第13条：

vector和string优先于动态分配的数组。

第14条：

使用reserve来避免不必要的重新分配

vector和string可以自动增长以便容器放入其中的数据。

1. 分配一块大小为当前容量的某个倍数的新内存，大多数情况下，容量以2倍增长（貌似visual studio不是）。
2. 把容器的所有元素从旧的内存拷贝到新的内存中。
3. 析构掉旧内存中的对象。
4. 释放旧内存。

例：程序effective\_stl\_test4

class Test

{

public:

Test(*string* name) : name\_(name)

{

*cout* << "Construct Test " << name\_ << *endl*;

}

Test(const Test& org) : name\_(org.name\_)

{

*cout* << "Copy construct Test " << name\_ << *endl*;

}

virtual ~Test()

{

*cout* << "Destruct Test " << name\_ << *endl*;

}

private:

*string* name\_;

};

Test t1("1");

Test t2("2");

Test t3("3");

*cout* << *endl*;

*vector*<Test> vec;

*cout* << "first push" << *endl*;

vec.*push\_back*(t1);

*cout* << "vector size = " << vec.*size*() << " capacity = " << vec.*capacity*() << *endl*;

*cout* << *endl*;

*cout* << "second push" << *endl*;

vec.*push\_back*(t2);

*cout* << "vector size = " << vec.*size*() << " capacity = " << vec.*capacity*() << *endl*;

*cout* << *endl*;

*cout* << "third push" << *endl*;

vec.*push\_back*(t3);

*cout* << "vector size = " << vec.*size*() << " capacity = " << vec.*capacity*() << *endl*;

*cout* << *endl*;

输出为：

Construct Test 1

Construct Test 2

Construct Test 3

first push

Copy construct Test 1

vector size = 1 capacity = 1

second push

Copy construct Test 2

Copy construct Test 1

Destruct Test 1

vector size = 2 capacity = 2

third push

Copy construct Test 3

Copy construct Test 1

Copy construct Test 2

Destruct Test 1

Destruct Test 2

vector size = 3 capacity = 3

Destruct Test 1

Destruct Test 2

Destruct Test 3

Destruct Test 3

Destruct Test 2

Destruct Test 1

从输出可以看出，第2次push和第3次push时，都是先将新插入的元素拷贝到vector中，再把之前插入的重新拷贝到容器中，然后再析构掉之前插入的元素。

size()：告诉你容器有多少个元素。

capacity()：告诉你容器利用已经分配的内存可以容纳多少个元素。这是容器能容纳的元素总数。capacity() - size()就是容器的剩余空间，如果为0，下次插入操作，就会重新分配内存空间。

resize(Container::size\_type n)：强迫容器改变到包含n个元素的状态。调用resize后，size()将返回n。如果n比当前的大小要小，则容器尾部的元素会被析构；如果n比当前的大小要大，要重新分配内存，并通过默认构造函数创建元素添加到容器尾部。

reserve(Container::size\_type n)：强迫容器把它的容量至少变为n。如果n比当前的容量大，则重新分配内存；如果n比当前的容量小，vector忽略此操作，string则可能把自己的容器减小为size()和n中的最大值。

使用reserve减少内存的重新分配：

例：effective\_stl\_test4

*vector*<Test> vec1;

vec1.*reserve*(3);

vec1.*push\_back*(t1);

vec1.*push\_back*(t2);

vec1.*push\_back*(t3);

输出为：

Copy construct Test 1

Copy construct Test 2

Copy construct Test 3

Destruct Test 1

Destruct Test 2

Destruct Test 3

第15条：

string的多样性。

第16条：

把vector和string传递给旧的C API。

把vector传递给旧的C API是可行的，而且可以通过旧的C API来修改vector元素的值，但不能改变vector的个数。

例：程序effective\_stl\_test5

// 利用C API初始化vector

*size\_t* FillVector(double\* array\_ptr, *size\_t* size)

{

for (*size\_t* i = 0; i < size; ++i)

{

array\_ptr[i] = i + 0.1;

}

return size;

}

int max\_number = 10;

*vector*<double> vec(max\_number); // 需要分配vector的大小

// 传递vector给C API函数

// 注意：C API数组的大小必须小于或等于vector的size

vec.*resize*(FillVector(&vec[0], 5));

PrintElements(vec, "C API Fill vector: ");

输出为：

C API Fill vector: 0.1 1.1 2.1 3.1 4.1

将string传递给C API是可行的，可以通过string::c\_str()函数转换为const char\*，这样就不能修改string的内容了。如果希C API望初始化或修改string，可以通过一种间接的方式：先将char字符写入vector容器，然后用vector<char>来初始化string。通过这种方式，还可以利用C API来初始化其它的容器，如list、deque、set等。

例：程序effective\_stl\_test5

*size\_t* FillString(char\* array\_ptr, *size\_t* size)

{

for (*size\_t* i = 0; i < size; ++i)

{

array\_ptr[i] = 'a' + i;

}

return size;

}

*vector*<char> vec1(max\_number);

*size\_t* chars = FillString(&vec1[0], vec1.*size*());

*string* str(vec1.*begin*(), vec1.*begin*() + chars);

*cout* << "str = " << str << *endl*;

输出为：

str = abcdefghij

第17条：

使用swap去除多余的容量。

容器中如果插入许多元素后，然后erase某些元素后，并不会改变容器的容量capacity。如果希望去除多余的容量，可以使用swap函数

例：程序effective\_stl\_test6

*vector*<int> vec;

vec.*reserve*(10);

vec.*push\_back*(1);

vec.*push\_back*(2);

*cout* << "capacity size = " << vec.*capacity*() << *endl*;

// 通过swap来清除多余的容量

// 创建一个临时的vector并拷贝vec

// 拷贝时只会为所拷贝的元素分配内存

// 所以这个临时的vector没有多余的空间

*vector*<int>(vec).*swap*(vec);

*cout* << "capacity size = " << vec.*capacity*() << *endl*;

// 清空vector

*vector*<int>().*swap*(vec);

输出为：

capacity size = 10

capacity size = 2

第18条：

避免使用vector<bool>，使用deque<bool>和bitset来代替。