链滴

C++ vector 心得整理

作者：Hanseltu

原文链接：https://ld246.com/article/1557241755949

来源网站：链滴

原文链接 C++ vector 心得整理

特别说明:

使用vector前，请先 #include <vector>。以下文章若无特别，则均假设如下：

vector<int> v, v1, v2;

int array[] = {0,1,2,3,4};

0. vector 快速使用

```
vector<int> v1;
vector<double> v2;

// 清除内容, 重新设定大小
v1.clear(), v2.clear();
v1.resize(5);
v2.resize(10);

// 新增元素
int i;
for(i=0; i<v1.size(); i++)  v1.push_back(i);
for(i=0; i<v2.size(); i++)  v2[i] = double(i);
```

提问：一个 new operator 常见问题是，new 有沒有办法顺便给初始值?

以 int *v = new int[3] 为例，new 时想顺便设定 v[0]=1, v[1]=2, v[2]=3，怎么做?

● **答:** 这在 C++11 之前做不到，g++ 编译参数下 -std=c++11 或 -std=c++0x 可以做得到 int *v = ew int[3]{1,2,3};

注意到大括号小括号的使用，

int *v1 = new int[3];

int *v2 = new int{3};

这两个是不同意思。v1 指的是配置 3 个 int ，不给初始值； v2 指的是配置 1 个 int，初始值给 3。括号里的东西统称为**initializer lists**

除了 new 可以这么做之外，vector 也可以这么搞。

vector<int> v{1,2,3}; v 就自动给三个元素 1, 2, 3 当初值。

1. 大小控制

(1.1) 检查是否为空 v.empty();

(1.2) 直接将v清空 v.clear();

(1.3) 检查目前v大小 v.size();

(1.4) 检查最大容量 v.maxsize();

(1.5) 检查目前容量 v.captacity();

(1.6) 清除所有内容 v.clear();

(1.7) 清除指定元素 v.erase(v.begin()); v.erase(v.end()-1);

(1.8) 读取并删除最后一个元素 v.pop_back(v.size()-1);

(1.9) 重新设定大小 v.resize(5);

关于 v.resize() 要说明一下，若开始是使用 vector<int> v(3,5)，将v设成3个元素，每个元素都为5，后调用 v.resize(2) 之后，边界大小被调整成2 ，若再调用 v.resize(6)，则 v[0], v[1] 为 5，v[2]~v[5] 为 0。

(1.10) 调整大小 v.reserve(6);

和 v.resize() 有所不同，v.resize() 是将 vector 重新调整大小。若开始是用 vector<int> v(3, 5)，使用 .reserve(6)，实际上 vector v 之大小仍为3，若调用 v.at(3) 将报错。另外，reserve是容器预留空间但并不真正创建元素对象，在创建对象之前，不能引用容器内的元素，因此当加入新的元素时，需要用 ush_back()/insert()函数。

2. 构造函数(Construction)

vector<Type> Var(Count, Val) ;

(1) Var : 是一个 vector 变量名 ， 里面元素存放的是 Type 类型的val。

(2) Count : 在初始化的時候，Var 就已有 Count 个元素。

(3) Val : Var 里，每个元素 (一共 Count 个)，都存放 Val。

(4) 如 :vector<int> v1(3, 1); v1 初始大小设 3，每个元素都设 1。

vector< vector<int> > v2(x , vector<int>(y, 1));

(1) v2 : 是一个 vector<int> 型数组。

(2) x : v2 共有 x 个 vector<int>

(3) vector<int>(y, 1) : 每个 v2[i] (就是 vector<int>)，再以 vector<int>(y, 1) 做为初值设定。

```
#define CNT 5
int array[] = {0,1,2,3,4};
int *arr2 = new int[CNT];
for(int i=0; i<CNT; i++) arr2[i] = i;
```

(2.1) 建立空的 vector: vector <int> f1; // 建立空的 vector

(2.2) 建立4个元素为100的v: vector <int> f2(4, 100);

(2.3) 用iterator直接从f2复制元素: vector <int> f3(f2.begin(), f2.end());

(2.4) 从 array(stack) 复制: vector <int> f4(array, array + sizeof(array)/sizeof(int));

(2.5) 从 array(heap) 复制: vector <int> f5(arr2, arr2+CNT) ;

注意，若 vector 初始化为empty，在建立任何元素之前，不可使用下标访问数组。如:

```
vector<int> v;
v[0] = 1;
v[1] = 2;
```

3. 遍历(travel)

(3.1) ,(3.2) 假设如下：

```
int i;
int array[] = {0,1,2,3,4};
vector<int> v(array, array+sizeof(array)/sizeof(array[0]);
```

(3.1) 使用下标访问 / function member - at

```
// 遍历, 0,1,2,3,4
for(int i=0; i<v.size(); i++) cout << v[i] << " ";
for(int i=0; i<v.size(); i++) cout << v.at(i) << " ";
```

(3.2) 使用 iterator

```
vector<int>::iterator it_i;
// 遍历
for(it_i=ff.begin(); it_i!=ff.end(); ++it_i) cout << *it_i << " ";
```

4. 将vector 或 array 赋值给另一个vector

```
int array[] = {0,1,2,3,4};
vector<int> v(10,0); // {0,0,0,0,0,0,0,0,0,0}
vector<int> v1;
v1.assign(10, 0); // v1 设为 10 个 0
v1.assign(v.begin(), v.end()); // v1 复制 v
v1.assign(v.begin(), v.begin()+5); // 复制 v 前5个元素到 v1
v1.assign(array, array+5); // 复制 array 前5个元素到 v1
```

5. 添加/移除元素

(5.1) 从末尾添加: for(i=0; i<10; i++) v.push_back(i);

(5.2) 从末尾移除: while(!v.empty()) v.pop_back();

6. 二维矩阵的使用

```
unsigned i,j;
vector<vector<int> > x; // x[M][N]
x.resize(M);
for(i=0; i!=M; ++i) x[i].resize(N);

for(i=0; i!=M; ++i)
    for(j=0; j!=N; ++j)
        x[i][j] = i*N+j;
```

7. vector中的内存管理

(7.1) 动态分配策略

可能对这个问题感兴趣的不多，有一个**广为流传的动态分配策略笔者感到很疑惑，不少书籍或文章

会提到，**当(实际用量) > (最大容量) 时是增长 2 倍**，而 1/2/2 = 0.25，所以当 **(实际用量) < 0.25 (大容量) 时就缩减为一半。**C++ Primer 里面有个例子，测试代码大致如下

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    size_t c1 = v.capacity();
    size_t c2;
    for(int i=0; i<100000; ++i) {
        v.push_back(i);
        c2 = v.capacity();
        if(c1!=c2) {
            c1 = c2;
            cout << "capacity = " << c1 << endl;
        }
    }
    return 0;
}
```

以上代码在 g++ 和 vc编译器下编译运行的结果并不相同，g++下是很有规律的**1 2 4 8 16 32 ....**，

而 vc 下结果是 **1 2 3 4 6 9 13 19 28 42 63 94 141 211 316 474 711 1066 1599 2398 3597 53 5 8092**...，这规律实在找不出来。

(7.2) *vector当做指针参数

如果从C语言转到C++来的可能会有一些疑惑，比如以下函数：

```
void display(int *arr, int n)
{
    int i;
    for(i=0; i<n; ++i)
        printf("%d", arr[i]);
}
```

那么使用 vector<int> arr要怎么传参数给display函数呢，实际上传&arr[0]即可。如：

```
int main()
{
    const size_t n = 3;
    vector<int> v(n, 1);
    display(&v[0], n);
    return 0;
}
```

那如果要传 vector<vector<int>> arr 给 void display(int **arr, int row, int col);函数呢？这个要么操作？实际开发过程中可能不会遇到这种情况，如果遇到笔者还没有找到合适的方法。

(7.3) Capacity / Size

有几个和大小 / 容量相关的注意事项说明一下。

若 vector 在管理内存时，经常在 heap 做 realloc 动作会降低程序性能，于是会有 capacity 和 size 个东西。所谓的 capacity 指的是目前在 heap 上配置的元素个数，但实际上使用的数量是 size，而是 capaicty (也就是 capacity >= size 必定成立)。故在讨论时，应当把capacity / size 分开思考，为不同的东西。

std::vector::size : 取得目前 vector ， 实际使用的元素个数。

std::vector::capacity: 取得目前 vector ， 可使用的元素个数。

std::vector::max_size: 取得 vecotr 「可以配置」的最大元素个数，注意它和 size 和 capacity 的不，这个数值其实很重要，特别是在做 resize / reverse 前建议先用这个做处理，降低成本。当 std::vector::reverse 比这个数值还大时，会丢出 length_error 异常。

std::vector::resize : 指定 vector 元素个数。若 resize 比原本元素还多时，多出来的预设值变为 0 当然也可用 vector::resize(Count, Val)，多出来的会以 Val 填满。**若 vector 放的是 class，resize 原本的还小时， 不会调用destructor。**

std::vector::reverse: reverse 是自己控制内存空间，若进行 reverse(100)，则是在 heap 里 **至少留 **了 100 个可用元素。考虑一种情况，若 reverse 比原本的 std::vector::size 还小时，这时并不把 heap 缩得比 size 还小，故用了 **至少保留**这字眼。

std::vector::empty: empty 并不会做任何 capacity / size 上的调整，它只是**判断**目前 vector 里是有元素存在。

std::vector::clear: clear 是降 vector 整个清掉，**若 vector 里放的是 class，clear 会逐一调用 destuctor，但 capacity 并不一定会直接清零**(且通常都不会动)。

std::vector::erase: erase 可指定将 vector 某个元素，或某个范围的元素清除，**会调用 destructor。**

(7.4) std::vector::at / operator[]

一般用到 vector 时，大多索引方式用的是下标 (operator [])，较少用 at，**这两个符号相似，但不完一样**，最大差别是，**std::vector::at 会检查 bounding，但 operator[] 不会检查 bounding，若超时，std::vector::at 会丢出 out_of_range 异常。**

部分参考：STL Note