

软件即沟通

从后台日志到 vibe coding：好产品背后，是对人的共情

竹小竹8167

费曼式写法 · 由多个 AI 代理撰写与互相审校

导读

从后台日志到 vibe coding：好产品背后，是对人的共情

先别打开需求文档，先去看看后台日志

假设你是一个软件团队的新人，入职第一天，想快速搞懂「我们的用户过得好不好」。你会怎么做？

多数人的选择是看需求文档、看产品路线图、听产品经理讲用户画像。这本书建议你做一件反直觉的事：**去把后台日志调出来，随机挑一个真实用户，把他过去一周的操作流水从头到尾读一遍。**

你会读到一个人——不是画像，是真人。你会看到他在哪个页面停留了四十秒什么也没点（他在犹豫），看到他把同一个表单提交了三次（前两次被拒绝了，但没人告诉他为什么），看到他在凌晨一点终于放弃（他明天还要上班）。没有一行红色报错，系统一切正常，而这个人的体验糟透了。

这本书的立场很简单：**使用软件就是跟软件沟通，而沟通的质量是可以看见、可以衡量、可以施工的。**界面是它的措辞，流程是它的节奏，报错是它说漏嘴的诚实时刻，日志是它以为没人听时的真心话。而这一切的背后，是做软件的人愿不愿意站在用软件的人那边想问题——我们管这个叫同理心，但在这本书里，它不是态度，是默认值、容错、解释、撤销这些能派活、能验收的工程决定。

这本书还管一个更当下的问题：AI 来了，普通人也开始用「说人话」的方式自己写软件了——vibe coding。沟通的对象从界面变成了 AI，用户的挣扎消失了吗？没有，它只是搬家了：从「点不动按钮」搬到了「说不清需求」。后半本书会讲这场迁徙，以及它对每一个做软件、用软件的人意味着什么。

十二章，一条线：从「点击是一句话」开始，到「技术的尽头是关切」结束。你可以从头读，也可以从最戳中你的那章开始——每一章都能独立成立。

那么，回到开头那个新人。他读完那一周的日志之后，做的第一件事，是把验证码错误的提示从「校验失败」改成了「验证码不对，注意区分大小写；收不到短信可以点这里语音接听」。第二天，那个断崖少了一半的人。

这就是这本书想让你也能做到的事。

第一章 · 点击是一句话

你以为你在用软件，其实你在跟它说话

想象你走进一家银行。柜台后面坐着一个柜员，你把存折递过去，说：「取五百块。」柜员看了你一眼，什么也没说，把存折从窗口推了回来。

你愣了一下，又递过去。又被推回来。第三次，柜员终于开口，说了三个字：「先取号。」

你一定遇到过这样的软件。点了一个按钮，什么都没发生——没有转圈，没有提示，没有报错，就好像你根本没点过。你只好再点一次，再点一次，然后页面突然跳出一行红字：「操作过于频繁」。

这就是这本书的出发点：**使用软件，本质上是一场对话**。你每一次点击、每一次输入、每一次滑动，都是你在对软件说一句话；而软件的每一个响应——或者没有响应——都是它的回答。

界面是一种语言

我们平时把软件的界面叫做 UI（User Interface，用户界面），大白话就是「软件的脸面」。但这个说法有个误导：它让人觉得界面是给人**看**的，像一张海报。实际上界面是给人**用**的，更像一场对话里的措辞。

大白话

大白话：海报只要好看就行，对话还得接得住话。界面是后者——你说一句，它得答一句，而且答案得让你知道下一步该干嘛。

想想看，一个按钮上的文字就是在跟你沟通。「提交」和「提交并发布，所有人可见」是两种完全不同的回答方式——前者让你猜，后者把后果说清楚了。一个加载中的转圈动画，翻译成成人话就是「我听到了，正在办，你稍等」。而那个点了没反应的按钮，翻译过来是：沉默。

人与人之间的沉默已经是糟糕的沟通了——你说了三句话对方一声不吭，你会怀疑电话断了。软件的沉默也一样，只是用户没法问「喂，你还在吗」，只能狂点。

对话质量是可以衡量的

把「使用软件」看成对话之后，很多原本模糊的争论突然变得具体了。

「这个软件好不好用」不再是玄学，而是这场对话里那些可以数出来的东西：

- 你说一句话，它多久回应？（响应速度）
- 它的回答你听得懂吗？（提示是不是人话）
- 你说错了，它是纠正你，还是惩罚你？（出错时的待遇）
- 它能听懂你没说出口的半句话吗？（默认值和预判）

这些都是沟通的问题，不是技术的问题。技术上，「点了没反应的按钮」和「点了立刻转圈的按钮」背后可能是完全相同的代码，只差一行——但在沟通的层面，它们一个是沉默，一个是「收到」。这一行代码的差距，就是用户摔鼠标和耐心等待的差距。

那谁在替软件说话？

这里有一个容易被忽略的环节：软件不会自己说话。**界面上的每一个字、每一个流程，都是某个产品经理或工程师替软件设计好的台词。**

所以「软件跟用户沟通」，拆开来看，其实是「做软件的人，隔着一层界面，在跟用软件的人沟通」。这场沟通很特别：说话的人不在场，回答要预先写好，而听众是成千上万个陌生人，每个人都可能在你意想不到的地方卡住。

这就解释了为什么有的软件让人觉得「贴心」——它背后的团队在写台词的时候，真的替你预演过你会遇到什么；而有的软件让人觉得「刁难」——它的台词是从数据库错误码直接抄来的，写的人根本没站在听的人那边想过。

下一章我们要去一个大多数用户永远看不到、但对这场对话最诚实的地方：后台日志。那里有用户没说出口的话。

第二章 · 日志里的呼救

用户不会告诉你他卡住了，但日志会

每一台服务器都在不停地写日记。用户每点一次按钮、每打开一个页面、每触发一次错误，后台就记下一行。这本日记叫日志（log）——软件运行过程的流水账，本来是为了让工程师排查故障用的。

大多数工程师看日志的方式是「找异常」：哪行红了，哪行报错了，哪里响应超时了。但如果你换一种看法，把日志当成用户行为的录像带，你会看到完全不同的东西。

一行日志背后的三十秒

来看一段很多产品后台都真实存在的日志模式：

```
14:02:11 用户打开「修改手机号」页面
14:02:19 提交，失败：验证码错误
14:02:40 提交，失败：验证码错误
14:02:58 重新发送验证码
14:03:31 提交，失败：验证码错误
14:03:47 重新发送验证码
14:04:12 重新发送验证码
14:04:40 用户离开了页面，再没回来
```

在工程师眼里，这可能只是「验证码校验失败 3 次，重发 3 次」——没有红色异常，系统一切正常。但如果你把这段时间轴想象成一个人坐在手机前的样子：他盯着屏幕等短信，短信没来；他把旧验证码又输了一遍，错误；他再点重发，把手机翻过来扣在桌上等，还是没来；最后他放下手机去干别的了。

这不是一次「校验失败」，这是一个人挣扎了两分半钟，然后放弃了。日志里全是这样的呼救，只是它们穿着技术事件的外衣。

大白话：日志就像医院的走廊摄像头。医生（工程师）平时只看心电监护仪报警（系统异常），但摄像头里能看到一个病人扶着墙站不起来，按了三次呼叫铃没人理，最后自己走了。监护仪上一切正常。

挣扎有形状

看得多了，你会发现用户的挣扎在日志里有几种反复出现的「形状」：

- **原地打转**：同一个页面来回刷新、同一个按钮反复点。翻译过来是「我没看懂，但我不甘心」。
- **步步后退**：一路返回到首页，然后从头再来一遍。翻译过来是「我知道我把自己弄丢了，重来」。
- **断崖**：在某一步之后，会话戛然而止。这一步就是整段流程里最难过的坎——它可能只是一个让人看不懂的下拉框。
- **深夜重试**：同一个操作在凌晨反复失败。这个人很可能很着急，而且明天还会再来。

每一种形状都是一句没说出口的话。而它们的共同点在于：**系统层面一切正常，没有一个报错**。如果只看异常监控，这些用户是隐形的。

为什么研发看不到这些

这不能怪工程师冷漠，这里有个结构性的问题：**离日志最近的人，恰恰是最不需要这些提示的人**。工程师对自己写的软件太熟了，他知道验证码在垃圾短信里、知道那个按钮要点左上角。他看日志时戴着「知情者」的眼镜，自动过滤掉了那些「这有什么好卡住的」的部分。

心理学里管这个叫知识的诅咒——人一旦掌握了某个知识，就很难想象「不知道这个知识」是什么感觉。写软件的人全都中了这个诅咒，而解药恰恰就躺在他们自己的服务器上：日志里那些打转、后退、断崖的轨迹，全都是「不知道的人」留下的脚印。

所以看日志的正确姿势，不是问「系统哪里坏了」，而是问「**如果是我的妈妈在用，她会在第几行卡住**」。同一个日志文件，这两个问题会带你看到完全不同的东西。

当然，日志里也有真正在「报错」的时刻——那是软件亲口对用户说话的时刻，也是整场对话里最诚实、最容易搞砸的时刻。下一章专门讲它。

第三章 · 报错是最诚实的对话

软件对你说真话的唯一时刻

平时，软件的界面是精心装修过的：图标圆润，配色舒服，文案经过打磨。但出错的那一刻，装修会突然塌掉，露出里面的毛坯房——「Error 500: Internal Server Error」，或者更糟的，一长串普通人看了会心跳加速的英文堆栈。

所以我说报错是软件最诚实的时刻：**它是系统内部状态不小心说漏了嘴**。而这个时刻，恰恰也是用户最脆弱、最需要被好好对待的时刻——事情已经出错了，用户已经有损失了（时间、数据，或心情），这场对话怎么处理，决定了他还信不信你。

一条好的错误提示只有三个问题

别看错误提示只有一两行字，它要回答的问题其实非常固定。用户看到报错的瞬间，脑子里只转三个念头：

1. **出了什么事？**——用我听得懂的话说。
2. **是我的错吗？**——我需不需要内疚、需不需要改什么？
3. **现在怎么办？**——给我下一步，别把我晾在这儿。

拿这三问去量一下最常见的错误提示，你会发现它们几乎全部不及格：

- 「Error 500」——三个问题一个都没回答。它甚至不是一个句子，是系统内部的房间号。
- 「操作失败」——回答了第一问但约等于没答（我知道失败了，我看见了），后两问全无。
- 「参数非法」——第二问的回答是「是你的错」，但不说错在哪，等于只说「你错了」却不判卷子。

而一条及格的错误提示长这样：**「没保存成功——是网络断了，不是你操作的问题。你写的内容都还在，网络恢复后点这里重试。」**三个问题全答了，而且顺序都对：先说事，再摘清责任，最后给出路。

大白话：好的报错像一个好护士。「只是皮外伤（出了什么事），不怪你，地太滑（谁的错），先坐这儿，我给你包扎，两分钟就好（怎么办）。」坏的报错像一张贴在急诊室门口的纸条：「故障代码 502。」

为什么工程师写不出人话报错

这不是文笔问题，是位置问题。错误提示通常写在代码的「异常分支」里——那是工程师最不想待的地方。写主流程的时候他意气风发，在实现一个激动人心的功能；写到出错处理时，他只想赶紧兜住别崩溃，于是随手把系统内部的错误码抛了出去。

于是用户看到的报错，其实是**工程师在最低落的时刻、对着系统（而不是对着人）说的话**。「Error 500」根本不是写给用户看的，它是工程师写给三天后排查问题的自己的备忘录，只是不小心显示到了屏幕上。

理解了这一点，改法就清楚了：错误提示不是代码的附属品，它是产品文案的一部分，应该用写首页标语的认真程度去写。有些团队甚至有一个规矩——**每一条会显示给用户的报错，都要过一遍「三个问题」检查**，过不了不许上线。这个规矩几乎零成本，但它把无数个「用户对屏幕不知所措」的瞬间，变成了「哦，那我再试试」。

比报错更好的是不报错

当然，最高明的错误处理是根本不让错误到达用户。这一层也有讲究：

- 能自动重试的，悄悄重试，别声张——网络抖一下就弹窗，是把系统的神经质传染给用户。
- 能提前拦的，提前拦——密码要八位就在输入框旁边写好，别等他提交完才说。
- 实在要报错的，按三个问题好好写。

你会发现这三层背后是同一种东西：**设计者提前替用户想到了会出什么事**。这种东西有个听起来很软的名字，叫同理心。但下一章我要说的是，同理心在这行里不是态度，是一门可以拆解、可以施工的技术。

第四章 · 同理心是一门技术

同理心不是态度，是施工工艺

「要有同理心」是做产品的人听得最多的一句话，也是最没用的一句话——它听起来像个人修养，没法验收，没法排期，开完会就散了。

但如果你把同理心拆开看，它在软件里其实是以**一个个具体的工程决定**存在的。每一个都能指出来、能写进需求、能测试。这一章我们拆四个最常见的。

默认值：替用户把最可能的答案先填上

打开一个新的记账软件，创建第一笔账，日期那一栏是空的，还是已经帮你填好了「今天」？

这个区别就是默认值——输入框里预先填好的、最可能被接受的答案。它背后的推理是同理心最标准的形式：**用户接下来大概率想干什么？**记一笔账，十次里九次记的是今天发生的事，那日期就应该是今天；新建文档，十次里九次接下来就要打字，那光标就应该已经在标题栏里闪。

默认值是一种「说了半句话就替对方补全」的能力。人和人之间，这是默契；软件里，这是设计者愿不愿意花一下午去研究「用户在这一步最常选什么」。

预判：在用户开口之前接住

比默认值再进一步的是预判。你在地图软件里搜索过一个地址，下次打开它，那个地址就在搜索历史最上面；你晚上十一点打开打车软件，它猜你要回家，家地址已经躺在那儿。

预判的本质是把「用户接下来要走的弯路」提前走掉。它的工程成本不低——要记录行为、要做推荐、要承担猜错的风险——所以它最能反映一个团队把「少让用户说一句话」看得有多重。

当然，预判有一条红线：**猜错的时候必须体面**。猜对了省一秒，猜错了如果挡路，用户会觉得被冒犯——「你凭什么觉得我要回家」。所以好的预判永远是「放在顺手可拿的位置」，而不是「替你做了决定」。推荐和强迫的区别，就是共情和自说自话的区别。

容错：假设用户一定会错

这一点听起来反直觉，但它是同理心最深的部分：**设计的时候就把「用户会犯错」当成前提，而不是当成意外**。

电话号码输入框，用户输成「138-1234-5678」带两个横杠。不友好的系统说「格式错误」；友好的系统默默把横杠删掉——因为横杠根本不是错误，是人记电话号码的自然方式。邮箱多输了一个空格、身份证号最后以为是数字 0 其实是字母 O……这些都不是「用户笨」，是人和机器对「同一个东西」的写法本来就不一样。

工程上这叫宽进（Postel 法则，「对自己发出的要严格，对接收的要宽容」的俗称）：接口尽量接受各种合理的写法，在系统内部悄悄规整好。用户甚至不会知道有这回事——这恰恰是它的美德。最好的共情是用户感受不到的共情，就像最好的服务员是你没注意到他添过水。

解释：不让用户猜后果

最后一个工程决定是解释。每个有后果的操作旁边，后果应该写得明明白白。「删除」还是「删除且不可恢复」？「提交」还是「提交后将进入审核，约 1 个工作日」？

用户在点按钮前的犹豫，本质是信息不对称：他不知道按下去会发生什么。解释就是把信息差抹平。这件事的成本低到只是一行字，但它要求写这行字的人心里装着一个问题：「如果我是第一次来，我现在最怕什么？」

大白话

大白话：同理心在这行里不是「我很善良」，是四张施工清单——默认值、预判、容错、解释。每一张都能派活、能验收、能写进测试用例。「我们的团队很有同理心」这句话不值钱；「我们表单里 90% 的字段有默认值，所有删除操作都能撤销，所有报错都过了三问检查」才值钱。

这四个决定组合起来，就是用户口中那种说不清道不明的「顺手」。下一章我们把它放到一个每个软件都有、也最容易变成灾难的场景里看：表单。

第五章 · 表单是审讯还是聊天

为什么填个表像被审问

你办过那种业务吗：打开页面，一张长长的表格，二十几个字段从上到下排开——姓名、证件类型、证件号、有效期、签发机关、联系地址、邮编、紧急联系人、紧急联系人与你的关系……滚两屏都滚不到底。

你对照证件一项一项填，填到「签发机关」卡住了，翻出证件找到那行小字抄上去。终于填完，点提交——页面一闪，顶部一行红字：「第 3 项格式有误」。而下面你填的二十几项，全没了。

这就是把数据库表格直接拍在用户脸上的结果。而它和「一场舒服的对话」之间的差距，正是这一章要讲的：**流程是有节奏的，节奏本身就是沟通。**

表单是数据库的形状，不是对话的形状

那张二十几字段的长表是从哪来的？多半是工程师看着数据库表结构做的：表里有多少列，页面上就有多少个输入框，一次提交，正好对应一条 `INSERT` 语句（往数据库里插一条记录的命令）。对系统来说，这是最省事的设计——页面的形状和数据库的形状一模一样。

但对话不是这么进行的。正常人和人办事，是一问一答的：「您贵姓？」「要办什么业务？」「证件带了吗？」每一问都比较小，回答完还有反馈。对话有回合——你说一句，我接一句，双方都随时知道「现在进行到哪了」。

二十个字段一页砸脸的表单，相当于柜员一口气念完二十个问题然后等你逐一背答案。它省的是系统的事，费的是用户的心。

把一张表拆成一场对话

很多产品后来把长表单改成了「向导」——一步一页，每页只问一两件事，顶部有个进度条。这不只是排版变化，是把数据库的形状翻译成了对话的形状：

- **一次只问一件事**，就像对话里一次只问一个问题。用户的心智负担从「管理二十个字段」降到「回答眼前这一问」。
- **进度条就是对话里的「咱们进行到哪了」**。「第 2 步，共 4 步」回答的是用户心里那个一直没问出口的问题：还有多久？
- **每填完一步就校验一步**，错了当场说，而不是攒到最后一次性清算。对话里没人会把你的二十句话听完才说「你第三句说错了」。
- **能替你填的就替你填**——上一章说的默认值在这里发威：证件类型默认「居民身份证」，因为九成用户选它。

注意，这些改法没有减少任何信息——要填的还是那些内容，变的只是**节奏**。同样是二十个问题，一口气问完是审讯，一问一答是办事。

节奏的成本谁付

这里有个诚实的账要算：把表单拆成对话，成本是真实存在的。开发量上去了一一分页、暂存、断点续填、每步校验，每一样都要写。所以「要不要拆」本质上是一个取舍：**开发团队多花的这一周，换来的是每一个用户少受的那十分钟罪**。

不友好的软件，几乎无一例外是把成本留给了用户侧；友好的软件，是团队把成本扛到了自己侧。这又一次回到了那个主题：界面是沟通，而沟通的质量，取决于说话的那一方愿意为听话的那一方多做多少。

大白话

大白话：数据库喜欢一次性要齐所有信息，人喜欢一次回答一个问题。流程设计就是决定按谁的脾气来。按数据库的脾气来，开发省事、用户受罪；按人的脾气来，开发费事、用户舒服。「易用性」三个字背后，其实是这笔账往哪边付。

但还有一个更深的问题：你拆了表单、写了人话报错，用户就满意了吗？事实是，大多数不满意的用户什么都不会说——他们直接消失。下一章讲怎么听见这些沉默的人。

第六章 · 沉默的大多数

骂你的用户是你的朋友

先记住一个行业里流传很广、也被反复验证过的比例：遇到问题时，**会开口投诉的用户只是极少数**，常见说法是二十几个人里大约只有一个会吭声。剩下的人呢？他们不写反馈、不打客服电话、不发微博骂你。他们只是默默关掉页面，然后再也不来。

这对产品团队是个陷阱：客服邮箱里那几封措辞激烈的邮件，很容易让人产生「用户意见我们都听到了」的错觉。其实那几封邮件只是冰山露出水面的一角——而且往往还是最不重要的一角，因为会写信的用户至少还在乎你，还愿意花力气救这段关系。真正致命的是水面下那些一句话都没留就走了的人。

会骂的用户是朋友，沉默的用户才是审判。

沉默不等于满意

要听见沉默的人，只能靠他们留下的脚印——行为数据。第二章我们讲了日志里「挣扎的形状」，这一章讲怎么从数据的形状里读出大多数人的态度。有几种模式几乎是通用的：

- **漏斗的断崖**：把「打开页面 → 填写 → 提交 → 成功」每一步的人数画出来，像一个漏斗。健康的漏斗是平缓变窄的；如果某一步突然掉下去一大截，那一步就是沉默的大多数集体放弃的地方。没人投诉，但数据在尖叫。
- **停留时间的两端**：一个页面停留太久和太短都可能是坏消息。太短是没找到想要的东西；太久，尤其是在一个简单的页面上太久，往往是「看不懂，正在猜」。
- **来了就走**：新用户第一天注册，第二天再没打开过。这个比例（行话叫次日留存，就是第二天还回来的用户占比）是最诚实的投票——它不受任何客服话术影响，只反映第一次体验到底好不好。

这些指标的共同点：它们度量的是用户**做了什么**，而不是用户**说了什么**。说的部分是那二十几分之一嗓门，做的部分才是全部人的投票。

数据能告诉你「哪里」，不能告诉你「为什么」

但这里必须泼一盆冷水，否则会滑向另一个极端——以为看数据就够了。

漏斗能告诉你 40% 的用户死在第三步，但它永远说不出第三步上那个具体的瞬间：是「签发机关」这四个字没人懂？是证件照片传了三次都失败？还是按钮在某些机型上根本点不动？数据是症状，不是诊断。

诊断要靠回到「人」的那一侧：找几个真实用户，坐在旁边看他们操作（行话叫可用性测试，就是观察真人用你的产品的过程），你会在十分钟内看到数据里潜伏了半年的问题。用户的手指悬在那个按钮上方犹豫的那两秒，任何报表都抓不到，但那两秒就是一切的答案。

大白话

大白话：数据像体检报告，能告诉你「肝功能异常」；可用性测试像医生问诊，才知道「哦，他天天熬夜喝酒」。只做体检不问诊，或者只问诊不体检，都治不好病。好的产品团队两样都做：用数据找到哪里在流血，用观察找到伤口长什么样。

听见沉默是一种能力

把前两章连起来看，逻辑就很完整了：日志和数据让你**知道**用户在挣扎（第二章的录像带、本章的漏斗），而同理心技术让你**修掉**那些挣扎（第四章的四个决定）。缺了前者，你修的都是想象出来的问题；缺了后者，你只会日复一日地看着用户流失。

很多团队的真实状态是两头都缺：不看日志，觉得「没投诉就是没问题」；不做可用性测试，觉得「我自己用着挺顺」。于是他们活在二十几分之一的嗓门营造的世界里，而沉默的大多数正在用脚投票。

听见沉默，是这个时代做软件的人最稀缺的能力之一。而下一章我们要讲一个更积极的主题：当你真的听见了、也修好了，用户会变成什么样——答案是，他们开始敢于犯错。

第七章 · 好软件是被原谅出来的

你敢随便点的软件，才是好软件

回忆一下你第一次用某个专业软件的感觉——比如财务系统或者视频剪辑软件。你盯着满屏的按钮，手悬在鼠标上不敢动，心里有个声音：「点错了怎么办？」

再回忆一下你用那些用得最顺手的软件的感觉。你发现区别了吗：顺手的软件里，你敢乱点。看到一个按钮不知道是什么，点一下看看；改了个设置不满意，改回来。你的心态从「别碰坏了」变成了「玩玩看」。

这两种心态的差距，不是用户勇不勇敢，是软件给了多大的安全感。而安全感几乎完全来自一个东西：犯了错的代价有多小。

撤销：软件界最伟大的发明之一

Ctrl+Z（撤销键）大概是全世界被按得最多的快捷键组合。它做的事情简单到一句话：让刚才那一步不算数。

但它的意义远超一个功能。撤销把用户和软件的关系整个翻转了：没有撤销的软件里，每个操作都是承诺——落子无悔，所以每一步都要犹豫；有撤销的软件里，每个操作都是试探——先做了看看，不行就退回来。前者培养的是战战兢兢的用户，后者培养的是敢于探索的用户。

而探索欲是用户变成熟手的唯一路径。没有人靠读说明书变成高手，高手都是乱点出来的——前提是这个软件允许乱点。所以「好软件是被原谅出来的」这句话可以改写成：用户是在一次次被原谅里，学会了这个软件。

比撤销更好的是自动保存

撤销管的是「做错了能退」，自动保存管的是「意外了不丢」。年纪稍长的读者大概都有过这样的记忆：写了一下午的文档，断电，全没了。那种绝望足以让人对一台机器记恨很多年。

现在的文档软件大多改成了「你打的每个字实时存下来」，保存按钮干脆消失了。这是个了不起的界面决策：它把「记得保存」这个本不该由人承担的责任，从用户肩上拿了下来，交给了机器。

注意这里发生了什么：**一个功能最好的归宿，是消失**。保存按钮没有变得更大更好点，它被消灭了，因为「保存」从来就不是用户的目标——用户的目标是「别丢」。软件直接实现了那个目标，中间的仪式就过时了。第四章说的「最好的共情是感受不到的共情」，这是最极致的例子。

渐进披露：别把整本说明书摊在桌上

安全感的另一个来源是渐进披露——大白话就是「用到什么才给你看什么」，而不是一上来把所有功能铺满屏幕。

对比一下两种新手第一屏：一种是几十个图标、五层菜单的专业软件；一种是手机拍照应用——打开就一个取景框和一个大圆钮。后者不是功能少，它也有滤镜、参数、编辑，只是都藏在「你需要时再出现」的地方。

渐进披露的逻辑和第五章的表单同理：一次性给全部信息，等于要求用户先读完说明书才能动手；按需要的节奏给信息，用户是「在游泳中学会游泳」。前者筛选出的是有耐心的少数，后者欢迎的是所有人。

安全感的账，最后还是人付的

和第五章一样，这些都有成本：撤销要保存历史状态，自动保存要处理冲突，渐进披露要设计「什么时候出现什么」。全是工程师的活。

所以「敢不敢乱点」这个用户体验问题，根子上还是个团队选择：愿不愿意把「容错」写进正式需求，而不是等上线后靠用户忍。那些让人越用越敢、越敢越熟的软件，背后都是替用户把「犯错的代价」一笔一笔付掉的团队。

到这里，本书的上半场讲完了：传统软件里，沟通发生在用户和界面之间，好的沟通靠团队提前替用户想好一切。但最近几年，出现了一种全新的软件生产方式——用户干脆绕过界面，直接对 AI 说出自己想要什么，让 AI 把软件做出来。下半场从这件事开始：当用户自己成了开发者，我们前面说的一切还成立吗？

第八章 · 当每个人都开始写软件

编程的门槛，塌了一面墙

2025 年初，AI 研究员 Andrej Karpathy（曾任特斯拉 AI 总监、OpenAI 创始成员）随口造了个词，结果传遍了整个行业：vibe coding，大意是「凭感觉编程」——你不再亲手写代码，而是用日常语言告诉 AI 你想要什么，AI 来写，你来看效果，不满意就再说一句，如此往复。

他描述自己的工作方式时半开玩笑地说：「我不怎么看代码了，我就说说想法，运行一下，看看效果，哪里不对就再跟它说。」这个词能病毒式传播，是因为它精准命名了一件正在大规模发生的事：**写软件的门槛，第一次从「会编程语言」降到了「会把需求说清楚」**。

这件事的影响，远不止是程序员效率提高了。它意味着用户和软件的关系发生了根本变化——上半本书里，用户是沟通链条的终点（团队设计界面，用户用）；现在，用户站到了链条的起点（用户描述需求，AI 生成软件）。

一个真实的画面

一个开小花店的人，想要一个能记录「哪个老顾客的结婚纪念日快到了、该提醒送花了」的小工具。在旧世界里，她的选项是：忍受通用软件的不合身，或者花几万块找人定制。

在 vibe coding 的世界里，她打开一个 AI 工具，打字：「帮我做一个网页，能录入顾客名字和纪念日，到日子前一周提醒我。」十分钟后，一个能用的东西出现在屏幕上。她说：「提醒方式改成发我微信……算了，发短信吧。」又过两分钟，改好了。

注意这个过程的形状：**它不再是「使用软件」，它变成了「提出需求、验收、再提」——她做的事情，和软件公司里甲方产品经理做的事情一模一样**。她甚至不知道什么是数据库，但她已经在指挥一个会数据库的「员工」了。

沟通没有消失，只是换了对象

上半本书说，用户使用软件是在跟软件沟通，沟通质量取决于做软件的人预写了什么。vibe coding 把这场沟通改造了：

- **沟通对象变了**：从「跟固定界面沟通」变成「跟 AI 沟通」。界面是死的，台词有限；AI 是活的，理论上什么都能聊。
- **沟通媒介变了**：从「点击和输入」（操作语言）变成「自然语言」（就是说话）。前者要学，后者天生会。
- **反馈周期变了**：传统软件「用户提需求 → 团队排期 → 几个月后上线」；vibe coding 把这个周期压到了分钟级。

但有一样东西没变，而且它是本书的核心论点：**沟通质量仍然决定一切，只是现在瓶颈从「界面做得好不好」变成了「用户说得清不清楚、AI 听得懂不懂」**。挣扎没有消失，它搬家了——从鼠标和屏幕之间，搬到了语言和意图之间。

新获得的自由，和新继承的麻烦

花店老板很快会遇到她的新麻烦：她想要「提醒短信在早上九点发，但如果那天是周日就推迟到十点」——这句话她以为自己说清了，AI 做出来的东西却在闰年二月出了错；她想改一个细节，说了一句「把上次那个改回来」，AI 却把三个地方都改了。

这些麻烦我们似曾相识：需求说不清、边界情况没想到、改一处动了三处——这就是软件工程几十年来的日常，只是以前由程序员承担，现在落到了每一个 vibe coding 的用户头上。**门槛塌了一面墙，但墙后面是个院子，院子里还是得有章法。**

这个章法是什么？下一章我们细看这个新界面——提示词——以及为什么「会跟 AI 说话」正在变成一种可以学习的手艺。

第九章 · 提示词是新界面

说人话，其实是一门精确的手艺

很多人第一次 vibe coding 的体验像过山车：第一句「帮我做个记账软件」，AI 真的做出来了一个能跑的页面，震撼；第五句「就按我说的那样改」，AI 改出了一个完全不是你要的东西，崩溃。

这中间发生了什么？发生的是：**自然语言既是最好的界面，也是最模糊的界面**。「把上次那个改回来」这句话，你对着人类同事说，他大概率懂——因为他记得上下文、知道你在乎什么、懂得「那个」指什么。AI 也会猜，但猜错的时候不会有人类的歉意表情提醒你「它可能没懂」。

所以跟 AI 说话这件事，正在分化出一种新手艺。它有个正式名字叫提示词（prompt，你给 AI 的指令），但把它当成「新界面」来理解更有用——只不过这个界面不是由别人设计好的按钮，而是你自己现场组织的语言。**上半本书讲设计师怎么把界面做好，这一章讲你自己怎么把话说好。**

三个立刻见效的说法

这门手艺有讲究，但入门意外地简单。三条就够大多数人用了：

- **说目标，也说验收标准**。「做一个提醒功能」是目标；「纪念日提前七天、当天上午九点发短信，发不出去就在页面上标红」是验收标准。前者让 AI 开工，后者让它知道什么叫「做完了」。你给 AI 的标准越具体，它就越接近一个靠谱的乙方。
- **说边界，尤其说「不要什么」**。「只改这一个页面，别动其他地方」——这听起来像废话，但 AI 默认会做它认为相关的一切改动，而软件里「相关」的范围经常超出你的想象。划清边界就是第四章里「解释后果」的镜像：现在轮到你说清后果了。
- **一次改一件事**。第五章讲表单要「一次只问一个问题」，对 AI 也一样：一句话里塞五个需求，它会在你意想不到的顺序里组合它们；拆成五轮，每轮你都能验收，错了也只错一处。

看出来了吗？这三条全是上半本书的原则，只是主客互换：默认值和解释，变成了你说的「验收标准」；一次一问的对话节奏，变成了「一次改一件事」；容错，变成了你要容忍 AI 猜错，并通过下一次对话纠正它。

大白话

大白话：以前你是用户，软件怎么对你好取决于别人；现在你是「甲方」，AI 怎么对你有用取决于你把需求说成什么样。上半本书是教乙方怎么做人，这一章是教你别当一个糊涂甲方。

你说不清的地方，正是你还没想清的地方

这是 vibe coding 最意外、也最有价值的副作用：**它逼你把需求想清楚。**

跟人类程序员提需求时，模糊可以被掩盖——程序员会开会、会追问、会基于经验脑补，很多模糊在来回扯皮中被消化了，你甚至没意识到自己的需求有洞。AI 不扯皮，它直接把你的模糊变成产品里一个具体的错误摆在你面前。你说「做得大气一点」，它就给你一个你可能不想要的「大气」。

很多 vibe coding 的用户都经历过这个顿悟时刻：改了五版都不对，终于坐下来把「我到底要什么」写成了三条具体的规则，AI 一次就做对了。AI 没有变聪明，是他自己的需求第一次变清楚了。

所以「你说不清它就做错」不只是 AI 的局限，它是一面镜子：**表达能力的边界，就是你能指挥的软件的边界。**这大概是这个时代最公平的规则之一。

但别急着自己扛下全部责任——好的沟通从来不是单方面的。下一章我们换一个方向看：AI 这一侧，又是怎么学会「懂你」的。

第十章 · AI 的同理心从哪来

AI 有同理心吗？

上一章把责任放在了用户一侧：说清楚，AI 才做得对。但这是个不公平的单方面结论——就像我们不能只要求用户「学会看说明书」。好的沟通是双向的，所以这一章问：**AI 这一侧，「懂你」的能力是从哪来的？**

先说结论：AI 没有人类意义上的同理心——它不会心疼你，也不会因为你在凌晨改需求而感动。但第四章教过我们，同理心在软件里从来不是情绪，是**可施工的机制**。AI 的「懂你」也一样，拆开看是三样东西：上下文、记忆、纠错循环。

上下文：它的「懂你」装在口袋里

AI 模型本身是一个巨大的、训练完成后就冻结的「静态大脑」。它对你这个人一无所知——它对你的全部了解，都来自当前这次对话里你们说过的话。这段话在行话里叫上下文（context），大白话就是「AI 的口袋里装了哪些纸」。

这个机制能解释很多 AI 的「谜之行为」：为什么对话长了它会「忘记」开头的要求？因为口袋容量有限，旧纸条被挤出去了。为什么你把需求文档贴给它之后它突然变聪明了？因为你往口袋里塞了好纸。为什么两个窗口问同一个问题答案不一样？因为两个口袋装的纸不同。

所以「AI 懂不懂你」很大程度上是个工程问题：口袋有多大、装了什么。这也是为什么好的 AI 工具会拼命帮你装纸——自动带上你的项目文件、你的历史对话、你的偏好设置。它做这些不是殷勤，是在替静态大脑弥补「生来不认识你」的缺陷。

记忆：从「这条对话」到「这个人」

上下文是一次性的：对话关掉，口袋清空。更进一步的「懂你」需要跨对话的记忆——记得你上个月说过「我们团队用中文写注释」，记得你偏好「先给结论再展开」。

现在的 AI 产品确实在往这个方向做。但这里有个值得警惕的权衡：记忆越好，越贴心，也越接近冒犯。还记得第四章说的那条红线吗——「猜对了省一秒，猜错了如果挡路，用户会

觉得被冒犯」。一个记错你偏好的 AI，比不记得你的 AI 更烦人，因为它会带着错误的自信反复猜错。**AI 的记忆和软件的预判一样，美德在于「顺手可拿」，而不是「替你决定」。**

纠错循环：真正的智能是接得住「不对」

第三样东西最重要，也最少被讨论：一个 AI 好不好用，不在于它第一次猜对多少，而在于你说「不对」之后，它怎么办。

回想上半本书的错误处理原则：承认出了什么事、摘清责任、给出下一步。好的 AI 在被纠正时做的事情惊人地像：它不会假装第一次的回答是对的，不会防御性地解释，而是把「不对」当成新信息，调整自己的理解，再试一次。vibe coding 的整个工作方式——说一句、看结果、再说一句——本质上就是把这个纠错循环开到最大转速。

大白话

大白话：不要看一个 AI「答对」时的表现，要看它「答错被指出」时的表现。就像一个员工，水平不看顺境，看被客户怼完之后的下一版方案。AI 的「同理心」，如果有的话，不在它的语气里，在这个纠错循环转得好不好里。

所以，AI 时代的同理心长在三个地方

总结一下这一侧的答案：用户感知到的「AI 很懂我」，背后是——口袋里的纸装得对不对（上下文）、跨对话的偏好记得恰不恰当（记忆）、被纠正后调得快不快（纠错循环）。

这三个机制都不是玄学，全是产品和工程的决定——和第四章的默认值、预判、容错、解释一脉相承。技术从按钮变成了语言，但同理心依然是施工出来的，不是祈祷出来的。

还有最后一块拼图：传统的团队靠日志和数据听见用户（第二、六章），那 AI 时代的产品团队呢？他们手里多了一样前所未有、也前所未有风险的东西——完整的对话记录。这是下一章，也是全书方法论的最后一块。

第十一章 · 对话记录是新的日志

用户的每句抱怨，现在都有原文了

回忆第二章：传统软件的团队想看懂用户，得从日志的「形状」里反推——原地打转是困惑，断崖是放弃。那些推断再准，也隔了一层：你看到的是「提交失败 3 次」，至于用户当时在想什么，全靠脑补。

AI 产品手里握着一个完全不同的东西：**对话记录**。用户不再是「点」出来的，是「说」出来的——他的需求、他的困惑、他语气里的不耐烦，全是原话。第二章里工程师要从日志形状里猜的东西，现在白纸黑字躺在那儿：「我不是要这个，我说的是纪念日提前一周，不是提前一天，你怎么又搞错了。」

这是产品史上前所未有的事：**用户研究的最大难题一直是「用户说不出自己要什么」，而现在用户的「要」和「不要」以最完整的形态被自动记录下来了**。冰山沉在水面下的那部分，第一次直接可见。

新的「日志阅读法」

但别以为这意味着理解用户变容易了。对话记录这个金矿有自己的读法，而且第二章的教训全都要重新学一遍：

- **还是找「挣扎的形状」**：用户在一句话上反复换措辞（「不是，我的意思是……」「换一种说法……」），就是新时代的「原地打转」——他没放弃，但沟通没通。某类需求反复以失败告终，就是新的「断崖」。
- **还是小心知识的诅咒**：工程师读对话记录时依然戴着知情者眼镜——「这需求明明一句话就能说清」。对，对你是一句话，对一个开花店的人不是。诅咒换了个场地，还在营业。
- **沉默的大多数依然存在**：vibe coding 里更多的人是「试了三次不对，关掉网页，再没回来」。他们的对话记录短得让人心疼——三句话，一个人对 AI 的全部信任额度就用完了。

金矿旁边的坑：隐私与傲慢

对话记录是前所未有的馈赠，也带来前所未有的责任，两个坑必须点名：

第一个是隐私。用户对 AI 说的话，常常比对任何界面都私密——因为语言的低门槛让人放松警惕。对话记录不是普通的点击流，它可能装着病历、工资单、没处说的烦恼。把它当成「免费的用户研究素材库」而不设边界，是这个行业正在集体面临的大考。看见一切，不等于有权看一切。

第二个坑更隐蔽，可以叫「答案的傲慢」。传统软件的错误是明目张胆的（报错、白屏），团队知道自己错了；AI 的错误经常是「看起来对了」——代码能跑，页面能开，但做的不是用户要的事。如果团队只看「任务成功率」这种指标，会得到一片祥和的假象。就像第二章说的：系统层面一切正常，用户层面水深火热——只不过现在「正常」的定义里还要加上「AI 觉得自己答对了」。

迭代飞轮的新形状

跨过这两个坑，AI 时代的沟通回路确实进化了。传统回路是：用户挣扎 → 团队从日志推断 → 排期改版 → 几个月后上线。AI 时代的回路可以是：用户在对话里挣扎 → 团队从对话记录里找到失败的句式 → **改进 AI 的理解能力或交互引导** → 下次对话当场受益。反馈周期从月级压到天级甚至小时级。

注意改进的落点变了：以前改的是按钮和页面，现在改的常常是「AI 怎么追问一句」「怎么在不确定时先确认再动手」「怎么把术语翻译成成人话再问一遍」。**产品设计的对象，从界面变成了对话本身。**这大概就是「软件即沟通」这个命题最字面的实现。

全书只剩最后一个问题了：从打孔卡片到命令行，从图形界面到指尖触摸，再到今天的自然语言——界面一直在变，有没有什么东西从来没变过？有。最后一章讲它。

第十二章 · 技术的尽头是关切

界面六十年，一直在朝人走过来

把软件的历史拉长看，会看到一条非常清晰的单行线：

- **打孔卡时代**：人要把自己的意图翻译成卡片上的洞，错一个洞，几小时的排队作废。人完全迁就机器。
- **命令行时代**：人学机器的语言——背命令、记参数。机器前进了一小步，人还在爬山。
- **图形界面时代**：机器学会了人的一部分直觉——东西是「窗口」里的「文件」，操作是「点」和「拖」。这是本书上半部分的主场。
- **触摸时代**：界面退得更远，三岁小孩划两下就会用。
- **对话时代**：也就是现在——机器终于开始学习人最主要的沟通方式：语言。vibe coding 是这条线在「制造软件」这件事上的延伸。

六十年，方向从来没变过：**每一次界面的进步，都是机器多走一步、人少走一步**。迁就越来越少，越来越接近「你照平时的样子说话做事，它就能懂」。

但有一个东西从来没变

界面变了一代又一代，这本书从第一章讲到现在，讲的其实是同一个东西在各个时代的化身：

在图形界面时代，它叫默认值、友好报错、撤销、表单的节奏；在 AI 时代，它叫上下文、记忆、纠错循环、说清验收标准。名字全换了，内核一个没变——**做系统的人，有没有站在用系统的人那边，把他没说的话、没料到的错、不敢点的按钮，提前想到。**

我们管这个叫共情。这本书花了很多篇幅证明它不是软绵绵的态度：它是日志里读出来的两分半钟挣扎，是报错文案过的三问检查，是花店老板娘写给 AI 的三条验收规则。共情在每个时代都穿着那个时代的工装，但里面是同一颗心。

AI 时代，这颗心更值钱，而不是更不值钱

有一种流行的担心：AI 什么都能做了，做软件的人还剩什么？这本书的答案是：剩下的恰恰是最重要的那部分。

写代码这个环节在贬值，但「知道该写什么」「知道用户在哪卡住」「知道一句需求背后真正的意图」在升值。AI 把执行的门槛拆了，于是所有的竞争都向沟通的两端集中——对需求理解得多深，对人关心得多真。第二章那个从日志里读出呼救的能力，第六章那个听见沉默的能力，在 AI 时代不是过时了，而是成了区分度。

甚至 vibe coding 的用户也一样：花店老板能不能做成她的工具，最终不取决于 AI——AI 对所有人一视同仁——而取决于她有多了解自己的顾客、多清楚自己的生意。**工具越强大，人与人之间的差别越回到人本身。**

最后，回到那场对话

这本书开始于一个画面：你点了按钮，什么都没发生，你觉得自己在对着一堵墙说话。

现在我们可以把这个画面补完了。墙不存在。每一次点击、每一句话，都真的会被听见——区别只在于，墙的另一边坐着什么样的团队：是把日志当流水账的，还是把日志当呼救的；是把报错写成房间号的，还是过三问检查的；是看到断崖率无动于衷的，还是心里装着那个凌晨一点放弃的用户的。

软件的界面会继续进化，也许再过十年，连「对话」都会显得笨重。但只要软件还是人做给人用的——哪怕中间隔着 AI——这场沟通就在继续，而它唯一的评判标准从来没变过：

屏幕那一头的人，有没有被当成一个具体的人。

这就是「软件即沟通」最后要说的话。技术的尽头不是更强的技术，是关切。