在模板 processVals 的声明语句中，可以看到我使用的是 typename 来声明模板中的型别形参，这仅仅是我的个人偏好。此处使用关键字 class 也同样可以。但当我演示 C++ 标准中的代码片断时，我会使用 class 来声明型别形参，因为标准中采用的就是那样的做法。

若某对象是依据同一型别的另一对象初始化出来的，则该新对象称为提供初始化依据的对象的一个**副本**，即使该副本是由移动构造函数创建的。这样称呼情有可原，因为 C++ 中并无术语用以区分某对象到底是经由复制构造函数创建的副本，还是经由移动构造函数创建的副本。[译注 2]

```
void someFunc(Widget w);        // someFunc 的形参 w 按值传递

Widget wid;                     // wid 是 Widget 型别的某个对象

someFunc(wid);                  // 在这个对 someFunc 的调用中，
                                // w 是 wid 经由复制构造函数创建的副本

someFunc(std::move(wid));       // 在这个对 someFunc 的调用中，
                                // w 是 wid 经由移动构造函数创建的副本
```

右值的副本通常经由移动构造函数创建，而左值的副本通常经由复制构造函数创建。这也就是说，如果你仅仅了解到某个对象是另一对象的副本，则还不能判断构造这个副本要花费多少成本。例如，在上述代码中，如果不知道传入 someFunc 的究竟是右值还是左值，就无法计算创建形参 w 所花费的成本（你还需要知道移动或复制 Widget 型别对象的具体成本）。

在函数调用中，调用方的表达式，称为函数的实参。实参的用处，是初始化函数的形参。在上面 someFunc 的第一次调用中，实参是 wid。而在第二次调用中，实参则是 std::move(wid)。在两次调用中，形参都是 w。实参和形参有着重大的区别，因为形参都是左值，而用来作为其初始化依据的实参，则既可能是右值，也可能是左值。这一点在完美转发（perfect forwarding）的过程中尤其关系重大，在这样的一个过程中，传递给某个函数的实参会被传递给另一函数，并保持其右值性（rvalueness）或左值性（lvalueness）（完美转发会在条款 30 中详细讨论）。

设计良好的函数都是**异常安全**的，这意味着它们至少会提供基本异常安全保证（即**基本保证**）。提供了基本保证的函数能够向调用者确保即使有异常抛出，程序的不变量不会受到影响（即不会有数据结构被破坏），且不会发生资源泄漏。而提供了强异常安全保证（即**强保证**）的函数则能够向调用者确保即使有异常抛出，程序状态会在调用前后保持不变。

译注 2：在原文中，“副本”和“复制构造函数”中的“复制”是同一个词“copy”。