C++ STL中vector的实现-lqk188-ChinaUnix博客

STL是C++的精髓，其中的vector又是用的最广的一种容器。下面介绍一下vector的一种简单实现（这里参考了Mark Weiss的数据结构C++实现一书），废话不多说，直接上代码，

点击(此处)折叠或打开

```
#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;

template <typename Object>
class Vector
{
public:
    explicit Vector(int initSize = 0) : theSize(initSize), theCapacity(initSize+SPARE_CAPACITY)
    {
        objects = new Object[theCapacity];
    }
    Vector(const Vector & rhs) : objects(NULL)
    {
        operator=(rhs);
    }
    ~Vector()
    {
        delete [] objects;
    }

    const Vector & operator=(const Vector & rhs)
    {
        if (this != &rhs)
        {
            delete [] objects;
            theSize = rhs.size();
            theCapacity = rhs.theCapacity;

            objects = new Object[capacity()];
            for (int k = 0; k < size(); k++)
                objects[k] = rhs.objects[k];
        }
        return *this;
    }
    void resize(int newSize)
    {
        if (newSize > theCapacity)
            reserve(newSize*2+1);
        theSize = newSize;
    }
    void reserve(int newCapacity)
```

```
45.       {
46.           if (newCapacity < theCapacity)
47.               return;
48.
49.           Object * oldArray = objects;
50.           objects = new Object[newCapacity];
51.           for (int k = 0; k < theSize; k++)
52.               objects[k] = oldArray[k];
53.
54.           theCapacity = newCapacity;
55.           delete [] oldArray;
56.       }
57.
58.       Object * operator[](int index)
59.       {
60.           return objects[index];
61.       }
62.       const Object & operator[](int index) const
63.       {
64.           return objects[index];
65.       }
66.       bool empty() const
67.       {
68.           return size() == 0;
69.       }
70.       int size() const
71.       {
72.           return theSize;
73.       }
74.       int capacity() const
75.       {
76.           return theCapacity;
77.       }
78.
79.       void push_back(const Object & x)
80.       {
81.           if (theSize == theCapacity)
82.               reserve(2*theCapacity+1);
83.           objects[theSize++] = x;
84.       }
85.
86.       void pop_back()
87.       {
88.           theSize--;
89.       }
90.
91.       const Object & back() const
92.       {
93.           return objects[theSize-1];
94.       }
95.
96.       typedef Object * iterator;
97.       typedef const Object * const_iterator;
98.
99.       iterator begin()
100.      {
101.          return &objects[0];
```

```
102.        }
103.
104.        const_iterator begin() const
105.        {
106.            return &objects[0];
107.        }
108.
109.        iterator end()
110.        {
111.            return &objects[theSize];
112.        }
113.
114.        const_iterator end() const
115.        {
116.            return &objects[theSize];
117.        }
118.
119.        enum
120.        {
121.            SPARE_CAPACITY = 16
122.        };
123.
124.    private:
125.        int theSize;
126.        int theCapacity;
127.        Object * objects;
128.    };
129.
130.    int _tmain(int argc, _TCHAR* argv[])
131.    {
132.        Vector<string> aVec;
133.        aVec.push_back("hello");
134.        aVec.push_back("my name is Lily");
135.        aVec.push_back("I hope we'll be friends");
136.        aVec.push_back("Thanks");
137.        Vector<string>::iterator iter = aVec.begin();
138.        while (iter != aVec.end())
139.            cout << *iter++ << endl;
140.        return 0;
141.    }
```

先从vector的private成员变量说起，定义theSize，即size()的返回值，该值表明了vector的size大小；theCapacity，即vector的容量，注意它与size不是一个概念，size是指一个容器里面装了多少元素，capacity指该容器目前占用的内存能容纳多少元素，但未必里面就有那么多元素，可能capacity非常大，但size很小；objects指针里面存放的是vector分配内存的地址。当定义一个空的vector时，capacity是多少？在这里是16，请见code的120行定义的SPARE_CAPACITY，以及11行的构造函数，当未指定initSize的时候，我们new一个SPARE_CAPACITY的数组，用来存放Object对象。

下一个需要关注的函数是第44行的reserve()，reserve就是指定vector去分配一个指定大小的内存，当然如果这个指定的大小小于当前vector的capacity，那我们就什么都不做，如果大于当前capacity，我们先new一个new capacity的内存，然后把vector当前保存的元素拷贝到新new的内存中，最后删除之前分配的内存（注意删除数组要用delete [] p的形式，否则会发生内存泄

漏）。
接下来我们来看一下迭代器的定义，我们这里把iterator直接定义为指向Object的指针，然后再定义begin(), end()等函数，这几个函数都很简单，直接返回&objects[x]即可。
最后我们看代码79行push_back的定义，这个函数是vector使用者用的最多的操作之一，实现push_back时要考虑vector容量是否已满的情况，如果容量已满，即theSize与theCapacity相等，则reserve一个2倍的capacity加1即可。在这里理论上只要reserve的容量大于当前capacity，那reserve多大的内存都可以，但是如果reserve的内存过小（比如每次只增加一个Object大小的内存，那么以后每次push_back操作都会引起reserve操作，而reserve操作是要涉及new delete动作的，频繁进行会非常耗时），所以一般每次容量已满的话都reserve一个2倍的当前capacity的大小是理想的方法。
测试程序非常简单，定义一个装string对象的vector，然后进行push_back，最后打印出来，显示如下，


```
hello
my name is Lily
I hope we'll be friends
Thanks
```

写一个vector对理解C++ STL绝对是非常有帮助的，看似简单的一个vector实际上里面涉及了C++的模板，重载等概念，对于一个C++的新手来说，能写对这些语法并且通过编译就不是一件容易的事，所以学习软件编程最好的方法还是写一些实实在在的程序，大有裨益。