



D2DS | COURSES | 2024

动手写定长数组Array

Array<T, N>类模板核心实现

@sunrisepeak | 2024.4

• 原生数组 VS Array数组

• Array类模板核心实现

• dslings - Array代码实现

- 基本介绍
 - 原生数组
 - Array数组
- Array核心实现
 - 类型定义 - 固定类型模板参数
 - 数据初始化 - 列表初始化器的使用
 - BigFive - 行为控制
 - 数据访问 - 下标访问运算符重载
 - 常用函数实现 - size/back
 - 迭代器支持 - 范围for
 - 功能扩展 - 负下标访问支持
- 总结

原生数组 VS Array数组



原生数组

- 常退化成指针丢失size信息
- 缺少一些常用功能(整体复制...)

Array数组

- 类型安全 / 携带size信息
- 减少潜在数组越界的风险
- 可以标准化接口, 能做一些特殊检查和扩展
- 性能几乎和原生数组一样

原生数组

```
void arr_init(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = -1;
    }
}

int main() {
    int arr[10];
    arr_init(arr, 10);
    return 0;
}
```

Array数组

```
void arr_init(const Array<int, 10> &arr) {
    for (int i = 0; i < arr.size(); i++) {
        arr[i] = -1;
    }
}

template <typename IntArrayType>
void process(const IntArrayType &arr) {
    for (int &val : arr) {
        val *= 2;
    }
}

int main() {
    Array<int, 10> arr;
    arr_init(arr, 10);
    return 0;
}
```

Vector ?

栈 堆

Array核心实现 - “固定类型”模板参数



在Array的模板参数中的第二个参数上使用N来标识数组长度信息, 需要注意的是这里的N和类型参数T时有差异的, 它是一个 `unsigned int` 类型的**非类型模板参数**, 可以简单视为固定类型(指定类型)的信息标识, 在模板参数中就指定类型。

```
template <typename T, unsigned int N>
class Array {

};
```



Array核心实现 - 数据初始化

数组的列表初始化, 在编程中非常常用和方便对数据做初始化的方式, 如下:

```
d2ds::Array<int, 5> intArr { 5, 4, 3, 2 /*, 1*/ };
```

对于自定义类型的Array模板, 要想支持这个特性可以使用 `initializer_list` 来获取列表中的数据, 并且使用迭代器进行数据的访问, 所以在Array中实现一个支持 `initializer_list` 的构造器即可

```
template <typename T, unsigned int N>
class Array {
public:
    Array(std::initializer_list<T> list) {
        int i = 0;
        for (auto it = list.begin(); it != list.end() && i < N; it++) {
            mData_e[i] = *it;
            i++;
        }
    }

private:
    T mData_e[N == 0 ? 1 : N];
};
```

列表初始化器

N为零情况检查

Array核心实现 - 行为控制 | BigFive



BigFive核心指的是一个类型对象的**拷贝语义**和**移动语义**, 也常被称为**三五原则**(见[cppreference](#))。多数情况下编译器是能自动生成这些代码的, 但作为一个库的话(特别是个数据结构容器库), 往往需要明确每个数据结构的行为, 这对数据结构中的数据复制和移动中的性能优化也是非常有帮助的。我们可以通过下面的类成员来实现其行为控制:

类成员	简述
<code>~ClassName()</code>	析构
<code>ClassName(const ClassName &)</code>	拷贝构造
<code>ClassName & operator=(const ClassName &)</code>	拷贝赋值
<code>ClassName(ClassName &&)</code>	移动构造
<code>ClassName & operator=(ClassName &&)</code>	移动赋值

Array核心实现 - 行为控制 | 拷贝语义



```
d2ds::Array<BigFiveTest::Obj, 5> intArr1;
d2ds::Array<BigFiveTest::Obj, 5> intArr2(intArr1);
```

拷贝构造函数常用于"一个同类型已存在的对象来初始化一个新对象"的场景, 对于Array来说主要实现把已存在对象中的数据进行复制到新对象中即可

```
template <typename T, unsigned int N>
class Array {
public: // bigFive
//...
    Array(const Array &dsObj) {
        for (int i = 0; i < N; i++) {
            mData_e[i] = dsObj.mData_e[i];
        }
    }
//...
};
```

拷贝构造函数

Array核心实现 - 行为控制 | 拷贝语义



```
d2ds::Array<BigFiveTest::Obj, 5> intArr1, intArr2;  
// ...  
intArr1 = intArr2;
```

通过赋值 = 运算符, 把一个Array中的数据复制到另一个数组中

```
template <typename T, unsigned int N>  
class Array {  
public: // bigFive  
// ...  
    Array & operator=(const Array &dsObj) {  
        D2DS_SELF_ASSIGNMENT_CHECKER  
        for (int i = 0; i < N; i++) {  
            mData_e[i] = dsObj.mData_e[i];  
        }  
        return *this;  
    }  
// ...  
};
```

拷贝赋值

Array核心实现 - 行为控制 | 移动语义



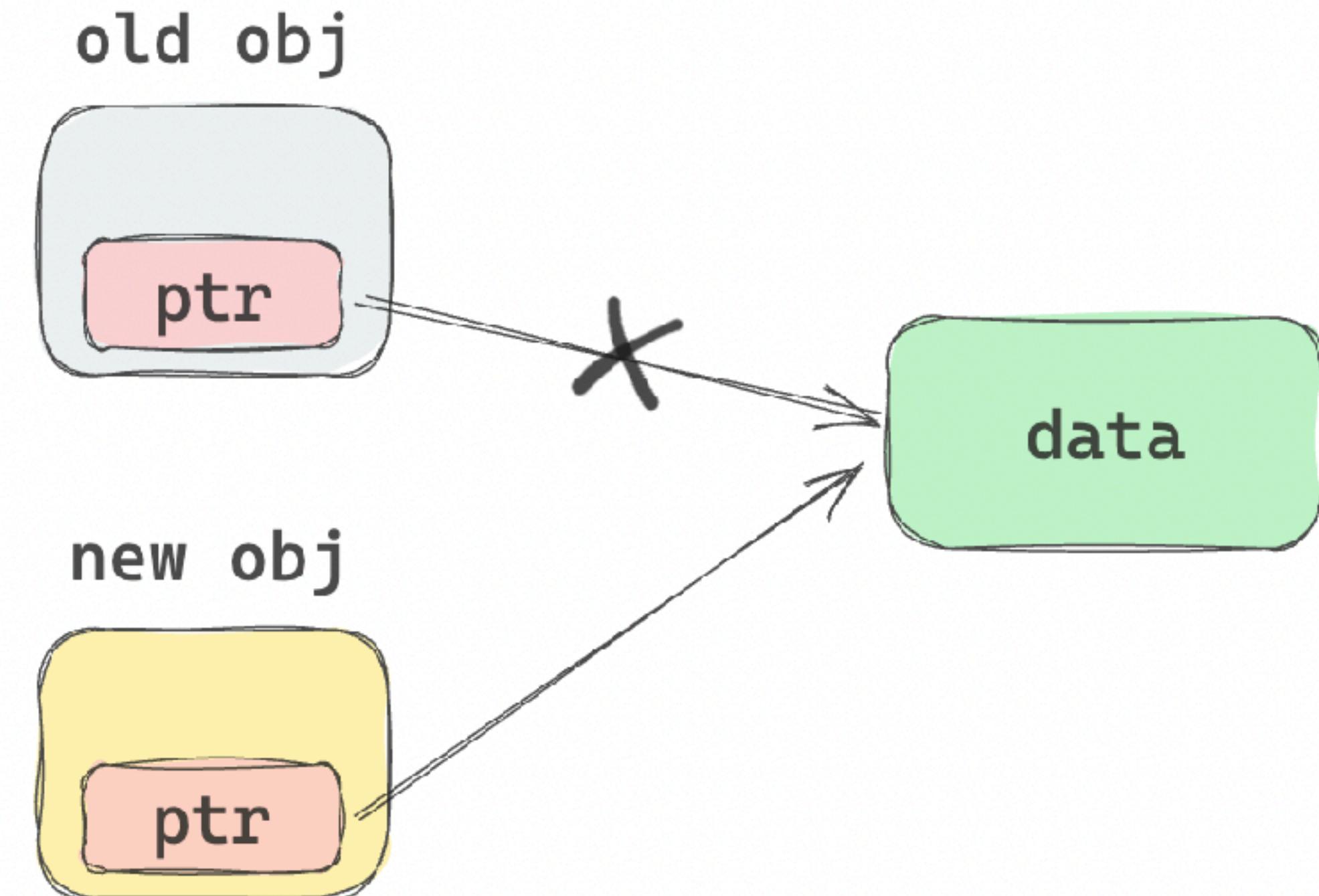
有些场景为了性能会使用移动语义, 只去改变数据的**所有权(ownership)**来避免数据资源的重复复制、频繁分配/释放带来的开销

移动构造

```
d2ds::Array<BigFiveTest::Obj, 5> intArr1;  
//...  
d2ds::Array<BigFiveTest::Obj, 5> intArr2 { std::move(intArr1) };
```



```
template <typename T, unsigned int N>  
class Array {  
public: // bigFive  
//...  
    Array(Array &&ds0bj) {  
        for (int i = 0; i < N; i++) {  
            mData_e[i] = std::move(ds0bj.mData_e[i]);  
        }  
    }  
//...  
};
```



Array核心实现 - 行为控制 | 移动语义



```
d2ds::Array<BigFiveTest::Obj, 5> intArr1, intArr2;  
// ...  
intArr1 = std::move(intArr2);
```

移动赋值和移动构造一样

```
template <typename T, unsigned int N>  
class Array {  
public: // bigFive  
// ...  
    Array & operator=(Array &&dsObj) {  
        D2DS_SELF_ASSIGNMENT_CHECKER  
        for (int i = 0; i < N; i++) {  
            mData_e[i] = std::move(dsObj.mData_e[i]);  
        }  
        return *this;  
    }  
// ...  
};
```

Array核心实现 - 行为控制



Note1: D2DS_SELF_ASSIGNMENT_CHECKER 宏是防止对象自我赋值情况的一个检测。例如:对象myObj赋值给自己(`myObj = myObj;`)。该宏的实现原理(`if (this == &dsObj) return *this;`)是通过地址检查来规避这种情况

Note2: 对于Array的移动语义是不够直观的, 在Vector实现中有更直观的使用, 且更多关于BigFive-拷贝语义和移动语义的介绍将放到C++基础章节

Array核心实现 - 数据访问 | 下标运算符重载



```
intArr[1] = 6;  
intArr[4] = intArr[0];
```

实现类数组的下标访问的核心就是, 在对应类型中实现对下标运算符 [] 的重载

```
template <typename T, unsigned int N>  
class Array {  
public:  
    //...  
    T & operator[](int index) {  
        return mData_e[index];  
    }  
    //...  
};
```

Array核心实现 - 常用函数



```
d2ds::Array<int, 5> intArr { 0, 1, 2, 3, 4 };
for (int i = 0; i < intArr.size(); i++) {
    d2ds_assert_eq(i, intArr[i]);
}
d2ds_assert_eq(4, intArr.back());
```

获取数据结构中的元素数量, 和获取最后一个元素都是常用的功能

还有很多其他常用函数
data/empty...

<https://sunrisepeak.github.io/d2ds-courses>

size

```
template <typename T, unsigned int N>
class Array {
public:
    unsigned int size() const {
        return N;
    }
    //...
};
```

back

```
template <typename T, unsigned int N>
class Array {
public:
    T back() const {
        return mData_e[N != 0 ? N - 1 : 0];
    }
    //...
};
```

Array核心实现 - 迭代器支持



对于Array的迭代器, 由于内部是使用数组来存储数据, 数据是在连续内存空间上的, 可以直接使用类型的指针作为迭代器类型来做迭代器支持和范围for的使用

```
template <typename T, unsigned int N>
class Array {
public:
    int * begin() {
        return mData_e;
    }

    int * end() {
        return mData_e + N;
    }

    //...
};
```

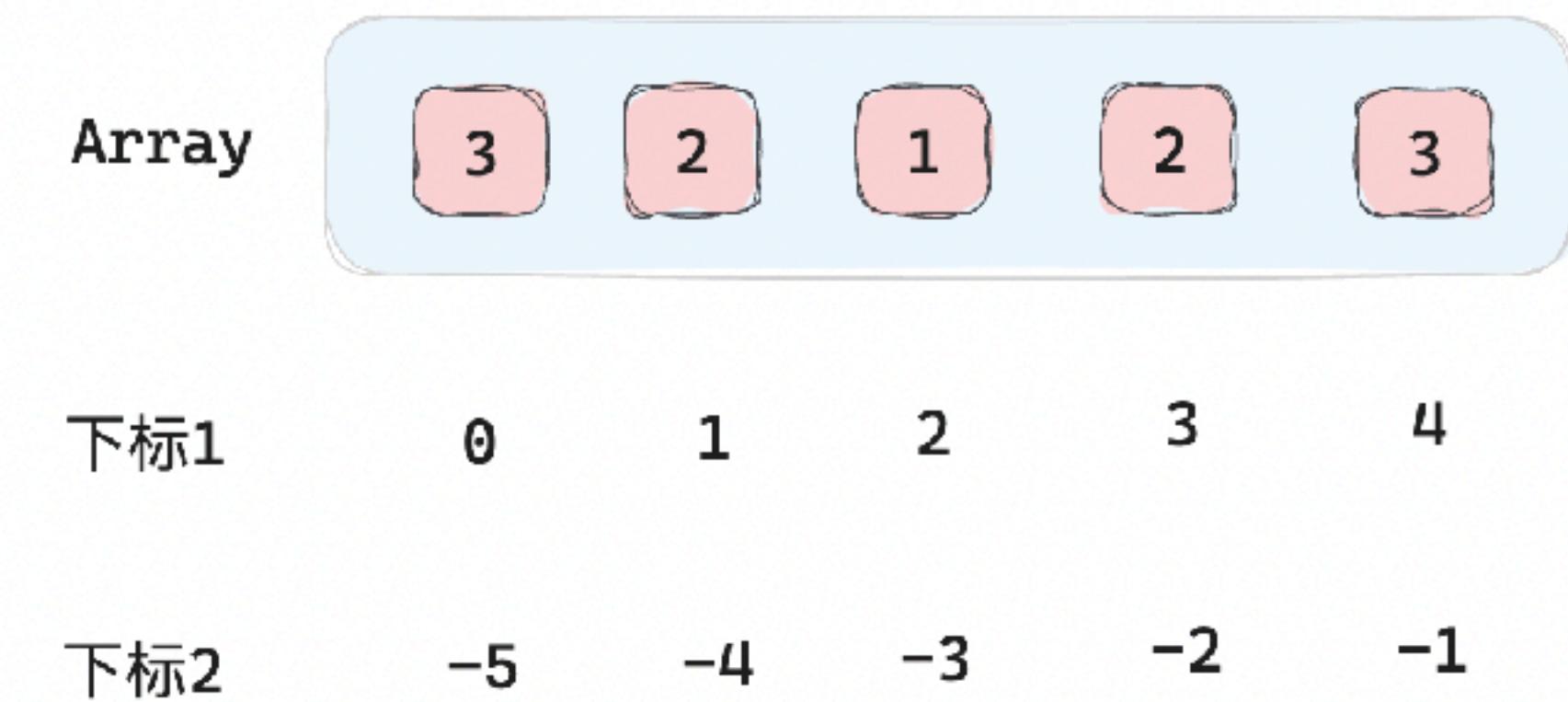
指针天然具备迭代器属性

```
for (int &val : arr) {
    val *= 2;
}
```

Array核心实现 - 数据访问 | 下标运算符重载



```
template <typename T, unsigned int N>
class Array {
public:
//...
    T & operator[](int index) {
        // add start
        if (index < 0)
            index = N + index;
        // add end
        d2ds_assert(index >= 0 && index < N);
        return mData_e[index];
    }
//...
};
```



总结



- 概念 - 类型安全 | 使用和扩展
- Array的核心实现(行为控制、数据访问、迭代器支持...)
- 常用函数size/back 和 功能扩展



dslings - Array代码练习

动手写 `Array<T, N>` 数据结构

@sunrisepeak | 2024.4

<https://sunrisepeak.github.io/d2ds-courses>