

iterator

抽象概念，提供一种方法，使之能够依序巡防某个聚合物（容器）所含的各个元素，而又无需暴露该容器的内部表达方式。

iterator将容器（containers）和算法（algorithm）结合。

iterator是一种smart pointer

行为类似指针，所以迭代器最重要的工作是对operator*和operator->进行重载，本质实现依靠指针。

迭代器的相应型别

typeof()的实现方式，利用function template的参数推导机制

例：以下函数传入迭代器，试图得知迭代器所指向的类型

```
template <class I>
inline void func( I iter ) { return func_impl( iter, *iter) }

template <class I, class T>
void func_impl1( I iter, T t )
{
    T tmp;    // 解决问题
    ...
}
```

编译时，自动推导出T的类型，即*iter。

Traits编程技巧

利用内嵌型别声明的编程技巧与编译器的template参数推导机制，增强C++未能提供的关于型别认证方面的能力（弥补C++语言本身不足）

value type —— 迭代器所指对象的型别

使用template参数推导机制，只能推导参数，若需要返回值，则方法失效。

使用<内嵌型别声明>的方法解决问题。即在定义自定义类的迭代器时，额外声明出型别，但对原生指针不适用（原生指针就是一个迭代器，但不需要额外定义，则无法额外声明。）

故，引入模板偏特化（对原先型别的进一步限制）：

```
template <typename T>
class C {...}          // 这个泛化版本允许接受T为任何型别
template <typename T>
class C<T*> {...}      // 这个特化版本仅适用于"T为原生"指针的情况，有
```

优先匹配的权利

特性萃取机：（新增一层间接关系）

```
template <typename I> // 泛化版本，传入一个迭代器
struct iterator_traits
{
    typedef typename I::value_type      value_type;    // 迭代器型别
    typedef typename I::iterator_category iterator_category; // 迭代器
类型
    typedef typename I::difference_type difference_type; // 两个迭
代器间的距离
    typedef typename I::pointer        pointer;        // 指向迭代器所
指之物（指针）
    typedef typename I::reference      reference;      // 迭代器所指
之物(引用)
};
template <class T>      // 特化版本，优先匹配，传入原生指针
struct iterator_traits<T*>
{
    typedef T value_type;
}
```

Traits编程思路：

自定义迭代器:内嵌型别声明 `typedef T value_type`

↓↓↓

iterator_traits模板类:多了一层间接关系的同时,引入偏特化,解决原生指针的型别声明(包括const 原生指针)

例:

```
template <class T>
struct iterator_traits<T*>
{ typedef T value_type; }
```

↓↓↓

使用: 例: `template<class I>`
 `typename iterator_traits<I>::value_type`
 `func(I ite)`
 `{ return *ite;}`

迭代器分类

- Input Iterator: 只读
- Output Iterator: 只写
- Forward Iterator: 读写, 单向移动访问
- Bidirectional Iterator: 读写, 双向移动访问
- Random Access Iterator: 读写, 随机访问

任何一个迭代器, 其类型永远应该落在『该迭代器所隶属之各种类型中, 最强化那个』

而算法参数模板的命名规则:以算法所能接受之『最低阶迭代器类型命名』

迭代器类型的声明, 目的:

- 是为了促成重载机制的成功运作
- 采用继承, 减少「单纯只做传递调用」的函数

std::iterator的保证

STL提供一个iterators class, 每个新设计的迭代器都继承自它, 即可符合STL规范。

```
template<class Category, class T, class Distance = ptrdiff_t, class
Pointer=T*, class Reference = T&>
struct iterator
{
    typedef Category      iterator_category;
    typedef T             value_type;
    typedef Distance      difference_type;
    typedef Pointer       pointer;
    typedef Reference     reference;
}
```

后三个参数皆有默认值, 新的迭代器只需提供前两个参数即可。

`__type_traits`

负责萃取型别的特性，如：是否具有non-trivial default ctor? 是否具有non-trivial copy ctor? 是否具有non-trivial assignment operator? 是否具有non-trivial dtor?

若答案否定，在对这个型别（类）进行构造、析构、拷贝、赋值时，就可以采用最有效率的措施。

如果class内含指针成员，并且对它进行内存动态配置，那么这个class就需要实现自己的non-trivial-xxx