百花齐放，锄其九九——Twitter的技术坎坷之路

作者 臧秀涛 发布于 2015年12月19日 | 讨论

12月18日，年末技术盛会ArchSummit北京2015正式召开。Twitter Senior Staff Engineer王天做主题演讲《百花齐放，锄其九九——Twitter的技术坎坷之路》，分享了Twitter在技术演进过程中遇到的挑战和解决之道。本文即根据演讲内容整理而成。

王天，2005年7月加入Google，从事移动搜索、新闻搜索、搜索质量等工作；2011年3月加入Twitter搜索部门，工作至今。他主要带领Twitter的搜索质量团队，改进实时搜索产品。

首先，王天通过一组数字分享了Twitter的一些信息：

- 微博客始祖，成立于2006年
- 3.2亿月活跃登录用户
- 10亿月活跃独立访问用户（包括网站嵌入推文）
- 80%流量来自移动设备
- 79%流量来自美国以外
- 每日数亿条，每年逾2000亿条推文
- 4300名员工，其中44%为工程师

Twitter是一个和世界息息相关的实时信息平台，经常会遇到一些可预测或不可预测的事件，从而面临很大压力。

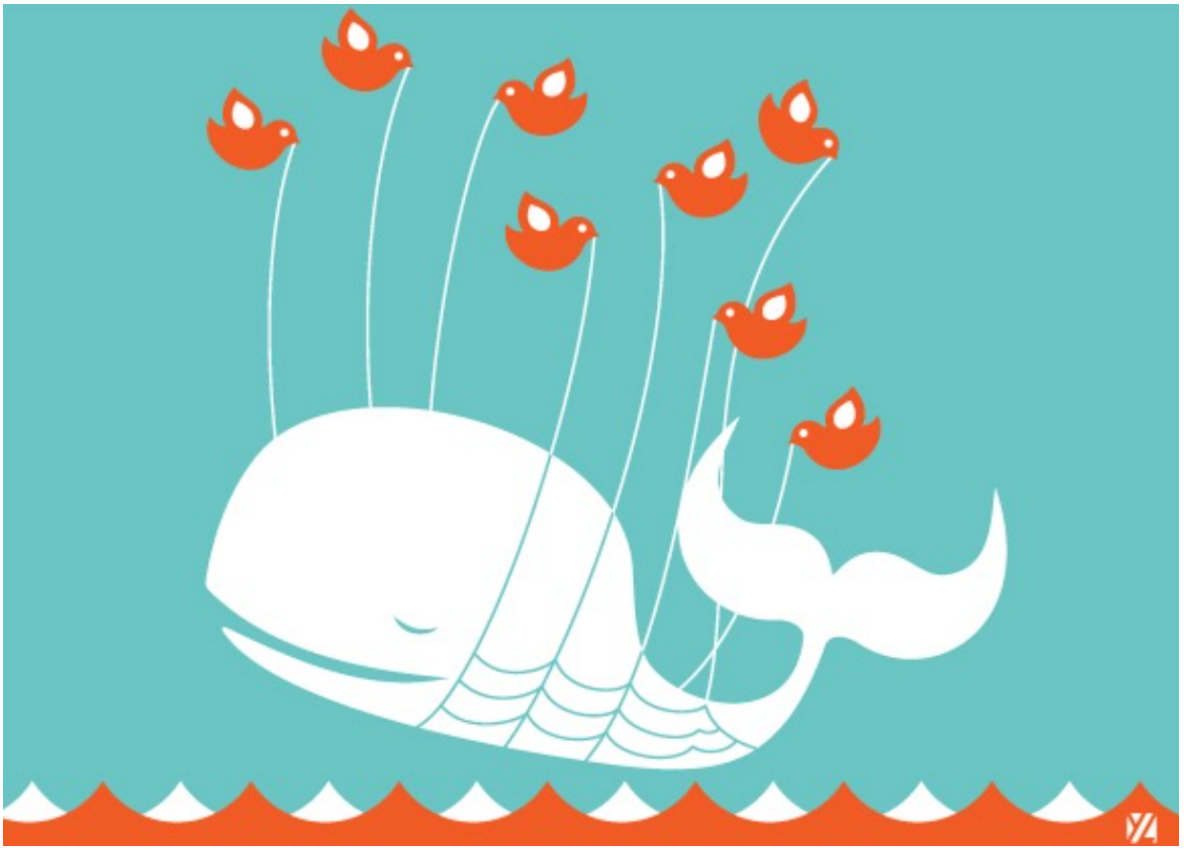
可预测的，比如：

- 奥运会开闭幕式
- NBA决赛
- NFL决赛
- 奥斯卡颁奖
- 日本《天空之城》重播 (2013.8)

不可预测的，比如：

- 日本海啸 (2011.3)
- 世界杯德国巴西半决赛 (2014.7)
- 奥斯卡Ellen自拍事件 (2014.3)
- 巴黎恐怖袭击
- 巴西音乐节的奇怪网站

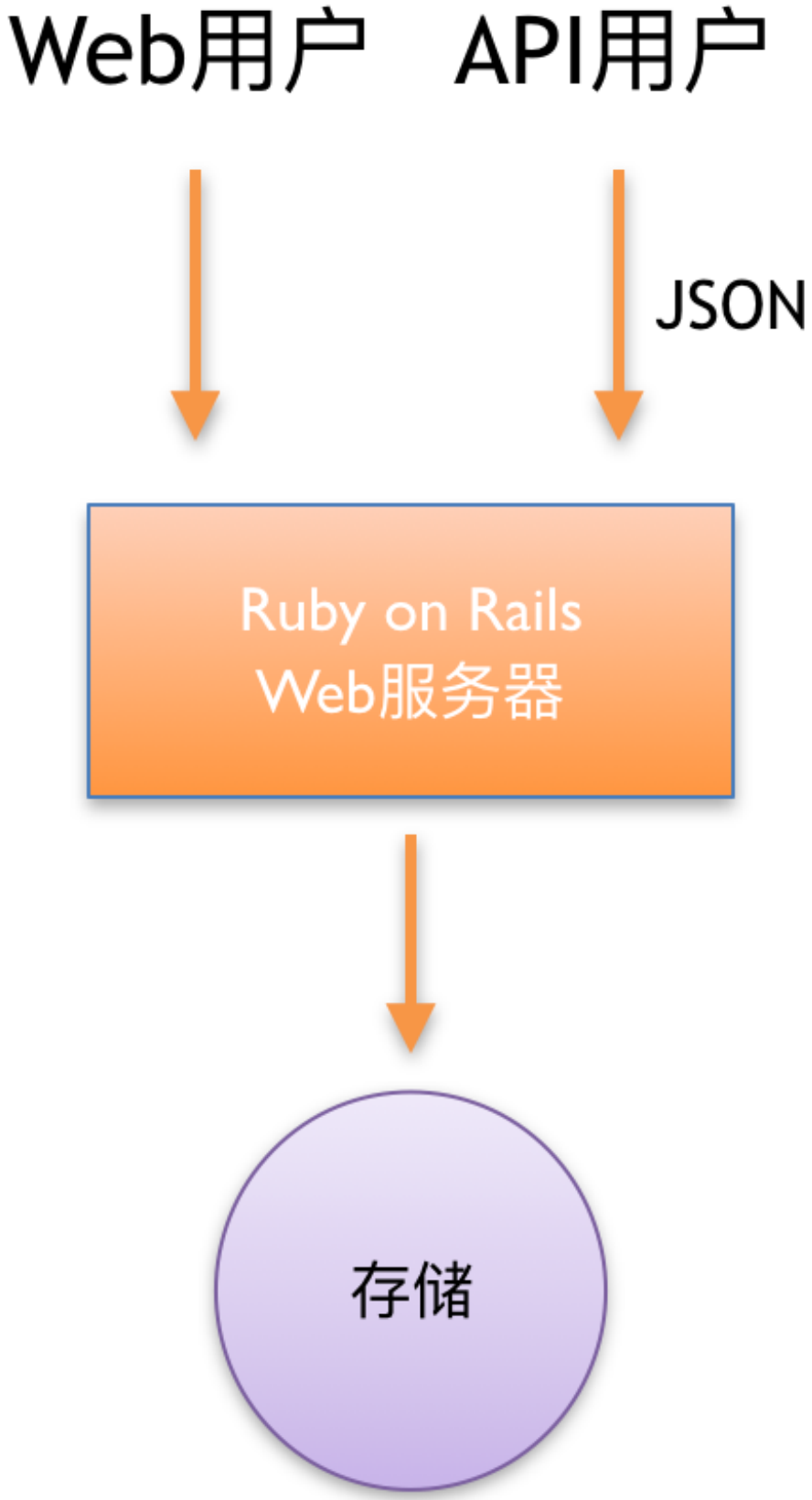
在去年的奥斯卡颁奖典礼现场，主持人Ellen Lee DeGeneres在Twitter上发了一张全明星自拍照，很多人去搜索、转发，给Twitter造成很大压力，致使系统宕机一段时间。之后，很多人又会去搜索Twitter宕机情况，情形进一步恶化。当遇到峰值无法应对的情况，则会出现Twitter特有的报错页面——Fail Whale。



Twitter的技术历史：从远古到现代

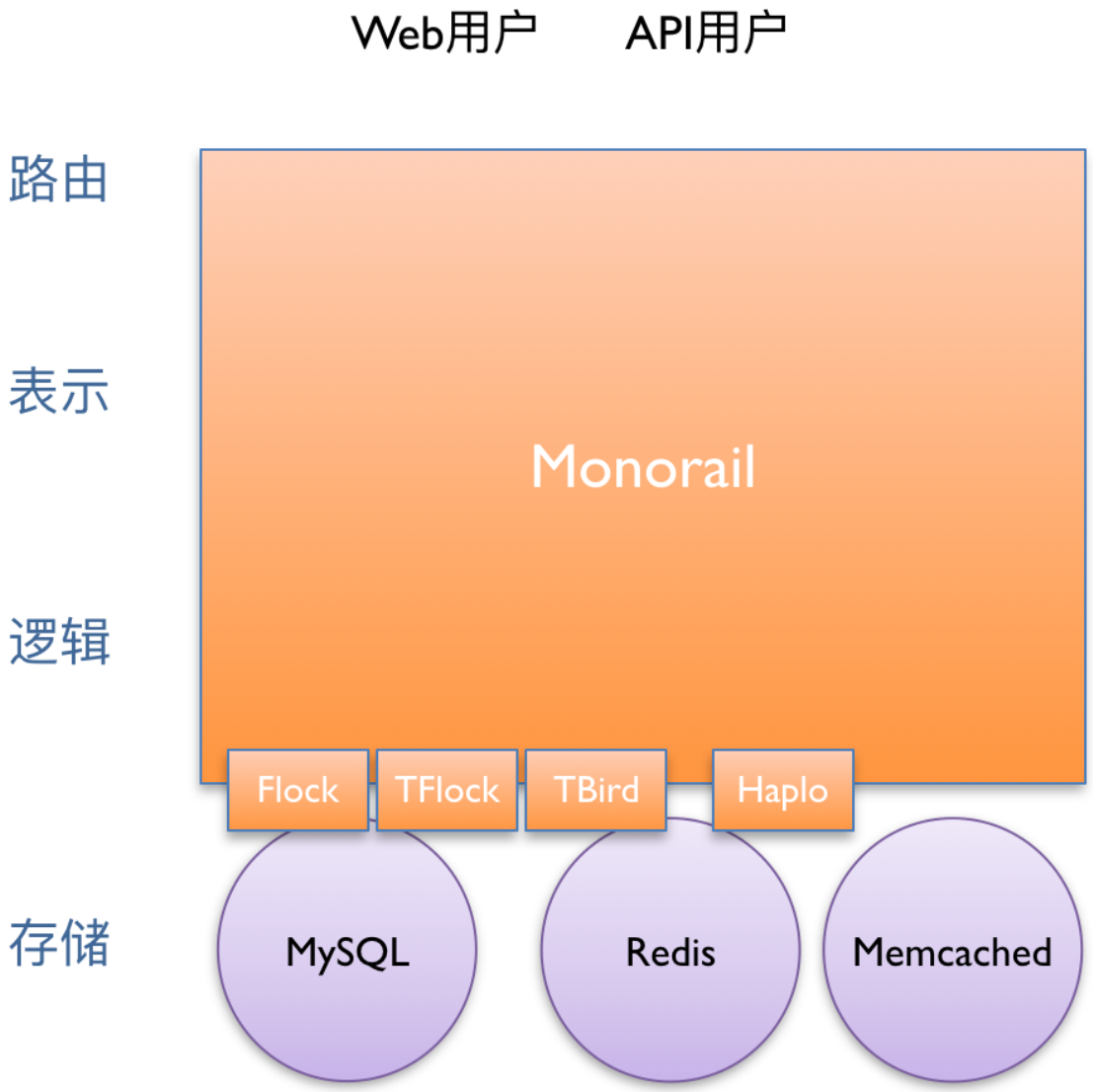
从2006年到2015年，回顾Twitter架构的十年演进之路，可以大概分为远古、古代、近代和现代4个时代来看。

1.远古时代



最初创办时，创始人Jack Dorsey考虑过用Python、C和OCaml编写。不过机缘巧合，他找到了Ruby on Rails的核心贡献者Florian Weber。所以Twitter选择了用RoR实现。

2.古代



随着Twitter用户规模不断增长，其Ruby on Rails部署规模已经是世界第一，最多时机器达到3000台。如图所示，所有逻辑都在Monorail中。当时有超过200名工程师往里面check in代码。难以加入新功能，发布周期很长。

这个架构存在的问题是：

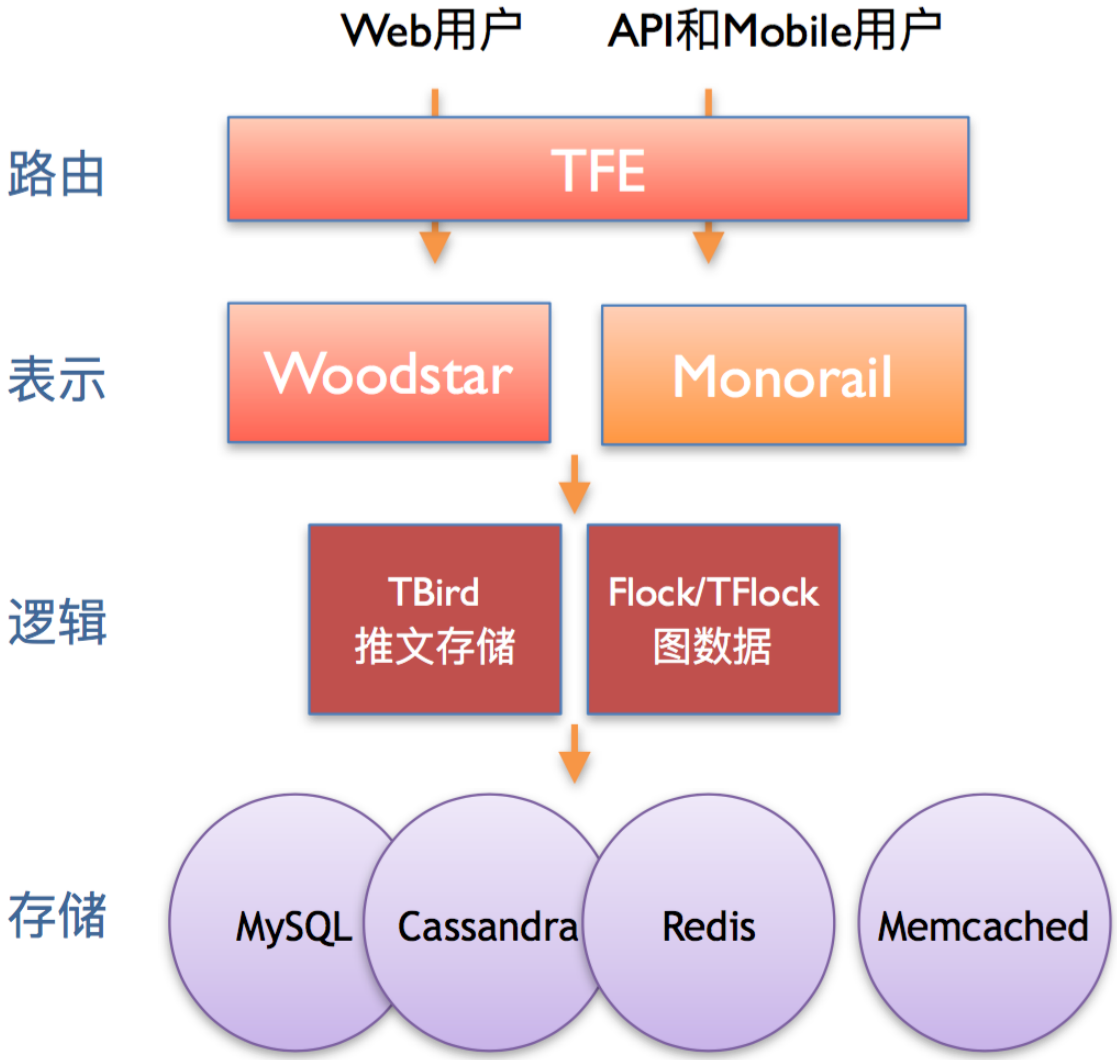
- 效率低下，延迟长，同步处理请求
- 单一数据库，热点明显
- 性能改善缓慢，增添机器的无底洞

当时没有很好地挺过2010年世界杯的考验。

技术债累积迅速，这也是很大的一个问题。

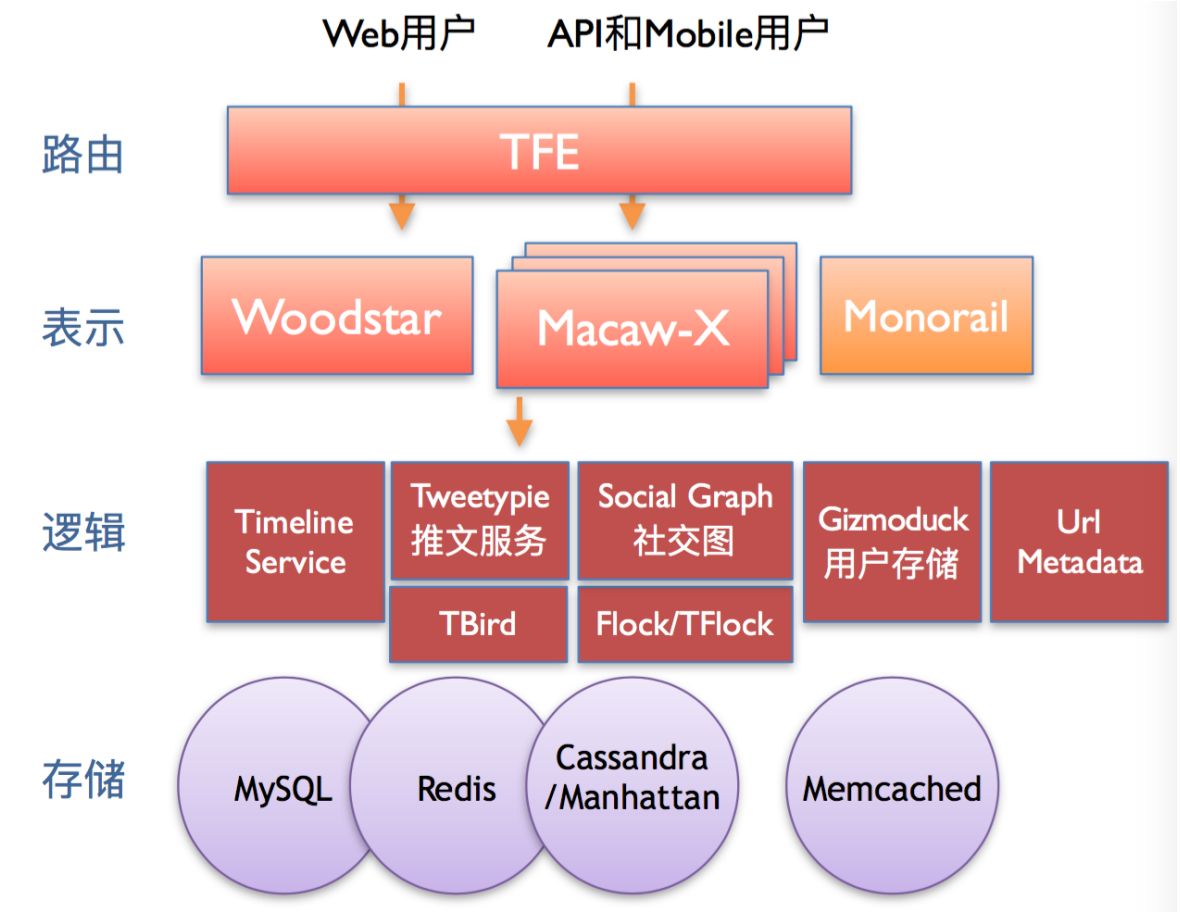
3.近代

这一阶段可以用两张图表示。



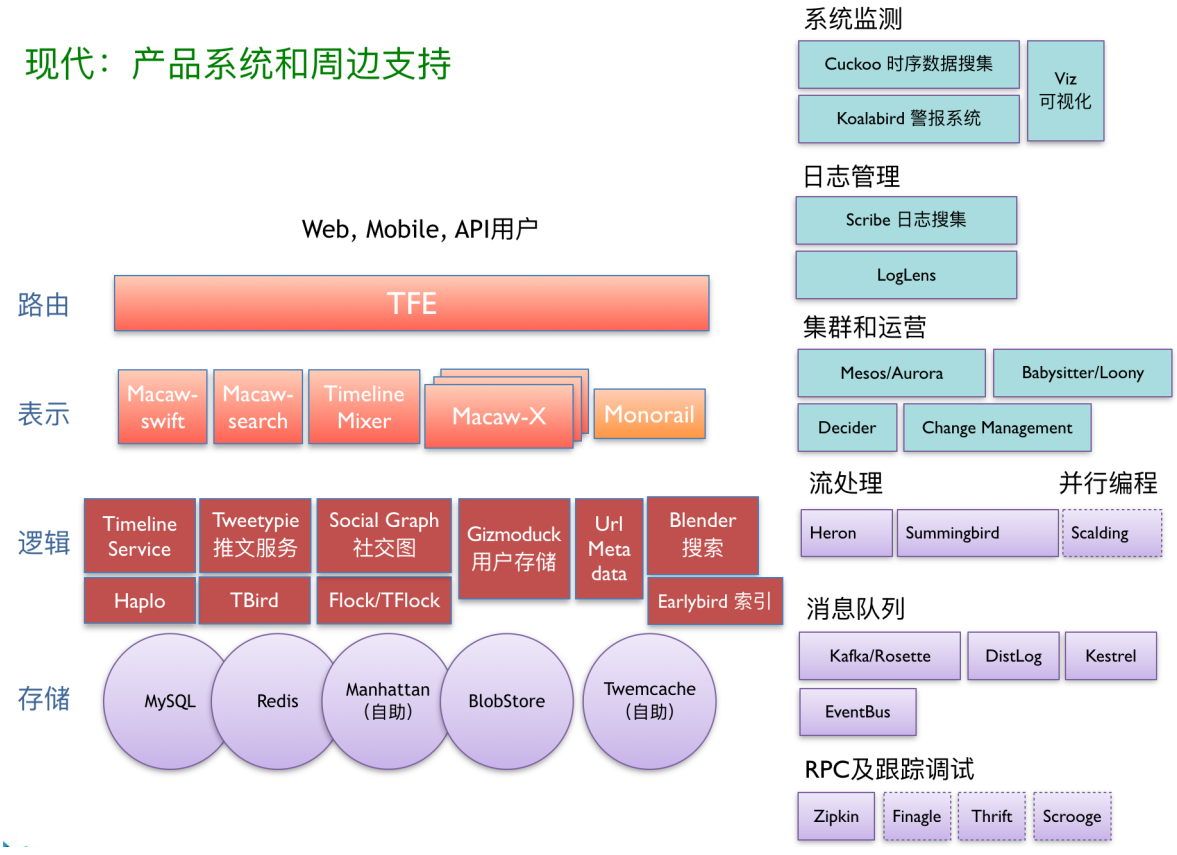
为减少耦合，对系统进行拆分。为提高效率，用Scala重写了服务器。在网络异步编程方面，开发了Finagle，这是基于Netty的一个异步编程库，也是用Scala编写的。存储方面，尝试创建较为高级的数据服务。

经过进一步分解，Monorail逐渐被分离出来。更多业务被分解出来；团队围绕模块组织。

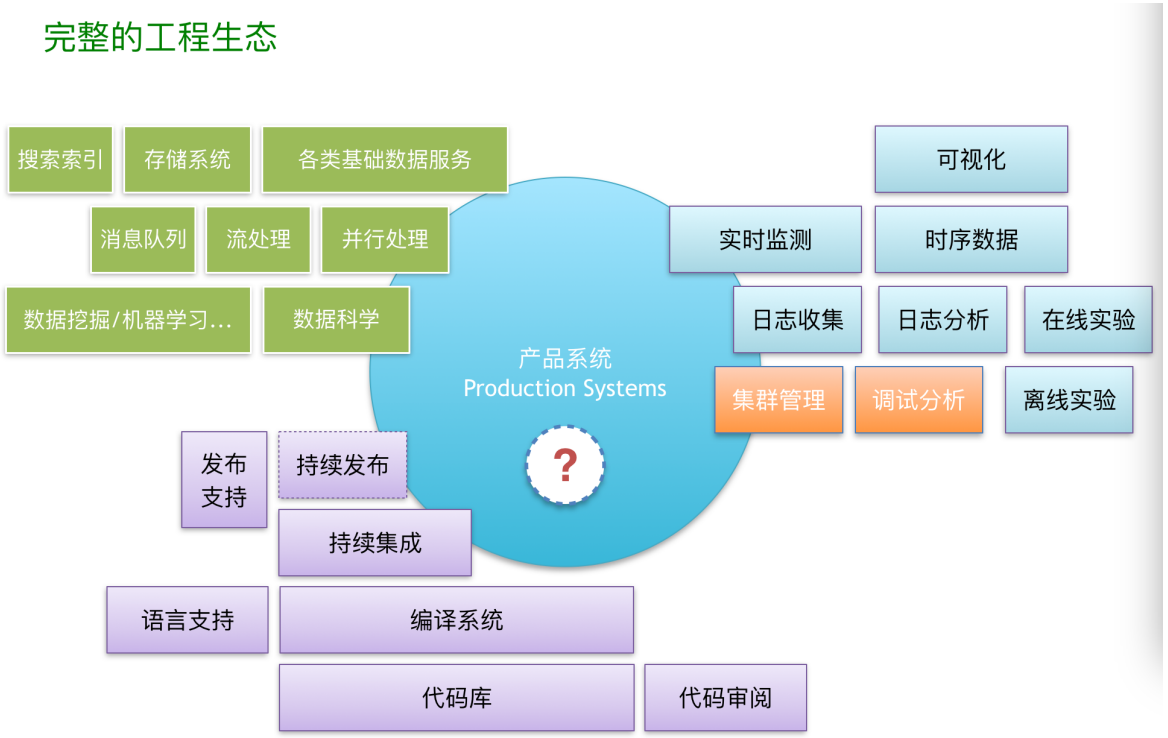


这个时期最重要的事情就是Monorail退休了。整个系统从Ruby平台迁移到JVM上。单机QPS处理能力从200~300提高到10000~20000，延迟减小到1/3；减少了90%资源使用。

4.现代：产品系统和周边支持



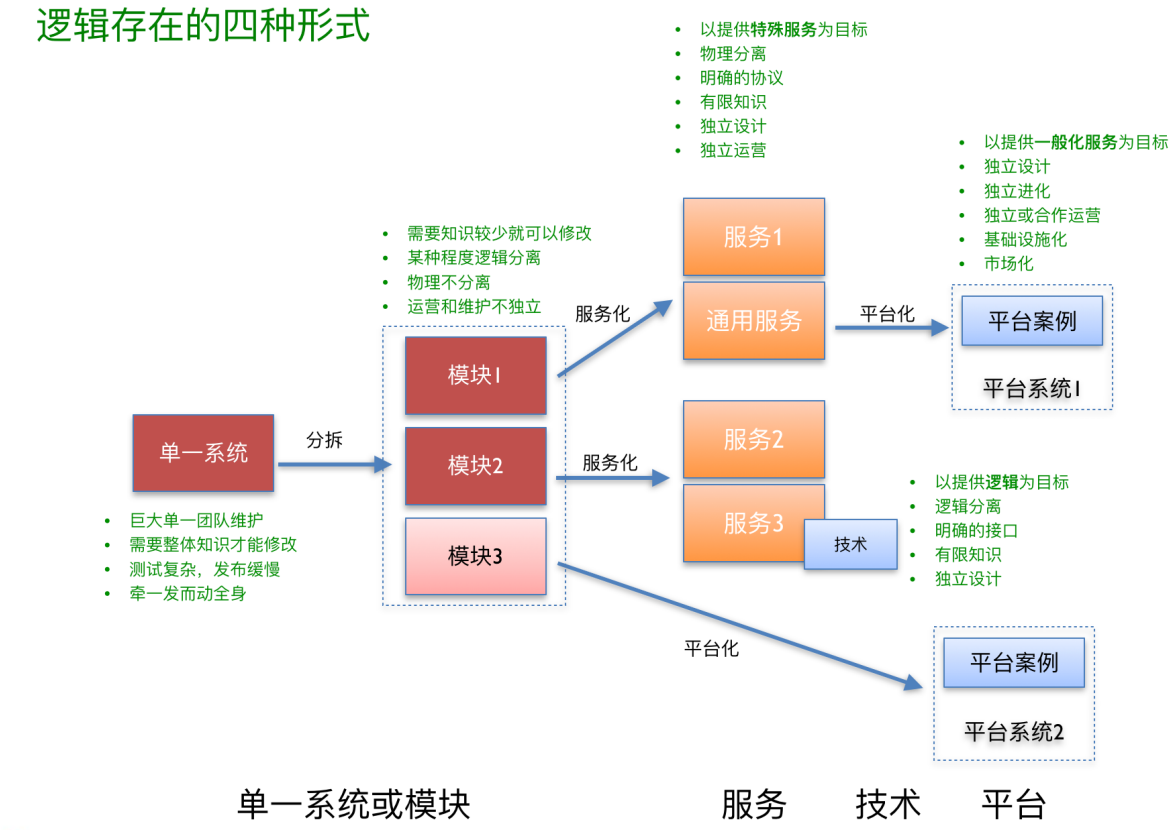
走到这一步，实际经过了非常多的系统拆分。时至今日，Twitter已经搭建起完整的工程生态。



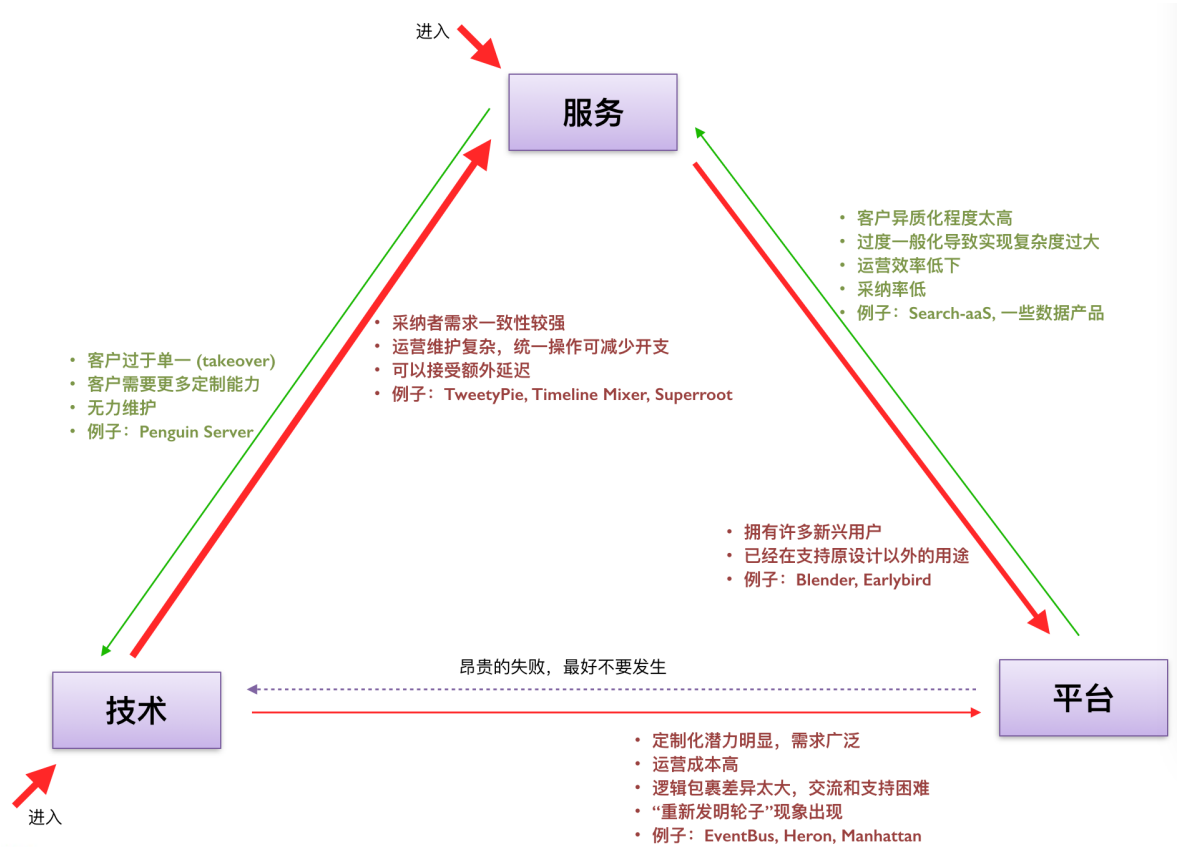
回顾发展历史，服务化是很重要的变化。最初，所有的东西都在一个大系统中，知识无法压缩，开发人员要关注很多东西；而在服务化之后，开发人员可以将精力放到具体的业务逻辑上，同时享受质量可以预测的服务。

下面再用一张图回顾一下Twitter这10年的技术演进史。

纵观拆分过程，可以总结出逻辑存在的不同形式。逻辑放到一起，就是单体结构。通过关注点分离，慢慢拆开，将逻辑拆到不同的模块中。通过服务化或平台化，可以将逻辑放到服务或平台中。具体如下图所示。



服务、平台和技术三者是可以相互转化的。红色的路径比较理想，而绿色则比较糟糕。



当把逻辑变成一个服务、技术或者平台时，这时候要慎重考虑，以对待顾客的方式对待使用该服务的同事。那么，如何当好一个服务生呢？

有几条要领：

- 用顾客需求驱动你的设计
  - 最简可行产品（MVP）
    - 不要实现既没有人需要也不能给你提供规划反馈的功能
  - 尽早实现效益
    - 部署之际已经能服务第一个客户
  - 考虑多顾客支持
    - 保证足够的灵活性
  - 尽早实现效益
    - 上马之际就能服务第一个客户
  - 注重客户体验
    - 好用的才会被采纳，被采纳的才能存活
- 用服务的语言来交流
  - 明确服务期望（服务级别协议：SLA）
  - 思考“收费”模式
  - 创造市场和社区

为方便他人使用，你可能需要提供设计文档、上手文档、示例代码、监测工具，并提供客户支持，还要协调未来功能规划，等等。

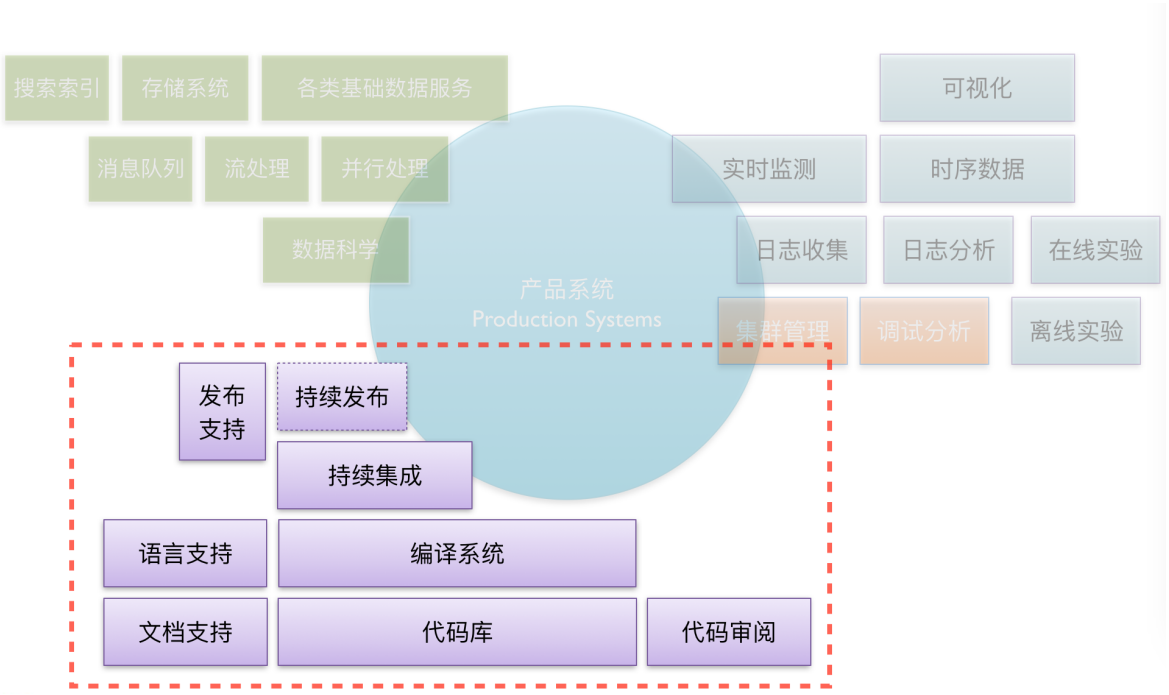
对于自己的服务、平台或技术，还要持之以恒地推广。包括推广你的服务和实践；扩大其用户群，增加采纳率；思考和类似服务共生和竞争的关系。

一些设计底线：

- 规范设计过程
  - 设计文档
  - 设计审核会议
- 确保符合当前最佳实践
- 有意识地提升工程质量底线
- 设立设计排查清单
- 充分讨论新技术引入的集成代价和支持代价
- 公开和协作
- 尽早引入利益方参与讨论
- 尽早思考产品化过程
- 设计导师：Design Shepherd

工程支持：磨刀不误砍柴工

在完整的工程生态中，有一些是幕后英雄：



它们和产品并没有太大的关系，主要是语言支持、编译构建等，但是这些东西是工程师使用最多的。提高这些东西的效率实际上对整个工程效率影响非常大。

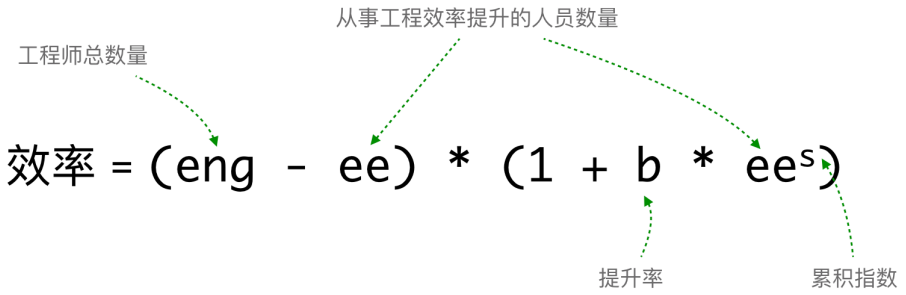
假设一个工程师一年工作2000小时（250天）：

效率提高比例	节省时间
1%	4.8分钟/天
2%	9.6分钟/天
5%	2.8小时/周
10%	2天/月

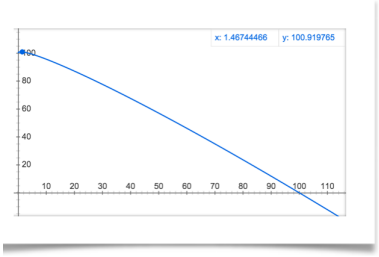
工程支持消耗的资源有限，但是可以让大部分工程师把精力用在刀刃上。

工程师的效率模型：

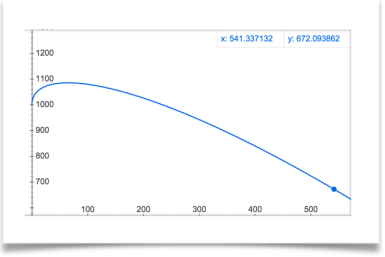
工程师的效率模型



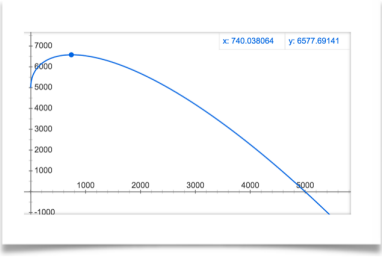
假设：b=2%，s=0.5



100个工程师：1人



1000个工程师：85人



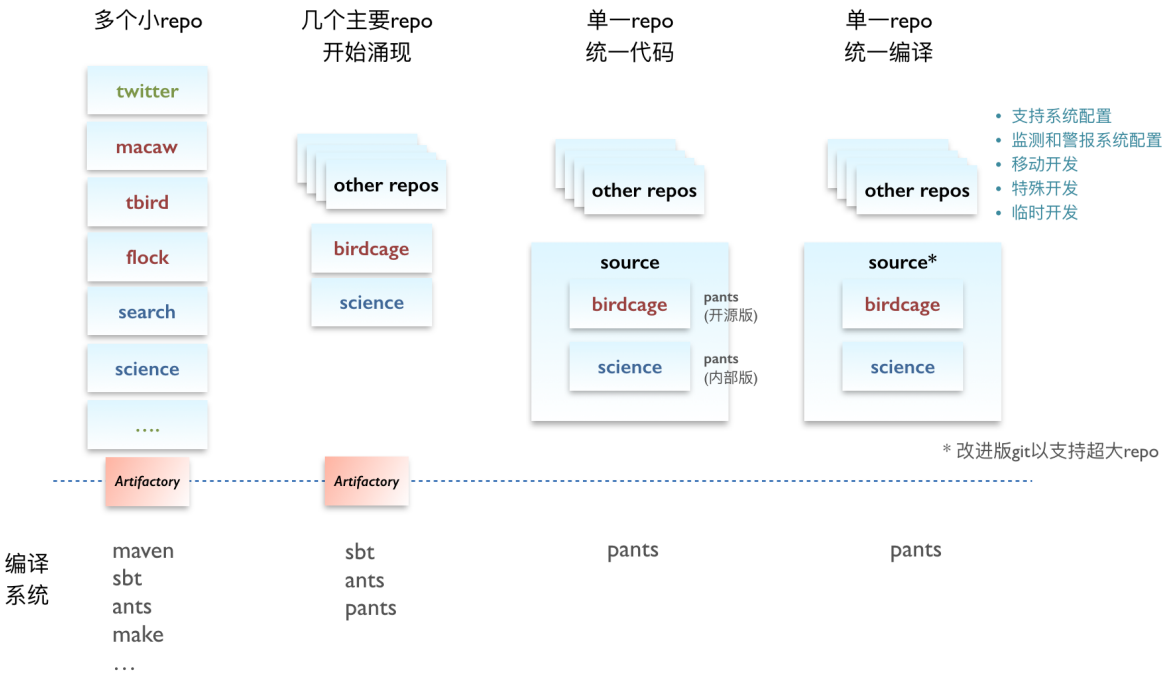
5000个工程师：740人

来源：http://www.gigamonkeys.com/flowers/

再来看一下Twitter的语言支持演变：

可以看到后面在收缩。语言过多还是会影响高效沟通。应该尽量减少。

Twitter的代码库和编译系统演进：



Twitter最开始有多个代码库。现在是单一repo，统一编译。其优势是，开发者能看到最新的、所有的东西。代码对所有人可见，可以直接协作。不过代码量非常庞大的情况下，这么做成本也非常高，像Twitter就自己修改了git之类的工具。可以说这是一个哲学选择。

其他工具：

- ReviewBoard：代码审核工具
  - 经过扩展以匹配公司代码审阅流程
- JIRA：任务规划和追踪
  - 各团队自行选择任务产生、分配和规划方式
- Confluence：公司内部Wiki
  - 维护团队文档、内部资料，指南等
- HipChat：聊天室
- DocBird：自开发和代码库集成的技术文档系统
- Google Docs
  - 协同编辑和审阅文档，共享文档、表格、幻灯片
- Google Calendar
  - 日历安排和协调

开源：

Twitter有很多项目都在github.com/twitter上开源了。