Bell Labs
文档蒸馏

从 *Blit* 到 *UTF-8* 的工具哲学

"The masters of the laser, the transistor,
the microwave-background, unix and many other
of the greatest inventions and discoveries
of the 20th century."
— doc.cat-v.org/bell_labs/

"A clear program is not made any clearer
by such presentation, and a bad program
is only made laughable."
— Rob Pike，*Notes on Programming in C*

"Disgusting, no? But it compiles and runs just fine."
— Tom Duff，发明 Duff's Device 之后

Bell Labs Computing Science Research Center

一个被一再追忆的研究所——

晶体管、激光、Unix、C、Plan 9、UTF-8 的发源地。

本书把 doc.cat-v.org/bell_labs/ 留下的论文目录

蒸馏为一本可以从头读到尾的小书。

蒸馏自 doc.cat-v.org/bell_labs/ 及其子页面

2026 年 5 月 15 日

目录

前言：为什么是 Bell Labs

"This talk is a polemic that distills the pessimistic side of my feelings about systems research these days."
— Rob Pike，*Systems Software Research is Irrelevant*

Bell Labs 是计算机历史上最难以复制的研究机构之一。它曾在一座普通的新泽西楼里同时养着八个诺贝尔奖、晶体管、激光、信息论、Unix、C、awk、yacc、sed、make 的继承者 mk、sam、acme、Plan 9、Inferno、UTF-8、Go ——而所有这一切发生时，研究员们使用着同一台食堂、同一组打印机、同一个 `cm.bell-labs.com` 主机名。计算机的 "黄金二十年" 并非分散在很多机构里——它在 1127 室的走廊里集中发生。

`cat-v.org` 圈子（`doc.cat-v.org`、`harmful.cat-v.org`、`quotes.cat-v.org`）以一种近乎执拗的态度持续维护着这段历史的电子档案。它的 `/bell_labs/` 目录是一份极小但极有品味的清单：二十余篇论文，几段视频，一份 "Bell Labs Innovations" 的奇怪歌曲歌词，以及一封 1983 年 Tom Duff 关于他自己刚发明的循环展开技巧的电子邮件。看上去散乱，实则连贯：它们一起描绘出一种工程文化——*用文本传输；用工具组合；用接口而非框架；用人类可读的格式而非二进制；用单一作者写明白一件事，而非用委员会写一套规范。*

本书把这个目录蒸馏为一本可以从头读到尾的小书。每一篇文档对应一章，按作者归类：

- 第一部分 Rob Pike——他写过的论文几乎占了 `bell_labs/` 的一半。从位图终端 Blit、并发窗口系统、透明窗口系统，到结构化正则、sam 命令语言、UTF-8 的诞生史、Face the Nation 的人脸文件服务器、Squeak 的鼠标语言、《为什么系统软件研究无关紧要》的演讲（utah2000）、《好、坏、丑陋》的 Unix 时钟一千兆秒纪念演讲，以及 1989 年那篇被无数程序员引用的《Notes on Programming in C》。
- 第二部分 Brian Kernighan——以批评者的姿态出现：《Why Pascal is Not My Favorite Programming Language》，以及与 Van Wyk 合作的脚本语言性能比较《Timing Trials》。
- 第三部分 Ken Thompson——Plan 9 上的新一代 C 编译器，以及与 Henry Baird 一起做的国际象棋谱字符识别。
- 第四部分 其他人——Andrew Hume 的 `mk`、David Presotto 的 Upas 邮件系统、Doug McIlroy 的 *Squinting at Power Series*、Luca Cardelli 的 *Crabs*、Tom Duff 的 Duff's Device。
- 附录 Bell Labs Innovations 主题曲歌词，以及一份 Dennis Ritchie 速写。

> 阅读方式建议。这是一本可以*跳读*的书。如果你只想要思想史，按章节顺序读 Pike 部分即可；如果你正在写代码，《Notes on Programming in C》、《Structural Regular Expressions》、《Mk》、《Duff's Device》和《Why Pascal is Not My Favorite Language》会立刻派上用场；如果你只想消遣，去读 utah2000、UTF-8 的诞生史和 Innovations Song 歌词。

每章末尾给出 `cat-v.org` 上原始文件的路径，方便检索原文 PDF。我们不重写论文，也不替代论文；本书是论文的*索引、提要与中文导读*，让那些不愿一头扎进十几篇英文 PDF 的读者，可以先从一本书的厚度里把握这块土地的形状。

第一部分

Rob Pike：多路复用一切

Chapter 1

Blit：一台多路复用的位图终端

“The Blit is a programmable bitmap graphics terminal designed specifically to run with the Unix operating system.”
— Rob Pike，1983

1.1 背景：被终端束缚的多任务系统

1983 年的 Unix 已经是真正意义上的多任务、多用户系统，但它的“外设”却仍是 1970 年代的字符终端：VT100、Teletype 之类，每秒几十到几千字符，单一光标，黑底白字。“多进程”在内核里运转着，却必须从一根 RS-232 线挤进单个屏幕；用户每次只能盯着一个 shell。Unix 的多路复用能力被终端这个瓶颈彻底掩盖了。

Rob Pike 与 Bart Locanthi 在 AT&T Bell Labs 设计的 **Blit** 直接针对这一点：把多窗口、位图绘制、本地交互全部塞进终端，用一根串口连接到主机，但每个窗口对应主机上的一个独立 `shell`/进程。对主机而言，多个 Blit 窗口看起来就像多个伪终端；对用户而言，每个窗口就是一台独立的 Unix。

1.2 摘要要点

> *The Blit is a programmable bitmap graphics terminal designed specifically to run with the Unix operating system. The software in the terminal provides an asynchronous multi-window environment, and thereby exploits the multiprogramming capabilities of the Unix system which have been largely under-utilized because of the restrictions of conventional terminals. …Because most of the functionality is provided by the terminal, the discussion focuses on the structure of the terminal’s software.*

论文要做的事情有三层：

1. 解释为什么要把窗口管理放到终端里——而不是放到主机上像 X11 那样做。

2. 描述用户接口：选区、菜单、按钮、滚动条等今天我们觉得理所当然的部件，在那时是要论证的。

3. 描述实现——终端固件结构、主机端 `mpx` 多路复用器、各种新颖应用（异步打印、调试器在另一个窗口里运行、远程编辑）。

1.3 为什么 Blit 重要

Blit 是窗口系统作为 **Unix** 工具这一观念的源头。它直接催生了：

- Pike 的 mux（Eighth Edition Unix 上的窗口系统）；
- Pike 后来在 Plan 9 上的 8½ 和 rio；
- *Window Systems Should Be Transparent*（本书第 3 章）所批评的“庞大窗口系统”的反例；
- Inferno 与 Plan 9 上至今仍可用的 acme、sam 的图形交互范式。

> 现场录像。 `cat-v.org` 还保留了 1982 年 Rob Pike 与 Bart Locanthi 演示 Blit 用户界面的视频片段（YouTube ID `emh22gT5e9k`）。今天看回 1982 年的位图终端用滑鼠选中文字、弹出菜单、把进程切到后台，会非常意外地觉得现代——我们今天的图形终端在概念上几乎没有进步过 **Blit** 一步。

原文路径。 `/bell_labs/blit/blit.pdf`

Chapter 2

并发的窗口系统：当窗口系统本身用 newsqueak 写出

"This is the software tool approach applied to windows."
— Rob Pike，*A Concurrent Window System*

2.1 用一种并发语言写窗口系统

如果你阅读早期 X Window 系统的源码，会立刻感到事件循环的"黑潭"形态：回调函数、状态机、各种 `XEvent` 类型 switch 分发——本来是一个明明并发的系统，却必须被压平到单线程的事件分发循环里。Pike 对此的回答是：为什么不用一门并发语言直接表达它？

这篇论文用 **newsqueak**（Pike 自己设计的一种 CSP 风格的小语言，是 Squeak 的延伸，后来直接影响了 Limbo 和 Go 的 `chan`）写出一个完整的窗口系统，客户程序通过同步通道和窗口系统通信，而不是通过回调注册。

2.2 摘要核心

> *When implemented in a concurrent language, a window system can be concise. If its client programs connect to the window system using an interface defined in terms of communication on synchronous channels, much of the complexity of traditional event-based interfaces can be avoided. …In particular, the window system itself may be run recursively to implement subwindows for multiplexed applications such as multi-file text editors. This is the software tool approach applied to windows.*

2.3 两个深远的设计后果

接口 = 通道协议。 窗口系统不提供 “API”，它提供一组通道。任何能在这些通道上读写的程序都是合法客户端。这种设计让把窗口系统自己作为一个客户端跑在另一个窗口系统里变得平凡——也就是 “递归窗口”，后来在 Plan 9 的 rio 里成为标准能力。

复合 = 组件。 既然每个交互组件都通过同样的通道协议工作，你就可以把它们管道串起来。一个多文件编辑器不是 “一个大程序”，而是 “一个窗口子系统 + N 个独立编辑会话”。

这条思路后来直接变成 Plan 9 的 `rio` 与 acme 的设计原则。Go 语言里 “Don’t communicate by sharing memory; share memory by communicating” 这一句口号，源头就在这里。

延伸阅读。 *Rio: Design of a Concurrent Window System*（Plan 9 第三版）；*Window Systems Should Be Transparent*（下章）。

原文路径。 `/bell_labs/concurrent_window_system/concurrent_window_system.pdf`

Chapter 3

窗口系统应当是透明的

"Commercial UNIX window systems are unsatisfactory. …Their clumsy user interfaces clutter the view of the operating system. A good interface should clarify the view, not obscure it."
— Rob Pike，*Window Systems Should Be Transparent*

3.1 论点：好接口让你看清下面的东西

这是 Pike 最具"刀子"气质的几篇短文之一。它把 X、SunView、Motif 这些 1980 年代末的商用 Unix 窗口系统一一摆上桌——它们的接口繁琐、复杂、装饰过多，把下面那个本来清爽的 Unix 内核盖得严严实实。Pike 主张：

> *A good interface should clarify the view, not obscure it.*

透明（transparent）在这里有两个意思：

1. 视觉上简洁，不要在屏幕上塞满工具条、按钮、装饰边框。
2. 语义上不增加新概念——你不应该需要学一套新的"窗口对象模型"才能写一个窗口程序。窗口和文件描述符应该一样轻。

3.2 Mux：作为反例的"好"

Pike 把 Bell Labs 自己内部的 mux 窗口系统作为案例来分析。mux 跑在已经被淘汰的硬件上，所以没人会拿它去卖；正因此它也没有被市场压力扭曲。它的特征：

- 没有窗口装饰，没有标题栏，没有最小化按钮；
- 选区与菜单都由滑鼠手势触发，屏幕在不操作时干净如纸；
- 实现极小；
- 用户感觉"像在用一个更安静的 Unix"，而不是"在用一个窗口系统"。

3.3 这种态度的延续

这篇论文是 cat-v.org 整个 “cat -v considered harmful” 文化的雏形。Pike 在大约同一时期还写了一篇更出名的 *UNIX Style, or cat -v Considered Harmful*，那是 harmful.cat-v.org 网站名字的来源。本书第 9 章会讨论那篇 “系统软件研究无关紧要” 的演讲——透明窗口系统的看法、cat -v 的看法、utah2000 的看法，它们其实是同一种审美在不同年份的回响：少装饰、多内涵、让工具去掉自己。

原文路径。 /bell_labs/transparent_wsys/transparent_wsys.pdf

Chapter 4

结构化的正则表达式

"The current UNIX® text processing tools are weakened by the built-in concept of a line."
— Rob Pike，*Structural Regular Expressions*, 1987

4.1 Unix 文本工具的隐藏假设："行"

`grep`、`sed`、`awk`、`cut` 这些 Unix 经典工具是"面向行"的——它们把输入按 `\n` 切开，每一行作为一条独立的小宇宙处理。绝大多数日常工作里这没问题；但当文件本身有多行结构时——注释块、函数体、HTML 标签、多行字符串——面向行的工具立刻变得笨拙。你必须不停地 `sed -n '/start/,/end/p'` 或者祈祷正则跨行不要崩。

4.2 Pike 的提议：用正则定义"结构"

Pike 在 1987 年提出，应当用正则表达式同时描述两件事：

1. 文件中要操作的区域（结构）；
2. 在这些区域内要寻找的内容。

也就是说，把目前 `sed` 的"地址 + 命令"推广到"选择子 + 命令"，而选择子本身可以嵌套：先在整个文件里选出所有 C 函数体，然后再在每个函数体里选出所有 `return` 语句，然后再把这些 `return` 语句中的某个变量替换掉。所有这些都用一种统一的管道式表达。

> *Using regular expressions to describe the structure in addition to the contents of files has interesting applications, and yields elegant methods for dealing with some problems the current tools handle clumsily. When operations using these expressions are composed, the result is reminiscent of shell pipelines.*

4.3 sam 与 acme：理论落地

这套设计后来直接落地为两个工具：

- **sam**（Pike 的命令式编辑器）的命令语言；
- **acme**（Pike 的可编程编辑器/IDE）的内部模型；

它们的基本动作不是 “跳到第 100 行”，而是 “选中所有匹配某个结构正则的区段”；然后链式地在这些区段上做选中、替换、过滤、发送到外部命令。今天的 Plan 9 from User Space 把 sam 与 acme 都移植到了普通 Unix 上。

原文路径。 `/bell_labs/structural_regexps/se.pdf`

延伸阅读。 本书第 5 章对 sam 命令语言的导读。

Chapter 5

sam 命令语言教程

"sam is an interactive text editor with a command language that makes heavy use of regular expressions."
— Rob Pike，sam tutorial

5.1 sam 的两种界面

sam 是 Pike 写的 "ed 之后的下一步"。它有两套界面：

- `sam -d`：纯字符模式，行为像 `ed` 的加强版，命令语言完全一样；
- GUI 模式（最初是在 Blit 上）：在屏幕上以位图显示文本，鼠标选区，但是命令窗口里跑的是同一种命令语言。

这一点非常重要：很多现代编辑器把 "可视编辑" 和 "可脚本化" 当作两套独立机制；sam 让它们用同一套语法。

5.2 命令语言核心

教程介绍的 sam 命令语言基于结构化正则（上一章）：

```
,p            % 打印整个文件
/foo/p        % 打印所有匹配 /foo/ 的行
,x/regex/cmd  % 在整个文件里对每个 regex 匹配执行 cmd
,y/regex/cmd  % 在整个文件里对 regex 之间的部分执行 cmd
,s/old/new/g  % 全局替换
```

`x` 与 `y` 是 sam 的精髓——它们让命令遍历结构，而不像 `ed`/`vi` 那样只能一次跳到一个位置。你可以构造出 "在每个 C 函数体里把每个出现的 `old_name` 改成 `new_name`" 这种链式操作，全在一行命令里。

5.3 为什么今天还值得读 sam 教程

即使你已经被 vim、emacs、acme、VS Code 包围，sam 的命令语言依然是少数几个“结构化文本编辑”范式的清晰示范——它的接班人 acme、Helix 的 selections-first 模型、Kakoune、甚至 vim 的 Visual Block，都在这条线上不同的位置。

延伸阅读。 *sam.cat-v.org* 与 acme 在 `acme.cat-v.org`；*The Text Editor sam*（Plan 9 第四版论文）。

原文路径。 `/bell_labs/sam_lang_tutorial/sam_tut.pdf`

Chapter 6

Face the Nation：把人脸放进文件系统

"The face server is implemented as a variant of the network file system, so the images are stored on one machine but appear to be regular files in the file systems of all the machines on the network."
— Pike & Presotto

6.1 一个奇怪的 Bell Labs 系统

Face the Nation 是 Pike 与 David L. Presotto 1980 年代末的一篇论文，描述了他们在 Eighth Edition Unix 上做的一个 "人脸文件服务器"。Computing Science Research Center 的成员们把彼此的脸（以及电子邮件通信对象的脸）扫描下来，存进一台中心服务器；但访问方式不是数据库查询，而是把网络服务器伪装成文件系统。

> *Digitized images of the faces of members of AT&T Bell Laboratories' Computing Science Research Center, and of many of their electronic mail correspondents, are stored in a network "face server" accessible from the Eighth Edition machines at Murray Hill. The faces decorate line printer banners, announce the arrival of mail, and perform a variety of other Information Age services. …This method of presenting the information is attractive for databases with inherent structure, as it permits standard Unix system utilities to access, manipulate and maintain the database.*

6.2 这是 Plan 9 的 "一切都是文件" 之前奏

读这篇论文的时候，请记住时间：那是 1980 年代末。"Web 服务"、"REST API"、"S3 bucket" 都还不存在。而他们已经在把一个分布式图像数据库做成 "看起来像一棵目录树"，让用户用 `cat /face/pike`、`cp /face/* banner/` 来工作。

这套思路在 Plan 9 上变成了 *9P* 协议与 "一切都是文件系统" 的设计原则——窗口系统、网络栈、CPU 调度、声音设备、X 服务器、Web 浏览器，都被实现为某个挂载在某条路径下

的文件系统。`cat-v.org` 的 `9p.cat-v.org` 子站点是这个理念的现代档案库。

原文路径。 `/bell_labs/face_the_nation/face.pdf`（亦有 HTML 版与 PostScript 版）

Chapter 7

Squeak：用一门小语言和鼠标说话

"Graphical user interfaces are difficult to implement because of the essential concurrency among multiple interaction devices."
— Cardelli & Pike

7.1 鼠标、键盘、按钮之间的并发

GUI 实现的本质难点不是绘图，而是并发：鼠标、键盘、定时器、按钮、滚动条……这些事件源各自有节奏，需要被合并成一份对用户有意义的"交互序列"。1980 年代主流的做法是回调（callback）：注册一堆 `onClick`、`onMouseMove` 之类的函数，让框架替你串起来。代价是状态被切碎在十几个函数里。

Luca Cardelli 与 Rob Pike 提出的 **Squeak** 是一门小语言——请不要把它和后来 Alan Kay 实验室的同名 Smalltalk 衍生品混淆——它从 CSP 的角度直面这个并发难题：

> *Squeak is a user interface implementation language that exploits this concurrency rather than hiding it, helping the programmer to express interactions using multiple devices.*

7.2 从 Squeak 到 newsqueak、Alef、Limbo、Go

Squeak 是 Pike 一连串"通信顺序进程小语言"的最早成员。后来他写出 newsqueak 把它推广到更广义的并发应用（见第 2 章"并发窗口系统"与第 18 章"Squinting at Power Series"），然后是 Alef（Plan 9 上的并发 C），然后是 Limbo（Inferno 的语言），最后是 Go——其中 `chan` 与 `select` 是从 Squeak 一路传下来的句法器官。

7.3 为什么这个 Squeak 不是 Alan Kay 的 Squeak

只是名字撞车。Alan Kay/Dan Ingalls 的 Squeak 是 1996 年开始的 Smalltalk-80 重实现项目。Pike-Cardelli 的 Squeak 比它早将近十年，与 Smalltalk 无关——它是面向设备并发的小型函数式语言。

延伸阅读。 *Limbo Programming Language* 与 *Alef Language Reference Manual*。

原文路径。 `/bell_labs/squeak/squeak.pdf`

Chapter 8

The Hideous Name：网络邮件地址的灾难

`research!ucbvax!@cmu-cs-pt.arpa:@CMU-ITC-LINUS:dave%CMU-ITC-LINUS`

"I cannot tell what the dickens his name is." —Shakespeare, Merry Wives of Windsor, II.ii.20.

— Pike & Weinberger

8.1 一个真实出现过的邮件地址

上面那行字符是 *The Hideous Name* 一文开头的一个真实邮件地址，出自卡耐基梅隆大学的邮件路由器，里头同时混合了 UUCP 的 `!` 路由、ARPA 的 `@` 域名、以及 `%`、`:` 等当年各家网关用来表达"下一跳"的种种分隔符。1985 年前后的互联网邮件系统就是这样工作的——一条邮件可能要被来回翻译四五次才能到达目的地。

8.2 一篇看似随意、实则严肃的论文

Rob Pike 与 P.J. Weinberger 写这篇论文是为了说一件简单的事：

> *The principles of good naming in computing have been known for decades. …Unfortunately, the situation with internetwork mail addresses is not as satisfactory.*

文件系统给出了一个早就被验证过的"命名好示范"：当你引入网络共享时，你只要把远程文件挂在本地路径下；然后所有现有的 Unix 工具——`cat`、`cp`、`ls`——都立刻能用，不需要任何"网络变体"。而邮件地址在 1980 年代恰恰反其道而行之，每一种网络都要求所有上下游知道自己存在并妥协。

文章用语异常风趣，但底下是一条工程原则：

> 命名应该统一、本地、与工具同源。让远程对象看上去和本地对象一样，这样所有处理本地对象的工具都自动能处理远程对象。

8.3 它的回响

这种命名学一路传到 Plan 9 的 9P：“一切都是文件” 意味着 “一切都用文件路径命名”——`/net/tcp/0/data`、`/mnt/9fans/foo/index.html`、`/proc/12345/mem`……你不必学十种 URL scheme，因为只有一种 scheme：路径。

原文路径。 `/bell_labs/the_hideous_name/the_hideous_name.pdf`

Chapter 9

UTF-8 的诞生：一个晚上的成果

"UTF-8 was designed, in front of my eyes, on a placemat in a New Jersey diner one night in September or so 1992."
— Rob Pike

9.1 背景：1992 年的字符集焦虑

1980 年代末到 1990 年代初，世界各国的人意识到 7 位 ASCII 不够用，ISO 10646 和 Unicode 在并行推进，但谁也不喜欢 16 位定宽编码：对于 Unix 工具而言，定宽 16 位编码里大量的 NUL 字节直接破坏 `strlen`、`open`、文件名解析。"像 ASCII 一样安全使用" 与 "能容纳所有字符" 之间需要一座桥。

X/Open 在 1992 年起草了一个叫 FSS/UTF 的多字节编码方案。**Rob Pike** 与 **Ken Thompson** 看过之后，觉得它差最后一脚——特别是从字节流中部任意一点开始恢复字符边界这个性质——便提出在一周内重新设计并实现。故事的真相通过 `cat-v.org` 上保留的 Pike 那封 2003 年的电子邮件留存下来：

> *What happened was this. We had used the original UTF from ISO 10646 to make Plan 9 support 16-bit characters, but we hated it. We were close to shipping the system when, late one afternoon, I received a call from some folks, I think at IBM …They wanted Ken and me to vet their FSS/UTF design. …So we went to dinner, Ken figured out the bit-packing, and when we came back to the lab after dinner we called the X/Open guys and explained our scheme. We mailed them an outline of our spec, and they replied saying that it was better than theirs …*

9.2 他们的六条设计准则

Pike 写给 Russ Cox 的回信里包含了 1992 年 9 月 8 日凌晨 3:22 ET 寄给 X/Open 的提案。其设计准则有六条（直接翻译）：

1. 与历史文件系统兼容——禁止零字节与 `/` 出现在多字节序列内部；
2. 与现有程序兼容——已经是 ASCII 的字节不会在多字节编码里再出现；
3. 易于与 **UCS** 互转；
4. 首字节决定字符长度——读到第一个字节就知道还要读几个；
5. 不要在字节数上挥霍；
6. 可同步——从字节流中部任意一点开始，丢弃至多一个字符即可定位下一个字符的起点。

第 6 条就是 Pike 与 Thompson 加进去而 X/Open 原始方案里没有的那一条；正是它把 UTF-8 从“又一种编码”变成了一个真正适合 Unix 流式工具的编码。

9.3 编码规则

从那封邮件里直接抄下来的表：

```
Bits  Hex Min   Hex Max   Byte Sequence in Binary
1   7  00000000 0000007F  0vvvvvvv
2  11  00000080 000007FF  110vvvvv 10vvvvvv
3  16  00000800 0000FFFF  1110vvvv 10vvvvvv 10vvvvvv
4  21  00010000 001FFFFF  11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
5  26  00200000 03FFFFFF  111110vv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv
6  31  04000000 7FFFFFFF  1111110v 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv
                                   10vvvvvv
```

后来 Unicode 把范围收紧到 21 位 / 4 字节以内，但这张表的 1-4 行至今仍是你打开任何 Web 页面、任何源码文件的底层契约。

9.4 时间线（来自 Russ Cox 在邮件归档里的考古）

- **1992** 年 **9** 月 **2** 日 **23:44** ——Ken 写下 `/usr/ken/utf/xutf` 的最初草稿；
- **1992** 年 **9** 月 **4** 日 **03:37** ——Ken 给 Rob 邮件：“you might want to look at /usr/ken/utf/utf.c”；
- **1992** 年 **9** 月 **8** 日 **03:22** ——修订完毕的提案邮寄给 `xojig@xopen.co.uk`；
- 几天之内——Plan 9 完整切到 UTF-8 编码。

不到一周完成设计、参考实现、操作系统改造、首次发行——这就是当时 1127 室的速度。

原文路径。 `/bell_labs/utf-8_history.html`

Chapter 10

utah2000：系统软件研究无关紧要

"If systems research was relevant, we'd see new operating systems and new languages making inroads into the industry, the way we did in the '70s and '80s."
— Rob Pike，犹他大学 2000 年讲座

10.1 一封"檄文"

2000 年 8 月，Rob Pike 在犹他大学做了一个标题极挑衅的讲座：*Systems Software Research is Irrelevant*（系统软件研究无关紧要）。讲稿在 `cat-v.org` 上以纯 HTML 形式留存——值得每一个研究系统软件、操作系统、语言、网络的人完整地读一遍。这里把核心论点蒸馏出来。

定义

Pike 自己给出的定义：

- **Systems**：操作系统、网络、语言——把程序连起来的东西。
- **Software**：字面意思。
- **Research**：主要指学术研究——大学加少数工业实验室。
- **Is**：现在。不是十年前；也希望不是十年后。
- **Irrelevant**：对工业界没有影响。

论题

> *Systems software research has become a sideline to the excitement in the computing industry. When did you last see an exciting non-commercial demo?*

10.2 为什么这件事发生了

Pike 列了一长串原因，本书把它整理成几条：

PC。 1980 年代的系统研究围绕新架构（RISC、iAPX/432、Lisp Machines）展开。PC 变成唯一架构后，“为不同架构做可移植” 这件事没了；可发挥的范围被压扁了。

微软。 容易当替罪羊。但它不是真问题，只是症状。

Web。 Web 是 1990 年代被工业拿走的成果。研究界做了一堆论文（缓存、代理、服务器架构），影响力却很小。

标准。 TCP/IP、HTTP、HTML、XML、CORBA、Unicode、POSIX、NFS、SMB……Pike 估算 Plan 9 里 90–95% 的工作是为了向外部强加的标准低头。留给创新的空隙非常小。

正统。 今天毕业的博士都用 Unix、X、Emacs、TeX。“二十年前一个学生会接触到多种操作系统，各有优劣”——这种暴露面被磨平了。“Narrowness of experience leads to narrowness of imagination.”

规模变了。 现代系统的设计–实现–发行周期是五年，这超出大多数研究生学位与小型课题组的范围。结果是 “不要建造，去测量”——大家不再做系统，而是做现成系统的性能比较。

Unix 杀死了系统研究。 1970 年代末、1980 年代初已经有人说 Unix 杀死了操作系统研究，因为没人会再去试别的；当时 Pike 不信，写这篇时已经 “勉强承认”。

Linux 是 “学术的 Windows”。 Linux/gcc/Netscape 这套 “神圣三位一体” 是因为它们*不是*微软才被推崇，而不是因为它们*是什么*。

创业公司。 它们与学术界争夺资金、人才、学生；让所有研究都被压在 “一年内 IPO” 的时间尺度上。

奶奶上网了。 工业界把火力集中在终端用户与设备上，“可编程性——曾经是计算机的大想法——已经被丢在了路边”。

10.3 他给出的 “事可做”

> *Go back to thinking about and* building systems. *Narrowness is irrelevant; breadth is relevant: it's the essence of* system.
> *Work on how systems behave and work, not just how they compare. Concentrate on interfaces and architecture, not just engineering. Be courageous. Try different things; experiment.* Try to give a cool demo.

具体方向他列出了五个候选：

1. **GUI。**至今只有一种 GUI 被严肃尝试过，最好的想法来自 1970 年代——而且*在退化*。
2. 组件架构。只有 Unix 管道是个真成功；交互式与分布式应用也应该可以拼装。
3. 分布式计算。未来在这里，但语言界做得太少。

4. 推送 **vs.** 拉取。Web 把所有东西强迫成“用户去取”；让数据来找用户。

5. 系统管理。不光彩但巨大，市场也巨大。

10.4 二十多年后回看

写这篇笔记是 2025/2026 年——utah2000 已过了二十几年。Pike 的诊断或许仍然准确，或许已经过时；但有几件事必须看到：

- **NixOS / Plan 9 / 9front** 仍是 Pike 期望的那种“有勇气”的系统，它们没有占领工业界，但有自己的脉络与人群——`cat-v.org` 自己就是这条线的存活证据。
- **Go** 出现在 utah2000 之后大约 9 年，Pike 自己是设计者之一。Go 把 utah2000 抱怨的“语言界对分布式做得太少”变成了一条产品线。
- **WebAssembly**、**Wayland**、**io_uring** 都可视为对 utah2000 中某些条目的迟到回应。

原文路径。 `/bell_labs/utah2000/utah2000.html`(HTML 全文);`/bell_labs/utah2000/utah200`
延伸阅读：`harmful.cat-v.org/cat-v/`（*UNIX Style, or cat -v Considered Harmful*）。

Chapter 11

The Good, the Bad, and the Ugly：Unix 时钟十亿秒

Time: Sept 8–9 2001
Unix time: **+1000000000s**
Location: Copenhagen

11.1 一场极妙的演讲场合

2001 年 9 月 9 日 01:46:40 UTC，Unix epoch 计数器越过了 1,000,000,000——这一秒被一群 Unix 老人选作庆生的时机。活动设在哥本哈根。Rob Pike 那天讲的题目是 *The Good, the Bad, and the Ugly: The Unix Legacy*。

`cat-v.org` 上只剩下幻灯片 PDF；本节把幻灯片的论点凝成一份简短的导读。

11.2 The Good

Pike 列出的“好”是 Unix 留给世界的东西：

- 文本流的统一抽象。所有工具的输入输出都是字节流，管道可以把它们组合起来。
- 文件系统层级。一棵树管理一切，从设备到配置到二进制。
- **shell** 即用户编程语言。命令行不仅是 UI，它本身可编程。
- 进程模型。`fork/exec` 简单到令人发指，却足够支持几十年的多任务计算。
- 小、组合、人类可读。awk、sed、grep、yacc、tr、tee、…

11.3 The Bad

但同一时间，Unix 也带来了一些“坏”：

- **C** 与不安全。buffer overflow、null pointer、隐式整数转换……半个世纪的安全事件源头。

- **stdio** 与 **\0** 结尾的字符串。做国际化、做字节流处理、做并发，全都被这套早期决策拖累。
- **shell** 的元字符地狱。空格、*、!、$…一不小心 `rm -rf` 就把别处的东西删了。

11.4 The Ugly

“丑” 则是 Unix 后来在工业界扩张时的代价：

- 标准战争。POSIX、SVR4、BSD、SUS 各家拉锯。
- **X Window** 的庞大与僵硬。
- 越来越复杂的内核接口——`ioctl` 的废墟、proc 文件系统的不一致、…
- **Unix** 杀死操作系统研究（与 utah2000 同一论点）。

原文路径。 `/bell_labs/good_bad_ugly/slides.pdf`

Chapter 12

Notes on Programming in C：1989 年的 C 编程哲学

"Under no circumstances should you program the way I say to because I say to; program the way you think expresses best what you're trying to accomplish in the program. And do so consistently and ruthlessly."
— Rob Pike，1989 年 2 月 21 日

12.1 这是一份课程讲义

1989 年 2 月 21 日，Rob Pike 在某门课上分发了一组短文——后来以 "pikestyle" 这个文件名在网上流传，`cat-v.org` 上 `/bell_labs/pikestyle` 完整保留了它的 HTML 版本。它不是论文，而是一组随笔，把 Pike 自己对 C 编程风格的观点压成了若干条短文。

由于这份文件本身就足够紧凑而每一段都值得逐字保留，本章按原文章节顺序简短重述其要旨，并在段后给出原文里最锋利的句子。

12.2 Typography（排印）

> *A program is a sort of publication. It's meant to be read by the programmer, another programmer (perhaps yourself a few days, weeks or years later), and lastly a machine.*

要点：

- 程序首先是给人看的；其次才是给机器看的。
- 不要被 "pretty printer" 与 Algol-68 报告式的排版迷住——清晰的程序不会因这种装饰而更清晰，糟糕的程序只会因此更可笑。
- 缩进（indentation）是少数确实有用的排印约定之一。
- 注释要简短，不要做横幅式装饰。

12.3 Variable names（变量名）

> *Length is not a virtue in a name; clarity of expression* is.

要点：

- 全局变量名应富信息量，因为使用它的上下文里没有线索：`maxphysaddr` 合理。
- 局部短命变量应短：循环索引就用 `i`；不要写 `elementnumber`。
- 一致性比“长名”重要。如果你叫 `maxphysaddr`，不要把它的兄弟叫 `lowestaddress`。
- Pike 偏好最小长度但最大信息的命名。
- 不喜欢嵌入大写字母（“CamelCase”），理由：“they jangle like bad typography.”

12.4 Use of pointers（指针）

> *Pointers are sharp tools, and like any such tool, used well they can be delightfully productive, but used badly they can do great damage (I sunk a wood chisel into my thumb a few days before writing this).*

要点：

- C 的指针不是“危险特性”而是强表达力的记号。
- `np` 表示一个节点的指针，`node[i]` 是一个表达式——后者依赖 `node`/`i`/外层程序的隐含约定才有意义。
- “Stepping through structures using pointers can be much easier to read than with expressions: less ink is needed.”

例子：

```
        parent->link[i].type        /* 表达式式 */
vs.
        lp->type                    /* 指针式 */
```

12.5 Procedure names（过程命名）

> *Procedure names should reflect what they do; function names should reflect what they* return.

例子：`if(checksize(x))` 写得不清楚是否在出错时返回真；`if(validsize(x))` 立刻说明意义。

12.6 Comments（注释）

> *A delicate matter, requiring taste and judgement. I tend to err on the side of eliminating comments.*

理由：

1. 代码清晰自带说明；
2. 注释不被编译器检查，会过时；
3. 注释会让代码“长肉”。

但允许的注释：解释全局变量与类型；介绍异常或关键过程；分隔大型计算的段落。
经典反例：

```
      i=i+1;           /* Add one to i */
```

以及更荒唐的：

```
      /**********************************
       *                                *
       *          Add one to i          *
       *                                *
       **********************************/
                  i=i+1;
```

Pike：“Don’t laugh now, wait until you see it in real life.”

12.7 Complexity（复杂度）：Pike 的六条规则

这一段最常被引用：

Rule 1. You can’t tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don’t try to second guess and put in a speed hack until you’ve proven that’s where the bottleneck is.

Rule 2. Measure. Don’t tune for speed until you’ve measured, and even then don’t unless one part of the code *overwhelms* the rest.

Rule 3. Fancy algorithms are slow when n is small, and n is usually small. Fancy algorithms have big constants. Until you know that n is frequently going to be big, don’t get fancy.

Rule 4. Fancy algorithms are buggier than simple ones, and they’re much harder to implement. Use simple algorithms as well as simple data structures.

Rule 5. Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

Rule 6. There is no Rule 6.

附：他给出几乎“实用程序数据结构”的完整清单——**array, linked list, hash table, binary tree.** 就这四个。其他东西大多是这四种的复合。

12.8 Programming with data（用数据来编程）

> *The complexity of the job at hand, if it is due to a combination of independent details, can be encoded …Finite state machines are particularly amenable to this form of attack …*

要点：把复杂度藏进*数据表*里，而不是 `if` 与 `switch` 链。设备驱动表、yacc 的语法表、解析器的状态机表——这都是“数据驱动”的例子。

Pike 在这里顺手把 Pascal 锤了一下：

> *One of the reasons data-driven programs are not common, at least among beginners, is the tyranny of Pascal. Pascal, like its creator, believes firmly in the separation of code and data.*

12.9 Function pointers（函数指针）

> *I argue that clear use of function pointers is the heart of object-oriented programming. …The O-O languages give you more of course — prettier syntax, derived types and so on — but conceptually they provide little extra.*

这是 Pike 这一篇里最被反复引用的一段——“面向对象不过是清晰使用函数指针”。配合上一节的“data-driven programming”，Pike 主张大多数情况下你不需要专门的 OO 语言：“You can get 90% of the benefit for no extra work and be more in control of the result.”

12.10 Include files（头文件）

> ***Simple rule: include files should never include include files.***

如果它们不包含彼此，而是在自己开头注明“请先包含 X、Y”，那么由调用者去处理顺序——就避免了多重包含（multiple inclusion）的灾难。`/usr/include/sys` 在 1989 年已经是这方面的反面教材。

原文路径。 `/bell_labs/pikestyle.html`（HTML 全文，本章已包含其多数要点）

第二部分

Brian Kernighan：作为批评者的实践者

Chapter 13

为什么 Pascal 不是我喜欢的编程语言

"Pascal, at least in its standard form, is just plain not suitable for serious programming."
— Brian Kernighan, 1981 年 4 月 2 日

13.1 时代背景：Pascal 的高峰

1981 年的计算机课程里 Pascal 是霸主：学术界把它当作教学语言的事实标准；教科书几乎默认你用 Pascal 学 "结构化编程"；工业界也越来越多地把它当作严肃软件的开发语言——Apple Lisa 与早期 Mac、UCSD Pascal、Modula-2 都在这条线上。Brian Kernighan——身在 Bell Labs、刚刚把 Unix/C 推向世界——则坐下来一条一条地写出他认为 Pascal 为什么不能用于严肃工程。

13.2 Kernighan 的主要批评

按论文中的论述次序简要罗列：

1. 类型系统过于刚硬，但又不真正安全。数组长度是类型的一部分——"长度为 10 的 `int` 数组" 和 "长度为 20 的 `int` 数组" 是不同的类型，你写不出一个能处理任意长度数组的函数。
2. 字符串只能用定长数组表达，而字符串长度又是类型的一部分——于是写一个 "读一行打印出来" 的子程序都要费力。
3. 没有静态局部变量（Algol 风格的 `own` 也不在 Pascal 里）；
4. 没有 **`break`/`continue`**，必须用 `goto` 或者标志变量；
5. **`for`** 循环的边界一次求值，循环变量在循环结束后值未定义；
6. 真布尔短路被规定为 **"implementation-defined"**，于是 `if p <>nil and p^.x > 0` 这种 "先检查空再用" 的写法不可靠；

7. 没有分离编译——标准 Pascal 把整个程序当一个单元；
8. 初始化数据麻烦——没有 `const` 数据初始化，“code 和 data 严格分离” 的设计哲学和 Turing/von Neumann 的存储程序原理相悖（这条 Pike 在第 11 章也借用过）；
9. **I/O** 模型奇怪——`get/put` 的预读模型、`eof` 的语义、文件类型嵌进语言核心。

13.3　为什么这篇文章重要

它不只是一份吐槽清单。它是 **C** 语言（特别是 **K&R** 的设计选择）的一份反向声明：你之所以在 C 里看到不定长字符串、看到分离编译、看到 `break`/`continue`、看到 `static` 局部变量、看到 `const` 与 `#define` 的初始化数据——是因为它们的设计者（Ritchie/Kernighan/Thompson）实际写了几年系统软件，亲身体会到没有这些东西意味着什么。这篇文章把这些教训倒过来讲给 Pascal 用户。

后续。1981 年之后 Modula-2、Modula-3、Ada、Delphi、Oberon 都在不同程度上修正了 Kernighan 列出的问题；今天没人再用标准 Pascal 写系统软件，但这篇文章读起来依然是一份关于语言设计的工程后果的入门读物。

原文路径。 `/bell_labs/why_pascal/why_pascal_is_not_my_favorite_language.pdf`

Chapter 14

Timing Trials：脚本与界面语言的速度学

“No benchmark result should ever be taken at face value.”

— Kernighan & Van Wyk

14.1 要测什么

Timing Trials 是 Brian Kernighan 与 Christopher Van Wyk 合作的一篇实测论文。他们挑了一组当年（1990 年代末）流行的脚本语言与界面语言——Awk、Perl、Tcl、Tcl/Tk、Java、Visual Basic——在一组代表性任务上跑性能基准。论文摘要里的口气几乎是失望：

> *We found enormous variation in performance, depending on many factors, some uncontrollable and even unknowable. There seems to be little hope of predicting performance in other than a most general way; if there is a single clear conclusion, it is that no benchmark result should ever be taken at face value.*

14.2 一般性结论

文章给出几条超越具体语言的观察：

- 编译执行通常比解释执行更快——一个程序在执行前被“编译”得越多，运行越快。
- 内存层级问题无所不在。缓存、垃圾回收等机制对运行时长的影响巨大，而用户对绝大多数细节没有直接控制。
- 操作系统与运行时给出的计时服务远远不够好。即使要稳定可重复地测一段小型纯计算内核都很难，当程序涉及 I/O 或图形时就更难。

14.3　为什么仍然值得看

Timing Trials 是 Kernighan 一贯风格的样本：别相信你没量过的话；量过的也别全信。今天 Web 上漫天飞舞的 “Rust vs Go vs Java 基准”，绝大多数复现度低、变量未控、忽略 cache/GC/JIT 阶段——Kernighan 与 Van Wyk 在二十多年前就已经替我们写下了警告。

原文路径。 `/bell_labs/timing_trials/timing_trials.pdf`

第三部分

Ken Thompson：实验与实物

Chapter 15

一种新的 C 编译器

"This paper describes yet another series of C compilers. …Some of the ideas were good and some not so good."
— Ken Thompson

15.1 第几代 C 编译器了?

Ken Thompson 这篇标题朴素得几乎可疑:"A New C Compiler"。"又一个" 的口吻直接传递了 Bell Labs 的工作方式——不停地推翻自己、试新方法、有的成功有的不成功,没必要为新版本起一个霸气的名字。

这篇论文描述的是 **Plan 9** 上用的 **C** 编译器,也就是后来一系列以平台前缀命名的 8c / 6c / 5c / vc / kc / qc 那一套。它们与 K&R 的 PCC、与 GCC 在设计哲学上有几个截然不同:

- 每个目标平台一个独立可执行文件:`8c` 是 386 编译器,`6c` 是 amd64,`5c` 是 ARM,`vc` 是 MIPS,…你不需要跨编译的 flag——`8c` 永远只输出 386 代码。
- 编译器只做粗糙的优化,把精细优化(寄存器分配、调度)尽量交给 link 阶段。
- **Plan 9 C** 删掉了一些大家觉得理所当然的 **C** 语法:`long long`、`enum` 行为、隐式 int 等等——Thompson 的态度是 "Some of the ideas were good and some not so good",他自己留下来。
- 以预编译头文件 + 简化的预处理器加快构建——Plan 9 上 C 程序的编译速度几十年来一直让 Linux 用户惊讶。

15.2 这套工具链如何活到今天

Plan 9 C 编译器后来被 Go 语言团队接管,变成了 *Go 工具链最初的实现*——`8a/8c/8g`、`6a/6c/6g` 的命名延续到 Go 1.4 之前;Go 1.5 自举之后这些 C 工具被 Go 写的 `compile`/`link` 取代,但内部数据结构与多目标平台思路依然脉络可寻。Russ Cox 与 Rob Pike 在 *Go at Google: Language Design in the Service of Software Engineering* 那篇里专门提到,Go 工具链刻意继承的就是这套 "每个目标一个程序" 的传统。

延伸阅读。 *How to Use the Plan 9 C Compiler*（Pike）；*Plan 9 C Compilers*（Thompson）。

原文路径。 `/bell_labs/new_c_compilers/new_c_compiler.pdf`

Chapter 16

Reading Chess：用语义去读印刷品

"By analyzing the syntax of game descriptions and applying the rules of chess, the error rate was reduced by a factor of 30 from what was achievable through shape analysis alone."
— Baird & Thompson

16.1 一个意外的图像识别故事

Ken Thompson 出现在国际象棋字符识别论文上多少有点意外——但若你知道他写过 Belle（1980 年代赢得世界电脑象棋冠军的硬件下棋机），而且整个 Bell Labs 里围棋、国际象棋、桥牌的研究氛围浓得几乎像沙龙，那这篇与 Henry Baird 合作的 *Reading Chess* 就完全在意料之中。

16.2 方法概要

任务：把几册棋谱百科——字面意义上的纸质书，印刷质量极差——扫描为图像，然后识别出全部对局并加以分析。传统的 OCR 做法是先做 layout analysis（找列、找行、找词、找字），然后做 character recognition；当原书印刷质量差、字符碎裂或互相粘连时，结果直接报废。

Baird & Thompson 做的事情：

1. 严格自上而下的 **layout** 分析：用一种新的 "斜度估计" 技术，先把图像几何对齐，然后从列到行到词，逐级分割，速度快且稳定。

2. 基于词的启发式搜索分割：碎裂、粘连、脏字符不在字符级处理，而是在词级别用启发式搜索重新切分。

3. 用国际象棋规则做语义校验：每一步棋必须是合法走法，因此对识别结果做*语法 + 规则*校验可以把错误率降到三十分之一。

4. 在大约 100 万字符（2850 盘棋、945 页）上做评测，排除有原始排印错的对局后，98% 的对局得到合法解释，字符级有效成功率 99.995%。

16.3 思想要点

> **当原始数据有清晰的高层语义结构时，让识别系统受这套结构约束，比单纯加强像素级模型要有效得多。**

今天的 LLM 时代充满了“end-to-end 神经网络识别一切”的范式，但回过头看 *Reading Chess*，你会发现 Bell Labs 的人在 1990 年代就用一种极简的、几乎是规则推理的方式达到了 99.995% 的字符识别率——前提是会下棋。

原文路径。 `/bell_labs/reading_chess/reading_chess.pdf`

第四部分

其他：工具与人

Chapter 17

Mk：make 的继承者

“Mk runs anywhere from 2 to 30 times faster than make.”

— Andrew Hume

17.1 为什么需要替换 make

make（Stuart Feldman, 1976）几乎是 Unix 工具谱里寿命最长的一员——半个世纪后我们仍在 `Makefile` 里头疼 tab/空格的区别。1980 年代末，Andrew Hume 在 Bell Labs 写了 **mk**，把 make 的设计重做一次。摘要里的口吻平实但目标明确：

> *Mk is an efficient general tool for describing and maintaining dependencies between files or programs. Mk is styled on, and largely compatible with, the UNIX tool make. The major advantages of mk over make are executing recipes in parallel, using pattern-matching metarules rather than suffix transformation rules, and deriving dependencies by transitive closure on all rules. Mk runs anywhere from 2 to 30 times faster than make.*

17.2 mk 的三件关键改进

并行执行 **recipe**。 在 make 里 `-j` 是后加的 hack；mk 一开始就把*并发*当作正常情况——recipe 的执行单元是命令组，调度器并发跑独立分支。

模式匹配的 **metarule**。 make 的 `%.o: %.c` 是 1990 年代后期 GNU make 才有的扩展；mk 一开始就用通用的模式元规则代替老式的“`.c.o:`”后缀规则。

依赖传递闭包。 mk 在所有规则上做传递闭包，从而能从一条规则推导出隐含依赖——make 需要你显式声明。

17.3　在 Plan 9 上的位置

`mk` 是 Plan 9 标准构建工具，整个 Plan 9 from User Space 的源码树今天仍用 mk。即使在 Linux 上，`plan9port` 把 mk 移植了过来，许多偏向 Plan 9 风格的项目（包括 9front、9legacy、若干个人项目）默认使用 mk 而不是 make。

延伸阅读。 *Maintaining Files on Plan 9 with Mk*（Hume & Flandrena）；*Plan 9 Mkfiles*（Flandrena）。

原文路径。 `/bell_labs/mk/mk.pdf`

Chapter 18

Upas：更简单的网络邮件

“upas, n. (in full upas-tree), a fabulous Javanese tree that poisoned everything for miles around.”
— 词源注脚

18.1 1980 年代的邮件混乱

The Hideous Name（本书第 7 章）里那个吓人的邮件地址，就是 1980 年代邮件系统的现实。一封邮件可能要经过 UUCP 跳板、ARPA、BITNET、学校内网、各自的网关，每过一道关就被一种新的语法改写一次。“地址”与“路由”搅在一起；用户必须手写出每一跳。

18.2 Presotto 的思路：写一个统一的中介

Upas 是 David L. Presotto 在 Eighth Edition Unix 上写的邮件接口，

> *Upas is a mail interface that routes messages between existing network specific mailers, users, and user mailboxes. It uses a regular expression based mini-language to convert mail addresses into the commands needed to route the mail to the intended destination.*

它的核心想法非常 cat-v.org：

- 不重新发明 SMTP / UUCP / BITNET 网关；
- 写一个基于正则表达式的小型规则语言，描述“如果地址长这样，就调用这个 mailer”；
- 把规则集中在一处管理，用户端的 `mail` 命令永远只与 Upas 对话；
- 邮箱本身保持平凡——一个目录、一些文件。

18.3　Plan 9 继承下来的 Upas

后来 Plan 9 上的 `upas` 被改写：用 9P 把邮箱挂载为一个文件系统（`upas/fs`）——你可以 `cd /mail/box/yourname`、`ls` 出每封邮件作为一个目录，`cat header`, `cat body`, `cat raw`。没有 IMAP，没有 POP3，没有 Maildir——一切都是 9P 上的文件。

Plan 9 from User Space 也把这版本 Upas 移植到了普通 Unix；今天少数几位 `cat-v.org` 的常客就以这种方式收发邮件。

原文路径。 `/bell_labs/upas_mail_system/upas.pdf`

Chapter 19

Squinting at Power Series：用通信进程算幂级数

"Communicating processes are the key to the simplicity of the algorithms."
— Doug McIlroy

19.1 Doug McIlroy 与函数式算法的优雅

Doug McIlroy——Unix 管道的发明人之一、awk 的早期推动者、*Mass-produced Software Components* 的作者——在这篇短小但极优雅的论文里把幂级数的乘法、复合、指数、反转等运算重写为*数据流*上的递归方程。工具是 Rob Pike 当时的 newsqueak（小语言、CSP 风格、有同步通道）。

19.2 思想要点

> *Data streams are an ideal vehicle for handling power series. Stream implementations can be read off directly from simple recursive equations that define operations such as multiplication, substitution, exponentiation, and reversion of series.*

具体而言：

- 一个幂级数 $f(x) = a_0 + a_1 x + a_2 x^2 + \dots$ 被实现为一个*进程*，每次被询问 "下一个系数" 时把它送出。
- 乘法、复合、求导、积分、反演（reversion）都可以用 newsqueak 中的几个 `chan` 与递归调用直接写出来。
- 由于通道天然处理同步与缓冲，过去在传统语言里需要手工管理的 "算到第几项了"、"谁催谁"、"中间值要不要 memoize" 等问题，都不见了。

19.3　它的影响

Squinting at Power Series 是“communicating processes 不只是并发，还是抽象工具”这种说法的一个极小但说服力极强的范例——你不会拿数据流去写一个 Web 服务器，但你会从这种风格里看到为什么 **select/channel** 优雅。后来 Haskell 社区把同样的想法做成了“惰性求值的无穷序列”——两条线路实际上走的是同一种数学思路。

原文路径。 `/bell_labs/squinting_at_power_series/squint.pdf`

Chapter 20

Crabs：一个“打破规则的窗口演示”

“Crabs is a graphic demo which violates most of the assumptions underlying well-structured window systems.”
— Luca Cardelli

20.1 螃蟹爬过你的窗口

Crabs 是 Luca Cardelli 在 Eighth Edition Unix 上写的一个图形 demo——你在 Blit 或者 mux 这类窗口系统下启动它，屏幕上会出现一些位图小螃蟹，它们会爬过你正在用的窗口，擦掉一段又一段像素，但下面的窗口系统不为它们重绘——等螃蟹走过去，屏幕上就有一道伤痕。

它的“技术贡献”是反着说的：通常窗口系统会假设客户程序只在自己窗口的矩形里画；Crabs 借助一些 hack（直接写到 `/dev/bitblt` 之类的低层接口）绕过这条假设，画在所有窗口之上。

> *Crabs is a graphic demo which violates most of the assumptions underlying well-structured window systems. It illustrates both the raw power of bitmap graphics and the restrictions which are usually imposed on its usage.*

20.2 它为什么值得记入档案

- Crabs 演示了窗口系统的边界是约定，不是物理。一个“行为规矩”的客户端不去随意写屏幕，并不是因为它做不到，而是因为大家都按约定来。
- Cardelli 后来去了 SRC，写了 Modula-3、做了 calculus of objects；Crabs 是他在 Bell Labs 留下的最有玩心的产物。
- Eighth Edition Unix 的 `crabs(9.6)` man page 至今可在 `man.cat-v.org/unix_8th/9/crabs` 查阅。

原文路径。 `/bell_labs/crabs/technical-memorandum.pdf`

Chapter 21

Duff's Device：一个 1983 年的循环展开

"Disgusting, no? But it compiles and runs just fine."

— Tom Duff, 1983 年 11 月 10 日

21.1 故事的开头

1983 年 11 月，Tom Duff 还在 Lucasfilm 工作。他要把一个 `short *to, *from` 的拷贝循环加速 50%——拷贝目标是 Evans & Sutherland Picture System II 的 PIO 寄存器，所以不能用 `memcpy`（因为目标地址不能自增）。传统做法是用 "loop unrolling"——把循环体展开成 8 份，再用一个 `switch` 处理余数。

那天晚上，Duff 写下了下面这段 C 代码。它是合法的 C；它依赖 C 语言里 `switch` 不自动 break 的特性，让 `switch` 的 `case` 标签穿插进 `do-while` 循环的中间。

```
send(to, from, count)
register short *to, *from;
register count;
{
    register n=(count+7)/8;
    switch(count%8){
    case 0: do{ *to = *from++;
    case 7:     *to = *from++;
    case 6:     *to = *from++;
    case 5:     *to = *from++;
    case 4:     *to = *from++;
    case 3:     *to = *from++;
    case 2:     *to = *from++;
    case 1:     *to = *from++;
        }while(--n>0);
    }
```

```
}
```

写完之后 Duff 给 dmr（Dennis Ritchie）发了一封邮件，开头写道：

> *Disgusting, no? But it compiles and runs just fine. I feel a combination of pride and revulsion at this discovery. If no one's thought of it before, I think I'll name it after myself.*
> *It amazes me that after 10 years of writing C there are still little corners that I haven't explored fully.*

21.2 为什么它是合法的 C

C 的 switch 语句的语义是跳转到匹配的 *case* 标签处；那之后控制流就是普通的顺序执行。没有任何规则规定标签必须在最外层；case 完全可以出现在 do-while 循环内部——当 switch 跳进去之后，循环条件仍然每次执行结束后才求值。

正是这种“跳进循环体中间”的能力，让 Duff 把“余数处理”和“主循环”合并成同一段 8 条赋值。他自己写：“If I were writing assembly language code, I'd just jump into the middle of the unwound loop to deal with the leftovers.” Duff's Device 是用 C 表达汇编的这种跳法。

21.3 Duff 自己后来澄清的几件事

cat-v.org 上保留了 Duff 1988 年回到 comp.lang.c 的一封长信，其中他特别强调几条经常被外界搞错的事：

1. 设备的核心是用 **C** 直接表达循环展开——不是用 memcpy 能替代的，因为他要拷贝到固定地址的 PIO 寄存器。
2. 是他发明的——他给 dmr 发邮件的日期是 1983 年 11 月 10 日 17:57 PST，邮件原文在 cat-v.org 上仍可读。
3. 语法合法。X3J11 委员会主席 Larry Rosler 与 dmr 都确认过；Bjarne Stroustrup 在 C++ 书里也用了它。
4. 不是过度优化的反面教材——“If your code is too slow, you must make it faster. If no better algorithm is available, you must trim cycles.”
5. 警告：在现代机器上要小心，过度展开会冲垮指令缓存；“Don't try to be smarter than an over-clever C compiler.”

21.4 它的文化地位

Duff's Device 已经超出了“一段优化代码”的范畴，变成了 C 语言民俗的一部分——“after 10 years of writing C there are still little corners that I haven't explored fully” 这句话被无数

程序员引用过，对应着每一种语言都会有的“啊原来还能这样”的瞬间。

它也是关于程序员的 **“pride and revulsion”** 的一段经典坦白——对自己写出的某段聪明代码同时感到自豪与厌恶；这种感觉是写软件的人共有的一种秘密心情。

原文路径。 `/bell_labs/duffs_device.html`（含 Duff 原始邮件与 1988 年澄清信全文）

附录 A：Bell Labs Innovations Song

“Bell Labs Innovations / The communication for the next generation”

— 主歌副歌

A.1 一首奇怪的公司歌

2000 年 Lucent Technologies 公关部出资制作了一首名为 *Bell Labs Innovations* 的主题歌，由 Michael Jacobs 与 Patrick Regan 词曲，Michael Bakic 剪辑，特别感谢 Ed Eckert 与 Lucent 档案部门——这是版面注脚原文。歌词把贝尔实验室一百年的发明史压缩为一份名词清单——发明者的名字、Nobel 奖、电话、雷达、晶体管、激光、Unix、C、CCD、宇宙背景辐射……`cat-v.org` 的版本由 Stanley Lieber 转写，仍标记为 “Work in progress”。

A.2 歌词节选（保留原英文）

> *Alexander Graham Bell*
> *Thomas Watson*
> *Tedy Bell*
> *George Campbell’s wave filter*
> *Calling ships ashore*
> *Frank Jewett*
> *Harrold Black*
> *Negative feedback*
> *Televisions, fax machines*
> *More and more and more*
>
> *Hifi talkies, movie sound*
> *Laying copper by the pound*
> *System engineering, quality control*

Karl Jansky
Herbert Ives
Davisson wins Nobel Prize
First speech synthesis
Beethoven in stereo

Bell Labs Innovations
The communication for the next generation
Bell Labs Innovation
Our contribution to the revolution

…
Penzias and Wilson hear the Big Bang noise
Automated switchboards
Ever moving forward
…
Jhon …Ken …Unix
Quantum wheels
C Language
Fiber Optics will endure
Electron beam lithography
…
Penzias and Wilson win the Nobel Prize
Bell Labs gets a new boss, by the name of Ian Ross
Lightwave goes long distance
Cellular gets digitized
S language
C++
…
Stormer, Laughlin, Tsui win the Nobel Prize
Free space optics
Petabits
Lucent's star is on the rise

完整歌词在 `/bell_labs/innovations_song/` 页面上（YouTube ID `U5V1sxAKu5I`）。

A.3 为什么把它收进这本书

它不是技术文档；但它精准展现了那种 `cat-v.org` 一直在与之搏斗的张力：一方面是 1127 室里实际发生过的科学与工程；另一方面是 Lucent/AT&T 公关部门给这场科学制造的、近乎卡通的庆典。这首歌把诺奖、发明史、品牌口号搅在一起，本身就是 “Unix 文化 vs. 大公

司公关文化” 的一份珍贵副产品。

附录 B：Dennis Ritchie 速写

B.1 基本资料

- **Unix 用户名：** `dmr`
- **Unix UID：** 7
- **写过的命令：** `cc`, `ld`, `as`, `db`, `strip`, `bcd`, `cmp`, `date`, `du`, …
- **创造的语言：** C, m4
- **首选编辑器：** `acme`
- **最喜爱的编程书：** *The Elements of Programming Style*（Kernighan & Plauger）；*Travels in Computerland*（Ben Ross Schneider）

B.2 留下的书

- *The C Programming Language*（与 Brian Kernighan 合著，1978/1988）——俗称 “K&R”、“The C Bible”。

B.3 论文与其他出版物

- *Unix Programmer's Manual* (1971)
- *The Limbo Programming Language*（与 Inferno 团队合著）

B.4 相关网络资源

- `cm.bell-labs.com/who/dmr/`（原站点已删除，存档于 archive.org，2013 年 6 月）
- `genius.cat-v.org/dennis-ritchie/`

> Dennis MacAlistair Ritchie 在 2011 年 10 月去世，与 Steve Jobs 同月。当时科技媒体几乎全部聚焦于 Jobs，而 Ritchie 的离世只在程序员社区中得到匹配的哀悼。没有 C，没有 Unix，没有 Bell Labs 的那一代——我们现在在用的几乎所有操作系统、所有手机、所有

> 汽车里的固件，都不会是今天的样子。

后记：当我们在读这些文档时，我们在读什么

本书蒸馏的全部材料只有二十余篇——`doc.cat-v.org` 的 `/bell_labs/` 目录其实很小。但它们之间存在一条难以言喻的内部联系：Blit、并发窗口系统、透明窗口、结构正则、sam、Face the Nation、Squeak、Hideous Name、UTF-8、utah2000、Good/Bad/Ugly、pikestyle——这些来自同一个房间、同一群人，用三十年时间反复琢磨一组相同的问题：

- 怎样让工具更小，组合更广。
- 怎样让接口透明，让下面的东西被看见。
- 怎样让一个名字（路径、地址、变量）准确表达它指的对象，不多不少。
- 怎样让并发显形，而不是把它压平成回调。
- 怎样让程序保持人类可读，而不变成抒情诗。

这五条几乎可以同时套用在 Unix、Plan 9、Inferno、newsqueak、Squeak、Alef、Limbo、Go——甚至套用在 `werc`、`acme`、`rio`、`9P`、`plumber`、`cat-v.org` 自己的网页设计上。1127 室的人持续写出的不是“一组论文”，而是一种审美——这种审美在今天看大型 Web 框架、看 systemd、看 Electron 应用、看 GraphQL，仍然能像 1989 年看 Pascal 时那样精准地说出*为什么这件事别扭*。

写这本蒸馏的目的不是替代原始文档——那些 PDF 全在 `cat-v.org` 上，今天下载下来仍然能打开、能阅读、能引用、能传给学生。本书的目的是给汉语读者一个入口与索引：读完这二十几章之后，你应该对 Bell Labs 这一脉的计算机文化形成清晰的地图，知道哪些论文值得花一个晚上自己去读，哪些可以留到下次再说。

> *“Be courageous. Try different things; experiment. Try to give a cool demo.”*
>
> *— Rob Pike, utah2000*