C++ STL 教程

C++ 标准模板库（Standard Template Library，STL）是一套功能强大的 C++ 模板类和函数的集合，它提供了一系列通用的、可复用的算法和数据结构。

STL 的设计基于泛型编程，这意味着使用模板可以编写出独立于任何特定数据类型的代码。

STL 分为多个组件，包括容器（Containers）、迭代器（Iterators）、算法（Algorithms）、函数对象（Function Objects）和适配器（Adapters）等。

使用 STL 的好处:

- **代码复用**：STL 提供了大量的通用数据结构和算法，可以减少重复编写代码的工作。
- **性能优化**：STL 中的算法和数据结构都经过了优化，以提供最佳的性能。
- **泛型编程**：使用模板，STL 支持泛型编程，使得算法和数据结构可以适用于任何数据类型。
- **易于维护**：STL 的设计使得代码更加模块化，易于阅读和维护。

C++ 标准模板库的核心包括以下重要组件组件：

组件	描述
容器（Containers）	容器是 STL 中最基本的组件之一，提供了各种数据结构，包括向量（vector）、链表（list）、队列（queue）、栈（stack）、集合（set）、映射（map）等。这些容器具有不同的特性和用途，可以根据实际需求选择合适的容器。
算法（Algorithms）	STL 提供了大量的算法，用于对容器中的元素进行各种操作，包括排序、搜索、复制、移动、变换等。这些算法在使用时不需要关心容器的具体类型，只需要指定要操作的范围即可。
迭代器（iterators）	迭代器用于遍历容器中的元素，允许以统一的方式访问容器中的元素，而不用关心容器的内部实现细节。STL 提供了多种类型的迭代器，包括随机访问迭代器、双向迭代器、前向迭代器和输入输出迭代器等。
函数对象（Function Objects）	函数对象是可以像函数一样调用的对象，可以用于算法中的各种操作。STL 提供了多种函数对象，包括一元函数对象、二元函数对象、谓词等，可以满足不同的需求。
适配器（Adapters）	适配器用于将一种容器或迭代器适配成另一种容器或迭代器，以满足特定的需求。STL 提供了多种适配器，包括栈适配器（stack adapter）、队列适配器（queue adapter）和优先队列适配器（priority queue adapter）等。

这些个组件都带有丰富的预定义函数，帮助我们通过简单的方式处理复杂的任务。

容器

容器是用来存储数据的序列，它们提供了不同的存储方式和访问模式。

STL 中的容器可以分为三类：

1、序列容器：存储元素的序列，允许双向遍历。

- std::vector：动态数组，支持快速随机访问。
- std::deque：双端队列，支持快速插入和删除。

std::list：链表，支持快速插入和删除，但不支持随机访问。

2、关联容器：存储键值对，每个元素都有一个键（key）和一个值（value），并且通过键来组织元素。

std::set：集合，不允许重复元素。

std::multiset：多重集合，允许多个元素具有相同的键。

std::map：映射，每个键映射到一个值。

std::multimap：多重映射，存储了键值对（pair），其中键是唯一的，但值可以重复，允许一个键映射到多个值。

3、无序容器（C++11 引入）：哈希表，支持快速的查找、插入和删除。

std::unordered_set：无序集合。

std::unordered_multiset：无序多重集合。

std::unordered_map：无序映射。

std::unordered_multimap：无序多重映射。

下面的程序演示了向量容器（一个 C++ 标准的模板），它与数组十分相似，唯一不同的是，向量在需要扩展大小的时候，会自动处理它自己的存储需求：

实例

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
   // 创建一个向量存储 int
   vector<int> vec;
   int i;

   // 显示 vec 的原始大小
   cout << "vector size = " << vec.size() << endl;

   // 推入 5 个值到向量中
   for(i = 0; i < 5; i++){
      vec.push_back(i);
   }

   // 显示 vec 扩展后的大小
   cout << "extended vector size = " << vec.size() << endl;

   // 访问向量中的 5 个值
   for(i = 0; i < 5; i++){
      cout << "value of vec [" << i << "] = " << vec[i] << endl;
   }

   // 使用迭代器 iterator 访问值
   vector<int>::iterator v = vec.begin();
   while( v != vec.end()) {
      cout << "value of v = " << *v << endl;
      v++;
   }

   return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
vector size = 0
extended vector size = 5
value of vec [0] = 0
value of vec [1] = 1
value of vec [2] = 2
value of vec [3] = 3
value of vec [4] = 4
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
```

关于上面实例中所使用的各种函数，有几点要注意：

push_back( ) 成员函数在向量的末尾插入值，如果有必要会扩展向量的大小。

size( ) 函数显示向量的大小。

begin( ) 函数返回一个指向向量开头的迭代器。

end( ) 函数返回一个指向向量末尾的迭代器。

STL 是 C++ 编程中不可或缺的一部分，它极大地扩展了 C++ 的功能，使得程序员能够编写出更加高效、可读性更强的代码。掌握 STL 的使用对于任何 C++ 程序员来说都是非常重要的。

C++ Web 编程　　C++ 标准库

2 篇笔记　　写笔记

865

C++ STL 之 vector 的 capacity 和 size 属性区别

size 是当前 vector 容器真实占用的大小，也就是容器当前拥有多少个容器。

capacity 是指在发生 realloc 前能允许的最大元素数，即预分配的内存空间。

当然，这两个属性分别对应两个方法：**`resize()`** 和 **`reserve()`**。

使用 **`resize()`** 容器内的对象内存空间是真正存在的。

使用 **`reserve()`** 仅仅只是修改了 capacity 的值，容器内的对象并没有真实的内存空间(空间是"野"的)。

此时切记使用 **`[]`** 操作符访问容器内的对象，很可能出现数组越界的问题。

下面用例子进行说明：

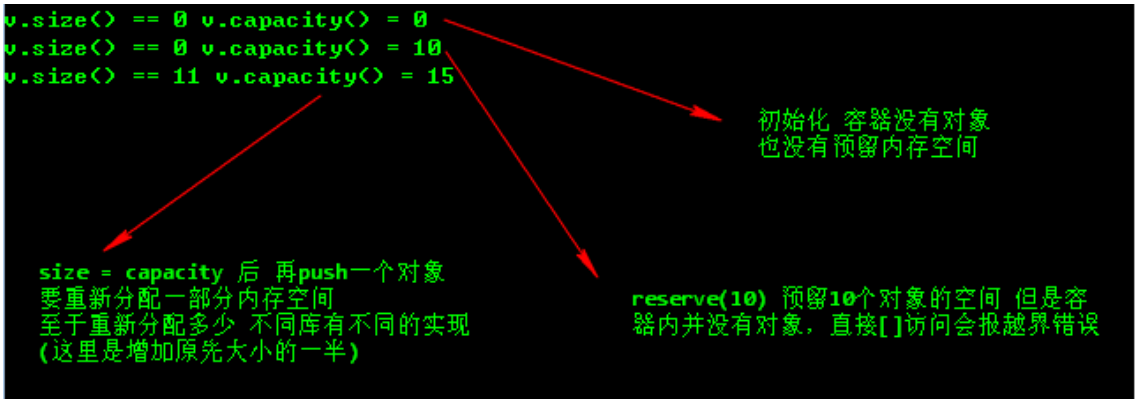
```
#include <iostream>
#include <vector>

using std::vector;
int main(void)
{
    vector<int> v;
    std::cout<<"v.size() == " << v.size() << " v.capacity() = " << v.capacity() <<
std::endl;
    v.reserve(10);
    std::cout<<"v.size() == " << v.size() << " v.capacity() = " << v.capacity() <<
std::endl;
    v.resize(10);
```

```
    v.push_back(0);
    std::cout<<"v.size() == " << v.size() << " v.capacity() = " << v.capacity() <<
std::endl;

    return 0;
}
```

运行结果为：(win 10 + VS2010)



注： 对于 **reserve(10)** 后接着直接使用 **[]** 访问越界报错(内存是野的)，大家可以加一行代码试一下，我这里没有贴出来。

这里直接用**[]**访问，vector 退化为数组，不会进行越界的判断。此时推荐使用 at()，会先进行越界检查。

> **相关引申：**
>
> 针对 capacity 这个属性，STL 中的其他容器，如 list map set deque，由于这些容器的内存是散列分布的，因此不会发生类似 realloc() 的调用情况，因此我们可以认为 capacity 属性针对这些容器是没有意义的，因此设计时这些容器没有该属性。
>
> 在 STL 中，拥有 capacity 属性的容器只有 vector 和 string。