### 记录时间2019.12.9-2019.12.15

**开发**： **BUG/问题**： **分析/解决**： **辨析研究**： **重点标记**： **阅读书籍**：

2019.12.9

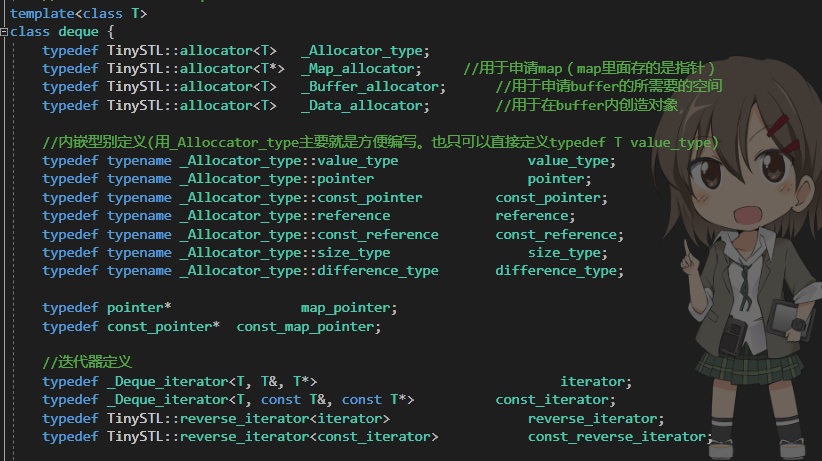
**开发1**：完成相关定义的工作

成员变量定义、成员函数的分类、初步设计代码调用层次

**问题1**：定义的型别有什么用？

一直都没有仔细思考过这个问题，之前都是照着书上抄的。现在来分析分析。

**分析1**：(换了字体，真香~等宽字体，比较适合编程)



问题的引发：在算法中，可能有地方会需要用到迭代器所指之物的类型来定义变量，这个时候我们去哪儿获得这个类型信息呢？？

解决办法：使用模板的参数推导机制

|  |
| --- |
| 模板相关知识点：  template<class T1, class T2> 模板形参  int getmax(T1 a, T2 b) 模板函数形参  ...  getmax<char, int>(‘a’,10);  模板实参 模板函数实参  上述代码中，getmax的尖括号中的类型可以省略，编译器会自动推导。因为模板形参和模板函数形参在位置上具有一一对象的关系。  模板自动推导的条件：   1. 模板形参和模板函数形参在位置上存在一一对应关系 2. 与模板函数返回值相关的模板参数无法进行推导 3. 需要推导的模板参数必须是位于模板参数列表的尾部   template<class T1, class T2>  T1 getmax(char a, T2 b){...}  int max = getmax(‘a’,10)  T2是可以推导的，满足3。 但是T1就不行了，不满足2，虽然满足1，在位置上和int对应了。 |

新问题：模板自动推导只能推导参数，函数返回类型可没法做到，这个时候想用迭代器所指的类型，作为函数的返回类型，应该怎么办？

解决办法：使用内嵌类型

新问题：只有类内部才可以使用typedef 定义类型别名，但是迭代器有时候是指针呀

解决办法：针对指针和const指针，进行特化

整套类型定义都要从iterator\_traits说起

template<class Iterator>

struct iterator\_traits{

typedef typename Iterator::iterator\_category iterator\_category;

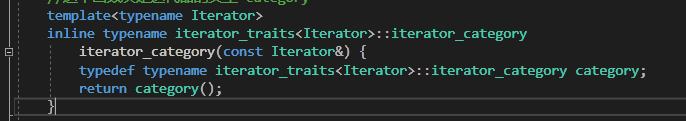
typedef typename Iterator::value\_type value\_type;

...

}

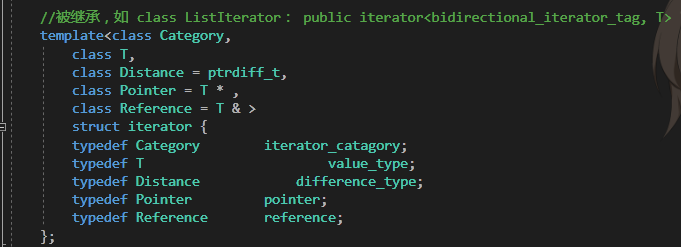
具体使用是

typedef typename iterator\_traits<模板参数>::value\_type value\_type; 这个模板参数，一般都是个迭代器



上面那一串其实转换一下就变成了 模板参数::value\_type。也就是说迭代器内部的定义了value\_type才行。

为了满足条件，我们创建了一个迭代器模板，里面就定义了这些型别，所有容器的迭代器都继承这个模板。

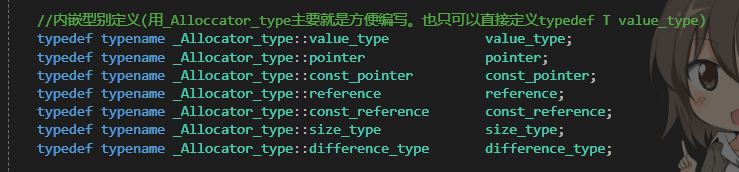


自定义的容器类模板，例如\_Deque\_iterator，然后容器内部在这样写

typedef \_Deque\_iterator<T> iterator

这样，当用户创建了容器对象时，就可以这样定义迭代器了

deque<int>::iterator iter;



这一串型别定义其实不是必要的，只是为了书写方便而已！ 统一规范。

像是value\_type、pointer啥的，主要是为了迭代器。

因为迭代器是承接算法和容器的

typedef \_Deque\_iterator<T, T&, T\*> iterator;枢纽。

看这一句，容器把自己内部元素是什么类型告诉迭代器了！然后迭代器又把这个信息自己包装一下，在算法中直接使用 Iterator::型别信息，这个型别信息就是容器内元素的信息啦~

迭代器只是作为一个中介，存储这些信息而已。

|  |
| --- |
| 迭代器和算法配合使用  vector<int> test;  ...  template<class Iter>  find(Iter \_First, Iter \_Last){...}  find( test.begin(), test.end() )  =>传入的参数展开看是这样 Iter 其实是 \_Vector\_iterator<int>  这样Iter：：value\_type 得到的是int了~ |

typename的作用：

举个例子value\_type转换以后其实是allocator<T>::valye\_type,因为是个模板嘛，在被具体化之前，编译器是不知道T是什么的，allocator<T>::valye\_type这一语句编译器就不知道是成员函数还是成员变量了。使用typename是告诉编译器，这个是个类型。

2019.12.10

先完成昨天没做完的分析...分析写在上面

**开发1**：规划算法开发事宜

先看书了解一下整个泛型算法怎么分类的，然后安排一个具体开发计划。 开一条分支algorithm，在上面做算法的开发。一周合并一次吧~

**阅读书籍**：STL源码剖析、C++标准库

为了让人顾名思义，STL设计者为算法命名时，引入两个特别的尾词：

1. \_if

如果算法有两种形式，**参数个数都相同**，但第一形式的参数要求传递一个值，第二形式的参数要求传递一个函数或仿函数，那么尾词加上\_if。例如：

find()用来搜寻具有某值的元素

find\_if() 接收一个被当做搜索准则的函数或仿函数，并搜寻第一个满足该准则的元素。

不是所有要求传递仿函数的算法都有尾巴\_if。如果有额外的参数来接收，那就可以同名。

1. \_copy

这个尾词用来表示在此算法中，元素不光被操作，还被会复制到目标区间。例如：

reverse()将区间中的元素颠倒次序，而reverse\_copy()则是逆序将元素复制到另一个区间。

STL算法可分为几大块：（C++标准库一书）

①非变动性算法

②变动性算法

③移除性算法

④变序性算法

⑤排序算法

⑥已序区间算法

⑦数值算法

划分成几个文件：

algobase.h 主要是常用的

algo.h

set\_algo.h

head\_algo.h 堆算法（建堆、调整堆、堆排序）

numeric.h

每天视情况而定，开发相对应的算法呀