面的比较运算符为basic_string而定义：

```
==, <, <=, !=, >, >=
```

同时也定义了+运算符，+运算符生成一个字符串和另一个字符串相连后的结果。此外，定义了I/O运算符<<和>>，这两个运算符可被用来输入和输出字符串。

也可以使用+运算符来连接一个字符串对象和另一个字符串对象或一个字符串对象和一个C样式的字符串。即，支持下面的变种：

字符串+字符串
字符串+C串
C串+字符串

也可以使用+运算符来连接一个字符到一个字符串的末尾。

basic_string类定义了常量npos，它是-1。这个常量表示可能最长的字符串的长度。

在说明中，通用的CharType类型表示被一个字符串存储的字符类型。因为在模板类中占位符类型的名称是任意的，所以basic_string声明了这些类型的typedef的形式，这使得类型名更具体了。由basic_string定义的类型如下表所示：

| | |
|---|---|
| size_type | 大致等于size_t的某个整型类型 |
| reference | 字符串内一个字符的引用 |
| const_reference | 字符串内一个字符的const引用 |
| iterator | 迭代器 |
| const_iterator | const迭代器 |
| reverse_iterator | 反向迭代器 |
| const_reverse_iterator | const反向迭代器 |
| value_type | 存储在字符串中的字符类型 |
| allocator_type | 分配器的类型 |
| pointer | 指向字符串中一个字符的指针 |
| const_pointer | 指向字符串中一个字符的const指针 |
| traits_type | char_traits<CharType>的一个typedef |
| difference_type | 可以存储两个地址间差值的类型 |

basic_string定义的成员函数如表36.1所示。因为大多数程序员将使用char字符串，也为了让描述更易于理解，该表使用string类型，但是这些函数也适用于wstring类型的对象（或任何其他basic_string的对象）。

表36.1 String成员函数

| 成员 | 说明 |
|---|---|
| string &append(const string &*str*); | 追加str到调用字符串的末尾。返回*this |
| string &append(const string &*str*,<br>size_type *indx*,<br>size_type *len*); | 追加str子串到调用字符串的末尾。被追加的子串在indx处开始，长度为len个字符。返回*this |
| string &append(const CharType **str*); | 追加str到调用字符串的末尾。返回*this |
| string &append(const CharType **str*,<br>size_type *num*); | 追加str中的前num个字符到调用字符串的末尾。返回*this |
| string &append(size_type len, CharType *ch*); | 追加由ch指定的len个字符到调用字符串的末尾。返回*this |
| template<class InIter><br>string &append(InIter *start*,<br>InIter *end*); | 追加由start和end指定的序列到调用字符串的末尾。返回*this |

（续表）

| 成员 | 说明 |
| --- | --- |
| string &assign(const string *&str*); | 把 str 赋给调用字符串。返回 *this |
| string &assign(const string *&str*,<br>size_type *indx*,<br>size_type *len*); | 把 str 的子串赋给调用字符串。所赋的子串在 indx 处开始、长度为 len 个字符。返回 *this |
| string &assign(const CharType **str*); | 把 str 赋给调用字符串。返回 *this |
| string &assign(const CharType **str*,<br>size_type *len*); | 把 str 中的前 len 个字符赋给调用字符串。返回 *this |
| string &assign(size_type *len*, CharType *ch*); | 把由 ch 指定的 len 字符赋到调用字符串的末尾。返回 *this |
| template<class InIter><br>string &assign(InIter *start*, InIter *end*); | 把由 start 和 end 指定的序列赋给调用字符串。返回 *this |
| reference at(size_type *indx*);<br>const_reference at(size_type *indx*) const; | 返回一个到由 indx 指定的字符的引用 |
| iterator begin( );<br>const_iterator begin( ) const; | 返回字符串中第一个元素的迭代器 |
| const CharType *c_str( ) const; | 返回一个指向C样式（即以null结束的）调用字符串的指针 |
| size_type capacity( ) const; | 返回字符串的当前容量。这是在需要分配更多的内存前它所包含的字符数 |
| int compare(const string *&str*) const; | 把 str 和调用字符串进行比较。它返回下列值之一：如果 *this < str，则为小于0的值；如果 *this == str，则为0；如果 *this > str，则为大于0的值 |
| int compare(size_type *indx*, size_type *len*,<br>const string *&str*) const; | 把str和调用字符串中的一个子串进行比较。子串在indx处开始且为len个字符长。它返回下列值之一：如果*this < str，则为小于0的值；如果*this == str，则为0；如果*this > str，则为大于0的值 |
| int compare(size_type *indx*, size_type *len*,<br>const string *&str*,<br>size_type *indx2*,<br>size_type *len2*) const; | 把str的一个子串和调用字符串中的一个子串进行比较。调用字符串中的子串在indx处开始且为len个字符长。在str中的子串在indx2处开始且为len2个字符长。它返回下列值之一：如果 *this < str，则为小于0的值；如果 *this == str，则为0；如果 *this > str，则为大于0的值 |
| int compare(const CharType **str*) const; | 把 str 和调用字符串进行比较。它返回下列值之一：如果 *this < str，则为小于0的值；如果 *this == str，则为0；如果 *this > str，则为大于0的值 |
| int compare(size_type *indx*, size_type *len*,<br>const CharType **str*,<br>size_type *len2* = npos) const; | 把 str 的一个子串和调用字符串中的一个子串进行比较。调用字符串中的子串在 indx 处开始且为 len 个字符长。str 中的子串在0处开始且为len2个字符长。它返回下列值之一：如果 *this < str，则为小于0的值；如果 *this == str，则为0；如果 *this > str，则为大于0的值 |
| size_type copy(CharType **str*,<br>size_type len,<br>size_type *indx* = 0) const; | 从 indx 开始，从调用字符串中复制 len 个字符到 str 所指的字符数组中。返回所复制的字符数 |
| const CharType *data( ) const; | 返回指向调用字符串中第一个字符的指针。 |
| bool empty( ) const; | 如果调用字符串为空，返回 true；否则，返回 false |
| iterator end( );<br>const_iterator end( ) const; | 返回指向字符串末尾的迭代器 |

（续表）

| 成员 | 说明 |
| --- | --- |
| iterator erase(iterator *i*); | 删除i所指的字符。返回指向所删除的字符后面那个字符的迭代器 |
| iterator erase(iterator *start*, iterator *end*); | 删除从start到end范围内所有字符。返回指向所删除的最后一个字符后面的迭代器 |
| string &erase(size_type *indx* = 0,<br>size_type *len* = npos); | 从 indx 开始，从调用字符串中删除 len 个字符。返回 *this |
| size_type find(const string &*str*,<br>size_type *indx* = 0) const; | 返回调用字符串内 str 第一次出现处的下标。搜索在下标 indx 处开始。如果没找到匹配，返回 npos |
| size_type find(const CharType **str*,<br>size_type *indx* = 0) const; | 返回调用字符串内 str 第一次出现处的下标。搜索在下标 indx 处开始。如果没有找到匹配，返回 npos |
| size_type find(const CharType **str*,<br>size_type *indx*,<br>size_type *len*) const; | 返回调用字符串内str的前len个字符第一次出现处的下标。搜索在下标 indx 处开始。如果没有找到匹配，返回 npos |
| size_type find(CharType *ch*,<br>size_type *indx* = 0) const; | 返回调用字符串内ch第一次出现处的下标。搜索在下标indx处开始。如果没有找到匹配，返回 npos |
| size_type find_first_of(const string &*str*,<br>size_type *indx* = 0) const; | 返回匹配str中任何字符的调用字符串内第一个字符的下标。搜索在下标 indx 处开始。如果没有找到匹配，返回 npos |
| size_type find_first_of(const CharType **str*,<br>size_type *indx* = 0) const; | 返回匹配str中任何字符的调用字符串内第一个字符的下标。搜索在下标 indx 处开始。如果没有找到匹配，返回 npos |
| size_type find_first_of(const CharType **str*,<br>size_type *indx*,<br>size_type *len*) const; | 返回调用字符串内满足特定条件的第一个字符的下标，其特定条件就是匹配str的前len个字符中的任何字符。搜索从下标 indx 处开始。如果没有找到匹配，返回 npos |
| size_type find_first_of(CharType *ch*,<br>size_type *indx* = 0) const; | 返回调用字符串内ch第一次出现处的下标。搜索从下标indx处开始。如果没有找到匹配，返回 npos |
| size_type find_first_not_of(<br>const string &*str*,<br>size_type *indx* = 0) const; | 返回不匹配str中的任何字符的调用字符串内第一个字符的下标。搜索从下标 indx 处开始。如果没有找到不匹配项，返回 npos |
| size_type find_first_not_of(<br>const CharType **str*,<br>size_type *indx* = 0) const; | 返回不匹配str中任何字符的调用字符串内第一个字符的下标。搜索从下标 indx 处开始。如果没有找到不匹配项，则返回 npos |
| size_type find_first_not_of(<br>const CharType **str*,<br>size_type *indx*,<br>size_type *len*) const; | 返回调用字符串内满足特定条件的第一个字符的下标，该特定条件是不匹配str中前len字符中的任何字符。搜索从下标 indx 处开始。如果没有找到不匹配项，返回 npos |
| size_type find_first_not_of(<br>CharType *ch*,<br>size_type *indx* = 0) const; | 返回调用字符串中不匹配 ch 的第一个字符的下标。搜索从下标 indx 处开始。如果没有找到不匹配项，返回 npos |
| size_type find_last_of(const string &*str*,<br>size_type *indx* = npos) const; | 返回匹配str中任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |
| size_type find_last_of(const CharType **str*,<br>size_type *indx* = npos) const; | 返回匹配str中任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |

（续表）

| 成员 | 说明 |
| --- | --- |
| size_type find_last_of(const CharType *str, size_type indx, size_type len) const; | 返回匹配 str 的前 len 个字符中的任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |
| size_type find_last_of(CharType ch, size_type indx = npos) const; | 返回调用字符串内最后一次出现 ch 时的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |
| size_type find_last_not_of( const string &str, size_type indx = npos) const; | 返回不匹配 str 中任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到不匹配项，返回 npos |
| size_type find_last_not_of( const CharType *str, size_type indx = npos) const; | 返回不匹配 str 中任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到不匹配项，返回 npos |
| size_type find_last_not_of( const CharType *str, size_type indx, size_type len) const; | 返回不匹配 str 的前 len 个字符中任何字符的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |
| size_type find_last_not_of(CharType ch, size_type indx = npos) const; | 返回不匹配 ch 的调用字符串内最后一个字符的下标。搜索在下标 indx 处结束。如果没有找到匹配项，返回 npos |
| allocator_type get_allocator( ) const; | 返回该字符串的分配器 |
| iterator insert(iterator i, const CharType &ch ); | 把 ch 插入到由 i 所指定的字符的前面。返回指向那个字符的迭代器 |
| string &insert(size_type indx, const string &str); | 把 str 插入到调用字符串中由 indx 指定的下标处。返回 *this |
| string &insert(size_type indx1, const string &str, size_type indx2, size_type len); | 把 str 的一个子串插入到调用字符串中由 indx1 指定的下标处。其中的子串从 indx2 开始，长度为 len 个字符。返回 *this |
| string &insert(size_type indx, const CharType *str); | 把 str 插入到调用字符串中由 indx 指定的下标处。返回 *this |
| string &insert(size_type indx, const CharType *str, size_type len); | 把 str 中的前 len 个字符插入到调用字符串中由 indx 指定的下标处。返回 *this |
| string &insert(size_type indx, size_type len, CharType ch); | 把 ch 值的 len 个字符插入到调用字符串中由 indx 指定的下标处。返回 *this |
| void insert(iterator i, size_type len, const CharType &ch) | 把 ch 的 len 个副本插入到由 i 指定的元素的前面 |
| template <class InIter> void insert(iterator i, InIter start, InIter end); | 把由 start 和 end 定义的序列插入到由 i 指定的元素的前面 |
| size_type length( ) const; | 返回字符串中的字符数 |
| size_type max_size( ) const; | 返回字符串能够包含的字符的最大数目 |
| reference operator[ ](size_type indx) const; const_reference operator[ ](size_type indx) const; | 返回一个到由 indx 指定的字符的引用 |

（续表）

| 成员 | 说明 |
| --- | --- |
| string &operator=(const string &*str*);<br>string &operator=(const CharType **str*);<br>string &operator=(CharType *ch*); | 分配指定的字符串或字符到调用字符串。返回 *this |
| string &operator+=(const string &*str*);<br>string &operator+=(const CharType **str*);<br>string &operator+=(CharType *ch*); | 把指定的字符串或字符追加到调用字符串的末尾，返回*this |
| void push_back (const CharType *ch*) | 把 ch 添加到调用字符串的末尾 |
| reverse_iterator rbegin( );<br>const_reverse_iterator rbegin( ) const; | 返回一个指向字符串末尾的反向迭代器 |
| reverse_iterator rend( );<br>const_reverse_iterator rend( ) const; | 返回一个指向字符串开始处的反向迭代器 |
| string &replace(size_type *indx*,<br>size_type *len*,<br>const string &*str*); | 用 str 中的字符串取代调用字符串中的 len 个字符，其中的 len 个字符从 indx 下标处开始。返回 *this |
| string &replace(size_type *indx1*,<br>size_type *len1*,<br>const string &*str*,<br>size_type *indx2*,<br>size_type *len2*); | 用 str 中在 indx1 处开始的字符串中的 len2 个字符，取代调用字符串中的 len1 个字符，其中的 len1 个字符从 indx1 下标处开始。返回 *this |
| string &replace(size_type *indx*,<br>size_type *len*,<br>const CharType **str*); | 用 str 中的字符串取代调用字符串中的 len 个字符，其中的 len 个字符从下标 indx 处开始。返回 *this |
| string &replace(size_type *indx*,<br>size_type *len1*,<br>const CharType **str*,<br>size_type *len2*); | 用 str 中字符串的 len2 个字符取代调用字符串中的 len1 个字符，其中的 len1 个字符从下标 indx 处开始。返回 *this |
| string &replace(size_type *indx*,<br>size_type *len1*,<br>size_type *len2*,<br>CharType *ch*); | 用由 ch 指定的 len2 个字符取代调用字符串中始于 indx 下标处的 len1 个字符。返回 *this |
| string &replace(iterator *start*,<br>iterator *end*,<br>const string &*str*); | 用 str 取代由 start 和 end 指定的范围。返回 *this |
| string &replace(iterator *start*,<br>iterator *end*,<br>const CharType **str*); | 用 str 取代由 start 和 end 指定的范围。返回 *this |
| string &replace(iterator *start*,<br>iterator *end*,<br>const CharType **str*,<br>size_type *len*); | 用 str 中的前 leg 个字符取代由 start 和 end 指定的范围。返回 *this |
| string &replace(iterator *start*,<br>interator end, size_type *len*,<br>CharType *ch*); | 用由 ch 指定的 leg 个字符取代由 start 和 end 指定的范围。返回 *this |

（续表）

| 成员 | 说明 |
| --- | --- |
| template <class InIter><br>string &replace(iterator *start1*,<br>interator *end1*,<br>InIter *start2*,<br>InIter *end2*); | 用由start2和end2指定的字符取代由start1和end1指定的范围的字符。返回 *this |
| void reserve(size_type *num* = 0); | 设置字符串的长度，以便它至少等于 num |
| void resize(size_type *num*)<br>void resize(size_type *num*, CharType *ch*); | 改变字符串的大小到由 num 指定的值。如果必须加长字符串，那么把 ch 指定了值的元素添加到末尾 |
| size_type rfind(const string &*str*,<br>size_type *indx* = npos) const; | 返回调用字符串内 str 最后一次出现处的下标。搜索在下标 indx 处结束。如果没有找到匹配项，则返回 npos |
| size_type rfind(const CharType **str*,<br>size_type *indx* = npos) const; | 返回调用字符串内 str 最后一次出现处的下标。搜索在下标 indx 处结束，如果没有找到匹配项，则返回 npos |
| size_type rfind(const CharType **str*,<br>size_type *indx*,<br>size_type *len*) const; | 返回调用字符串内 str 中的前 len 个字符最后一次出现处的下标。搜索在下标 indx 处结束。如果没有找到匹配项，则返回 npos |
| size_type rfind(CharType *ch*,<br>size_type *indx* = npos) const; | 返回调用字符串内 ch 最后一次出现处的下标。搜索在下标 indx 处结束。如果没有找到匹配项，则返回 npos |
| size_type size( ) const; | 返回当前字符串中的字符数 |
| string substr(size_type *indx* = 0,<br>size_type *len* = npos) const; | 返回调用字符串内从下标indx开始的len个字符的一个子串 |
| void swap(string &*str*) | 把存储在调用字符串中的字符和 ob 中的字符进行交换 |

## 36.2 char_traits 类

类 char_traits 描述了与一个字符相关联的几个属性，它的模板规范如下所示：

```
template<class CharType> struct char_traits
```

其中，CharType 规定了字符的类型。

C++ 库提供了 char_traits 的两个特性：一个用于 char 字符，另一个用于 wchar_t 字符。char_traits 类定义了下面 5 种数据类型：

| | |
| --- | --- |
| char_type | 字符类型。对于 CharType，这是一个 typedef。 |
| int_type | 能够容纳 char_type 类型字符或 EOF 字符的整数类型。 |
| off_type | 能够表示流中偏移量的一个整数类型。 |
| pos_type | 能够表示流中位置的一个整数类型。 |
| state_type | 存储转换状态的对象类型（适用于多字节字符）。 |

char_traits 的成员函数示于表 36.2 中。

表 36.2 char_traits 成员函数

| 成员 | 说明 |
| --- | --- |
| static void assign(char_type &*ch1*,<br>const char_type &*ch2*); | 把 ch2 赋给 ch1 |
| static char_type *assign(char_type **str*,<br>size_t num,<br>char_type *ch2*); | 把 ch2 赋给 str 中的前 num 个字符。返回 str |

（续表）

| 成员 | 说明 |
| --- | --- |
| static int compare(const char_type *str1, const char_type *str2, size_t num); | 把str1中的num个字符和str2中的字符进行比较。如果字符串相同，返回0。否则，如果str1比str2小，返回小于0的数。如果str1比str2大，返回大于0的数 |
| static char_type *copy(char_type *to, const char_type *from, size_t num); | 把num个字符从from复制到to。返回to |
| static int_type eof( ); | 返回文件结束（end-of-file）字符 |
| static bool eq(const char_type &ch1, const char_type &ch2); | 比较ch1和ch2。如果字符相同，返回true。否则，返回false |
| static bool eq_int_type(const int_type &ch1, const int_type &ch2); | 如果ch1等于ch2，返回true。否则，返回false |
| static const char_type *find(const char_type *str, size_t num, const char_type *ch); | 返回指向str中ch第一次出现处的指针。只检查前num个字符。如果失败，返回一个空指针 |
| static size_t length(const char_type *str); | 返回str的长度 |
| static bool lt(const char_type &ch1, const char_type &ch2); | 如果ch1小于ch2，返回ture。否则，返回false |
| static char_type *move(char_type *to, const char_type *from, size_t num); | 把num个字符从from复制到to处。返回to |
| static int_type not_eof(const int_type &ch); | 如果ch不是EOF字符，那么返回ch。否则，返回EOF字符 |
| static char_type to_char_type(const int_type &ch); | 把ch转换成一个char_type并返回结果 |
| static int_type to_int_type(const char_type &ch); | 把ch转换成一个int_type并返回结果 |