第十六章 模板

1. 模板

↓

针对同一个算法，基于不同的数据类型会产生不同的实现，对开发者和用户来说都造成了压力

↓

此时，通过模板进行泛式编程，即可忽略数据类型的差异，定义同一算法的函数族或类族

2. 类模板

语法：

```
template <class T> class 类名
{
    类的实现
};
```

T是抽象的数据类型

用于在模板中充当泛化参数类型

或

```
template <typename T>
class 类名
{
    类体
};
```

注：模板中可以包含多个抽象数据类型

template <class T1, class T2>

在类模板的使用时，各泛化出的实体应在定义时指定数据类型 → T为int

如：vector <int> int_array;

在类模板中，如果成员函数希望外定义时，应以如下形式

template <typename T> 前缀，表明为模板内

返回类型 类名<T>:: 函数名（参数列表）

{ 指定作用域

函数体；

}

```
template <class T>
class Stack {
  private:
    vector<T> elems;     // 元素

  public:
    void push(T const&);  // 入栈
    void pop();               // 出栈
    T top() const;            // 返回栈顶元素
    bool empty() const{       // 如果为空则返回真。
        return elems.empty();
    }
};
```

类模板

```
template <class T>
void Stack<T>::push (T const& elem)
{
    // 追加传入元素的副本
    elems.push_back(elem);
}
```

函数定义

注：有时候，为使开发者自定义的类或函数适用模板，需要根据模板的要求对自定义类或函数进行重载或修改

3. 函数模板

语法：template <class T> 返回类型 函数名（参数列）

T为抽象数据类型

或

template <typename T> 返回类型 函数名（参数列）

```
template <typename T>
inline T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}
int main ()
{

    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;
```

4、模板中的迭代器

实为指针，指向线性结构内元素

对线性结构进行元素遍历访问的数据结构，提供了一种访问线性结构的标准方式，可以使用++，--操作进行位置移动和元素遍历

如：

```
for(vector<int>::iterator it = vec.begin(); it != vec.end(); it++)
```

数据类型

指代为vector<int>内的迭代器

允许开发者在类的内部自行定义友元内部类 iterator
来实现迭代器功能

↓

此时要在线性结构类内定义 begin() 和 end() 函数，
并在 iterator 类内重载 ++，--等