生而有涯，而知无涯，已有求无，殆矣

## 一些关键字和概念

### namespace命名空间

1. 用于解决名称冲突
2. 命名空间必须声明在全局作用域下
3. 命名空间可以存放变量、函数、结构体、类

namespace A

{

int m\_A = 0;

void func() {};

struct Person {};

class Animal {};

}

1. 命名空间可以嵌套命名空间

namespace B

{

int m\_A = 100;

namespace C

{

int m\_A = 200;

}

}

1. 命名空间是开放的，可以随时向命名空间下添加成员

namespace B

{

int m\_A = 100;

}

namespace B

{

int m\_B = 1000;

}

1. 命名空间可以是匿名的

namespace

{

int m\_C = 10000; // 如果命名空间 是 匿名 变量前隐式加了关键字 static

int m\_D = 20000;

}

1. 命名空间可以起别名

namespace C

{

int m\_E = 100000;

}

namespace D = C;

void test()

{

namespace E =C;

cout << E::m\_E << endl;

cout << D::m\_E << endl;

cout << C::m\_E << endl;

}

### const

1. 全局区域const变量无法修改
2. 局部区域const变量如果使用常量赋值，也无法修改，但是：
   1. 编译能成功
   2. 运行也能成功
   3. 结果不会改变，因为当前变量指向的是符号表
   4. 修改的只是一个临时空间，耍你玩的
3. c语言的const默认是外部链接属性，c++的默认是内部链接属性
4. c++尽量使用const代替#define

### using

好像有声明，编译指令的意思 不太懂

1. using声明

using namespace std;

1. 提前告诉编译器 sunWuKongId用 KingGlory下的

using KingGlory::sunWuKongId;

### 引用

1.语法：

数据类型 & 引用名 = 原名

2.引用必须初始化

3.对数组引用

//直接建立引用

//pArr就是arr的引用名

int arr[5] = { 1, 2, 3, 4, 5 };

int(&pArr)[5] = arr;

//先定义数组类型，再通过类型定义数组的引用

typedef int(ARRAY\_TYPE)[5];

// 数据类型 & 别名 = 原名

ARRAY\_TYPE & pArr2 = arr;

4.引用本质上是一个指向不可修改的指针，不同的引用操作，编译器会做不同的“优化”

### 内联函数

1. 函数的声明和实现都加关键字inline

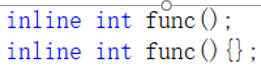
2. 用于替换宏函数

3. 类的成员函数都隐藏的加了关键字inline

4. inline只是建议编译器把该函数当做内联函数处理，具体是否是内联函数不一定

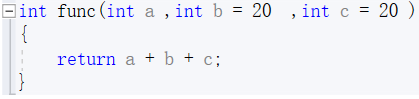
5. 不加inline关键字，编译器也可能把一些函数当做内联函数

示例：

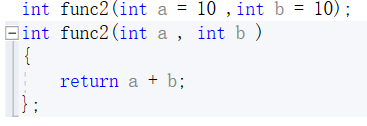


### 函数的默认参数

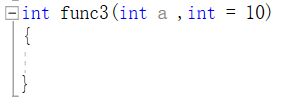
1. 如果某个位置已经有了默认参数，那么从这个位置起，后面的参数都必须有默认参数

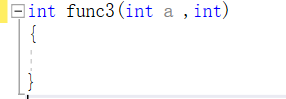


2. 函数的声明和实现只能有一个默认参数



3.占位参数





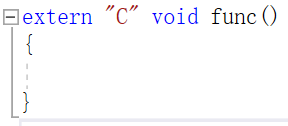
### 函数重载

1. 在同一作用域下，函数的名称相同，函数的参数个数、类型、顺序不同，代表了不同的函数

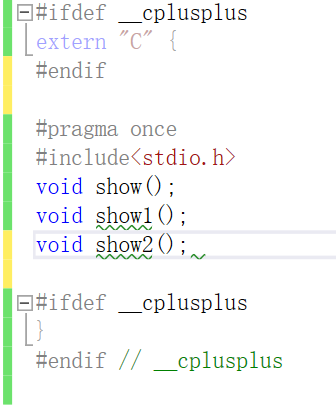
2. 实现原理是，编译器会改变函数真正的名称，所以这些函数的名称也是不同的

### exterm C

1. 方式一，单个使用



2. 方式二，批量使用、头文件使用，通过宏 \_\_cplusplus 在编译时自动判断当前编译器是C还是C++



### 异常处理

1. 关键字

throw 抛出异常

try 捕获异常 相当于if

catch 捕获异常 相当于else

2. 示例

[src\cpp\异常抛出\main.cpp](src/cpp/异常抛出/main.cpp)

多态继承的示例

[src\cpp\异常抛出\main1.cpp](src/cpp/异常抛出/main1.cpp)

### 标准输入cin

#### 1.从标准输入中读取一个字符

char ch = cin.get();

#### 2.从标准输入中读取一个字符串

cin.get( buf , sizeof(buf) );//不会读取换行符，换行符遗留在缓冲区中

cin.getline( buf , sizeof(buf) )//不会读取换行符，换行符扔掉，不会留在缓冲区

#### 3.忽略字符

cin.ignore();//忽略一个字符，括号中可以有参数，填几就忽略几个字符

#### 4.偷窥一个字符

cin.peek();//偷窥就是查看缓冲区中的下一个字符，但是不会取走

#### 5.放回字符

char ch =cin.get();

cin.putback(ch);//放回字符，原来样放回

#### 6.代码示例

[src\cpp\标准输入cin\main.cpp](src/cpp/标准输入cin/main.cpp)

### 标准输出cout

随便记一下，代码示例：

[src\cpp\标准输出cout\08 标准输出流.cpp](src/cpp/标准输出cout/08%20标准输出流.cpp)

### 文件IO

[src\cpp\文件IO\main.cpp](src/cpp/文件IO/main.cpp)

## 面向对象

### 封装

### 继承

#### 1.继承的权限

a. pulic，protected，private继承后权限不变，如果权限不够自动降级,注意父类私有属性子类无法访问

b. 详细：

公共继承

父类中公共权限成员，到子类中变为公共权限

父类中保护权限成员，到子类中变为保护权限

父类中私有权限成员，到子类中不可访问

保护继承

父类中公共权限成员，到子类中变为保护权限

父类中保护权限成员，到子类中变为保护权限

父类中私有权限成员，到子类中不可访问

私有继承

父类中公共权限成员，到子类中变为私有权限

父类中保护权限成员，到子类中变为私有权限

父类中私有权限成员，到子类中不可访问

#### 2.查看对象模型

a. 找到vs开发人员命令提示符

b. 切换到文件所在目录，输入下面命令

c. cl /d1 reportSingleClassLayout类名 文件名

#### 3.不会被继承的函数

默认构造 拷贝构造 析构 operator=

#### 4.函数重定义

子类中只要有和父类相同名称的函数，父类中的所有该名称函数将被隐藏，如需访问，可以同过加作用域访问

#### 5.创建子类对象时需要创建父类对象

a. 一般来说父类对象必须要有默认构造

b. 如果父类对象没有默认构造，子类对象的构造，需要初始化列表法给父类构造传参

### 多态

多态就是父类 指针 或 引用 调用子类对象

#### 1.动态多态的满足条件

继承关系，父类中写了虚函数，子类重写了父类中的虚函数

#### 2.多态的原理

vfptr虚函数指针， vftable虚函数表，有兴趣在深入了解

#### 3.虚函数

多态使用： 父类的指针或者引用 指向子类的对象

#### 4.纯虚函数

a. 关键字 virtual

b. 纯虚函数可以不写实现，virtual void fun()=0;

c. 类中如果有纯虚函数，这个类就是抽象类，不可以创建对象，

d. 子类继承抽象类，如果想要创建对象就必须重写父类中的纯虚函数

#### 5.虚析构和纯虚析构

a. 纯虚析构需要在类外自己写实现

b. 存在的意义，释放父类时可以自己找到子类的析构调用，避免内存泄漏

c. 可以自己释放真实对象，不用虚析构

d. 使用虚析构后delete运作模式会改变，别乱delete会出错

#### 5.函数重写

1. 子类重写父类中的虚函数就是重写

2. 重写和重载一样需要返回值、函数名、形参列表一致，才是重写

### 构造函数

1. 创建对象需要调用构造方法

2. 构造函数的语法



3. 拷贝构造，初始化对象时调用，默认是值拷贝，所以如果有堆区数据就是拷贝地址（浅拷贝）

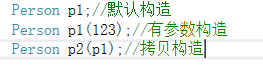
4. 默认构造的提供规则

a.默认会有无参构造和拷贝构造

b.只要提供了任何构造方法，就不会再提供无参构造

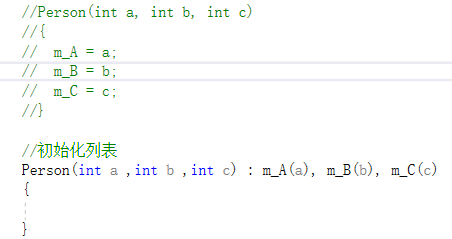
c.只要不自己提供拷贝构造，都会有默认的拷贝构造

5. 构造的调用





6. 初始化列表，其中参数有对象也可以使用



### new

1. 创建存放在堆区的对象，用完需要delete释放；

2. 如果利用new在堆区开辟 自定义类型数据的数组，必须有默认构造函数

a.Person \* personArray = new Person[10];

b.释放堆区数组 delete [] personArray;

### struct和class

1. 在c++下struct和class是一样的，只有默认权限不同

2. struct默认权限是public class是private

### 类的静态成员

1. 独立于对象存在，所以只有一份所有对象共用，也可以不使用对象单独使用

2. 独立于对象所以，静态属性函数外部赋值

3. 独立于对象所以，静态方法无法使用非对象静态成员

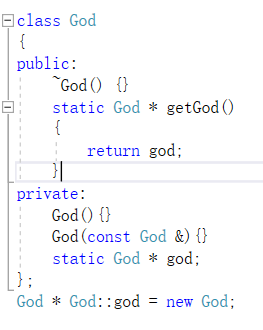
### 单例模式

1. 类只能创建一个对象就叫单例模式

2. 私有化所有构造

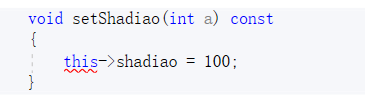
3. 创建一个私有的静态成员存储唯一对象的指针，该成员在类的外部赋值

4. 可以有一个公共方法返回这个唯一对象指针



### 类的常函数和常对象

1. 常函数表示this的值无法修改，(声明该函数不会改变成员属性的值)



2. 常对象成员属性(this)的值无法修改，常对象只能调用常函数，这是为了防止常对象的成员属性被修改



3. 可以有特殊成员，用 mutable 关键字声明，声明后该值可以在常函数中修改



### 友元

1. 友元就是允许对方访问该类的私有成员

2. 全局函数作为友元



3. 成员函数为友元



4. 类作为友元



### 运算符重载

1. 主要是让对象能使用运算符，方便操作，减少代码量

2. operator关键字

3. 代码示例

[src\cpp\运算符重载\test.cpp](src/cpp/运算符重载/test.cpp)

[src\cpp\运算符重载\MyString.h](src/cpp/运算符重载/MyString.h)

[src\cpp\运算符重载\MyString.cpp](src/cpp/运算符重载/MyString.cpp)

## 模板

模板也叫泛形编程 关键字 template<>

示例1函数模板：



示例2类模板：



代码示例1模板应用：

[src\cpp\模板\main.cpp](src/cpp/模板/main.cpp)

[src\cpp\模板\Person.hpp](src/cpp/模板/Person.hpp)

代码示例2自定义数组案例：

[src\cpp\模板\自定义数组案例\ARR.hpp](src/cpp/模板/自定义数组案例/ARR.hpp)

[src\cpp\模板\自定义数组案例\mian.cpp](src/cpp/模板/自定义数组案例/mian.cpp)

笔记示例：

[src\cpp\模板\Day18笔记.docx](src/cpp/模板/Day18笔记.docx)

## string类

### 1. 构造函数：

string();//创建一个空的字符串 例如: string str;

string(const string& str);//使用一个string对象初始化另一个string对象

string(const char\* s);//使用字符串s初始化

string(int n, char c);//使用n个字符c初始化

### 2. 基本赋值操作：

string& operator=(const char\* s);//char\*类型字符串 赋值给当前的字符串

string& operator=(const string &s);//把字符串s赋给当前的字符串

string& operator=(char c);//字符赋值给当前的字符串

string& assign(const char \*s);//把字符串s赋给当前的字符串

string& assign(const char \*s, int n);//把字符串s的前n个字符赋给当前的字符串

string& assign(const string &s);//把字符串s赋给当前字符串

string& assign(int n, char c);//用n个字符c赋给当前字符串

string& assign(const string &s, int start, int n);//将s从start开始n个字符赋值给字符串

### 3. 基本字符操作：

char& operator[](int n);//通过[]方式取字符

char& at(int n);//通过at方法获取字符

### 4. string拼接操作：

string& operator+=(const string& str);//重载+=操作符

string& operator+=(const char\* str);//重载+=操作符

string& operator+=(const char c);//重载+=操作符

string& append(const char \*s);//把字符串s连接到当前字符串结尾

string& append(const char \*s, int n);//把字符串s的前n个字符连接到当前字符串结尾

string& append(const string &s);//同operator+=()

string& append(const string &s, int pos, int n);//把字符串s中从pos开始的n个字符连接到当前字符串结尾

string& append(int n, char c);//在当前字符串结尾添加n个字符c

### 5. string比较操作：

compare函数在>时返回 1，<时返回 -1，==时返回 0。

比较区分大小写，比较时参考字典顺序，排越前面的越小。

大写的A比小写的a小。

int compare(const string &s) const;//与字符串s比较

int compare(const char \*s) const;//与字符串s比较

### 6. string查找和替换：没找到返回-1

int find(const string& str, int pos = 0) const; //查找str第一次出现位置,从pos开始查找

int find(const char\* s, int pos = 0) const; //查找s第一次出现位置,从pos开始查找

int find(const char\* s, int pos, int n) const; //从pos位置查找s的前n个字符第一次位置

int find(const char c, int pos = 0) const; //查找字符c第一次出现位置

int rfind(const string& str, int pos = npos) const;//查找str最后一次位置,从pos开始查找

int rfind(const char\* s, int pos = npos) const;//查找s最后一次出现位置,从pos开始查找

int rfind(const char\* s, int pos, int n) const;//从pos查找s的前n个字符最后一次位置

int rfind(const char c, int pos = 0) const; //查找字符c最后一次出现位置

string& replace(int pos, int n, const string& str); //替换从pos开始n个字符为字符串str

string& replace(int pos, int n, const char\* s); //替换从pos开始的n个字符为字符串s

### 7. string子串：

string substr(int pos = 0, int n = npos) const;//返回由pos开始的n个字符组成的字符串

### 8.string插入和删除操作：

string& insert(int pos, const char\* s); //插入字符串

string& insert(int pos, const string& str); //插入字符串

string& insert(int pos, int n, char c);//在指定位置插入n个字符c

string& erase(int pos, int n = npos);//删除从Pos开始的n个字符

### 9. string和c-style字符串转换：

string 转 char\*

string str = "itcast";

const char\* cstr = str.c\_str();

char\* 转 string

char\* s = "itcast";

string str(s);

10. 案例：

[src\cpp\string类\main.cpp](src/cpp/string类/main.cpp)

## STL标准模板库

### 1. 迭代器

//迭代器的定义

//容器::iterator 迭代器名 = 返回迭代器的方法;

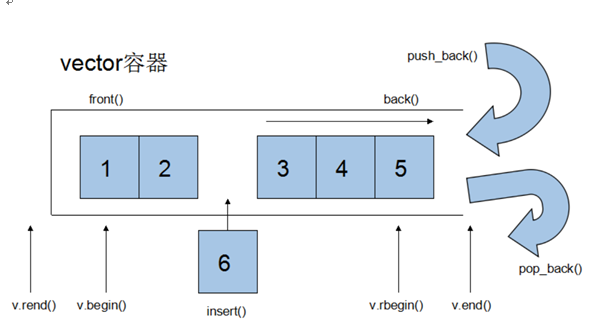
list<Person>::iterator it= p.begin();

|  |  |  |
| --- | --- | --- |
| 输入迭代器 | 提供对数据的只读访问 | 只读，支持++、==、！= |
| 输出迭代器 | 提供对数据的只写访问 | 只写，支持++ |
| 前向迭代器 | 提供读写操作，并能向前推进迭代器 | 读写，支持++、==、！= |
| 双向迭代器 | 提供读写操作，并能向前和向后操作 | 读写，支持++、--， |
| 随机访问迭代器 | 提供读写操作，并能以跳跃的方式访问容器的任意数据，是功能最强的迭代器 | 读写，支持++、--、[n]、-n、<、<=、>、>= |

### 2. 容器

#### 2. vector容器

单端动态数组，连续空间，有未雨绸缪机制。可预留空间，提高效率。头部操作效率低。使用随机迭代器



拷贝构造:

vector<T> v; //采用模板实现类实现，默认构造函数

vector(v.begin(), v.end());//将v[begin(), end())区间中的元素拷贝给本身。

vector(n, elem);//构造函数将n个elem拷贝给本身。

vector(const vector &vec);//拷贝构造函数。

赋值操作

assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。

assign(n, elem);//将n个elem拷贝赋值给本身。

vector& operator=(const vector &vec);//重载等号操作符

swap(vec);// 将vec与本身的元素互换。

vector的大小操作：

size();//返回容器中元素的个数

empty();//判断容器是否为空，判断size

resize(int num);//重新指定容器的长度为num，若容器变长，则以默认值填充新位置。如果容器变短，则末尾超出容器长度的元素被删除。

resize(int num, elem);//重新指定容器的长度为num，若容器变长，则以elem值填充新位置。如果容器变短，则末尾超出容器长>度的元素被删除。

capacity();//容器的容量

reserve(int len);//给容器预留容量，避免重复开辟、释放，提高效率

vector数据存取操作：

at(int idx); //返回索引idx所指的数据，如果idx越界，抛出out\_of\_range异常。

operator[];//返回索引idx所指的数据，越界时，运行直接报错

front();//返回容器中第一个数据元素

back();//返回容器中最后一个数据元素

vector插入和删除操作：

insert(const\_iterator pos, int count, ele);//迭代器指向位置pos插入count个元素ele.

push\_back(ele); //尾部插入元素ele

pop\_back();//删除最后一个元素

erase(const\_iterator start, const\_iterator end);//删除迭代器从start到end之间的元素

erase(const\_iterator pos);//删除迭代器指向的元素

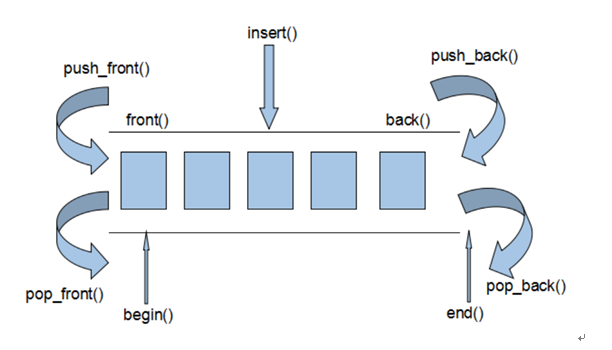
clear();//删除容器中所有元素，删除的是元素个数，空间容量不变

案例：

[src\cpp\STL标准模板库\vector单端动态数组\main.cpp](src/cpp/STL标准模板库/vector单端动态数组/main.cpp)

#### 3.deque容器

双端队列，非连续空间，使用随机迭代器



构造方法：

deque<T> deqT;//默认构造形式

deque(beg, end);//构造函数将[beg, end)区间中的元素拷贝给本身。

deque(n, elem);//构造函数将n个elem拷贝给本身。

deque(const deque &deq);//拷贝构造函数。

deque赋值操作：

assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。

assign(n, elem);//将n个elem拷贝赋值给本身。

deque& operator=(const deque &deq); //重载等号操作符

swap(deq);// 将deq与本身的元素互换

deque大小操作：

deque.size();//返回容器中元素的个数

deque.empty();//判断容器是否为空

deque.resize(num);//重新指定容器的长度为num,若容器变长，则以默认值填充新位置。如果容器变短，则末尾超出容器长度的元素被删除。

deque.resize(num, elem); //重新指定容器的长度为num,若容器变长，则以elem值填充新位置,如果容器变短，则末尾超出容器长度的元素被删除。

deque双端插入和删除操作：

push\_back(elem);//在容器尾部添加一个数据

push\_front(elem);//在容器头部插入一个数据

pop\_back();//删除容器最后一个数据

pop\_front();//删除容器第一个数据

deque数据存取：

at(idx);//返回索引idx所指的数据，如果idx越界，抛出out\_of\_range。

operator[];//返回索引idx所指的数据，如果idx越界，不抛出异常，直接出错。

front();//返回第一个数据。

back();//返回最后一个数据

deque插入操作：

insert(pos, elem);//在pos位置插入一个elem元素的拷贝，返回新数据的位置。

insert(pos, n, elem);//在pos位置插入n个elem数据，无返回值。

insert(pos, beg, end);//在pos位置插入[beg,end)区间的数据，无返回值。

deque删除操作：

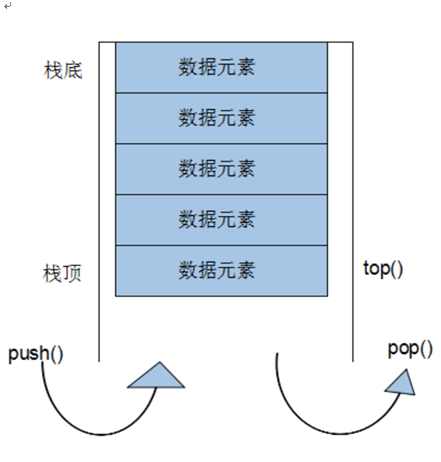
clear();//移除容器的所有数据

erase(beg, end);//删除[beg,end)区间的数据，返回下一个数据的位置。

erase(pos);//删除pos位置的数据，返回下一个数据的位置。

#### 4. stack容器

栈容器，没有迭代器



stack构造函数：

stack<T> stkT;//stack采用模板类实现， stack对象的默认构造形式：

stack(const stack &stk);//拷贝构造函数

stack赋值操作：

stack& operator=(const stack &stk);//重载等号操作符

stack数据存取操作：

push(elem);//向栈顶添加元素

pop();//从栈顶移除第一个元素

top();//返回栈顶元素

stack大小操作：

empty();//判断堆栈是否为空

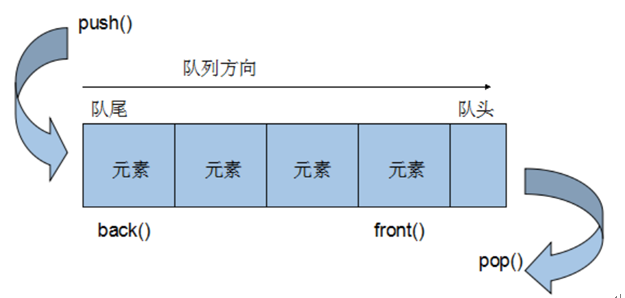
size();//返回堆栈的大小

代码示例：

[src\cpp\STL标准模板库\stack栈容器\main.cpp](src/cpp/STL标准模板库/stack栈容器/main.cpp)

#### 5. queue容器

队列容器，没有迭代器



queue构造函数：

queue<T> queT;//queue采用模板类实现，queue对象的默认构造形式：

queue(const queue &que);//拷贝构造函数

queue存取、插入和删除操作：

push(elem);//往队尾添加元素

pop();//从队头移除第一个元素

back();//返回最后一个元素

front();//返回第一个元素

queue赋值操作：

queue& operator=(const queue &que);//重载等号操作符

queue大小操作：

empty();//判断队列是否为空

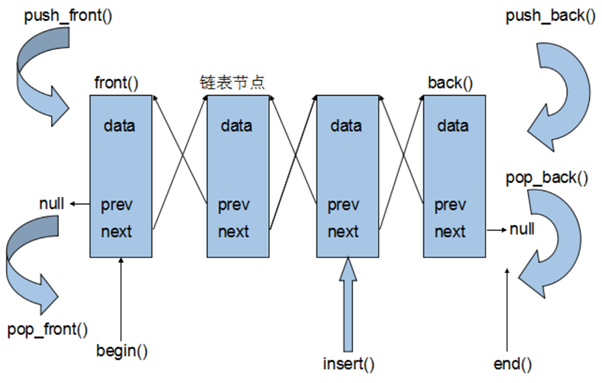
size();//返回队列的大小

代码示例：

[src\cpp\STL标准模板库\queue队列\main.cpp](src/cpp/STL标准模板库/queue队列/main.cpp)

#### 6. list容器

双向链表，使用双向迭代器



list构造函数：

list<T> lstT;//list采用采用模板类实现,对象的默认构造形式：

list(beg, end);//构造函数将[beg, end)区间中的元素拷贝给本身。

list(n, elem);//构造函数将n个elem拷贝给本身。

list(const list &lst);//拷贝构造函数。

list数据元素插入和删除操作：

push\_back(elem);//在容器尾部加入一个元素

pop\_back();//删除容器中最后一个元素

push\_front(elem);//在容器开头插入一个元素

pop\_front();//从容器开头移除第一个元素

insert(pos, elem);//在pos位置插elem元素的拷贝，返回新数据的位置。

insert(pos, n, elem);//在pos位置插入n个elem数据，无返回值。

insert(pos, beg, end);//在pos位置插入[beg,end)区间的数据，无返回值。

clear();//移除容器的所有数据

erase(beg, end);//删除[beg,end)区间的数据，返回下一个数据的位置。

erase(pos);//删除pos位置的数据，返回下一个数据的位置。

remove(elem);//删除容器中所有与elem值匹配的元素。

list大小操作：

size();//返回容器中元素的个数

empty();//判断容器是否为空

resize(num);//重新指定容器的长度为num，

若容器变长，则以默认值填充新位置。

如果容器变短，则末尾超出容器长度的元素被删除。

resize(num, elem);//重新指定容器的长度为num，

若容器变长，则以elem值填充新位置。

如果容器变短，则末尾超出容器长度的元素被删除。

list赋值操作：

assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。

assign(n, elem);//将n个elem拷贝赋值给本身。

list& operator=(const list &lst);//重载等号操作符

swap(lst);//将lst与本身的元素互换。

list数据的存取：

front();//返回第一个元素。

back();//返回最后一个元素。

list反转排序：

reverse();//反转链表，比如lst包含1,3,5元素，运行此方法后，lst就包含5,3,1元素。

sort(); //list排序

代码示例：

[src\cpp\STL标准模板库\list双向链表\main.cpp](src/cpp/STL标准模板库/list双向链表/main.cpp)

#### 7. set/multiset容器

set容器，插入数据会自动排序的容器，不允许元素重复

multiset和set一样但是允许元素重复

使用什么迭代器暂时不清楚待补充

构造函数：

set<T> st;//set默认构造函数：

mulitset<T> mst; //multiset默认构造函数:

set(const set &st);//拷贝构造函数

set赋值操作：

set& operator=(const set &st);//重载等号操作符

swap(st);//交换两个集合容器

set大小操作：

size();//返回容器中元素的数目

empty();//判断容器是否为空

set插入和删除操作：

insert(elem);//在容器中插入元素。

clear();//清除所有元素

erase(pos);//删除pos迭代器所指的元素，返回下一个元素的迭代器。

erase(beg, end);//删除区间[beg,end)的所有元素 ，返回下一个元素的迭代器。

erase(elem);//删除容器中值为elem的元素。

set查找操作：

find(key);//查找键key是否存在,若存在，返回该键的元素的迭代器；若不存在，返回set.end();

count(key);//查找键key的元素个数

lower\_bound(keyElem);//返回第一个key>=keyElem元素的迭代器。

upper\_bound(keyElem);//返回第一个key>keyElem元素的迭代器。

equal\_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器。

代码示例:

[src\cpp\STL标准模板库\set容器\main.cpp](src/cpp/STL标准模板库/set容器/main.cpp)

#### 8. map/multimap容器

map容器可以有一个键，一个值

键和set容器是一样的

map构造函数：

map<T1, T2> mapTT;//map默认构造函数:

map(const map &mp);//拷贝构造函数

map赋值操作：

map& operator=(const map &mp);//重载等号操作符

swap(mp);//交换两个集合容器

map大小操作：

size();//返回容器中元素的数目

empty();//判断容器是否为空

map插入数据元素操作：

map.insert(...); //往容器插入元素，返回pair<iterator,bool>

map<int, string> mapStu;

// 第一种 通过pair的方式插入对象

mapStu.insert(pair<int, string>(3, "小张"));

// 第二种 通过pair的方式插入对象

mapStu.inset(make\_pair(-1, "校长"));

// 第三种 通过value\_type的方式插入对象

mapStu.insert(map<int, string>::value\_type(1, "小李"));

// 第四种 通过数组的方式插入值

mapStu[3] = "小刘";

mapStu[5] = "小王";

map删除操作：

clear();//删除所有元素

erase(pos);//删除pos迭代器所指的元素，返回下一个元素的迭代器。

erase(beg, end);//删除区间[beg,end)的所有元素 ，返回下一个元素的迭代器。

erase(keyElem);//删除容器中key为keyElem的对组。

map查找操作：

find(key);//查找键key是否存在,若存在，返回该键的元素的迭代器；/若不存在，返回map.end();

count(keyElem);//返回容器中key为keyElem的对组个数。对map来说，要么是0，要么是1。对multimap来说，值可能大于1。

lower\_bound(keyElem);//返回第一个key>=keyElem元素的迭代器。

upper\_bound(keyElem);//返回第一个key>keyElem元素的迭代器。

equal\_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器。

代码示例：

[src\cpp\STL标准模板库\map容器\main.cpp](src/cpp/STL标准模板库/map容器/main.cpp)

#### 9. pair<>对组

1. 示例

pair<set<int>::iterator, set<int>::iterator> pit = s1.equal\_range(5);

cout << "下一个元素的迭代器\*(pid.second)="<<\*(pit.second) << endl;//下一个元素的迭代器

cout << "当前元素的迭代器\*(pid.first)="<<\*(pit.first) << endl;;//当前元素的迭代器

2. 示例

pair<map<int, Person>::iterator, map<int, Person>::iterator> dz = p.equal\_range(2);

cout << "ID=\t" << dz.first->first << endl;

cout << "age=\t" << dz.first->second.age << endl;

cout << "name=\t"<<dz.first->second.name << endl;

cout<<endl << endl;

cout << "ID=\t" << dz.second->first << endl;

cout << "age=\t" << dz.second->second.age << endl;

cout << "name=\t" << dz.second->second.name << endl;

### 3. 算法

内建函数

template<class T> bool less<T>//小于 降序排列

template<class T> bool greater<T>//大于 升序排列

#### 1. 算法头文件

1. 算法主要是由头文件<algorithm> <functional> <numeric>组成。

2. <algorithm>是所有STL头文件中最大的一个,其中常用的功能涉及到比较，交换，查找,遍历，复制，修改，反转，排序，合并等...

3. <numeric>体积很小，只包括在几个序列容器上进行的简单运算的模板函数.

4. <functional> 定义了一些模板类,用以声明函数对象。

#### 2. 遍历算法

##### 1. 遍历算法 遍历容器元素

@param beg 开始迭代器

@param end 结束迭代器

@param \_callback 函数回调或者函数对象

@return 函数对象

for\_each(iterator beg, iterator end, \_callback);

##### 2. transform算法 将指定容器区间元素搬运到另一容器中

注意 :目标不知道为什么要用resize指定长度，不然报错

@param beg1 源容器开始迭代器

@param end1 源容器结束迭代器

@param beg2 目标容器开始迭代器

@param \_cakkback 回调函数或者函数对象

@return 返回目标容器迭代器

for\_each(iterator beg1, iterator end1, iterator beg2, \_callbakc)

##### 3. 代码：

[src\cpp\STL标准模板库\遍历算法\main.cpp](src/cpp/STL标准模板库/遍历算法/main.cpp)

#### 3.查找算法

##### find算法 查找元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value 查找的元素

@return 返回第一个找到元素的迭代器

find(iterator beg, iterator end, value)

##### find\_if算法 条件查找

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param callback 回调函数或者谓词(返回bool类型的函数对象)

@return 返回第一个找的元素的迭代器

find\_if(iterator beg, iterator end, \_callback);

##### adjacent\_find算法 查找相邻重复元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@return 返回相邻元素的第一个位置的迭代器

adjacent\_find(iterator beg, iterator end);

##### binary\_search算法 二分查找法

注意: 在无序序列中不可用

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value 查找的元素

@return bool 查找返回true 否则false

bool binary\_search(iterator beg, iterator end, value);

##### count算法 统计元素出现次数

@param beg 容器开始迭代器

@param end 容器结束迭代器

@return int返回元素个数

##### count\_if算法 统计元素出现次数

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param callback 回调函数或者谓词(返回bool类型的函数对象)

@return int返回元素个数

count\_if(iterator beg, iterator end, \_callback);

##### 代码示例

[src\cpp\STL标准模板库\查找算法\main.cpp](src/cpp/STL标准模板库/查找算法/main.cpp)

#### 4. 排序算法

##### merge算法 容器元素合并，并存储到另一容器中

注意 : 两个容器必须是有序的

@param beg1 容器1开始迭代器

@param end1 容器1结束迭代器

@param beg2 容器2开始迭代器

@param end2 容器2结束迭代器

@param dest 目标容器开始迭代器

##### sort算法 容器元素排序

@param beg 容器1开始迭代器

@param end 容器1结束迭代器

@param \_callback 回调函数或者谓词(返回bool类型的函数对象)

sort(iterator beg, iterator end, \_callback)

##### random\_shuffle算法 对指定范围内的元素随机调整次序

@param beg 容器开始迭代器

@param end 容器结束迭代器

random\_shuffle(iterator beg, iterator end);

##### reverse算法 反转指定范围的元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

reverse(iterator beg, iterator end)

##### 代码示例：

[src\cpp\STL标准模板库\排序算法\main.cpp](src/cpp/STL标准模板库/排序算法/main.cpp)

#### 5. 拷贝和替换算法

##### copy算法 将容器内指定范围的元素拷贝到另一容器中

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param dest 目标起始迭代器

copy(iterator beg, iterator end, iterator dest)

##### replace算法 将容器内指定范围的旧元素修改为新元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param oldvalue 旧元素

@param newvalue 新元素

replace(iterator beg, iterator end, oldvalue, newvalue)

##### replace\_if算法 将容器内指定范围满足条件的元素替换为新元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param callback函数回调或者谓词(返回Bool类型的函数对象)

@param oldvalue 新元素

replace\_if(iterator beg, iterator end, \_callback, newvalue);

##### swap算法 互换两个容器的元素

@param c1容器1

@param c2容器2

swap(container c1, container c2);

##### 代码示例：

[src\cpp\STL标准模板库\拷贝和替换算法\main.cpp](src/cpp/STL标准模板库/拷贝和替换算法/main.cpp)

#### 6. 常用算术算法

##### accumulate算法 计算容器元素累计总和

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value累加值

accumulate(iterator beg, iterator end, value)

##### fill算法 向容器中添加元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value t填充元素

fill(iterator beg, iterator end, value)

##### 代码示例：

[src\cpp\STL标准模板库\常用算数生成算法\main.cpp](src/cpp/STL标准模板库/常用算数生成算法/main.cpp)

#### 7. 常用的集合算法

##### set\_intersection算法 求两个set集合的交集

注意:两个集合必须是有序序列

@param beg1 容器1开始迭代器

@param end1 容器1结束迭代器

@param beg2 容器2开始迭代器

@param end2 容器2结束迭代器

@param dest 目标容器开始迭代器

@return 目标容器的最后一个元素的迭代器地址

##### set\_union算法 求两个set集合的并集

注意:两个集合必须是有序序列

@param beg1 容器1开始迭代器

@param end1 容器1结束迭代器

@param beg2 容器2开始迭代器

@param end2 容器2结束迭代器

@param dest 目标容器开始迭代器

@return 目标容器的最后一个元素的迭代器地址

##### set\_union算法 求两个set集合的并集

注意:两个集合必须是有序序列

@param beg1 容器1开始迭代器

@param end1 容器1结束迭代器

@param beg2 容器2开始迭代器

@param end2 容器2结束迭代器

@param dest 目标容器开始迭代器

@return 目标容器的最后一个元素的迭代器地址

##### 代码示例：

[src\cpp\STL标准模板库\常用集合算法\main.cpp](src/cpp/STL标准模板库/常用集合算法/main.cpp)