C++完美转发实现原理：万能引用、引用折叠

1、概念：

1.1、万能引用(Universal Reference)：

使用 `T&&` 类型的形参既能绑定右值，又能绑定左值，只有发生类型推导的时候，`T&&` 才表示万能引用；否则，表示右值引用。
即**万能引用只存在模板中**，模板中的 `&&` 不代表右值引用，而是万能引用，其既能接收左值又能接收右值。

1.2、左/右值引用变量的值类型都是左值：

引用类型的唯一作用就是限制了接收的类型，后续使用中都退化成了左值；引用的值类型和变量类型不一样，左/右值引用变量的**值类型都是左值**，而不是左值引用或者右值引用。

例一：

```
void func2(int&& val) { cout << "右值" << endl; }  // 右值引用函数
void func2(int& val) { cout << "左值" << endl; }  // 左值引用函数
void func1(int&& val)
{
    func2(val);  // 此时val是左值，调用第二行的函数，输出“左值”
}
```

1.3、引用折叠：

所有的折叠引用最终都代表一个引用，要么是左值引用，要么是右值引用。规则是：如果任一引用(形参的引用类型与传入的实参引用类型)为左值引用，则结果为左值引用。否则（即两个都是右值引用），结果为右值引用。

1.4、完美转发(perfect forwarding)：

是指在函数模板中，完全依照模板参数的类型，将参数传递给函数模板中调用的另一个函数。
模板的万能引用只是提供了能够接收同时接收左值引用和右值引用的能力，但是引用类型的唯一作用就是限制了接收的类型，后续使用中都退化成了左值，我们希望能够在传递(转发)过程中保持它的左值或者右值的属性，即执行例二后输出右值。

目的： 数据是左值就转发成左值，右值就转发成右值。

例二：

```
void func2(int&& val) { cout << "右值" << endl; }  // 右值引用函数
void func1(int&& val)
{
    func2(std::forward<int>(val));  // val按照右值传递(转发)给func2函数，本质为类型转换为右值
}
```

2、完美转发实现原理：

```
// 完美转发原型：
T&& forward(T&& t) { return static_cast<T&&>(t); }

// 用法：    template<typename T>
void func1(T && val) { func2(std::forward<T>(val)); }

// 当传入左值引用
void func1(T& && val) { func2(static_cast<T& &&>(val)); }
// 引用折叠后：
void func1(T& val) { func2(static_cast<T&>(val)); }

// 当传入右值引用
void func1(T&& && val) { func2(static_cast<T&& &&>(val)); }
// 引用折叠后：
void func1(T&& val) { func2(static_cast<T&&>(val)); }
```

即：传入的左值，传递给调用函数仍然为左值，传入的右值，传递给调用函数的仍为右值，不会退化为左值。

总结：

1、完美转发实现了参数在函数调用传递(转发)过程中保持它的左值或者右值的属性不变。

参考资料：

1、C++中的万能引用和完美转发
2、C++: 左值引用(&), 右值引用(&&),万能引用(template &&)详解 与 完美转发(forward) 实现剖析