# 什么是“对用户友好”

## 什么是“对用户友好”


当我提到一个工具“对用户不友好”(user-unfriendly)的时候，我总是被人“鄙视”。难道这就叫“以其人之道还治其人之身”？想当年有人对我抱怨 Linux 或者 TeX 对用户不友好的时候，我貌似也差不多的态度吧。现在当我指出 TeX 的各种缺点，提出新的解决方案的时候，往往会有美国同学眼角一抬，说：“菜鸟们抱怨工具不好用，那是因为他们不会用。LaTeX 是‘所想即所得’，所以不像 Word 之类的上手。”

殊不知他面前这个“菜鸟”，其实早已把 TeX 的配置搞得滚瓜烂熟，把 TeXbook 翻来覆去看了两遍，”double bend” 的习题都全部完成，可以用 TeX 的语言来写宏包。而他被叫做“菜鸟”，这是一个非常有趣的问题。所以现在抛开个人感情不谈，我们来探讨一下这种“鄙视”现象产生的原因，以及什么叫做“对用户友好”。

首先我们从心理的角度来分析一下为什么有人对这种“对用户不友好”的事实视而不见，而称抱怨的用户为“菜鸟”。这个似乎很明显，答案是“优越感”。如果每个人都会做一件事情，如何能体现出我的超群智力？所以我就是要专门选择那种最难用，最晦涩，最显得高深的东西，把它折腾会。这样我就可以被称为“高手”，就可以傲视群雄。我不得不承认，我以前也有类似的思想。从上本科以来我就一直在想，同样都会写程序，是什么让计算机系的学生与非计算机系的学生有所不同？经过多年之后的今天，我终于得到了答案（以后再告诉你）。可是在多年以前，我犯了跟很多人一样的错误：把“难度”与“智力”或者“专业程度”相等同。但是其实，一个人会用难用的工具，并不等于他智力超群或者更加专业。

可惜的是，我发现世界上有非常少的人明白这个道理。在大学里，公司里，彰显自己对难用的工具的掌握程度的人比比皆是。这不只是对于计算机系统，这也针对数学以及逻辑等抽象的学科。经常听人很自豪的说：“我准备用XX逻辑设计一个公理化的系统……”可是这些人其实只知道这个逻辑的皮毛，他们会用这个逻辑，却不知道它里面所有含混晦涩的规则都可以用更简单更直观的方法推导出来。

爱因斯坦说：“Any intelligent fool can make things bigger and more complex… It takes a touch of genius - and a lot of courage to move in the opposite direction.”我现在深深的体会到这句话的道理。想要简化一个东西，让它更“好用”，你确实需要很大的勇气。而且你必须故意的忽略这个东西的一些细节。但是由于你的身边都是不理解这个道理的人，他们会把你当成菜鸟或者白痴。即使你成功了，可能也很难说服他们去尝试这个简化后的东西。

那么现在我们来谈一下什么是“对用户友好”。如何定义“对用户友好”？如何精确的判断一个东西是否对用户友好？我觉得这是一个现在仍然非常模糊的概念，但是程序语言的设计思想，特别是其中的类型理论(type theory)可以比较好的解释它。我们可以把机器和人看作同一个系统：

1. 这个系统有多个模块，包括机器模块和人类模块。
2. 机器模块之间的界面使用通常的程序接口。
3. 人机交互的界面就是机器模块和人类模块之间的接口。

4. 每个界面必须提供一定的抽象，用于防止使用者得到它不该知道的细节。这个使用者可能是机器模块，也可能是人类模块。
5. 抽象使得系统具有可扩展性。因为只要界面不变，模块改动之后，它的使用者完全不用修改。

在机器的各个模块间，抽象表现为函数或者方法的类型(type)，程序的模块(module)定义，操作系统的系统调用(system call)，等等。但是它们的本质都是一样的：他们告诉使用者“你能用我来干什么”。很多程序员都会注意到这些机器界面的抽象，让使用者尽量少的接触到实现细节。可是他们却往往忽视了人和机器之间的界面。也许他们没有忽视它，但是他们却用非常不一样的设计思想来考虑这个问题。他们没有真正把人当成这个系统的一部分，没有像对待其它机器模块一样，提供具有良好抽象的界面给人。他们貌似觉得人应该可以多做一些事情，所以把纷繁复杂的程序内部结构暴露给人（包括他们自己）。所以人对“我能用这个程序干什么”这个问题总是很糊涂。当程序被修改之后，还经常需要让人的操作发生改变，所以这个系统对于人的可扩展性就差。通常程序员都考虑到机器各界面之间的扩展性，却没有考虑到机器与人之间界面的可扩展性。

举个例子。很多 Unix 程序都有配置文件，它们也设置环境变量，它们还有命令行参数。这样每个用户都得知道配置文件的名字，位置和格式，环境变量的名字以及意义，命令行参数的意义。一个程序还好，如果有很多程序，每个程序都在不同的位置放置不同名字的配置文件，每个配置文件的格式都不一样，这些配置会把人给搞糊涂。经常出现程序说找不到配置文件，看手册吧，手册说配置文件的位置是某某环境变量 FOO 决定的。改了环境变量却发现没有解决问题。没办法，只好上论坛问，终于发现配置文件起作用当且仅当在同一个目录里没有一个叫 “.bar” 的文件。好不容易记住了这条规则，这个程序升级之后，又把规则给改了，所以这个用户又继续琢磨，继续上论坛，如此反复。也许这就叫做“折腾”？他何时才能干自己的事情？

TeX 系统的配置就更为麻烦。成千上万个小文件，很少有人理解 kpathsea 的结构和用法，折腾好久才会明白。但是其实它只是解决一个非常微不足道的问题。TeX 的语言也有很大问题，使得扩展起来非常困难。这个以后再讲。

一个良好的界面不应该是这样的。它给予用户的界面，应该只有一些简单的设定。用户应该用同样的方法来设置所有程序的所有参数，因为它们只不过是一个从变量到值的映射（map）。至于系统要在什么地方存储这些设定，如何找到它们，具体的格式，用户根本不应该知道。这跟高级语言的运行时系统(runtime system)的内存管理是一个道理。程序请求建立一个对象，系统收到指令后分配一块内存，进行初始化，然后把对象的引用(reference)返回给程序。程序并不知道对象存在于内存的哪个位置，而且不应该知道。程序不应该使用对象的地址来进行运算。

所以我们看到，“对用户不友好”的背后，其实是程序设计的不合理使得它们缺少抽象，而不是用户的问题。这种对用户不友好的现象在 Windows，Mac，iPhone, Android 里也普遍存在。比如几乎所有 iPhone 用户都被洗脑的一个错误是“iPhone 只需要一个按钮”。一个按钮其实是不够的。还有就是像 Photoshop, Illustrator, Flash 之类的软件的菜单界面，其实把用户需要的功能和设置给掩藏了起来，分类也经常出现不合理现象，让他们很难找到这些功能。

如何对用户更加友好，是一两句话说不清楚的事情。所以这里只粗略说一下我想到过的要点：

1. 统一：随时注意，人是一个统一的系统的一部分，而不是什么古怪的神物。基本上可以把人想象成一个程序模块。

2. 抽象：最大限度的掩盖程序内部的实现，尽量不让人知道他不必要知道的东西。不愿意暴露给其它程序模块的细节，也不要暴露给人。“机所不欲，勿施于人”。

3. 充要：提供给人充分而必要（不多于）的机制来完成人想完成的任务。

4. 正交：机制之间应该尽量减少冗余和重叠，保持正交(orthogonal)。

5. 组合：机制之间应该可以组合(compose)，尽量使得干同一件事情只有一种组合。

6. 理性：并不是所有人想要的功能都是应该有的，他们经常欺骗自己，要搞清楚那些是他们真正需要的功能。

7. 信道：人的输入输出包括5种感官，虽然通常电脑只与人通过视觉和听觉交互。

8. 直觉：人是靠直觉和模型(model)思考的，给人的信息不管是符号还是图形，应该容易在人脑中建立起直观的模型，这样人才能高效的操作它们。

9. 上下文：人脑的“高速缓存”的容量是很小的。试试你能同时想起7个人的名字吗？所以在任一特定时刻，应该只提供与当前被关注对象相关的操作，而不是提供所有情况下的所有操作供人选择。上下文菜单和依据上下文的键盘操作提示，貌似不错的主意。