C++ 容器类 <vector>

简介

C++ 标准库（Standard Template Library, STL）是 C++ 的一个重要组成部分，它提供了一组通用的模板类和函数，用于处理数据集合。<vector> 是 STL 中的一个容器类，用于存储动态大小的数组。

<vector> 是一个序列容器，它允许用户在容器的末尾快速地添加或删除元素。与数组相比，<vector> 提供了更多的功能，如自动调整大小、随机访问等。

语法

在 C++ 中，使用 <vector> 需要包含头文件 <<vector>>。以下是一些基本的语法：

声明一个 vector：

```
std::vector<int> myVector;
```

添加元素：

```
myVector.push_back(10);
```

访问元素：

```
int firstElement = myVector[0];
```

获取元素数量：

```
size_t size = myVector.size();
```

清空 vector：

```
myVector.clear();
```

声明与初始化

<vector> 需要指定元素类型，可通过多种方式进行初始化：

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

int main() {
    std::vector<int> vec1;                  // 空的vector
    std::vector<int> vec2(5);               // 长度为5的vector，元素默认初始化
    std::vector<int> vec3(5, 10);           // 长度为5的vector，元素值为10
    std::vector<int> vec4 = {1, 2, 3, 4};   // 使用初始化列表初始化

    return 0;
}
```

实例

下面是一个使用 <vector> 的简单示例，包括输出结果。

实例

```cpp
#include <iostream>
#include <vector>

int main() {
    // 声明一个存储整数的 vector
    std::vector<int> numbers;

    // 添加元素
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // 输出 vector 中的元素
    std::cout << "Vector contains: ";
    for (int i = 0; i < numbers.size(); ++i) {
        std::cout << numbers[i] << " ";
    }
    std::cout << std::endl;

    // 添加更多元素
    numbers.push_back(40);
    numbers.push_back(50);

    // 再次输出 vector 中的元素
    std::cout << "After adding more elements, vector contains: ";
    for (int i = 0; i < numbers.size(); ++i) {
        std::cout << numbers[i] << " ";
    }
    std::cout << std::endl;

    // 访问特定元素
    std::cout << "The first element is: " << numbers[0] << std::endl;

    // 清空 vector
    numbers.clear();

    // 检查 vector 是否为空
    if (numbers.empty()) {
        std::cout << "The vector is now empty." << std::endl;
    }

    return 0;
}
```

输出结果:

```
Vector contains: 10 20 30
After adding more elements, vector contains: 10 20 30 40 50
The first element is: 10
The vector is now empty.
```

常用成员函数

以下是 <vector> 中的一些常用成员函数：

函数	说明
push_back(const T& val)	在末尾添加元素
pop_back()	删除末尾元素
at(size_t pos)	返回指定位置的元素，带边界检查
operator[]	返回指定位置的元素，不带边界检查
front()	返回第一个元素
back()	返回最后一个元素
data()	返回指向底层数组的指针
size()	返回当前元素数量
capacity()	返回当前分配的容量
reserve(size_t n)	预留至少 n 个元素的存储空间
resize(size_t n)	将元素数量调整为 n
clear()	清空所有元素
insert(iterator pos, val)	在指定位置插入元素
erase(iterator pos)	删除指定位置的元素
begin() / end()	返回起始/结束迭代器

实例

1、基本操作

实例

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

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // 输出所有元素
    std::cout << "Vector elements: ";
    for (int i = 0; i < vec.size(); ++i) {
        std::cout << vec[i] << " ";
    }
    std::cout << std::endl;

    // 获取第一个和最后一个元素
    std::cout << "First element: " << vec.front() << std::endl;
    std::cout << "Last element: " << vec.back() << std::endl;
```

```
    return 0;
}
```

2、动态增加和删除元素

实例

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

int main() {
    std::vector<int> vec;
    vec.push_back(10);
    vec.push_back(20);
    vec.push_back(30);

    std::cout << "Vector size: " << vec.size() << std::endl;
    std::cout << "Vector capacity: " << vec.capacity() << std::endl;

    // 删除最后一个元素
    vec.pop_back();
    std::cout << "After pop_back, size: " << vec.size() << std::endl;

    return 0;
}
```

3、边界检查和安全访问

实例

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

int main() {
    std::vector<int> vec = {1, 2, 3};

    try {
        std::cout << vec.at(2) << std::endl;  // 正常输出
        std::cout << vec.at(5) << std::endl;  // 超出范围，抛出异常
    } catch (const std::out_of_range& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

4、预分配容量

实例

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

int main() {
    std::vector<int> vec;
    vec.reserve(10);  // 预留容量，避免频繁分配内存

    for (int i = 0; i < 10; ++i) {
        vec.push_back(i);
        std::cout << "Capacity after push_back(" << i << "): " << vec.capacity() << std::endl;
    }
```

```
    return 0;
}
```

与其他容器对比

特性	std::vector	std::array	std::list
大小	动态可变	编译时固定	动态可变
存储位置	连续内存	连续内存	非连续内存
访问性能	随机访问快速	随机访问快速	随机访问慢，适合顺序访问
插入和删除性能	末尾操作性能高，其他位置较慢	不支持	任意位置插入和删除较快
内存增长方式	容量不足时成倍增长	无	无

<vector> 是 C++ STL 中一个非常有用的容器，它提供了动态数组的功能，使得元素的添加和删除变得更加灵活和方便。通过上述示例，初学者可以快速了解 <vector> 的基本用法和操作。随着学习的深入，你将发现 <vector> 在实际编程中的强大功能和广泛应用。