面向对象程序语言C++
编写泛型函数

周雅倩
zhouyaqian@fudan.edu.cn
2016/4/1

什么是泛型函数

- 很多容器操作可以用于多种容器
  - vector, string和list都允许调用insert函数来插入元素，也允许调用erase函数来删除元素。
  - 这些操作对于所有支持它们的类型都有同样的接口。
  - 每个容器都提供了相应的迭代器类型。

分析字符串

- 通过循环实现：

```
for(std::vector<string>::const_iterator it=bottom.begin();
      it!=bottom.end(); it++)
      ret.push_back(*it);
```

- 通过vector的insert函数实现：

```
ret.insert(ret.end(), bottom.begin(), bottom.end());
```

- 更为通用的解决方案：把复制元素和在容器末尾插入元素的概念分离开来。

```
copy(bottom.begin(), bottom.end(),back_inserter(ret));
```

泛型算法　迭代器适配器

未知类型的中值

```
template <class T>
T median(std::vector<T> vec)
{
        typedef typename
std::vector<T>::size_type vec_sz;
        vec_sz size = vec.size();
        if(size == 0)
                throw
domain_error("median of an empty
vector");
        std::sort(vec.begin(),vec.end());
        vec_sz mid = size/2;
        return size %2 == 0 ?
(vec[mid]+vec[mid-1])/2 : vec[mid];
}
```

```
double median(vector<double> vec)
{
        typedef
std::vector<double>::size_type vec_sz;
        vec_sz size = vec.size();
        if(size == 0)
                throw
domain_error("median of an empty
vector");
        sort(vec.begin(),vec.end());
        vec_sz mid = size/2;
        return size %2 == 0 ?
(vec[mid]+vec[mid-1])/2 : vec[mid];

}
```


模板实例化

- 实例化动作经常不是在编译期间而是在链接期间发生的。
- 当前大多数系统环境都要求这个模板的定义（不仅是声明）必须是系统环境可以访问的。
  - 许多系统环境都要求模板的头文件直接或者通过#include指令而将源文件包含进去。

数据结构独立性

- C++标准库的一个主要贡献是，它确立了一种算法设计思想：
  - 算法能够用迭代器来作为算法与容器之间的“黏合剂”从而获得数据结构的独立性。
  - 算法所用到的迭代器都要求有某些操作，我们能以这些操作作为基础而分解算法。

5种迭代器

- 顺序只读访问
- 顺序只写访问
- 顺序读写访问
- 可逆访问
- 随机访问

顺序只读访问

```
template <class In, class X>
In find(In begin, In end, const X&x){
        while(begin!=end && *begin != x)
                ++begin;
        return begin;
}

template <class In, class X>
In find(In begin, In end, const X&x){
        while(begin==end || *begin == x)
        return begin;
                begin++;
        return find(begin,end,x);
}
```


顺序只写访问

```
template <class In, class Out>
Out copy(In begin, In end, Out dest){
        while(begin!=end)
                *dest++=*begin++;
        return dest;
}
```


顺序读写访问

```
template <class For, class X>
void replace(For begin, For end, const X&x, const X&y){
        while(begin!=end){
                if(*begin==x) *begin=y;
                begin++;
        }
}
```


可逆访问

```
template <class Bi>
void reverse(Bi begin, Bi end){
        while(begin!=end){
                --end;
                if(begin!=end)
                        swap(*begin++,*end);
        }
}
```


随机访问

```
template <class Ran, class X>
bool binary_search(Ran begin, Ran end, const X &x){
        while(begin < end){
                Ran mid=begin+(end-begin)/2;
                if(x<*mid)        end=mid;
                else if(x>*mid) begin=mid+1;
                else return true;
        }
        return false;
}
```