Choice between vector::resize() and vector::reserve()

I am pre-allocating some memory to my a `vector` member variable. Below code is minimal part

```
class A {
  vector<string> t_Names;
public:
  A () : t_Names(1000) {}
};
```

Now at some point of time, if the `t_Names.size()` equals `1000` . I am intending to increase the size by `100` . Then if it reaches `1100` , again increase by `100` and so on.

My question is, what to choose between `vector::resize()` and `vector::reserve()` . Is there any better choice in this kind of scenario ?

Edit: I have sort of precise estimate for the `t_Names` . I estimate it to be around `700` to `800` . However in *certain* (seldom) situations, it can grow more than `1000` .

c++ vector

edited Sep 13 '11 at 8:36

asked Sep 13 '11 at 6:38

iammilind
33k 10 72 157

13 You realize that doing this means that vector growth is no longer *amortized constant time* and you lose one of the performance benefits of using `std::vector` . – Blastfurnace Sep 13 '11 at 7:37

4 Answers

The two functions do vastly different things.

The `resize()` method (and passing argument to constructor is equivalent to that) will insert or delete appropriate number of elements to the vector to make it given size (it has optional second argument to specify their value). It will affect the `size()` , iteration will go over all those elements, push_back will insert after them and you can directly access them using the `operator[]` .

The `reserve()` method only allocates memory, but leaves it uninitialized. It only affects `capacity()` , but `size()` will be unchanged. There is no value for the objects, because nothing is added to the vector. If you then insert the elements, no reallocation will happen, because it was done in advance, but that's the only effect.

So it depends on what you want. If you want an array of 1000 default items, use `resize()` . If you want an array to which you expect to insert 1000 items and want to avoid a couple of allocations, use `reserve()` .

EDIT: Blastfurnace's comment made me read the question again and realize, that in your case the correct answer is **dont't preallocate** manually. Just keep inserting the elements at the end as you need. The vector will automatically reallocate as needed and will do it *more* efficiently than the manual way mentioned. The only case where `reserve()` makes sense is when you have reasonably precise estimate of the total size you'll need easily available in advance.

EDIT2: Ad question edit: If you have initial estimate, than `reserve()` that estimate and if it turns out to be not enough, just let the vector do it's thing.

edited Sep 7 '14 at 20:17

answered Sep 13 '11 at 6:49

Jan Hudec
33k 4 49 78

I have edited the question. I do have certain estimate for the `vector` . – iammilind Sep 13 '11 at 8:37

"The only case where reserve() makes sense is when you have reasonably precise estimate of the total size you'll need easily available in advance." - not strictly true, since calling `reserve()` yourself at specific times might sometimes help you manage any pointers or iterators that you have for elements of the vector (and in particular that they're invalidated by reallocation). Not that there's any sign in this

question that such things are going on. And it's true that you need to know an upper bound on how many elements you'll add during the time your pointers/iterators are needed. – Steve Jessop Sep 13 '11 at 9:20

3 @Jan: well, it's fragile or not according to how difficult you've made it for yourself to maintain the required property. Something like `x.reserve(x.size() + newdata); vector<int>::iterator special_element = get_special_element(x); for (int i = 0; i < newdata; ++i) { if some_function(i, special_element) x.push_back(i); }` is pretty robust as far as reserving the space is concerned. I've no idea how many elements will actually be added, but I have an upper bound. Of course when in doubt, with vectors you can just use indexes instead of iterators, the difference is usually negligible. – Steve Jessop Sep 13 '11 at 9:33

3 Your wording makes sense to someone already aware of the correct answer, but could easily mislead people needing to ask the question. "resize()...will insert given number of elements to the vector" - only true the first time it's used - it generally inserts the difference between the requested number and the pre-existing `size()`. "The reserve() method only allocates memory" - it may or may not allocate memory depending on whether `capacity()` is already sufficient, it may also need to move elements and deallocate their original memory. "want to avoid a couple of allocations" and copies etc – Tony D Dec 28 '11 at 8:46

2 Actually, reserving before pushing is vital and must be used. Assume that you are coding some kind of 3d model loader and the model has like 15000 vertices. If you try to push_back each vertex while loading without pre-allocating them first, it will take serious time. I personally experienced that, I tried to load a car .obj model with near 100000 vertices, It took 30 seconds. Then I refactored the code using pre-allocation with .reserve(), now it takes 3 seconds. Just putting a .reserve(100000) in the beginning of the code saved 27 seconds. – deniz Oct 12 '13 at 7:25

`resize()` not only allocates memory, it also *creates* as many instances as the *desired* size which you pass to `resize()` as argument. But `reserve()` only allocates memory, it doesn't create instances. That is,

```
std::vector<int> v1;
v1.resize(1000); //allocation + instance creation
cout <<(v1.size() == 1000)<< endl;   //prints 1
cout <<(v1.capacity()==1000)<< endl; //prints 1

std::vector<int> v2;
v2.reserve(1000); //only allocation
cout <<(v2.size() == 1000)<< endl;   //prints 0
cout <<(v2.capacity()==1000)<< endl; //prints 1
```

Output (online demo):

```
1
1
0
1
```

So `resize()` may not be desirable, if you don't want the default-created objects. It will be slow as well. Besides, if you `push_back()` new elements to it, the `size()` of the vector will further increase *by allocating new memory* (which also means moving the existing elements to the newly allocated memory space). If you have used `reserve()` at the start to ensure there is already enough allocated memory, the `size()` of the vector will increase when you `push_back()` to it, **but it will not allocate new memory again until it runs out of the space you reserved for it**.

edited Mar 3 '14 at 15:45

phonetagger
3,401 7 25

answered Sep 13 '11 at 6:47

Nawaz
173k 42 370 584

2 After doing `reserve(N)`, we can use `operator []` harmlessly. correct ? – iammilind Sep 13 '11 at 8:48

While most implementations will allocate the exact amount you request by `reserve`, the specification only requires it allocates at least that much, so some implementations may round up to some boundary and thus show higher capacity than 1000. – Jan Hudec Sep 13 '11 at 8:49

6 @iammilind: No, if the index is greater or equal to `v.size()`. Note that `reserve(N)` doesn't change `size()` of vector. – Nawaz Sep 13 '11 at 8:51

3 @iammilind: INcorrect. After calling reSERVE, no entries are added, only enough memory for adding them is obtained. – Jan Hudec Sep 13 '11 at 8:51

Thanks Nawaz; my bad point. Feeling lazy to research enough for the relation between `[]` and `reserve()`. – iammilind Sep 13 '11 at 8:59

From your description, it looks like that you want to "reserve" the allocated storage space of vector t_Names.

Take note that `resize` initialize the newly allocated vector where `reserve` just allocates but does not construct. Hence, 'reserve' is **much faster** than 'resize'

You can refer to the documentation regarding the difference of resize and reserve

answered Sep 13 '11 at 6:51

dip
475 3 7

1 Please refer here instead: vector and capacity *(why?)* – sehe Sep 13 '11 at 8:06

Thanks for the addition of link, sehe – dip Sep 13 '11 at 8:58

reserve when you do not want the objects to be initialized when reserved. also, you may prefer to logically differentiate and track its count versus its use count when you resize. so there is a behavioral difference in the interface - the vector will represent the same number of elements when reserved, and will be 100 elements larger when resized in your scenario.

> Is there any better choice in this kind of scenario?

it depends entirely on your aims when fighting the default behavior. some people will favor customized allocators -- but we really need a better idea of what it is you are attempting to solve in your program to advise you well.

fwiw, many vector implementations will simply double the allocated element count when they must grow - are you trying to minimize peak allocation sizes or are you trying to reserve enough space for some lock free program or something else?

answered Sep 13 '11 at 6:57

justin
81.4k 10 116 169