第24章 三向比较

《现代C++语言核心特性解析》 谢丙堃

“太空飞船（spaceship）”运算符

- 两个比较的操作数可能产生三种结果

```
bool b = 7 <=> 11 < 0; // b == true
```

- 请注意，运算符<=>的返回值只能与0和自身类型来比较，如果同其他数值比较编译器会报错

```
bool b = 7 <=> 11 < 100; // 编译失败，<=>的结果不能与除0以外的数值比较
```

三向比较的返回类型

- std::strong_ordering类型有三种比较结果
  - std::strong_ordering::less
  - std::strong_ordering::equal
  - std::strong_ordering::greater

三向比较的返回类型

- std::weak_ordering类型也有三种比较结果
  - std::weak_ordering::less
  - std::weak_ordering::equivalent
  - std::weak_ordering::greater

三向比较的返回类型

- std::partial_ordering类型有四种比较结果
  - std::partial_ordering::less
  - std::partial_ordering::equivalent
  - std::partial_ordering::greater
  - std::partial_ordering::unordered

对基础类型的支持

- 对两个算术类型的操作数，对各操作数进行一般算术转换，然后进行比较。其中整型的比较结果为std::strong_ordering，浮点型的比较结果为std::partial_ordering。
- 对于无作用域枚举类型和整型操作数，枚举类型会转换为整型再进行比较，无作用域枚举类型无法与浮点类型比较。

```
enum color {
    red
};

auto r = red <=> 11;   //编译成功
auto r = red <=> 11.1; //编译失败
```

对基础类型的支持

- 对两个相同枚举类型的操作数，比较结果为其底层类型的比较结果，如果枚举类型不同，则无法编译。
- 对于其中一个操作数为bool类型的情况，另外一个操作数必须也是bool类型，否则无法编译。比较结果为std::strong_ordering。
- 不支持作比较的两个操作数为数组的情况，会导致编译出错

```
int arr1[5];
int arr2[5];
auto r = arr1 <=> arr2; // 编译失败
```

对基础类型的支持

- 对于其中一个操作数为指针类型的情况，需要另一个操作数是同样类型的指针，或者是可以转换为相同类型的指针，最终比较结果为std::strong_ordering。

```
char arr1[5];
char arr2[5];
char* ptr = arr2;
auto r = ptr <=> arr1;
```

自动生成的比较运算符函数

- C++20标准规定，当用户为自定义类型声明了三向比较运算符，那么编译器会为其自动生成<、>、<=和>=四种运算符函数

自动生成的比较运算符函数

- 为什么不生成==运算符函数

```
template<typename T>
std::strong_ordering operator<=>(const std::vector<T>& lhs, const std::vector<T> & rhs)
{
    size_t min_size = min(lhs.size(), rhs.size());
    for (size_t i = 0; i != min_size; ++i) {
        if (auto const cmp = std::compare_3way(lhs[i], rhs[i]); cmp != 0) {
            return cmp;
        }
    }
    return lhs.size() <=> rhs.size();
}
```

兼容旧代码

- 用户自定义类型中，实现了<、==运算符函数的数据成员类型，在该类型的三向比较中将自动生成合适的比较代码

```
struct Legacy {
    int n;
    bool operator==(const Legacy& rhs) const {
        return n == rhs.n;
    }
    bool operator<(const Legacy& rhs) const {
        return n < rhs.n;
    }
};
struct ThreeWay {
    Legacy m;
    std::strong_ordering operator<=>(const ThreeWay &) const = default;
};

ThreeWay t1, t2;
bool r = t1 < t2;
```

兼容旧代码

- 自动生成代码

```
struct ThreeWay {
    Legacy m;
    std::strong_ordering operator<=>(const ThreeWay& rhs) const {
        if (m < rhs.m) return std::strong_ordering::less;
        if (m == rhs.m) return std::strong_ordering::equal;
        return std::strong_ordering::greater;
    }
};
```

感谢您的观看

欢迎关注