c++ 字符串流 sstream（常用于格式转换）【转载】- wyu123

wyu123 粉丝 - 111 关注 - 0

使用stringstream对象简化类型转换

C++标准库中的<sstream>提供了比ANSI C的<stdio.h>更高级的一些功能，即单纯性、类型安全和可扩展性。在本文中，我将展示怎样使用这些库来实现安全和自动的类型转换。

为什么要学习

如果你已习惯了<stdio.h>风格的转换，也许你首先会问：为什么要花额外的精力来学习基于<sstream>的类型转换呢？也许对下面一个简单的例子的回顾能够说服你。假设你想用sprintf()函数将一个变量从int类型转换到字符串类型。为了正确地完成这个任务，你必须确保证目标缓冲区有足够大空间以容纳转换完的字符串。此外，还必须使用正确的格式化符。如果使用了不正确的格式化符，会导致非预知的后果。下面是一个例子：

int n=10000;

chars[10];

sprintf(s,"%d",n);// s中的内容为“10000”

到目前为止看起来还不错。但是，对上面代码的一个微小的改变就会使程序崩溃：

int n=10000;

char s[10];

sprintf(s,"%f",n);// 看！错误的格式化符

在这种情况下，程序员错误地使用了%f格式化符来替代了%d。因此，*s*在调用完sprintf()后包含了一个不确定的字符串。要是能自动推导出正确的类型，那不是更好吗？

进入stringstream由于*n*和s的类型在编译期就确定了，所以编译器拥有足够的信息来判断需要哪些转换。<sstream>库中声明的标准类就利用了这一点，自动选择所必需的转换。而且，转换结果保存在stringstream对象的内部缓冲中。你不必担心缓冲区溢出，因为这些对象会根据需要自动分配存储空间。

你的编译器支持<sstream>吗？<sstream>库是最近才被列入C++标准的。（不要把<sstream>与标准发布前被删掉的<strstream>弄混了。）因此，老一点的编译器，如GCC2.95，并不支持它。如果你恰好正在使用这样的编译器而又想使用<sstream>的话，就要先对它进行升级更新。

<sstream>库定义了三种类：istringstream、ostringstream和stringstream，分别用来进行流的输入、输出和输入输出操作。另外，每个类都有一个对应的宽字符集版本。简单起见，我主要以stringstream为中心，因为每个转换都要涉及到输入和输出操作。

注意，<sstream>使用string对象来代替字符数组。这样可以避免缓冲区溢出的危险。而且，传入参数和目标对象的类型被自动推导出来，即使使用了不正确的格式化符也没有危险。

string到int的转换string result="10000";
int n=0;
stream<<result;
stream>>n;//n等于10000

重复利用stringstream对象

如果你打算在多次转换中使用同一个stringstream对象，记住再每次转换前要使用clear()方法；

在多次转换中重复使用同一个stringstream（而不是每次都创建一个新的对象）对象最大的好处在于效率。stringstream对象的构造和析构函数通常是非常耗费CPU时间的。

在类型转换中使用模板

你可以轻松地定义函数模板来将一个任意的类型转换到特定的目标类型。例如，需要将各种数字值，如int、long、double等等转换成字符串，要使用以一个string类型和一个任意值*t*为参数的to_string()函数。to_string()函数将*t*转换为字符串并写入result中。使用str()成员函数来获取流内部缓冲的一份拷贝：

template<class T>

void to_string(string & result,const T& t)

{

ostringstream oss;//创建一个流

oss<<t;//把值传递如流中

result=oss.str();//获取转换后的字符转并将其写入result
}

这样，你就可以轻松地将多种数值转换成字符串了：

to_string(s1,10.5);//double到string

to_string(s2,123);//int到string

to_string(s3,true);//bool到string

可以更进一步定义一个通用的转换模板，用于任意类型之间的转换。函数模板convert()含有两个模板参数out_type和in_value，功能是将in_value值转换成out_type类型：

template<class out_type,class in_value>

out_type convert(const in_value & t)

{

stringstream stream;

stream<<t;//向流中传值

out_type result;//这里存储转换结果

stream>>result;//向result中写入值

return result;

}

这样使用convert()：

double d;

string salary;

string s="12.56";

d=convert<double>(s);//d等于12.56

salary=convert<string>(9000.0);//salary等于”9000”

结论

在过去留下来的程序代码和纯粹的C程序中，传统的<stdio.h>形式的转换伴随了我们很长的一段时间。但是，如文中所述，基于**stringstream的转换拥有类型安全和不会溢出这样抢眼的特性**，使我们有充足得理由抛弃<stdio.h>而使用<sstream>。<sstream>库还提供了另外一个特性—可扩展性。你可以通过重载来支持自定义类型间的转换。

一些实例：

stringstream通常是用来做数据转换的。

相比c库的转换，它更加安全，自动和直接。

例子一：基本数据类型转换例子 int转string

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

int main()
{
   std::stringstream stream;
   std::string result;
   int i = 1000;
   stream << i; //将int输入流
   stream >> result; //从stream中抽取前面插入的int值
   std::cout << result << std::endl; // print the string "1000"
}
```

运行结果：

例子二：除了基本类型的转换，也支持char *的转换。

```
#include <sstream>
#include <iostream>

int main()
{
   std::stringstream stream;
   char result[8] ;
   stream << 8888; //向stream中插入8888
   stream >> result; //抽取stream中的值到result
   std::cout << result << std::endl; // 屏幕显示 "8888"
}
```

例子三：再进行多次转换的时候，必须调用stringstream的成员函数clear().

```
#include <sstream>
#include <iostream>
int main()
{
   std::stringstream stream;
   int first, second;
   stream<< "456"; //插入字符串
   stream >> first; //转换成int
   std::cout << first << std::endl;
   stream.clear(); //在进行多次转换前，必须清除stream
   stream << true; //插入bool值
   stream >> second; //提取出int
   std::cout << second << std::endl;
}
```

运行clear的结果

没有运行clear的结果

注：关于stream.clear()和stream.str(""），作用还不太清楚。又说clear是清除标志位，str("")是清楚stream内容的。但在多次转换过程是，的确是使用clear才准确，这是验证过的。