C++ template Metaprograming

templates 用模板 2 个帖子回复 · 4 订阅 · 8 最后更新 · 1天前更新

目录

- 前言
- template metafunction
- type traits
  - 模板可选数据类型参数
  - 使用模型作为 “接口数”
- C++ 库模板元函数的控！
  - 模板别名 “P”
  - 一个
  - 一种类不conditional的控种
  - SFINAE
- C++ 库模板元编程的控之 && integral_constant
  - 关于模板元函数的调用语法
  - is_void
  - 模板元编程 && variable template
- Unevaluated Operands
  - std::declval
  - void_t
  - 重写is_copy_assignable
- 总结

前言

最几天刚看了Cppcon，Walter E.Brown关于metaprogramming的演讲，确实精彩，感觉在模板元编程入门到新手以上，只要学习以过高深。这篇博客主要把演讲里的内容总结一遍，顺带把现代的template metaprogramming的入门教程，编程大家沉默的元编程的“控制心得”，这部分内容会用到一些比较高级的技巧，请注意。

什么是metaprogramming？

- 利用编译期系统进行编程的手段。
- 模板元编程是元编程的一种，利用模板的实例化机制，来实现 图灵完 的计算。

模板元编程之作用：

- 提高代码的灵活性。
- 提高运行期的效率。

模板元编程新标准以前的工作等价编译期语句，这是需要代价的，新编译期语句增加，编译时间就是这个新的代价。

notice：

- 在模板元编程中，常见：你的程序等价于cyclic-free ** compile-time。（没有) mutability ，2) 虚函数，因为虚函数基于运行期，3) RTTI 等等一系列会发生运行期行为的东西。

在模板元编程中：

- 没有可变性。
- 没有可变性。
- 所有东西都不可变！！！（我认为这点是最重要的，所以单列出来）

很多人认为模板元编程很难，其实是因为他们不熟悉，everything都是从不熟悉到熟悉。
在接下来的文章中，没有运行期。

- no run-time
- no run-time
- no run-time（我必须让我自己牢记这点！！！）

template metafunction

```
template <int N>
struct abs {
    static_assert(N != INT_MIN);// 编译期断言 assert
    static int const/enum value = N > 0 ? N : -N;// 代码形式的条件表达式
                                                 // 必须是常数值，否则变成运行期 if 语句
};
```

- 这是一个最简单不过的模板类，也是它全像函数一样工作，所以我们称之为模板元函数。

模板类一个有，而且这个模板可编译期值等动作。

- static_assert是C++11大约2部静态断言，因为NT_MIN的特殊值报错了。
- 这里我们的函数一样，定义了一个static的value变量并赋初值。这实际上是是我们的函数语句。用变量初始化并代替函数语句，注意，这是编译期语句，编译期生成的语句，而非运行期语句。而即使这个变量编译期。

模板元函数调用语法：

```
int const/enum N = 10;
abs n = value;
```

- 在上面，abs是模板元函数名称，n是参数，value是函数返回值。
- so，它像函数一样工作。

编译时可以像函数一样工作，调用其他的模板元函数，那么，yes，它也应该可以调用自己，递归。

```
// 这是一个递归模板类
template <int const/enum N>
// 这个模板元函数的返回值是一个布尔 值，这为后面的 if 模板特化做准备
// 所以这个元函数的返回值类型
struct unsigned_N_unsigned_N
{
    static unsigned const/enum value = gcd<N, N% N>::value;// 递归调用自身gcd
};

template <unsigned N>
struct gcd<N, N> {
    static_assert(N != 0);
    static unsigned const/enum value = N;
};
```

- 我们在主模板中使用两个模板参数，我们默认又为unsigned和non-type模板参数，并用在gcd这个类型定义的计算以，这是好事情 但它实际是，我要把参数value，放于主动并用参数value，方法我已经告诉你了，剩下的就是你的事情了。of course，编译器会帮忙解决剩下事情。
- 正如我们前面的函数语言一样，需要一个初始值，我们利用模板特化(partial specialization)来实现初始条件处理。

C++引入了constexpr关键字，那么与模板元编程相对应的是？

- constexpr还是一个函数，只不过它能发生在编译期间了，还是普通函数的方式调用。
- but，模板元编程能使用的是structures，这也使我们能够多多的工具，而且是pattern的，这在函数是是不可能的，因为函数内部不允许。
- struct可以递归。

- 我们的或的pattern是不能编译期，那么，我们就调用了 模板特化(partial specialization)来实现对他的部分处理。

C++引入了constexpr，那么与模板元编程相对应的是？

- constexpr还是一个函数，只不过它能发生在编译期间了，还是普通函数的方式调用。
- but，模板元编程能使用的是structures，这也使我们能够多多的工具，而且是pattern的，这在函数是是不可能的，因为函数内部不允许。
- struct可以递归。
- 我们的或的pattern是不能编译期，那么，我们就调用了 模板特化(partial specialization)来实现对他的部分处理。
- 这次我们不需要自己来做这一件事，因为我们这里要讲的是模板