**迭代器**

1. 为什么用迭代器，因为迭代器比较通用
2. 现在我们拿上节的vector来进行举例子
   1. 常规 iterator
      1. 代码

//初始化一下

vector<int> vec(4, 8);

//迭代器的定义，首先类型的声明，要与我们生命的是一样的e

vector<int>::iterator iter;

//那么现在我们可以把迭代器指向vec的开头

iter = vec.begin();

//那么现在我们用迭代器来=进行元素的遍历

for (; iter != vec.end(); iter++)

{

std::cout << \*iter << " ";

}

* 1. reverse\_iterator（反向迭代）
     1. vector<int>::reverse\_iterator iter;
     2. 相对应得是rbegin、rend
  2. const\_iterator（不能通过迭代来进行值得修改）
     1. 相对应得就是cbegin、cend
  3. 那么我们之前学过auto类型自动判断，所以上面得步骤可以进行简化
     1. 代码

//初始化一下

vector<int> vec(4, 8);

//那么现在我们可以把迭代器指向vec的开头

auto iter = vec.rbegin();

//那么现在我们用迭代器来=进行元素的遍历

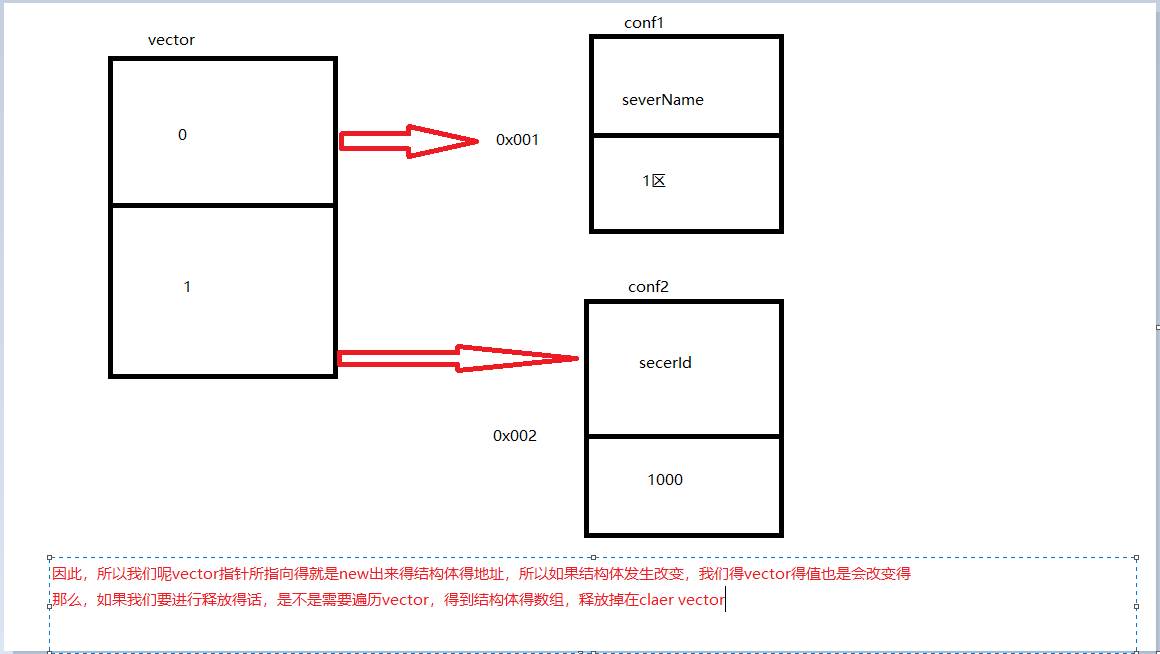
for (; iter != vec.rend(); iter++)

{

std::cout << \*iter << " ";

}

1. 迭代器的失效
   1. 其实很简单，就是我们用迭代器得时候不要进行增加，删除操作就可以得
   2. 要是进行增加和删除操作。最好就是直接break
2. String的结合迭代器的实战
   1. 用vector来存放结构体
   2. 用实例化两个结构体，生成两个内存
   3. 把两个结构体push进去vector
   4. 遍历vector释放两个内存
   5. Clear 清空vector



* 1. 代码

#include <vector>

#include <algorithm>

using std::vector;

struct config {

char key[40];

char val[100];

};

//我们来实现一个函数，就是通过key，来得到val，大家如果有看我得c语言，应该就会有印象

//传进来、引用或者指针比较节省内存

char \*getInfo(vector<config \*> &res, const char \*key) {

for (auto iter = res.begin(); iter != res.end(); iter++) {

if (\_stricmp((\*iter)->key, key) == 0) {

return (\*iter)->val;

}

}

return nullptr;

}

int main() {

//用new开辟两个内存空间

config \*conf1 = new config;

config \*conf2 = new config;

strcpy\_s(conf1->key, sizeof(conf1->key), "serverName");

strcpy\_s(conf1->val, sizeof(conf1->val), "1区");

strcpy\_s(conf2->key, sizeof(conf2->key), "serverId");

strcpy\_s(conf2->val, sizeof(conf2->val), "1000");

vector<config \*> res;

//把两个结构体指从屁股插进去

res.push\_back(conf1);

res.push\_back(conf2);

char \*p = getInfo(res, "serverName");

std::cout << p;

//那么现在我们来进行内存得释放吧

//那么现在我们不能用auto，而是要明确类型了

for (vector<config\*>::iterator iter = res.begin(); iter != res.end(); iter++) {

delete(\*iter);

}

res.clear();

return 0;

}