第14章 强枚举类型

《现代C++语言核心特性解析》 谢丙堃

枚举类型的弊端

- 枚举类型可以隐式转换为整型

```
enum School {
  principal,
  teacher,
  student
};
enum Company {
  chairman,
  manager,
  employee
};

School x = student;
Company y = manager;
bool b = student >= manager;  // 不同类型之间的比较操作
int z = student;              // 隐式转换为int
```

枚举类型的弊端

- 无法指定枚举类型的底层类型

```
enum E {
 e1 = 1,
 e2 = 2,
 e3 = 0xfffffff0
};

int main()
{
 bool b = e1 < e3;
 std::cout << std::boolalpha << b << std::endl;
}
```

使用强枚举类型

- 强枚举类型具备的3个新特性：
  - 枚举标识符属于强枚举类型的作用域。
  - 枚举标识符不会隐式转换为整型。
  - 能指定强枚举类型的底层类型，底层类型默认为int类型。

使用强枚举类型

- 例子

```
enum class HighSchool {
    student,
    teacher,
    headmaster
};
enum class University {
    student,
    professor,
    principal
};

HighSchool x = student;                    // 编译失败，找不到student的定义
bool b = University::student < HighSchool::headmaster;  // 编译失败，比较的类型不同
int y = University::student;               // 编译失败，无法隐式转换为int类型
```

使用强枚举类型

- 例子

```
enum class E : unsigned int {
    e1 = 1,
    e2 = 2,
    e3 = 0xfffffff0
};

int main()
{
    bool b = e1 < e3;
    std::cout << std::boolalpha << b << std::endl;
}
```

列表初始化有底层类型枚举对象

- C++17标准开始，对有底层类型的枚举类型对象可以直接使用列表初始化

```
enum class Color {
  Red,
  Green,
  Blue
};
int main()
{
  Color c{ 5 };    // 编译成功
  Color c1 = 5;    // 编译失败
  Color c2 = { 5 }; // 编译失败
  Color c3(5);     // 编译失败
}
```

使用using打开强枚举类型

- C++20标准扩展让using功能可以打开强枚举类型的命名空间

```
enum class Color {
  Red,
  Green,
  Blue
};
const char* ColorToString(Color c) {
  switch (c) {
  case Color::Red: return "Red";
  case Color::Green: return "Green";
  case Color::Blue: return "Blue";
  default:
    return "none";
  }
}
```

```
enum class Color {
  Red,
  Green,
  Blue
};
const char* ColorToString(Color c) {
  switch (c) {
  using enum Color;
  case Red: return "Red";
  case Green: return "Green";
  case Blue: return "Blue";
  default:
    return "none";
  }
}
```

感谢聆听
欢迎关注