它们比数组更强大、更灵活，可以动态地增长（通常也可以收缩）、管理自己的内存、跟踪它们保存了多少对象、限制它们所支持的操作的算法复杂性，等等

第1条：小心选择容器。

向量、字符串、双端队列和列表。

标准STL关联容器，set、multiset、map和multimap。

非标准关联容器hash\_set、hash\_multiset、hash\_map和hash\_multimap。

vector是默认情况下应该使用的序列类型。当序列中间有频繁的插入和删除时，应该使用list。当大多数插入和删除发生在序列的开头或结尾时，双端队列是选择的数据结构。

标准的连续内存容器是vector、string和deque。非标准rope也是一个连续内存容器。

表示链表的容器（如list和slist）是基于节点的，所有标准关联容器都是如此。(They它们通常被实现为平衡树。）非标准的散列容器使用不同的基于节点的实现，正如你将在第25条中看到的。

是否需要能够在容器中的任意位置插入新元素？如果是这样，您需要一个序列容器;关联容器不行。

■你关心元素在容器中是如何排序的吗？如果没有，散列容器成为一个可行的选择。否则，您将需要避免使用哈希容器。

容器必须是标准C++的一部分吗？如果是这样的话，那么就排除了散列容器、slist和rope。

你需要什么类型的迭代器？如果它们必须是随机访问迭代器，那么从技术上讲，您只能使用vector、deque和string，但您可能也想考虑rope。(See第50条关于绳索的信息。）如果需要双向迭代器，则必须避免使用slist（见第50条）以及散列容器的一种常见实现（见第25条）

■当插入或删除发生时，避免移动现有的容器元素是否重要？如果是这样，你需要远离连续内存容器（见第5条）。

你需要事务语义来处理插入和擦除吗？也就是说，您是否需要可靠地回滚插入和擦除的能力？如果是这样的话，你需要使用一个基于节点的容器。如果您需要多元素插入的事务语义（例如，范围形式-见第5项），你会想选择列表，因为列表是唯一一个为多元素插入提供事务语义的标准容器。事务处理语义对于那些对编写异常安全代码感兴趣的程序员来说尤其重要。

条款2：小心容器独立代码的错觉。

STL基于泛化。数组被泛化到容器中，并根据它们所包含的对象类型进行参数化。

函数被一般化为算法，并根据它们使用的迭代器类型进行参数化。

指针被泛化为迭代器，并根据它们所指向的对象的类型进行参数化。

条款3：使容器中的对象的复制便宜且正确。

1. 图形用户界面, 文本, 应用程序

   描述已自动生成**STL容器的“文明”行为**：
   * STL容器（如vector）被设计为**按需创建对象**，而不是像数组那样在定义时就默认构造所有对象。
   * 例如，vector<Widget> vw;不会立即创建任何Widget对象，只有在添加元素时才会创建。
   * 通过reserve方法，可以为vector预分配内存空间，但仍然不会构造对象，直到实际需要时。
2. **与数组的对比**：
   * 数组在定义时会立即构造所有元素，即使这些元素可能不会被使用或会被立即覆盖。例如：
   * 这种行为可能导致不必要的性能开销，特别是当对象构造成本较高时。

第4项：调用empty而不是检查size（）是否为零。

第5条：与单元素对应函数相比，更喜欢范围成员函数。图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

当处理标准序列容器时，与范围成员函数相比，单元素成员函数的应用对内存分配器有更多的要求，更频繁地复制对象，和/或执行冗余操作。

第7条：当使用新指针的容器时，记住在容器被销毁之前删除指针。

图形用户界面, 文本, 应用程序

描述已自动生成

第8条：永远不要创建auto\_ptrs的容器。

第9条：在擦除选项中仔细选择。

文本

描述已自动生成

第10项：注意分配器约定和限制。

分配器可能有问题，需要谨慎使用。

项目11：理解自定义分配器的合法用途。

从根本上说，你获取一些共享内存，然后在其中构造一个向量，该向量使用共享内存进行自己的内部分配。当你完成向量的使用后，你调用它的析构函数，然后释放向量占用的内存。代码并不复杂，但它比上面声明一个局部变量要复杂得多。除非你真的需要一个容器（而不是它的元素）位于共享内存中，否则我建议你避免这个手动四步的分配/构造/销毁/释放过程。

正如这些例子所展示的，分配器在很多上下文中都很有用。只要遵守相同类型的所有分配器必须相等的约束，那么使用自定义分配器来控制通用内存管理策略、集群关系以及共享内存和其他特殊堆的使用就不会有任何问题。

条款12：对STL容器的线程安全性有现实的期望。