要想使用标准C++中string类，必须要包含

#include <string>// 注意是<string>，不是<string.h>，带.h的是C语言中的头文件

|  |
| --- |
| using std::string;  using std::wstring;  或  using namespace std;  下面你就可以使用string/wstring了，它们两分别对应着char和wchar\_t。  string和wstring的用法是一样的，以下只用string作介绍： |

## string类的构造函数

|  |
| --- |
| * string(const char \*s); //用c字符串s初始化 * string(int n,char c); //用n个字符c初始化 |
| //此外，string类还支持默认构造函数和复制构造函数，如string s1；string s2="hello"；都是正确的写法。当构造的string太长而无法表达时会抛出length\_error异常 ； |

## string类的字符操作

|  |
| --- |
| * const char &operator[](int n)const; * const char &at(int n)const; * char &operator[](int n); * char &at(int n); |
| operator[]和at()均返回当前字符串中第n个字符的位置，但at函数提供范围检查，当越界时会抛出out\_of\_range异常，下标运算符[]不提供检查访问。 |
| * const char \*data()const;// 返回一个非null终止的c字符数组 * const char \*c\_str()const;// 返回一个以null终止的c字符串 * int copy(char \*s, int n, int pos = 0) const;// 把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目 |

## string的特性描述

|  |  |
| --- | --- |
| int capacity()const; | //返回当前容量（即string中不必增加内存即可存放的元素个数） |
| int max\_size()const; | //返回string对象中可存放的最大字符串的长度 |
| int size()const; | //返回当前字符串的大小 |
| int length()const; | //返回当前字符串的长度 |
| bool empty()const; | //当前字符串是否为空 |
| void resize(int len,char c); | // 把字符串当前大小置为len，并用字符c填充不足的部分 |

## string类的输入输出操作:

string类重载运算符operator>>用于输入，同样重载运算符operator<<用于输出操作。

|  |  |
| --- | --- |
| getline(istream &in,string &s); | 用于从输入流in中读取字符串到s中，以换行符'\n'分开 |

## string的赋值

|  |
| --- |
| * string &operator=(const string &s);// 把字符串s赋给当前字符串 * string &assign(const char \*s);//用c类型字符串s赋值 * string &assign(const char \*s,int n);// 用c字符串s开始的n个字符赋值 * string &assign(const string &s);// 把字符串s赋给当前字符串 * string &assign(int n,char c);// 用n个字符c赋值给当前字符串 * string &assign(const string &s,int start,int n);// 把字符串s中从start开始的n个字符赋给当前字符串 * string &assign(const\_iterator first,const\_itertor last);// 把first和last迭代器之间的部分赋给字符串 |

## string的连接

|  |
| --- |
| * string &operator+=(const string &s); //把字符串s连接到当前字符串的结尾 * string &append(const char \*s); //把c类型字符串s连接到当前字符串结尾 * string &append(const char \*s,int n); //把c类型字符串s的前n个字符连接到当前字符串结尾 * string &append(const string &s); //同operator+=() * string &append(const string &s,int pos,int n);// 把字符串s中从pos开始的n个字符连接到当前字符串的结尾 * string &append(int n,char c); //在当前字符串结尾添加n个字符c * string &append(const\_iterator first,const\_iterator last);// 把迭代器first和last之间的部分连接到当前字符串的结尾 |

## string的比较

|  |
| --- |
| * bool operator==(const string &s1,const string &s2)const;// 比较两个字符串是否相等，运算符">","<",">=","<=","!="均被重载用于字符串的比较 * int compare(const string &s) const;// 比较当前字符串和s的大小 * int compare(int pos, int n,const string &s)const;// 比较当前字符串从pos开始的n个字符组成的字符串与s的大小 * int compare(int pos, int n,const string &s,int pos2,int n2)const;// 比较当前字符串从pos开始的n个字符组成的字符串与s中从pos2开始的n2个字符组成的字符串的大小 * int compare(const char \*s) const; * int compare(int pos, int n,const char \*s) const; * int compare(int pos, int n,const char \*s, int pos2) const;   // compare函数在>时返回1，<时返回-1，==时返回0 |

## string的子串

|  |
| --- |
| * string substr(int pos = 0,int n = npos) const;// 返回pos开始的n个字符组成的字符串 |

## string的交换

|  |
| --- |
| * void swap(string &s2); //交换当前字符串与s2的值 |

## string类的查找函数

|  |
| --- |
| * int find(char c, int pos = 0) const;// 从pos开始查找字符c在当前字符串的位置 * int find(const char \*s, int pos = 0) const;// 从pos开始查找字符串s在当前串中的位置 * int find(const char \*s, int pos, int n) const;// 从pos开始查找字符串s中前n个字符在当前串中的位置 * int find(const string &s, int pos = 0) const;// 从pos开始查找字符串s在当前串中的位置 |
| 查找成功时返回所在位置，失败返回string::npos的值 |
|  |
| * int rfind(char c, int pos = npos) const;// 从pos开始从后向前查找字符c在当前串中的位置 * int rfind(const char \*s, int pos = npos) const; * int rfind(const char \*s, int pos, int n = npos) const; * int rfind(const string &s,int pos = npos) const; |
| * int find\_first\_of(char c, int pos = 0) const;//从pos开始查找字符c第一次出现的位置 * int find\_first\_of(const char \*s, int pos = 0) const; * int find\_first\_of(const char \*s, int pos, int n) const; * int find\_first\_of(const string &s,int pos = 0) const; |
| * int find\_first\_not\_of(char c, int pos = 0) const; * int find\_first\_not\_of(const char \*s, int pos = 0) const; * int find\_first\_not\_of(const char \*s, int pos,int n) const; * int find\_first\_not\_of(const string &s,int pos = 0) const;   //从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos |
| * int find\_last\_of(char c, int pos = npos) const; * int find\_last\_of(const char \*s, int pos = npos) const; * int find\_last\_of(const char \*s, int pos, int n = npos) const; * int find\_last\_of(const string &s,int pos = npos) const; |
| * int find\_last\_not\_of(char c, int pos = npos) const; * int find\_last\_not\_of(const char \*s, int pos = npos) const; * int find\_last\_not\_of(const char \*s, int pos, int n) const; * int find\_last\_not\_of(const string &s,int pos = npos) const;   //find\_last\_of和find\_last\_not\_of与find\_first\_of和find\_first\_not\_of相似，只不过是从后向前查找 |

## string类的替换函数

|  |
| --- |
| * string &replace(int p0, int n0,const char \*s);//删除从p0开始的n0个字符，然后在p0处插入串s * string &replace(int p0, int n0,const char \*s, int n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符 * string &replace(int p0, int n0,const string &s);//删除从p0开始的n0个字符，然后在p0处插入串s * string &replace(int p0, int n0,const string &s, int pos, int n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符 * string &replace(int p0, int n0,int n, char c);//删除p0开始的n0个字符，然后在p0处插入n个字符c * string &replace(iterator first0, iterator last0,const char \*s);//把[first0，last0）之间的部分替换为字符串s * string &replace(iterator first0, iterator last0,const char \*s, int n);//把[first0，last0）之间的部分替换为s的前n个字符 * string &replace(iterator first0, iterator last0,const string &s);//把[first0，last0）之间的部分替换为串s * string &replace(iterator first0, iterator last0,int n, char c);//把[first0，last0）之间的部分替换为n个字符c * string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串 |

## string类的插入函数

|  |
| --- |
| * string &insert(int p0, const char \*s); * string &insert(int p0, const char \*s, int n); * string &insert(int p0,const string &s); * string &insert(int p0,const string &s, int pos, int n);   //前4个函数在p0位置插入字符串s中pos开始的前n个字符   * string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c * iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置 * void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符 * void insert(iterator it, int n, char c);//在it处插入n个字符c |

## string类的删除函数

|  |
| --- |
| * iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置 * iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置 * string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串 |

## string类的迭代器处理：

string类提供了向前和向后遍历的迭代器iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。

用string::iterator或string::const\_iterator声明迭代器变量，const\_iterator不允许改变迭代的内容。

常用迭代器函数有：

|  |
| --- |
| * const\_iterator begin()const; * iterator begin(); //返回string的起始位置 |
| * const\_iterator end()const; * iterator end(); //返回string的最后一个字符后面的位置 |
| * const\_iterator rbegin()const; * iterator rbegin(); //返回string的最后一个字符的位置 |
| * const\_iterator rend()const; * iterator rend(); //返回string第一个字符位置的前面 |
| rbegin和rend用于从后向前的迭代访问，通过设置迭代器string::reverse\_iterator,string::const\_reverse\_iterator实现 |

## 字符串流处理：

通过定义ostringstream和istringstream变量实现，#include <sstream>头文件中。例如：

|  |
| --- |
| string input("hello,this is a test");  istringstream is(input);  string s1,s2,s3,s4;  is>>s1>>s2>>s3>>s4;//s1="hello,this",s2="is",s3="a",s4="test"  ostringstream os;  os<<s1<<s2<<s3<<s4;  cout<<os.str(); |

用法大全（转） (2012-10-12 16:26:33)转载▼

标签： 杂谈

之所以抛弃char\*的字符串而选用C++标准程序库中的string类，是因为他和前者比较起来，不必担心内存是否足够、字符串长度等等，而且作为一个类出现，他集成的操作函数足以完成我们大多数情况下(甚至是100%)的需要。我们可以用 = 进行赋值操作，== 进行比较，+ 做串联（是不是很简单?）。我们尽可以把它看成是C++的基本数据类型。

# 第1节

## 声明一个C++字符串

声明一个字符串变量很简单：

string Str;

这样我们就声明了一个字符串变量，但既然是一个类，就有构造函数和析构函数。上面的声明没有传入参数，所以就直接使用了string的默认的构造函数，这个函数所作的就是把Str初始化为一个空字符串。

String**类的构造函数和析构函数如下：**

|  |
| --- |
| a) string s; //生成一个空字符串s  b) string s(str) //拷贝构造函数 生成str的复制品  c) string s(str,stridx) //将字符串str内"始于位置stridx"的部分当作字符串的初值  d) string s(str,stridx,strlen) //将字符串str内"始于stridx且长度顶多strlen"的部分作为字符串的初值  e) string s(cstr) //将C字符串作为s的初值  f) string s(chars,chars\_len) //将C字符串前chars\_len个字符作为字符串s的初值。  g) string s(num,c) //生成一个字符串，包含num个c字符  h) string s(beg,end) //以区间beg;end(不包含end)内的字符作为字符串s的初值  i) s.~string() //销毁所有字符，释放内存 |

## 字符串操作函数

|  |
| --- |
| a) =,assign() //赋以新值  b) swap() //交换两个字符串的内容  c) +=,append(),push\_back() //在尾部添加字符  d) insert() //插入字符  e) erase() //删除字符  f) clear() //删除全部字符  g) replace() //替换字符  h) + //串联字符串  i) ==,!=,<,<=,>,>=,compare() //比较字符串  j) size(),length() //返回字符数量  k) max\_size() //返回字符的可能最大个数  l) empty() //判断字符串是否为空  m) capacity() //返回重新分配之前的字符容量  n) reserve() //保留一定量内存以容纳一定数量的字符  o) [ ], at() //存取单一字符  p) >>,getline() //从stream读取某值  q) << //将谋值写入stream  r) copy() //将某值赋值为一个C\_string  s) c\_str() //将内容以C\_string返回  t) data() //将内容以字符数组形式返回  u) substr() //返回某个子字符串  v)查找函数  w)begin() end() //提供类似STL的迭代器支持  x) rbegin() rend() //逆向迭代器  y) get\_allocator() //返回配置器 |

下面详细介绍：

### 2.1 C++字符串和C字符串的转换

C ++提供的由C++字符串得到对应的C\_string的方法是使用data()、c\_str()和copy()，其中，

* **data()**以字符数组的形式返回字符串内容，但并不添加’\0’。
* **c\_str()**返回一个以‘\0’结尾的字符数组。
* **copy()**则把字符串的内容复制或写入既有的c\_string或字符数组内。

C++字符串并不以’\0’结尾。我的建议是在程序中能使用C++字符串就使用，除非万不得已不选用c\_string。

### 2.2 大小和容量函数

一个C++字符串存在三种大小：

a)现有的字符数，函数是size()和length()，他们等效。Empty()用来检查字符串是否为空。

b)max\_size() 这个大小是指当前C++字符串最多能包含的字符数，很可能和机器本身的限制或者字符串所在位置连续内存的大小有关系。我们一般情况下不用关心他，应该大小足够我们用的。但是不够用的话，会抛出length\_error异常。

c)capacity()重新分配内存之前string所能包含的最大字符数。

这里另一个需要指出的是reserve()函数，这个函数为string重新分配内存。重新分配的大小由其参数决定， 默认参数为0，这时候会对string进行非强制性缩减。

还有必要再重复一下C++字符串和C字符串转换的问题，许多人会遇到这样的问题，自己做的程序要调用别人的函数、类什么的（比如数据库连接函数Connect(char\*,char\*)），但别人的函数参数用的是char\*形式的，而我们知道，c\_str()、data()返回的字符数组由该字符串拥有，所以是一种const char\*,要想作为上面提及的函数的参数，还必须拷贝到一个char\*,而我们的原则是能不使用C字符串就不使用。那么，这时候我们的处理方式是：如果此函数对参数(也就是char\*)的内容不修改的话，我们可以这样Connect((char\*)UserID.c\_str(), (char\*)PassWD.c\_str()),但是这时候是存在危险的，因为这样转换后的字符串其实是可以修改的（有兴趣地可以自己试一试），所以我强调除非函数调用的时候不对参数进行修改，否则必须拷贝到一个char\*上去。当然，更稳妥的办法是无论什么情况都拷贝到一个char\*上去。

### 2．3元素存取

我们可以使用下标操作符[]和函数at()对元素包含的字符进行访问。但是**应该注意的是操作符[]并不检查索引是否有效（有效索引0~str.length()），如果索引失效，会引起未定义的行为**。而at()会检查，如果使用 at()的时候索引无效，会抛出out\_of\_range异常。

有一个例外不得不说，const string a;的操作符[]对索引值是a.length()仍然有效，其返回值是’\0’。其他的各种情况，a.length()索引都是无效的。举例如下：

|  |
| --- |
| const string Cstr("const string");  string Str("string");  Str[3]; //ok  Str.at(3); //ok  Str[100]; //未定义的行为  Str.at(100); //throw out\_of\_range  Str[Str.length()] //未定义的行为  Cstr[Cstr.length()] //返回‘\0’  Str.at(Str.length());//throw out\_of\_range  Cstr.at(Cstr.length()) ////throw out\_of\_range |

我不赞成类似于下面的引用或指针赋值：

char& r=s[2];

char\* p= &s[3];

因为一旦发生重新分配，r,p立即失效。避免的方法就是不使用。

### 2.4比较函数

C ++字符串支持常见的比较操作符（>,>=,<,<=,==,!=），甚至支持string与C-string的比较(如 str<"hello")。在使用>,>=,<,<=这些操作符的时候是根据"当前字符特性"将字符按字典顺序进行逐一得 比较。字典排序靠前的字符小，比较的顺序是从前向后比较，遇到不相等的字符就按这个位置上的两个字符的比较结果确定两个字符串的大小。同时，string ("aaaa") 另一个功能强大的比较函数是成员函数compare()。他支持多参数处理，支持用索引值和长度定位子串来进行比较。他返回一个整数来表示比较结果，返回值意义如下：0-相等 〉0-大于 <0-小于。

举例如下：

|  |
| --- |
| string s("abcd");  s.compare("abcd"); //返回0  s.compare("dcba"); //返回一个小于0的值  s.compare("ab"); //返回大于0的值  s.compare(s); //相等  s.compare(0,2,s,2,2); //用"ab"和"cd"进行比较 小于零  s.compare(1,2,"bcx",2); //用"bc"和"bc"比较。 |

### 2．5 更改内容

首先讲赋值，第一个赋值方法当然是使用操作符=，新值可以是string(如：s=ns) 、c\_string(如：s="gaint")甚至单一字符（如：s=’j’）。还可以使用成员函数assign()，这个成员函数可以使你更灵活的对字符串赋值。还是举例说明吧：

|  |
| --- |
| s.assign(str); //不说  s.assign(str,1,3);//如果str是"iamangel" 就是把"ama"赋给字符串  s.assign(str,2,string::npos);//把字符串str从索引值2开始到结尾赋给s  s.assign("gaint"); //不说  s.assign("nico",5);//把’n’ ‘I’ ‘c’ ‘o’ ‘\0’赋给字符串  s.assign(5,’x’);//把五个x赋给字符串 |

* 把字符串清空的方法有三个：s="";s.clear();s.erase();(我越来越觉得举例比说话让别人容易懂！)。

string提供了很多函数用于插入（insert）、删除（erase）、替换（replace）、增加字符。

先说增加字符（这里说的增加是在尾巴上），函数有 +=、append()、push\_back()。

举例如下：

|  |
| --- |
| s+=str;//加个字符串  s+="my name is jiayp";//加个C字符串  s+=’a’;//加个字符  s.append(str);  s.append(str,1,3);//不解释了 同前面的函数参数assign的解释  s.append(str,2,string::npos)//不解释了  s.append("my name is jiayp");  s.append("nico",5);  s.append(5,’x’);  s.push\_back(‘a’);//这个函数只能增加单个字符 对STL熟悉的理解起来很简单 |

也许你需要在string中间的某个位置插入字符串，这时候你可以用insert()函数，这个函数需要你指定一个安插位置的索引，被插入的字符串将放在这个索引的后面。

|  |
| --- |
| s.insert(0,"my name");  s.insert(1,str); |

这种形式的insert()函数不支持传入单个字符，这时的单个字符必须写成字符串形式(让人恶心)。既然你觉得恶心，那就不得不继续读下面一段话：为了插入单个字符，insert()函数提供了两个对插入单个字符操作的重载函数：insert(size\_type index,size\_type num,chart c)和insert(iterator pos,size\_type num,chart c)。其中size\_type是无符号整数，iterator是char\*,所以，你这么调用insert函数是不行的：insert(0,1, ’j’);这时候第一个参数将转换成哪一个呢？所以你必须这么写：insert((string::size\_type)0,1,’j’)！

第二种形式指出了使用迭代器安插字符的形式，在后面会提及。顺便提一下，string有很多操作是使用STL的迭代器的，他也尽量做得和STL靠近。

删除函数erase()的形式也有好几种（真烦！），替换函数replace()也有好几个。举例吧：

|  |
| --- |
| string s="il8n";  s.replace(1,2,"nternationalizatio");//从索引1开始的2个替换成后面的C\_string  s.erase(13);//从索引13开始往后全删除  s.erase(7,5);//从索引7开始往后删5个 |

### 2．6提取子串和字符串连接

题取子串的函数是：substr(),形式如下：

s.substr();//返回s的全部内容

s.substr(11);//从索引11往后的子串

s.substr(5,6);//从索引5开始6个字符

把两个字符串结合起来的函数是+。（谁不明白请致电120）

### 2．7输入输出操作

1．>> 从输入流读取一个string。

2．<< 把一个string写入输出流。

另一个函数就是getline(),他从输入流读取一行内容，直到遇到分行符或到了文件尾。

### 2．8搜索与查找

查找函数很多，功能也很强大，包括了：

find()

rfind()

find\_first\_of()

find\_last\_of()

find\_first\_not\_of()

find\_last\_not\_of()

这些函数返回符合搜索条件的字符区间内的第一个字符的索引，没找到目标就返回npos。所有的函数的参数说明如下：

第一个参数是被搜寻的对象。第二个参数（可有可无）指出string内的搜寻起点索引，第三个参数（可有可无）指出搜寻的字符个数。比较简单，不多说不理解的可以向我提出，我再仔细的解答。当然，更加强大的STL搜寻在后面会有提及。

最

后再说说npos的含义，string::npos的类型是string::size\_type,所以，一旦需要把一个索引与npos相比，这个索引值必须是string::size)type类型的，更多的情况下，我们可以直接把函数和npos进行比较（如：if(s.find("jia")==

string::npos)）。