第十六讲

Dynamic Array

薛浩

2023 年 5 月 30 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 5：常见数据结构

在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

- 链表
- 动态数组
- 二叉堆
- 二叉搜索树

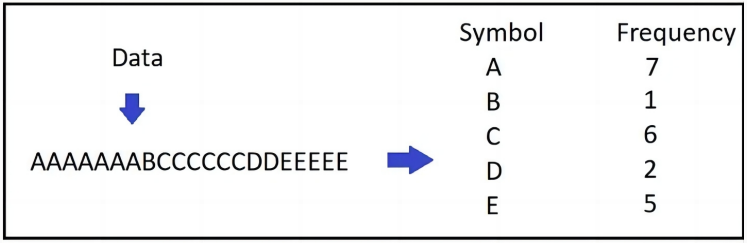

Figure 1: 数据结构和算法

如何使用链表和动态数组创建容器?

目录

复习：数据抽象

List 类

```
class List {
public:
    List();
    ~List();

    List& insert(char ch);
    void clear();
    void print();

private:
    Node* m_head;
};
```

RAII

RAII 是源自 C++ 的一种编程范式，将使用前必须获取的**资源**（Resource）绑定到对象的生命周期上。RAII 保证了任何能够访问对象的函数都可以访问对应的资源，同时还保证了对象生命周期结束时，资源可以正确释放。

该技术充分利用了语言的核心特性（对象生命周期、作用域、对象初始化顺序等）避免资源的泄漏并保证了异常安全。

- 将每个资源封装进类：构造函数获取资源，析构函数释放资源
- 总是通过遵循 RAII 规范的类的对象来使用资源：资源和对象的生命周期绑定

标准库提供的容器基本都是符合 RAII，除此之外，对用户自定义资源还可以使用 std::unique_ptr 等包装器管理动态内存等资源。

练习：基于 List 的 Stack 类

```
class OurStack {
public:
    OurStack();
    ~OurStack();
    int size(); // O(1)
    bool isEmpty(); // O(1)
    void clear(); // O(N)
    void push(std::string value); // O(1)
    std::string pop(); // O(1)
    std::string peek(); // O(1)

private:
    ...
};
```

动态数组

数组的动态分配和释放

对象的动态分配和释放直接使用 new 和 delete：

```
int* p = new int;
delete p;
```

数组的分配和释放需要增加一对中括号，使用 new[] 和 delete[]：

```
int* arr = new int[10];
delete[] arr;
```

动态数组

动态数组（Dynamic Array）可以动态地调整数组的大小，克服了静态数组大小固定的限制。动态数组通过重新分配一个更大的数组，拷贝原来数组中的元素来实现大小的调整。

为了避免多次调整大小增加计算成本，动态数组一般会大幅度调整数组的大小，一般在 1.5 到 2 之间。

对于 n 个元素的数组，当执行 n+1 次存储时，仅最后一次涉及扩容操作。分摊到每个元素，每次插入的时间复杂度依然为常数时间：

$$\frac{n\Theta(1) + \Theta(n)}{n+1} = \Theta(1)$$

一起来写 vector 容器

练习：vector 类

```
class vector {
public:
    vector(size_t capacity = 10);
    ~vector();
    bool empty();
    size_t size();
    size_t capacity();
    int* begin();
    int* end();
    int* insert(int* pos, int value);
    void reserve(size_t n);
    ...
};
```

练习：vector 动态数组

认识一个类，应该先从 private 部分开始，了解其底层表示。

- 使用动态数组作为底层的表示结构
- 记录数组的分配大小和逻辑大小

```
class vector {
    ...

private:
    int* m_elems;
    size_t m_capacity;
    size_t m_size;
};
```

名称空间 namespace

由于 vector 和标准库 std::vector 发生命名冲突，为了避免，使用 namespace 增加了 cs101 名称限定符。在使用时需要添加限定名 cs101::vector。

```
namespace cs101 {
    class vector { ... };
}
```

练习：vector 基本接口

- 标准库默认构造函数不分配数组，此处使用默认参数分配 10 个元素的数组
- 析构函数只需要删除动态数组，注意使用 delete[]
- 其他一些查询函数较为简单，直接处理私有成员

```
vector(size_t capacity = 10);
~vector();

bool empty();
size_t size();
size_t capacity();
```

指针运算 vs 数组索引

指针运算是只对指针进行加减法的运算。数组索引操作本质上等同于指针运算：

```
arr[3]
*(arr + 3)
```

对指针进行加减法运算，编译器将自动计算元素的宽度。假设 `p` 指向数组首地址，`q` 指向数组末地址，以下表达式将得到数组的元素个数。

```
p - q
```

指针也支持自增自减操作：

```
p++
```

练习：vector 迭代器

迭代器（iterator）是指针（pointer）的泛化或抽象，允许 C++ 程序（特别是标准库算法）以统一的方式操作不同的数据结构。为了开发的简便，此处只用了裸指针的别名表示迭代器：

```
class vector {
public:
    using iterator = int*;
```

为了支持 range-base for 循环，需要实现 begin 和 end：

```
class vector {
public:
    // Iterators
    int* begin();
    int* end();
```

练习：vector 动态扩容

动态数组最简单的实现是分配一个足够大的数组，防止溢出；但缺点是灵活性差，空间浪费。

参考标准库，实现 reserve 方法，实现动态内存管理

- 当 n 小于等于当前 capacity 时，不作任何操作
- 当 n 大于当前 capacity 时，增加 capacity 以便可以容纳 n 个元素

参考标准库，实现 insert 方法，注意参数 pos 为迭代器（裸指针）

- 在 pos 指向的位置插入 value 值
- 必要时需要扩容，可以利用 reserve 方法
- 返回 pos 迭代器，以便需要时可以访问插入的元素

如何使用链表和动态数组创建容器?