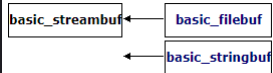


C/C++编程：流缓冲类std::basic_streambuf _std::streambuf_OceanStar的学习笔记的博客-CSDN博客

成就一亿技术人!

```
template<
class CharT,
class Traits = std::char_traits<CharT>
> class basic_streambuf;
```



该模板类的主要作用是**抽象原生设备**

该模板被设计为所有流缓冲区类的基础虚拟类：

IO流对象象 [std::basic_istream](#) 及 [std::basic_ostream](#)，还有所有派生自它们的对象（[std::ofstream](#)、[std::stringstream](#) 等），都完全以 [std::basic_streambuf](#) 实现。

在内部，[basic_streambuf](#)class是精心设计的基类，旨在为所有派生类提供统一的公共接口；这些公共函数调用派生类可以重写以实现特定行为的虚拟受保护成员。这些被覆盖的虚拟功能可以通过一组受保护的函数访问内部的[basic_streambuf](#)类

、

流缓冲区对象在内部至少保留：

一种 locale 对象，用于与语言环境相关的操作。

一组内部指针，用于保留输入缓冲区：[eback](#), [gptr](#), [egptr](#)。

一组内部指针，用于保留输出缓冲区：[pbase](#), [pptr](#), [epptr](#)。

模板实例化

[streambuf](#)

[wstreambuf](#)

成员函数

pubimbue

```
locale pubimbue (const locale& loc);
```

功能：

将loc 关联到stream buffer.

返回值：

返回stream buffer.在调用这个函数之前的locale

getloc

```
locale getloc() const;
```

功能：

获取当前语言环境

如果成员函数 [pubimbue](#) 已经被这个 stream buffer调用，那么返回设置的值，否则返回当前的全局语言环境

返回值：

返回locale对象当前与流缓冲区关联的对象。

in_avail

```
streamsize in_avail ();
```

功能：

获取可读取的字符数

此值取决于get指针（是否有直接可用的读取位置gptr）

返回值：

可读取的字符数

值-1表示没有其他字符可用

```
#include <iostream>
```

```
#include <fstream>
```

```
int main () {
```

```
std::ifstream ifs ("test.txt");
```

```
if (ifs.good()) {
```

```
    std::streambuf* pbuf = ifs.rdbuf();
```

```
char c; ifs >> c;
```

```
std::streamsize size = pbuf->in_avail();

std::cout << "first character in file: " << c << '\n';

std::cout << size << " characters in buffer after it\n";

}

ifs.close();

return 0;
}
```

编辑并运行

first character in file: s
2 characters in buffer after it

sngetc、sbumpc、sgetc、sgetn

```
int_type sbumpc ();
```

功能：
sbumpc：获取当前字符并前进到下一个位置
返回值：
返回受控输入序列当前位置处的字符，并将位置指示器前进到下一个字符。
请注意，尽管类似，但以下成员函数具有不同的行为：

成员功能	描述
sgetc()	返回当前位置的字符。
sbumpc()	返回当前位置的字符，并将当前位置前进到下一个字符。
sngetc()	将当前位置前进到下一个字符并返回下一个字符。

```
int_type sgetc ();
int_type sngetc ();
```

功能：
sgetc：从输入序列中读取一个字符
sngetc：前进到下一个位置并读取一个字符
如果输入序列读取位置不可用，则返回[underflow\(\)](#)
返回值：
get指针指向的字符的值

```
streamsize sgetn (char_type * s, streamsize n);
```

功能：
sgetn：获取字符序列
返回值：
成功读取的字符数。如果小于count输入序列，则结束。

```
#include <iostream>
#include <fstream>

效果：逐个字符读取文件内容并输入

int main () {
std::ifstream istr ("test.txt");
if (istr) {
std::streambuf * pbuf = istr.rdbuf();
while ( pbuf->sgetc() != std::streambuf::traits_type::eof() )
{
char ch = pbuf->sbumpc();
std::cout << ch;
}
istr.close();
}
return 0;
}
```

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

int main()
{
```

```
std::stringstream stream("Hello, world");

std::cout << "sgetc() returned '" << (char)stream.rdbuf()->sgetc() << "'\n";

std::cout << "peek() returned '" << (char)stream.peek() << "'\n";

std::cout << "get() returned '" << (char)stream.get() << "'\n";

}
```

sgetc() returned 'H'
peek() returned 'H'
get() returned 'H'

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

int main()
{
std::stringstream stream("Hello, world");

std::cout << "sgetc() returned '" << (char)stream.rdbuf()->sngetc() << "'\n";

}
```

```
if (sgetc() returned 'e'

#include <iostream>
#include <fstream>

int main () {
std::ifstream istr ("test.txt");
if (istr) {
std::streambuf * pbuf = istr.rdbuf();

std::streamsize size = pbuf->pubseekoff(0,istr.end);
pbuf->pubseekoff(0,istr.beg);

char *contents = new char[size];

pbuf->sgetn(contents, size);

istr.close();

std::cout.write(contents, size);
}
return 0;
}
```

放置区	
sputc	写一个字符到放置区域，并令 next 指针前进
sputn	调用 xputn()

std::basic_streambuf::sputc、std::basic_streambuf::sputn

```
int_type sputc( char_type ch );
```

功能：

sputc：将一个字符写入输出序列。

sputn：将n个字符写入到输出序列

如果输出序列写位置不可用（缓冲区已满），则调用overflow(ch)

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

int main ()
{
std :: ostringstream s ;
s.rdbuf ()- > sputc ('a');
```

```
std::cout << s.str() << ' \n ' ;
}
```

a

```
#include <iostream>
#include <fstream>

int main () {
    const char sentence[] = "Sample sentence";

    std::ofstream ostr ("test.txt");
    if (ostr) {
        std::streambuf * pbuf = ostr.rdbuf();
        pbuf->sputn (sentence, sizeof(sentence)-1);
        ostr.close();
    }

    return 0;
}
```



std::basic_streambuf::sputbackc

```
int_type sputbackc (char_type c) ;
```

功能：将字符放回获取区域

```
#include <iostream>

int main()
{
    char ch;

    std::streambuf *pbuf = std::cin.rdbuf();

    std::cout << "Please, enter some letters and then a number: ";

    do {
        ch = pbuf->sgetc();

        if ( (ch>='0') && (ch<='9') )
        {
            pbuf->sputbackc(ch);
        }

        long n;

        std::cin >> n;

        std::cout << "You entered number " << n << '\n';

    } while ( ch != std::streambuf::traits_type::eof() );

    return 0;
}
```

本示例从标准输入中一个接一个地获取字符。找到第一个数字后，反击被调用以将流中的位置恢复到该数字，以便使用提取运算符将其提取为数字的一部分>>。

```
Please, enter some letters and then a number: 123
You entered number 123
|
```

std::basic_streambuf::sungetc

```
int_type sungetc () ;
```

功能：

如果在获取区域有一个后备仓位（`gptr () > eback ()`），然后递减下一个指针（`gptr ()`）并返回它现在指向的字符。

如果没有备用位置，则在可能的情况下调用`ebackfail ()`备份输入序列。

I/O流功能[basic_istream::unget](#)是根据此功能实现的

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

int main()
{
    std::stringstream s("abcdef");
    char c1 = s.get();
    char c2 = s.rdbuf()->sungetc();
    char c3 = s.get();
    char c4 = s.get();

    std::cout << c1 << c2 << c3 << c4 << '\n';

    s.rdbuf()->sungetc();
    s.rdbuf()->sungetc();

    int eof = s.rdbuf()->sungetc();
    if (eof == EOF)
        std::cout << "Nothing to unget after 'a'\n";
}
```

```
aaab
Nothing to unget after 'a'
```

寻位	
setbuf 、 pubseekoff 、 pubseekoff	若容许则以用户定义数组替换缓冲区
seekoff	用相对寻址重定位输入序列、输出序列或两者中的下一位置指针
seekpos	用绝对寻址重定位输入序列、输出序列或两者中的下一位置指针
sync	将缓冲与关联的字符序列同步

std::basic_streambuf::pubsetbuf、std::basic_streambuf::setbuf

```
basic_streambuf<CharT, Traits>* pubsetbuf( char_type* s, std::streamsize n ) (1)

protected:
virtual basic_streambuf<CharT, Traits>* setbuf( char_type* s, std::streamsize n ) (2)
```

功能：设置相对于其他位置的输入或者输出序列的位置指示器

- 1) 调用最终派生类上的 `setbuf(s, n)`
- 2) 此函数的基类版本无效果。派生类可覆写此函数，以允许移除或替换受控制字符序列（缓冲区）为用户提供的数组，或为任何实现特定的目的。

参数

s	-	指向用户提供的缓冲区中首个 CharT 的指针
n	-	用户提供缓冲区中的 CharT 元素数

返回值

- 1) `setbuf(s, n)` 的返回值
- 2) `this`

```
#include <iostream>
#include <fstream>

int main () {
    std::fstream filestr ("test.txt");
    if (filestr) {
        std::streambuf* pbuf = filestr.rdbuf();
        long size = pbuf->pubseekoff(0,std::fstream::beg);

        std::cout << "The file size is " << size << " characters.\n";
        filestr.close();
    }

    return 0;
}
```

```
.....
The file size is 8 characters.
```

```
#include <fstream>
```

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

int main()
{
    int cnt = 0;
    std::ifstream file;
    char buf[1024*10 + 1];

    file.rdbuf()->pubsetbuf(buf, sizeof buf);

    file.open("/usr/share/dict/words");

    for (std::string line; getline(file, line);) {
        cnt++;
    }

    std::cout << cnt << '\n';
}
```

std::basic_streambuf::pubseekoff、std::basic_streambuf::seekoff

```
pos_type pubseekoff( off_type off, std::ios_base::seekdir dir,
                    std::ios_base::openmode which = ios_base::in | ios_base::out );    (1)
protected:
virtual pos_type seekoff( off_type off, std::ios_base::seekdir dir,
                        std::ios_base::openmode which = ios_base::in | ios_base::out );    (2)
```

相对某其他位置，设置输入/输出序列的位置指示器。

- 1) 调用最终导出类的 seekoff(off, dir, which)
- 2) 此函数的基类版本无效果。导出类可覆写此函数以允许位置指示器的相对寻位。

参数

off	要设置位置指示器到的相对位置。	
dir	定义要应用相对偏移到的基位置。它能为下列常量之一：	
	常量	解释
	beg	流的开始
	end	流的结尾
	cur	流位置指示器的当前位置
which	定义输入和/或输出序列何者有影响。它能为下列常量之一或其组合：	
	常量	解释
	in	影响输入序列
	out	影响输出序列

```
#include <iostream>
#include <fstream>

int main () {

    std::fstream filestr ("test.txt");
    if (filestr) {
        std::streambuf* pbuf = filestr.rdbuf();
        long size = pbuf->pubseekoff(0,filestr.end);
        if (size>20) {
            char buffer[11];

            pbuf->pubseekpos(10);

            pbuf->sgetn (buffer,10);

            buffer[10]=0;
        }
    }
}
```

```

        std::cout << buffer << '\n';
    }
    filestr.close();
}
return 0;
}

```

效果：读取并打印文件的第11至20个字符

std::basic_streambuf::sync、std::basic_streambuf::pubsync

```

int pubsync () ;          (1)

protected:
virtual int sync();      (2)

```

功能：

(1) 同步流缓冲区

(2) 此函数的基类版本无效果。导出类可覆写此函数以允许将底层设备与缓冲区同步。

返回值：

默认定义 [同步](#) 在 [流缓冲](#) 总是返回零，表示成功。

派生类可以覆盖此默认行为，并最终返回-1以指示失败。

```

#include <iostream>
#include <fstream>

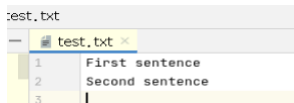
int main () {
    std::ofstream ostr ("test.txt");
    if (ostr) {
        std::streambuf * pbuf = ostr.rdbuf();

        pbuf->sputn ("First sentence\n",15);
        pbuf->pubsync();
        pbuf->sputn ("Second sentence\n",16);

        ostr.close();
    }
    return 0;
}

```

test.txt



```

1 First sentence
2 Second sentence
3

```

std::basic_streambuf::pbump

```

#include <iostream>
#include <string>
#include <fstream>

struct showput_streambuf : std::filebuf
{
    using std::filebuf::pbump;
    std::string showput() const {
        return std::string(pbase(), pptr());
    }
};

int main()
{
    showput_streambuf mybuf;
}

```

```

    mybuf.open("test.txt", std::ios_base::out);

std::ostream str(&mybuf);

    str << "This is a test" << std::flush << "1234";

    std::cout << "The put area contains: " << mybuf.showput() << '\n';

    mybuf.pbump(10);

    std::cout << "after pbump(10), it contains " << mybuf.showput() << '\n';
}

```

std::basic_streambuf::setp

```
void setp (char_type* new_pbase, char_type* new_epptr);
```

设置指针的值，这些指针定义受控输出序列的缓冲部分的边界([pbase](#) and [epptr](#))。

该置入指针 ([pptr](#)) 会自动设置为该序列的开头。

这是一个受保护的成员，其他成员函数可以调用该成员来更改描述受控输出序列的缓冲部分的数组。

```

#include <iostream>

#include <array>

#include <streambuf>

template<std::size_t SIZE, class CharT = char>
class ArrayedStreamBuffer : public std::basic_streambuf<CharT>
{
public:
    using Base = std::basic_streambuf<CharT>;
    using char_type = typename Base::char_type;

    ArrayedStreamBuffer() : buffer_{}
    {
        Base::setp(buffer_.begin(), buffer_.end());
    }

    void print_buffer()
    {
        for (const auto& i: buffer_) {
            if (i == 0) {
                std::cout << "\\0";
            } else {
                std::cout << i;
            }
            std::cout << ' ';
        }
        std::cout << '\n';
    }

private:
    std::array<char_type, SIZE> buffer_;
};

int main () {
    ArrayedStreamBuffer<10> streambuf;
    std::ostream stream(&streambuf);

    stream << "hello";
    stream << ", ";

    streambuf.print_buffer();

    return 0;
}

```



```
}  
-----  
hello, \0 \0 \0 \0
```