std：：move不会移动任何东西，而完美的转发是不完美的。移动操作并不总是比复制更便宜;即使是这样，也不总是像您期望的那样便宜;并且它们并不总是在移动有效的上下文中调用。构造“type&&”并不总是表示右值引用。

std::move 并不移动任何东西，std::forward 也不转发任何东西。

std::move 和 std::forward 仅仅是执行类型转换的函数（实际上是函数模板）。std::move 会无条件地将其参数转换为右值，而 std::forward 仅在特定条件满足时才执行这种转换。仅此而已。

std::move 的主要作用就是将传入的参数转换为右值引用，使得该对象可以被当作右值处理，进而可能触发移动构造函数或移动赋值运算符，实现资源的高效转移，但它本身并不执行实际的移动操作。

它能做类型转换，但不会进行移动操作。

右值引用仅仅是一种引用类型。C++17结构体绑定，右值初始化和结构初始化（一个为窃取，一个窃取）。

std::move 时要注意对象是否为 const 类型，避免产生意外的复制操作。

std::forward 是一个条件转换，常用于函数模板中，将通用引用参数转发给其他函数。在 logAndProcess 函数模板中：

* 当调用 logAndProcess(w) 传递左值时，std::forward<T>(param) 会将 param 作为左值转发给 process 函数，从而调用 process(const Widget& lvalArg)。
* 当调用 logAndProcess(std::move(w)) 传递右值时，std::forward<T>(param) 会将 param 作为右值转发给 process 函数，从而调用 process(Widget&& rvalArg)。

函数模版中，参数总是被当做左值处理，因此需要完美转发判断，参数是否由右值初始化得来，如果是右值得来的，那么转换为右值

通用引用的最大特点是其灵活性，它可以根据绑定的对象是左值还是右值，自动调整为左值引用或右值引用。这种特性使得通用引用在实现完美转发等功能时非常有用。例如 std::forward 就是利用通用引用的特性，实现了将参数以原始的左值或右值属性转发给其他函数的功能。**auto 声明变量。模板函数参数、**

**不要将std：：move或std：：forward应用于本地对象，如果它们有资格进行返回值优化。**

**完美转发由于精确匹配度的问题，在重载覆盖继承等方面，可能会有问题。标记转发**

**完美转发也可能失败，花括号传递参数，参数类型推导不出来。**

**0和null指针。**

**常量也不可。不可推导。**

**重载的函数名和模板名**