4.10 数组的替代品

4.10.1 vector 类

1. vector 类 是一种动态数组, 可以在运行阶段设置 vector 对象的长度, 可以在末尾附加新数据, 也可以在中间插入新数据.
    - 调整vector长度时需要使用对应的一些method才可以实现.
2. 它是 new 创建动态数组的替代品 (事实上, vector 类也是使用 new 和 delete 管理内存, 只不过它可以自动完成)

vector 类在使用的时候需要注意的特性:

1. 使用时需要包含头文件 vector: `include <vector>`
2. vector类 也包含在 std 命名空间中
3. vector 需要使用不同的语法来指出它存储的类型:

- `vector<类型> 变量名;`
    - 如: `vector<int> vi;` 和 `vector<double> vd;`
- `vector<类型> 变量名(n);`, 用圆括号()来指定元素个数, 而不是数组那种方括号:
    - n 可以是整型常量, 也可以是整型变量, 这是因为 vector 长度是可变的. (数组的长度必须是常量)
    - 如: `vector<int> vi(100);` 和 `vector<double> vd(200);`

缺点:

- vector 定义的数组效率比较低.

4.10.2 array 类

出现array类的背景:

- 使用固定长度的数组, 但是直接用数组的话不够安全(没有边界检查)
- vector的话执行效率不够高

使用的套路:

1. 需要包含 array 模板类: `#include <array>`
2. 使用格式:
    - `array<typeName, n_elem> arrName;`, 注意 `n_elem` 必须是常量.

- 如: `array<int, 5> ai;` 和 `array<double, 10> ad;`

代码案例:

```
#include <iostream>
#include <vector>
#include <array>

using namespace std;        // array, vector 都需要用到 std 命名空间

int main(void)
{
    double a1[4] = {1.2, 2.4, 3.6, 4.8};

    vector<double> a2(4);           // 注意使用的是圆括号!!
    a2[0] = 1.0 / 3.0;      // 0.333333
    a2[1] = 1.0/ 5.0;       // 0.2
    a2[2] = 1.0 / 7.0;      // 0.142857
    a2[3] = 1.0 / 9.0;      // 0.111111

    array<double, 4> a3 = {3.14, 2.72, 1.62, 1.41};   // 在<>中指明数组的元素个数, array 中必须是常量

    array<double, 4> a4;

    a4 = a3;        // 可以直接赋值 (注意! 传统的数组是不允许直接对拷的!!!)

    cout << "a1[2]: " << a1[2] << " at " << &a1[2] << endl;  // a1 是传统的数组
    cout << "a2[2]: " << a2[2] << " at " << &a2[2] << endl;  // a2 是vector, 注意看它的地址, 开头的一串是和 a1, a3, a4 不同的, 因为 vector在 `堆` 里, 传统数组和 array 都在 `栈` 里
    cout << "a3[2]: " << a3[2] << " at " << &a3[2] << endl;  // a3 是array
    cout << "a4[2]: " << a4[2] << " at " << &a4[2] << endl;  // a4 是array

    // C++ 不检查传统数组的越界问题!!!
    a1[-2] = 20.2;      // -2 是从第0个元素往前找两个元素, 此时已经发生了 `数组越界`, 但是编译器没有报错, 可能只给一个 warning

    cout << "a1[-2]: " << a1[2] << " at " << &a1[-2] << endl;  // a1 是传统的数组
    cout << "a3[2]: " << a3[2] << " at " << &a3[2] << endl;  // a1 是传统的数组
    cout << "a4[2]: " << a4[2] << " at " << &a4[2] << endl;  // a4 是array


    // vector 和 array 也是不禁止越界的行为的, 下面的代码能直接跑
    a2[-2] = 0.000001;
    a3[200] = 3.144444;
    cout << "a2[-2]: " << a2[-2] << " at " << &a2[-2] << endl;     // a2 是vector
    cout << "a3[200]: " << a3[200] << " at " << &a3[200] << endl;  // a3 是array

    // 但是, vector 和 array 有一个 at() 方法, 它会检查越界, 如果越界了, 会抛出一个异常
    a2.at(1) = 2.3;
    cout << "a2.at(1): " << a2.at(1) << " at " << &a2.at(1) << endl;  // a2 是vector

    // a2.at(-1) = 2.3;    // 抛出异常, 越界了

    /*
    array 和 vector 还可以使用 begin() 和 end() 去确定边界  ==> 在16章讲
    */

    return 0;
}
```