C++11完美转发及实现方法详解

C++11 标准为 C++ 引入右值引用语法的同时，还解决了一个 C++ 98/03 标准长期存在的短板，即使用简单的方式即可在函数模板中实现参数的完美转发。那么，什么是完美转发？它为什么是 C++98/03 标准存在的一个短板？C++11 标准又是如何为 C++ 弥补这一短板的？别急，本节将就这些问题给读者做一一讲解。

首先解释一下什么是完美转发，它指的是函数模板可以将自己的参数“完美”地转发给内部调用的其它函数。所谓完美，即不仅能准确地转发参数的值，还能保证被转发参数的左、右值属性不变。

> 在 C++ 中，一个表达式不是左值就是右值。有关如何判断一个表达式是左值还是右值，可阅读《C++右值引用》一文做详细了解。

举个例子：

```
01.  template<typename T>
02.  void function(T t) {
03.      otherdef(t);
04.  }
```

如上所示，function() 函数模板中调用了 otherdef() 函数。在此基础上，完美转发指的是：如果 function() 函数接收到的参数 t 为左值，那么该函数传递给 otherdef() 的参数 t 也是左值；反之如果 function() 函数接收到的参数 t 为右值，那么传递给 otherdef() 函数的参数 t 也必须为右值。

显然，function() 函数模板并没有实现完美转发。一方面，参数 t 为非引用类型，这意味着在调用 function() 函数时，实参将值传递给形参的过程就需要额外进行一次拷贝操作；另一方面，无论调用 function() 函数模板时传递给参数 t 的是左值还是右值，对于函数内部的参数 t 来说，它有自己的名称，也可以获取它的存储地址，因此它永远都是左值，也就是说，传递给 otherdef() 函数的参数 t 永远都是左值。总之，无论从那个角度看，function() 函数的定义都不“完美”。

> 读者可能会问，完美转发这样严苛的参数传递机制，很常用吗？C++98/03 标准中几乎不会用到，但 C++11 标准为 C++ 引入了右值引用和移动语义，因此很多场景中是否实现完美转发，直接决定了该参数的传递过程使用的是拷贝语义（调用拷贝构造函数）还是移动语义（调用移动构造函数）。

事实上，C++98/03 标准下的 C++ 也可以实现完美转发，只是实现方式比较笨拙。通过前面的学习我们知道，C++ 98/03 标准中只有左值引用，并且可以细分为非 const 引用和 const 引用。其中，使用非 const 引用作为函数模板参数时，只能接收左值，无法接收右值；而 const 左值引用既可以接收左值，也可以接收右值，但考虑到其 const 属性，除非被调用函数的参数也是 const 属性，否则将无法直接传递。

这也就意味着，单独使用任何一种引用形式，可以实现转发，但无法保证完美。因此如果使用 C++ 98/03 标准下的 C++ 语言，我们可以采用函数模板重载的方式实现完美转发，例如：

```
01.  #include <iostream>
02.  using namespace std;
03.
04.  //重载被调用函数，查看完美转发的效果
05.  void otherdef(int & t) {
06.      cout << "lvalue\n";
07.  }
08.  void otherdef(const int & t) {
09.      cout << "rvalue\n";
10.  }
11.
12.  //重载函数模板，分别接收左值和右值
13.  //接收右值参数
```

```
14.  template <typename T>
15.  void function(const T& t) {
16.      otherdef(t);
17.  }
18.  //接收左值参数
19.  template <typename T>
20.  void function(T& t) {
21.      otherdef(t);
22.  }
23.
24.  int main()
25.  {
26.      function(5);//5 是右值
27.      int  x = 1;
28.      function(x);//x 是左值
29.      return 0;
30.  }
```

程序执行结果为：

```
rvalue
lvalue
```

从输出结果中可以看到，对于右值 5 来说，它实际调用的参数类型为 const T& 的函数模板，由于 t 为 const 类型，所以 otherdef() 函数实际调用的也是参数用 const 修饰的函数，所以输出“rvalue”；对于左值 x 来说，2 个重载模板函数都适用，C++编译器会选择最适合的参数类型为 T& 的函数模板，进而 therdef() 函数实际调用的是参数类型为非 const 的函数，输出“lvalue”。

显然，使用重载的模板函数实现完美转发也是有弊端的，此实现方式仅适用于模板函数仅有少量参数的情况，否则就需要编写大量的重载函数模板，造成代码的冗余。为了方便用户更快速地实现完美转发，C++ 11 标准中允许在函数模板中使用右值引用来实现完美转发。

C++11 标准中规定，通常情况下右值引用形式的参数只能接收右值，不能接收左值。但对于函数模板中使用右值引用语法定义的参数

来说，它不再遵守这一规定，既可以接收右值，也可以接收左值（此时的右值引用又被称为“万能引用”）。

仍以 function() 函数为例，在 C++11 标准中实现完美转发，只需要编写如下一个模板函数即可：

```
template <typename T>
void function(T&& t) {
    otherdef(t);
}
```

此模板函数的参数 t 既可以接收左值，也可以接收右值。但仅仅使用右值引用作为函数模板的参数是远远不够的，还有一个问题继续解决，即如果调用 function() 函数时为其传递一个左值引用或者右值引用的实参，如下所示：

```
int n = 10;
int & num = n;
function(num); // T 为 int&

int && num2 = 11;
function(num2); // T 为 int &&
```

其中，由 function(num) 实例化的函数底层就变成了 function(int & & t)，同样由 function(num2) 实例化的函数底层则变成了 function(int && && t)。要知道，C++98/03 标准是不支持这种用法的，而 C++ 11标准为了更好地实现完美转发，特意为其指定了新的类型匹配规则，又称为引用折叠规则（假设用 A 表示实际传递参数的类型）：

- 当实参为左值或者左值引用（A&）时，函数模板中 T&& 将转变为 A&（A& && = A&）；
- 当实参为右值或者右值引用（A&&）时，函数模板中 T&& 将转变为 A&&（A&& && = A&&）。

> 读者只需要知道，在实现完美转发时，只要函数模板的参数类型为 T&&，则 C++ 可以自行准确地判定出实际传入的实参是左值还是右值。

通过将函数模板的形参类型设置为 T&&，我们可以很好地解决接收左、右值的问题。但除此之外，还需要解决一个问题，即无论传入的形参是左值还是右值，对于函数模板内部来说，形参既有名称又能寻址，因此它都是左值。那么如何才能将函数模板接收到的形参连同其左、右值属性，一起传递给被调用的函数呢？

C++11 标准的开发者已经帮我们想好的解决方案，该新标准还引入了一个模板函数 forword<T>()，我们只需要调用该函数，就可以很方便地解决此问题。仍以 function 模板函数为例，如下演示了该函数模板的用法：

```
#include <iostream>
using namespace std;

//重载被调用函数，查看完美转发的效果
void otherdef(int & t) {
    cout << "lvalue\n";
}
void otherdef(const int & t) {
    cout << "rvalue\n";
}

//实现完美转发的函数模板
template <typename T>
void function(T&& t) {
    otherdef(forward<T>(t));
}

int main()
{
    function(5);
    int  x = 1;
    function(x);
    return 0;
}
```

程序执行结果为：

```
rvalue
lvalue
```

注意程序中第 12~16 行，此 function() 模板函数才是实现完美转发的最终版本。可以看到，forword() 函数模板用于修饰被调用函数中需要维持参数左、右值属性的参数。

总的来说，在定义模板函数时，我们采用右值引用的语法格式定义参数类型，由此该函数既可以接收外界传入的左值，也可以接收右值；其次，还需要使用 C++11 标准库提供的 forword() 模板函数修饰被调用函数中需要维持左、右值属性的参数。由此即可轻松实现函数模板中参数的完美转发。

优秀文章

C语言创建线程thread_create（）
Spark Streaming简介
MATLAB R2016b的目录结构
PHP数据类型转换
MyBatis和Hibernate的区别
Linux使用注意事项（新手必看）
JSP application.getAttributeNames()方法：获取所有的属
jQuery removeClass()方法删除class
Spring MVC重定向和转发
MongoDB覆盖索引查询