中南大學
CENTRAL SOUTH UNIVERSITY

面向对象程序设计

Object Oriented Programing

第六章 STL、I/O流类和异常处理

张宝一

地理信息系

zhangbaoyi@csu.edu.cn

第六章：STL、I/O流类和异常处理

- 标准模板库STL 
- I/O流类
- 异常处理
- 本章小结

第六章：STL、I/O流类和异常处理

- 标准模板库（Standard Template Library，STL）是美国国家标准化组织和国际标准化组织于98年制定的标准，其最主要与最常用的三部分为：

1. STL容器类container classes：**存放其他对象（和类型相似）的对象**

vector stack queue deque list set map等

2. STL算法库algorithm library：

copy sort search merge permute等

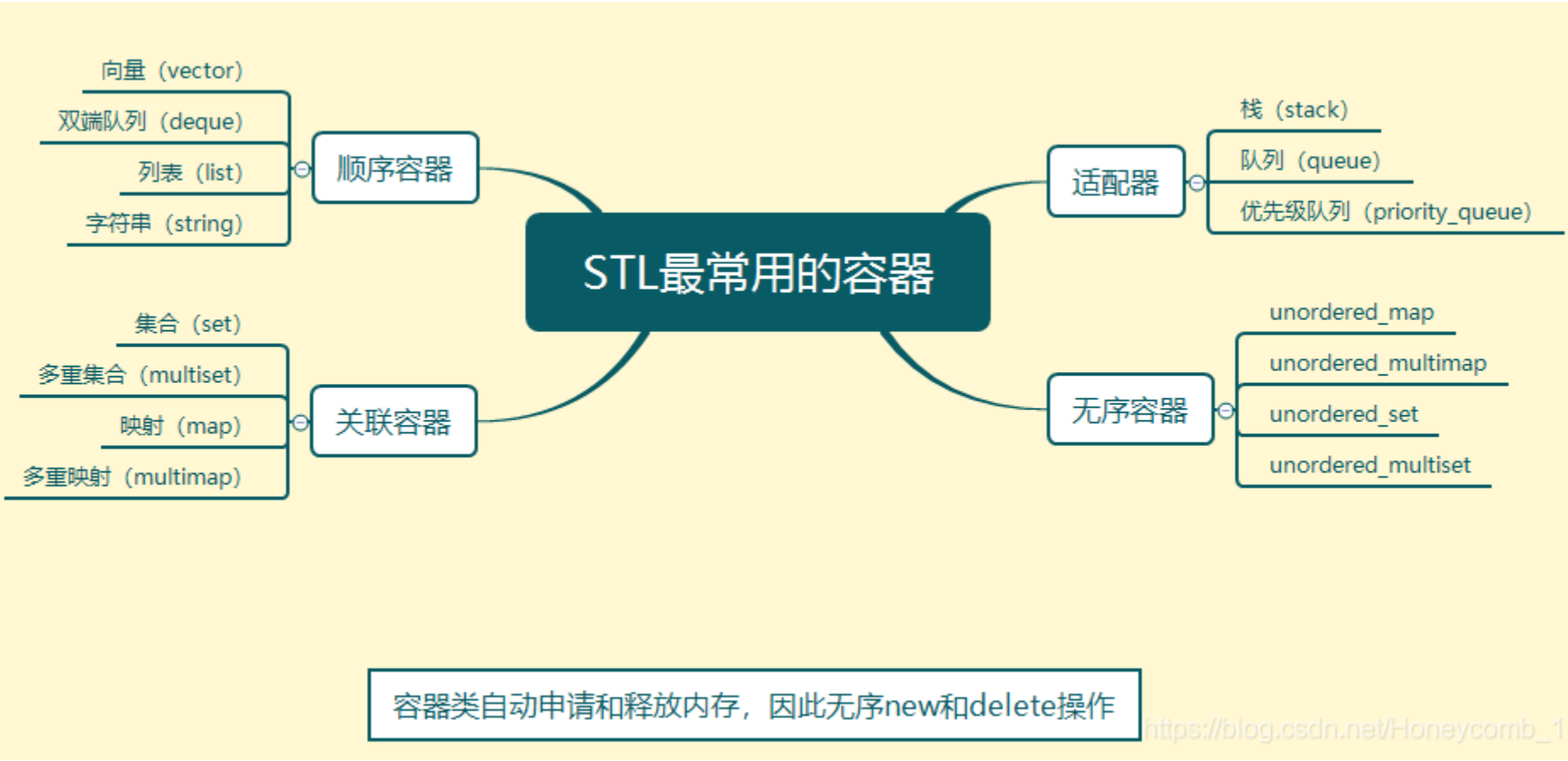
3. STL提供了多种类型的迭代器，可分别进行正向、反向、双向和随机遍历操作。

6.1 STL标准模板库

- 字符串类string
- vector
- list

6.1 STL标准模板库

6.1.1 C++ STL中使用类模板定义的常用容器



6.1 STL标准模板库

6.1.1 字符串类——string类

string：一个专门存储和处理字符类型的STL容器，具有强大的功能。

- #include <string>
- string类是表示字符串的字符串类
- string在底层实际是：basic_string模板类的别名，

  typedef basic_string<char, char_traits, allocator>

  string
- 不能操作多字节或者变长字符（wchar_t）的序列

6.1 STL标准模板库

6.1.1 字符串类——string类

string：一个专门存储和处理字符类型的STL容器，具有强大的功能。

```
void main()
{
    string s1;   // 构造空的string类对象s1
    string s2("hello");   // 用常字符串构造string类对象s2
    string s3(s2);   // 拷贝构造s3
}
```

6.1 STL标准模板库

6.1.1 字符串类——string类

常用成员函数	实现功能描述
size()	返回字符串有效长度
empty()	判断字符串是否为空，是返回true，否则返回false
clear()	清空有效字符，空间没有释放（不改变底层空间大小）
operator+=	在字符串后面追加单个字符或字符串
operator()	返回pos位置的字符
begin(), end()	begin返回一个迭代器，它指向容器c的第一个元素，end获取最后一个字符下一个位置的迭代器
c_str	返回const char*格式字符串
find	从字符串pos位置开始往后找字符c，返回该字符位置
substr	在str中从pos位置开始，截取n个字符，然后返回

6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

vector容器的声明遵循C++ STL的一般声明原则：

vector<变量类型> 变量名称

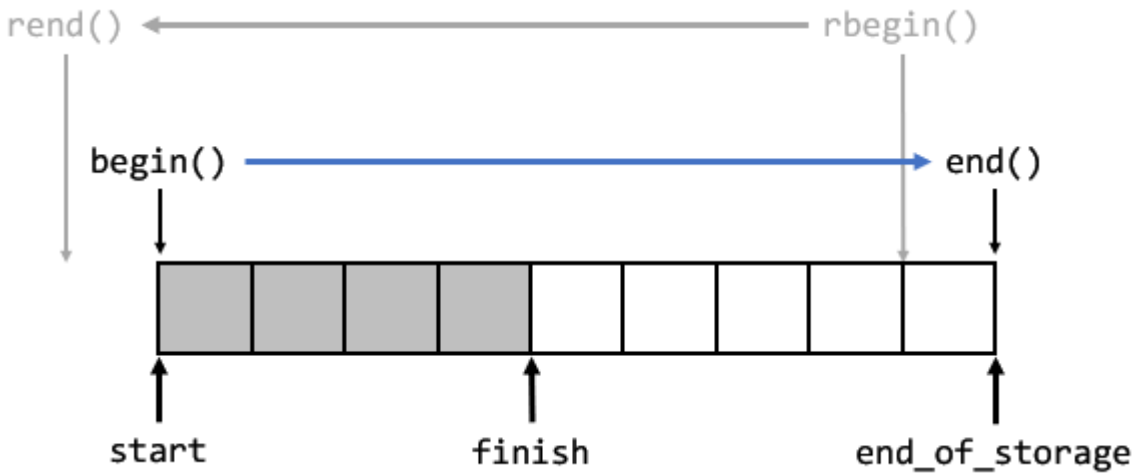
例：

```
#include<vector>
vector<int> vec;
vector<string> vec;
vector<double> vec;
struct node{...};  vector<node> vec;
```

6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

```
// TEMPLATE CLASS vector
template<class _Ty,   class _Alloc = allocator<_Ty> >
class vector
    : public _Vector_alloc<_Vec_base_types<_Ty, _Alloc> >
{
    // varying size array of values
};
```



6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

vector：一个封装了动态大小数组的顺序容器。它能够存放各种类型的对象，vector是一个能够存放任意类型的动态数组。

- #include<vector>
- vector使用动态分配数组来存储元素，每插入一个新元素时，数组就扩容（当数组满时，分配一个新数组，然后将元素转移到此数组）
- vector访问元素高效，末尾添加和删除元素相对高效。对于其它元素的删除和插入操作，效率低

6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

常用成员函数	实现功能描述
begin(), end()	返回vector的首、尾**迭代器**
front(), back()	返回vector的首、尾**元素**
push_back()	从vector末尾加入一个元素
size()	返回vector当前的长度（大小）
pop_back()	从vector末尾删除一个元素
clear()	清空vector
capacity	获取容量大小
empty	判断是否为空（空返回true，非空返回false）
resize(size_t n);	改变容器中存储数组的大小
reserve	改变容器大小，即capacity

6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

```
#include <vector>
#include <iostream>
int main( ){
  using namespace std;
  vector <int>::iterator v1_Iter, v2_Iter, v3_Iter, v4_Iter, v5_Iter;
  // Create an empty vector v0
  vector <int> v0;
  // Create a vector v1 with 3 elements of default value 0
  vector <int> v1( 3 );
  // Create a vector v2 with 5 elements of value 2
  vector <int> v2( 5, 2);
```

6.1 STL标准模板库

6.1.2 可变大小数组——vector类模板

```
// Create a vector v3 with 3 elements of value 1 and with the
//allocator  of vector v2
vector <int> v3( 3, 1, v2.get_allocator( ) );
// Create a copy, vector v4, of vector v2
vector <int> v4( v2 );
// Create a vector v5 by copying the range v4[_First, _Last)
vector <int> v5( v4.begin( ) + 1, v4.begin( ) + 3 );
cout << "v1 =" ;
for ( v1_Iter = v1.begin( ) ; v1_Iter != v1.end( ) ; v1_Iter++ )
        cout << " " << *v1_Iter;
```

6.1 STL标准模板库

6.1.3 双向链表——list类模板

list容器的声明遵循C++ STL的一般声明原则：

list<变量类型> 变量名称

例：

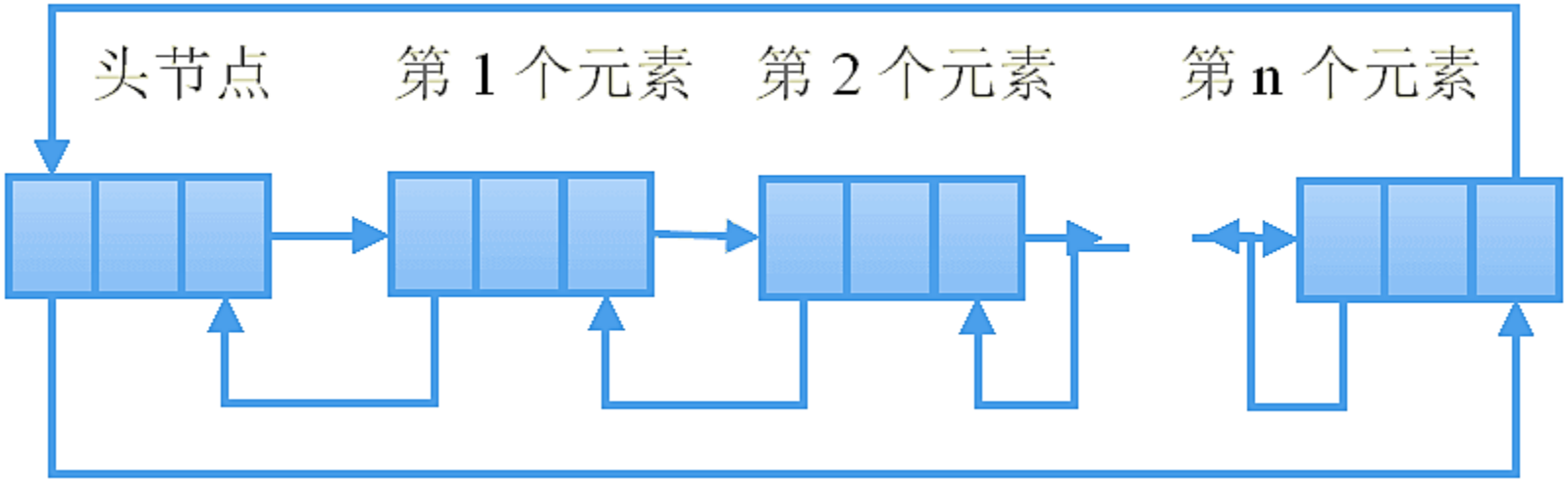
```
#include<list>
list<int> li;
list<string> ls;
list<double> ld;
struct node{...};  list<node> ln;
```

6.1 STL标准模板库

6.1.3 双向链表——list类模板

list：实现了双向链表的数据结构，数据元素是通过链表指针串成逻辑意义上的线性表，选择对链表的任一位置的元素进行插入，删除和查找都是非常高效的。

- #include<list>



6.1 STL标准模板库

6.1.3 双向链表——list类模板

□ 创建list

```
# 创建一个空的双向链表
list<int> L;
# 创建一个具有n个int元素的双向链表
list<int> L(10);
```

6.1.3 双向链表——list类模板

口 元素插入和元素遍历

- **push_back**：在尾部插入元素。
- **push_front**：在首部插入元素。
- **insert**：在指定位置（迭代器）处插入某个元素。
- **find**：查找可以在链表中查找元素，如果找到该元素，则返回该元素的迭代器位置，反之，则返回end()迭代器位置。

6.1.3 双向链表——list类模板

口 元素删除

- remove()：删除链表中一个元素，值相同的元素都会被删除。
- pop_front()：删除链表首元素。
- pop_back()：删除链表尾元素。
- erase()：删除迭代器位置上的元素。
- clear()：清空链表。

6.1 STL标准模板库

6.1.3 双向链表——list类模板

□ 排序等函数

- **sort()**：对链表进行排序，升序排列。
- **unique()**：可以剔除连续重复元素，只保留一个。

```
list<int> L;
L.push_back(3);
L.push_back(1);
L.push_back(2);
L.push_back(3);
L.push_back(4);
```


```
L.sort();
```


```
L.unique();
```

6.1 STL标准模板库

6.1.4 关联容器——map类模板

- 映像容器是一种特殊的集合（参看后面介绍的集合容器set），也有称其为字典（dictionary）或关联数组（associative array）的。
- 映像容器提供对“（key，value）”数对进行有效存取与管理的机制。其中的key 是作为键出现的，而value 则作为对应于该键的一个具体数据值。要求键key在容器中是唯一的，而其对应的value 数据值则可以重复。只所以系统能对容器map 进行有效的存取管理，是因为可以按照排序后的key 值来对容器进行维护。
- 该容器也提供诸如begin 及end 这样的游标（迭代子）；重载了下标运算符“[]”，以进行基于key 值的存取与插入；
- 通过使用成员函数find、count、lower_bound、upper_bound，可用于查找或统计基于key 值的元素。另外，也提供insert、erase、empty、size 等成员函数。

6.1 STL标准模板库

6.1.4 关联容器——map类模板

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

int main()
{
   map< string, int>  m;
   m[ "zero" ] = 0;
   m[ "one" ] = 1;
   m[ "two" ] = 2;
   m[ "three" ] = 3;
   m[ "four" ] = 4;
   m[ "five" ] = 5;
   m[ "six" ] = 6;
   m[ "seven" ] = 7;
   m[ "eight" ] = 8;
   m[ "nine" ] = 9;
   cout<< m[ "three" ]
       <<endl
        << m[ "five" ] <<endl
        << m[ "seven" ]<<endl;

return 0;
}
```

6.1 STL标准模板库

6.1.4 关联容器——multimap类模板

- 与一般映像map 不同的是，多重映像multimap 中允许出现相同的键值。多重映像容器中，没重载下标运算符“[]”。
- 本容器的成员函数“iterator find(const Key& key);”，返回的是容器中第一个键值为key 的iterator（游标值）-- 注意该容器允许出现相同键值。其他成员函数与一般映像map 相类似。

6.1 STL标准模板库

6.1.4 关联容器——set类模板

- 模拟数学中集合的概念，不可有重复元素，意味着是一个无序的集合（虽然系统实现时按有序元素列来处理，以提高实现效率，但并不影响原有的有关集合的数学概念）。
- 常用的成员函数有：begin、end、erase、clear、count、empty、find、insert、lower_bound、upper_bound、size 等。
- 与前面介绍的容器map 有类似之处，只是没有关键字key 而已，所以也就没有（也不需要）重载下标运算符“[]”。

6.1 STL标准模板库

6.1.4 关联容器——set类模板

```
set< int >  s;
s.insert( - 999 );     s.insert( 18 );
s.insert( 321 );s.insert( 18 );
s.insert( -999 );
set< int >::const_iterator  it; it = s.begin();
    while ( it != s.end()) cout << *it++ << endl;
int key;
cout<< "Enter an integer: ";    cin >> key;
it = s.find( key );
if  (it == s.end()) cout<< key << "is not in set." <<endl;
else cout<< key << "is in set." << endl;
```

6.1 STL标准模板库

6.1.4 关联容器——multiset类模板

- 与一般集合set 不同的是，多重集合multiset 中允许出现相同的元素。表面上该容器也是一个无序的集合（虽然系统实现时按有序元素列来处理，以提高实现效率）。常用的成员函数有许多都与set 相同，不再赘述。
- 若想找出多重集合容器中所有值为k 的元素，并返回这些值所对应的迭代子iterator（泛型指针）的范围，可使用如下的成员函数：

```
pair<const_iterator,
const_iterator>equal_range(const Key& k)
const;
```

6.1 STL标准模板库

6.1.5 迭代器

- 每个容器的类描述包含该容器类所提供的迭代器类
- 每个通用算法的描述包含该算法所使用的迭代器和容器种类
- 迭代器支持的操作：
  - 是否相等== 是否不等!=
  - *间接引用对象

6.1 STL标准模板库

6.1.5 迭代器

```
#include <iostream>
#include <list>
using namespace std;
typedef list<int>          INTLIST;
ostream& operator<< ( ostream& os, const INTLIST& s){
   INTLIST::const_iterator       i = s.end();
   do{
            --i;
            os << *i << endl;
   }while(i != s.begin());
   return os;
}
```

6.1 STL标准模板库

6.1.6 算法库

- 算法库中则包括了各种基本算法，如，sort，copy，search，reverse 等。通过使用算法，可对容器中的对象进行诸如查找、排序、拷贝、置换等各种不同的操作。C++标准算法库中共提供了多达数十种的不同算法，而这些算法主要通过一种称为迭代子（iterator）也有称为游标的面向对象的泛型指针来操作（遍历容器中的不同对象），以达到对各对象进行处理的目的。

第六章：STL、I/O流类和异常处理

- 标准模板库STL
- I/O流类 
- 异常处理
- 本章小结

6.2 I/O流类

6.2.1 C++中输入/输出流类

I/O流库中的常用类

类　名	作　　用	头文件声明
ios	抽象基类	iostream
istream ostream iostresam	通用输入流和其他输入流的基类 通用输出流和其他输出流的基类 通用输入输出流和其他输入输出流的基类	iostram
ifstream ofstream fstream	输入文件流类 输出文件流类 输入输出文件流类	fstream
istrstream ostrstream strstream	输入字符串流类 输出字符串流类 输入输出字符串流类	strstream

6.2 I/O流类

6.2.1 C++中输入/输出流类

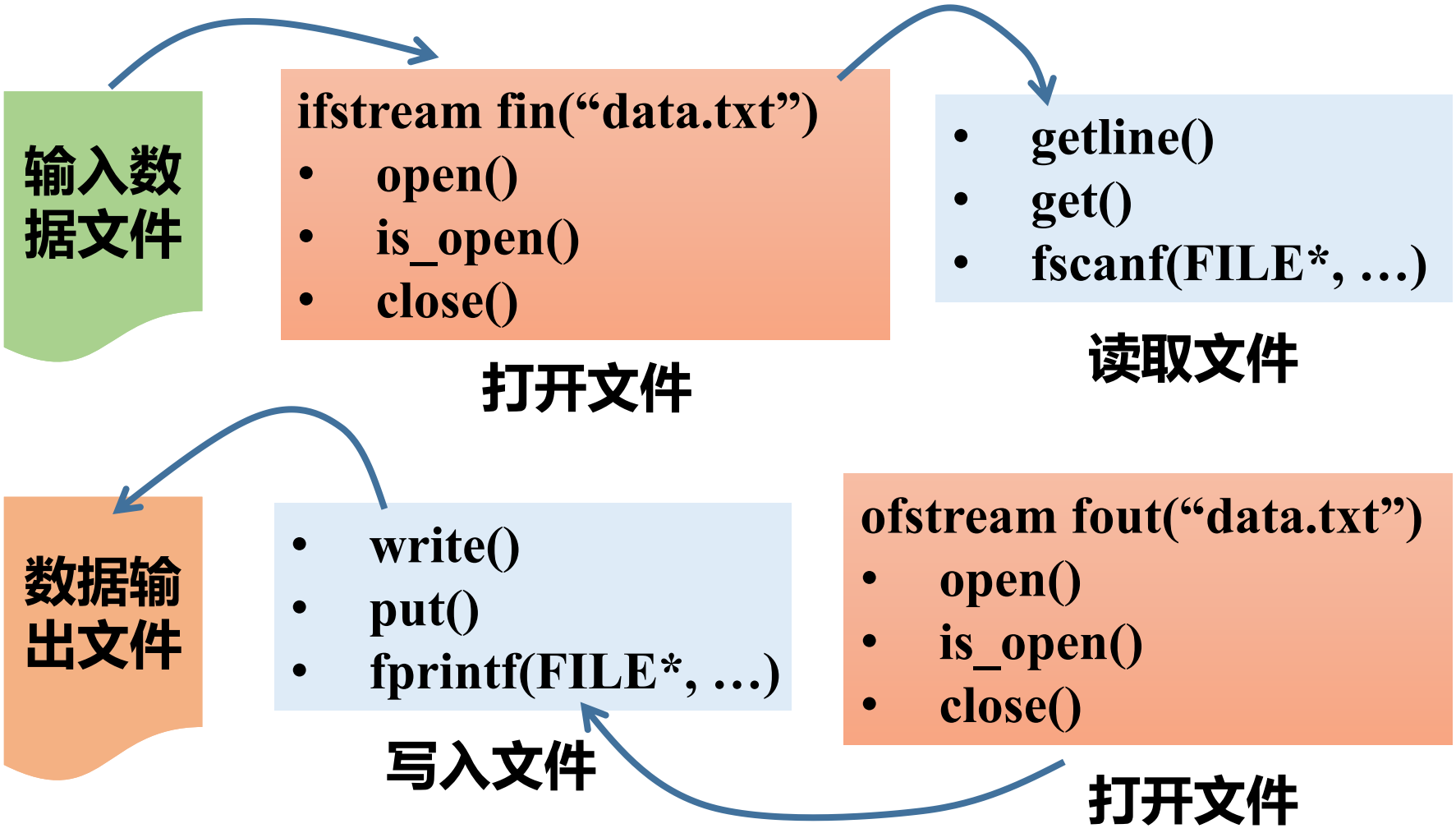
C++ 通过以下几个类支持文件的输入输出：

使用文件I/O流类需要#include < fstream>

- **ofstream**：写操作（输出）的文件类（由ostream引申而来），将文件从内存写入存储设备（本地磁盘）
- **ifstream**：读操作（输入）的文件类(由istream引申而来)，将文件从存储设备加载到内存中
- **fstream**：可同时读写操作的文件类（由iostream引申而来)

6.2 I/O流类

6.2.1 C++中输入/输出流类



6.2 I/O流类

6.2.2 打开文件

Member Function	Meaning
open(*name*)	Opens a file for the stream using the default mode
open (*name*, *flags*)	Opens a file for the stream using *flags* as the mode
close()	Closes the streams file
is_open()	Returns whether the file is opened

标　志	函　数
ios::in	打开一个输入文件，用这个标志作为ifstream的打开方式，以防止截断一个现成的文件
ios::out	打开一个输出文件，当用于一个没有ios::app、ios::ate或ios::in的ofstream时，ios::trunc被隐含
ios::app	以追加的方式打开一个输出文件
ios::ate	打开一现成文件（不论是输入还是输出）并寻找末尾
ios::nocreate	仅打开一个存在的文件（否则失败）
ios::noreplace	仅打开一个不存在的文件（否则失败）
ios::trunc	如果一个文件存在，打开它并删除旧的文件
ios::binary	打开一个二进制文件，缺省的是文本文件

6.2 I/O流类

6.2.2 打开文件

如果open函数只有一个文件名参数，则以默认的方式打开文件，几个类的默认方式为：

- fstream f;　f.open(“data.txt”);

  fstream：　ios::in | ios::out

- ifstream f;　f.open(“data.txt”);

  ifstream：　ios::in

- ofstream f;　f.open(“data.txt”);

  ofstream：ios::out | ios::trunc

6.2 I/O流类

6.2.2 打开文件

- 判断文件是否成功打开

a）成员函数is_open()来检查一个文件是否已经被顺利打开。

```
fstream file(“Hello.txt”, ios::out | ios::app | ios::binary);
if ( file.is_open() )
{// add code
  return ture;
}
```

6.2 I/O流类

6.2.2 打开文件

- 判断文件是否成功打开

b) 直接对文件流对象进行判断

```
fstream file("Hello.txt", ios::out | ios::app | ios::binary);
if(!file)  //! 操作符已被重载
{ // add code
    return false;
}
```

6.2.3 关闭文件

当文件读写操作完成之后，我们必须将文件关闭以使文件重新变为可访问的。关闭文件需要调用成员函数close()，它负责将**缓存中的数据排放出来**并关闭文件，其函数原型为：

```
void close();
```

6.2 I/O流类

6.2.4 ASCII文件读写

- **读取**：使用流提取运算符（ >> ）从文件读取信息。唯一不同的是，使用的是ifstream 或 fstream 对象，而不是 cin 对象。读数据时，遇到空格或者换行返回停止读入。

- **写入**：使用流插入运算符（ << ）向文件写入信息。唯一不同的是，使用的是 ofstream 或 fstream 对象，而不是 cout 对象。

- 使用文件流的put，get，getline等成员函数进行字符的输入输出。

6.2 I/O流类

6.2.5 二进制文件读写

- 对二进制文件的操作
  - 用成员函数read和write读写二进制文件
  - 与文件指针有关的流成员函数
  - 随机访问二进制文件

第六章：STL、I/O流类和异常处理

- 标准模板库STL
- I/O流类
- 异常处理 
- 本章小结

6.3 异常处理

6.3.1 异常处理

```
int a=2;
int b = 0;
int c = a/b;   // ?
```

使用异常处理的好处：

- 提升代码的健壮性。
- 方便以代码的调试与维护。
- 程序设计需要异常处理。

6.3 异常处理

6.3.1 异常处理

C++ 异常处理涉及到三个关键字：

try、catch、throw

throw: 使用 throw 关键字， 当程序出现问题时会抛出一个异常。

6.3 异常处理

6.3.1 异常处理

□ try-catch异常处理

```
try
{
    throw E();
}
catch (H h)
{
    // 何时我们可以能到这里呢
}
```

6.3 异常处理

6.3.1 异常处理

```
double division(int a, int b)
{
    if( b == 0 )
    {
        throw "Division by zero condition!";
        cout << "throw 后面的语句不再执行" <<endl;
    }
    return (a/b);
}
```

```
int main(void)
{
    try
    {
        division(5, 0);
    }
    catch(const char* error)
    {
        cout << error <<endl;
    }
    return 0;
}
```

本章小结

重点：

- STL常用的容器
- string
- vector
- List
- 文件输入输出

难点：

- vector的使用
- 文件输入输出的使用