C++ string详解，C++字符串详解

C++ 大大增强了对字符串的支持，除了可以使用C风格的字符串，还可以使用内置的 string 类。string 类处理起字符串来会方便很多，完全可以代替C语言中的字符数组或字符串指针。

string 是 C++ 中常用的一个类，它非常重要，我们有必要在此单独讲解一下。

使用 string 类需要包含头文件 `<string>`，下面的例子介绍了几种定义 string 变量（对象）的方法：

```
01.  #include <iostream>
02.  #include <string>
03.  using namespace std;
04.
05.  int main(){
06.      string s1;
07.      string s2 = "c plus plus";
08.      string s3 = s2;
09.      string s4 (5, 's');
10.      return 0;
11.  }
```

变量 s1 只是定义但没有初始化，编译器会将默认值赋给 s1，默认值是 `""`，也即空字符串。

变量 s2 在定义的同时被初始化为 `"c plus plus"`。与C风格的字符串不同，string 的结尾没有结束标志 `'\0'`。

变量 s3 在定义的时候直接用 s2 进行初始化，因此 s3 的内容也是 `"c plus plus"`。

变量 s4 被初始化为由 5 个 `'s'` 字符组成的字符串，也就是 `"sssss"`。

从上面的代码可以看出，string 变量可以直接通过赋值操作符 `=` 进行赋值。string 变量也可以用C风格的字符串进行赋值，例如，s2 是用一个字符串常量进行初始化的，而 s3 则是通过 s2 变量进行初始化的。

与C风格的字符串不同，当我们需要知道字符串长度时，可以调用 string 类提供的 length() 函数。如下所示：

```
01.  string s = "http://c.biancheng.net";
02.  int len = s.length();
03.  cout<<len<<endl;
```

输出结果为 22 。由于 string 的末尾没有 '\0' 字符，所以 length() 返回的是字符串的真实长度，而不是长度 +1。

转换为C风格的字符串

虽然 C++ 提供了 string 类来替代C语言中的字符串，但是在实际编程中，有时候必须要使用C风格的字符串（例如打开文件时的路径），为此，string 类为我们提供了一个转换函数 c_str()，该函数能够将 string 字符串转换为C风格的字符串，并返回该字符串的 const 指针（const char*）。请看下面的代码：

```
string path = "D:\\demo.txt";
FILE *fp = fopen(path.c_str(), "rt");
```

为了使用C语言中的 fopen() 函数打开文件，必须将 string 字符串转换为C风格的字符串。

string 字符串的输入输出

string 类重载了输入输出运算符，可以像对待普通变量那样对待 string 变量，也就是用 >> 进行输入，用 << 进行输出。请看下面的代码：

```
#include <iostream>
#include <string>

using namespace std;

int main(){
    string s;
    cin>>s;  //输入字符串
    cout<<s<<endl;  //输出字符串
    return 0;
}
```

运行结果：
http://c.biancheng.net http://vip.biancheng.net↙
http://c.biancheng.net

虽然我们输入了两个由空格隔开的网址，但是只输出了一个，这是因为输入运算符 >> 默认会忽略空格，遇到空格就认为输入结束，所以最后输入的 http://vip.biancheng.net 没有被存储到变量 s。

访问字符串中的字符

string 字符串也可以像C风格的字符串一样按照下标来访问其中的每一个字符。string 字符串的起始下标仍是从 0 开始。请看下面的代码：

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

int main(){
    string s = "1234567890";
    for(int i=0,len=s.length(); i<len; i++){
        cout<<s[i]<<" ";
    }
    cout<<endl;
    s[5] = '5';
    cout<<s<<endl;
    return 0;
}
```

运行结果：
1 2 3 4 5 6 7 8 9 0
1234557890

本例定义了一个 string 变量 s，并赋值 "1234567890"，之后用 for 循环遍历输出每一个字符。借助下标，除了能够访问每个字符，也可以修改每个字符，`s[5] = '5';` 就将第6个字符修改为 '5'，所以 s 最后为 "1234557890"。

字符串的拼接

有了 string 类，我们可以使用 `+` 或 `+=` 运算符来直接拼接字符串，非常方便，再也不需要使用C语言中的 strcat()、strcpy()、malloc() 等函数来拼接字符串了，再也不用担心空间不够会溢出了。

用 `+` 来拼接字符串时，运算符的两边可以都是 string 字符串，也可以是一个 string 字符串和一个C风格的字符串，还可以是一个 string 字符串和一个字符数组，或者是一个 string 字符串和一个单独的字符。请看下面的例子：

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

int main(){
    string s1 = "first ";
    string s2 = "second ";
```

```
    char *s3 = "third ";
    char s4[] = "fourth ";
    char ch = '@';

    string s5 = s1 + s2;
    string s6 = s1 + s3;
    string s7 = s1 + s4;
    string s8 = s1 + ch;

    cout<<s5<<endl<<s6<<endl<<s7<<endl<<s8<<endl;

    return 0;
}
```

运行结果：
first second
first third
first fourth
first @

string 字符串的增删改查

C++ 提供的 string 类包含了若干实用的成员函数，大大方便了字符串的增加、删除、更改、查询等操作。

一. 插入字符串

insert() 函数可以在 string 字符串中指定的位置插入另一个字符串，它的一种原型为：

```
string& insert (size_t pos, const string& str);
```

pos 表示要插入的位置，也就是下标；str 表示要插入的字符串，它可以是 string 字符串，也可以是C风格的字符串。

请看下面的代码：

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

int main(){
    string s1, s2, s3;
    s1 = s2 = "1234567890";
```

```
08.     s3 = "aaa";
09.     s1.insert(5, s3);
10.     cout<< s1 <<endl;
11.     s2.insert(5, "bbb");
12.     cout<< s2 <<endl;
13.     return 0;
14. }
```

运行结果：
12345aaa67890
12345bbb67890

insert() 函数的第一个参数有越界的可能，如果越界，则会产生运行时异常，我们将会在《C++异常（Exception）》一章中详细讲解如何捕获这个异常。

更多 insert() 函数的原型和用法请参考：
http://www.cplusplus.com/reference/string/string/insert/

二. 删除字符串

erase() 函数可以删除 string 中的一个子字符串。它的一种原型为：

```
string& erase (size_t pos = 0, size_t len = npos);
```

pos 表示要删除的子字符串的起始下标，len 表示要删除子字符串的长度。如果不指明 len 的话，那么直接删除从 pos 到字符串结束处的所有字符（此时 len = str.length - pos）。

请看下面的代码：

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1, s2, s3;
07.     s1 = s2 = s3 = "1234567890";
08.     s2.erase(5);
09.     s3.erase(5, 3);
10.     cout<< s1 <<endl;
11.     cout<< s2 <<endl;
12.     cout<< s3 <<endl;
13.     return 0;
14. }
```

运行结果：
1234567890
12345
1234590

有读者担心，在 pos 参数没有越界的情况下， len 参数也可能会导致要删除的子字符串越界。但实际上这种情况不会发生，erase() 函数会从以下两个值中取出最小的一个作为待删除子字符串的长度：

- len 的值；
- 字符串长度减去 pos 的值。

说得简单一些，待删除字符串最多只能删除到字符串结尾。

三. 提取子字符串

substr() 函数用于从 string 字符串中提取子字符串，它的原型为：

```
string substr (size_t pos = 0, size_t len = npos) const;
```

pos 为要提取的子字符串的起始下标，len 为要提取的子字符串的长度。

请看下面的代码：

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

int main(){
    string s1 = "first second third";
    string s2;
    s2 = s1.substr(6, 6);
    cout<< s1 <<endl;
    cout<< s2 <<endl;
    return 0;
}
```

运行结果：
first second third
second

系统对 substr() 参数的处理和 erase() 类似：

- 如果 pos 越界，会抛出异常；

- 如果 len 越界，会提取从 pos 到字符串结尾处的所有字符。

四. 字符串查找

string 类提供了几个与字符串查找有关的函数，如下所示。

1) find() 函数

find() 函数用于在 string 字符串中查找子字符串出现的位置，它其中的两种原型为：

```
size_t find (const string& str, size_t pos = 0) const;
size_t find (const char* s, size_t pos = 0) const;
```

第一个参数为待查找的子字符串，它可以是 string 字符串，也可以是C风格的字符串。第二个参数为开始查找的位置（下标）；如果不指明，则从第0个字符开始查找。

请看下面的代码：

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

int main(){
    string s1 = "first second third";
    string s2 = "second";
    int index = s1.find(s2,5);
    if(index < s1.length())
        cout<<"Found at index : "<< index <<endl;
    else
        cout<<"Not found"<<endl;
    return 0;
}
```

运行结果：
Found at index : 6

find() 函数最终返回的是子字符串第一次出现在字符串中的起始下标。本例最终是在下标6处找到了 s2 字符串。如果没有查找到子字符串，那么会返回一个无穷大值 4294967295。

2) rfind() 函数

rfind() 和 find() 很类似，同样是在字符串中查找子字符串，不同的是 find() 函数从第二个参数开始往后查找，而 rfind() 函数则最多查找到第二个参数处，如果到了第二个参数所指定的下标还没有找到子字符串，则返回一个无穷大值4294967295。

请看下面的例子：

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

int main(){
    string s1 = "first second third";
    string s2 = "second";
    int index = s1.rfind(s2,6);
    if(index < s1.length())
        cout<<"Found at index : "<< index <<endl;
    else
        cout<<"Not found"<<endl;
    return 0;
}
```

运行结果：

Found at index : 6

3) find_first_of() 函数

find_first_of() 函数用于查找子字符串和字符串共同具有的字符在字符串中首次出现的位置。请看下面的代码：

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

int main(){
    string s1 = "first second second third";
    string s2 = "asecond";
    int index = s1.find_first_of(s2);
    if(index < s1.length())
        cout<<"Found at index : "<< index <<endl;
    else
        cout<<"Not found"<<endl;
    return 0;
}
```

运行结果：

Found at index : 3

本例中 s1 和 s2 共同具有的字符是‘ s’，该字符在 s1 中首次出现的下标是3，故查找结果返回3。