Bit Focus

拿去吧, 四字节整数类型

Posted at 2011-03-26 09:08:36 | Updated at 2022-02-08 05:31:31

对于 C(++) 之流在平台相关这种水深火热之中挣扎的语言, 对机器字长, 以及整数类型字长在一些情况下是极其敏感的. 所以每个大型项目建设之前, 在某个头文件里面定义一大堆 `int32` 或者是 `unsigned8_t` 之类的固定字长类型显得非常必要, 而且让工程看起来很专业的样子. 然而, 由于标准库中没有, 项目之间又互相不信任, 结果是几乎每个库都会自定一套, 比如 boost 就有 `int8_t`, `int_least8_t` 等类型, 而 Qt 更是拿出了叫做 `qulonglong` 的类型来卖萌. 虽然说是跨平台, 但是如果迁移平台时使用了错误版本的库呢? 因此有时依然需要写一个类似下面的程序来人肉验证

```
int main()
{
    std::cout << sizeof(int4_type) << " "
              << sizeof(int8_type) << std::endl;
    return 0;
}
```

然而, 人都是不可靠的, 如果有人对你说, 喔, 我跑了上面的程序, 结果是 "4 8", 也许你仍会强迫症一样, 自己再验证一次. 好了, 也许你已经厌烦了, 这么一点破事情, 难道不能交给机器做么?

理想的解决方案是, 只需要这样一个模板就好了

```
struct IWantAnIntegralTypeOf;

int main()
{
    IWantAnIntegralTypeOf<4> this_is_a_4_bytes_int;
    IWantAnIntegralTypeOf<8> this_is_a_8_bytes_int;
    return 0;
}
```

不是吗?

这是完全可行的, 实际上, 可以把 C 支持的所有内建整数类型先拿出来, 组成一个类型链表来搜索, 像下面这样

```
struct __null__ {};

struct c_char {
    typedef char type;
    typedef __null__ next;
};

struct c_short {
    typedef short type;
    typedef c_char next;
};

struct c_int {
    typedef int type;
    typedef c_short next;
};
```

```
    typedef long type;
    typedef c_int next;
};

struct c_long_long {
    typedef long long type;
    typedef c_long next;
};

struct __head__ {
    typedef c_long_long next;
};
```

接下来, 弄个模板来搜, 搜索当然是用模板偏特化的方法

```
template <typename _TypeNode, int _SizeOf, bool _SizeMatched>
struct type_finder;

template <typename _TypeNode, int _SizeOf>
struct type_finder<_TypeNode, _SizeOf, true>
{
    typedef typename _TypeNode::type type;
};

template <typename _TypeNode, int _SizeOf>
struct type_finder<_TypeNode, _SizeOf, false>
{
    typedef
        typename type_finder<
                typename _TypeNode::next
              , _SizeOf
```

```
};
```

搜索入口则类似

```
template <int _SizeOf>
struct gimme_the_type {
    typedef typename type_finder<__head__, _SizeOf, false>::type type;
};
```

再来调一下最开始想要的那个模板

```
template <int _SizeOf>
struct IWantAnIntegralTypeOf {
    typedef typename gimme_the_type<_SizeOf>::type;

    operator type& ()
    {
        return value;
    }

    operator type const& () const
    {
        return value;
    }

    IWantAnIntegralTypeOf(type init_value = 0)
        : value(init_value)
    {}
private:
    type value;
};
```

大功告成. 整理一下代码, 下面是一个完整的例子

```
struct __this_platform_does_not_support_such_a_type__ {
    typedef void type;
};

struct c_char {
    typedef char type;
    typedef __this_platform_does_not_support_such_a_type__ next;
};

struct c_short {
    typedef short type;
    typedef c_char next;
};

struct c_int {
    typedef int type;
    typedef c_short next;
};

struct c_long {
    typedef long type;
    typedef c_int next;
};

struct c_long_long {
    typedef long long type;
    typedef c_long next;
};

template <typename _TypeNode, int _SizeOf, bool _SizeMatched>
```

```
template <typename _TypeNode, int _SizeOf>
struct type_finder<_TypeNode, _SizeOf, true>
{
    typedef typename _TypeNode::type type;
};

template <typename _TypeNode, int _SizeOf>
struct type_finder<_TypeNode, _SizeOf, false>
{
    typedef
        typename type_finder<
                typename _TypeNode::next
              , _SizeOf
              , _SizeOf == sizeof(typename _TypeNode::next::type)>::type type;
};

template <int _SizeOf>
class IWantAnIntegralTypeOf {
    struct __head__ {
        typedef c_long_long next;
    };

    typedef typename type_finder<__head__, _SizeOf, false>::type type;

    type value;
public:
    operator type& ()
    {
        return value;
    }
```

```
    operator type const& () const
    {
        return value;
    }

    IWantAnIntegralTypeOf(type init_value = 0)
        : value(init_value)
    {}
};

int main()
{
    IWantAnIntegralTypeOf<2> i2(0x7fff);
    std::cout << i2 << std::endl;
    i2 += 1;
    std::cout << i2 << std::endl;
    return 0;
}
```

好了, 拿去吧, 你要的那么多字节的整数类型.

Post tags: Generic Programming Metaprogramming C++ Template

Comments:

Kavin Yao <^> said, at 2013-11-30 19:36:52.431090 (UTC)

如果没理解错的话，每次使用IWantAnIntegralTypeOf都有隐式类型转换，这么做的开销如何?

Neuron Teckid said, at 2013-12-24 15:20:47.669240 (UTC)

如果不是不同类型混着赋值, 使用常数对变量赋初始值不会有开销 (编译器应该会优化这部分代码).