第27章 常量表达式

《现代C++语言核心特性解析》 谢丙堃

常量的不确定性

- 无法保证编译期确定的常量
  - 可以确定的情况

```
const int index0 = 0;
#define index1 1
switch (argc)
{
case  index0:
    std::cout << "index0" << std::endl;
    break;
case index1:
    std::cout << "index1" << std::endl;
    break;
default:
    std::cout << "none" << std::endl;
}
```

```
const int x_size = 5 + 8;
#define y_size 6 + 7
char buffer[x_size][y_size] = { 0 };

enum {
    enum_index0 = index0,
    enum_index1 = index1,
};

std::tuple<int, char> tp = std::make_tuple(4, '3');
int x1 = std::get<index0>(tp);
char x2 = std::get<index1>(tp);
```

常量的不确定性

- 无法保证编译期确定的常量
  - 无法确定的情况

```
int get_index0() {
    return 0;
}
int get_index1() {
    return 1;
}
int get_x_size() {
    return 5 + 8;
}
int get_y_size() {
    return 6 + 7;
}
const int index0 = get_index0();
#define index1 get_index1()
```

常量表达式值

- 一个用constexpr说明符声明的变量或者数据成员，它要求该值必须在编译期计算。

```
constexpr int x = 42;
char buffer[x] = { 0 };
```

- 和const的对比

```
int x1 = 42;
const int x2 = x1;          // 定义和初始化成功

int x1 = 42;
constexpr int x2 = x1;      // 编译失败，x2无法用x1初始化
```

常量表达式函数

- 常量表达式函数的返回值可以在编译阶段就计算出来

```
constexpr int square(int x)
{
    return x * x;
}
```

- 约束条件（C++11标准）：
  - 函数必须返回一个值，所以它的返回值类型不能是void。
  - 函数体必须只有一条语句：return expr，其中expr必须也是一个常量表达式。
  - 函数使用之前必须有定义。
  - 函数必须用constexpr声明。

常量表达式函数

- 约束条件（C++11标准）—— 反例

```
constexpr int next(int x)
{
    return ++x;
}

int g()
{
    return 42;
}
constexpr int f()
{
    return g();
}
```

常量表达式构造函数

- 作用于自定义类型

```
class X {
public:
    X() : x1(5) {}
    int get() const
    {
        return x1;
    }
private:
    int x1;
};

constexpr X x;                      // 编译失败
char buffer[x.get()] = { 0 };       // 编译失败，x.get()无法在编译阶段计算
```

常量表达式构造函数

- 修改代码:

```
class X {
public:
    constexpr X() : x1(5) {}
    constexpr X(int i) : x1(i) {}
    constexpr int get() const
    {
        return x1;
    }
private:
    int x1;
};

constexpr X x;
char buffer[x.get()] = { 0 };
```

常量表达式构造函数

- 规则：
  - 构造函数必须用constexpr修饰。
  - 构造函数初始化列表里必须是常量表达式。
  - 构造函数的函数体必须为空

C++14标准对常量表达式函数的增强

- 新规则：
  - 函数体允许声明变量，除了没有初始化，static和thread_local变量。
  - 函数允许出现if和switch语句，不能使用go语句。
  - 函数允许所有的循环语句，包括for，while，do-while。
  - 函数可以修改生命周期和常量表达式相同的对象。
  - 函数的返回值可以声明为void。
  - constexpr修饰的成员函数不再具有const属性

constexpr lambdas表达式

- 从C++17开始，lambda表达式在条件允许的情况下都会隐式声明为constexpr：

```
constexpr int foo()
{
    return []() { return 58; }();
}

auto get_size = [](int i) { return i * 2; };
char buffer1[foo()] = { 0 };
char buffer2[get_size(5)] = { 0 };
```

if constexpr

- if constexpr的条件必须是编译期能确定结果的常量表达式。
- 条件结果一旦确定，编译器将只编译符合条件的代码块。

```
void check1(int i) {
    if constexpr (i > 0) {
        // 编译失败，不是常量表达式
        std::cout
        << "i > 0"
        << std::endl;
    }
    else {
        std::cout
        << "i <= 0"
        << std::endl;
    }
}
```

```
void check2()
{
    if constexpr (sizeof(int) > sizeof(char)) {
        std::cout
        << "sizeof(int) > sizeof(char)"
        << std::endl;
    }
    else {
        std::cout
        << "sizeof(int) <= sizeof(char)"
        << std::endl;
    }
}
```

允许constexpr虚函数

- C++20标准明确允许在常量表达式中使用虚函数

```
struct X
{
    constexpr virtual int f() const { return 1; }
};

int main() {
    constexpr X x;
    int i = x.f();
}
```

C++20标准对常量表达式的其他修改

- 允许在constexpr中进行平凡的默认初始化
- 允许在constexpr函数中出现Try-catch
- 允许在constexpr中更改联合类型的有效成员
- 允许dynamic_cast和typeid出现在常量表达式中

使用consteval声明立即函数

- 确保函数在编译期就执行计算

```
consteval int sqr(int n) {
  return n*n;
}
constexpr int r = sqr(100);  // 编译成功
int x = 100;
int r2 = sqr(x);  // 编译失败
```

使用constinit检查常量初始化

- 用于具有静态存储持续时间的变量声明上，它要求变量具有常量初始化程序

```
constinit int x = 11;              // 编译成功，全局变量具有静态存储持续
int main() {
  constinit static int y = 42;     // 编译成功，静态变量具有静态存储持续
  constinit int z = 7;             // 编译失败，局部变量是动态分配的
}
```

- 要求变量具有常量初始化程序

```
const char* f() { return "hello"; }
constexpr const char* g() { return "cpp"; }
constinit const char* str1 = f(); // 编译错误，f()不是一个常量初始化程序
constinit const char* str2 = g(); // 编译成功
```

感谢您的观看

欢迎关注