ThoughtWorks®

TECHNOLOGY RADAR

Our thoughts on the technology and trends that are shaping the future

JANUARY 2015

thoughtworks.com/radar

最新动态

本版精彩集锦

DevOps 领域的爆炸式增长

本版的技术雷达中，我们花了大量篇幅来评估 DevOps 领域中浩如烟海的各项技术，并且这些技术仍在以爆炸式的速度增长和创新。容器化、云产品以及它们的各种排列组合，使得在这个领域中的创新简直该用疯狂二字来形容。微服务架构风格的普及和流行，进一步扩大了架构技术与 DevOps 之间的交集，所以，我们预计，在这个领域中的疯狂创新仍将持续下去。

下一代数据平台引人注目

大数据并不是新名词（事实上，这段时间我们强烈反对炒作此概念），但是，我们也开始关注相关技术并研究如何在企业中使用它们。像数据湖和 Lambda 架构这类技术是放眼“企业数据平台”的新方式，不管你是否真有“大”数据需要处理，它们都可以派上用场。

开发人员关注于与安全相关的工具

每周都有“数据泄露或滥用”的新“故事”发生，而社会公众对于“安全的、尊重隐私的系统”的需求一直在增长。本版的技术雷达中精选了很多工具，比如 Blackbox、TOTP 双重因素验证以及 OpenID 连接等，它们能帮助开发人员建立安全的系统以及基础设施。

贡献者

ThoughtWorks 技术顾问委员会由以下人员组成：

Rebecca Parsons（首席技术官）
Martin Fowler（首席科学家）
Badri Janakiraman
Brain Leke
Claudia Melo
Erik Doernenburg
Evan Bottcher
徐昊
Cartwright
James Lewis
Jeff Norris
Jonny LeRoy
Mike Mason
Neal Ford
Rachel Laycock
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy

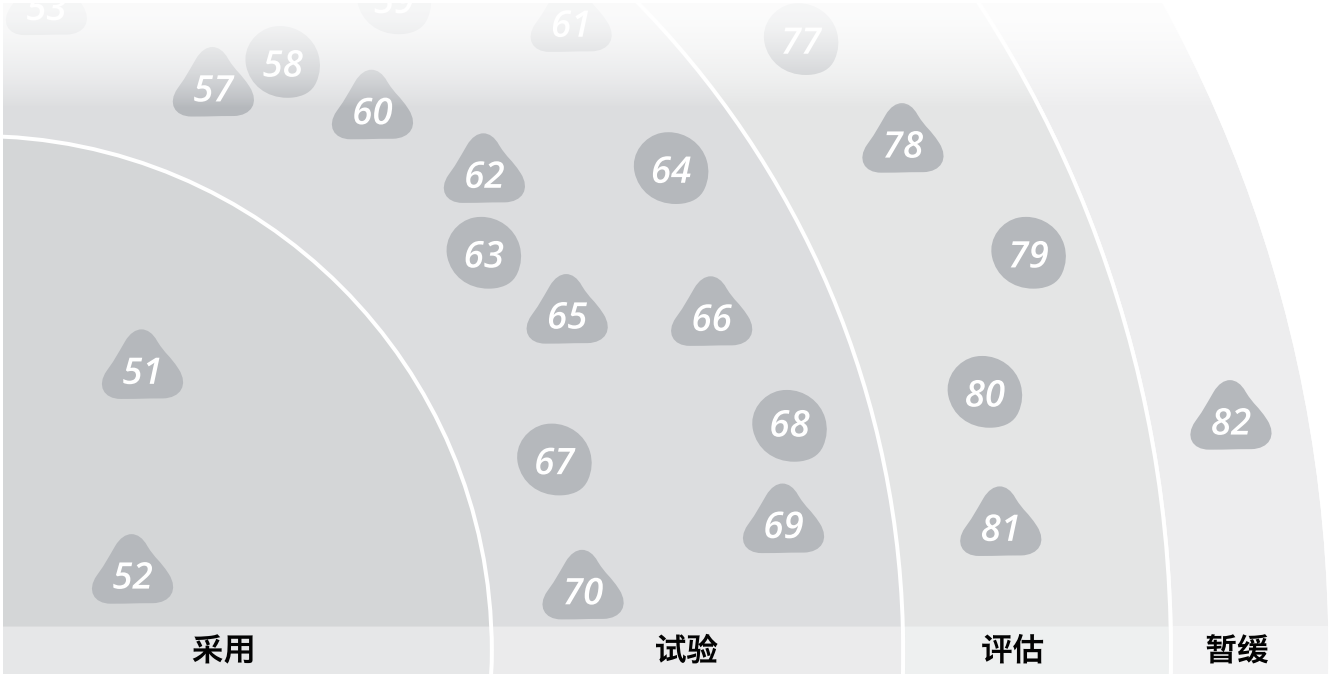
本期技术雷达中文译者：

边蕤
程迪
崔力强
陈熙
陈浩
常晏玮
杜杰
范文博
黄博文
韩锴
贺刚
贾朝阳
蒋帆
刘阳
刘彩红
刘峰
李青
李腾
李晓东
芮青华
师洁
孙镌宸
屠磊
佟达
伍斌
王海彬
王妮
王匡胤
汪志成
魏广程
肖战菲
姚琪琳
于舟
于晓强
赵国庆
周远征
张霄翀
张莹
张清波
朱本威
颛清山

关于技术雷达

ThoughtWorks 人酷爱技术。我们对技术进行构建、研究、测试、开源、描述，并始终致力于对其进行改进 - 以求造福大众。我们的使命是支持卓越软件并和掀起 IT 革命。我们创建并分享 ThoughtWorks 技术雷达就是为了支持这一使命。由 ThoughtWorks 中一群资深技术领导组成的 ThoughtWorks 技术顾问委员会创建了该雷达。他们定期开会讨论 ThoughtWorks 的全球技术战略以及和对行业有重大影响的技术趋势。

这个雷达以独特的形式记录技术顾问委员会的讨论结果，从首席信息官到开发人员， 雷达为各路利益相关方提供价值。这些内容只是简要的总结，我们建议您探究这些技术以了解更多细节。这个雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言及框架。如果雷达技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。这四个环是：



我们强烈主张业界采用这些技术。我们会在适当时候将其用于我们的项目。

值得追求。重要的是理解如何建立这种能力。企业应该在风险可控的项目中尝试此技术。

为了查明它将如何影响企业，值得作一番探究。

谨慎研究。

自上次雷达发表以来新出现或发生显著变化的技术以三角形表示，而没有变化的技术则以圆形表示。每个象限的详细图表显示各技术发生的移动。我们感兴趣的技术实在太多，远不是如此大小的文档能合理容纳的，因此我们略去了上期雷达中已包含的许多技术，为新技术腾出空间，略去某项技术并不表示我们不再关心它。

要了解关于雷达的更多背景，请参见 www.thoughtworks.com/radar/faq

THE RADAR

技术

采用
1. Focus on mean time to recovery
2. 前向保密
3. Structured logging

试验
4. Canary builds
5. Datensparsamkeit
6. Front end instrumentation
7. Hipster batch
8. Humane registry
9. Inverse Conway Maneuver
10. Living CSS Style Guides
11. Local storage sync
12. NoPSD
13. Partition infrastructure along team bounds
14. 没有 PUT 的 REST
15. Static site generators
16. 定制服务模板

评估
17. Append-only data store
18. Blockchain beyond bitcoin
19. Enterprise Data Lake
20. 机器映像管道
21. Pace 分层应用程序战略 步进分层应用程序战略

暂缓
22. 云提升加转移
23. Long lived branches with Gitflow
24. Microservice envy
25. Programming in your CI/CD tool
26. SAFe™
27. Security sandwich
28. Separate DevOps team
29. 测试作为独立组织
30. Velocity 当成生产率

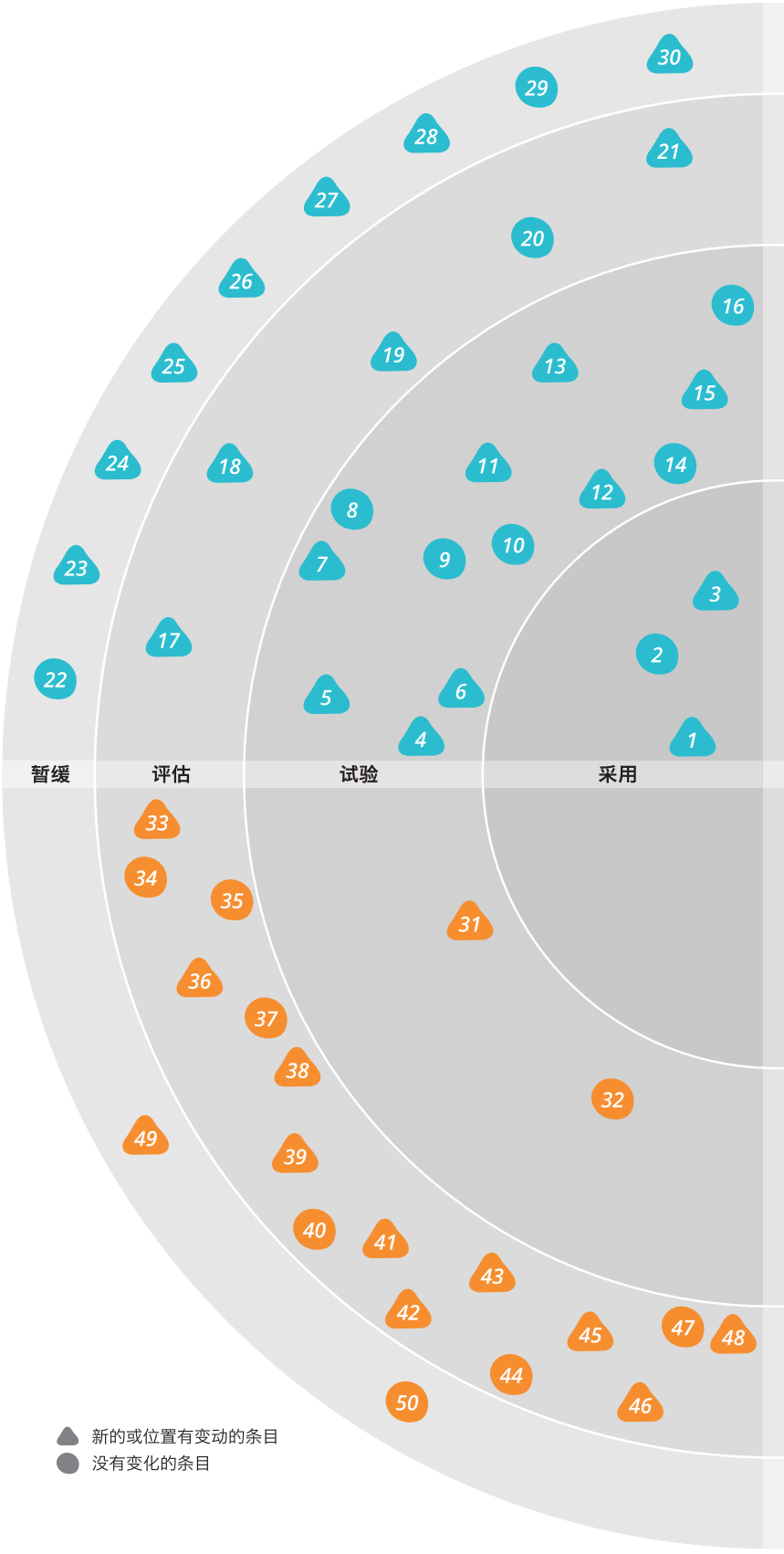
平台

采用

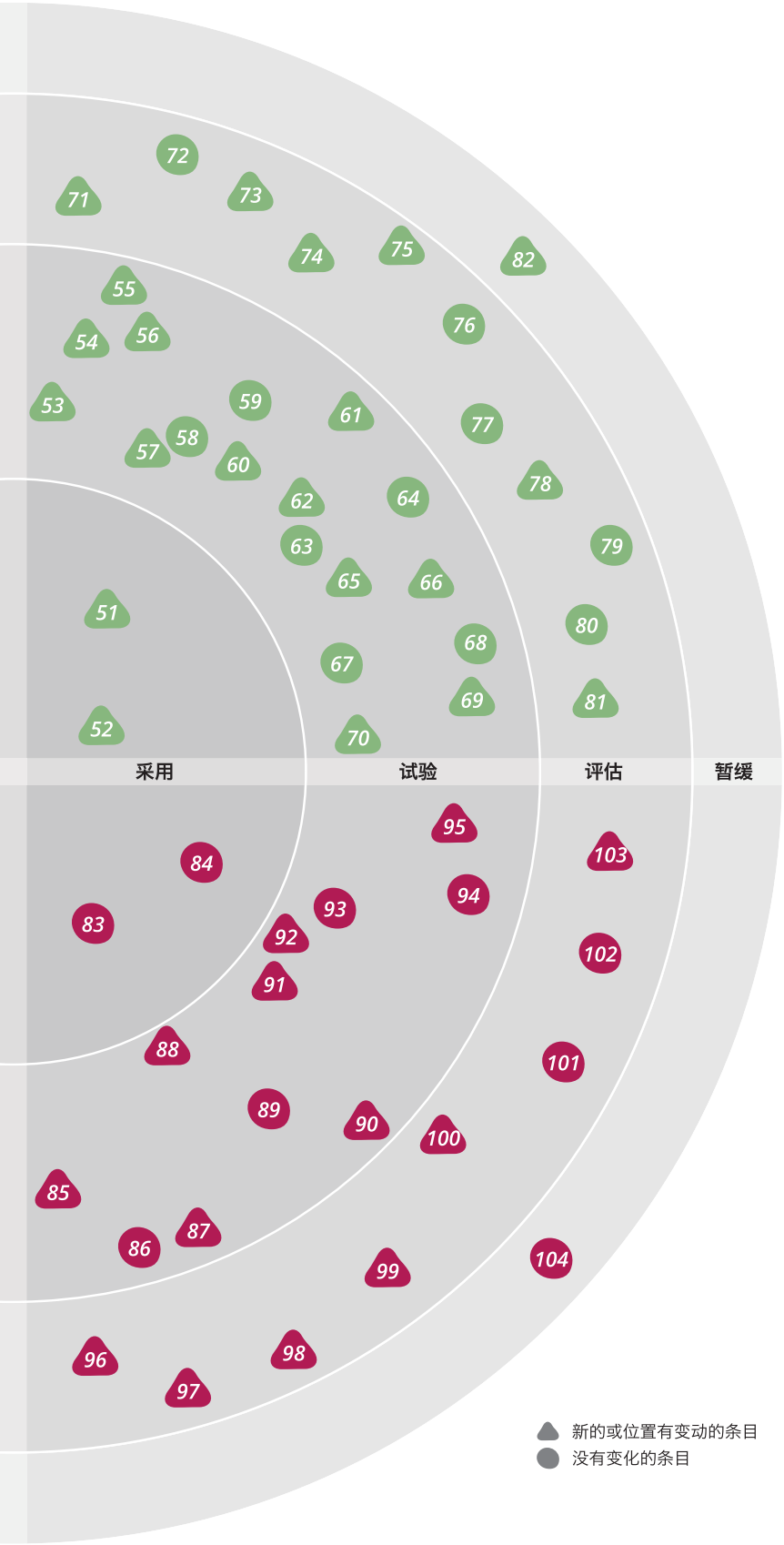
试验
31. DigitalOcean
32. iBeacon

评估
33. Apache Mesos
34. ARM 服务器 SoC
35. CoAP
36. CoreOS
37. EventStore
38. Jackrabbit Oak
39. Linux security modules
40. Mapbox
41. MariaDB
42. Netflix OSS Full stack
43. OpenAM
44. OpenID Connect
45. SDN
46. Text it as a service / Rapidpro.io
47. TOTP 二元认证
48. U2F

暂缓
49. CMS 即平台
50. OSGi



THE RADAR



工具

采用
51. Flyway
52. Go CD

试验
53. Appium
54. Boot2docker
55. Composer
56. Cursive
57. Docker
58. Foreman
59. GenyMotion
60. Gitlab
61. Grunt.js
62. IndexedDB
63. Packer
64. Pact 和 Pacto
65. Papertrail
66. Postman
67. SnapCI
68. Snowplow Analytics 和 Piwik
69. Swagger
70. Xamarin

评估
71. Blackbox
72. Consul
73. Dc.js
74. Gorilla REPL
75. Gulp
76. leaflet.js
77. Mountebank
78. Packetbeat
79. Roslyn
80. Spark
81. Terraform

暂缓
82. Citrix for development

语言和框架

采用
83. Go 语言
84. Java 8

试验
85. AngularJS
86. Core Async
87. Dashing
88. Django Rest
89. HAL
90. Ionic Framework
91. Nashorn
92. Om
93. Q 和 Bluebird
94. R 作为计算平台
95. Retrofit

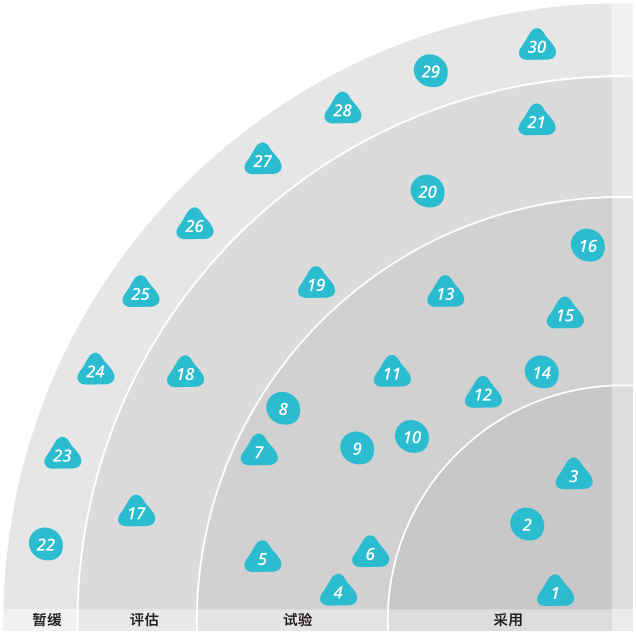
评估
96. Flight.js
97. Haskell Hadoop library
98. Lotus
99. React.js
100. Reagent
101. Rust
102. Spring Boot
103. Swift

暂缓
104. JSF

技术

许多项目都存在外部代码依赖，这些依赖中很大一部分是由开源项目提供的。为了确保构建过程可被重现，我们总是与固定版本的外部依赖进行集成。但这就意味着我们与这些类库的新版本集成并不及时，这将导致后面大量的合并工作。我们见到的避免这个问题的方式之一是夜间的 **Canary Build**，它会尝试使用所有外部依赖的最新版本进行构建。如果成功，就代表我们可以将外部依赖修改为相应的版本。

数据紧缩 (http://martinfowler.com/bliki/Datensparsamkeit.html) 来自于德国隐私法规，它表述了一种观点：仅保存在业务上或者合法性上绝对必要的个人信息。客户隐私依然是一个热门话题。像 Uber 这样的企业正在收集高度个人化的客户数据，然而他们的安全措施却非常糟糕 (http://washingtonpost.com/blogs/the-switch/wp/2014/12/01/is-ubers-rider-database-a-sitting-duck-for-hackers)。这是一场注定要发生的灾难。即便是在尚无法律规定的地区，我们也应该尽量遵循数据紧缩的原则，或者使用“去身份识别技术” (http://en.wikipedia.org/wiki/De-identification) 来减少所存储的信息——如果你根本就不保存这些信息，你也就用不着担心有谁会盗取它。



作为一种集成方式，通过 HTTP 使用 ATOM 风格的事件摘要 (feed) 最近受到了很多关注。大多数情况下我们并不需要维持一个实时更新的服务，使用旧式计划批处理进程来创建和发布摘要就够了。在和云技术结合后，比如 Amazon 的 S3 文件存储和超媒体链接，可以创建一个高度可用、简单而又可测试的解决方案。我们团队已经开始把这个新旧交替的方法叫做‘**Hipster batch**’。

当我们实现单页应用 (single-page applications) 时，“离线使用”这个问题迟早都会浮出水面。鉴于很难正确的将离线模式加入一个现有的应用中，“离线优先”的思想已经成为了开发单页应用的一种趋势。本地存储同步 **local storage sync** 就是一种我们曾经成功运用过的重要实现技术。使用这种技术，面向用户的代码将不再发送请求到后端系统，而是仅仅从本地存储 (local storage) 中获取数据。并由一个后台服务对本地存储和后端系统进行同步，通常，会调用某种形式的 REST API。

NoPSD (http://thoughtworks.github.io/p2/issue02/continuous-design/) 运动，旨在将设计活动整合进“迭代 - 反馈”循环中，以构建出优秀的软件。取这个名字，不是在讽刺 Adobe 的软件，而是要去除“PSD文件”这个“终极权威设计图”。团队不应在项目一开始就制定一个完美的像素级设计规范，而是要开始拥抱持续设计 (Continuous Design)：把设计师加入到交付团队中，使用低保真技术来做原型设计，并使用目标产品实际用到的 UI 技术 (通常是 HTML 和 CSS) 来共同细化设计。

采用

1. Focus on mean time to recovery
2. 前向保密
3. Structured logging

试验

4. Canary builds
5. Datensparsamkeit
6. Front end instrumentation
7. Hipster batch
8. Humane registry
9. Inverse Conway Maneuver
10. Living CSS Style Guides
11. Local storage sync
12. NoPSD
13. Partition infrastructure along team bounds
14. 没有 PUT 的 REST
15. Static site generators
16. 定制服务模板

评估

17. Append-only data store
18. Blockchain beyond bitcoin
19. Enterprise Data Lake
20. 机器映像管道
21. Pace 分层应用程序战略
    步进分层应用程序战略

暂缓

22. 云提升加转移
23. Long lived branches with Gitflow
24. Microservice envy
25. Programming in your CI/CD tool
26. SAFe™
27. Security sandwich
28. Separate DevOps team
29. 测试作为独立组织
30. Velocity 当成生产率

技术 接上页

这种方法提高了对真实用户反馈的响应速度，支持横跨多种设备及其不同形态对设计进行测试，积极拥抱数码产品和产品创作过程的灵动本性。

我们的很多客户与负责构建、部署、支持他们的应用和服务的交付团队合作，在组织中实现了DevOps (技术运维)。然而不幸的是，在交付团队实施 DevOps 的道路上，经常会受困于生产环境中超级用户权限的归属问题。在大多数的组织中，生产环境是被共享的，因此提供宽泛的权限有很大的风险。一个行之有效的措施是**将基础设施按照团队边界进行分隔**，于是每个团队通过安全又相互隔离的访问进行工作，而不会影响到其他系统。如果是使用云环境，只需要对齐账户结构与团队边界，这种方式更容易实现。

Middleman 和 Jekyll 这样的**静态网站生成器**已经广泛的用于简单网站或博客的创建，同时我们也越来越多的看到它们用作更复杂技术栈的一部分。随着越来越多的团队开始尝试使用静态的预生成内容，认为“ HTTP 回复的内容只能根据请求来动态创建”的默认假设正在改变。

因为有像 Clojure 这样的函数式编程语言的默认支持，不可变数据结构 (Immutable data structures) 开始变得越来越流行。“不可变性”使得代码更容易被写、读和理解。使用“只追加”式的数据存储 **append-only data store** 同样带给数据库层面一些这方面的优点，同时也使得审计和历史查询更加简单。实现方式有很多种，从专门使用 Datomic (www.datomic.com) 这样的“只追加”数据库，到简单以“只追加，不更新”的方式来使用传统数据库。

尽管比特币和其他各种密码货币 (cryptocurrencies) 现在备受关注，但让我们同样激动的是，使用 **Blockchain** 来代替比特币和财务交易的是非常可能的。Blockchain 是一种不依赖于中央服务就能验证共享总账的机制。我们已经看到 Blockchain (不管是底层技术或是公开的比特币 Blockchina) 被用在身份验证、所有权、记录、投票、云存储甚至管理智能设备网络等各种系统的核心部分。如果你正在构建的系统需要在去中心化的网络中建立信任，那么 Blockchain 是一项值得尝试的技术。

一个**企业级数据湖 Data Lake** 是对大量原始格式的数据进行的不可变数据存储，它不仅作为其它处理流的资源，还可以为大量使用高效处理引擎的内部技术消费者 (Consumer) —— 例如 HDFS 或基于 Hadoop 的 HBase，Spark 或 Storm 处理框架 —— 提供直接访问通道。与典型系统相比，这些系统会将原始格式的数据收集到严格受限的空间中，而且只会把严格受控的 ETL 过程生成的结果提供给这些消费者使用。

拥抱“数据湖”的理念就是要去除因缺少 ETL 开发人员或是堆积了太多前台数据模型设计而造成的瓶颈。它旨在帮助开发人员以敏捷的模式，基于他们需要数据的时机和方式 (在合理的限度内)，来创建他们自己的数据处理管道。这也与我们评价很高的另一模型 —— 技术运维 (DevOps) 模型有很多相同点。

Gitflow 是基于 Git 的一个严格的分支发布模式。虽然不是一个糟糕透顶的模式，但我们经常看到它被滥用。如果特性 (feature) 分支和开发 (develop) 分支是短暂并且频繁合并的，那么你确实是在发挥 git 的威力，它让团队合作变得更容易。然而，我们经常看到的一个问题是，这些“**长寿分支**”导致可怕的合并冲突，很多人重新开始使用 git 来避开这个问题。合并就是合并！无论你使用代码控制工具还是使用模式都不会改变这一点。如果你等一天或两天以上再合并，你就很有可能会遇到一个大的合并冲突。如果你在一个更大的团队中，这将成为一个真正的“问题”。如果你有更多的人在等待合并，这将进一步成为一个严重的瓶颈。成功引入像 Gitflow 这样的模式需要团队遵守“频繁合并、成功合并”的纪律。所以，当然可以使用这个模式，然而必要条件是得严格遵守“防止出现长寿分支”的纪律。

我们依然深信，在提高团队自主权和应对更加快速频繁的变更方面，微服务能给组织提供显著的优势。然而分布式系统所带来的额外的复杂性，需要更多的成熟度和投入。要做好微服务，就必须了解微服务给开发、测试和运维带来的变化。我们担心的是，一些团队在还没有弄清楚这些变化之前，就仓促地采用了微服务的方式。我们的建议非常简单，**不要艳羡微服务**，在一股脑地开发更多微服务之前，从一个或两个服务开始，让你的团队有时间来调整并探索合理的微服务粒度。

我们仍然看到，一些团队会直接把复杂的多行命令插入到 CI 或 CD 工具的配置中。通常，这些内嵌的命令也包含了一些只在构建环境起作用的步骤，包括 CI 相关的环境变量，一些只会在 CI 环境创建、修改文件和模板的步骤，等等。这让构建环境变成了一头“怪兽”——其结果在开发人员的本地机器上是无法复现的。

这个问题之所以非常严重是因为，CI/CD 工具应该去暴露你代码中的问题，然而它自己却成为一个“复杂性怪兽”，行为难以调试，结果也难以复现。

我们可以将复杂的构建过程提取成一个简单的脚本，这个脚本能通过一行命令调用，这样就避免了在 **CI/CD 工具中编程**。它能够在任何一台开发者的机器上执行，这样也就消除了构建环境本身的独特性超。把敏捷实践扩大到企业级，是一项持续的挑战。**SAFe™** 在相关的众多方案中脱颖而出。虽然 SAFe™ 为一些需要关注的方面提供了一份很有用的检查单，然而因为囊括了像“发布火车(release train)”和“门控流程 (gated control processes)”这样已不再被敏捷社区认可的大型发布策略，此类方案往往容易被误用。企业倾向于寻找的具有一定程度通用性、可横跨多方努力的方案，看起来，这正是 SAFe™ 所提供的，它所具有的某种程度的可定制性使它更具价值。事实上，其他的精益方案，比如像“改进道场 (Improvement Katas)”这样，主张边试验、边持续改进的实践，为不断扩大敏捷实践的规模提供了一个更好的模型。

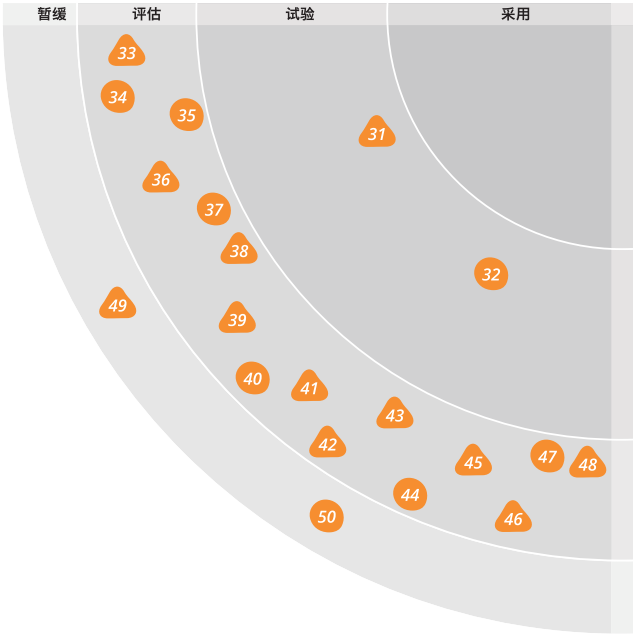
Scaled Agile Framework® 或 SAFe™ 都是 Scaled Agile 公司的商标。

对于安全，传统的方式依赖于前期的需求规格以及最后阶段的验证。这种“**安全三明治**”的方式很难应用于敏捷团队，因为大部分设计都贯穿于整个过程，而且也没有利用到持续交付所提供的自动化便利。公司或组织应着眼于如何在整个敏捷开发周期中注入安全实践。这包括：正确评估威胁模型以做前期设计；考虑何时将安全问题划分为独立的故事 (Story)、验收标准或全局的非功能需求；在构建流水线 (Build Pipeline) 中引入静态或动态的自动化安全测试；考虑如何将更深层的测试，如渗透测试，引入到持续交付的发布过程中。正如 DevOps 的出现使得过去相互博弈的团队能够重新合作一样，同样的事情也正发生在安全人员和开发人员身上。

平台

Mesos (http://mesos.apache.org/) 是一个平台，它通过抽象出底层的计算资源，使得建立大规模可扩展的分布式系统变得更加容易。它可以用来为 Docker 提供一个调度层，或者充当 AWS 之类云计算平台的一层抽象。Twitter 已经用它来助力扩展其基础设施，取得了很好的效果。基于 Mesos 之上的工具也开始出现，例如 Chronos (http://nerds.airbnb.com/introducing-chronos)，它是一种分布式的，能容错的 cron 的替代工具。

CoreOS 是一个被设计为运行大型的、可扩展的系统的 Linux 发行版。部署在一个 CoreOS 实例上的所有应用程序都运行在单独的 Docker 容器中，并且 CoreOS 还提供了一套工具来帮助管理它们，包括 etcd —— 它们自己的分布式配置存储。较新的服务，比方说 fleet，通过确保一定数量的服务实例始终保持运行来帮助集群管理。FastPatch 通过主动－被动根分区模式，从而允许原子的 CoreOS 升级，并在出现问题时帮助快速回滚。如果你已经熟知 Docker，这些新的发展都使得 CoreOS 非常值得进一步探究。



Jackrabbit Oak (http://jackrabbit.apache.org/oak/)，之前叫做 Jackrabbit 3，是一个可扩展和高性能的层次化内容存储系统，可用于内容管理系统的基础。其存储方式除了使用文件系统之外，也支持 MongoDB 和 RDMS，其更适合于大数据量的场景。虽然它使用 Java 实现，但是可以通过类似于 JCR 这样的标准从各种平台很方便地对其进行访问。

对于不得不管理生产环境的系统管理员来说，服务器固化 (server hardening) 是一种相当普通的旧技术，但是它在开发社区尚未普及。

然而，DevOps 文化的崛起导致像 SElinux、AppArmor 和 Grsecurity 这样的工具重获关注，这些工具致力于简化服务器固化 —— 至少在 Linux 生态圈。每个工具都有各自的优缺点，目前还很难去选出唯一一个最合适的，也就是说，我们强烈建议，所有的团队至少评估一下哪些 **Linux 的安全模块**是更适合他们自己的，并且把安全与服务器固化作为开发流程中的一部分。

自 Oracle 收购了 MySQL 之后，越来越多的非开源模块被绑定到 MySQL 的企业版中。这产生了对 MySQL 前景的担忧。**MariaDB** (https://mariadb.org/) 是一个由社区开发的且仅支持 GPL 软件许可的 MySQL 分支。它旨在保持开源的纯粹性，在完全兼容 MySQL 的同时，又具有与之匹敌的竞争力。其令人瞩目的用户群包括大型互联网组织 Google 和 Wikipedia，以及重要的Linux 发行商 RedHat 和 SUSE。

采用

试验
31. DigitalOcean
32. iBeacon

评估
33. Apache Mesos
34. ARM 服务器 SoC
35. CoAP
36. CoreOS
37. EventStore
38. Jackrabbit Oak
39. Linux security modules
40. Mapbox
41. MariaDB
42. Netflix OSS Full stack
43. OpenAM
44. OpenID Connect
45. SDN
46. Text it as a service / Rapidpro.io
47. TOTP 二元认证
48. U2F

暂缓
49. CMS 即平台
50. OSGii

平台 接上页

除非用户恰巧正进入全球性的分布式视频流媒体业务，否则我们并愿不推荐全盘采用 **Netflix 全栈式的开源软件框架**。即便如此，其技术栈还是充满了大量的有趣的想法，并以开源的形式加以实现。有一些工具，例如 Asgard，被高度耦合到一个几乎是现成的架构中，使得难以独立使用它们。而另一些工具，例如我们之前在技术雷达中提到过的 Ice 和 Hystrix，则是可以被单独使用的。我们认为即便团队没有选择使用上述全栈式的框架，他们也应该理解被封装在这些工具中的思想和方法。

当 Oracle 停止开发原来 Sun 公司的 OpenSSO (一个访问开源软件的管理平台) 之后，这个平台被 ForgeRock 接收，并被集成到他们的 pen Identity Suite 中。 这个套件现在被命名为**OpenAM** (http://www.forgerock.com/en-us/platform/access-management/), 它定位于一个具有可伸缩性的开源平台， 支持各种联合身份认证标准，包括 OpenID Connect 和 SAML 2.0。这些标准对于实现安全的微服务架构是非常必要的。

软件定义网络 (Software Defined Networking，**SDN**) 是一个很宽泛的话题，但正变得越来越重要。使用软件来配置网络设备的能力，正在使得应用部署终止的边界线变得模糊。它无所不包，从诸如 AWS Load Balancers 或 CoreOS 的 Flannel (https://github.com/coreos/flannel) 这样的虚拟网络设备，到支持像 OpenFlow (http://opennetworking.org/Openflow) 这样的标准的网络设备。云服务提供商之前一直在关注计算和存储，而我们则期待更多的 SDN 工具能为处理部署在本地和云端的系统同时带来更多效率提升 。

文本化即服务 (Text-it-as-a-service)/Rapidpro (http://rapidpro.io/) 为业务提供轻松地创建或修改复杂的短信服务应用的能力，同时无需开发人员的大力支持。相对于 USSD 会话来说，文字信息的成本更低。这提供了一个更加经济实惠的方式，来针对功能手机 (feature phones) 构建具有伸缩性的应用程序。这在我们的项目中已经看到了成功的案例。流程非常容易构建；行为也可以在诸如发送短消息、邮件，甚至是调用外部 API 等任意时点被触发 。

保持在线账户的安全，既极端重要，又是公认地困难。双因子认证 (two-factor authentication) 极大地提高了安全性，我们已经推荐过 TOTP 这样一个不错的解决方案。现在这个领域又有了新的成员，Universal 2nd Factor(**U2F**)。该解决方案基于公钥加密和价格低廉的 USB 硬件 Token。虽然最初由 Google 开发，它现已成为由 FIDO 联盟管理的一个标准。我们确实喜欢它所承诺的针对钓鱼和中间人 (man-in-the-middle) 攻击提供了更好的保护的承诺，但是也有一些担忧，因为它的标准目前引用了一种特定的椭圆曲线数字签名算法，而该算法被认为是有缺陷的。

工具

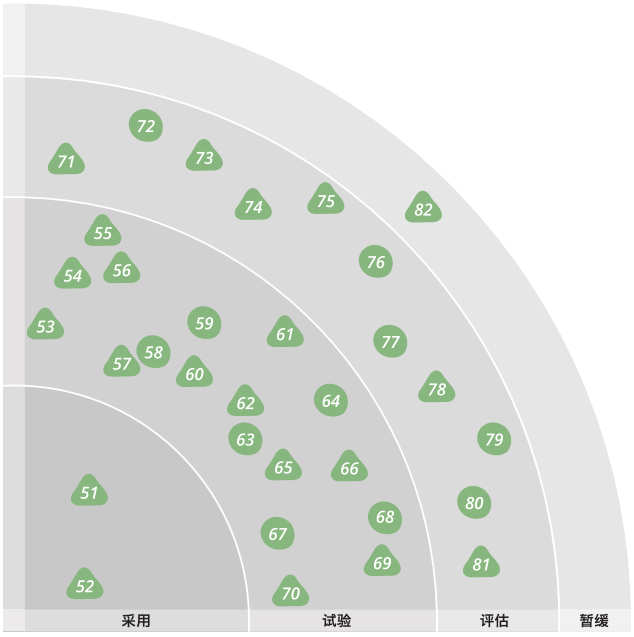
随着诸如持续交付这样的技术变得更加主流，自动化数据库迁移已成为许多软件开发团队的基本技能。虽然此领域工具众多，但我们继续推荐 **Flyway**，因为它在迁移时具有低摩擦性(low-friction)的特点。Flyway 拥有一个充满活力的开源社区，而且同时支持传统数据库和像亚马逊 Redshift 和谷歌 Cloud SQL 这样基于云的数据库。

想要快速和可靠地将高质量的软件持续交付到生产环境中，必须协调大量的自动化步骤。由 ThoughtWorks 打造的开源工具 **GO CD** (http://www.go.cd/) 正是解决此问题的利器。它以部署流水线 (deployment pipelines) 的概念为核心思想，在大量节点之上处理复杂的工作流，并实现了在不同环境之间将可信赖的产出物 (artifacts) 以一种透明和可追溯的方式进行传送 (promotion)。

尽管可以在持续集成工具之上打造部署流水线，但是我们的团队看到了专为此目的而构建的工具所带来的好处。

Boot2docker 是一个在其中运行着 Docker 的轻量级的 linux 发行版本，它以针对 OSX 和 Windows 的方式被打包成一个虚拟机。这是开始体验 Docker 的一种很棒的方式。对于使用微服务的团队来说，它也是一种在一台本地开发机上运行多个服务的有效方式，以达到开发和测试的目的。相比之下，使用多个 vagrant 虚拟机的开销就显得太大了。

尽管依赖管理的想法已经不再新颖，且已被认为是一种基础的开发实践，但它还是没有被 PHP 社区广泛采用。**Composer** (https://getcomposer.org/) 是 PHP 世界中一个依赖管理工具，它强烈地受到了其他技术栈的依赖管理工具 (如 Node 的 npm，Ruby的 Bundler) 的影响。



Cursive (https://cursiveclojure.com) 是一个 Clojure 的 IDE 工具，是 Intellij 的一个插件。虽然目前是预览版本，但是我们已经发现在较大的 Clojure 代码库上进行工作时，Cursive 非常有用。Cursive 在重命名和导航方面功能强大，并且展现出稳定和可靠的特点，这一点对于混合 JVM 语言的环境来说确实很棒。对于采用 Clojure 的那些团队，Cursive 降低了现有开发人员入门 Clojure 的门槛。

Gitlab 是一个需要在本地部署 (on-premise) 的 Git 代码库托管平台。像诸如 Github 和 BitBucket 这样的托管版本控制服务为开源社区开发者提供开发工作流工具那样，Gitlab 为那些开发私有软件的团队提供了同样令人熟悉和通用的工具。一方面 Gitlab 的社区版软件能够免费获得，另一方面其商用企业版能够提供技术支持，以及与 LDAP 服务器的深度集成。

采用
51. Flyway
52. Go CD

试验
53. Appium
54. Boot2docker
55. Composer
56. Cursive
57. Docker
58. Foreman
59. GenyMotion
60. Gitlab
61. Grunt.js
62. IndexedDB
63. Packer
64. Pact 和 Pacto
65. Papertrail
66. Postman
67. SnapCI
68. Snowplow Analytics 和 Piwik
69. Swagger
70. Xamarin

评估
71. Blackbox
72. Consul
73. Dc.js
74. Gorilla REPL
75. Gulp
76. leaflet.js
77. Mountebank
78. Packet beat
79. Roslyn
80. Spark
81. Terraform

暂缓
82. Citrix for development

工具 接上页

随着单页面应用和离线优先 (offline-first) 的开发方式变得更加可行和普遍，在 web 浏览器中持久化数据的需求与日剧增。本地存储 (Local Storage) 非常容易使用，而且得到了大多数浏览器的支持。而对于更加复杂的使用场景，**IndexedDB** 就能派上用场了。尽管有时候 IndexedDB 不失为一个好的解决方案，但是我们仍然建议只有在绝对必要的情况下再使用它，这是因为使用它会增加复杂性，而且其 API 也多少有些笨拙。我们在对 LocalForage (https://github.com/mozilla/localForage) 框架的使用中也获得了良好的体验，它在那些各式各样的浏览器端持久化解决方案之上提供了一个抽象层。

Postman (http://www.getpostman.com/features) 是一个 Chrome 浏览器扩展程序 (extension)，它在浏览器上充当了 REST 客户端的角色，允许用户创建 Request 并检查 Response。在进行 API 开发或实现调用已有 API 的客户端时，这个工具很有用处。它提供了一组扩展程序，可被当作一个功能全面的测试运行器来使用，但是我们还是要劝阻大家不要使用它所提倡的录制 (record) 和重放 (replay) 风格的测试方式。

Blackbox (https://github.com/StackExchange/blackbox) 是一个能直接在源代码库中加密特定文件的简单工具。这在需要存储密码或者私钥时特别有用。Blackbox 支持 Git、Mercurial 和 Subversion中的使用，并以 GPG 进行加密。每一个用户拥有自己的密钥，从而能够很容易地撤销对某个粒度级别的访问权限。

我们之前推荐过 D3.js，在本次雷达中，我们想将之前的推荐扩展到 **Dc.js** (http://dc-js.github.io/dc.js/) 。这是一个基于 D3 的图表绘制库，用于探索大型多维数据集。它能与 D3 一起，轻松地创建优美的具有交互能力的图形。所不同的是，为了一种更简单的、可用来创建通用图表类型的编程模型，它牺牲了创建几乎任何类型的数据视图的灵活性。

GorillaREPL (http://gorilla-repl.org/) 是一个用来创建经过精美渲染的文档的工具，该文档包含文本、可运行的 Clojure 代码和 plots。它在某些方面类似于 i Python notebooks，在数据分析或者代码教程方面应该会特别有用。除此之外，GorillaREPL 也非常有趣！它以一种创造性的方式来展示 Clojure 对不可变值进行简单抽象的能力。

随着分布式系统变得愈加复杂，一些用来帮助用户了解系统在生产环境下是如何运行的工具变得很有用。**Packetbeat** (http://packetbeat.com/) 就是这么一款开源工具，它使用代理来探查节点间的网络流量，帮助用户查看流量模式、出错率等其他一些有用的信息。它需要安装 Elasticsearch (http://elasticsearch.org/overview/elasticsearch) 和 Kibana (http://www.elasticsearch.org/overview/kibana) 之后才能工作。但如果这些工具已经是日志聚合的一部分了，那么它就可以轻松地嵌入其中，为用户提供更多的产品环境系统内部信息。利用 **Terraform**，用户可以以声明式定义的方式，来对云端的基础设施进行管理。由 Terraform 实例化的那些服务器配置信息，通常会留给诸如 Puppet、Chef 或 Ansible 这样的工具来处理。我们之所以喜欢 Terraform，一方面是因为其文档语法的可读性相当好，另一方面是因为它支持多家云服务提供商，但却没有企图在这些提供商之上提供一个矫揉造作的抽象构件。但在现阶段，Terraform 还是一个新的工具，尚未实现所有的东西。我们也发现它的状态管理还有些脆弱，经常需要笨拙的手动操作来解决一些问题。

出于安全及遵循规范的原因，那些离岸 (offshore) 团队时常会被要求使用 **Citrix**，来连接到一个在岸 (onshore) 虚拟桌面进行**开发**。虽然在某些情形下 Citrix 是一个良好的工具，但是它所提供的远程开发体验极其糟糕，以至于经常会让一个离岸团队陷入瘫痪。市面上还是存在许多更好的技术解决方案，比如 NoMachine 远程桌面，或者 Cloud9 IDE。它们所带来的体验令人感觉更加可行。当然更好的解决方案应该是去化解深层次的有关安全性及合乎规范的顾虑。既然可以信任远程团队在源代码上进行开发，且能将代码提交到代码库中，那么也应该要试着信任他们，使他们的本地机器上也拥有源代码。这样他们将会更加具有生产力！

语言和框架

在团队所在区域可视化一些醒目信息的重要性已被多次提及，我们非常肯定让每一个成员看到并理解产品状态或团队正在做什么的价值。**Dashing** (http://dashing.io/) 是一个基于 ruby 的信息展示系统，已被使用了很多年，我们常常用它来创建简单明了的并且适合大显示器显示的可视化信息。它易于扩展，并支持从各类资源导入信息（如构建系统，票据，故事跟踪工具，生产环境监视系统等）。

我们已在多个项目中使用过 **Django Rest 框架** (http://django-rest-framework.org/)，它是一个灵活可定制的框架，用来做 web API 的构建很容易。django 可用来构建 Rest 风格的 python API，并将其暴露给前端。Django Rest 提供可浏览的web API，能够将数据从 API 作出响应到客户端接收到的过程可视化。此外，它还提供一些非常好的验证策略，也支持自定义策略的实现。

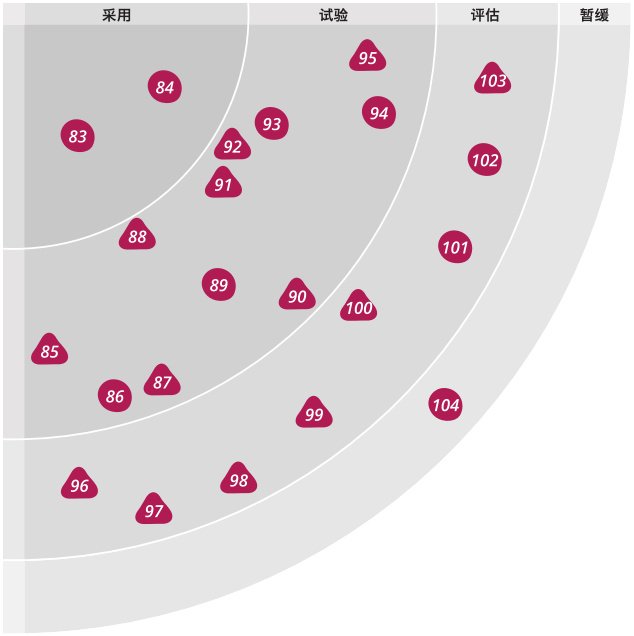
Ionic Framework (http://ionicframework.com/) 是一个开源前端框架，它提供了一套面向移动设备的 HTML、CSS 和 JavaScript 控件库，以及一系列用来创建高互动性应用的工具。Ionic Framework 使用 SASS 构建，并针对 AngularJS 进行了优化。由于它易于安装、方便测试，我们已经见到它成功应用于在多个项目当中。如果你是“性能控”，想要寻找一个无缝集成的前端框架，那么我们推荐你了解一下这个框架。

Nashorn 是一个面向 Java 8 发布的全新 JavaScript 引擎。当我们需要在浏览器和服务器执行完全相同的代码时——例如做校验或者数据迁移 —— Nashorn 就是 Java 世界里的可选方案之一，虽然 Java 应用中很少会遇到这种边缘场景。尽管可以通过 Node 或 Avatar 来使 Nashorn 承载整个应用，但是我们还不确定这是不是一种好的实现。

Retrofit (http://square.github.io/retrofit/) 通过将一个 REST API 转化为 Java 接口，为在 Android 项目上构建 HTTP 客户端提供了一种可靠的方法。它与 OkHttp 集成，允许开发人员为请求提供自定义的异常处理，使用 GSON 自动解析 JSON，同时还拥有一个非常完善的支持社区。

在众多 JavaScript 框架中，我们强烈推荐 **Flight.js** (http://flightjs.github.io/) Flight 是个极其轻量级的框架，只需要很少的工作量就能在 DOM 节点中添加行为。它有事件驱动和基于组件的特点，可使促使开发人员编写出低耦合的代码。这也使得测试单个组件变得十分容易。但要格外注意组件之间的交互。Flight.js 对测试的支持还不是很多，且容易卷入事件漩涡。继承与组合，我们喜欢组合，与之类似，我们非常喜欢 Flight.js 对行为使用了函数的 mixin。

虽然在 ThoughtWorks 的语言爱好者中有很多 **Haskell** 的粉丝，但是直到最近我们才有项目在使用它。目前，有一些向Haskell 语法中加入 **Hadoop** 的 map/reduce jobs 的开源项目引起了一些程序员和/或数据科学家的兴趣。



采用
83. Go 语言
84. Java 8

试验
85. AngularJS
86. Core Async
87. Dashing
88. Django Rest
89. HAL
90. Ionic Framework
91. Nashorn
92. Om
93. Q 和 Bluebird
94. R 作为计算平台
95. Retrofit

评估
96. Flight.js
97. Haskell Hadoop library
98. Lotus
99. React.js
100. Reagent
101. Rust
102. Spring Boot
103. Swift

暂缓
104. JSF

语言和框架 接上页

我们并不知道是谁为 **Lotus** (http://lotusrb.org/) 命名，只能假设他们太年轻以至于没有使用过某些办公协作产品。Lotus 是一个新的MVC 框架，基于 Ruby 实现，其模块化部署可以让你只使用需要的部分。它是整个 Ruby-on-Rails 框架 (今年升级到10) 的时髦替代者。Lotus 有潜质把 Ruby MVC 全栈开发简化得像 1-2-3一样简单。

当下前端 JavaScript 框架层出不穷，也是因为如此，不时会有一个新想法蹦出来，让人反思。**React.js** 是一个 UI/View 框架，提供以反应式数据流 (Reactive Data Flow) 生成 HTML 标签的 JavaScript 函数。尽管我们对于将代码和标签混合使用的方式保持谨慎，但 React.js 确实让UI组件具有更好的封装，也更容易组装。React.js 正逐渐引起很多开发人员的关注，针对它的工具和示例也越来越多，这将会促进 React.js 的发展。

作为 Om 的替代品，**Reagent** (http://holmsand.github.io/reagent/) 是一个将 React.js 包装进 ClojureScript 的轻量级简约技术。与Om 提供全面的 Clojure-idiomatic 前端框架不同，Reagent 利用Clojure的表现力，更专注于简单组件，以及可读性更高的DSL 编写 HTML。当使用 Clojure 数据渲染 HTML时，Reagent保持了 React.js 的性能和可理解性，且不需要在代码中引入额外标记。

与已出现多年的 Objective-C 相比，苹果新的开发语言 **Swift** (http://apple.com/swift) 有很多的改进，强调了函数式编程思想，语法也更加现代化。如果你工作在 Apple 平台上，那么就非常值得考虑升级到该语言。
