类必须实现不变式（Classes Should Enforce Invariants）

Bjarne Stroustrup: 我的基本原则式真正的类必须有一个接口，有一个隐含的不变式（invariant）

Bill Venners: 不变式（invariant）是什么东西？

Bjarne Stroustrup: 什么使一个对象（object）有效？是不变式（invariant）。我以vector举例说明。vector知道自己有n个元素（element），vector也知道自己有一个指针指向这些元素。以上两点就是不变式（invariant）。如果vector实际上竟然有n+1个元素，就出问题了。如果vector包含的指针为0，也表示有bug。所以在定义一个类前，你必须明确什么是不变式（invariant），类的接口必须体现了不变式（invariant）。如果类的成员函数（member function）不能体现不变式，也许这个函数应该放到类的外面比较好。牢记类的不变式，你就能得到一个简洁而小的接口，易于理解和维护。

Bill Venners: 不变式（invariant）是建立类的必要条件？因为类的责任就是维护不变式（invariant）？

Bjarne Stroustrup: 说得对。

Bill Venners: 不变式就是类中的各数据间的关系。

Bjarne Stroustrup: 对。如果任何数据都可以随意定义自己的值，就没有必要创建类。使用结构体（structure）就行了。考虑有一个数据结构只有姓名和地址两个字段，如果随便什么字符串都可以作为有效的姓名或者地址。那么结构体（structure）就够了。把姓名和地址作为私有成员封装到一个类中，再提供一堆访问私有成员的接口函数，是很愚蠢的。更愚蠢的就是再去设计一个基类，把这些接口函数设计为虚函数。没有必要。

Bill Venners: 你说没有必要是因为，数据的表示（representation）有且仅有一种。如果你要把这一种表示（representation）定义为一个函数，意味着将来你会修改表示（representation）。

Bjarne Stroustrup: 对。但是有些表示（representation）并不改变。例如整数，浮点数，复数等等。设计时你不得不做决定。

仍旧以上面的姓名和地址的数据结构为例。接下来，如果你从简单的结构体（structure）转向了真正的类。也许你不会把那个类的名字叫做“姓名和地址的组合类”。也许你会把那个类命名为“个人通讯录”。然后你觉得需要确保“个人通讯录”中的地址应该是有意义的字符串，等等等等。接下来你必须考虑数据的不同表示（representation）。例如姓名字段定义为私有数据成员吗？是否需要定义虚函数？你必须做设计而不是随意定义一些类和函数，你需要使用C++的语法来表达你的核心的设计思想，而不是简单的定义几个私有数据就完事了。（译者按：原文为semantics that you are defending，字面意思是保卫你所使用的符号的意义。）

例如，构造函数（constructor）建立了环境供成员函数（member function）在其中操作数据。也就是说，构造函数（constructor）建立了不变式（invariant）。要建立不变式（invariant），你必须获得资源。析构函数（destructor）做的事情正相反。资源可以是内存，文件，锁，socket等等。

1.4. 设计简单的界面（Designing Simple Interfaces） Bill Venners: 你刚才说不变式（invariant）帮助你决定了什么应该进入界面（interface）。你能进一步讲一讲你是如何设计界面的吗？如果一个函数有责任维护不变式（invariant)就应该在类中，对吗？

Bjarne Stroustrup: 对。

Bill Venners: 任何使用了数据，但是不维护（defend）不变式（invariant）的操作，就不需要放在class中？

Bjarne Stroustrup:让我举个例子。有时候某些操作必须直接访问数据才能完成。例如如果你要改变vector的大小，前提条件你必须能够移动其中的元素（element），修改存储大小的变量。如果你没有直接修改数据的权限，为了完成操作，你必须访问有权限的接口函数。但是一个在vector中查找指定元素的操作最好不要定义为vector的成员函数。

另一个例子是日期类，修改日期类中的年月日数据的操作当然应该是类的成员函数，但是查找下个周末的函数不应该是成员函数。我曾经看到日期类有60或者70个接口函数。接口函数直接访问数据。也就是说，如果你修改了类的数据结构，你就必须检查并修改所有类的接口函数。

如果日期类只有比如说10个必需的接口。你可以把其余的50个接口建立在这10个接口之上。其余50个接口可以放在支持库中（supporting library）。现在这种设计思想已经被普遍接受了。甚至在java中也是如此。

过去二十年我就在鼓吹这种设计思想。但是人们就是喜欢把所有的东西都丢到类和子类中去。比如说前面那个糟糕的日期类。如果你需要一些简单的工具操作日期，你必须继承自那个日期类。最后一切都乱七八糟。我仅仅是想自由的组合一些工具进行简单的操作，为什么需要继承呢？如果我想组合使用我的一些工具和你的一些工具，难道我需要定义一个子类，同时继承你的日期类和我的日期类吗？继承引入了不必要的依赖关系。