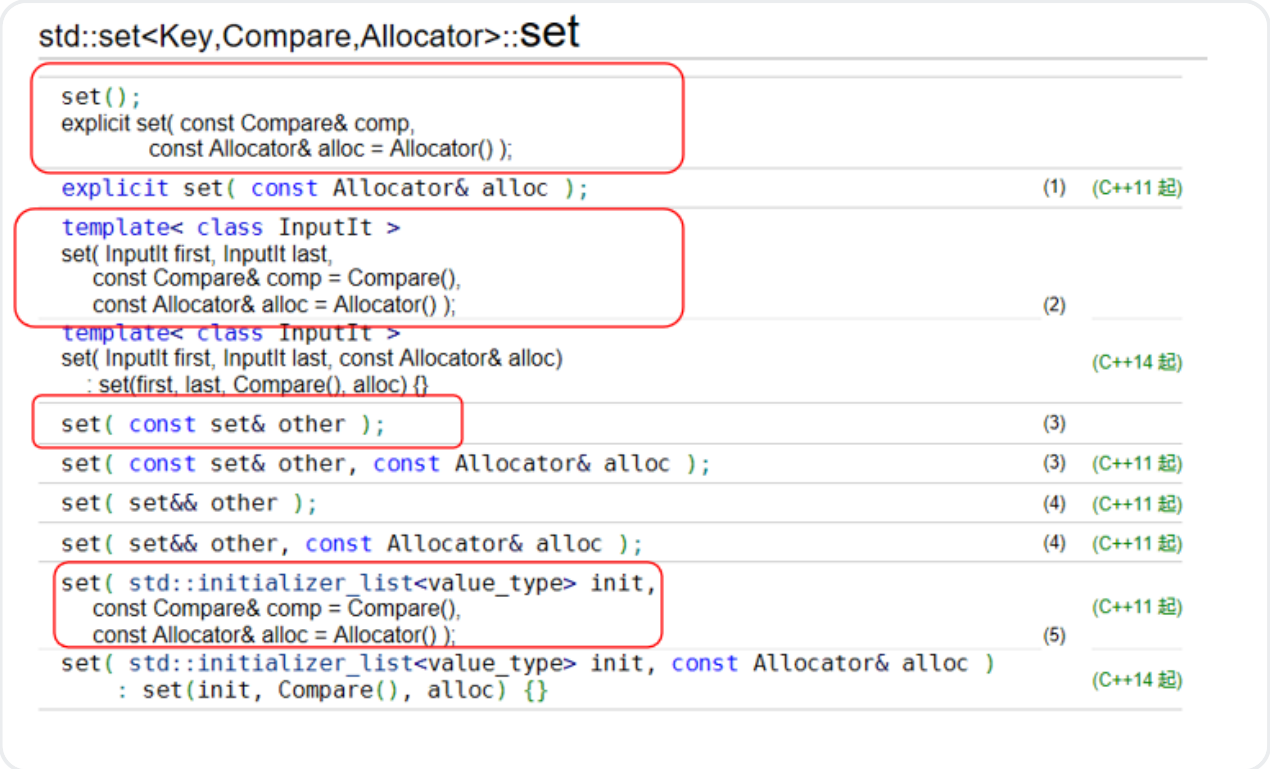
第六章 关联式容器

学到这里，我们可以提前学习一些STL的内容了，以帮助我们完成作业。本章我们介绍两个容器set、map，它们属于STL中的关联式容器。

set

set的构造

包含在头文件< set >，打开C++参考文档，主要关注这样的几个构造函数



1. 无参构造
2. 迭代器方式进行构造，传入一个first迭代器，传入一个last迭代器
3. 拷贝构造
4. 标准初始化列表（大括号的形式）

```
set<int> number;
set<int> number2 = {1,3,9,8,9};
```

set的创建与vector很类似，尝试着调用以上四种构造方式进行创建，并实现对set中元素的遍历

```cpp
set<int> nums;
set<int> nums2{2,3,2,1,9,3,7,0,1};
set<int> nums3 = nums2;
set<int> nums4(nums2.begin(),nums2.end());

//set不支持下标访问
//nums2[0];

//迭代器方式进行遍历
set<int>::iterator it = nums2.begin();
/* auto it = nums2.begin(); */
for(;it != nums2.end();++it){
    cout << *it << " ";
}
cout << endl;

for(auto & nu : nums2){
    cout << nu << " ";
}
cout << endl;
```

set的特征：

（1）set中存放的元素是唯一的，不能重复；

（2）默认情况下，会按照元素进行升序排列；

set的查找操作

std::set<Key,Compare,Allocator>::count

```
size_type count( const Key& key ) const;    (1)
```

count：输入一个值，在set中查找，如果有就返回1，没有就返回0

std::set<Key,Compare,Allocator>::find

```
iterator find( const Key& key );                (1)
const_iterator find( const Key& key ) const;    (2)
```

find:输入一个值，在set中进行查找，如果找到，就返回这个元素相应的迭代器。若找不到，则返回end()获取的迭代器。

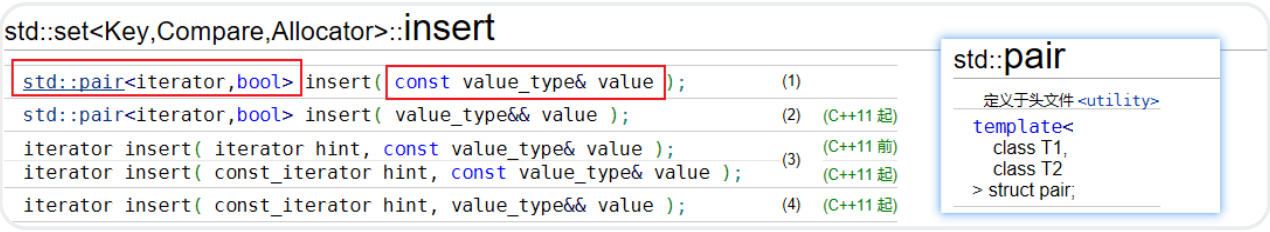
——请实践一下这两个函数的使用

```
//set的查找操作
cout << nums2.count(9) << endl;
cout << nums2.count(10) << endl;

auto it2 = nums2.find(1);
cout << *it2 << endl;

it2 = nums2.find(8);
if(it2 == nums2.end()){
    cout << "hello" << endl;
}
```

set的插入操作



可以看到insert函数的第一种形式中，参数是一个key，返回的值是一个pair类型（包含一个迭代器和一个bool值）

我们先来看看pair是什么

—— pair定义在头文件< utility >中，类似于结构体，可以存储两种不同类型的变量。

当然，C++中结构体已经演变为了类，所以可以认为一个特定的pair是一个类，包含两个对象成员(它们的类型在定义pair时给出)。

重点关注：pair的对象成员如何访问

```
#include <utility>
void test1(){
    pair<int,string> num = {1,"wangdao"};
    cout << num.first << ":" << num.second << endl;
}
```

对set进行插入

- 插入单个元素

insert函数的返回类型是pair类型，包含两个对象成员，第一个是对应set的迭代器，第二个是bool值

如果插入成功，则返回“**插入元素对应迭代器**和**true**”；

如果插入失败，则返回**“阻止插入的元素（原本就有的这个元素）对应迭代器**和**false**”.

```
pair<set<int>::iterator,bool> ret = number.insert(8);
   if(ret.second){
       cout << "该元素插入成功:"
           << *(ret.first) << endl;
   }else{
       cout << "该元素插入失败，表明该元素已存在" << endl;
   }
```

```
//set的插入操作
//插入一个元素
auto ret = nums2.insert(1);
if(ret.second){
    cout << "插入成功" << endl;
    cout << *(ret.first) << endl;
}else{
    cout << "插入失败，元素已存在" << endl;
    cout << *(ret.first) << endl;
}
```

- 插入多个元素

```
int arr[5] = {18,41,35,2,99};
number.insert(arr,arr + 5); //思考，如果想要插入arr的全部元素，此处应该是 arr + 5 还是 arr + 4 ?
```

```cpp
    //插入多个元素
    int arr[5] = {18,41,35,2,99};
    //传入首迭代器和尾后迭代器
    nums2.insert(arr,arr + 5);

    for(auto & nu : nums2){
        cout << nu << " ";
    }
    cout << endl;

    //传入大括号的列表
    nums2.insert({100,1,111});
    for(auto & nu : nums2){
        cout << nu << " ";
    }
    cout << endl;
```

注意：

- **set容器不支持下标访问**，因为没有operator[ ] 重载函数
- **不能通过set的迭代器直接修改key值**，set的底层实现是红黑树，结构稳定，不允许直接修改。

```cpp
    auto it3 = nums2.begin();
    ++it3;
    ++it3;
    cout << *it3 << endl;
    *it3 = 300;

```

```
ray@ubuntu:~/HaiBao/54th/day11$ g++ set_map.cc
set_map.cc: In function ‘void test0()’:
set_map.cc:79:12: error: assignment of read-only location ‘it3.std::_Rb_tree_co
nst_iterator<int>::operator*()’
     *it3 = 300;
            ^~~
```

map

map的构造

map中存放的元素的类型是pair类型（键值对），构造map需要关注三种方式，也可以把它们结合到一起。如下：

```cpp
void test0(){
    map<int,string> number = {
        {1,"hello"},
        {2,"world"},
        {3,"wangdao"},
        pair<int,string>(4,"hubei"),
        pair<int,string>(5,"wangdao"),
        make_pair(9,"shenzhen"),
        make_pair(3,"beijing"),
        make_pair(6,"shanghai")
    };
}
```

使用迭代器方式遍历map，注意访问map的元素pair的内容时的写法

```cpp
    map<int,string>::iterator it = number.begin();
    while(it != number.end()){
        cout << (*it).first << " " << it->second << endl;
        ++it;
    }
    cout << endl;
```

```
1 hello
2 world
3 wangdao
4 hubei
5 wangdao
6 shanghai
9 shenzhen
ray@ubuntu:~
```

```cpp
 map<int,string> number = {
     {1,"hello"},
     {2,"world"},
     {3,"wangdao"},
     pair<int,string>(4,"hubei"),
     pair<int,string>(5,"wangdao"),
     make_pair(9,"shenzhen"),
     make_pair(3,"beijing"),
     make_pair(6,"shanghai")
 };
```

```cpp
//对map进行遍历
for(auto & nu : number){
    cout << nu.first << " " << nu.second  << endl;
}

cout << endl;
/* auto it = number.begin(); */
map<int,string>::iterator it = number.begin();
for(; it != number.end(); ++it){
    /* cout << (*it).first << " " << (*it).second << endl; */
    cout << it->first << " " << it->second << endl;
}
```

map的特征：

（1）元素唯一：创建map对象时，舍弃了一些元素，key值相同的元素被舍弃。key不同，即使value相同也能保留

（2）默认以key值为参考进行升序排列

map的查找操作

根据key值在map中进行查找

count函数的返回值：如果找到返回1，如果没找到返回0（size_t类型）

find函数的返回值：如果找到返回相应元素的迭代器，如果没找到返回end( )的结果

——请实践一下这两个函数的使用

```
void checkFind(const map<int,string> & rhs,int key){
    auto it = rhs.find(key);
    if(it == rhs.end()){
        cout << "该元素不在map中" << endl;
    }else{
        cout << "该元素在map中" << endl;
        cout << it->first << " " << it->second << endl;
    }
}
```

```
//map中进行查找
size_t cnt1 = number.count(1);
size_t cnt2 = number.count(10);
cout << cnt1 << endl;
cout << cnt2 << endl;

/* map<int,string>::iterator it2 = number.find(1); */
/* auto it3 = number.find(10); */
/* cout << "it2:" << it2->first << " " << it2->second << endl; */
/* cout << "it3:" << it3->first << " " << it3->second << endl; */
checkFind(number,10);
cout << endl;
checkFind(number,5);
```

map的插入操作

`std::pair<iterator,bool> insert( const value_type& value );`	(1)	
`template< class P >` `std::pair<iterator,bool> insert( P&& value );`	(2)	(C++11 起)
`std::pair<iterator,bool> insert( value_type&& value );`	(3)	(C++17 起)
`iterator insert( iterator hint, const value_type& value );`	(4)	(C++11 前)
`iterator insert( const_iterator hint, const value_type& value );`		(C++11 起)
`template< class P >` `iterator insert( const_iterator hint, P&& value );`	(5)	(C++11 起)
`iterator insert( const_iterator hint, value_type&& value );`	(6)	(C++17 起)
`template< class InputIt >` `void insert( InputIt first, InputIt last );`	(7)	
`void insert( std::initializer_list<value_type> ilist );`	(8)	(C++11 起)
`insert_return_type insert(node_type&& nh);`	(9)	(C++17 起)
`iterator insert(const_iterator hint, node_type&& nh);`	(10)	(C++17 起)

插入单个元素，此时insert函数的返回值是一个pair（第一个对象成员是map元素相应的迭代器，第二个对象成员是bool值）

```
    pair<map<int,string>::iterator,bool> ret = number.insert(pair<int,string>(7,"nanjing"));

    if(ret.second){
        cout << "该元素插入成功" << endl;
        //ret.first取出来的是指向map元素（pair<int,string>）的迭代器
        //再用箭头运算符访问到的是int和string的内容
        cout << ret.first->first << " : " << ret.first->second << endl;
    }else{
        cout << "该元素插入失败" << endl;
    }
    cout << endl;
```

```
cout << "############" << endl;
/* auto ret = number.insert(pair<int,string>(7,"nanjing")); */
pair<map<int,string>::iterator,bool> ret =
    number.insert(pair<int,string>(7,"nanjing"));
if(ret.second){
    cout << "该元素插入成功" << endl;
    cout << ret.first->first << " " << ret.first->second << endl;
}else{
    cout << "插入失败" << endl;
}

cout << endl;
number.insert({9,"hangzhou"});
for(auto & nu : number){
    cout << nu.first << " " << nu.second  << endl;
}
```

插入一组元素

```
1      //再创建一个map
2      map<int,string> number2 = {{1,"beijing"},{18,"shanghai"}};
3
4      //迭代器方式
5      number2.insert(number.begin(),number.end());
6
7      //列表方式
8      cout << endl;
9      number2.insert({{4,"guangzhou"},{22,"hello"}});
```

map的下标操作

map支持下标操作

1. map下标操作返回的是map中元素（pair）的value
2. 下标访问运算符中的值代表key，而不是传统意义上的下标
3. 如果进行下标操作时下标值传入一个不存在的key，那么会将这个key和空的value插入到map中
4. 下标访问可以进行写操作

```cpp
//下标中的是key值，并非传统的下标
cout << number[1] << endl;

cout << endl;
//支持写操作
number[3] = "cpp";
for(auto & nu : number){
    cout << nu.first << " " << nu.second  << endl;
}
//如果下标对应的key不存在，
//会插入一个元素进入map
//value按默认方式构造
cout << number[6] << endl;

cout << endl;
for(auto & nu : number){
    cout << nu.first << " " << nu.second  << endl;
}

number[7] = "hubei";
for(auto & nu : number){
    cout << nu.first << " " << nu.second  << endl;
}
```