**STL**

**Allocator**

SGI STL使用alloc而不是allocator

**将内存分配/回收和数据构造/析构分离**

**Construct** \* 1：placement new

**Destroy** \* 4：两种重载特化，一种指针参数，一种迭代器参数

指针参数直接析构，迭代器判别是否trival：value type->traits->POD

**一级配置器**：malloc和free，配有malloc\_handler和realloc\_handler

**二级配置器**：16个不同大小的链表和内存池，大小超过128B调用一级配置器

List->memory pool->heap

内存池不足时，malloc，malloc失败则寻找有无未使用的list，无则调用一级配置器 p68

**Uninitiated\_copy / fill / fill\_n**

判断是否POD，是则直接调用高阶函数copy / fill，不是则依次拷贝构造/构造

对char特化，memmov

**POD：p73**

**Traits & Iterator**

如何获得迭代器所指的**对象型别(value type)**？p85

利用函数模板的**参数推导机制**，但是只能在目标函数内使用，无法返回

**声明类型内嵌**：

Typedef T value\_type;便可将value tupe作为返回值(前面要加上typename)

特例：并非所有迭代器都是类类型/class type，例如原生指针，因此无法类型内嵌

解决方法：**模板偏特化**

类模板有一个以上的template参数，可以对某个参数偏特化 p87

Iterator\_traits类接收迭代器类型T作为其模板参数

若T为迭代器，Iterator\_traits读取T的声明类型内嵌作为value type返回

若T为普通指针，Iterator\_traits专门特化一个T\*的版本，返回T作为value type

相当于迭代器或者指针都通过Iterator\_traits这一个间接封装类返回value type

PS. 对于const T\*也可以偏特化

为了使traits工作，每个迭代器都应该内部定义自己的value typ p89

**迭代器五种类型**

Value\_type

difference\_type

reference\_type(T&)

pointer\_type(T\*)

**iterator\_category，继承关系**

input ->

forward -> bidirectional -> random

output->

**执行时才确定**使用哪个版本影响效率，最好在**编译时就确定：函数重载**

为五种迭代器类型定义**五个class**，只用于激活不同的迭代器函数

STL算法用其能够接受的**最低类型的迭代器**命名参数**，因为派生类自动向基类转换**

**接口函数只使用最低类型迭代器，内部根据迭代器类型自动重载不同的底层函数**

**SGI \_\_type\_traits**

提取型别(type)的特性：**是否具有trival的ctor，dtor，copy ctor，assign？**

Trival则使用最有**效率**的操作，non-trival则调用构造/析构函数

C++基本型别都是non-trival

**Container**

**序列式容器：(关系是内含而不是继承)**

**Array -> vector -> heap -> priority\_queue**

**List**

**Deque -> stack / queue**

**关联式容器：**

**RB-Tree -> map, set, multimap, multiset**

**Hashtable -> unordered\_map/set, unordered\_multimap/set**

**Vector**

三个指针：**start，finish（size），end\_of\_storage（capaci），后两个为尾后表示**，迭代器是**普通指针**

新增元素时，如果超过容量则**扩容至两倍**，两倍不够则更大

由于是**连续线性空间**，扩容涉及到**重新申请空间、元素移动、空间释放**

Push\_back：先构造临时变量，然后调用移动/拷贝构造函数push

Emplace\_back：直接在尾部构造，效率更高

**List**

**双向链表**，迭代器为双向的，为节点指针的封，**SGI list还是环形链表**

P129，list操作不会使原迭代器失效，区别于vector

含有一个指向尾端的空白节点，初始时其prev和next都指向自身

**Deque**

**双向开口**，**分段的线性连续空间的组合**，可以**常数时间进行头尾操作**

**结构：**

**中控map**：一小块**连续**区间，每个元素为指针，指向一块连续的缓冲区

**缓冲区**：**连续的线性空间**，是deque存储的主体

**Start和finish**两个迭代器

**Map大小** 一旦map不够则重新分配

**迭代器：**

为一个单独且复杂的类，包含**多个指针：**

**Curr**：迭代器当前所指的缓冲区的元素

**First**：当前元素所在的缓冲区的头指针

**Last**：当前元素所在的缓冲区的尾后指针

**Node**：指向map中指向当前缓冲区的元素/节点

Deque初始状态有一个空的缓冲区

**Stack / Queue**

本身是容器适配器，默认选择deque作为底层容器

无迭代器

**Heap / priority\_queue**

更像是一组算法，而非容器，**STL默认大顶堆**

本质上为对底层容器进行heap算法，一般默认vector

**Push heap**：新节点插入到尾端，然后向上调整

**Pop heap**：将根节点和尾部互换，然后adjust down

**RB-Tree**

**根为黑，NULL为黑**

**红节点不可连续，任一道路上黑节点个数相同**

**新插入的节点初始为红色，后续可能调整颜色**

**数据结构：**

**header头节点（红色）**，其parent为root，left和right分别指向树中最大最小值

节点数目count

用于比较的仿函数

**插入**：p210，分为三种情况

迭代器为**双向迭代器**

**Begin为header的left，end为header**

插入时根据key比较来确定插入位置，节点存储value的值

根据**仿函数**从value得到key（set为本身，map为pair访问？）

可以**插入key相同的重复节点**，**<=向左**，即multiset/map

**Set**

Set的**value即为key**，因此**不可以改变value**，即**迭代器为const iter**

关联式容器应该使用其特有的**find函数**

**Map**

**Map元素内容为pair**，即包含key和value

**Map可以改变value，但不能改变key，因此其迭代器比较特殊**

Map的**下标运算符**返回值可以作为**左值或者右值**，因为是**reference返回**

**Hashtable**

哈希表，数据结构为**list的vector**

**一个vector元素/即list上面可能包含多个node，因为哈希碰撞**

**迭代器**：

遍历vector中每条list**，从一条list尾部跳到下一条list头部**

**只能前进，不能后退**

**Algorithm**

**函数模板，泛型编程**

**使用class T抽象化算法操作的对象的型别，使用迭代器抽象化操作对象的区间和移动**

**Copy**：强化效率，无所不用其极

对于**POD数据**，可以**直接内存复制**以提升效率

**SGI STL的copy**使用**函数重载**、**type traits**、**偏特化**等及其来加强效率

要求**输入是输入迭代器，输出是输出迭代器**

**函数重载**：对于**char和wchar\_t**

**泛化版本**：一个完全泛化、两个偏特化

**完全泛化**：**参数为迭代器，判断迭代器类型**

**Random iter则用n遍历，**用**赋值运算符**

否**则first++，**用**赋值运算符**

**偏特化**：**参数为T\*或const T\***，判断T是否有**trival operator**

**有则memove**

**无则用n遍历，调用non-trival的赋值运算符**

**Vector的copy是T\***偏特化路线

**Sort**：要求**random iteration**

数据大量是采用**快速排序**

数据量小于某个门槛（5-20）使用**插入排序**

为了防止递归过深（检查分割层次）还会使用**堆排序**

**Functors**

**仿函数/函数对象**，在**调用时可以像函数一样被调用**

仿函数行为类似函数，即**重载了函数运算符的类**

想要**将某种操作作为算法的参数**：

设计一个函数，**将函数指针**当作参数，但**无法满足抽象性的要求，也无可配接性**

设计一个**仿函数(**语言层面上是个class)，**用该仿函数产生的一个对象作为函数参数**

**为了可以被配接器修饰，仿函数也要定义自己的types/型别**

一元/二元，算术/逻辑/关系，证同/选择/投射

**Adapters**

配接器，将**一个class的接口**转换为**另一个class的接口**，使得原本因为接口不同而不能合作的classes可以一起运作

**仿函数配接器，迭代器配接器，容器配接器**

迭代器配接器：重载++，\*等操作

Insert iterators：

将赋值操作变为插入操作

迭代器内部维护一个容器，=操作转化为插入操作

没有++，--等操作

Iostream iterators

将迭代器绑定到iostream上

Reverse iterator

正向迭代器后退一格的位置为其反向迭代器