C++ STL vector容器访问元素的几种方式

学会如何创建并初始化 vector 容器之后，本节继续来学习如何获取（甚至修改）容器中存储的元素。

访问vector容器中单个元素

首先，vector 容器可以向普通数组那样访问存储的元素，甚至对指定下标处的元素进行修改，比如：

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> values{1,2,3,4,5};
    //获取容器中首个元素
    cout << values[0] << endl;
    //修改容器中下标为 0 的元素的值
    values[0] = values[1] + values[2] + values[3] + values[4];
    cout << values[0] << endl;
    return 0;
}
```

运行结果为：

```
1
14
```

> 显然，vector 的索引从 0 开始，这和普通数组一样。通过使用索引，总是可以访问到 vector 容器中现有的元素。

值得一提的是，`容器名[n]`这种获取元素的方式，需要确保下标 n 的值不会超过容器的容量（可以通过 capacity() 成员函数获取），否则会发生越界访问的错误。幸运的是，和 array 容器一样，vector 容器也提供了 at() 成员函数，当传给 at() 的索引会造成越界时，会抛出`std::out_of_range`异常。

举个例子：

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> values{1,2,3,4,5};
```

```
07.     //获取容器中首个元素
08.     cout << values.at(0) << endl;
09.     //修改容器中下标为 0 的元素的值
10.     values.at(0) = values.at(1) + values.at(2) + values.at(3) + values.at(4);
11.     cout << values.at(0) << endl;
12.     //下面这条语句会发生 out_of_range 异常
13.     //cout << values.at(5) << endl;
14.     return 0;
15. }
```

运行结果为：

```
1
14
```

> 读者可能有这样一个疑问，即为什么 vector 容器在重载 [] 运算符时，没有实现边界检查的功能呢？答案很简单，因为性能。如果每次访问元素，都去检查索引值，无疑会产生很多开销。当不存在越界访问的可能时，就能避免这种开销。

除此之外，vector 容器还提供了 2 个成员函数，即 front() 和 back()，它们分别返回 vector 容器中第一个和最后一个元素的引用，通过利用这 2 个函数返回的引用，可以访问（甚至修改）容器中的首尾元素。

举个例子：

```
01. #include <iostream>
02. #include <vector>
03. using namespace std;
04. int main()
05. {
06.     vector<int> values{1,2,3,4,5};
07.     cout << "values 首元素为：" << values.front() << endl;
08.     cout << "values 尾元素为：" << values.back() << endl;
09.     //修改首元素
10.     values.front() = 10;
11.     cout <<"values 新的首元素为：" << values.front() << endl;
12.     //修改尾元素
13.     values.back() = 20;
14.     cout << "values 新的尾元素为：" << values.back() << endl;
15.     return 0;
16. }
```

输出结果为：

```
values 首元素为：1
values 尾元素为：5
values 新的首元素为：10
values 新的尾元素为：20
```

另外，vector 容器还提供了 data() 成员函数，该函数的功能是返回指向容器中首个元素的指针。通过该指针也可以访问甚至修改容器中的元素。比如：

```
01.  #include <iostream>
02.  #include <vector>
03.  using namespace std;
04.  int main()
05.  {
06.      vector<int> values{1,2,3,4,5};
07.      //输出容器中第 3 个元素的值
08.      cout << *(values.data() + 2) << endl;
09.      //修改容器中第 2 个元素的值
10.      *(values.data() + 1) = 10;
11.      cout << *(values.data() + 1) << endl;
12.      return 0;
13.  }
```

运行结果为：

```
3
10
```

访问vector容器中多个元素

如果想访问 vector 容器中多个元素，可以借助 size() 成员函数，该函数可以返回 vector 容器中实际存储的元素个数。例如：

```
01.  #include <iostream>
02.  #include <vector>
03.  using namespace std;
04.  int main()
05.  {
06.      vector<int> values{1,2,3,4,5};
07.      //从下标 0 一直遍历到 size()-1 处
08.      for (int i = 0; i < values.size(); i++) {
```

```
09.         cout << values[i] << " ";
10.     }
11.     return 0;
12. }
```

运行结果为：

```
1 2 3 4 5
```

注意，这里不要使用 capacity() 成员函数，因为它返回的是 vector 容器的容量，而不是实际存储元素的个数，这两者是有差别的。

> 关于 vector 容器 capacity() 和 size() 的差别，可以阅读《STL vector容量（capacity）和大小（size）的区别》一文。

或者也可以使用基于范围的循环，此方式将会逐个遍历容器中的元素。比如：

```
01. #include <iostream>
02. #include <vector>
03. using namespace std;
04. int main()
05. {
06.     vector<int> values{1,2,3,4,5};
07.     for (auto&& value : values)
08.         cout << value << " ";
09.     return 0;
10. }
```

运行结果为：

```
1 2 3 4 5
```

另外还可以使用 vector 迭代器遍历 vector 容器，这里以 begin()/end() 为例：

```
01. #include <iostream>
02. #include <vector>
03. using namespace std;
04. int main()
05. {
06.     vector<int> values{1,2,3,4,5};
07.     for (auto first = values.begin(); first < values.end(); ++first) {
08.         cout << *first << " ";
09.     }
```

```
10.     return 0;
11. }
```

运行结果为：

```
1 2 3 4 5
```

> 当然，这里也可以使用 rbegin()/rend()、cbegin()/cend()、crbegin()/crend() 以及全局函数 begin()/end()，它们都可以实现对容器中元素的访问。