|  |  |
| --- | --- |
| 1. allocator类 2. allocator类2 3. allocator分配未构造的内存 4. 拷贝和填充未初始化内存的算法 | Allocator<stirng> alloc;  Auto const p = alloc.allocate(n);  Alloc.construct(q++);  Alloc.construct(q++,”hi”)  Alloc.construct(q++,10,’c’)  while (q1 != p1) {  alloc1.destroy(--q1);// 释放构造了的string，逆序，只能对构造了的进行destory  }  alloc1.deallocate(p1, 3);//归还内存  allocator<int> all2;  auto p2 = all2.allocate(ve.size() \* 2);  auto q2 = uninitialized\_copy(ve.begin(), ve.end(), p2);//目的位置，返回的是最后一个位置  uninitialized\_fill\_n(q2, ve.size(), 42);// |
| 1. allocator类 2. new 将内存分配和对象构造组合绑定了一起，delete将对象析构和内存释放组合了一起 3. 希望内存分配和对象构造分离可以使用allocator类 4. 列子，string被赋值了两次，一次默认初始化，一次赋值时   有可能没有用到内存也被附上了初始值，因为默认初始化  更重要的，没有默认构造函数的类无法动态分配数组   1. allocator类2 2. 头文件memory中是一个模板需指定可以分配的对象类型 3. 分配的内存是原始的 未构造的 4. 根据对象类型来确定恰当的内存大小和对齐位置。分配后返回第一个内存的地址吧 5. allocator分配未构造的内存   alloc1.construct(q1++);  alloc1.construct(q1++, 10, 'c');  alloc1.construct(q1++, "hi");  note:用construct构造对象。若没有构造的对象的内存使用了 未定义的  note:只能对真正构造了的元素进行destroy操作  destroy了元素后，可以将内存归还系统。Deallocate完成。大小参数需一样 不能大于初始化的大小   1. 拷贝和填充未初始化内存的算法 2. 两个伴随算法，拷贝和填充,定义在memory中 3. 可以用迭代器指定位置，或者迭代器指向的内存地址+n的形式 4. Copy返回最后一个构造的元素之后的位置，fill\_n不返回 | |