针对字符串处理，C++中并没有专门的内置类型。C语言是使用字符数组以及相应的指针来表示字符串。前面章节已经介绍了C++包容的C语言字符串处理方式。下面将讲解C++标准库封装字符串处理的字符串类类型。通过库提供的字符串类型公开方法接口，开发者可以很方便地定义操作字符串，使得操作字符串变得方便简易。

* 13.1.1 string字符串对象构造

C++库中提供的字符串标准类是通过模板定义来实现的。其中，模板类basic\_string主要用于表示两类字符串操作：一种是目前应用程序中常用的string类型；另一种是支持宽类型的字符操作。目前本章所讲述的是string类型。名称空间std中string其实只是basic\_string模板类的别名。如果采用typedef定义string作为basic\_string字符类型实例类的别名，其定义语句如下。

typedef basic\_string<char> string; //定义模板类char型实例化类类型别名string

其中，模板类basic\_string通过char字符型实例化其类型，即此时处理的字符串为char型。然后通过typedef关键字定义string为该类类型别名。这样，开发者就可以直接把string作为其类类型操作。下面所讲述的库提供的字符串类就是直接采用string名称来说明。

C++标准库字符串类的对象实例定义时会涉及该类构造函数来实现构造。对于标准的字符串类，标准库提供了多个构造函数。开发者需要根据定义对象实例的方式，来决定调用哪个构造函数来构造字符串对象实例。下面讲解定义实现几种字符串对象构造实例，帮助读者了解标准库字符串类所提供的字符串对象构造方式。

string mystring1(“string”); //采用字符串指针构造并初始化string(constchar \*str)

string mystring2(6,’a’); //采用指定数目的字符构造并初始化string(int n,char c);

string mystring3; //直接调用字符串无参数构造函数，构造一个空字符串

string mystring4 = “string”; //直接调用字符串类构造函数隐式执行转换初始化

在定义中，字符串类string提供了多种构造函数，实现了不同方式定义其对象实例并初始化。

q 第一种构造方式，通过传入字符串指针构造并初始化字符串对象。

q 第二个则是通过传入字符个数以及对应的字符来初始化既定个数字符的字符串对象。

q 第三个则仅仅是通过构造函数来分配字符串变量空间，暂时先不作任何初始化工作。

q 第四种方式是通过调用字符串类构造函数隐式地去执行了转换操作，给定义的字符串对象赋值。

通常软件开发中采用第三种构造方式。程序中仅仅先定义字符串对应的对象，而赋值操作在实际应用中进行。字符串类string使用时，要注意包含字符串标准库提供的头文件#include<string>。这样可以在应用程序中具体的使用字符串标准库的接口。

13.1.2 string字符串赋值

C++标准库字符串类提供的字符串赋值操作有多种操作接口可选择。研究字符串类标准定义可以看出，字符串类赋值操作主要有两种方式：一种利用赋值符号重载实现，即直接使用等于号的重载而实现。另一种是使用提供的assign()方法接口实现字符串赋值操作。下面将会通过几个赋值操作的实例，来讲解标准字符串类提供的赋值操作。

1.准备实例

打开UE工具，创建新的空文件并且另存为chapter1301.cpp。该代码文件随后会同makefile文件一起通过FTP工具传输至Linux服务器端，客户端通过scrt工具访问操作。程序代码文件编辑如下所示。

/\*\*

\* 实例chapter1301

\* 源文件chapter1301.cpp

\* 字符串赋值操作

\*/

#include <iostream>

#include <string>

using namespace std;

/\*主函数入口\*/

int main()

{

stringmystring1; //字符串类对象mystring1定义

mystring1= "string"; //字符串类对象mystring1赋值，存放字符串“string”

cout<<"mystring1value:"<<mystring1<<endl; //打印输出字符串对象mystring1的值

stringmystring2(mystring1); //字符串对象mystring2定义，传入mystring1初始化

cout<<"mystring2value:"<<mystring2<<endl; //打印输出字符串对象mystring2的值

stringmystring3; //字符串对象mystring3定义

mystring3.assign(mystring1); //通过assign方法为mystring3对象赋值

cout<<"mystring3value:"<<mystring3<<endl; //打印输出字符串对象mystring3的值

mystring3.assign(mystring2,1,3); //通过assign方法将mystring2的值从1～3位截取赋值

cout<<"mystring3value:"<<mystring3<<endl; //打印输出字符串对象mystring3的值

stringmystring4; //字符串类对象mystring4定义

mystring4.assign(mystring1,1,string::npos); //通过assign方法将mystring1值从1到末尾截取赋值

cout<<"mystring4value:"<<mystring4<<endl; //打印输出字符串对象mystring4的值

mystring4.assign("hello"); //通过assign方法将字符串”hello”赋值给mystring4

cout<<"mystring4value:"<<mystring4<<endl; //打印输出字符串mystring4的值

stringmystring5; //字符串类对象mystring5定义

mystring5.assign(4,'c'); //通过assign方法将4个字符’c’组合赋值给mystring5

cout<<"mystring5value:"<<mystring5<<endl; //打印输出字符串mystring5值

return0;

}

本实例主要通过标准字符串类演示字符串对象相互赋值的操作情况。程序主要在主函数中演示字符串赋值操作，程序具体详解见程序注释与后面讲解。

本实例中主要使用标准字符串类定义字符串对象，演示了标准字符串类提供的几种赋值方式的操作。程序中使用到字符串类时，需要include包含提供的头文件<string>。对于标准字符串提供的等号赋值运算操作，初学者应该比较的熟悉。C++字符串类提供的assign()方法的操作更加灵活的实现了字符串赋值的种种可能出现的情况。该方法相对于等号赋值符的重载，并通过重载方法实现的赋值操作显得更加灵活。

上述实例代码编写非常的简单，仅仅一个chapter1301.cpp代码源文件。程序主流程处理在main函数中完成。下面依次讲解五个字符串的定义和初始化实现方式。

（1）首先定义string类的一个空字符串对象mystring1，随后的代码采用等号运算符为其赋值。此时mystring1字符串中存放的即为字符串“string”，下一行代码则打印输出验证mystring1中的值。

（2）定义字符串对象mystring2时，采用另外一种构造并初始化的方式。将mystring1字符串中的内容作为新定义字符串mystring2的初始化值。此时mystring2字符串对象中存放的即为mystring1的内容，为字符串“string”。随后，打印输出并验证。

（3）字符串对象mystring3的定义同mystring1。通过定义构造一个空字符串对象mystring3，随后采用标准字符串类提供的assign()方法为其赋值。此时调用的assign()方法，根据传入实参mystring1为字符串对象mystring3赋值。所以，mystring3中存放着字符串mystring1的值，打印输出验证mystring3中存放着字符串“string”。之后，mystring3又调用assign()方法。通过提供的重载assign()赋值方式，把mystring2字符串第一个字符到第三个字符的字符串赋给mystring3字符串。此时mystring3中存放着字符串“tri”。

注意：字符串中字符串位置从0开始计算。

（4）字符串变量mystring4则通过调用assign()重载方法，将mystring1字符串从位置1到字符串结尾处赋值给字符串mystring4，所以mystring4字符串中存放的字符串为“tring”。随后，打印输出验证。

（5）字符串mystring5则调用assign()方法，并采用指定字符方式来赋值。该方法根据传入的实参使用4个'c'字符初始化该字符串。最后，打印输出mystring5中存放着字符串“cccc”。

13.1.3 string字符串连接

标准字符串类提供了字符串连接的操作接口。该类接口主要实现了运算符“+”、“+=”的重载。另外，用户还可以使用成员函数append()提供的字符串连接功能。标准字符串类提供的连接操作接口原型如下。

string operator+(const string &str); //将字符串str连接到当前字符串的尾部

string &operator+=(const string &str); //将字符串str连接到当前字符串尾部并赋值给当前字符串变量

string &append(const char \*str); //将char\*表示的字符串连接至当前字符串尾部

string &append(const char \*str,int n); //将char\*表示的字符串的前n个字符连接到当前字符串尾部

string &append(const string &str); //将字符串str连接至当前字符串尾部，并赋值给当前字符串

string &append(const string &str,intpos,int n); //将str从指定的pos位置开始后的n个字符连接到当前字符串尾部

string &append(int n,char c); //将指定个数为n的字符c添加到字符串尾部

string &append(const\_iteratorfirst,const\_iterator last); //将迭代器first到last之间的子串连接到当前字符串尾部

根据标准字符串类提供的连接接口，下面通过一个完整实例演示字符串连接功能接口的应用情况。

1.准备实例

打开UE工具，创建新的空文件并且另存为chapter1302.cpp。该代码文件随后会同makefile文件一起通过FTP工具传输至Linux服务器端，客户端通过scrt工具访问操作。程序代码文件编辑如下所示。

/\*\*

\* 实例chapter1302

\* 源文件chapter1302.cpp

\* 字符串连接操作

\*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

stringmystring1; //定义字符串对象mystring1

mystring1= "string"; //初始化字符串对象mystring1，初始值为字符串”string”

cout<<"mystring1value:"<<mystring1<<endl; //打印输出字符串对象mystring1的值

stringmystring2; //定义字符串对象mystring2

mystring2= mystring1 + "123"; //字符串对象mystring1与字符串”123”通过’+’符号连接赋值

cout<<"mystring2value:"<<mystring2<<endl; //打印输出字符串对象mystring2的值

stringmystring3; //定义字符串对象mystring3

mystring3+= mystring2; //通过’+=’符号将mystring2连接赋值给mystring3

cout<<"mystring3value:"<<mystring3<<endl; //打印输出字符串对象mystring3的值

mystring3.append("456"); //字符串对象mystring3调用append方法追加字符串”456”

cout<<"mystring3value:"<<mystring3<<endl; //打印输出字符串对象mystring3的值

stringmystring4; //定义字符串对象mystring4

mystring4.append("hello",4); //字符串对象mystring4调用append方法，截取”hello”前4 //位，进行追加

cout<<"mystring4value:"<<mystring4<<endl; //打印输出字符串对象mystring4的值

mystring4.append(mystring1); //mystring4调用append方法，追加mystring1的值

cout<<"mystring4value:"<<mystring4<<endl; //打印输出字符串mystring4的值

stringmystring5; //定义字符串对象mystring5

mystring5.append(mystring1,2,3); //字符串对象mystring5调用append方法，截取mystring1 //的2位置起3个字符追加给mystring5

cout<<"mystring5value:"<<mystring5<<endl; //打印输出字符串对象mystring5的值

mystring5.append(2,'c'); //字符串对象mystring5调用append方法，追加2个’c’字符 //给字符串对象mystring5

cout<<"mystring5value:"<<mystring5<<endl; //打印输出字符串对象mystring5的值

return0;

}

本实例程序主要通过标准字符串类提供的字符串连接接口，演示字符串连接的基本操作。程序主要在主函数中实现，具体程序剖析见程序注释与后面的讲解。

下面对代码依次进行介绍。

（1）首先定义字符串对象mystring1，并通过等号赋值运算符为该字符串赋值。此时打印输出字符串mystring1变量中的内容为“string”。

（2）定义空字符串对象实例mystring2，并通过连接符号重载操作，将字符串“123”连接到字符串mystring1。同时，采用赋值运算符将连接结果存放于字符串mystring2对象中，并打印mystring2字符串。此时，字符串对象mystring2中内容为“string123”。

（3）定义字符串变量mystring3，并使用运算符“+=”重载实现的接口，将字符串mystring2的内容连接至空字符串mystring3，并存放到mystring3中。随后，打印输出mystring3的字符串对象内容为“string123”。

（4）调用方法成员append(const char \*str)，将字符串“456”追加到字符串mystring3尾部，并打印其结果。此时mystring3字符串中存放的字符串内容为“string123456”。

（5）定义字符串变量mystring4，并调用标准字符串提供的连接方法成员方法append()，将字符串“hello”中前4个字符连接到字符串mystring4中。此时mystring4字符串存放的结果为“hell”。字符串对象mystring4再调用append()，将字符串mystring1连接并存放至字符串mystring4中。此时打印输出mystring4字符串内容为“hellstring”。

（5）定义字符串对象mystring5，并调用append()，将字符串mystring1中从位置2开始往后3个字符连接至字符串mystring5中。打印输出此时的mystring5，其内容为“rin”。随后调用append()方法，将2个‘c’字符连接至mystring5字符串尾部。此时，打印输出字符串mystring5内容为“rincc”。

13.1.4 string字符串字符访问

标准字符串string类提供的下标主要用于字符串中的字符访问。标准string类中主要重载实现“[]”运算符操作来为下标方式访问字符串中具体的字符。另外，标准字符串string类也提供了at()方法成员，以函数定义方式访问字符串中单个字符操作。string类中字符串处理的方法原型如下。

char &operator[](int n); //字符串string类提供重载下标操作符方法

const char &operator[](int n)const; //字符串string类提供重载下标操作符方法，不同的是为常量访问

char &at(int n); //字符串string类提供访问字符串字符方法

const char &at(int n)const; //字符串string类提供访问字符串字符方法，不同的是为常量访问

上述string类提供的字符访问操作都同时拥有两个版本。一种是允许访问并修改字符串中字符的操作。另一种则是只能以常量的方式访问字符串中字符，不允许调用者修改。至于具体程序使用哪种方式，需要根据实际应用场景作出选择。

在以上的两种方式中，实现的功能都是访问并返回当前字符串中指定位置的字符。其中，at()方法成员特别增加了为了防止访问非法范围的异常判断，从而提供了所谓的访问检查。所以，at()方法更加安全。

读者应该能够联想到前面类中运算符重载章节的字符串封装实例。C++允许自定义类型的优势之一就在于允许根据需要自行封装实现一些公共性操作类类型，从而供开发者在应用程序中使用。封装实现的字符串类中就提到下标操作符的封装实现。下面同样通过一个完整实例，演示访问字符串中字符的操作。

1.准备实例

打开UE工具，创建新的空文件并且另存为chapter1303.cpp。该代码文件随后会同makefile文件一起通过FTP工具传输至Linux服务器端，客户端通过scrt工具访问操作。程序代码文件编辑如下所示。

/\*\*

\* 实例chapter1303

\* 源文件chapter1303.cpp

\* 字符串字符访问

\*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

stringmystring1; //定义字符串对象mystring1

mystring1= "string"; //初始化字符串对象mystring1，初始化为字符串"string"

mystring1[0]= 'h'; //字符串mystring1第一个位置赋值为字符’h’

mystring1[1]= 'e'; //字符串mystring1第二个位置赋值为字符’e’

mystring1[2]= 'l'; //字符串mystring1第三个位置赋值为字符’l’

mystring1[3]= 'l'; //字符串mystring1第四个位置赋值为字符’l’

mystring1[4]= '0'; //字符串mystring1第五个位置赋值为字符’0’

mystring1.at(5)= '1'; //字符串mystring1第六个位置赋值为字符’1’

cout<<"mystring1value:"<<mystring1<<endl; //打印输出字符串mystring1的值

conststring mystring2 = "hello"; //定义字符串常量对象mystring2，初始化值为”hello”

charch1,ch2; //定义字符变量ch1、ch2

ch1= mystring2[3]; //将字符串对象mystring2的第四个字符赋值给ch1

ch2= mystring2.at(4); //将字符串对象mystring2的第五个字符赋值给ch2

cout<<"ch1value:"<<ch1<<" "<<"ch2value:"<<ch2<<endl; //打印字符变量ch1、ch2

return0;

}

本实例主要通过字符串类提供的标准访问字符接口，演示字符串中访问具体字符的操作功能。程序主要在主函数中完成，具体剖析见程序注释与后面的讲解。

本实例主要演示了字符串类string提供的访问字符串字符的操作运用情况。实例程序中首先定义字符串对象mystring1，随后给mystring1赋值，此时该字符串内容为“string”。然后使用字符串string的下标运算符访问并修改字符串mystring1中从0到4位置的字符。接着调用at方法访问并修改字符串第5个位置的字符。最后打印输出字符串mystring1中存放的字符串值为“hell01”。

为体现常量下标以及常量at()方法的运用，定义常量字符串mystring2并赋初值“hello”，即字符串常量mystring2不能再作任何修改。随后定义两个字符变量ch1与ch2，通过字符串常量mystring2下标操作访问字符串第3个字符，并将其返回值为字符ch1赋值。另外，通过调用at()方法访问字符串第4个字符返回并给ch2赋值。最后，打印输出ch1与ch2字符变量中的获取到的常量字符串中指定位置的字符。

13.1.5 string字符串比较

比较运算接口主要有两种方式来实现：第一种是通过重载实现比较运算符；第二种则是通过提供对外compare()方法实现字符串比较操作。除了运算符重载实现的比较操作，下面给出string类中实现比较操作的方法成员原型定义。

int compare(const string &str);

int compare(const char \*str);

int compare(size\_type index,size\_type length,conststring &str);

int compare(size\_type index,size\_type length,constchar \*str);

int compare(size\_type index1,size\_type length1,conststring &str,size\_type index2,size\_type length2);

int compare(size\_type index1,size\_type length1,constchar \*str2,size\_type index2,size\_type length2);

上述定义中，compare()方法成员有多个重载版本实现。它们主要用于不同情况下字符串比较操作。需要注意的是compare()方法的返回值。该方法实现的比较功能通过其返回值的判断比较的操作数情况。当返回1时，表明操作数一大于操作数二；当返回-1时，表明操作数一小于操作数二；当返回值为0，则表明两个操作数相等。下面将会照例通过一个完整实例来演示字符串的比较操作。

1.准备实例

打开UE工具，创建新的空文件并且另存为chapter1304.cpp。该代码文件随后会同makefile文件一起通过FTP工具传输至Linux服务器端，客户端通过scrt工具访问操作。程序代码文件编辑如下所示。

/\*\*

\* 实例chapter1304

\* 源文件chapter1304.cpp

\* 字符串比较操作

\*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

stringmystring1,mystring2; //定义字符串类对象mystring1与mystring2

mystring1= "hello"; //给字符串对象mystring1初始化赋值

mystring2= "string"; //给字符串对象mystring2初始化赋值

if(mystring1!= mystring2) //通过’!=’运算符比较两个字符串对象是否相等

{

if(mystring1> mystring2) //通过’>’运算符比较两个字符串对象，比较哪个字符串大

{

cout<<"mystring1is large than mystring2!"<<endl; //打印输出比较结果信息

}

elseif(mystring1 < mystring2) //通过’<’运算符比较两个字符串对象，比较哪个字符串小

{

cout<<"mystring1is less than mystring2!"<<endl; //打印输出比较结果信息

}

}

elseif(mystring1 == mystring2) //通过’==’运算符比较两个字符串对象是否恒相等

{

cout<<"mystring1is equal mystring2!"<<endl; //打印输出比较结果信息

}

stringmystring3; //定义字符串类对象mystring3

mystring3= "string"; //给字符串对象mystring3初始化赋值

cout<<"compare(conststring &str)const:" //在打印输出中调用compare方法比较字符串对象mystring2

<<mystring3.compare(mystring2)<<endl; //与mystring3，并输出比较结果

cout<<"compare(intpos,int n,const string &str)const:" //在打印输出中调用compare方法比较字符串对象

<<mystring3.compare(2,3,mystring2)<<endl; // mystring2第2个位置后的3个字符组成的串与

//mystring3，并输出比较结果

cout<<"compare(intpos1,int n1,const string &str,int pos2,int n2)const:" //在打印输出中调用compare方法

<<mystring3.compare(1,3,mystring2,2,4)<<endl; //比较mystring3位置1往后3个 //字符组成的串与mystring2从位 //置2往后4个字符组成的串，并 //输出比较结果

cout<<"compare(constchar \*str):" //在打印输出中调用compare方法比较常量字符串

<<mystring3.compare("name")<<endl; //”name”与mystring3，并输出比较结果

cout<<"compare(intpos,int n,const char \*str)const:" //在打印输出中调用compare方法比较常量字符串

<<mystring3.compare(2,3,"name")<<endl; //”name”从位置2往后3个字符组成的串与 //mystring3，并输出比较结果

cout<<"compare(intpos1,int n1,const char \*str,int pos2,int n2)const:" //在打印输出中调用compare方法

<<mystring3.compare(2,3,"name",1,2)<<endl; //比较常量字符串”name”从位置1

//往后2个字符组成的串与mystring3从

//位置2往后3个字符组成的串，并输出

//比较结果信息

return0;

}

本实例主要通过标准字符串类提供的接口，演示字符串比较运算操作。程序主要在main函数内部实现，具体程序剖析见程序注释与后面讲解。

本实例程序中主要演示了标准字符串string类提供的两种方式的比较操作接口。本实例在主程序中完成，下面将会将程序分为两个部分来分别说明。

第一个部分说明字符串对象通过if这样的控制结构来进行比较操作。标准字符串string类通过重载比较运算符的调用对比两个字符串mystring1与mystring2大小。使用if/elseif控制结构首先调用“!=”运算符重载操作判断两个字符串内容。如果判断为真则分开判断两个字符串内容是否符合“>”运算符的运算，如果判断为真则打印对应的信息。

如果判断为假，则判断两个字符串是否符合“<”运算。如果符合，则打印对应信息；否则，则跳出，执行对应的“==”比较判断。判断为真，则执行对应代码体；否则，程序继续往下执行。由于mystring1字符串对应的内容为“hello”，mystring2对应的字符串内容为“string”，根据两个字符串对应的单个字符判断。第一个字符串'h'小于's'，则比较结果为mystring1小于mystring2。打印对应的信息为“mystring1 less than mystring2!”。

第二个部分演示了标准字符串string类中的compare()方法成员完成字符串比较的应用。首先定义字符串对象mystring3并初始化值为“string”。代码“mystring3.compare(mystring2)”表明调用字符串string类提供的方法compare()，将mystring2作为参数传入进行比较并打印输出其比较结果。由于此时两个字符串对象中内容一致，则打印输出比较值为0。

代码“mystring3.compare(2,3,mystring2)”说明调用了compare()方法比较字符串mystring3从第2位置开始往后3个字符组成的字符串与mystring2，其比较结果打印输出为-1，即新组成的字符串小于mystring2字符串值。

代码“mystring3.compare(1,3,mystring2,2,4)”说明调用了compare()方法比较了mystring3字符串从位置1开始往后3个字符组成的字符串与mystring2从位置2开始往后4个字符组成的字符串，最后输出比较结果为1。

此后的代码则是采用C语言中字符串指针方式来演示上述标准字符串string类相同操作实现。这里就不用过多解释了，提供的方法接口与字符串对象处理同。

13.1.6 string字符串查找与替换

标准字符串string类提供了非常丰富的查找方法。这里主要以find方法实现为主，大致可以分为6种不同的方法函数定义。同时在6种不同的函数基础上，每种方法根据不同需要又重载实现了4次，即总共为字符串查找操作定义实现了24个查找方法接口，从而使得字符串查找操作非常丰富。下面就标准字符串提供的24种字符串查找操作方法接口基本使用情况作出如下说明。

//当前字符串中查找对应的子串或字符

size\_type find(conststring &str,size\_type index);

size\_type find(constchar \*str,size\_type index);

size\_type find(constchar \*str,size\_type index,size\_type length);

size\_type find(charch,size\_type index);

//当前字符串中查找对应子串或字符，从字符串的尾部开始查找

size\_type rfind(conststring &str,size\_type index);

size\_type rfind(constchar \*str,size\_type index);

size\_type rfind(constchar \*str,size\_type index,size\_type length);

size\_type rfind(charch,size\_type index);

//当前字符串中查找对应的子串或字符中任意一个字符首次出现的位置

size\_type find\_first\_of(conststring &str,size\_type index = 0);

size\_type find\_first\_of(constchar \*str,size\_type index = 0);

size\_type find\_first\_of(constchar \*str,size\_type index,size\_type num);

size\_type find\_first\_of(charch,size\_type index = 0);

//当前字符串中查找对应的子串或字符中任意一个字符最后一次出现的位置

size\_type find\_last\_of(conststring &str,size\_type index = npos);

size\_type find\_last\_of(constchar \*str,size\_type index = npos);

size\_type find\_last\_of(constchar \*str,size\_type index,size\_type num);

size\_type find\_last\_of(charch,size\_type index = npos);

//当前字符串中查找不包含对应的子串或字符中的第一个字符的位置，从字符串前向后依次查找

size\_type find\_first\_not\_of(conststring &str,size\_type index = 0);

size\_type find\_first\_not\_of(constchar \*str,size\_type index = 0);

size\_type find\_first\_not\_of(constchar \*str,size\_type index,size\_type num);

size\_type find\_first\_not\_of(charch,size\_type index = 0);

//当前字符串中查找不包含对应的子串或字符中的第一个字符的位置，从字符串尾部向前依次查找

size\_type find\_last\_not\_of(conststring&str,size\_type index = npos);

size\_type find\_last\_not\_of(const char\*str,size\_type index = npos);

size\_type find\_last\_not\_of(constchar \*str,size\_type index,size\_type num);

size\_type find\_last\_not\_of(charch,size\_type index = npos);

以上方法原型为标准字符串string类针对字符串查找提供的功能接口。初学者在学习时需要掌握这样一种方式，即根据提供的库中方法接口的基本功能以及参数的说明，能够正确的理解并在所开发的应用程序中正确调用。

从上述提供的24个字符串查找方法接口来看，根据不同的处理需求共分类4类。其中每种方式重载实现是根据同样的需求而定义的接口。

q 第一类find方法，主要功能是在当前调用的字符串中查找对应的子串或字符。如果找到，则返回找到子串或者字符的起始位置的下标；否则，返回值为string::npos。它表示该查找方法中无法在当前字符串中查找到对应的子串或者字符。

q 第二类rfind方法则表示查找子串或者字符从当前的字符串尾部开始逆向向前查找。同样，如果找到，则返回对应的下标位置；否则，返回string::npos值，表示无法找到对应的子串或字符。

q 第三类find\_first\_of方法表示在当前字符串中查找子串或者字符中任意一个字符首次找到的位置。找到，则返回；否则，返回值为string::npos，表示未找到。

q 第四类find\_last\_of方法则在当前字符串中查找对应的子串或字符中任意一个字符最后一次出现的位置。返回值同上，都是成功，则返回对应位置下标；否则返回string::npos值，表示未找到。

q 第五类方法find\_first\_not\_of则表示在当前字符串中查找不包含传入的子串或字符中的第一个字符，该查找方法从头开始，找到则返回对应的位置。

q 第六类find\_last\_not\_of方法则提供功能与find\_first\_not\_of方法相同，只是该方法是从字符串的尾部开始查找。

13.1.7 string字符串中插入字符

string类提供在字符串中插入字符的操作。它主要由方法insert()来实现。该方法重载实现了多次，基本包含了大多数情况下的字符串插入功能。字符串插入操作的基本原型如下。

string& insert(size\_type index,conststring& str); //当前字符串位置index之前插入字符串对象str

string& insert(size\_type index,const char\*str); //当前字符串位置index之前插入字符串str

string& insert(size\_type index,size\_typenum,char ch); //当前字符串位置index之前插入num个字符ch

string& insert(size\_type index1,conststring& str,size\_type index2,size\_type num); //当前字符串位置index1前 //插入字符串str下标位置为index2往后的num个字符

iterator insert(iterator i,const char& ch); //当前字符串迭代器i之前插入ch返回新元素的迭代器

void insert(iterator i,size\_type num,constchar& ch); //当前字符串迭代器i之前插入num个字符ch，无返回

void insert(iterator i,iterator start,iterator end); //当前字符串迭代器i之前插入迭代器begin到end范围 //内的元素，无返回

上述是7种insert方法接口重载的实现。字符串插入方法接口基本说明已经在注释中标注。下面将会通过一个完整字符串插入操作实例，帮助读者理解字符串插入操作的实现。

1.准备实例

打开UE工具，创建新的空文件并且另存为chapter1305.cpp。该代码文件随后会同makefile文件一起通过FTP工具传输至Linux服务器端，客户端通过scrt工具访问操作。程序代码文件编辑如下所示。

/\*\*

\* 实例chapter1305

\* 源文件chapter1305.cpp

\* 字符串插入字符操作

\*/

#include <iostream>

using namespace std;

int main()

{

stringmystring1,mystring2; //定义字符串对象mystring1与mystring2

mystring1= "hello"; //字符串对象mystring1初始化赋值为”hello”

mystring2= "C++"; //字符串对象mystring2初始化赋值为”C++”

mystring1.insert(5,mystring2); //字符串对象mystring1调用insert方法，将当前

//mystring1字符串第5个字符位置之前，插入

//mystring2字符串

cout<<"afterinsert mystring2,mystring1's value:" //打印输出此时mystring1的值

<<mystring1<<endl;

mystring1.insert(5,"Linux"); //字符串对象mystring1调用insert方法，将当前

//mystring1字符串第5个字符位置之前，插入

//字符串”Linux”

cout<<"afterinsert \"Linux\",mystring1's value:" //打印输出此时mystring1的值

<<mystring1<<endl;

mystring1.insert(13,2,'!'); //字符串对象mystring1调用insert方法，将当前

//mystring1字符串第13个字符位置之前，插入

//2个’!’字符

cout<<"afterinsert \"!\",mystring1's value:" //打印输出此时mystring1的值

<<mystring1<<endl;

mystring2.insert(0,mystring1,0,5); //字符串对象mystring2调用insert方法，在mystring2

//第0个位置上插入mystring1从0到位置5之间的字符

cout<<"afterinsert mystring1,mystring2's value:" //打印输出此时mystring2的值

<<mystring2<<endl;

return0;

}

本实例主要通过标准字符串类提供的insert接口，演示字符串插入操作。程序主要在main函数内部实现，具体程序剖析见程序注释与后面讲解。

本实例中主要演示了字符串插入操作接口的前4种应用。后面3种关于迭代器应用于字符串插入操作接口的使用情况会放在STL模板章节介绍。实例依然在一个源代码文件中演示。主程序中首先定义两个字符串对象mystring1与mystring2，随后分别给两个字符串对象赋值。

两个字符串对象定义并赋值初始化之后，字符串对象mystring1调用插入方法insert()，根据传入的实参(5,mystring2)决定了调用插入方法中的insert(size\_type index,const string& str)。该方法表明在当前调用的字符串mystring1第5个位置之前插入字符串对象mystring2内容。此时打印输出mystring1的内容为“helloC++”。

随后字符串对象mystring1调用insert()方法，直接通过传入的字符串常量实参插入到当前字符串对象内容指定的位置之前。insert(5,"Linux")方法调用表明在当前mystring1字符串第5个位置前插入字符串“Linux”。此时，mystring1打印输出结果为“helloLinuxC++”。

mystring1字符串对象调用insert(size\_type index,size\_type num,char ch)方法，根据传入的实参从当前字符串mystring1中第13个位置前插入2个'!'字符。最终输出mystring1字符串对象内容为“helloLinuxC++!!”。mystring2字符串对象调用插入方法insert()，从当前字符串mystring2第0个位置开始插入字符串mystring1从第0位置后的5个字符，其结果输出mystring2对象内容为“helloC++”。

---------------------

作者：布衣1983

来源：CSDN

原文：https://blog.csdn.net/wangfengwf/article/details/11580763

版权声明：本文为博主原创文章，转载请附上博文链接！