交互测试有时称为模拟（mocking，*https://oreil.ly/IfMoR*）。在本章中，我们避免使用这个术语，因为它可能会与模拟框架混淆，模拟框架既可用于打桩，也可用于交互测试。

正如本章后面所讨论的，交互测试在某些情况下是有用的，但是在可能的情况下应该避免使用，因为过度使用很容易导致脆弱测试。

## 实际实现

尽管测试替身是非常宝贵的测试工具，但是我们对测试的第一选择是在有测试依赖项时，直接使用系统的实际实现；也就是说，与生产代码中使用的实现相同。在测试中使用与生产环境中相同的代码时，具有更高的保真度，使用实际实现有助于实现这一点。

在谷歌，对实际实现的偏好随着时间的推移而增强，正如我们所看到的，过度使用模拟框架有一种倾向，即用与真实实现不同步的重复代码污染测试，从而使重构变得困难。我们将在本章后面更详细地讨论这个主题。

倾向于实际实现的测试被称为经典测试（*https://oreil.ly/OWw7h*）。还有一种称为模拟（mockist）测试的测试风格，在这种风格中，首选的是使用模拟框架，而不是实际实现。尽管软件行业的一些人[包括第一个模拟框架（*https://oreil.ly/AuqwY7*）的创建者]习惯于模拟测试，但在谷歌我们发现这种模拟测试方法很难扩展。它要求工程师在设计被测系统时遵循严格的指导方针（*http://jmock.org/oopsla2004.pdf*），而谷歌大多数工程师的默认行为是以更适合经典测试风格的方式编写代码。

### 实际实现比隔离更好

考虑到这些实际实现中的所有代码都将在测试中执行，使用依赖项的实际实现让被测系统更加真实。相反，使用测试替身的测试将被测系统与其依赖项隔离开来，这样测试就不会在被测系统的依赖项中执行代码。

我们更喜欢使用实际实现的测试，因为它们能给我们更多的信心，让我们相信被测系统工作正常。如果单元测试过多地依赖于测试替身，工程师可能需要运行集成测试或手动验证其功能是否按预期工作，以便获得相同的可信度。执行这些额外的任