

指数

符号	API 注释, 193文档的好
@deprecated 注释, 322	处, 187
@DoNotMock 注释, 266	C++,文档, 193
一个	代码搜索,曝光, 359概念文档和, 198
A/B 差异测试, 299限	声明类型不应被模拟, 266伪造, 270服务 UI
制, 300运行预提交,	后端提供公共 API,
304 SUT 行为, 296	292
ABI 兼容性, 434	通过公共 API 进行测试, 234-237对错误
Abseil,兼容性承诺, 434对抗组交互, 47咨	道歉, 94
询弃用, 316	AppEngine 示例,导出资源, 454
人工智能面部识别软件,不利于	审阅者的批准印章, 412 次Google 代码变更批
一些人群, 73种子数据,偏	准, 167 次为失败而架构, 523 次基于工件的构建系
差, 282	统, 380-386功能视角, 381使用
飞机, 寓言, 108警觉疲劳, 318	Bazel 实现具体化, 381其他巧妙的 Bazel 技
“始终发挥领导作用”	巧, 383-386时间、规模和权衡,
永远做决定, 108	390 次提问, 48 次询问团队成员是否需
决定,然后迭代, 110	要任何东西,
总是要离开, 112	100
始终保持扩展,来自代码分析	询问社区, 50-52
器的116 个分析结果, 404 个注释,每个测试,记录所	禅宗管理技术, 95
有权,	断言
308	在对被测系统的多个调用之间, 243在 Java 测试中,
Ant, 376通	使用 Truth
过向命令行提供任务来执行构建, 377被更现代的构	库, 248与被测系统有直接关系的存根函数,
建系统取代,	275在 Go 中测试断言, 248验证 SUT 的行为, 295原子
378	变化,障碍,
测试套件中的反模式, 220	463-465
蜜蜂	

VCS 中提交的原子性, 328,332工程师的关注 (QUANTS),
131受众评论, 199编写大型测试, 305大规模更改的授权,
473自动构建系统, 372自动测试
代码正确性检查, 172限制, 229自动
化 自动 A/B 发布, 512在持续集成中, 483-485代码审查,
179 CaaS 中的自动化工作, 518-520自动调
度, 519简单自动化, 519
团队成员的自主权, 104自动扩展, 522

B
倒退,阻止折旧过程, 322
向后兼容性和对效率改进的反应, 11批处理作业与服务作
业, 525
Bazel, 371,380
依赖版本, 394扩展构建系统,
384具体化, 381构建步骤并行化, 382
使用命令行执行构建,

382
每次仅重建最小目标集、383隔离环境、384确保
外部依赖性确定、
385使用工具链实现平台独立性、
384
远程缓存和可重现构建, 387
速度和正确性, 372工具作为依赖
项, 383文档的开始、中间和结束
部分, 202
行为
代码审查变更, 181测试代替方法,
241-246

根据测试行为来命名测试, 244
构建测试以强调行为, 243
未预料到的,测试, 284更新测试更
改, 234最佳实践,风格指南规则执行, 152
碧昂丝规则, 14,221偏
见, 18互动
中的小表达, 48普遍存在, 70二进制,互动,功能
测试, 297无可指责的事后分
析, 39-41,88

Blaze, 371,380
全局依赖图, 496盲区,识别, 109在
Web 搜索延迟案例研究中, 110
Boost C++ 库,兼容性承诺, 435分支管理, 336-339 VCS
中的分支名称, 330 dev 分支,
337-339 Google 中少数长期存在
的分支, 343发布分支,
339正在进行的工作类似于分支, 336 “聪明的
混蛋”, 57脆弱的测试,
224防止, 233-239争取不变的测试, 233测试状
态,而不是交互, 238通
过公共 API 进行测
试, 234-237记录/重放
系统导致, 492过度使用存根, 273浏览
器和设备测试, 297

Buck, 380
错误抨击, 299错
误修复, 181,234错误
在开
发后期被发现,成本, 207在实际实施中导致测试失败的
连锁反应, 265逻辑隐藏了测试中的错误, 246仅靠程
序员的能力无法阻止,

210
BUILD 文件,重新格式化, 162 个构建
脚本,基于任务
的构建系统的困难,
379
写作作为任务, 378
构建系统, 371-398

处理模块和依赖关系, 390-396	名人, 29计算机
管理依赖关系, 392-396最小化模块可见性, 392使用细粒度模块和 1:1:1 规则, 391现代, 375-390基于工件, 380-386依赖项和, 375分布式构建, 386-390基于任务, 376-380时间、规模和权衡, 390目的, 371使用其他工具代替, 372-375编译器, 373 shell 脚本, 373	务中的集中化与定制化, 537-539集中版本控制系统 (VCS), 332未来, 348内部开发,谷
构建器模式, 252构建工件, 376构建脚本和, 378在基于工件的构建系统中, 380Bazel, 381为每个人构建, 77捆绑分销模式, 441巴士因子, 31,112	歌的 Piper, 340操作随更改大小线性扩展, 463事实来源, 334未提交的本地更改和分支上的已提交更改, 336大规模更改的变更管理,
碳	470
C 语言,用其编写的项目,更改, 10C++	变更列表搜索, 411变更列表 (CL),可读性批准, 63代码变更批准或变更评分, 412 LSC 流程中的变更创建, 473
API,参考文档, 193Boost 库,兼容性承诺, 435兼容性承诺, 434 Google 员工开发指南, 200 googlemock 模拟框架, 262开源命令行标志库, 452	评论, 408提交, 413创建, 402-406大规模 (参见大规模变更)跟踪历史记录, 414在 VCS 中跟踪, 328生产代码的变更类型, 233了解状态, 410-412编写良好的变更描述, 178编写小变更, 177混乱和不确定性,保护您的团队免受混乱工程的影响, 222, 302
scoped_ptr 到 std::unique_ptr, 467使用外部依赖缓存构建结果, 395	切斯特森栅栏原理, 49Churn Rule, 13clang-tidy, 160与 Tricorder 集成, 422类注释, 194类和技
Python 中的 CamelCase 命名, 154金丝雀分析, 301金丝雀部署, 482规范文档, 188职业,团队成员跟踪, 101胡萝卜加大棒的管理方法, 86催化剂,存在, 96牛与宠物的类比适用于代码库中的更改, 474适用于服务器管理, 524	术讲座, 52经典测试, 265“干净”和“可维护”代码, 10LSC 流程中的清理, 477清晰的测试,编写, 239-248
CD (参见持续交付)	测试中不包含逻辑, 246使大型测试易于理解, 307使测试完整而简洁, 240测试行为,而不是方法, 241-246行为驱动测试, 242方法驱动测试, 241根据被测试行为命名测试, 244

构建测试以强调行为，
243
编写清晰的失败消息， 247 “聪明”的代码， 10
Clojure 包管理生态系统， 447云提供商,公有云与私有云， 543
指导低绩效者， 90代码

测试的好处， 213-214代码是负担,而
不是资产， 167,313使用 g3doc 嵌入文档，
190
将测试表达为， 212知识共享，
56质量， 131代码覆盖率， 222代码格
式化程序， 161代
码审查， 56,62-66，
165-183好处， 166,170-176
代码一致性， 173代码理解， 172代码正确
性， 171知识共享， 175心理和文
化， 174最佳实践， 176-180
尽可能自动化， 179礼貌和专业， 176
将审阅者保持在最低限度， 179
撰写良好的变更描述， 178撰写
小变更， 177代码作为负担， 167流程，
400大规模变更， 467,476他们
在 Google 的工作方式， 167-169代码所有
权， 169-170步骤， 166类型， 180-182行为
变化,改进和

优化， 181错误修复和回
滚， 181全新审查， 180重构和大规模
更改， 182

代码搜索， 178,351-370
Google 的实施， 361-365排名， 363-365搜索
索引， 361 Google 员
工如何使用它， 353-355
回答代码库中某些内容的位置， 353

回答某人何时和何人引入了代码， 355回答代码为何以意
想不到的方式运行，
354回答其他人如何做某事， 354回答代码库的某个部分
在做什么， 354规模对
设计的影响， 359-361索引延迟， 360搜索查询延迟，
359需要单独
的 Web 工具的原因， 355-359
与其他开发工具集成， 356-359
Google 代码库的规模， 355专业化， 356
零设置全局代码视图，
356实施中的权衡， 366-369
完整性,全部与最相关结果， 366完整性,头部与
分支与全部历史
与工作区， 367完整性,头部存储库， 366表现力,标记
与子字符串与正则表达式， 368 UI，
352
代码共享,测试和248-255
定义测试基础架构， 255共享助手和验证，
254共享设置， 253共享值， 251过于 DRY 的测
试， 249测试应该是
DAMP， 250代码库分析，
大规模变更和， 470注释,参考文
档生成， 193影响灵活性的因素， 16
可扩展性，
12可持续性， 12代码库范围一致性的价值， 64代码实
验室， 60评论变更批评， 408

评论
代码， 193
条风格指南规则， 145 个社区
跨组织,共享知识，
62

从社区获取帮助， 50-52编译器与静态分析的集成， 426
编译器升级（示例）， 14-16编译器,使用而不是构建系统，
373测试的完整性和简洁性， 240文档的完整性、
准确性和清晰度， 202代码的理解， 172强制弃用， 317

计算即服务 (CaaS)， 517-545
选择计算服务， 535-544
集中化与定制化， 537-539抽象级别,无服务器，
539-543公
有与私有， 543随时间和规模变化， 530-535容器作为一种抽象， 530-532一项服务统治
一切， 533提交的配置， 535驯服计算
环境， 518-523辛劳自动化， 518-520容器化和多租
户， 520-522

编写托管计算软件， 523-530

为失败而架构， 523批处理与服务，
525连接到服务， 528管理状态，
527一次性代码， 529概念文档， 198
居高临下和不受欢迎的行
为， 47单元测试的配置
问题， 283共识,构建， 96代码库内的一致
性， 146优势， 146构建一致性,规则， 153确保代码审查， 173例
外,向实际情况让步，

150
完全一致的低效率
大型代码库， 148
单一版本规则， 342制定标准， 148

建设性批评， 37消费者驱动的契约
测试， 293容器化和多租户， 520-522适当调整和
自动缩放， 522容器作为抽象， 530-532

容器和隐式依赖关系， 532上下文,理解， 49持续构建
(CB)， 483持续交付 (CD)， 483，
505-515

将部署分解为可管理的部分， 507改变团队文化以在部署中
建立纪律，
513隔离评估变更,标志保护功能， 508 Google 的 CD 习
语， 506质量和用户关注,仅运
送使用过的内容， 511向左移动并尽早做出数据驱
动的决策， 512追求敏捷,设置
发布列车，

509-510
持续部署 (CD),发布分支和， 339持续集成 (CI)， 14,479-503警
报，
487-493

CI 挑战， 490密封测试，
491核心概念， 481-487自动
化， 483-485持续测试，
485-487快速反馈循环，
481-483开发分支和， 338项目所需的
绿地评审， 180在 Google 实施，
493-503

案例研究,Google Takeout， 496-502
TAP,全局持续构建， 494-496
住在 Head 依赖管理处，
442

谷歌的系统， 223 个合同伪造，
272 个合作小组互动， 47 个
构建系统的正确性， 372 个代码的正确性， 171 个成
本

在软件工程中， 12通过在开发早期发
现问题来减少， 17权衡和， 18-23在时间和规模之间做
出决定（示例）， 22
分布式构建（示例）， 20决
策输入， 20,20

决策失误, 22成本类型, 18白板笔 (示例), 19

批评,学会给予和接受, 37

批评代码审查工具, 165,353,399-416

 变更批准, 412代码审查流程, 400代码审查工具原则, 399提交变更, 413创建变更, 402-406分析结果, 404差异, 403紧密工具集成, 406差异查看器, Tricorder 警告, 421请求审查, 406-408理解和评论变更, 408-412静态分析修复视图, 424加密哈希, 385罪魁祸首查找和故障隔离, 490使用 TAP, 495文化

在部署中建立纪律, 513 LSC 周围规范的变化, 469培育知识共享文化, 56-58代码审查的文化优势, 174学习文化, 43数据驱动, 19,22健康的自动化测试文化, 213当今 Google 的测试文化, 228

客户,文档, 192

CVS (并发版本系统), 328,332

德

DAMP, 249

 是DRY的补充,而不是替代, 251

 测试重写为 DAMP, 250仪表板和搜索系统 (批评), 411库中的数据结构和列表, 158数据驱动的文化, 19承认错误, 22数据驱动的决策,尽早做出, 512数据中心,自动化管理, 523调试与测试, 210决策

承认错误, 22

决定,然后迭代, 110在工程组中,理由, 19

决策输入, 20高层领导的决策, 108

将子问题委托给团队领导, 113依赖关系

Bazel 将工具视为每个目标的依赖项, 383构建系统和, 375在测试中使用真实实现时的构造, 268

容器和隐式依赖项, 532依赖管理与版本控制, 336外部,导致测试中的不确定性, 268

外部,编译器和, 373分叉/重新实现与添加依赖项, 22在基于任务的构建系统中, 377在 Bazel 中确定外部依赖项, 385在构建系统中管理模块, 392-396

自动与手动管理, 394使用外部依赖项缓存构建结果, 395外部依赖项, 393内部依赖项, 392单版本规则, 394外部依赖项的安全性和可靠性, 396传递外部依赖项, 395新的,防止引入弃用系统, 322在共享设置方法中的值上, 253用测试替身替换类中的所有内容, 265测试范围, 219未知,在弃用期间发现, 318

依赖注入

 框架, 261引入接缝, 260依赖管理, 429-457难度,原因, 431-433冲突需求和钻石依赖, 431-433导入依赖, 433-439

兼容性承诺, 433-436考虑因素, 436	开发人员的幸福感,重点关注,静态分析, 419开发人员工具,代码搜索集成, 356-359
谷歌的处理方式, 437-439理论上, 439-443捆绑分销模式, 441	开发人员工作流程,大型测试和, 304-309编写大型测试, 305
Live at Head, 442没	运行大型测试, 305-308消除不稳
有任何变化(静态依赖模型), 439语义版本控制, 440语义版	定性, 306使测试易于理解, 307拥有
本控制的局限性, 443-449	大型测试, 308加快测试速度, 305
	开发人员工作流程,使静态分析成为部分
最低版本选择, 447动机, 446过度约束, 444过度承诺兼容性, 445质疑它是否有效, 448	共420
	DevOps 技
拥有无限的资源, 449-455	术生产力哲学, 32 个基于主干的开发流行起来,
	327
导出依赖项, 452-455部署分解成可管理的部分, 507建立纪	DevOps 研究与评估 (DORA) 没有长期分支和, 343基于主干的
律, 513部署配置测试, 298弃用, 311-323作为扩展问题的示例, 13难度, 313-315在设计期间, 315管理流程, 319-322弃用工具, 321-322里程碑, 320流程所有者, 320	开发与高绩效组织之间的预测关系, 343发布分支研究, 339钻石依赖问题, 394,431-433差异代码更改, 403更改摘要和差异视图, 404方向,给予团队成员, 104灾难恢复测试, 302发现 (弃
旧文档, 203防止对弃用对象进行新的使用,	用), 321分布式构建, 386-390在 Google, 389
	远程缓存, 386远程执行, 387权衡和成本示例, 20分布式版本控制系统 (DVCS),
477	
原因, 312 API 弃用的	332
静态分析, 417类型, 316-319建议性弃用, 316强制弃用, 317弃用警告, 318	历史数据压缩, 367场景,没有明确的事实来源, 335事实来源, 334多样性使其可行, 74理解需求, 72
描述性和有意义的短语 (见 DAMP)设	
计文档、195 次新代码或项目的	Docker, 531文
设计评审、180 次最终将被弃用的设计系统,	档, 53-55, 185-205关于, 185好处, 186-187
315	代码, 56
测试中的确定性, 267开发分支, 337-339没有长期分支, 343开	
发人员指南, 59	代码搜索集成, 358创建, 54

代码更改, 178了解你的
读者, 190-192读者类型, 191哲学,
201-204开始、中间和结束部
分, 202弃用文档, 203
良好文档的参数, 202谁、什么、为什么、何时、何地
和如何,

201

推广, 55视为代
码, 188-190
Google wiki 和, 189类型,
192-199概念, 198设计
文档, 195登陆页面,
198参考, 193-195教程, 196更
新, 54当您需要技术作家
时, 204文档注释, 145文档
审查, 199-201文档
知识, 45文档受众
的领域知识, 191

DRY (不要重复自己)原则
测试和代码共享, DAMP, 而不是DRY, 248-255

DAMP 作为 DRY 的补充, 251测试太过
DRY, 249违反更清晰的测
试, 241

DVCS (分布式版本控制系统)
项目)

埃

爱迪生, 托马斯, 38软件
工程师教育, 72
需要更具包容性的教育, 74提高效率, 改变
代码, 11自我, 失败, 36, 93艾森豪威尔, 德怀特·D, 118
谷歌的电子邮件, 51
爱默生, 拉尔夫·沃尔多, 150端到
端测试, 219工程经
理, 82, 86-88, 88 (另见领导团
队, 经理和技术

领导)当
代管理者, 87

让团队知道失败也是一种选择,
87

经理作为四个字母的单词, 86工程生
产力通过测试提高, 231可读
性程序和, 65

工程生产力研究 (EPR) 团队,
65

工程生产力, 测量, 123-138评估测量的价值, 125-128
目标, 130指标, 132原因, 123-125选择具
有目标和信
号的有意义的
指标, 129-130信号,
132采取行动并跟踪绩效后的结果

形成研究, 137用数据验
证指标, 133-137公平和包容的工程, 69-79
偏见和, 70建立多元文化能力, 72-74挑战既定流
程, 76使多样
性具有可操作性, 74多样性的必要性, 72种
族包容性, 70拒绝单一方法, 75保持好奇
心, 不断前进, 78价值观与结果, 77
错误检查工具, 160

易出错工具 (Java), 160
@DoNotMock 注释, 266与
Tricorder 集成, 422代码中容易出错
和令人惊讶的构造, 避免, 149测试执行时间, 267加速
测试, 305文档
受众的经验水平,

191

实验和功能标记, 482专业知识 全有或全
无, 44来
自专家的个性化建
议, 45和共享交流论坛, 14开发与探索问题,
363探索性测试, 229, 298外在动机与内在动
机, 104

F

“尽早失败、快速失败、经常失败”， 31 个解决测试失

败的失败， 213 个针对托管计算软件
失败的架构， 523 个实际实施中的错误导致测试失败的级
联， 265 个清晰的代码有
助于诊断测试失败，

218

查找罪魁祸首并隔离故障， 490快速失败并迭代， 38失
败是一种选择， 87使用 TAP
进行故障管理， 495失败的大
型测试， 307测试失败的原因， 239系统故障测
试， 222为测试编写清晰的故障
消息， 247伪造， 263,269-272伪造的
密封后端， 491伪造的保真度， 271伪造
的重要性， 270测试伪造， 272何时无法使用伪造， 272何
时编写伪造， 270静态分析
中的误报， 419静态分析中的误报，
419功能标志， 482新功
能， 234联合/虚拟单一存储库
(VMR) 样式存储库，
346反馈

加快进度， CI 中有32 个快速反馈循环， 481-483用
于文档记录， 54为团队成员提供硬反馈，
Tricorder 中有98 个集成反馈
渠道，

423

向开发人员征求关于静态分析的意见，
419

伪造的
保真度， 271
SUT 的保真
度， 290测试替身的
保真度， 258
测试的保真度， 282
文件注释， 194 VCS 中的文件锁
定， 329,332文件系统抽象，
531文件系统,VCS 作为扩展方式， 329

保护标志的功能， 508不稳定测试，
216,267,490消除大型测试中的不
稳定因素， 306费用， 218

Forge,389,496分
支/重新实现与添加依赖项,22函数注释,195函数式编程
语言,381功能测试,
219测试一个或多个交互二进制文
件,

297

G

g3doc, 190
盖茨,比尔, 28生
成的文件,代码搜索引擎和, 366
天才神话, 28
Gerrit 代码审查工具, 414
Git, 333
改进, 347合成 monorepo
行为, 346给定/当/然后,表达行为, 242交替使用
when/then 块, 244结构良好的测试, 243

Go 编程语言

兼容性承诺, 434 gofmt 工具案例研
究, 161-163标准包管理生态系统,

447

测试断言, 248进入/链接,
60与规范文档一
起使用, 188,201定义目标, 129团队领导设定明确的目标,
97

目标/信号/指标 (GSM) 框架, 129-133

目标, 130指
标, 132信号,
132在可读性过
程研究中使用指标,
134

谷歌助理, 492
Google 搜索, 6以及
Google 内部计算产品的分叉, 538 Google 的大型测
试, 286手动测试功
能, 210

细分延迟问题, 113
Google Takeout 案例研究, 496-502
Google 网络服务器 (GWS), 209
谷歌维基 (GooWiki) , 189
“Googley” , 41岁
Gradle, 376
 依赖版本, 394 Ant 的改进, 378
greenfield 代码审查, 180 grep 命令, 352群聊, 50

咕噜声, 376

H
 “黑客”或“聪明”的代码, 10
Hamming, Richard, 35, 36幸福,为你的团队追踪, 99
 在办公室外和职业生涯中, 100 次哈希洪水攻击, 9 次哈希排序 (示例) , 9
 次闹鬼墓地, 44,464

心脏出血, 10
 “Hello World” 教程, 196辅助方法共享辅助程序和验证, 254共享值, 252

密封代码,非确定性和, 268密封 SUT, 290好处, 291密封测试, 491

Google Assistant, 492
英雄崇拜, 28隐藏你的工作
 天才神话和, 29有害影响, 30-34巴士因子, 31工程师和办公室, 32
 放弃及早发现缺陷或问题, 31
 进展速度, 32雇用软件工程师会降低雇用标准 (反模式) , 92
 招聘容易被说服的人 (反模式) , 89使多样性可行, 75历史,在代码搜索中建立索引, 367诚实,对你的团队诚实, 98

 “希望不是一种策略” , 89测试中的沙漏反模式, 220

人性化问题,在团队中被忽视, 90人性化问题, 解决, 29谦逊, 35成为“Googley” , 41实践, 36-39混合 SUT, 291

海勒姆定律, 8单元测试中的考虑, 284弃用和, 313
 哈希排序 (示例) , 9

冰淇淋锥体反模式在测试中, 220,287幂等性, 529

IDE (集成开发环境)使用代码搜索的原因, 355-359

 静态分析和,427图像识别, 种族包容和, 70命令式编程语言, 381实施意见, 145, 193重要与紧急问题, 118对现有代码的改进, 代码审查, 181知识共享的激励和认可, 57增量构建, 基于任务的构建系统的难度, 379索引

代码搜索与 IDE, 355从代码搜索索引中删除文件, 366索引存储库的多个版本, 367
Code Search 中的延迟, 360
Code Search 中的搜索索引, 361个人工程师,提高生产力, 124影响,开放, 40未经授权的影响 (案例研究) , 83信息孤岛, 43信息,规范来源, 58-61代码实验室, 60开发人员指南, 59转到/链接, 60静态分析, 61不安全性, 28批评和, 37天才神话中的表现, 29集成测试, 219

智力复杂性 (QUANTS), 131交互测试, 238, 275-280
适当使用, 277最佳实践, 277避免过度指
定, 278仅执行状态改变功能,
277

更喜欢状态测试, 275交互测试的局限
性, 276使用测试替身, 264代码的互操作性,
151行内差异显示字符级差
异, 403内在动机与外在动机, 104
迭代,让您的团队感到舒适,

110

J

使用
Truth 库在测试中进行 Java 断言, 248 javac 编译
器, 373
Mockito 模拟框架, 262阴影, 342第三方 JAR
文件, 373

杰文斯悖论, 21
史蒂夫·乔布斯, 28岁, 92岁
乔丹,迈克尔, 28岁
JUnit, 305

K

库中的关键抽象和数据结构,列表, 158知识共享, 43-67代码
审查的好处, 175
询问社区, 50-52学习的挑战,
43心理安全的关键作用, 46-48增长你
的知识, 48,49提出问题, 48理解背
景, 49通过与他人合作增加知识

ers, 31哲
学, 45可读性流程和代码审
查, 158扩展组织的知识, 56-62

培育知识共享文化, 56-58

建立规范的信息来源, 58-61保持循环, 61-62通
过代码审查进行标
准化指导, 62-66教导他人,
52-56

近藤玛丽119
Kubernetes 集群, 533
个赞, 58
Kythe, 470
与代码搜索集成, 351交叉引用导航, 406

L

登陆页面, 198
大规模变更工具和流程, 148大型测试, 217 (另请参阅更
大规模的测试) 大
规模变更, 372,459-478
原子变更的障碍, 463-465异构性, 464合并
冲突, 463没有闹鬼的
墓地, 464技术限制,
463测试, 465代码审查, 182基于
主干开发的重要性, 343基础设
施, 468-472
变更管理, 470代码库洞
察, 470语言支持, 471

操作 RoseHub, 472政策和文
化, 469测试, 471跳过的大
型测试, 285
流程, 472-477授权, 473变更创建,
473清理, 477分片和
提交, 474-477品质,
460责任, 461-462测
试, 466-468代
码审查, 467搭乘 TAP 列车, 466
scoped_ptr 到
std::unique_ptr, 467大型测
试, 281-309优
势, 282

挑战和限制, 285特点, 281测试保真度, 282大型测试和开发人员工作流程, 304-309编写大型测试, 305运行大型测试, 305-308 Google 的大型测试, 286-289	聘用软弱的人, 89忽视人性问题, 90忽视低绩效者, 89像对待孩子一样对待你的团队, 92询问团队成员是否需要什么, 100
Google 规模和, 288时间和, 286大型测试的结构, 289-296被测系统 (SUT), 290-294测试数据, 294验证, 295大型测试类型, 296-304	工程经理, 86-88失败作为一种选择, 87经理人的历史, 86当今的经理, 87
A/B 差异 (回归), 299浏览器和设备测试, 297部署配置测试, 298灾难恢复和混沌工程, 302	满足团队成员的不同需求, 103动机, 104经理和技术主管, 81-83案例研究, 无权影响, 83工程经理, 82技术主管, 82技术主管经理, 82
探索性测试, 298交互二进制文件的功能测试, 297性能、负载和压力测试, 297	从个人贡献者转变为领导角色, 83-86人们不想成为管理者的原因, 84仆人式领导, 85其他技巧和窍门, 101积极模式, 93-100
探测器和金丝雀分析, 301UAT, 301用户评估, 303单元测试没有提供良好的风险缓解覆盖范围, 283-284执法面部识别数据库, 种族偏见, 74领导力, 聪	成为催化剂, 96成为老师和导师, 97成为禅师, 94诚实, 98放弃自我, 93消除障碍, 96设定明确的目标, 97追踪幸福, 99学习, 46 (另请参阅知识共享)挑战, 43
明的混蛋和, 57领导力, 成长为一个真正优秀的领导者, 107-122	
解决 Web 搜索延迟问题 (案例研究), 110-112	
永远在做决定, 108永远在离开, 112永远在扩大规模, 116决定, 然后迭代, 110确定关键的权衡, 109确定盲点, 109重要问题与紧急问题, 118-119学会放弃, 119保护你的精力, 120领导团队, 81-105反模式, 88-93成为每个人的朋友, 91降低招聘标准, 92	LGTM (对我而言不错)来自审阅者的盖章, 166变更批准, 401代码所有者的批准和, 168正确性和理解性检查, 167来自主要审阅者, 178含义, 412与可读性批准分离, 173技术主管在之后提交代码变更, 168库、编译器和, 373 Tricorder 中的 linters, 425

Linux

开发人员, 28内核补
丁,真相来源, 335
Live at Head 模型, 442负载,
测试, 297日志查看
器,代码搜索集成, 357逻辑,不进行测试, 246

LSC (参见大规模变化)

M

邮件列表, 50测试的
可维护性, 232 “管理综合症”,
84经理和技术主管,
81-83反模式, 88-93成为每个人的朋友,
91降低招聘标准, 92招
聘容易被说服的人, 89忽视人性化
的问题, 90忽视表现不佳的人, 89像对待
孩子一样对待您的团队,
92案例研究,没有权威的影响, 83
工程经理, 82,86-88当代, 87失败
是一种选择, 87经理的历史, 86从个人贡献者
转向领导角色, 83-86人们不想成为经理的原因, 84仆人式
领导, 85积极模式, 93-100成为催化
剂, 96成为老师和导
师, 97成为禅师, 94诚实,
98放弃自我, 93消除障碍, 96
设定明确的目标, 97追踪幸福感, 99技术主管, 82技术主
管经理 (TLM), 82手动
测试, 286

Markdown, 190团
队成员掌握, 104
Maven, 376
对 Ant 的改进, 378测量, 123 (另
请参阅工程生产力,测量
(英)

在难以量化的领域, 20 个中等测试,
217
Meltdown 和 Spectre, 11指
导, 46成为团队的
老师和导师,
97
通过代码审查实现标准化, 62-66
合并冲突、更改大小和463
合并
分支合并流程,开发为,
330
协调开发分支合并, 338开发分支和, 338 VCS 中的合并
跟踪, 329方法驱动测试, 241
示例测试, 241示例方法命名模式, 246
指标

评估测量的价值, 125-128在GSM框架中, 129,
132有意义的,选择目标和信号,

129-130
使用数据进行验证, 133-137弃用过程中
的迁移, 322
从过时的系统迁移用户,

317
弃用流程的里程碑, 320
最低版本选择 (MVS), 447移动设备,浏览器和设备测
试, 297模拟, 257 (另请参阅测试替身)交互测试和, 264模
拟对象的误用,导
致测试脆弱,

224
模拟变得陈旧, 283模拟框架,
261适用于主要编程语言,
262通过交互
测试完成, 264过度依赖, 239,259通过存根, 264
模拟测试, 265

Mockito 使
用示例, 262存根示例,
263模块,处理构建系统,
390-396

管理依赖关系, 392-396

最小化模块可见性, 392使用细粒度模块和
1:1:1 规则, 391
monorepos, 345反
对意见, 346组织引用其优点,
346激励你的团队, 103内在动机与外在动机,
104

代码块的移动检测, 403多元文化能力,建设,
72-74
社会不平等如何影响工作场所
ces, 74
多机 SUT, 291多租户.容器化
和, 521-522用于服务作业的多租户, 534

多租户框架服务器, 540

N
个命名资源,在机器上管理,
531
网络端口、容器和, 531新闻通讯, 61没有完美的
二进制, 509非状态
改变函数, 278测试中的非确定性
行为, 216, 218,

267
来自批评的通知, 402

O
办公时间,用于知识共享, 52 1:1:1 规则, 391一次性代
码, 529

单版本规则, 340, 342, 394 monorepos
和, 345
开源软件 (OSS) 依赖管理和, 430
monorepos 和, 347开源 gflags, 452

玫瑰中心操作, 472现有代码优
化,代码审查, 181交互测试过度规范, 278代码所有权,
169-170
弃用流程所有者, 320绿地审查, 180 Google
monorepo 中的细粒度所有权,

340
拥有大型测试, 308

磷

Pact 合约测试, 293
Pants, 380并
行化构建步骤难度在基于任务的系统
中, 378在 Bazel 中, 383并行化测试, 267鸚鵡学
舌, 44

帕斯卡·布莱斯, 191耐
心而友善地回答问题,
49
耐心,学习, 39同伴奖励,
58
Perforce,修订版本号发生变化, 336代码库中的性能优化,
151测试, 297软
件工程师的性能 绩效评级中的缺陷, 76忽略低绩效者,
89人员成
本, 18

“彼得原理”, 84
派珀, 340
代码搜索集成, 353 种工具构建于其之上,
406 种大规模变更政策, 469 种
礼貌和专业的代码审查,

176
事后分析,无可指责, 39-41,88提交前审查,
400提交前, 179 Tricorder 检
查, 425持续测试
和, 485大型测试的基础设施,
305优化,490,494在开发分支中合并
测试, 338与提交后测试相比, 486探测
器, 301划分问题空间的问题,
113-116重要与紧急, 118产品稳定性,开发分支
和, 337测试中的生产风险,
292测试, 487
代码审查中
的专业性, 176编写巧妙的代码和, 10

软件工程与, 3, 23编程指导, 157编程语言建议
更难得到正确的领域,

158

避免使用容易出错和令人惊讶的结构, 149 个新功能的细分
和建议

使用它们, 158记录,

202命令式和功能性,

381对新功能和尚未充分理解的功能的限制, 152逻辑, 246参考文档, 193每种语言的样式指南, 142支持大规模更改,

471项目健康

(pH) 工具, 228 Tricorder 中的项目级自

定义, 425 Proto 最佳实践分析器, 424协议缓冲静态分析, 425提供商,文档, 192代码审查

的心理益处, 174心理安全, 46-48通过

指导进行建设, 46通过建设催化您的团队, 87在大群体中,

47缺乏, 43公共计算服务与私人计算服务,

543公共 API, 237团队成员的目的, 105文档用户的目的, 191 Python, 28 unittest.mock 框架, 262 Python 风格指南避免使用诸如反射之类的强力功能, 149 CamelCase 与 snake_case

命名, 154代码缩进, 149

Q

定性指标, 133 CD 中的质量和

用户关注度, 511代码质量, 131

工程生产力指标中的 QUANTS,

130

在可读性过程研究中, 134查询相关信号,

364查询独立信号, 363

问答系统 (YAQS), 51 个问题,询问 (参见询问问题)

R

面部识别数据库中的种族偏见, 74种族包容性, 70

Rake, 376代

码搜索中的排名, 363-365查询相关信号, 364

查询独立信号, 363结果多样性, 365检索,

365

RCS (修订控制系统), 329,332可读性, 56,62-66 Google代码变更

更审批, 168通过代码审查确保,

173可读性流程, 56关于, 63优点, 64实际实现,使用而不是测试替身, 264-269:决定何时使用实际实现, 266-269

依赖关系构建, 268测试中的确定性, 267

执行时间, 267宁愿现实主义也不

愿孤立, 265回忆偏差,

134近因偏差, 134对知识共享的认可, 57关于研究

结果的建议, 137记

录 / 重放系统, 293,

492文档中的冗余, 202重构, 233代码审查, 182大

规模和使用参考进行排名, 364基于搜索和替换, 360未提交

的工作类似于分支, 337参考文档, 193-195

类注释, 194文件注释, 194函数注释, 195参

考,用于排名, 363回归

测试, 299 (另请参阅 A/B

差异测试)正则表达式 (regex) 搜索, 368重新实现 / 分叉与

添加依赖

项, 22

发布分支, 339

谷歌和344 个候选版本

测试, 486 个版本力求敏捷,建立发布列车,

509

满足发布期限, 510没有二进制文件是完美的,

509外部依赖项的可靠性, 396分

布式构建中的远程缓存, 386

Google 的远程缓存, 389分布式构建的远程执行, 387

Google 远程执行系统 Forge、389 个存储库、328 个 DVCS 项目的中央存储库,

333

细粒度与 monorepos 相比, 345存储库分支, Google 未使用, 223代表性测试, 512资源限制,CI 和, 490尊重, 35成为 “Googley”, 41实践, 36-39,57搜索结果多样性, 365检索, 365代码审阅者, 保持在最低限度, 179调整规模和自动缩放, 522风险

让失败成为一种选择, 87独自工作, 30障碍,移除, 96回滚, 181

Rosie工具, 470 LSC

进程中的分片和提交, 474-477

规范代码的规则,样式指南中的141类规则

建立一致性的规则, 153执行最佳实践的规则, 152避免危险的规则, 151未涵盖的主题, 153更改, 154-157执行, 158-163 gofmt 案例研究, 161-163使用代码格

式化程序, 161使用错误检测器, 160指导原则,

143-151避免容易出错和令人惊讶的构造, 149

保持一致, 146向实用性让步, 150为代码阅读器而不是作者进行优化, 144规则必须发挥作用, 144拥有的理由, 142规则,在 Bazel 中定义, 384

S

抽样偏差, 134沙盒密封测试和, 492 Bazel 的使用, 384满意度 (QUANTS), 131可扩展性 分叉和, 22静态分析工具, 418规模 在时间和之间做出决定, 22在软件工程中, 4软件工程中的问题, 5规模和效率, 11-17编译器升级 (示例), 14-16在开发人员工作流程中尽早发现问题, 17不可扩展的策略, 12可扩展性好的策略, 14通过代码库的一致性实现扩展, 147规模对代码搜索设计的影响, 359-361

调度,自动化, 519,524测试范围, 219-221, 281定义单元范围, 237最小可能测试, 289 C++ 中的 scoped_ptr, 467评分更改, 413接缝, 260代码搜索中的搜索索引, 361搜索查询延迟,代码搜索和, 359外部依赖项的安全性, 396应对威胁和漏洞, 10外部依赖项引入的风险,

385

种子数据, 294寻求者 (文档), 191自信, 36自动驾驶团队,建设, 112-116

语义版本字符串， 394语义版本控制， 440限制， 443-449	总结性想法， 549-550弃用和， 311编程与， 3,23版本控制系统和， 329规模和效率， 11-17时间和变化， 6-11权衡和成本， 18-23软件工程师代
最低版本选择， 447动机， 446过度约束， 444	码审查和， 170办公室， 32源代码控制依赖管理和， 430
过度承诺兼容性， 445质疑它是否有效， 448	
SemVer（参见语义版本控制）仆人式领导， 85无服务器， 539-543关于， 540优缺点， 541无服务器框架， 541	Git 作为主导系统， 333将文档移动到， 189事实来源， 334-336
服务,连接到软件进行管理	一个版本作为单一事实来源， 340个场景,没有明确的事实来源， 335个正在进行的工作和分支， 336个稀疏n-gram 解决方案,代码中的搜索索引
计算,528服务作	
业,526多租户, 534	
着色（Java 中）、 342 LSC	搜索， 362构建
进程中的分片和提交， 474-477	系统速度， 371加速测试， 305
共享环境 SUT， 291 shell 脚本,用于构建， 373左移， 17,32尽早做出数据驱动的决策， 512仅运送需要使用的内容， 511信号定义， 129	Spring Cloud Contracts， 293堆栈框架,代码搜索集成， 357分阶段推出， 512标准化,缺乏,在更大的测试中， 285状态测试， 238优于交互测试， 275
目标/信号/指标 (GSM) 框架， 129	状态,管理， 527状态改变函数， 277静态分析， 417-428有效,特征， 418-419可扩展性， 418
单点故障 (SPOF)， 44领导者， 112单机 SUT， 291单进程	可用性， 418示例， 417使其工作,关键经验教训， 419-421授权用
SUT， 290使用 LSC 对代码库进行小修复， 462小测试， 216,231， 281社交互动	户做出贡献， 420
	关注开发人员的幸福
	感， 419使静态分析成为核心开发的一部分
成为“谷歌人”， 41指导	
低绩效员工， 90团队互动模式， 47实践中的谦逊、尊重和信任， 36-39	操作工作流程, 420
	Tricorder 平台， 421-427编辑和浏览
	代码时进行分析， 427编译器集成， 426集成反馈渠道， 423
支柱， 34为什么	集成工具， 422每个项目的定制， 424预提交， 425建议的修复， 424
支柱很重要， 35社交技能， 29社会成本， 18软件工	
程 巧妙的代码和， 10	

静态分析工具, 61代码正确性, 172静态依赖模型, 439 C++ 中的 std::unique_ptr, 153,468路灯效果, 129压力测试, 297存根, 263, 272-275

适当使用, 275过度使用的危险, 273文档, 192风格仲裁者, 156代码风格指南, 59,141

制定规则的优势, 142应用规则, 158-163规则类别, 151建立一致性的规则, 153执行最佳实践的规则, 152避免危险的规则, 151未涵盖的主题, 153更改规则, 154-157对规则做出例外, 156流程, 155风格仲裁者, 156创建规则, 143-151

指导原则, 143-151每种编程语言, 141编程指导, 157子字符串搜索, 369 Subversion, 332成功,循环, 116 基于后缀数组的解决方案,在代码搜索中搜索索引, 362补充检索, 365可持续性代码库, 12分叉和, 22对于软件, 4系统测试, 219被测系统 (SUT), 290-294处理依赖但辅助的服务, 293示例, 290测试对行为的保真度, 282在交互二进制文件的功能测试中, 297更大的测试, 288生产与隔离密封 SUT, 305减少问题测试边界的大小,

生产和 Webdriver 测试的风险躯干, 292范围,测试范围和, 289播种 SUT 状态, 294行为验证, 295

电话

TAP (参见测试自动化平台)基于任务的构建系统, 376-380阴暗面, 378维护和调试构建脚本的难度, 379并行化构建步骤的难度, 378执行增量构建的难度,

379 时间、规模和权衡, 390 老师和导师, 97 团队 巩固团队身份, 115工程师和办公室,意见, 32 天才神话和, 28领导, 81 (另见领导团队)软件工程作为团队努力,

34-42 成为“Googley”, 41无可指责的事后文化, 39-41实践中的谦逊、尊重和信任, 36-39社交互动的支柱, 34为什么社交互动支柱很重要, 35技术主管 (TL), 82技术主管经理 (TLM), 82技术讲座和课程, 52技术名人现象, 29技术评论, 199技术作家,撰写文档, 204 节奏和速度 (QUANTS), 131

测试自动化平台 (TAP), 223,494-496罪魁祸首查找, 495故障管理, 495提交前优化, 494资源限制和, 496测试 LSC 分片, 475训练模型和 LSC 测试, 466大型测试的测试数据, 294测试替身, 219,257-280在 Google, 258示例, 259

伪造, 269-272对软件开发的影响, 258交互测试, 275-280模拟框架, 261接缝, 260存根, 272-275使用技术, 262-264伪造, 263交互测试, 264存根, 263
不忠实, 283在脆弱交互测试中使用, 238使用真实实现代替, 264-269

决定何时使用真实实现, 266-269
宁愿现实主义,也不愿孤立, 265测试基础设施, 255测试不稳定性, 491测试范围 (参见测试范围) 测试规模, 215在实践中, 219大型测试, 217,281中型测试, 217所有规模的共同属性, 218小型测试, 216测试范围和, 220单元测试, 231测试套件, 208大型,陷阱, 224测试流量, 294可测试性 可测试代码, 260尽早编写可测试代码, 261测试, 207-230
作为原子更改的障碍, 465
在 Google 规模上, 223-225自动化,限制, 229自动化以跟上现代发展, 210测试代码的好处, 213-214持续集成和, 480 CI 中的持续测试, 485-487设计测试套件, 214-223

Beyoncé 规则, 221代码覆盖率, 222测试范围, 219-221测试规模, 215密封, 491

Google 的历史, 225-229当代测试文化, 228入门课程, 226
测试认证程序, 227
厕所测试 (TotT), 227大规模变更基础设施, 471大规模变更中的较大 (参见更大规模的测试), 466-468编写测试的原因, 208-214

Google Web 服务器,故事, 209测试假货, 272编写,运行,在自动化测试中做出反应, 212-213

马桶测试 (TotT), 227 次测试

因过度使用存根而变得脆弱, 273
过度使用存根会导致效果降低, 273过度使用存根会导致不清晰,
273
使其易于理解, 307过度使用存根,示例, 274重构以避免存根, 274加速, 305第三方目录, 437时间

在时间和规模之间做出决定, 22在版本控制系统中, 329更大的测试和时间的流逝, 286软件项目中的时间和变化, 6-11旨在不做任何改变, 10哈希排序 (示例), 9

海伦定律, 8程序的生命周期, 3
TL (参见技术主管)
TLM (参见技术主管经理)基于令牌的搜索, 368工具链,由 Bazel 使用, 384
Torvalds,Linus, 28可追溯性,维护指标, 130跟踪代码更改历史,批评,

414
工作跟踪系统, 119权衡成本/收益, 18-23在时间和规模之间做出决定 (示例), 22分布式构建 (示例), 20

决策中的错误, 22白板笔(示例), 19对于
领导者, 109工程生产力, 130网络搜索延迟案
例研究, 111关键,识
别, 109传递依赖关系, 392外部, 395严格,
执行, 393部落知识, 45

Tricorder 静态分析平台, 322,421-427在编辑和浏览代码时进行分
析, 427
编译器集成, 426新检查标准, 422
集成反馈渠道, 423集成工具, 422每
个项目的自定义, 424提交前检查, 425建议的
修复, 424基于三元组的方法,代码中的搜索索引

搜索, 361基于
主干的开发, 327,339与良好技术成果的相关性, 339
Live at Head 模型和442高性能组织与
343之间的预测关系

源代码控制问题和, 429信任, 35成为

“Googley”, 41代码审查和,
400练习, 36-39像对待孩子一
样对待您的团队(反模
式), 92信任您的团队并放弃自我, 93脆弱性和, 40

真值断言库, 248教程, 196糟糕教程
的示例, 196示例,
糟糕的教程变得更好, 197

乌
UAT (用户体验测试), 301
用户界面
服务 UI 到后端的端到端测试
结束, 292
在相当小的 SUT 示例中, 288测试,不可靠且成本
高昂, 292

不变测试, 233单元测试,
231-256单元测试中的常见缺
陷, 283-284配置问题, 283突发行为和真空效
应, 284

负载下出现的问题, 284意外行为、输入和
副作用, 284不忠实的测试替身, 283

测试的执行时间, 267测试软件的生命周
期, 286单元测试的局限性, 282测试的可
维护性、重要性, 232窄范围测试(或单
元测试), 219防止脆弱测试, 233-239良好单元测试的属
性, 285测试和代码共享,DAMP,而不是DRY, 248-255

DAMP 测试, 250定义
测试基础结构, 255共享帮助程序和验证,
254共享设置, 253共享值, 251编写清晰的测
试, 239-248将逻辑排
除在测试之外, 246使测
试完整而简洁, 240测试行为,而不是
方法, 241-246编写清晰的失败消息,
247单元(在单元测试中), 237

Unix,开发人员, 28不可重现的
构建, 385升级, 4编译器升级示例,
14-16软件项目
的生命周期和重要性, 6静态分析的可用性,
418用户评估测试, 303 CD 中的用户焦点,仅运送使用过的
内容,

511
用户
工程师为所有用户开发软件, 72首先关注偏见和歧视影响最
大的用户, 78将对用户群体的考虑推迟到开发后期, 76

V
真空效应,单元测试和, 284

验证,共享帮助者, 254公平工程中的价值观与结果, 77

范罗苏姆,吉多, 28岁

VCS (版本控制系统), 327 (另请参阅版本控制)在细粒度存储库之间进行混合

和 monorepos, 346早期, 329速度是一项团队运动, 507供应商业化项目的依赖项, 396版本控制, 327-336关于, 328在Google, 340-345一些长期存在的分支, 343

单一版本规则, 340,342

发布分支, 344场景,多个可用版本, 341分支管理, 336-339集中式与分布式 VCS, 331-334与依赖管理相比, 336未来, 346重要性, 329-331 monorepos, 345真相之源, 334-336 虚拟机 (VM), 524

用于多租户计算服务中的隔离, 521虚拟单一存储库 (VMR), 346,347可见性,最小化用于构建系统中的模块, 392漏洞,显示, 40

西

Web 搜索延迟案例研究, 110-112 Webdriver Torso 事件, 292明确指定的交互测试, 279谁、什么、何时、何地 and 为什么问题,在文档中回答, 201工作区与全局存储库的差异, 368 本地,代码搜索支持, 362批评与批评之间的紧密集成, 406 撰写评论 (针对技术文档), 199

是

YAQS (“又一个问题系统”), 51

是

禅师, 94岁

关于作者

Titus Winters自 2010 年起就职于 Google,担任高级软件工程师。目前,他是 C++ 标准库设计全球小组委员会主席。在 Google,他是 Google C++ 代码库的库负责人:12,000 名不同的工程师在一个月内编辑 2.5 亿行代码。过去七年来,Titus 和他的团队一直在使用现代自动化工具来组织、维护和发展 Google C++ 代码库的基础组件。在此期间,他启动了多个 Google 项目,这些项目被认为是人类历史上最大的十大重构之一。在帮助构建重构工具和自动化的过程中,Titus 直接接触了工程师和程序员为了“让某些东西正常工作”而可能采取的大量捷径。这种独特的规模和视角影响了他对软件系统的维护和支持的所有思考。

Tom Manshreck自 2005 年起担任 Google 软件工程部门的技术撰稿人,负责开发和维护 Google 基础设施和语言方面的许多核心编程指南。自 2011 年起,他成为 Google C++ 库团队的成员,负责开发 Google 的 C++ 文档集、与 Titus Winters 合作推出 Google 的 C++ 培训课程,以及记录 Google 的开源 C++ 代码 Abseil。Tom 拥有麻省理工学院的政治学学士学位和历史学学士学位。在加入 Google 之前,Tom 曾在 Pearson/Prentice Hall 和多家初创公司担任总编辑。

Hyrum Wright是 Google 的一名高级软件工程师,自 2012 年以来一直在该公司工作,主要负责 Google C++ 代码库的大规模维护。

海勒姆对 Google 代码库进行的个人编辑比公司历史上任何其他工程师都要多,并且领导着 Google 的自动变更工具组。

Hyrum 获得了德克萨斯大学奥斯汀分校的软件工程博士学位,还拥有德克萨斯大学的硕士学位和杨百翰大学的学士学位,并且偶尔担任卡内基梅隆大学的客座教员。他是会议上的活跃演讲者,也是软件维护和演进学术文献的贡献者。

版权页

《谷歌软件工程》封面上的动物是红鹮（*Phoenicopterus ruber*）。这种鸟主要分布在中美洲和南美洲海岸附近以及墨西哥湾,但它们有时会飞到美国佛罗里达州南部。红鹮的栖息地包括泥滩和沿海咸水泻湖。

火烈鸟标志性的粉红色羽毛是在它们长大后形成的,它的食物中含有类胡萝卜素。由于这些色素更容易在它们的天然食物中找到,野生火烈鸟的羽毛往往比圈养的火烈鸟更鲜艳,尽管动物园有时会在它们的食物中添加补充色素。火烈鸟通常高约 42 英寸,翼尖呈黑色,翼展约 5 英尺。火烈鸟是一种涉水鸟,有带蹼的三趾粉红色脚。虽然雄性和雌性火烈鸟之间没有共同的区分,但雄性火烈鸟往往体型稍大一些。

火烈鸟是滤食性动物,它们用长腿和长颈在深水中觅食,一天中的大部分时间都在寻找食物。它们的喙内有两排鳞片,这些鳞片是梳状的刷毛,可以过滤种子、藻类、微生物和小虾等食物。火烈鸟生活在多达 10,000 只的大群中,当它们在一个地方吃完所有食物后就会迁徙。除了是群居鸟类外,火烈鸟的叫声也非常响亮。它们会发出定位呼叫来帮助寻找特定的配偶,还会发出警报呼叫来警告更大的群体。

尽管美洲红鹮曾被认为与大红鹮（*Phoenicopterus roseus*）属于同一物种,后者可能分布在非洲、亚洲和南欧,但现在美洲红鹮被认为是一个独立的物种。虽然美洲红鹮目前的保护状况被列为无危,但 O'Reilly 封面上的许多动物都濒临灭绝;它们对世界都很重要。

封面插图由 Karen Montgomery 绘制,基于卡塞尔自然史的黑白版画。封面字体为 Gilroy Semibold 和 Guardian Sans。文本字体为 Adobe Minion Pro;标题字体为 Adobe Myriad Con -dense;代码字体为 Dalton Maag 的 Ubuntu Mono。

The O'Reilly logo, featuring the word "O'REILLY" in a bold, white, sans-serif font, with a registered trademark symbol (®) to the upper right of the "Y".

O'REILLY®

还有更多这样的事。

体验书籍、视频、在线直播培训课程以及来自
O'Reilly 和我们 200 多个合作伙伴的更多内容
所有内容均可在一个地方获取。

了解更多信息请访问 oreilly.com/online-learning