

关于C++模板的类型转换的讨论

目录

- 讨论地址
- 原问题
- 我的解答
 - 首先看ff的情况。
 - 再来看f的情况。

这两天在饮水思源的C板，关于C++模板的类型转换的一个讨论，后面是我的解答。

讨论地址

<http://bbs.sjtu.edu.cn/bbstcon,board,C,reid,1330078933,file,M.13>

原问题

今天在书上看到模板演绎的时候可以允许cast-down，于是我写了个东西：

```
1  template <bool _Test, class _Type = void>
2  struct enable_if { };
3
4  template<class _Type>
5  struct enable_if<true, _Type> {
6      typedef _Type type;
7  };
8
9  class A { };
10 class B : A { };
11
12 template <typename T>
13 struct traits { static int const value = false;
14 };
15
16 template <>
17 struct traits<A> { static int const value = true;
18 };
19
20 template <typename T>
21 void f(T, typename enable_if<traits<T>::value>::type* = 0) { }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

```
26 template <typename T>
27 class BB {};
28
29 template <typename T>
30 class DD : public BB<T> {};
31
32 template <typename T> void ff(BB<T>) {};
33
34 int main(int argc, char * argv[])
35 {
36     A a; B b;
37     DD<long> dd;
38     //f(b);
39     ff(dd);
40 }
```

奇怪的是重载决议的时候，`f` 的情况下它就不让我特化的 `f<A>` 进来。

但是在 `ff` 的情况下，`ff<BB<long>>` 却进来了。

在VC10和GCC3.4下测试

我的解答

我们来设身处地地作为编译器，看一遍到底发生了什么。

约定符号 `#`：`A#B` 是把 `B` 带入 `A<T>` 的参数 `T` 之后实例化得到的结果。

首先看ff的情况。

```
1 DD<long> dd;
```

处理到这句的时候，编译器看到了 `DD<long>` 的实例化，于是去实例化 `DD#long`，继而实例化了 `BB#long`。

```
1 ff(dd);
```

这句，首先计算重载函数集合。

第一步，需要从参数 `DD#long -> BB<T>` 推断 `ff<T>` 的 `T`。根据函数模板参数推断规则：

```
:code: `class_template_name<T>` 类型的参数，可以用于推断 :code: `T` 。
```

于是编译器推断 `T` 为 `long`。这里就算不是 `BB` 而是完全无关的 `CC` 都可以推断成功，只要 `CC` 也是一个 `CC<T>` 形式的模板。

第二步，模板特化匹配。因为只有一个模板，所以匹配了最泛化的 `ff<T>`。

第三步，模板实例化。

推断了 `long -> T` 之后，编译器实例化 `ff#long`。

重载函数集合：`{ff#long}`

然后重载抉择找到唯一的可匹配的实例 `ff#long`，检查实际参数 `DD#long` 可以隐式转换到形式参数 `BB#long`，从而生成了这次函数调用。

再看f的情况。

```
1 f(b);
```

计算候选重载函数集合。

第一步，对所有 `f` 模板推断实参。根据函数模板参数推断规则：

带有 `:code:`T`` 类型的参数，可以用于推断 `:code:`T`` 。

于是 `B -> T` 被推断出来了。

第二步，模板特化匹配。

这里 `B` 不是 `A`，所以不能用 `f<A>` 特化，只能用 `f<T>` 模板。

第三步，模板实例化。

`B` 带入 `f<T>` 实例化成 `f#B` 的过程中，实例化 `traits#B`。

由于没有针对 `B` 的特化，所以用 `traits<T>` 模板，
`traits#B::value=false`，进而 `enable_if#false` 没有 `type`，
出错。

唯一的模板匹配出错，重载函数集合为空，SFINAE原则不能找到合适的匹配，于是报错。