C++ Strings(字符串)

Constructors	构造函数，用于字符串初始化
Operators	操作符，用于字符串比较和赋值
append()	在字符串的末尾添加文本
assign()	为字符串赋新值
at()	按给定索引值返回字符
begin()	返回一个迭代器，指向第一个字符
c_str()	将字符串以 C 字符数组的形式返回
capacity()	返回重新分配空间前的字符容量
compare()	比较两个字符串
copy()	将内容复制为一个字符数组
data()	返回内容的字符数组形式
empty()	如果字符串为空，返回真
end()	返回一个迭代器，指向字符串的末尾。（最后一个字符的下一个位置）
erase()	删除字符
find()	在字符串中查找字符
find_first_of()	查找第一个与 value 中的某值相等的字符
find_first_not_of()	查找第一个与 value 中的所有值都不相等的字符
find_last_of()	查找最后一个与 value 中的某值相等的字符
find_last_not_of()	查找最后一个与 value 中的所有值都不相等的字符
get_allocator()	返回配置器
insert()	插入字符
length()	返回字符串的长度
max_size()	返回字符的最大可能个数
rbegin()	返回一个逆向迭代器，指向最后一个字符
rend()	返回一个逆向迭代器，指向第一个元素的前一个位置
replace()	替换字符
reserve()	保留一定容量以容纳字符串（设置 capacity 值）
resize()	重新设置字符串的大小
rfind()	查找最后一个与 value 相等的字符（逆向查找）
size()	返回字符串中字符的数量
substr()	返回某个子字符串
swap()	交换两个字符串的内容

构造函数(Constructors)

```
  string();
  string( size_type length, char ch );
  string( const char *str );
  string( const char *str, size_type length );
  string( string &str, size_type index, size_type length );
  string( input_iterator start, input_iterator end );
```

字符串的构造函数创建一个新字符串，包括:

- 以 length 为长度的 ch 的拷贝（即 length 个 ch）
- 以 str 为初值 (长度任意),
- 以 index 为索引开始的子串，长度为 length, 或者
- 以从 start 到 end 的元素为初值.

操作符(Operators)

```
  ==
  >
  <
  >=
  <=
  !=
  +
  +=
  []
```

你可以用 ==, >, <, >=, <=, and !=比较字符串. 可以用 + 或者 += 操作符连接两个字符串, 并且可以用[]获取特定的字符.

添加文本(append)

```
  basic_string &append( const basic_string &str );
  basic_string &append( const char *str );
  basic_string &append( const basic_string &str, size_type index,
size_type len );
  basic_string &append( const char *str, size_type num );
  basic_string &append( size_type num, char ch );
  basic_string &append( input_iterator start, input_iterator end );
```

append() 函数可以完成以下工作：

- 在字符串的末尾添加 str,
- 在字符串的末尾添加 str 的子串,子串以 index 索引开始，长度为 len
- 在字符串的末尾添加 str 中的 num 个字符,
- 在字符串的末尾添加 num 个字符 ch,
- 在字符串的末尾添加以迭代器 start 和 end 表示的字符序列.

赋值(assign)

```
  basic_string &assign( const basic_string &str );
  basic_string &assign( const char *str );
  basic_string &assign( const char *str, size_type num );
  basic_string &assign( const basic_string &str, size_type index,
size_type len );
  basic_string &assign( size_type num, char ch );
```

函数以下列方式赋值：

- 用 str 为字符串赋值,
- 用 str 的开始 num 个字符为字符串赋值,
- 用 str 的子串为字符串赋值,子串以 index 索引开始，长度为 len
- 用 num 个字符 ch 为字符串赋值.

at

```
  reference at( size_type index );
```

at()函数返回一个引用，指向在 index 位置的字符. 如果 index 不在字符串范围内, at() 将报告"out of range"错误，并抛出 out_of_range 异常。

begin

```
  iterator begin();
```

begin()函数返回一个迭代器，指向字符串的第一个元素.

c_str

```
  const char *c_str();
```

c_str()函数返回一个指向正规 C 字符串的指针, 内容与本字符串相同.

容量(capacity)

```
  size_type capacity();
```

capacity()函数返回在重新申请更多的空间前字符串可以容纳的字符数. 这个数字至少与 size()一样大.

比较(compare)

```
  int compare( const basic_string &str );
  int compare( const char *str );
  int compare( size_type index, size_type length, const basic_string
&str );
  int compare( size_type index, size_type length, const basic_string
&str, size_type index2,
  size_type length2 );
  int compare( size_type index, size_type length, const char *str,
size_type length2 );
```

compare()函数以多种方式比较本字符串和 str,返回:

返回值	情况
小于零	this < str
零	this == str
大于零	this > str

不同的函数:

- 比较自己和 str,
- 比较自己的子串和 str,子串以 index 索引开始，长度为 length
- 比较自己的子串和 str 的子串，其中 index2 和 length2 引用 str，index 和 length 引用自己
- 比较自己的子串和 str 的子串，其中 str 的子串以索引 0 开始，长度为 length2，自己的子串以 index 开始，长度为 length

拷贝(copy)

```
size_type copy( char *str, size_type num, size_type index );
```

copy()函数拷贝自己的 num 个字符到 str 中（从索引 index 开始）。返回值是拷贝的字符数

data

```
const char *data();
```

data()函数返回指向自己的第一个字符的指针.

empty

```
bool empty();
```

如果字符串为空则 empty()返回真(true)，否则返回假(false).

end

```
iterator end();
```

end()函数返回一个迭代器，指向字符串的末尾(最后一个字符的下一个位置).

删除(erase)

```
iterator erase( iterator pos );
iterator erase( iterator start, iterator end );
basic_string &erase( size_type index = 0, size_type num = npos );
```

erase()函数可以:

- 删除 pos 指向的字符，返回指向下一个字符的迭代器,
- 删除从 start 到 end 的所有字符，返回一个迭代器,指向被删除的最后一个字符的下一个位置
- 删除从 index 索引开始的 num 个字符，返回***this**.

参数 *index* 和 *num* 有默认值，这意味着 erase()可以这样调用：只带有 *index* 以删除 index 后的所有字符，或者不带有任何参数以删除所有字符.

查找(find)

```
  size_type find( const basic_string &str, size_type index );
  size_type find( const char *str, size_type index );
  size_type find( const char *str, size_type index, size_type length );
  size_type find( char ch, size_type index );
```

find()函数:

- 返回 str 在字符串中第一次出现的位置（从 index 开始查找）。如果没找到则返回 **string::npos**,
- 返回 str 在字符串中第一次出现的位置（从 index 开始查找，长度为 length）。如果没找到就返回 **string::npos**,
- 返回字符 ch 在字符串中第一次出现的位置（从 index 开始查找）。如果没找到就返回 **string::npos**

find_first_of

```
  size_type find_first_of( const basic_string &str, size_type index =
0 );
  size_type find_first_of( const char *str, size_type index = 0 );
  size_type find_first_of( const char *str, size_type index, size_type
num );
  size_type find_first_of( char ch, size_type index = 0 );
```

find_first_of()函数:

- 查找在字符串中第一个与 str 中的某个字符匹配的字符，返回它的位置。搜索从 index 开始，如果没找到就返回 **string::npos**
- 查找在字符串中第一个与 str 中的某个字符匹配的字符，返回它的位置。搜索从 index 开始，最多搜索 num 个字符。如果没找到就返回 **string::npos，**
- 查找在字符串中第一个与 ch 匹配的字符，返回它的位置。搜索从 index 开始。

find_first_not_of

```
  size_type find_first_not_of( const basic_string &str, size_type index
= 0 );
  size_type find_first_not_of( const char *str, size_type index = 0 );
  size_type find_first_not_of( const char *str, size_type index,
size_type num );
  size_type find_first_not_of( char ch, size_type index = 0 );
```

find_first_not_of()函数:

- 在字符串中查找第一个与 str 中的字符都不匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**
- 在字符串中查找第一个与 str 中的字符都不匹配的字符，返回它的位置。搜索从 index 开始，最多查找 num 个字符。如果没找到就返回 **string::nops**
- 在字符串中查找第一个与 ch 不匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**

find_last_of

```
  size_type find_last_of( const basic_string &str, size_type index =
npos );
  size_type find_last_of( const char *str, size_type index = npos );
  size_type find_last_of( const char *str, size_type index, size_type
num );
  size_type find_last_of( char ch, size_type index = npos );
```

find_last_of()函数:

- 在字符串中查找最后一个与 str 中的某个字符匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**
- 在字符串中查找最后一个与 str 中的某个字符匹配的字符，返回它的位置。搜索从 index 开始，最多搜索 num 个字符。如果没找到就返回 **string::nops**
- 在字符串中查找最后一个与 ch 匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**

find_last_not_of

```
  size_type find_last_not_of( const basic_string &str, size_type index
= npos );
  size_type find_last_not_of( const char *str, size_type index = npos);
  size_type find_last_not_of( const char *str, size_type index,
size_type num );
  size_type find_last_not_of( char ch, size_type index = npos );
```

find_last_not_of()函数:

- 在字符串中查找最后一个与 str 中的字符都不匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**
- 在字符串中查找最后一个与 str 中的字符都不匹配的字符，返回它的位置。搜索从 index 开始，最多查找 num 个字符如果没找到就返回 **string::nops**

- 在字符串中查找最后一个与 ch 不匹配的字符，返回它的位置。搜索从 index 开始。如果没找到就返回 **string::nops**

get_allocator

```
  allocator_type get_allocator();
```

get_allocator()函数返回本字符串的配置器.

插入(insert)

```
  iterator insert( iterator i, const char &ch );
  basic_string &insert( size_type index, const basic_string &str );
  basic_string &insert( size_type index, const char *str );
  basic_string &insert( size_type index1, const basic_string &str,
size_type index2, size_type num );
  basic_string &insert( size_type index, const char *str, size_type
num );
  basic_string &insert( size_type index, size_type num, char ch );
  void insert( iterator i, size_type num, const char &ch );
  void insert( iterator i, iterator start, iterator end );
```

insert()函数的功能非常多:

- 在迭代器 i 表示的位置前面插入一个字符 ch,
- 在字符串的位置 index 插入字符串 str,
- 在字符串的位置 index 插入字符串 str 的子串(从 *index2* 开始，长 num 个字符),
- 在字符串的位置 index 插入字符串 str 的 num 个字符,
- 在字符串的位置 index 插入 num 个字符 ch 的拷贝,
- 在迭代器 i 表示的位置前面插入 num 个字符 ch 的拷贝,
- 在迭代器 i 表示的位置前面插入一段字符，从 start 开始，以 end 结束.

长度(length)

```
  size_type length();
```

length()函数返回字符串的长度. 这个数字应该和 size()返回的数字相同.

max_size

```
  size_type max_size();
```

max_size()函数返回字符串能保存的最大字符数。

rbegin

```
  const reverse_iterator rbegin();
```

rbegin()返回一个逆向迭代器，指向字符串的最后一个字符。

rend

```
  const reverse_iterator rend();
```

rend()函数返回一个逆向迭代器,指向字符串的开头(第一个字符的前一个位置)。

替换(replace)

```
  basic_string &replace( size_type index, size_type num, const
basic_string &str );
  basic_string &replace( size_type index1, size_type num1, const
basic_string &str, size_type index2,
  size_type num2 );
  basic_string &replace( size_type index, size_type num, const char
*str );
  basic_string &replace( size_type index, size_type num1, const char
*str, size_type num2 );
  basic_string &replace( size_type index, size_type num1, size_type
num2, char ch );
  basic_string &replace( iterator start, iterator end, const
basic_string &str );
  basic_string &replace( iterator start, iterator end, const char *str );
  basic_string &replace( iterator start, iterator end, const char *str,
size_type num );
  basic_string &replace( iterator start, iterator end, size_type num,
char ch );
```

replace()函数:

- 用 str 中的 num 个字符替换本字符串中的字符,从 index 开始
- 用 str 中的 num2 个字符（从 index2 开始）替换本字符串中的字符，从 index1 开始，最多 num1 个字符
- 用 str 中的 num 个字符（从 index 开始）替换本字符串中的字符
- 用 str 中的 num2 个字符（从 index2 开始）替换本字符串中的字符，从 index1 开始，num1 个字符
- 用 num2 个 ch 字符替换本字符串中的字符，从 index 开始
- 用 str 中的字符替换本字符串中的字符,迭代器 start 和 end 指示范围
- 用 str 中的 num 个字符替换本字符串中的内容,迭代器 start 和 end 指示范围，
- 用 num 个 ch 字符替换本字符串中的内容，迭代器 start 和 end 指示范围.

保留空间(reserve)

```
void reserve( size_type num );
```

reserve()函数设置本字符串的 capacity 以保留 num 个字符空间。

resize

```
void resize( size_type num );
void resize( size_type num, char ch );
```

resize()函数改变本字符串的大小到 *num*, 新空间的内容不确定。也可以指定用 ch 填充。

rfind

```
size_type rfind( const basic_string &str, size_type index );
size_type rfind( const char *str, size_type index );
size_type rfind( const char *str, size_type index, size_type num );
size_type rfind( char ch, size_type index );
```

rfind()函数:

- 返回最后一个与 str 中的某个字符匹配的字符，从 index 开始查找。如果没找到就返回 **string::npos**
- 返回最后一个与 str 中的某个字符匹配的字符，从 index 开始查找,最多查找 num 个字符。如果没找到就返回 **string::npos**
- 返回最后一个与 ch 匹配的字符，从 index 开始查找。如果没找到就返回 **string::npos**

size

```
size_type size();
```

size()函数返回字符串中现在拥有的字符数。

substr

```
basic_string substr( size_type index, size_type num = npos );
```

substr()返回本字符串的一个子串，从 index 开始，长 num 个字符。如果没有指定，将是默认值 string::npos。这样，substr()函数将简单的返回从 index 开始的剩余的字符串。

交换(swap)

```
void swap( basic_string &str );
```

swap()函数把 str 和本字符串交换。