时刻关注企业软件开发领域的变化与创新

架构师

6月 ARCHITECT

特别专题

探索中的探索式测试

探索式测试的秘密

探索式测试中的几种误区

测试工程师的学习之旅

虚拟座谈会：TDD有多美？

Java深度历险（八）-- Java I/O

对象已死？

正则表达式（五）：
浅谈两种匹配操作

Guardian.co.uk从Java切换到Scala

上线：准备和部署软件包时
开发和运维的角色

InfoQueue
Enterprise Software Development Community

每月8号出版

篇首语 | Editor's letter

HTML5 时代来临？

这一段时间社区里好不热闹，而且如果是经常上微博的朋友，一定会发现今天这儿有个 HTML5 活动，明天那儿有个 HTML5 大会，来头好像都不小，背后都有大公司的支持，比如微软、盛大什么的。我上周就参与了一次创新工场主办的"HTML5 在中国"会议，现场有近 800 人参加，好不壮观。而且 IE、Chrome、Firefox 和 Opera 等四大浏览器厂商悉数到场，分享自己对 HTML5 的支持；很多在移动领域创业的游戏公司，包括全球最大的休闲游戏网络之一 Spil Games，还有新浪游戏等，当然还有创新工场的当家人李开复老师。

那么喧哗，那么热闹，难道 HTML5 的时代真的已经来临？在下这个结论之前，我们最好还是深呼一口气，然后心定神闲地品味这个问题。正如现场和一位资深的 HTML5 领域的朋友交流，他意味深长地说，HTML5 现在太热闹了，好像有点危险。如果我们能够看到现在对 HTML5 比较看好的公司或者个人，多是和游戏，而且是移动设备上的游戏公司相关的，那么多少你可能会悟出一点道理。现在地球人都知道，互联网领域热钱涌动，而尤以移动领域最为突出。而移动领域原本是一个标准不统一的地方，如果能有一种技术，可以跨不同的移动设备，就像当年 Java 说"一次编译，处处运行"一样，那么它肯定备受欢迎，而 HTML5 就这样被幸运地选中了。

一种技术能否持久，我们需要看得是它能真正解决多少问题，虽然现在新兴语言层出不穷，包括 Java 的变种语言，但 Java 还是拥有大量忠实的粉丝，因为不论是在传统企业开发还是互联网领域，Java 的稳定性和对大数据量架构的支持都无人能敌。我们当然希望 HTML5 也是这样，可以让一种应用在所有的浏览器中都保持一样的体验，不论是在 PC 端还是在移动设备上，这可以让前端技术开发人员少了多少无端的烦恼啊。

可惜目前来看，这还只是个美好的愿景。最后，我们来看一下 HTML5 小组成员，也是这次"HTML5 在中国"的讲师秀野堂主在演讲中总结的几个浏览器的不足，让自己激动的大脑清醒清醒：

- 非 iPhone 手机浏览器共同的问题：不支持 webSocket
- localStorage 吃不消密集读写，会崩溃
- 浏览器在对 HTML5 标准的支持上有走回头路现象
- 如果计划使用 HTML5，要密切注意浏览器厂商和标准的变化
- 如果现在上马 HTML5，你会在 API 应用上面临被浏览器绞杀的局面

本期主编：霍泰稳

目　录

[篇首语]

HTML5 时代来临？ 1

[人物专访]

淘宝网商品库优化实践访谈 4

[热点新闻]

MYSQL 与 NOSQL--SQL 与 NOSQL 的融合 8

JQUERY 1.6 发布：性能提升，同时包含破坏性的变更 13

阅读者（十二）：番茄工作法图解 15

腾讯力推开放平台 Q+ 21

微软 85 亿美元收购 SKYPE 引发热议 25

D 语言和 VISUAL D 简介 31

GOOGLE STORAGE 全面开放 33

[特别专题]

探索式测试的秘密 35

探索式测试中的几种误区 42

测试工程师的学习之旅 47

虚拟座谈会：TDD 有多美？ 52

[推荐文章]

JAVA 深度历险（八）--JAVA I/O 61

对象已死？ 68

正则表达式（五）：浅谈两种匹配操作 73

GUARDIAN.CO.UK从JAVA切换到SCALA..... 82

上线：准备和部署软件包时开发和运维的角色 89

[新品推荐]

COLM语言简介 97

ENTITY FRAMEWORK POWER TOOLS简介 97

SIWPAS 1.0——针对EE6 WEB PROFILE基于TOMCAT的服务器 97

JETBRAINS发布INTELLIJ IDEA 10.5，全面支持JAVA 7..... 98

下一个版本的WINDOWS PHONE MANGO包含了大量新的特性 98

AKKA 1.1 发布，对FUTURES和性能做出改进，降低了依赖性 98

SPEC EXPLORER 2010 3.5 版改进了更改检测和批处理 99

ESSENTIAL: 新语言工作台 99

MICROSOFT发布IOS版WINDOWS AZURE工具包 99

[推荐编辑]

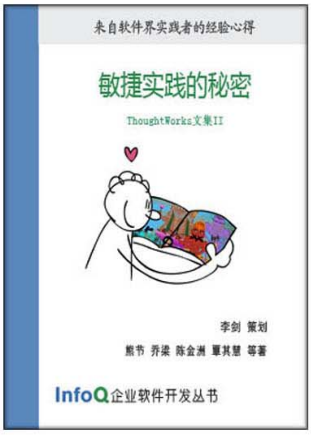
翻译团队编辑 金毅 100

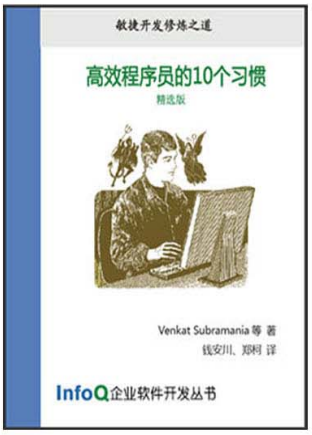
[封面植物]

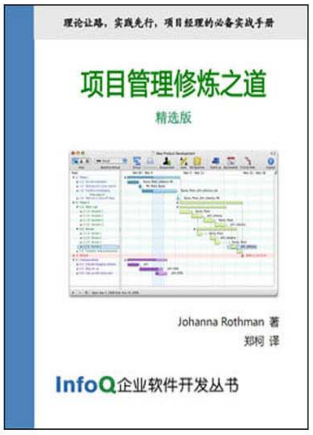
紫斑牡丹 101

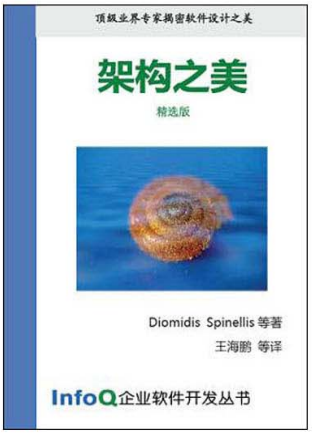
InfoQ企业软件开发丛书

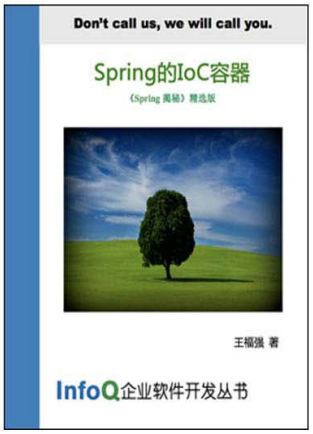
欢迎免费下载











商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

人物专访 | Interview

淘宝网商品库优化实践访谈

优化听起来好像没什么内容，但实际上我们做了很多的工作。首先要有明确的目标，我们的目标是什么，是为了提高性能还是提高安全性，还是提高其他的什么指标，这个目标要清晰和完整。第二点是测量，你不能说我拍脑袋看到某某某，他怎么优化的，然后我也上去这样照着优化，这不是我们做事的风格。我们要测量现在的瓶颈在哪里？我们的系统遇到了什么问题?

余锋（淘宝花名：褚霸）淘宝核心系统数据库组技术专家，先后在网易，迅雷，网龙等国内知名的 IT 企业从事研究和开发工作。有超过 15 年的网络、内核以及底 层软件开发经验，专注于高性能分布式服务器的研究和实现，擅长构建大规模集群服务器，对 NoSQL 系统和分布式文件存储有深入的研究和实践，同时致力于 Erlang 分布式系统的研究和推广。目前负责开发和维护淘宝 MySQL 数据库分支，解决淘宝数据库高速发展遇到的技术和性能瓶颈。

InfoQ：余锋先生您好。欢迎您参加 QCon0：04 并给我们做精彩的演讲。也谢谢您接受我们的采访。您能向观众朋友大概介绍一下您自己，包括您现在负责的具体工作是什么吗？

余锋：大家好，我隶属于淘宝网核心系统数据库组。我们的团队在淘宝主要负责 MySQL 分支的维护。大家知道淘宝去年开始在做去 Oracle 化的工作，由于涉及到历史数据保留，替代方案当时选择的是 MySQL。由于淘宝很多的产品都用到了数据库，这也促使 MySQL 即将会慢慢被大规模使用。Oracle 是一个商业的产品，所以它有很多的特性是非常适合大规模数据使用的。同时 MySQL 作为一个开源的产品，也具有很多商业数据库所没有的特性，于是我们就要去填补这个空白。我们的工作就是在开源的 MySQL 基础上做一些改进和优化，特别是性能方面的优化。MySQL 最初的设计不是用来存储大规模数据的，但淘宝的数据量非常惊人，所以在 I/O 方面，尤其是 CPU I/O 层面会有很大瓶颈，因此我们的主要目标也是解决 IO 方面的瓶颈问题

InfoQ： 在谈到具体的 MySQL 优化策略之前，我想大家都比较好奇，介于淘宝现在的数据容量，每天的吞吐量是一个什么样的情况？

余锋：淘宝的产品线很多，以我们目前接触到的淘宝的商品库为例，淘宝商品库是什么概念？就是卖家卖的所有的商品都保存在数据库上。我们都知道淘宝商品是非常多的，有可能是十亿级别的，而且淘宝每年的数据都会翻番，过去亦是如此，未来也会如此，动辄几十亿条的记录。在淘宝上，由于卖家是非常活跃的，所以每天的交易数也是非常多的，而且每天的访问量也是相当大的。更比如在一些高峰期，比如节假日，像去年的四个一光棍节促销，当时一天的访问量可就是平常的好几倍，几十倍的样子。具体的数字，在网站上应该会看到，这个数字也在不停的变化，大概会在 10 亿的这么一个数量级上。

InfoQ： 在 Oracle 切换到 MySQL 后，可能会有一些商业上的特性在 MySQL 上无法得到体现。所以在做优化之前，是否设定了一些策略的标准，比如说一定要达到什么样的标准才算是成功的优化？

余锋：我们的标准主要基于以下三个方面，首先是安全性。因为淘宝卖家的商品都存在上面，一大堆数据，商品没了，淘宝就会很难受；其次是性能，因为数据量大，而且未来增长的速度会非常快，以前我们的系统是跑在 Oracle 的小机上面，它的成本是非常高的。现在改用高性能的 PC 服务器，性价比较高，而且 Oracle 数据库的能力不足以维持淘宝的更新速度；最后是可运维性，Oracle 是有一套完整的工具，利用它们我们可随时查看相关的状态和预测它的一些变化。MySQL 作为开源产品，这部分的功能稍显欠缺，因此我们就需要根据运维团队和 DBA 的需求去开发一些工具。

InfoQ： 就是在日常优化工作的过程中帮助你去运维？

余锋：对，这也算优化的一方面，算过程的一个方面。

InfoQ： 在产品的优化过程中，是否会发现用 MySQL 替代原来的 Oracle 有些问题还是无法避免的，您是如何处理的？

余锋：是的，是有这样的问题，比如 Oracle 的双机备份，这块就做得非常好，整体的方案都不错。相比之下 MySQL 这点就比较弱，我们必须在过程中去深化解决这个问题。还有像 Oracle 使用的高端存储，它的高端存储的 I/O 能力是非常强的。现在我们改用 PC 服务器，如果用硬盘来做的话，I/O 能力会很差，甚至说 I/O 的能力在一两千 I/O PS 就已经算很高的了，这样的话相比原来的高端存储会相差一个数量级。我们在过去几个月顺利地解决了这个问题，而且结果证明，性能要比高端存储还要好。

InfoQ： 那么这个优化的工作现在是一个什么样的状态，进行中还是已经上线？

余锋：从去年 8 月开始到现在已经持续了好几个月，这个月就可以上线。其实主要的攻坚工作在两个月前就已经做好了，由于要采购设备，所以安排在这月上线，上线后即可替代原来

的 Oracle 数据库。

InfoQ： 就目前的结果来看优化成果如何？

余锋：从机器的运算能力上来看的话，比原来要高很多，从设备的总体的价格来看的话，相比原来小机的价格，现在的价格相当于原来的几分之一。说的通俗一点就是，花更少的钱做更多的事情，基本是这样的一个情况。

InfoQ： 优化 MySQL 的过程中有没有涉及到水平分片，或者垂直分片，大概的策略是什么？

余锋：这块因为之前的 MySQL 从业务的角度上我们已经优化过了。涉及到业务上的复杂 SQL 查询，基本上都把它当成 Key-Value 来处理。所以说在水平切割上就变得非常简单，我们只需要根据用户的这些 ID 把它平均地分割到集群的各个机器上去即可，所以说这块会做得非常简单，也不会有那些非常复杂的 SQL。

InfoQ： 刚才提到在去 Oracle 化过程中，最终选择了 MySQL。是怎样的原因使你没有去选择 NoSQL 或是其它技术来作为提高性能优化方面的考虑呢？

余锋：主要考虑到以下这几个方面，第一是因为 MySQL 已经是非常成熟的产品，而且也是经过大规模系统考验的，所以它的稳定性，开发的便捷性还是比较有优势的。从另外一个角度上来讲，也能最大限度的保障历史系统的迁移，这是第一个方面。此外，之所以没有采用 NoSQL 主要基于以下几个方面的考虑，第一由于市面上流行的 NoSQL 方案相对来说都比较年轻。然而对于淘宝商品库来讲，它是作为淘宝非常重要的库，是不允许我们有丝毫闪失的，不能出现任何问题。第二点，现在流行的 NoSQL 方案，从程序的架构上来讲，就是对传统数据库的过程进行了精简。但是它的存储、内存等问题其实还是存在的。只是解决问题的思路变得轻量。但是这些问题依旧在的。现在我们在数据库层面，因为平常都是从引擎层面去开发，在原码级别去做优化，甚至包括操作系统级的优化。所以我们可以很清楚的知道问题在哪里，应该解决什么问题，最终能达到什么样的效果。那这样做的好处就是我们既能解决问题，又能充分利用传统数据库的方便性以及稳定性。

InfoQ： 请您分享一下，在数据库优化方面，尤其是 MySQL 优化这部分，在优化的过程种需要遵循那些原则？

余锋：优化听起来好像没什么内容，但实际上我们做了很多的工作。首先要有明确的目标，我们的目标是什么，是为了提高性能还是提高安全性，还是提高其他的什么指标，这个目标要清晰和完整。第二点是测量，你不能说，我拍脑袋看到某某某，他怎么优化的，然后我也上去这样照着优化，这不是我们做事的风格。我们要测量现在的瓶颈在哪里？我们的系统遇到了什么问题，这点我们会借助一些工具来实现。比如说测量 CPU 的使用，我们会非常精

确的去测量 I/O 的使用，比如说在设备层面，I/O 是怎么使用的，在软件系统层面 I/O 是怎么使用的，在数据库引擎层面 I/O 是怎么使用的。通过非常精细的测量，我们就能找出，对 I/O 使用不恰当的地方，然后就是要去解决这种问题，为什么会比设想的要多呢，或者为什么要比设想的要慢呢，我们会针对此类问题做理论上的分析，甚至会去翻翻源码它到底是怎么实现的，这样我们就可以比较有针对性将问题解决。需要特别提到的是，我们这次有一个创新，用高速的 SSD 盘去做 2 级 Cache，因为传统的 MySQL 数据库都是引擎里面发挥 pool 的功能，然后直接在软件系统存储。就是引擎里边吐出来东西到那边去，命不中东西从磁盘读进去，由于是传统的磁盘，所以它的寻道时间是非常长的。那它的 I/O 其实是不高的，但是吞吐量大。为了解决问题，我们采用一种叫 PCI-E 的 Flash 卡。这个卡的特点是，它是电子盘，所以它没有寻道时间，随机特性非常好，同时它的读写吞吐量非常高，读能大概到 1G，写能达到几百兆。采用其作为二级 cache，数据从 InnoDB 引擎这边出来，不到磁盘去，而是先写到高速卡中。由于高速卡很快，它完成的 I/O PS 是微秒级的，几十微秒就写完了，然后传统磁盘却是十个毫秒级的。所以从数据库引擎的角度来看，只需写入几十微妙就解决了。写入数据开始是随机的，随机累计起来以后它就不是随机的，我们通过把它排序，就可以使其变为顺序的。然后我们再把这顺序的数据利用磁盘的高吞吐量特点，由于数据库半夜时候它可能会比较轻松。那这时候，把它导到磁盘中去，这样就大大的提高整个系统的 IO 能力。这是比较大的一个创新点。

InfoQ： 那我们对您的采访就到这里结束了，谢谢您。

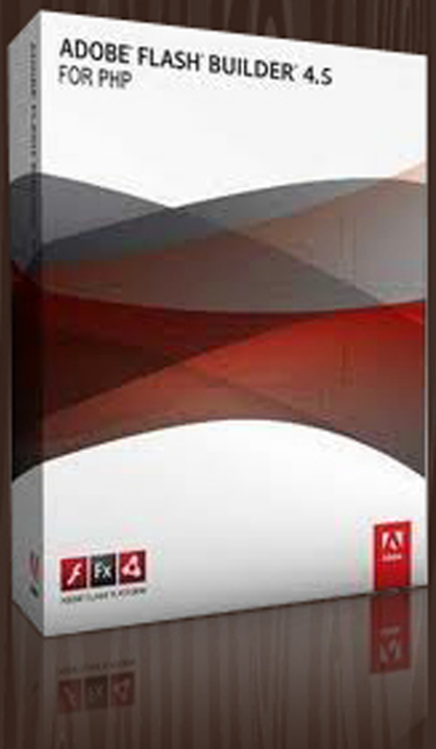
余锋：谢谢。

原文链接：http://www.infoq.com/cn/interviews/yf-taobao-optimization

相关内容：

- 面向开发的MySQL性能优化
- 优化使用BigTable的原则与方针
- Flash/SSD下的MySQL性能优化
- 为速度而生——看百姓网如何优化网速
- 空间数据库实践

Adobe



Adobe Flash Builder 4.5 高级版

隆重发布

更强的功能
更多的设备
更少的编程

免费下载试用版

Adobe官方下载（本地服务器）

- Flash Builder 4.5高级版试用版（Windows，581.2M，点击免费高速下载）
- Flash Builder 4.5高级版试用版（Mac，679.8M，点击免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版（Windows，956.5M，点击免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版免费（Mac，911.3M，点击免费高速下载）

Adobe官方下载（国外服务器）

- Flash Builder 4.5高级版试用版（Windows和Mac版本，免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版（Windows和Mac版本，免费高速下载）

InfoQ

更多下载

热点新闻 | News

MySQL 与 NoSQL--SQL 与 NoSQL 的融合

作者 孙立

写这一篇内容的原因是MySQL5.6.2 突然推出了memcached的功能。NoSQL to InnoDB with Memcached的出现，可以看出NoSQL对关系数据库的确产生了巨大的影响，个人觉得这是一个非常大的进步，可以让开发人员更加方便的使用NoSQL和关系数据库。NoSQL一般被认为性能高于关系数据库，那么直接在InnoDB之上提供NoSQL功能并和MySQL共存是否是一个更好的选择呢？

MySQL with HandlerSocket

去年在twitter上看到 HandlerSocket的出现，并宣称性能是Memcached的两倍时，非常令人吃惊，居然可以达到 750000qps。接着HandlerSocket成为NoSQL领域谈论的焦点之一， 大量的人开始想要尝试，并做过一些自己的性能测试。 下图是HandlerSocket的结构图：

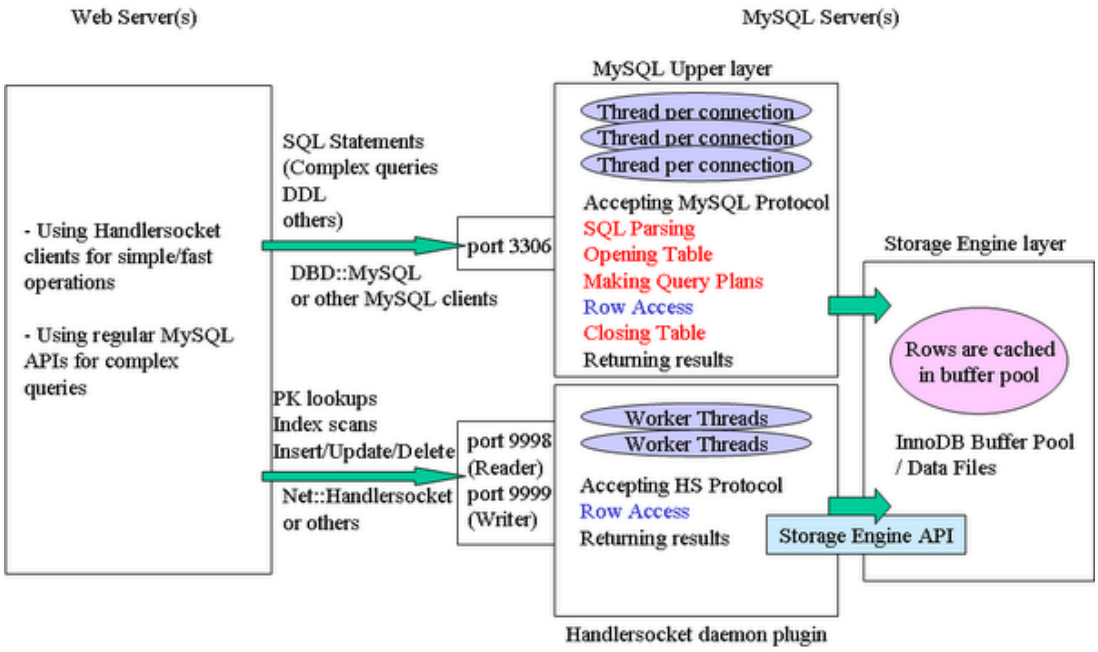


图 1 HandlerSocket 结构图（来源于官方）

HandlerSocket 的出现，给我们眼前一亮的感觉。原来 InnoDB 的性能已经足够好，并可以直接提供 NoSQL 的功能。最大的好处就是可以共享 MySQL 的功能，DBA 以前的经验一样可以用。但是有些小小的风险：

- HandlerSocket 没有与 MySQL 一起发布版本，因此对于使用 MyISAM 引擎的用户是无缘的。不过现在 Percona-Server 已经集成了 HandlerSocket，可以非常方便的使用。
- 目前大规模的成功案例并不多，国内也只有少部分公司在尝试，我知道的有飞信开放平台，据说还不错。
- 官方给出的测试数据在应用场景上其实并不充分，至少测试的场景跟我们实际使用的场景相差很大。但是毫无疑问，HandlerSocket 的性能比直接使用 MySQL 肯定要高效得多。

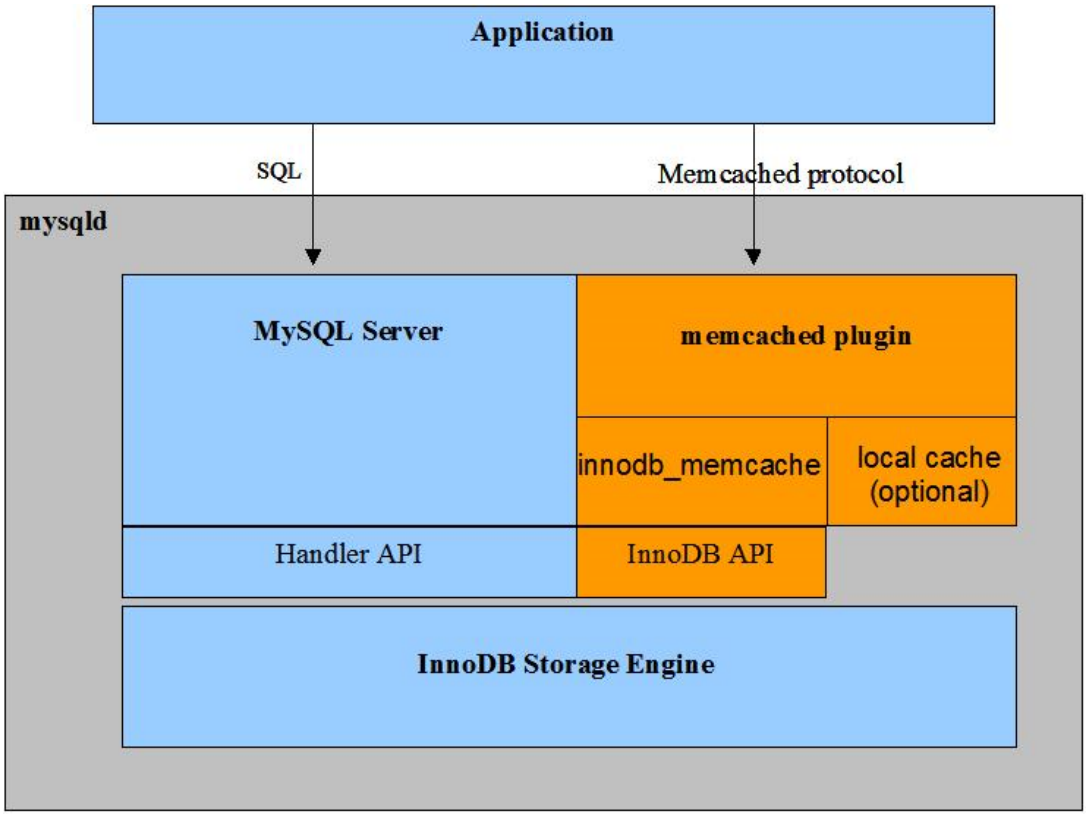
InnoDB with Memcached

也许是因为 HandlerSocket 的火爆的冲击，也许是受 HandlerSocket 的启发，MySQL 开始关注 NoSQL 领域的应用，并在 MySQL5.6.2 版本增加了通过 Memcached 协议直接访问原生 Innodb API 的功能。

InnoDB with Memcached 是在提供 MySQL 服务的同一进程中提供 Memcached 服务 ，这与 HandlerSocket 的架构模式几乎是一样的。虽然目前 InnoDB with Memcached 还是预览版本，但是我个人更看好它，因为：

- 它使用 Memcached 协议，并同时支持文本和二进制协议，在 client 的选择和成熟度上就要胜出许多；
- 其支持的三种 cache 模式，不但可以省去开发中使用 Memcached 来缓存数据的麻烦，并且具有更好的可靠性和数据一致性；
- 在应用程序中，可以使用高效的 memcached 协议来操作数据，同时也可以使用 sql 进行复杂的查询操作；

注意：目前通过 memcached 的更新操作不会记录到 binlog 中，未来的版本会支持。

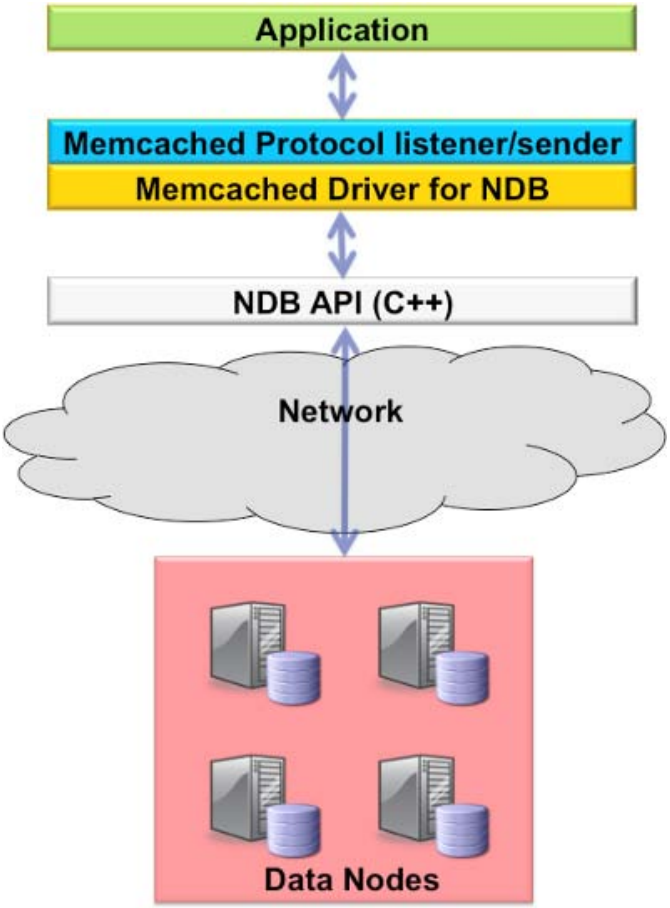


图二 InnoDB with Memcached

Memcached and MySQL Cluster

显而易见，我们会想到 MySQL Cluster 结合 Memcached 是一个更好的组合，MySQL Cluster 提供了 99.999%高可用性，并真正提供了去中心化的无缝高可扩展性。还有什么比这更人兴奋的呢。

MySQL已经提供了这样的功能，源代码在 这里 。这里有一个O'Reilly MySql Conference大会的 PPT演示 ，你也可以看下这个功能开发者的 一篇博客 。



图三 NDB with Memcached

MySQL Cluster 虽然具有高可靠性和无缝扩展的优势，但是对于复杂 SQL 查询的效率却不能令人满意。不过对于仅仅依赖于 key-value 查询和写入的海量数据存储需求，MySQL Cluster with Memcached 应该是个很好的选择。

总结

Memcached 协议由于其简单、协议轻量、存在大量的 client，所以提供兼容 Memcached 协议的产品比较占据先天的优势。

MySQL 提供 NoSQL 的功能，个人觉得并不是 MySQL 耐不住寂寞，而是的确在响应用户的需求。我前面的文章也说过，“NoSQL 只是一个概念，并不是一个数据库 产品，MySQL 也可以是 NoSQL”，现在也正应了这句话。NoSQL 从架构上就约束了开发者的架构和开发方式，从而提高扩展性和性能，而 NoSQL 和 MySQL 的融合，也同时提供了复杂查询功能。

虽然 MySQL 提供了 NoSQL 功能，如果你要尝试的话，你的数据库设计必须从 NoSQL 出发，然后再考虑 SQL 查询功能。

SQL 与 NoSQL 的融合的确会给开发者带来方便，比如最近很流行的 Mongodb，它吸引开发最大的点就是支持简单的关系查询。SQL 与 NoSQL 的融合可能是未来很多数据库产品的一个趋势。但是纯 NoSQL 数据库的优势也是显著的，就是他的简单、高效、易扩展。

参考链接：

- Mongodb的SQL支持
- Tokyocabinet的table支持
- Schema-Free vs NoSQL
- MySQL Cluster - NoSQL access with some SQL
- 我的新浪微博 @sunli1223 NoSQL微群

原文链接：http://www.infoq.com/cn/news/2011/05/nosql-mysql

相关内容：

- 面向开发的MySQL性能优化
- MySQL/HandlerSocket和VoltDB：NoSQL的竞争者
- 关系数据库还是NoSQL数据库
- NoSQL开篇——为什么要使用NoSQL
- NoSQL架构实践（二）——以NoSQL为主

热点新闻 | News

jQuery 1.6 发布：性能提升，同时包含破坏性的变更

作者 Roopesh Shenoy 译者 胡伟红

jQuery 1.6 刚刚发布了，有几处性能和跨浏览器兼容性的改进，专门重写了属性模块。

新版本中的部分重要改进：

- attr()、val()和 data()方法有了更好的性能；
- 在 attr()方法中支持 Boolean 属性；
- 添加了钩子函数，允许对 attr()和 val()方法进行扩展；
- map()方法添加了对对象的支持（将JavaScript对象的属性映射到函数）
- 允许使用相对值（“+=”，“-=”）更新 CSS；
- 添加了 deferred.always(), deferred.pipe()方法，减少了代码量，提高了代码易读性；
- 同步动画 – 现在所有动画都同步到相同的时间间隔. 使用新的浏览器功能 动画也更顺畅；
- find()，closest()和 is()可接收所有的DOM元素和jQuery对象作为参数。

所有这些新特性连同整个bug修复列表在 官方博文 中都有详细说明。

不幸的是 其中包含了部分破坏性的变更 团队已经在官方更新中用单独的章节强调了它们：

变更	jQuery 1.5.2 示例	jQuery 1.6 示例
data()方法使用破折号创建驼峰格式。	data-max-value="15" 创建 { max-value: 15 }	data-max-value="15" 创建 { maxValue: 15 }
区分了 Property 与 Attribute。attr()方法现在不能用于获取 preperty 的值。为了处理 property 的值，引入了 prop()和 removeProp()	如果复选框被选中，checkbox.attr("checked") 返回(true)	checkbox.attr("checked") 返回空字符串。 要判别是否选中，需在事件处

方法。		理程序中使用 $(this).prop("checked") 或 $(this).is(":checked")。

不出意外的话，第二项变更将会影响到现有打算升级到 1.6 的那些项目。

这一版本是在 1.5 发布后 3 个月及 1.5.2 发布后 1 个月推出的。目前 1.7 的工作已经开始了，jQuery团队已经接受了 功能建议书 。

jQuery是一个用于快速Web开发的JavaScript库，它简化了HTML文档遍历、事件处理、动画以及Ajax交互。

原文链接：http://www.infoq.com/cn/news/2011/05/jquery-1.6-released

相关内容：

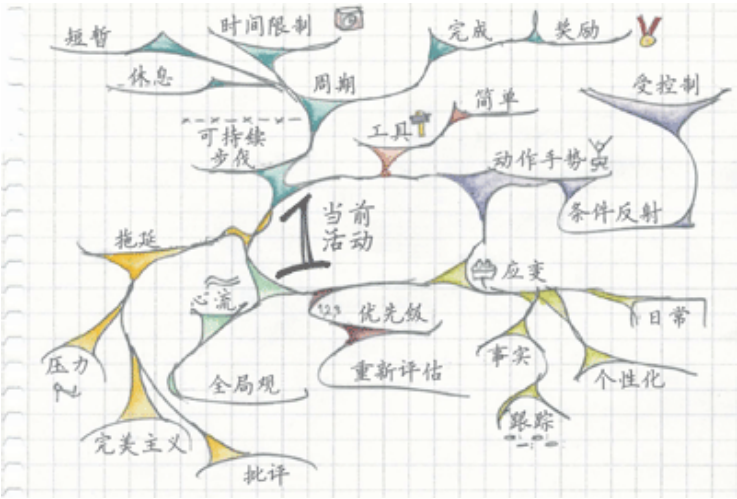
- 在Rails中应用jQuery
- 使用jQuery Mobile和JSON创建移动应用程序
- HTML5 已经起飞
- Globalization：JavaScript全球化插件
- Data Link：jQuery的数据绑定插件

热点新闻 | News

阅读者（十二）：番茄工作法图解

作者 胡键

又一种时间管理的方法？！而且还叫什么“番茄”？！当第一次听到“番茄工作法”这个名字时，这就是我的感觉。接下来才知道原来这里的“番茄”实际上指的是该方法中一个道具的名字：番茄钟。这是一种番茄状的计时钟，轻轻一扭，它便开始工作。当到达所定时间时，比如 25 分钟，它会响铃，通知你时间已到。



为什么一种时间管理方法会冠以一个物品的名称呢？问题的答案就蕴含在它的方法中：

使用道具

3 样东西：纸、笔和番茄钟。

3 种表格，每种表格一张纸：任务清单、今日待办和记录，分别用于记录一段时间内的任务、今日要完成的任务以及每日总结。

实际方法

任务计划：先填完任务清单，接着每天从任务清单中抽取出今天要完成的任务，从而形成今日待办。一般来讲，任务清单每隔一个阶段做一次，今日待办则是每天都要进行。

任务执行：从今日待办中选出优先级最高的任务，开启番茄钟，工作，番茄钟响铃，立即休息。以此循环序列依次完成今日待办中的任务，每次番茄钟结束时，需进行记录。

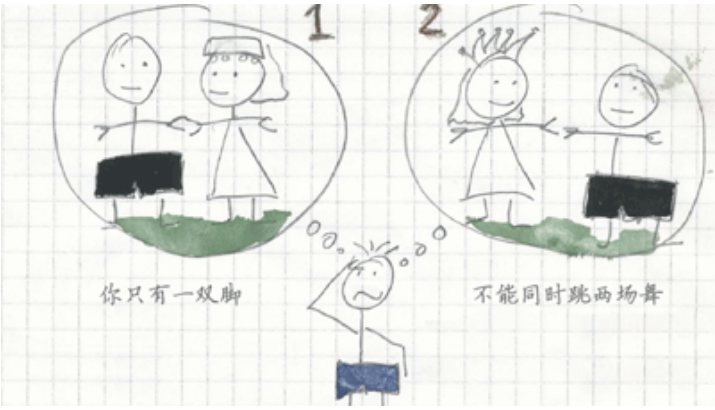
任务检查：每日结束时，检查完成的番茄钟数，了解自己任务执行的情况。

任务反馈：根据检查结果，重新评估计划列表。

看出来了吗，“番茄”在这里不仅仅指的是番茄钟，而且还是度量你全神贯注完成任务的单位，既然担负着这样重要的责任，这种方法以它命名自然就不奇怪了。等等，上面的方法哪里体现了全神贯注的意思？没有吗？对不起，那是我疏忽了！全神贯注、全力以赴正是番茄工作法强调的核心，这也是它不同于其他 GTD 方法的之所在：在每个番茄钟里，你必须排除一切干扰，全身心地投入到手头的任务上。

这怎么可能，有谁没有在工作时接到过电话？！我听到了你的叫嚷。没错，我有过。番茄工作法也没有要求你做到与世隔绝，完全“两耳不闻窗外事”。至于如何处理这种“突发状况”，这里先卖个关子，否则把内容全讲了，叫人家如何卖书，做人不能这么不厚道;)。

上面的内容你都可以在《番茄工作法图解》这本书里找到，而且更详细。虽然它并不是我读到的关于该工作法的第一本书（实际上它是第二本），但却是最轻松愉快的一本。不仅仅是因为它的作者本身就有计算机背景（它的出版公司是大名鼎鼎的 The Pragmatic Programmers，别跟我说你没有读过或听说过它出版的那本著名的《程序员修炼之道》），而且还因为书中配以了大量妙趣横生的插图，还是彩色的！



整本书的篇幅并不大，不过区区 136 页，而且还有大量的插图。翻得快的话，一两个小时就可以读完。可短并不代表内涵就不丰富。恰恰相反，从这本书中，你可以看到：

- 为何番茄工作法卓有成效
- 相关的背景理论知识
- 番茄工作法的核心方法
- 处理工作中断
- 通过记录的历史数据对未来进行预估
- 根据自己的特点进行调整

- 在团队中应用方法

本书的节奏非常快，但又不因为快而致使重点丢失。这种风格跟 InfoQ 的迷你书非常相似，对于如今快节奏生活的人们来说，这或许是一种快速获取新知识的好方法吧。团队应用，是本书的另一个亮点，同时也是其他 GTD 书籍里少有涉及的内容。考虑到作者本身就是从事软件开发，确切的说是使用敏捷方法的开发人员，将它应用到结对编程这样的实践里应该不足为奇。至于如何应用呢，还是到书中看个究竟吧。

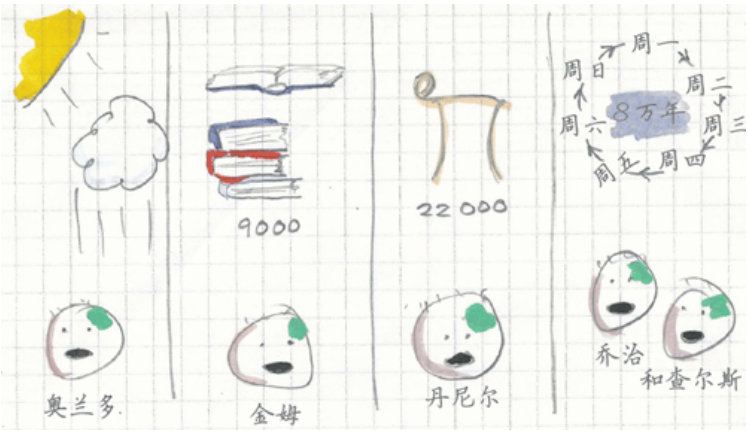
另外值得一提的是，给本书作序的两位也来头不小，一位就是番茄工作法的发明者本人，而另一个本则是广受欢迎的 InfoQ 迷你书《硝烟中的 Scrum 和 XP》的作者。有他二人的作序，这会是一本怎样的书呢？:)。

最后，作为本书中的额外收获，对番茄工作法感兴趣的读者，还可以从 番茄工作法的网站 上下载 免费的中文版 ，它正是我读到的第一本关于番茄工作法的书。

InfoQ中文站SOA社区编辑 胡键

大家好，我正好使用番茄工作法写这篇《番茄工作法图解》的书评。雷锋说："人的生命是有限的，为人民服务是无限的，我要把……"，我们或许只能仰视雷锋同志的崇高精神，但是都意识到生命是有限的、宝贵的，大家需要时间来赚钱养家……如果你在工作或者生活当中总感觉时间不够用或者做事效率较低，那么番茄工作法可能会改变你的状态，番茄工作法并不是一个特别新的模式，已经流行了一段时间，而这本《番茄工作法图解》是又一本介绍该方法的著作，有什么独到之处？

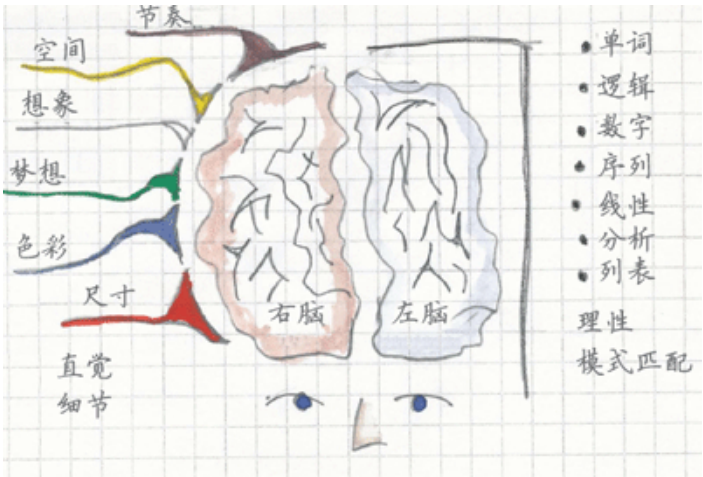
- 作者 Staffan Nöteberg[瑞典]并不是番茄工作法的创始人，但是他是该方法的亲身实践者和受益者，所以他以自己的实际经历写成了这本书，其中有自己的建议、实践体会等等，很适合想尝试番茄工作法的朋友。
- 这本书很薄，没有多少字，适合性子急的读者，估计都会一口气读完，不会因为恐惧"大部头"而放弃对番茄工作法的了解。
- 图文并茂，每一页上半部分都是作者的手绘图，下半部分是几行文字，多感观的信息输入会加深读者的印象，提高了趣味性，不会感觉厌倦。
- 翻译得比较地道，因为这本书并不是什么软件开发某个特定领域的专业书籍，原作内容天马行空，属于散文式的结构，涉及的领域比较广泛，如心理学、社会行为学等，翻译起来有难度，我个人觉得中文版本做得比较尽心。



说了这本书的特点，再说说番茄工作法的内容。简而言之，就是通过定时机制在一段时间内让自己集中精力解决一件事情，记住：一段时间只做一件事情。番茄工作法建议把时间段分为 25 分钟，当然开始的时候如果无法保证精力集中的状态，可以将时间段减少，例如 15 分钟，等加强训练之后，可以再递增。另外，这段时间只允许关注一件事情，不能分心。大家在平时工作中总是面临各种各样的事情，有紧急的，有长期的，在选取任务时必须查看待办事情清单，确定优先级最高的任务， 然后评估其使用多少个番茄钟（25 分钟），一旦确定下来，就要集中精力做这件事情。就像我现在一样，评估写这篇书评需要花费两个番茄钟，然后开始动笔。一旦动笔之后，避免干扰，工作情境的切换特别耗费人的精力，也会影响士气，如果现在被人打扰，那么将他们的问题记录下来，稍后再处理，继续把心思用在既定的事情上。完成一个番茄钟后，强制自己休息一段时间（5 分钟），离开电脑，离开办公桌，最好去外面走一走。然后再回来继续下一个番茄钟，OK，我的第一个番茄钟结束了，需要休息了，离开键盘，5 分钟后再见!

我回来了，刚才去走廊里伸了伸懒腰，呼吸了新鲜空气，回来再设置一个番茄钟。《番茄工作法图解》提到了在尝试番茄工作法时可能会遇到的各种问题。我在看了这本书之后开始使用番茄工作法，面临的第一个问题是总想超时，你知道，一旦开始了某项工作，不知不觉就到 25 分钟了，舍不得放下，但是仔细想想，还是要休息的，因为你或许能够再坚持一个番茄钟，但不是可持续发展的路线，短暂的休息会让你效率更高。

哦，我的手机短信提醒铃声响了，待会再看，我不能走神，继续写书评。



提到番茄钟，大家可千万不要把厨房里的定时器拿到公司的办公室，那嘎嘎的计时声.....按照《番茄工作法图解》的建议，我找到了一些免费、开源的软件番茄钟，向大家推荐 Pomodoro Timer，无需安装，直接运行，在右下角是个番茄的图标，随时可以开始一个新的番茄钟，能够设定时间段的长度，并把以前的完成情况做了历史统计，我最喜欢它

的一点在 于，它可以模仿计时的声音！也就说，虽然不能使用实际的闹钟，但是通过模拟计时声，你也可以身临其境。作者Staffan Nöteberg特意提到，这种计时的声音可以引起紧迫感，有助于集中精力完成任务，当然还要视个人的具体情况而定。

书评写的差不多了，但是还有时间，番茄工作法建议大家不要提前完成任务，在剩余时间内进行总结和回顾，也就是反馈非常重要，有助于为以后更好的进行工作提供经验和改进建议。每天可能会有十几个番茄钟的工作，作者 Staffan Nteberg 建议大家在一天工作之后，对照着完成的工作清单和番茄钟的历史，反思一下出现的问题，比如有多少次主动或被动中断？哪些任务的番茄钟数量评估的有问题？今后如何改进等等。不断的自我反馈会有助于更好地实施番茄工作法。

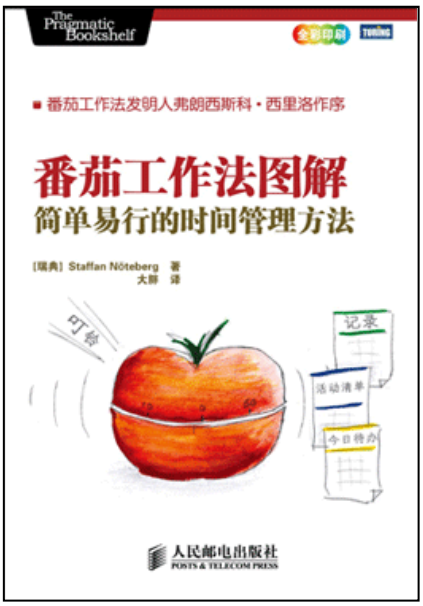
除此之外，《番茄工作法图解》还提到了其他与该方法有关的问题，比如在多人合作时怎么实践番茄工作法、如何应付中断等。

总之，对番茄工作法感兴趣的朋友可以阅读《番茄工作法图解》，应该会对该方法有一个完整的了解，当然，光说不练假把式，希望大家在放下这本书时，能够积极的投身到方法的实践中，相信会对大家提高工作效率有积极的促进作用，等你的好消息！

番茄钟还有 5 分钟，我的这篇书评对应的两个番茄钟就要结束了，我再回顾一下这篇书评，看看有什么严重问题。

OK，检查完毕，番茄钟也响了，正如它的提示声音：well done！

InfoQ中文站Java社区编辑 崔康



图书信息

原书名： Pomodoro Technique Illustrated

原出版社： Pragmatic Bookshelf

作者： (瑞典)Staffan Nöteberg

译者： 大胖

出版社：人民邮电出版社

ISBN：9787115246691

出版日期：2011 年 2 月

内容简介

本书介绍了时下最流行的时间管理方法之一——番茄工作法。作者根据亲身运用番茄工作法的经历，以生动的语言，传神的图画，将番茄工作法的具体理论和实践呈现在读者面前。番茄工作法简约而不简单，本书亦然。在番茄工作法一个个短短的 25 分钟内，你收获的不仅仅是效率，还会有意想不到的成就感。

本书适合所有志在提高工作效率的人员，尤其是软件工作人员和办公人员。

番茄工作法的来源

番茄工作法是弗朗西斯科·西里洛于 1992 年创立了。他在大学生活的头几年，曾一度苦于学习效率低下，“于是我和自己打赌，下猛药，狠狠鄙视自己说：‘我能学一会儿习吗？真正学上 10 分钟？’ 我得找个计时教练，谁来替我掐表呢？后来我找到了，是一枚厨房定时器，形状像 ‘西红柿’（Pomodoro ，意大利语的“番茄”）。就这样，我邂逅了我的番茄钟。”

说明：本文插图全部来自于《番茄工作法图解》书中的插图。

原文链接：http://www.infoq.com/cn/news/2011/05/Pomodoro_Technique_Illustrated

相关内容：

- 番茄工作法：试试看？
- 阅读者（九）：编程人生
- 阅读者(十一):简单之美
- 敏捷团队如何雇佣人才
- 为什么“赶工”没有收效

热点新闻 | News

腾讯力推开放平台 Q+

作者 崔康

这几天，有关腾讯推出开放平台 Q+的消息引起了国内开发社区的广泛关注，大家都在努力从其简短的视频介绍中找出 Q+背后的故事。19 日，腾讯公司的两位高管向媒体详细阐述了 QQ 的开放战略和 Q+平台的重要功能，让大家更深入地了解其中的意义和影响。

5 月 16 日，腾讯通过官方网站正式推出开放平台Q+，并提供了 宣传视频 。随着Q+的推出，除了QQ提供给广大用户的服务外，第三方开发商也能够以合作者的身份，与腾讯一起为QQ用户提供更细致、更有针对性的服务。据 了解 ，Q+平台为用户提供了一个虚拟的应用桌面：

> Q+”是一个基于 Windows 系统的开放式应用平台，当用户一键切换到“Q+”后，它将接管原有的 Windows 桌面。通过它，用户可以便捷地进入互联网丰富多彩的世界，享受到第三方应用商提供的丰富的互联网应用。
>
> 初一看 Q+和自家的 WebQQ 以及其他桌面类软件（360 桌面等）比较相似，但就目前来看，Q+的野心似乎更加庞大。据了解，“腾讯将尝试以 API 接口的形式通过 Q+向第三方应用商提供如内容分享、文件传输、语音视频等核心功能组件，第三方应用商则可通过这个平台进行调用，将这些用户使用最多、最喜爱的核心功能植入到创新应用中，从而直接服务于超过 6.47 亿的 QQ 用户创造更大的价值”。一旦这种合作形式步入正轨，也就意味着 Q+平台上的应用数量将不成问题，无论是微博、在线视频、即时交流这些热门应用还是其他互联网应用，只需要一个 Q+平台即可实现。
>
> 想象一下，如果腾讯将 Q+整合进 QQ 客户端或者和 WebQQ 打通（相信这个可能性会相当高），QQ 用户将自然而然成为 Q+平台用户，几乎不需要额外使用成本，接受起来也会比较容易。腾讯若能进一步充分利用云计算，将用户的 Q+桌面同步到服务器，就可以大大降低对于操作系统的依赖。这也意味着将来无论你到哪里，只要有一台能上网的电脑，登录 Q+平台后就和自己的电脑相差无几了。

为了让开发社区更深入地了解 Q+平台，腾讯 CTO 熊明华、即时通信产品部总经理殷宇在 19 日向媒体详细解释了 QQ 的开放战略。

据殷宇 介绍 ，腾讯Q+开放平台首期将开放四个方面的能力：

> 首先，开放应用平台，可以为第三方应用提供展现、搜索查找的功能；
>
> 其次，开放统一账号体系（Open ID），即第三方应用进入平台之后不需要重新建立账号体系，可以直接通过 QQ 统一的账号体系分享其用户；
>
> 再次，开放推送通知的能力，比如在 Q+平台上，视频、音乐甚至电商类的应用可以通过弹出消息来通知用户；
>
> 最后，开放分享能力，如果用户在使用应用中需要将信息分享给好友，可以直接在平台中分享给好友。
>
> 不过，腾讯的开放举措将逐步推进，为了不让第三方开发者仅仅将 Q+开放平台当做一个推送广告、营销等信息的途径，腾讯方面后续将建立完整的审核条例、举报机制，以保证让优质应用进入开放平台。

腾讯CTO熊明华 表示 ，Q+是腾讯开放战略中重要一环，QQ庞大用户群和核心价值带给第三方，也希望第三方应用为腾讯带来活跃的用户。对于第三方应用可能与腾讯自身服务存在竞争关系的问题，熊明华指出选择权在用户：

> 我们平台是开放的，用户选择可以装谁，由用户喜好决定。
>
> 我们提供一个平台，由用户和第三方开发者选择，在我们的体验中，有个应用盒子，用户可以寻找喜欢的应用。至于排序、推荐，我们会按照最受欢迎、最新推出、下载量等标准进行分类。用户的评价越高，位置可能越高。
>
> 腾讯对产业的认识不断进步，腾讯更多采取平台战略，如果第三方开发出很好的应用，可以去投资。我们持合作共赢的态度。如果团队好，可以帮助团队一起成长。

对于双方的分成模式，殷宇 表示 还在探讨中："游戏已经有成熟的盈利模式可以借鉴，如道具收费；下载不一定是收费的模式，对于不同的应用，不会一刀切。"

针对业内人士对Q+平台产品创意保护的顾虑，熊明华 回应 称Q+会保护开发者的创意，不允许其他公司以纯粹抄袭的方式进行复制：

> 相关应用是有审核机制，但同时也欢迎有更好的应用登录平台。开发者如发现权益受损，可透过举报投诉，与冲突开发商沟通协商甚至通过法律手段解决
>
> 腾讯方面开放的策略不会限制某一个开发者，应用存活下来的情况仍然是用户说了算。

腾讯推出开放平台是本着双赢的想法，一方面新型应用开发者需要我们的用户资源等，而腾讯方面也需要更多好的应用来拉升活跃用户，这或许会成为腾讯新的盈利增长点。

互联网评论家魏武挥 撰文 把Q+与Windows桌面、360 桌面和百度框计算做了对比：

> 在 Windows 系统上，很早就有所谓的 widget（即插件），这些插件都是些很轻的软件，用户可以从网上下载下来，成为自己的桌面侧边栏工具，一般以天气预报、CPU 耗能、随手贴记事，当然还有小游戏为主。这些插件来自于各种各样的公司或开发团队，并没有一个巨头将它们组织起来。事实上，微软对这个事也是有点三心二意地任其发展的意味，道理也很简单，你们怎么竞争，不都是在我的平台上嘛。
>
> 后来 360 推了一个名为“360 桌面”的应用，这是一个能够把本机安装的软件自动收拢到桌面的一个“目录夹”的功能，自然地，在这个“目录夹” 里，也会有 360 向你推荐的你尚未安装的软件。利用 360 桌面，你就不必使用这样一个路径来完成软件使用了：开始—>程序—>软件选择。有些人会想，在 Windows 桌面上建一个软件快捷方式不就得了，但显然桌面上太过凌乱，不是什么好的体验。更何况 360 桌面，还能向你推荐软件。
>
> 但 360 桌面走的是类似 App Store 的路子，它并没有替代系统的野心——至少从产品本身来看。“Q+平台”则不是。用户在登录 QQ 后，可一键完成从操作系统向“Q+”的切换，直接使用各种第三方应用。在切换完成后，Windows 系统那些诸如“开始”的按钮便全然不见了。
>
> Q+平台”，最适合的类比是百度的“框计算”。在今天的百度搜索框里，当你输入“小游戏”时，第一个你能看到的，就是共 38 页、每页 20 个的 小游戏图标，你可以直接点击这些图标进行游戏而无需离开百度网。百度框计算的正面说法是帮助用户尽快找到他们想要的东西而无需繁琐点击，但负面的批评，则是认为百度意图将整个互联网都框进它自家的网站中。
>
> 腾讯这个平台，与百度在“云”上发力所不同的是，它直接在“端”上发力——你的桌面。从路径上讲，只要用户使用电脑，想进入云的前一个动作总是从端开始。故而，“Q+平台”比框计算具有更大的野心，以及对用户更强悍的控制力。从竞争策略上讲，腾讯已经比百度棋高一招，因为这是和做操作系统差不多的东西了。

搜狐IT评论 说腾讯的开放是走了一步有风险的好棋：

> 互联网进入了一个聚焦时代，流量开始集中，用户习惯也逐渐形成。小企业不得不把有限的资源投向产品本身，因为推广已经太不容易。于是，它们选择投奔巨头，为流量买单。B2C 行业尤为明显，在淘宝、凡客诚品和京东商城上，联营品牌越来越多。
>
> 当这个概念深入到整个互联网行业时，腾讯就成了众多企业眼中的票房保证。高朋网就是个例子。腾讯和 Groupon 各持 50%股份，这个奇怪的股权比例以后也许会经常出现。说白了，彼此可以互惠互利，Groupon 通过腾讯刷流量，拿到了潜在的市场份额，而腾讯有自己的 QQ 团购，它需要行业巨头的经验。
>
> 其实，腾讯早为“开放”做足了准备。通过腾讯游戏、QQ 空间，腾讯积累了经营一个开放平台所需要的大部分经验，从产品规划，到产品变现，以及对整个产业链的梳理，腾讯都在有的放矢。多年之后，也许我们毫无觉察，但 Q 币已经成为互联网的“法定货币”。
>
> 不得不说，腾讯比其他巨头更适合开放，IM 天然具有平台性，这也是百度做 Hi，阿里巴巴做阿里旺旺的初衷。但腾讯太成熟了，平台在短期内也许会伤害到它的主营业务。腾讯 2011 年一季度的财报显示：总收入 9.668 亿美元，比去年同期增长 50%，盈利更是达到 4.399 亿美元，净利率由上一季度 的 40.1%增至 45.5%。以腾讯目前的规模来说，单是保持这样的高增长率和净利率就是一件不可能完成的任务，更何况腾讯的收入绝大多数来自于收费用 户，而不是广告。
>
> 从长远来看，“开放”会给腾讯带来一次质变，但这绝不应该是一场零和游戏。腾讯应该做好牺牲短期利益的心理准备。简而言之，至少腾讯产业共赢基金应该多投向那些小的竞争对手，而不是自己。

细心的读者可能发现，最近腾讯除了推出 Q+开放平台，还实施了两次投资入股：

> 5 月 6 日，腾讯以 45 亿元溢价购入华谊兄弟 4.6%的股权，成为其第一大流通股东。
>
> 5 月 17 日，腾讯 公布 了对艺龙网的战略投资，以 8840 万美元价格获得了艺龙网 16%的股权，成为其第二大股东。

腾讯的下一步棋会是什么？拭目以待！

原文链接：http://www.infoq.com/cn/news/2011/05/tencent-open-platform

相关内容：Google开源实时通信项目WebRTC、开放平台回顾与前景展望

热点新闻 | News

微软 85 亿美元收购 Skype 引发热议

作者 崔康

微软斥资 85 亿美元现金 收购 网 络电话服务商Skype，这是其公司 36 年历史上金额最大的收购交易。虽然Skype目前在全球拥有 6.63 亿用户 ,但由于其主要提供免费的视频电话服 务，所以长期处于亏损状态。微软以高价收购Skype，用意何在？这笔交易会对互联网格局产生哪些影响？InfoQ中文站综合了各方观点，希望给读者一个全面的事件分析。

微软CEO鲍尔默和Skype CEO贝茨在 新闻发布会 上 表示，Skype技术将整合到Xbox Kinect和Windows Phone中，微软将继续向非微软平台的Skype客户端进行投资和提供支持。分析人士认为，从长远来看此次收购对微软是有利的。从战略层面来 看，Skyp庞大的用户群无疑将让微软主导网络电话和视频会话市场。收购也挫败了竞争对手谷歌利用Skype技术完善通话服务的计划。鲍尔默认为，“快速 增长”的用户数是微软高价收购Skype的原因之一。

《南方都市报》撰文 分析，微软以如此高价完成收购显示了其在互联网领域争夺市场的巨大决心，并将与谷歌、苹果展开正面交锋：

> 85 亿美元收购一家年营业额不到 10 亿美元且一直没有盈利的公司，微软此举确实出乎市场意料。在微软 36 年的历史中 ,通常并不会发起巨额收购。斥资 85 亿美元收购 Skype，这将是微软成立以来规模最大的一笔收购。2006 年，微软曾花费约 60 亿美元，收购互联网广告公司 aQuantive。3 年前，微软一度希望以 480 亿美元收购雅虎但被后者拒绝，不过，让雅虎股东失望的是，雅虎今天的市值尚不及 480 亿美元的一半。
>
> 85 亿美元的报价，充分说明了微软收购 Skype 的决心之大。在互联网领域，微软一直没有太大的作为，谷歌的兴起，让微软面临巨大挑战；如今，整个世界的目光都盯在了网络新贵 Facebook 上。在世人眼里，微软在互联网时代是个不折不扣的落伍者。
>
> 不过，微软一直不甘心自己被互联网时代所淘汰，微软 CEO 鲍尔默将互联网视为该公司的主要战场之一。MSN 即时通讯、必应（bing）搜索、网络广告等成了微软在互联网领域迎战谷歌的三大阵地。然而，相对于谷歌在互联网行业的呼风唤雨，微软的互联网战略一直没有真正展现出让业界看到其能够挑战谷歌的核心优势。在微软的营收和利润

中，来自互联网业务的贡献几乎可以忽略不计。投资者们越来越担心，在苹果、谷歌及其他公司的竞争下，微软的两大核心业务（Windows 操作系统和 Office 办公软件）的利润承受压力。

此次微软不惜以 85 亿美元的天价将 Skype 收入囊中，在业界看来，无疑是微软在互联网领域的背水一战。

业内人士分析，如果微软收购 Skype，那么它将能够把这种大受欢迎、用户众多的服务整合到必应中，与谷歌展开竞争；还可以整合到 Windows Phone 平台中，在手机领域与苹果 iOS 和谷歌 Android 竞争；甚至与 Office 整合，帮助企业用户更好地协作。

在互联网领域，不论是在搜索、网络广告等核心业务领域，微软的市场份额都远小于谷歌；而在火热的智能手机和平板电脑领域，微软的 Windows Mobile 操作系统更是被谷歌的 Android、RIM 的黑莓和苹果的 iOS 挤压得几乎失去立足空间。

对于微软收购 Skype，业界分析其最大的目的，就是将在微软开拓移动手机市场中发挥作用。微软在去年发布了手机操作系统 WindowsPhone7，但是，这个操作系统并没能完全帮助微软提高市场份额。通过收购 Skype，微软希望阻止谷歌在互联网通讯领域占据更大优势。独立分析师罗伯·安德鲁（Rob Enderle）就表示："这是微软与谷歌之间战略抗衡的一部分。"

美国科技博客 TechCrunch 撰稿人埃里克·施恩菲尔德也分析称，Skype 集成到 Windows Phone 系统中将是一个强有力的结合，并将给予微软挑战其它竞争对手谷歌、苹果的机会。"别忘了 Skype 早已在 iPhone、Android 平台得到了很好的应用。"

此外，Skype 还可以用于电话、视频会议和在线会议等通讯用途，这十分适合于微软的企业业务，有利于微软与思科的 WebEx 展开竞争。

《纽约时报》也评论说，这次交易是迄今为止微软在个人用户市场"最聪明"的一次收购行为：

> 首先，微软将会获得 Skype 的 1.07 亿用户，这些用户平均每月通话量高达 100 分钟。更重要的是，拿下 Skype 之后，微软将会在价值几百亿美元的电信市场成为一个重要的玩家。
>
> 众所周知的是，今天绝大多数消费者正在从传统的固定电话、移动电话转向包括 Skype 在内的网络电话服务。拥有 Skype 的微软将站在网络电话领域的最前沿，同时给准备进入这一领域的竞争对手带来巨大压力。

谷歌和苹果都已经在网络电话上投入了数年时间。苹果一直在力推 iChat 和 FaceTime，谷歌也在普及其网络通信相关产品 Chat 和 Voice。不过，谷歌和苹果都有一个共同的问题，他们很难在竞争对手的平台上生存。苹果用户无法和谷歌以及微软的用户进行视频聊天，谷歌在跨平台方面做了很多努力，但是仍有不少障碍。

不过 Skype 则没有这一问题，作为免费服务的 Skype 可以运行在苹果、Linux 以及 Windows 电脑上，也支持 Android 和 iPhone 等手机平台，甚至可以在电视机上运行。

《PCWorld》列举了微软必须购买Skype的 五大原因 ：

1. 企业协作

微软收购 Skype，其中很明显的原因就是针对企业协作领域。微软 Lync 服务将电脑连接至 PBX，以提供 VoIP 通话、即时通讯和视频会议服务，Lync Online 通过云提供服务。Skype 拥有相同的工具 Skype Connect。收购 Skype 将能使微软在这一领域大展拳脚，并防止谷歌与苹果进军这一市场。

2. 推动必应

微软还可以借助 Skype 推动微软必应搜索引擎的增长。Skype 在全球预计拥有 6.6 亿名用户，每月活跃用户人数达 1.7 亿。微软当然会竭尽全力把这些用户推向必应。此外，微软预计还会推出一些 Bing-Skype 双向直接链接。

3. Windows Phone 7

尽管微软作出了各种努力，但 Windows Phone 7 增长仍持续低迷。微软与诺基亚之间的合作无疑会带来帮助，但双方的合作还需要数年才能真正“瓜熟蒂落”。而且仅仅依靠交易本身所取得的效果也并不足够。

收购 Skype 或许可以帮助微软推动 Windows Phone 7 的发展。将 Skype 进一步整合进 Windows Phone 7 操作系统之中，而不是像 Skype 在 Android 和 iOS 平台仅仅作为客户端来使用，这将使 Android 或 iOS 难以与 Windows Phone 7 相媲美。

4. 使 Skype 远离谷歌

收购 Skype，微软不仅仅是为了从中受益，微软还希望借此使 Skype 远离谷歌的“魔爪”。如果谷歌收购了 Skype，这对微软来说无疑是一个沉重的打击。想想吧，如果谷歌收购了 Skype，并将 Skype 整合进 Google Apps、Gmail、Google Talk 和 Google Voice 等谷歌产品之中，再想想如果谷歌将 Skype 与 Android 平台进行一些深度整合，这对微软来说，无疑是一场灾难。即便微软并未很好的将 Skype 整合进自家产品之中，但这也避免了谷

歌通过收购 Skype 来获得巨大的价值。

5. 打造品牌效应

微软需要收购 Skype 的最后一个原因是，该交易有利于打造微软的品牌效应。

打造品牌不仅仅是获得了大量的正面宣传，它还会有巨大的经济效益。WPP 旗下市场研究机构 Millward Brown 日前发布了 2011 年 BrandZ 全球品牌百强排行榜，榜单显示苹果已经超过谷歌成为全球最有价值的品牌。其品牌价值高达 1530 亿美元，微软在该榜单中位列第五，品牌价值为 780 亿美元。所以打造品牌会对经济效益产品实质影响。

不过，纽约时报也 指出 ，微软收购Skype难讨运营商欢心：

> Skype 允许用户绕过运营商，通过数据网络免费或者便宜打电话，如此一来，人们可能会更加远离昂贵的语音套餐，从而使运营商的营收大大受损。
>
> 当然，Skype 已经在大量智能手机上亮相一段时间，包括 iPhone 和 Android 手机。
>
> 伯恩斯坦（Bernstein）分析师克莱格·莫菲特（Craig Moffett）说，如果硬件巨头和软件企业决定向此类服务投资数十亿美元，那么风险就会大增。他说："微软向该服务、向 Skype 提供支持，事情就会完全不同。" 如此一来，微软和运营商的关系会冷却，微软正打算和诺基亚合作，夺回丢失的领地，被运营商冷落不是个好兆头。
>
> 克莱格·莫菲特说："最终，在自己的店铺销售什么还是运营商说了算。不清楚微软如何说服运营商，让它们将 Skype 当成朋友。"

《华尔街日报》则从投资的角度对微软收购Skype之后的盈利能力表示了 担忧 ：

> Skype 现在的投资者大都是在 18 个月前投资入股的，那时 Skype 的估值只有 27.5 亿美元。相比之下，微软那些长期受苦受难的股东想要见到收购 Skype 带来的稳健收益恐怕是不容易了。
>
> 是的，Skype 有庞大的用户群和著名的品牌。截至去年 12 月 31 日，其用户数为 6.63 亿。但就算是在互联网时代，这也不足以保证如此高价的收购案能产生收益。毕竟，微软还需要一个像样的商业模式。
>
> 但 Skype 没有这样的商业模式。尽管利用 Skype 客户端与固话和手机通话需要缴费，但两个 Skype 客户端之间通话却是免费的。这意味着注册使用 Skype 的人越多，打免费电话的人也就越多。
>
> Skype 提交的监管备案文件显示，尽管从 2008 年至 2010 年，注册用户数增加了 104%，

> 但每月平均付费用户数只增加了 52%，只有区区 880 万。大多数人打的还是两个客户端之间的免费电话：从 2008 年到 2010 年，免费通话时长从 655 亿分钟上升至 1,943 亿分钟。相比之下，计费通话时长只从 69 亿分钟上升到 128 亿分钟。
>
> Skype 希望能针对其用户群销售广告，特别是在日益重要的视频领域。不过，想要销售广告的网站之间竞争也很激烈，Skype 成功的可能性还要打折扣。
>
> 不过，对于微软来说，收购 Skype 在业务上还是有创收的可能。Skype 可以帮助微软在新生的统一通信系统（unified-communications）市场上立足。统一通信系统能够无缝融合电话、电子邮件、即时信息和各种电话会议。市场研究机构 Gartner 的分析师 Steve Cramoysan 估计，到 2015 年，统一通信系统的硬件和软件的全球市场价值可达到 189 亿美元。
>
> 但微软也只是有可能在这个市场分得一杯羹，同时微软也得面临一些严峻的现实：统一通信系统迟迟未能取得成功，主要是因为很多企业不到万不得已并不愿意淘汰各自通信基础设施的不同组成部分。另外，这也是一个竞争激烈的市场。
>
> 此外，Skype 还要证明自己是一个可行的商业产品。那种认为 Skype 提供的消费者服务的质量不佳的观念可能会伤害 Skype 影响各项业务的能力。
>
> 鉴于这些不确定性，除了能证明收购价格高的离谱，此外什么也无法证明。微软开出的收购价是 Skype2010 年扣除利息、税项、折旧及摊销前盈利的 32 倍。微软必须有所作为才能改变投资者的情绪，但这宗交易显然不能。

据 ZOL报道，两位前微软高管李开复和唐骏也对此次收购交易发表了自己的看法：

> 前微软全球副总裁李开复先生，在微博上暗示微软之前错失了很多收购的机会，现在终于觉醒。他表示："多次被抢走好公司后，微软决定狠狠出价：450 亿收购雅虎、60 亿收购 Aquantive、还有现在的 85 亿收购 Skype。"
>
> 同样是曾出任微软中国总裁的唐骏先生，则认为 Skype 是一家有用户没有利润的公司，不值得微软用 85 亿美元收购，并不赞成收购。他分析了微软收购的主要 原因是因为："只能说微软实在需要一家有用户的互联网公司，当年微软没有收购雅虎是正确的，要是现在不收购 Skype，也会是个正确的决定……"

当年被eBay"抛弃"的Skype最终落入微软囊中，传言中的竞购者Google成为看客。搜狐IT指出，最近的Google相关书籍《In the Plex》披露了其中的内幕：

> 据悉，2009 年，负责 Google Voice 的产品经理名叫 Wesley Chan，当时作为改造产品的 Voice 尚在开发之中。
>
> 随着 eBay 转让 Skype 的消息传出，部分 Google 高层开始探讨是否是个机会，并开始参与收购谈判。当时 Wesley Chan 参加了尽职调查，他甚至去欧洲亲自查看了 Skype 总部。但他认为，收购 Skype 不明智，因为 Skype 使用的是 P2P 技术，和 Google 的 云计算新理念不符合。Wesley Chan 认为，P2P 属于一项老技术，因为会耗光用户的上网带宽。如果 Google 要收购 Skype，则必须重新编写 Skype 代码，另外，收购过程也面 临监管机构长期的批准审核过程，Wesley Chan 预计需要耗时一年半到两年。
>
> 不过当时，Google 许多人主张买下 Skype，其中包括当时的 CEO 施密特。
>
> Wesley Chan 于是开始破坏收购计划。他碰巧认识一些在 Google 有影响力的人士，其中包括 Google 的前十名员工之一 Salar Kamangar。两人找到创始人布林 ,告诉他收购 Skype 是一个糟糕的交易。
>
> 在最终的会议上，布林对于收购 Skype 作出了负面的表态，其中提出一系列无法得到满意回答的问题，比如这是不是一次统计数据驱动的收购？谁负责到欧洲出差整合 Skype？ 政府需要多长时间才会批准？
>
> 随后布林作出总结表态："我为什么要冒这个风险，我们有数亿 Gmail 用户，还要 Skype 做什么？收购 Skype 是我听到的最愚蠢的主意。"施密特听罢此话，直接走出了会议室，表示自己有反对意见。
>
> 在这一过程中，Google 另外一位创始人佩齐看着笔记本，不表态。而在 Wesley Chan 看来，佩齐其实和施密特一样，希望收购 Skype。

毫无疑问，在这次收购交易中，Skype的投资方获益匪浅。据 腾讯科技 报 道，2009 年，各投资方以 19 亿美元的现金和 1.25 亿美元的票据方式，从eBay手中购买了Skype的大量股权。经过了这些交易之后，eBay在 Skype的股权比例也下降至三分之一左右。如今微软将以 85 亿美元的价格收购Skype，以私人资产公司银湖（Silver Lake）为主的Skype投资方将从微软的收购交易中获得 50 多亿美元的资金，相当于它们投资额的三倍之多。

微软收购 Skype 的故事才刚刚开始，后期发展如何，InfoQ 中文站也会持续关注报道

原文链接：http://www.infoq.com/cn/news/2011/05/microsoft-skype

相关内容：五年Skype架构师之路的感言、Skype故障带给我们的经验与教训、自由软件基金会谨慎支持Google应对专利官司、Google就Oracle有关Android的诉讼作出回应

热点新闻 | News

D 语言和 Visual D 简介

作者 Jonathan Allen 译者 侯伯薇

D语言是一种由Digital Mars公司创造的系统编程语言，它专注于“把C和C++的能力和高性能与先进的语言——像Ruby和Python——的高生产力结合起来。”尽管它仍然是一种静态类型的语言，可以直接编译成本地代码，而且语法看起来与Java或者C#非常类似，但是它做出了很多有意义的改进。

这些改进包括明确地支持惰性参数。当我们使用关键字“lazy”标记一个形参时，就意味着相关的实参会被赋值 0 次或者多次。这与 Haskell 之类的语言不同，它会延迟对实参赋值 0 次或者 1 次，但不会再多。

为了保证程序的正确性，特别是在并行以及多线程的领域中的正确性，D 语言支持把“immutable”和“const”作为一等概念。和 C++不同的是，在 D 语言中的 immutable 和 const 修饰符是可传递的。如果我们用 immutable 标记了一个变量，那么不但不能对其进行变更，也不能变更通过它所能够访问的任何变量。const 关键字的工作方式和 immutable 类似，不过特定的对象还是可以通过 non-const 的引用来修改变量的值。

D 语言还包括了很多来自于其它语言的流行特性。例如，它拥有 for-each 循环、字符串转换、数组分片、闭包、try-catch-finally 块，以及值类型和引用类型的分离等等。同时，它还考虑到了直接对内存的访问，80 位的浮点型数字、结构成员排列控制以及内联程序集。

通常，因为安全性，把源代码转换为本地代码会付出高昂的代价。C 和 C++是最容易受到类似于缓冲区溢出之类的编码错误影响的，其它大多数本地语言也都一样。D 语言通过既在编译时又在运行时执行数组边界检查来避免这些问题。同样，非零长度的数组的内容会自动初始化为它们的默认值。

关于内存管理，D 语言和.NET 非常类似。默认情况下，引用类型会存储在垃圾回收堆中，而值类型或者会存储在堆栈中，或者存储在其它对象中。D 语言也可是分配堆栈来存储引用类型，但这不可避免要使用指向它们的指针。和.NET 一样，你还可以让指针指向非托管的内存。让这些彼此分离非常重要，因为很多你能够在非托管的指针上能够执行的操作，在托管

指针上执行是不安全的。

对于能够使用非快捷版Visual Studio的开发者来说，有一个项目叫做 Visual D，它提供了Visual Studio 插件。Lloyd在一篇文章中介绍了如何 在你的环境中安装Visual D。

当前D语言有四种实现。Digital Mars公司 为Windows、Linux、OS X和FreeBSD提供了免费的编译器。还有一种 Gnu D编译器，它是构建在gcc之上的。而 LLVM也发布了他们的D语言编译器的beta版本，最后还有 针对.NET的部分实现。

原文链接：http://www.infoq.com/cn/news/2011/05/Visual-D

相关内容：

- Code2Plan：免费的Visual Studio敏捷项目管理插件
- Visual Studio 2010 中的扩展管理器
- 阅读者(四)：C++必知必会
- Google开源其C++ Mocking框架
- 现在谈论C++与Java的遗产是否有些为时过早？

热点新闻 | News

Google Storage 全面开放

作者 Abel Avram 译者 崔康

Google在 I/O 2011大会上宣布Storage服务面向所有开发人员开放，不再需要邀请。该服务添加了OAuth 2.0 支持、通过API Console实现简便账户管理、新的欧洲存储区域和新API版本。

Google Storage for Developers (GSD) 是用于在Google云中存储数据的RESTful服务。该服务在 I/O 2010 大会上推出 ，但是只对有限的开发人员开放。在一年之后的I/O 2011 大会上，Google宣布该服务面向所有开发人员（只要有Google账号），一直到今年年底都免费提供最多 5GB的存储空间。

GSD包含一些引人注目的特性：数据跨多个数据中心复制和备份，保证读写一致性，最多容纳 5TB的对象。虽然存储域（bucket）应该存在独立的域名空间，但是我们发现到目前为止该功能还不能使用，Online Storage Manager无法通过常用的名字如“bucket”或者“test”创建新域，可能是因为这些名字已经被其他域占用。

Google还宣布了一系列该服务的 改进特性 ：

- OAuth 2.0 支持，存储访问更简单、通过 API Console管理账户
- 简化存储共享——存储所有者可以像任意拥有 Google 账户的开发人员开放。只有所有者承担相应的费用
- 新的欧洲存储区域、支持最多 5TB 的对象
- 新 API 版本，2.0

Amazon S3提供了类似的 5GB免费服务， 付费标准更低一些：0.14 美元/GB/月 vs. 0.17 美元/GB/月 ,Amazon的价格在存储超过 1TB之后更低一些 对后来的 5000TB费用是 0.055 美元/GB/月。其他GET/PUT /POST、数据上传和下载的费用都类似。

原文链接：http://www.infoq.com/cn/news/2011/05/Google-Storage-for-Developers

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

我们的价值观：
创造可信赖的内容

我们的愿景：
帮助传播企业软件开发相关的知识和创新

我们的读者：
中高端技术人员，如架构师、项目经理、高级工程师等

我们的社区：
Java、.NET、Ruby、SOA、敏捷、架构和运维

我们的特质：
个性化RSS定制、国际化内容同步更新

我们的团队：
超过60位领域专家担当社区编辑

特别专题 策划：InfoQ 中文站； 执行：霍泰稳

探索中的探索式测试

探索式测试是敏捷测试的重要组成部分，和一般性测试不同，它所关注的是“意料之外”的软件缺陷。作为一个研究性、启发性和严肃性并存的测试方法，探索式测试已经成为一般性测试的重要补充。但是，目前这一“新”技术正如它的名字一样，也处在探索的初级阶段，能够熟练掌握该技术并能应用的技术人员并不多。这也是我们本期《架构师》组织这一专题的初衷，希望能够通过在此领域已经有所研究的专家，分享他们的经验，普及一下读者对探索式测试的理解！

探索式测试的秘密 — 探索性测试中的几种误区

测试工程师的学习之旅 — 虚拟座谈会：TDD 有多美？

特别专题

探索式测试的秘密

作者 季哥

前段时间的 QCon北京 2011 大会 里面有个探索式测试的分享，是 Erik Petersen 演讲（资料下载） 的，我由于一些原因，没有在现场向大师学习，之后发现他讲的非常好，效果也很好，引起了很多人对于探索式测试的兴趣。这里也感谢InfoQ的给力，我把了 Erik的PPT看了两遍，有非常深刻的体会。也确实在某些方面开导了我。自己本身学习探索式测试快一年有余了，也不乏和国外的ET的一些大师 (James Bach, CemKaner)进行交流。

这里我先抛一个问题出来：为什么产品提交到测试这边，我们的测试工程师可以发现很多开发或用户都发现不能的 Bug 呢？大家可能觉得这个原因肯定有很多，比如有这些：

1. 我们是测试工程师，我们是专门干这个的
2. 我们很好地理解了需求，我们使用了很多且很好的测试设计技术
3. 我们有较多时间进行测试执行
4. 开发人员或用户没有时间去进行测试
5. 开发人员或用户没有时间不知道怎么去进行测试
6. 我们的运气好，突然发现了这些好的 Bug

(当然还有很多其他的因素，这里就不列出所有的了)

那么再有一个问题出来：为什么有些测试人员能发现其他测试人员不能发现的 Bug 呢？大家可能觉得这个原因肯定有很多，比如有这些：

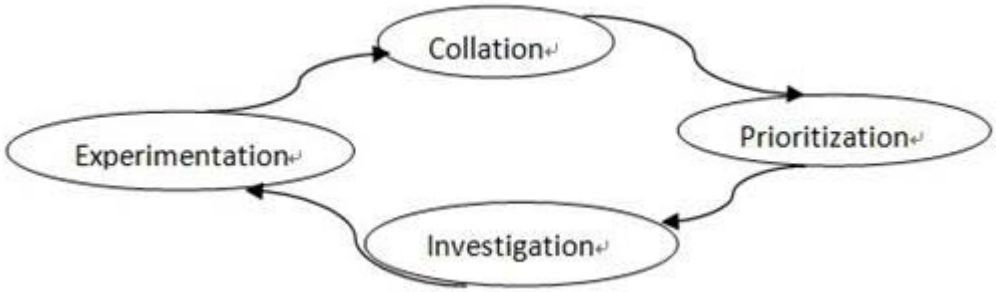
1. 其他测试人员对于业务需求不熟悉，该测试人员对于业务非常精
2. 其他测试人员没有很好的状态，注意力不够集中
3. 其他测试人员的测试经验较少，该测试人员测试经验非常丰富
4. 其他测试人员的运气不怎么好，该测试人员是突然灵感出来发现的

5. 其他测试人员的测试环境和测试数据不一样

(当然还有很多其他的因素，这里就不列出所有的了)

同样的问题还有很多，比如：为什么我们很多好的 bug 都不是通过我们已经写好的测试用例发现的呢？其实这里面我并不是想去搞清楚很多不同的原因，根据 80/20 原则，我只想关注一个主要的，那就是经验丰富的测试工程师是如何进行测试的，是如何进行探索式测试的，他们的思维过程到底是什么样的，和我们通常的测试有什么不一样的。

很多人都会问到底什么是探索式测试，也有很多人知道很多时候我们就是在做探索式测试(只是我们自己不知道而已)，不管怎样，我们都期望把很好的测试方法或手段传承下去，让新加入测试行业的同学都可以吸收这个武林秘籍。根据看 Erik 的 PPT，我这边大概抽象了下探索式测试的思维过程：



这个思维模型简称为 CPIE

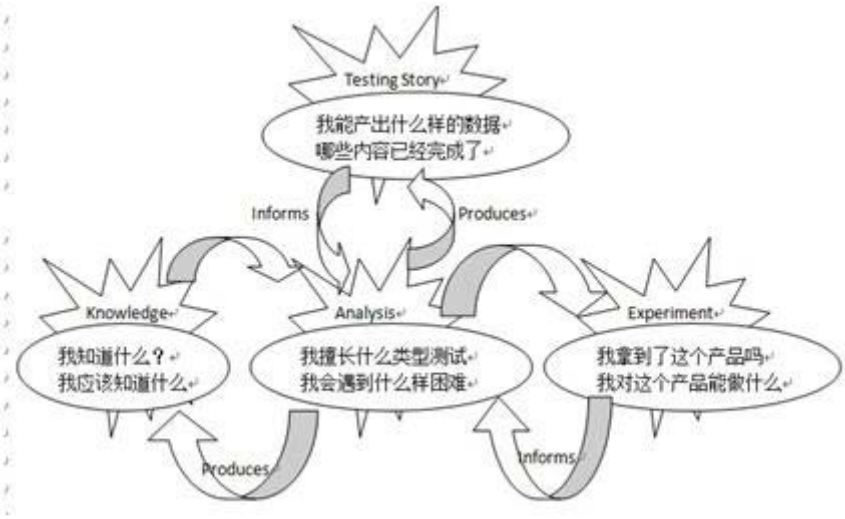
- 简要说明：Collation，这个 action 注意就是我们需要收集所有关于 SUT 的所有信息，去了解和理解。
- Prioritization，我们要对所有需要测试的任务或模块或特性进行优先级的划分，这里不说划分原则。
- Investigation，划分好后，就需要对应确定即将测试的任务进行仔细的分析并预测其可能输出的结果。
- Experimentation，这个 action 就是需要我们实际的去进行测试，看看我们的预测是否正确，我们的信息是否正确，就会变化到去影响 Collation 阶段。

这里面部分人看的出来，还是有点站在整个项目或产品测试的角度去进行测试，那实际上对于某个小的需求和功能，是如何进行探索式测试的呢？

这里面还需要强调的是探索式测试是一个测试方法，不是测试技术，大家都知道有很多测试

技术，特别是测试设计技术，这样的话，就意味着进行着探索式测试，就可以完全使用这些测试技术，可以融合这些测试技术。

下面来看下 ET 的大师 James Bach 是怎么来看待 ET 的思维过程的：



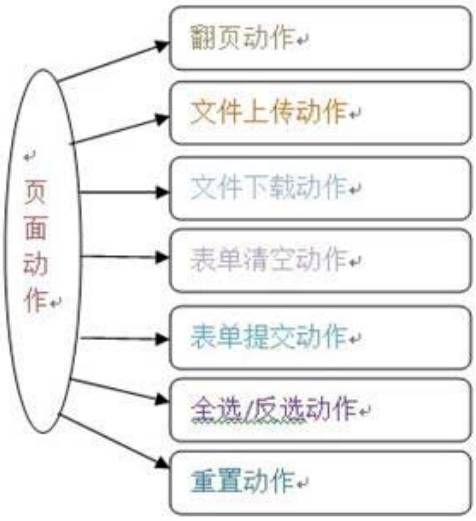
这里面可以看到和 Erik 的观点还是有很多类似的地方的，都强调 Experiment，也就是说之前我们做的再好的测试设计和用例，只要在测试执行的时候才知道好还是不好，还有没有更好的测试思路。同样可以发现这些都是一个循环的过程，ET 过程中，测试设计和测试执行是互相驱动和完善的过程，这也是和我们平时的 SBT(Script Based Testing)的最大区别。

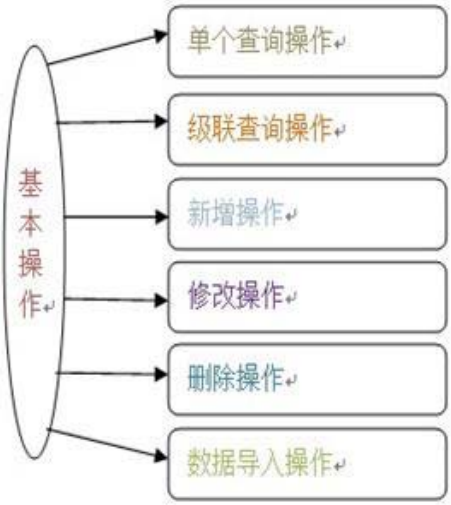
个人认为之所以优秀的测试人员能发现一些隐含比较深的 bug，主要有以下几个关键的因素：

1. 对于基本的测试设计技术的使用达到炉火纯青的地步
2. 对于错误猜测测试方法有一定的理解和应用
3. 对于开发思维习惯和异常的用户使用思路有一定的了解

这里面谈到了错误猜测方法，Erik 也提到了，这个方法在 ET 过程中使用得非常普遍，也许很多人会认为该方法很大程度上依赖于个人的测试经验积累，不错，但不意味着新的测试人员不能很好地使用该方法去进行 ET。这是又回到了模型的概念了，个人理解的模型有三个层面，如下：

1. 功能测试模型：一些常用功能的测试思路的大集合(增删改查，web 页面测试等)





2. 线下 bug 模型：基于线下测试发现的优秀 bug 抽象的模型(主要针对于常出现 bug 的地方，提出注意事项)

模型解释：

- 多线程创建，更新，删除某类数据，多线程方式包括同时打开多个页面或浏览器；使用工具多线程并发操作来校验数据操作的原子性
- 多浏览器或一个浏览器里多个 Tab 进行测试，考虑 cookie 里面的值的变化是否影响后续的操作；或使用场景探索模型去多角度校验页面信息和数据库数据的正确性

3. 线上故障模型：基于线上事故产生的原因进行抽象的模型(环境较复杂和特殊，完善我们的测试设计思路和创新出新的测试方法)

目前淘宝也在做这三方面的测试模型，以应用于 ET 的培训资料，使测试经验不丰富的同学也能够快速掌握错误猜测方法去进行 ET 测试，把这些模型完全掌握且能够应用得好，那我

们还怕我们的功能覆盖率不够全吗。

说到覆盖率 接下来说下 ET 在 Coverage 上是怎么考虑的。我们说的 Coverage 一般就是 Product coverage，同样也是这个被测产品的一部分。那么对于 Product coverage 又包括哪些方面的 coverage 呢？

第一个就是 **Structure**，也就是产品的一个因素，对于这个 Structural Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品是怎么构成的，我们要 cover 的就是构成这个产品的部 分。下面以打印机产品为例，看看 Structural Coverage 到底要考虑什么：

- 打印需要用到的文件
- 实现打印功能的代码模块
- 在这个模块里面的代码语句
- 在这个模块里面的代码分支

可以看到这个时候我们关注的是产品的内部结构。

第二个就是 **Function**，也是产品的一个因素，对于这个 Functional Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品能够做什么？我们要 cover 的就是这个产品做得什么 样。同样以打印机产品为例，看看 Functional Coverage 到底要考虑什么：

- 打印，页设置，打印预览
- 打印 range，打印复制，zoom
- 打印所有的，当前页，或指定的 range

可以看到这个时候我们关注的是产品的功能或特性。

第三个就是 **Data**，也是产品的一个因素，对于这个 Data Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品能够对数据方面有什么考虑？我们要 cover 的就是这个 产品能够处理什么样的数据。同样以打印机产品为例，看看 Data Coverage 到底要考虑什么：

- 打印文档的类型
- 文档里面的元素，文档的大小和结构

- 关于怎么打印的数据(比如 zoom factor; no. of copies)

可以看到这个时候我们关注的是产品使用过程中不同的数据处理。

第四个就是 **Platform**，也是产品的一个因素，对于这个 Platform Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品依赖什么才能使用？我们要 cover 的就是这个产品怎么 处理不同的依赖的。同样以打印机产品为例，看看 Platform Coverage 到底要考虑什么：

- 打印机，Spoolers，network behavior
- 计算机
- 操作系统
- 打印机驱动程序/设备

可以看到这个时候我们关注的是产品使用过程中不同的环境和依赖。

第五个就是 **Operation**，也是产品的一个因素，对于这个 Operations Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品怎么使用的？我们要 cover 的就是这个产品使用的步骤 是否合理/正确。同样以打印机产品为例，看看 Operations Coverage 到底要考虑什么：

- 默认情况下使用
- 真实环境下使用
- 真实的场景下使用
- 复杂的流程下使用

可以看到这个时候我们关注的是产品使用的场景(包括稳定性，可用性，安全性，可扩展性，性能，可安装性，兼容性，可测性，维护性，本地性等)。

第六个就是 **Time**，也是产品的一个因素，对于这个 Time Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品在什么时间情况下会受影响？我们要 cover 的就是这个产品在不同的时间下会表现什么样。同样以打印机产品为例，看看 Time Coverage 到底要考虑什么：

- 尝试在不同的网络或端口的速度使用

- 一个文档打印完，紧接着打印另一个文档，或隔很长时间再打印
- 尝试与时间相关的限制，比如使用 spooling, buffering, timeouts
- 尝试 hourly，daily，月底，或年底打印报告
- 尝试从不同的 2 个工作站同时打印

可以看到这个时候我们关注的是产品使用的时候是否受时间影响。

上面我们可以看到 ET 在考虑覆盖率上还是有一点自己独特的角度，但感觉不是很具体，无细节，对于某一个类型的 Coverage 需要做出全面的分析，还有一个就是这些不同类型的 coverage 会经常组合在一起来使用的。至于组合的策略，需要在实际项目过程才能体会。

原文链接：http://www.infoq.com/cn/articles/exploratory-testing-secret

相关内容：

- 阅读者（五）：测试驱动开发的艺术
- QCon北京 2011 大会精彩回顾和总结
- 颠覆者生存：互联网产品测试观点
- 窦涵之谈ATDD及CI的整合
- 基于静态依赖的测试

特别专题

探索式测试中的几种误区

作者 崔康

探索式测试（Exploratory Testing）是敏捷测试中的重要组成部分，其价值与一般性测试如用户故事测试或者自动化测试不同，它所关注的是“意料之外”的软件缺陷，探索式测试作为一个研究性、启发性和严肃性并存的测试方法，是一般性测试的重要补充。随着敏捷测试的推广，探索式测试逐渐受到大家的关注和重视。本文主要探讨了测试工程师在探索式测试方面的一些误区，并尝试纠正这些问题。

误区 1：探索式测试是一种测试技术。

探索式测试本身不是一种测试技术，相反，它是一种可以应用于广泛测试技术的方式或态度。所以很遗憾，测试工程师可能在现实世界中找不到一款名为“探索式测试工具”的软件来帮助他们直观地完成探索式测试。《测试计算机软件》作者 Cem Kaner 明确地将探索式测试定义为“一种强调每位测试人员自由和责任的测试，每个测试人员通过将学习、测试设计、测试执行和测试结果解释作为贯穿项目的平行活动，从而持续地优化他们的工作价值”。

例如，在 Web 应用项目中，UI 功能测试必不可少。探索式测试作为一种思想就可以而且应该贯穿于 UI 功能测试中。

具体来说，测试工程师在某个时刻准备按照事先编写好的测试用例开始测试某个网站用户注册的页面时，一步步的走下去，输入框、日期选择器……一切看起来都很顺利，但是等一下，探索式测试是一种启发式的方法，在循规蹈矩地做着一般性功能测试的时候，我们需要同时来点不一样的东西。

盯住用户注册界面！大脑或许可以这样开始思考：我看到光标在闪动，噢，我之前测试是用鼠标点击的，如果用户只用键盘操作，光标在各个输入域的遍历是顺序的吗？动手测试之！接下来，鼠标点击一下提交按钮，注册成功！噢，如果失败会是什么样子？用户需要重新填写所有数据吗？动手测试之！下一步，用户注册页面测试完毕，不过，它有帮助文档或者提示信息吗？作为测试工程师我们可能已经非常熟悉产品了，可第一次访问网站并准备注册信息的用户了解必要的内容吗？注册完的用户到底知不知道自己有什么权限呢？动手测试

之……

凡此种种，都是些启发式思维的小例子，从中可以看出，探索式测试作为一种方法，可以运用于任何一般性测试中，如单元测试、功能测试、性能测试、系统测试等等，只要有探索性的思想并贯彻于实践中，探索式测试就会发挥它的重要作用，找到一般性测试没有涵盖的危险区域。

误区 2：探索式测试是一种黑盒测试。

测试工程师和测试经理对黑盒测试并不陌生。黑盒测试指的是测试人员把待测产品看做一个包装完好的盒子，无法看到其中的实现细节，只通过产品规格说明书、用户反馈等途径来设计测试用例并执行、分析。探索式测试作为一种测试方法，在黑盒测试中确实可以得到??很好地应用，比如在上面提到的 UI 功能测试中，探索式测试可以作为一种有益的补充发现常规测试顾及不到的测试点。但是，探索式测试提倡的原则之一就是“努力深入了解待测产品”。常规测试用例覆盖范围不全面的一个重要原因就在于对产品的理解不够深刻，所以难以挖掘出深层次的问题。探索式测试更趋近于白盒测试，要求测试工程师对产品有比较透彻的理解，从而创造性地提出和执行一些“出乎意料”的测试。

举个例子，假设我们正在测试一款 Web 应用，部署在 Java 虚拟机为基础的 Web 服务器（以 WebSphere Application Server 为例）上。按照常规测试用例，我们可能测试了多用户并发访问的情况、集群环境的情况等等，看起来黑盒测试就 OK 了。

接下来，我们尝试从白盒测试的角度开始探索式测试。瞧瞧产品用例和源代码，噢，看到了“synchronized (……)”、“lock.unlock()”这样的语句，很自然地，我们意识到产品在使用多线程编程，这意味着什么呢？多核！哦，常规测试中产品是在单核还是多核环境中？单核上的性能怎么样？多核上的性能有提高吗？线性程度有多大？负载均衡吗？这些问题成为了探索式测试的猎物，设计测试用例，并动手执行之，然后分析结果。

接着看代码，看到了很多“new ……”，还看到了产品代码中预分配了一些内存池，很自然地想到内存管理，当然 Java 的内存管理都由 Java 虚拟机掌控，测试工程师还能做些什么？有的！如果仔细研究 Java 虚拟机的机制，就会发现初始堆大小、最大堆大小、垃圾回收日志是否输出、垃圾回收的不同算法等设置，默认情况下 Web 应用的运行情况如何？随着用户增加，堆会不会由于太小而导致运行失败？堆大小变化了，原来的垃圾回收算法是否能够保证 Web 应用的响应时间如之前快捷？这些问题又是探索式测试的启发式线索，我们据此可以设计用例、执行测试。

伴随着对产品的了解越来越深入，探索式测试会逐步发现更多的隐藏的潜在风险，通常情况

下在白盒状态下的探索式测试更具价值，因为其成果都是建立在坚实的知识和理解基础上，其指向更有针对性。

误区 3：探索式测试就是随机测试。

探索式测试与随机测试的关系有些微妙。随机测试一般是指??在无文档、无计划下的软件测试，通常能够在测试初期快速地发现重要的缺陷或者在测试末期快速地进行回归测试，在某些资料中，将随机测试作为探索式测试的一部分。从表面上看，“随机”和“探索”两个词之间的确存在着一些联系，但是细一想，探索并不意味着随机，虽然是启发式的思维方法，但是其仍然遵循着一定的有序过程，虽然这种过程无法用文字记录下来，也无法适用于所有测试场景，但是其思维过程仍然具有合理性、有序性。所以，探索式测试是更高级的随机测试，它借鉴了随机测试中自由、开放的特质，但是又融入了启发式思维的元素，较之更为严谨、正式。

下面我们来看一看 James Whittaker 博士在讨论软件故障模型（Software Fault Model）时，对用户界面??采用探索式测试提出几条的启发式建议：

1. 确保所有类型的输入错误信息都触发一次，努力找出开发人员可能会忽视的无效值。
2. 针对所有输入域，尝试错误类型的数值和可能会被错误对待和处理的字符串。研究产品运行的操作系统和编程语言的局限性和差异性，创建一些存在问题的字符串，并执行测试。
3. 在每个输入域，键入所允许的最长字符。
4. 选择一些能够引起底层数据运算的输入域，多次重复输入相同的值，观察结果是否相同。
5. 对每一个输出域，思考其在不同情况下出现的不同结果显示，然后将这些情况分别应用到测试中，观察输出域是否和预料相同。
6. 对每一个输出域，思考哪些结果是不能或者不应该出现的，然后想办法触发这些非法的结果，看看是否能够实现。
7. 尝试触发递归式的函数调用，如文档中嵌套自身文档，超链接中嵌套自身链接，类似的错误。
8. 尝试触发数据的存储溢出，观察由此导致的结果。

看到上面的建议，我们不会认为这是在做随机测试。针对用户界面的探索式测试显得很有“条理”，虽然随机测试也可能会涉及到上面提到的几条建议，但这是“随机”的，测试工程师

无法控制测试覆盖的范围。探索式测试则不同，它会存在文字记录，会做覆盖率分析，比随机测试更为有序和可控。

误区 4：探索式测试阶段在一般性测试之后。

这是一个普遍的误解，总有人认为探索式测试是一种后期测试，当一般性测试都结束之后，再搬出探索式测试来找找漏洞。这里再强调一遍，探索式测试是一种测试思想和方法，不是测试技术，它可以应用于广泛的一般性测试之中。正因为如此，在敏捷世界中，探索式测试和一般性测试所处的位置是一样的，没有前后之分，只要测试工程师需要探索式测试，那么它就可以存在。

以一个迭代周期为例，在计划阶段，测试工程师通过了解用户的需求和用户故事的制定和优先级评定来审视当前的测试计划和流程，挖掘现有计划中未覆盖的重要测试点，这也是探索式测试的组成部分。在正式开发阶段，测试工程师在测试过程中与开发人员不断的交流，从中寻找潜在的产品问题，有时候开发人员的一句提醒就可以成为测试的重要线索，而探索式测试就是要凸显这些发现问题的机会。在收尾阶段，测试工程师会总结此轮迭代的缺陷列表和分析，在回顾过程中可能发现其他的问题，为下一次的探索式测试做准备。

如果探索式测试位于迭代的后期，那么测试工程师就会错失计划、开发等阶段深入参与项目的机会，或者说即使深入参与了项目，但是如果没有探索式测试的意识贯穿于其中，那么发现问题的机会往往会从身边溜走，而这是测试工程师最忌讳的事情。所以，如果你需要探索式测试，那么就把这种思想应用于测试的各个阶段，尽可能最大化它的价值。

误区 5：探索式测试需要老手来做。

探索式测试看起来很神秘、很酷，所以大家可能有些望而生畏，觉得是一个很高深的东西，怀疑自己是否适合做探索式测试，导致的结果就是在一个测试团队里可能会把探索式测试重担交给富有经验的老手来做。这种做法正确吗？我们不妨先来分析一下新手与老手的主要特点。

新手缺少测试经验，没有经过系统的学习和思维训练， 喜欢按照指令来做事情，不过接受新事物能力强，偶尔会因为不懂状况而搞出一些“小意外”；老手则是久经沙场，对测试技术和方法有充分的理解，喜欢有所突破，不过惯性思维影响了接受新鲜事物的意识。这里所说的特点是笔者在日常工作中观察的结果。分析完特点，再回头看看探索式测试，它是一种启发式的思维方式，我们需要创新，需要跳出常规测试的条条框框，需要找到“意外”的问题。对照新手、老手的特点，探索式测试更适合谁呢？这不是一个是非问题，新手、老手各

有其优点：新手能够快速打开自己的创新思路，老手可以快速深入产品的内部机理。所以，不论是新手还是老手，都不要为自己贴上标签，探索式测试工程师需要一定的技能，具备了这些特质，你就可以出色地完成探索式测试，敏捷测试专家 Lisa Crispin 总结了必要的技能：

1. 小心的观察者：观察不正常和不期望的结果，并对正确性的假定很小心，能够细微的观察软件特征或模式。
2. 认真的思考者：在运行中检查测试并将其改到非预期的方向上，能够解释寻找缺陷的逻辑并提供清晰的测试状态。
3. 系统的叛逆者：思维严密、系统化，同时还要具有多样化的观点。
4. 资源的挖掘者：探索测试人员应该发掘更多他们可以使用的工具、技术、测试数据、朋友和信息源。

总结

探索式测试还处于摸索和发展阶段，有关对它的认识还在不断的演变当中，探索式测试作为一种测试方法，值得测试工程师应用于广泛的常规测试中，它需要建立在对产品深入了解的基础上，不是黑盒测试，它虽是启发式的思维方式，但是又具有条理性、可控性，探索式测试需要贯穿于整个测试周期中，以发挥其价值。探索式测试的适合人员不是以新手老手为衡量标准，必要的技能才是考量的要素。

相关内容：

- 我和敏捷团队的五个约定
- 通过测试驱动开发和结对编程提高生产水平
- 组织敏捷测试
- 自动化测试——敏捷测试的基石
- 敏捷测试工程师的十条法则

特别专题

测试工程师的学习之旅

作者 Lisa Crispin 译者 崔康

软件行业发展迅猛。越来越多的团队开始重视测试，他们利用测试驱动开发。全新的或者改进的自动化测试和驱动框架层出不穷。团队在采用更多自动化回归测试之 后，需要测试人员具有精湛的探索性测试技能。但是大部分人在校园里中学不到这些必要的技能，那么测试人员是如何炼成的呢？

同时，我发现一些人在努力寻找让自己满意的测试工作。测试人员经常问我如何融入敏捷开发，或者哪些技能可以帮助他们找到满意的工作。如果没有编程经验，他们会担心技术上无法立足于敏捷团队。我认为虽然技术很重要，但是态度决定一切。如果你乐于学习，并且努力帮助团队交付优秀的产品，那么作为测试人员，你前途一片光明。我的建议是抓住一切学习机会，主动获取新技能。

我发现许多人从例子中学到的东西比较多，所以这里分享本人的几个故事，讲述了我的学习动力是如何促进事业发展的，希望能够为读者自己的职业进步提供一些启发。

开发人员、测试人员还是领域专家？

测试人员的背景非常广泛。在过去十年间，随着越来越多的开发人员对测试产生兴趣，我见到许多开发人员更乐于把自己视为测试人员。还有许多测试人员来自于业务领域，他们的领域专长对于开发非常有价值。技术作者，必须弄清楚应用程序的行为才能正确表述，所以经常让自己变成测试人员。许多人都是碰巧担当了这个角色，我也是！

来说说我自己的故事吧。我的职业生涯起初是一名开发人员，而且我喜欢编程。测试自动化（本质上属于开发人员的任务）是我最喜欢的工作之一。我热爱测试。我乐于了解业务并想办法促其成功。拥有技术背景让我既适应开发团队又适应业务团队。下面的故事讲述了我的学习旅程：从早期的开发岁月到参与敏捷团队。

对测试的早期认识

和许多人一样，我是偶然进入了软件开发领域。我最初在 Texas 大学的 Austin 数据处理部门

获得了“开发实习生”的职位。

我的培训老师其实比我早几周入职，也是刚刚接受培训，他们刚学会了编程，然后又教会了我。很快，我就了解了 Easytrieve、Cobo 和 4GL 还有层次型数据库的基础知识。我们以相同的方式编写代码，所以彼此的程序易于操作。现在回想起来，集体性的代码所有权非常有用。

在这次培训的数月之后，我很高兴的接受了教育协调员的工作，不仅仅监督开发人员培训，还负责培训最终用户。我们通过课程教育老师们如何执行简单的查询和报告，这节省了开发人员大量的工作。我从这一年的经历（期间我还在日常开发工作）中学到了很多：如何教授他人。

我惊讶的发现从客户和其他开发人员身上受益良多。我们（开发人员和分析人员）与客户坐在一起，讨论他们的需求，并现场画出原型。我们一一展示直到他们确认需求。我曾经加入一个团队来规划图书馆的在线编目系统，与图书管理人员坐在一起了解卡片编目系统是如何运作的。学习不同的领域是我工作中最有趣的部分。我们对测试一无所知，但是与客户的合作帮助我们在发布产品之前提高了软件的质量。

在最初的开发/分析工作中，我学会了如何领导他人。我的老板曾经告诉我做领导意味着确保其他人知道我的团队所做的贡献。我学会了以身作则。在以后的工作中我一直谨记在心，想办法让老板和其他业务上的人员知道我的团队和我自己带来的价值。

在转变中学习

几年后，我在一家大型软件公司担任技术支持，那时对测试和质量保证的概念不太了解。同事和我出于自我防范的意识做了大量测试工作：在客户发现缺陷之前最好由我们自己来找到这些问题。某天，老板问：“谁想做 DB2 培训？”没人了解 DB2，但是我主动请缨。很快，我成为了团队中 SQL 和 DB2 专家。

公司发现在客户之前找到缺陷好处多多，所以决定创建第一个测试团队。我再次自愿参与。因为我了解 SQL，所以我测试了使用 Oracle 和 Sybase 数据库的项目，这些都比我们自己的数据库产品在市场上更受欢迎。

在新的工作中，我开始学习测试的方方面面。我参加了一次测试研讨会了解了更多知识。我们开始尝试测试自动化。我们的软件适应于所有操作系统，因此我有幸学习 VAX/VMS、Wang、OS2、AS400 和八种不同的 UNIX 系统。虽然这些经历写到简历上不是那么好看，但是在所有平台上维护测试环境是宝贵的经验。

我们的团队同时负责打包发布。我理解了发布说明和准确文档的重要性，以及如何管理 alpha 和 beta 测试。起初这些任务让人觉得很困难，即使现在我也觉得自动化测试是这样。但是

我很幸运的受到了来自外部课程、自学教材和同事等各方面的培训和支持。我试着克服困难，不断想办法掌握新技能。

随着在测试、自动化、数据库和操作系统等各方面的广泛经验，我拥有了不凡的技能。这本不是我的目标，起初我只是想学习新知识！不论是技术能力还是有关业务的什么东西，我喜欢在新领域中探索，很值得去做。当公司遇到财政危机时，我找到了一份不错的新工作。

个人关系创造机会

我的新工作很有趣，而且有机会学到新技能。例如，我成为了团队的 Powerbuilder 专家。我能够花费几个月时间来学习一种测试工具并搭建自动化 GUI 测试集。最重要的是，一些过去的同事也加入了这家新公司，让我领悟到——这世界真是小啊！

几年之后，在互联网热潮中，我加入了一家 web 创业公司。我对测试 web 应用一无所知，但是因为我曾经使用过多年各种测试自动化工具，所以我在因特网上想找到适合 web 应用的工具。

当我查看工具列表网站时，“OCLC”几个字母吸引了我的眼球。当我在参与在线图书馆编目项目时深入了解了 OCLC，因为 OCLC 一直被用于编目书籍和向图书馆提供服务。奇怪的是，它们在出售一款名为 WebArt 的测试工具，我决定购买。它的开发者 Tip House 过来培训我们如何测试 web 应用和自动化测试。

和许多测试人员一样，我总是在想如何更好地及时交付高质量的软件。互联网世界比数据库产品变化快得多，我对缓慢、瀑布型的过程感到沮丧。尝试一种不同方式的机会很快就出现了。当我们的创业小公司被一家大公司收购的时候，一些同事离职选择自主创业，他们给我一本名为《Extreme Programming Explained》的书，说：“我们准备尝试极限编程。”当我读了这本书，我觉得自己必须尝试一下，请求他们带上我。

第一次加入 XP 团队之后，我开始学习在 XP 模式下测试人员应该如何工作，并分享到在线敏捷社区中（虽然那时我们还不称之为“敏捷”）。我惊讶的发现 XP 专家和其他敏捷实践者非常受欢迎。当 Bob Martin 大叔过来培训我们时，他建议我给 Ward Cunningham 打电话请教测试中的问题，并提供了他的手机号。Ward 与我讨论了一个小时！如果我听说类似 Ron Jeffries 或者 Kent Beck 来访或者出席一个我参加的会议，我会想办法与他们见面，而他们总是很慷慨地花时间会解答问题。Brian Marick 帮助我创建了一个敏捷测试邮件组，使我受益良多。

贡献社区获得机会

当我的团队还有那些我通过会议、用户组和邮件列表结识的朋友都认识到敏捷测试技术的好处时，我决定不应该让其他的测试人员和团队继续重复昨天的故事。在 XP 社区的鼓励下，Tip House 和我合著了一本书《Testing Extreme Programming》。许多人帮忙审阅了草稿并反馈意见，包括 Janet Gregory。Janet 和我开始组织研讨会和教程。

极限编程的核心是人，而事业成功的要素也是。我运用了个人关系，并最终成为一名演讲者、教练和书籍作者。我不仅成为了一名更出色的测试人员，我还学会了沟通的最佳方式。我经常参加会议，向他人学习，并在研讨会和培训班中阐述自己的观点。这一切都因为我想学习，并花时间与朋友发展良好的工作关系。

我也体验了回报的乐趣和价值。我的第一个 XP 团队与其他组织发起了本地的 XP 用户组。我在第一次会议上做了演讲！过去十年间，我通过这个用户组见到了许多优秀的朋友并受益良多，而它只是占用了我的一些时间而已。我努力回报以前获得的所有帮助。我参加了本地用户组，志愿帮忙组织会议，维护着一个测试邮件列表，和其他公司组织一些短期的研讨会，和对测试和敏捷开发存在疑问的团队进行网络和电话会议。我发现帮助别人越多，自学的也就越多。这感觉太好了——回报他人就是帮助自己。

学无止境：开阔眼界

我已经从事软件测试这个职业许多年了，但是不觉得厌倦。我每天都在学习新东西：要么是技术，要么是业务运作上的发现。在同事或者用户组、会议甚至 Twitter 上的同行的协助下，我尝试了新的开源工具，并且学习了新的脚本语言。这可能很困难，但是值得付出。

例如，我努力学习 Ruby，因为我从没掌握过一种面向对象语言。我阅读相关书籍并从同事中获得帮助，通过 Ruby 编写的脚本让我有更多时间关注更有趣的测试。我参加了一些组织以改进测试工具，如关注测试自动化的 Austin Workshop 和敏捷联盟功能测试工具委员会。我不仅了解了更多工具，而且见到了许多能提供帮助的同行。

为什么如此重要？

我希望其他测试人员在看到本文时能够感受到我对自己工作的热爱（虽然有时会沮丧：希望自己掌握更多的技能！）来自早期技术支持团队的朋友惊讶于为何我找到一份新工作这么容易，而他们依然挣扎于讨厌的工作中。我不比他们任何人更聪明：我花时间学习并抓住新机会！对学习的时间投入和参与技术社区活动对我的职业发展产生了回报。

这就是我希望读者从我的学习之旅中得到的启示：对自己的职业发展负责。不要局限于技术

或者测试技能。了解公司业务领域使你能够帮助他们做出正确选择。现在，请走出你的封闭角落，想一想如何帮助团队和公司。加入一个在线的测试俱乐部，或者志愿帮助本地的测试用户组。买一本新书或者阅读在线的教程。今天就启程可以让你的学习之旅走得更远一些。你会更加喜欢自己的工作，你会获得更多机会，你会让我们所有人为你感到骄傲。

下面是一些测试人员学习的资源：

- **Collaboration Explained: Facilitation Skills for Software Project Leaders**, Jean Tabaka, Addison-Wesley 2006. These skills will serve you even if you aren't a 'project leader'.
- **Everyday Scripting with Ruby: For Teams, Testers and You**, Brian Marick, Pragmatic Bookshelf, 2007
- "Writing Maintainable Automated Tests", Dale Emery, 2009,
- Continuous Integration and Testing Conference
- "Google's '20 percent time' in action", Alex K.
- Exploratory testing articles and blog posts, Jonathan Kohl,
- Agile user groups
- Software Testing Club
- Weekend Testers
- Articles about agile testing

关于作者

Lisa Crispin与Janet Gregory合著了《Agile Testing: A Practical Guide for Testers and Agile Teams》，参与编写了《Beautiful Testing》，她在过去十年间在敏捷团队中担任测试人员，喜欢通过写作、演讲和参与敏捷测试社区来分享其经验。如果想了解更多有关Lisa的信息，请 访问 www.lisacrispin.com。

原文链接：http://www.infoq.com/cn/articles/testers-learning-journey

相关内容：

- Google的产品质量之道、云测试
- 阅读者（五）：测试驱动开发的艺术
- 开启淘宝精益测试
- 张银奎谈软件调试和软件测试

特别专题

虚拟座谈会：TDD 有多美？

作者 张凯峰

【编者按】最近酷壳的一篇 有关TDD的文章 引起了广泛关注，对于TDD一些人有自己不同的见解，为此InfoQ中文站特地邀请了InfoQ内外的敏捷专家特别是有丰富TDD实践经验的人，就TDD为InfoQ的读者分享他们自己的经验和体会。

InfoQ：请介绍你自己，以及 TDD 的实践经验。

> **熊节**：ThoughtWorks 中国公司首席咨询师，曾参与《重构：改善既有代码的设计（中文版）》、《J2EE 核心模式（原书第 2 版）》、《Contributing to Eclipse 中文版》等图书的翻译。目前正在从事 Ruby on Rails 的项目，并致力于敏捷方法与思想的推广。熊节从 2003 年开始实践 TDD，从此以后使用此方法编写每一段交付使用的代码，直至今天。

> **鲍央舟**：OutSofting 的敏捷咨询师。在从事咨询工作之前，从事软件工程师的工作。接触、使用过一些 TDD 的实践，但是由于大环境的原因，没有深入使用。在从事咨询工作之后，和一些有经验的 TDD 高手做过 Pair，也对 TDD 有了更深入的理解。

> **滕振宇**：独立敏捷教练，InfoQ 敏捷社区编辑，国内唯一的认证 Scrum 教练（Certified Scrum Coach）。从 2005 年接触到持续集成并开始实践，接下来开始接触并实践 TDD 以及 Scrum 等敏捷方法。

> **田乐**：无限讯奇高级开发工程师，工作有 7 年，是一个前端后端都做的程序员。从 07 年开始使用 TDD。

> **段念**：Google 中国高级测试经理，10 来年软件开发与测试经验，主要工作方向在软件测试。在某些项目中实践过 TDD，ATDD，在推动开发工程师的开发测试过程中对 TDD 有一些思考。

InfoQ：TDD 跟 Test 是什么关系呢？TDD 的 T 就是 Unit Test 吗？

熊节：可以是。也可以不是。关键在于，是不是都无所谓。我前两个星期就在干一个用 Cucumber+Selenium 直接写 spec 来驱动开发的项目，照样干得很清爽。

关键在于，TDD 的 T，是什么测试都无所谓。它就是设计。或者如果要把词汇定义得再准确点的话，你脑袋里的那个东西是“设计”，你在写生产代码之前写下来的那段东西是“设计的展现形式”，只不过它恰好是一个可以执行的 JUnit 或者 whatever 测试用例。

那么接下来应该问的是：什么是合理的设计的展现形式？ 答案是，它应该是无二义的、可执行的设计的验收条件。因此你不仅可以说“我有一个设计”，并且可以说“我把设计记录下来了”，并且可以说“我写的代码是符合设计的”。而这后面两件事（尤其是最后一件事），我没有看到其他任何一种设计方法可以有效地保障。

所以“关注测试而不是设计”这种说法，它不是对不对的问题，它是可笑的。当你说一种设计方法不关注设计，你到底是在说什么呢？或者你不认为它是一种设计方法？那么当你说你使用了一种设计方法但你并不认为它是一种设计方法，你到底是在说什么呢？

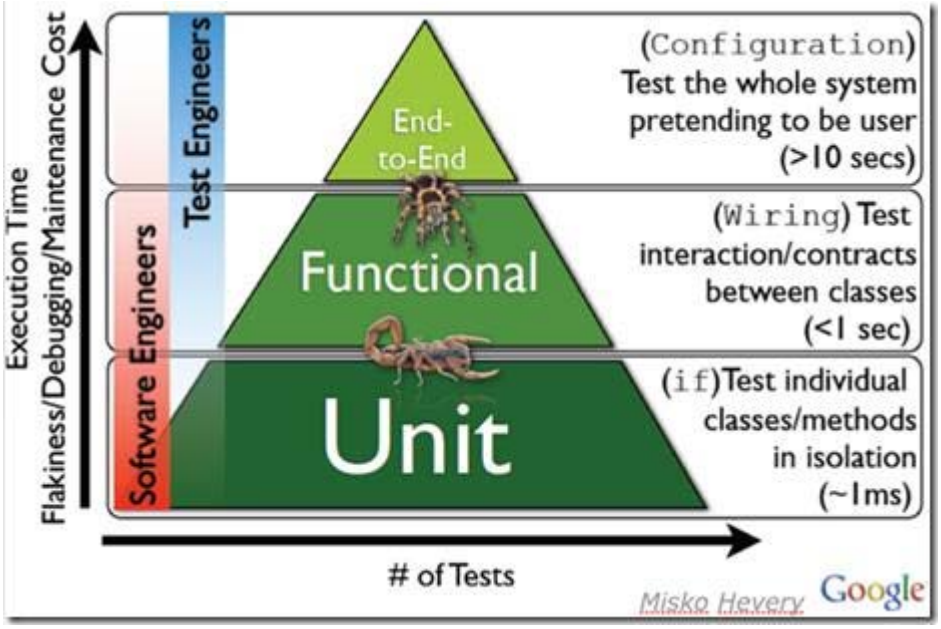
鲍央舟：TDD 更是一种工作方式，编码观念，而 Test 是这种观念中的一部分实践。具体说来，TDD 的观念是先明确下一步要做的一小样东西，然后恰到好处的实现要做的东西，最后审核所做的质量，以此循环。Test 是明确下一步东西后的产出，对实现的正确引导，也是审核将来代码质量的一个工具。按术语来说，TDD 的 T 确实是 unit test。Unit test 是跟代码质量，编码思路密切相关的。也有 ATDD，BDD 之类的，与代码的外部功能相关。

滕振宇：TDD 跟测试的关系：

- 测试是 TDD 的必然结果。如果团队一直在实践 TDD，所有的代码都会有相应的测试，所有的测试其实就是整个系统的脚手架。
- TDD 方式的开发是从写测试开始的。
- 使用 TDD 时，功能开发总是实现沟通结束条件，也就是在何种情况下，可以认为功能完成，这个结束条件是以测试体现的。
- 实践 TDD 时，写代码只有两种目的：1. 让一个失败的测试通过。2. 在不添加新功能（也就是不需要添加新的测试）的前提下，让代码、结构或者测试更加清晰、整洁、易懂。

狭义上 TDD 的测试指的是单元测试，但是随着敏捷开发方法的发展，TDD 又逐渐延伸发展出了 ATDD（Acceptance Test Driven Development）和 BDD（Behavior Driven Development）

等。每种方法关注于不同的问题。在这里我用 Google 的敏捷和 TDD 教练 Misko Hevery 基于 Mike Cohn 的测试金字塔延伸出来的金字塔来说明敏捷开发中各种测试的关系。



- 基于 UI 的测试，这一部分不可能做测试驱动，因为 UI 总是在最后完成的，不可能提前。
- 功能测试，这一部分是关注于系统的业务逻辑，通常是用 ATDD 和 BDD，是集成测试。这一部分保证系统做正确的事情(Do Right Things)。在迭带初期阶段，讨论用户故事的结束条件，用例子来体现，这些例子就形成了自动化的功能测试。
- 单元测试，这一部分保证系统的每个单元（类和方法级别）的逻辑正确，更关注于正确地做事情(Do Things Right)。

但是这些不同的测试之间也没有明确的界限，每个人有不同的理解或者实现方式，很多人也会先从简单的单元测试着手，然后逐步把单元测试重构组合成集成测试用例。

田乐：TDD 是从 Test 开始的，驱动开发过程的动力是失败的测试，Test 是驱动设计的工具。TDD 肯定需要有 Test，不过 Test 大部分的时候都不是为了 TDD 而写的。TDD 的 Test 不是 Unit Test，它可以是 Acceptance test（ATDD）、Functional test 等等。一般来说 TDD 可以穿透测试的几个水平分类。

Test Category 理论我记得以前 InfoQ 有写。单元测试对应的是集成测试。单元测试一般指对函数级别或者 OO 环境中的类级别的测试，特点就是没有外部依赖。集成测试是测试组件之间协作的测试，一般都会验证和用户需求有关的一些行为是否可以完成，也可以验证设计的组件协作是否可以完成。大型的系统还会把最顶端的集成测试叫做系统测

试。AT 或者 UAT 是从系统的外部行为验证软件的功能是否可以完成，它们在这种分层维度里面应该算系统测试，只是它要求这种系统测试是黑盒的。

功能测试是对应非功能测试的，就是是指验收的场景是否和 Use Case 有关。还有回归测试，它一般是用来验证曾经发现的缺陷的。

TDD 中的测试可以是上面所有分类中的任何一种。

段念：TDD 并不是石头里蹦出来的孙悟空，DBC（Design By Contract）可以看作是 TDD 的前身。在 DBC 的观点里，设计应该以规约（Contract）的形势体现，规约定义了被开发对象的行为。延续这个观点到 TDD，很容易就能理解，TDD 中的 T，在表现形式上是“测试”，但其实，它更应该被理解为“对被实现对象”的行为限定，也就是 DBC 中的规约。“测试”只是用来体现规约的形式。

单元测试通常被定义为“对应用最小组成单位的测试”，它的测试对象通常是函数或是类，在对类的设计和实现应用 TDD 时，为类建立的测试通常与类的单元测试相当类似，因此 TDD 中的 T 往往被误认为是单元测试本身。实际上，这两者还是有显著的区别的。首先，TDD 中的 T 描述的是规约，是设计的一部分；其次，TDD 中的 T 并不明确要求 T 对实现代码的覆盖率；第三，TDD 的 T 的侧重点是“描述被实现对象应该具有的行为”，而不仅仅是“验证该类的行为是否正确”。当然，TDD 中的 T 在形式上是测试，在重构中也可以作为被实现对象的行为验证框架。

单元测试、集成测试、系统测试、用户验收测试是基于传统软件开发过程的划分，在传统软件开发观点中，这几类测试不仅意味这测试对象的不同，同样也意味着不同的测试在开发周期中处于不同的位置。但在敏捷开发中，如果继续使用这几个名词，最多也只能保留它们在测试对象方面的含义。对于 TDD 来说（ATDD 和 BDD 可以认为是 TDD 的变体），在不同的测试类别中都可以应用之，唯一的区别在于 T 面向的对象不同。

InfoQ：你认为实施 TDD 需要怎样的前提条件？TDD 难在哪儿？

熊节：要使用一种设计方法，你就必须（1）会做设计；（2）做设计。它难在有些项目不做设计，有些人不会做设计。

鲍央舟：实施 TDD 的前提是对 TDD 实践背后的观念有所理解，也需要有经验者的指导。TDD 难在习惯和观念的转变。以前的工作方式已经在大脑和肌肉中固定下来，无法短期更改。另外，在短期可能呈现开发速度更慢的现象，需要管理层也对 TDD 以及质量有所理解，才能给予正确的支持。

滕振宇：前提条件很简单，就是了解这种工作方式（可以通过读书，可以通过跟教练一起结对等），然后去坚持。

TDD 很难：

- 关键是人的因素，人的个人修养。类似于健身，人人都知道经常锻炼的好处，但是真正没有几个人去坚持，这需要很高的纪律性。
- 要了解的东西太多，需要不断学习，如果不了解如何做好的设计，怎样去解耦等等，否则 TDD 会做得很痛苦。多数人却比较懒散。
- 大家往往认为老的系统代码太烂，耦合度太高，无从下手。

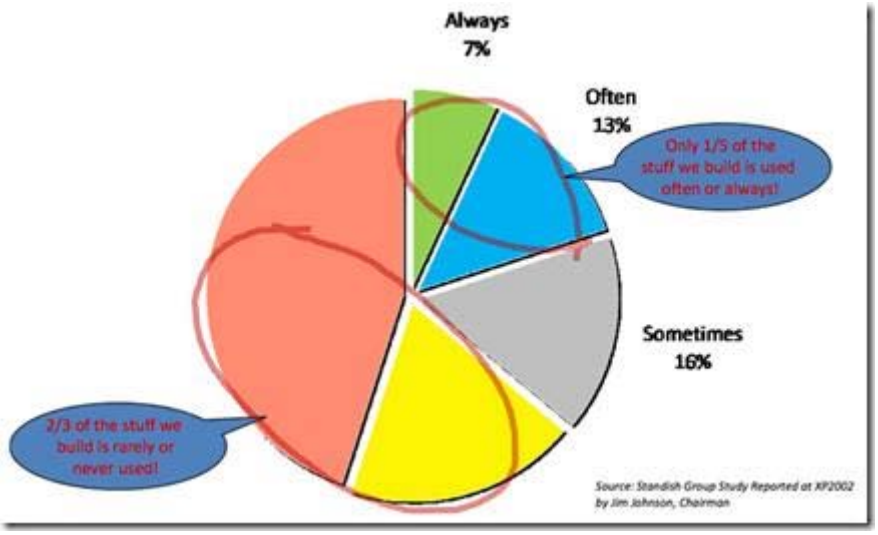
田乐：前提条件就是需求明确，知道需求的边界，了解如何可以验收。另一方面 TDD 要经过一些练习。我觉得 TDD 的一个难点就是把它当作一个推动设计的工具，而不是停留在保证质量（如检查边界条件）这个层面。另外，由于 TDD 一般是从最外面的抽象开始的，所以我个人觉得 TDD 最开始的抽象模型选择也是一个难点。

段念：实施 TDD 是对开发者行为的比较大的改变，在我的经历中，遇到的主要难点应该是改变既有的开发工程师的开发习惯吧。TDD 技术本身没有什么特别的要求，任何组织都可以直接应用。

InfoQ：TDD 之于需求、设计、代码质量是怎样的关系和影响？

鲍央舟：对于需求来说，TDD 更能引导开发人员做出真正符合需求的东西，不会过渡开发。对于设计来说，TDD 的实践能帮你清理思路，但不能教会你做好的设计。对于质量来说，TDD 保证所有的代码都有测试覆盖，肯定能提高质量。

滕振宇：需求方面根据 2002 年 Standish Group 的报告，我们软件系统中有 65%的功能是客户从来不用或者很少用到的。传统意义上大家认为敏捷开发应该让我们的团队开发得更快，生产率更高，这其实是很大的误解。与提高效率相比，使用 Scrum，TDD 能够帮我们从整个需求中确定真正有用的那 35%，而且往往这 35%的功能往往实现起来并不是那么困难。因为做得少，所以做得更快。



TDD与设计、代码质量方面，我想引用Keith Braithwaite在XPDay的分享话题“Measure for Measure”。他分析了很多的开源项目，发现有趣的现象，有自动化测试的那些项目，质量要好于没有TDD的项目（参见下图，所有有自动化测试的项目斜率(Slope)都是>2 的，想要了解斜率的具体含义，可以去听Keith的话题分享）

Codebase	Slope	Automated unit tests?
Jasml 0.1	0.95	N
Sunflow 0.06.3	1.59	N
m-e-c scehdule α3-10	1.69	N
NanoXML 2.2.1	1.77	N
Syncbuilder1999	1.84	N
Itext 1.4.8	1.88	N
Xcool 0.1	1.93	N
Ant 1.7.0	2.25	Y
Jfreechart 1.0.3	2.30	Y
MarsProject 2.79	2.33	Y
Log4j 1.2.14	2.43	Y
Junit 3.8.1	2.49	Y
Jmock 1.1.0	2.79	Y
Spring 2.0.1	2.78	Y

“**田乐**：TDD 会让你减少无用功，因为它迫使你从需求验收的角度入手，你必须在进入细节之前找到这个需求的边界，找到那些验收条件。TDD 会锻炼自顶向下的的抽象分解能力，这对仅习惯从细节入手的程序员来说很有帮助。TDD 可以提高你的代码的可测试性，它的节奏还可以帮助你做到简单设计。还有大家常说的，TDD 有助于提高代码的内部质

量。

段念：ATDD 可以向上直接追朔到需求，使用 ATDD 方式，可以避免功能镀金，这是 TDD 技术带来的一个大的好处。在设计方面，TDD 并不追求在最开始的时候得到一个完备的设计，而是遵循“够用的设计”的原则，保证开发者可以在短时间内得到可用的设计与实现。“够用的设计”是一个在敏捷环境下非常有效的原则，当然，也有些人反对这个原则，认为随意的设计无法随着应用的复杂性增加则很好的适应。在我看来，由于需求本身的不确定性，很难期望在一开始的时候就能给出保证能满足将来需求的设计，既然这样，不如遵循“够用的设计”的原则，通过重构等方式不断修正和优化设计。TDD 对代码质量本身没有明确的关注，但如果开发者自觉地不断应用重构技术消除代码中的 bad smell，在组织级别设计强制的 code review，以及对单元测试覆盖率给出明确要求，则可以在很大程度上让代码质量保持在高水平上。

InfoQ： 你认为实施 TDD 容易犯的错误是什么？TDD 的不足在哪些方面？

熊节：错误？陈皓同学已经向我们展示了。当你使用一把锤子时，你能犯的最大的错误就是尝试用它把钉子撬出来而不是砸进去。

不足？当你不知道该怎么用最理想的方式来描述你的设计，好吧，不管是因为什么原因，你当然只好退而求其次。你尽可以把设计的复杂和设计能力的欠缺归咎于你用的锤子。反正 TDD 又不会说话。不过，当你甚至不能为你的设计写出一个测试，你究竟打算用什么方式向别人讲述这个设计呢？或者也跟着退而求其次，反正我已经有设计了你就别管这个设计是什么反正我做出什么你就用什么吧——没错，很多人一直是这样工作的。

鲍央舟：过度编码！写完测试后，代码不止使新测试通过，还实现了很多别的东西。不足只是真正掌握比较难而已。

滕振宇：容易犯的错误：

- TDD 的名字取得不好，大家往往产生误解，认为只要先写测试，再写代码就是 TDD。在很多号称使用 TDD 的公司，这些测试甚至是测试人员写的（这不就是瀑布里面的 V 模型么？）
- 不重构，只要让测试用例通过就结束。对于一个 TDD 的实践者来说，他往往花很少的时间去实现功能，让用例通过。他会花绝大部分时间去重构，去让代码变得更加容易理解，设计更加清晰。

不足之处：

- 太需要实践者的坚持，“上士闻道，勤而行之，中士闻道，若即若离，下士闻道，大笑之，不笑不足以为道。”但是往往只有少数人会真正踏踏实实去实践。有些人三天打鱼两天上网，另外一些人会去嘲笑这种做法。
- 很难去向公司上层以及项目经理去证明其价值。

“**田乐**：我觉得 TDD 最容易出现的就是节奏问题，由于有的时候 coding 的非常尽兴，就没有遵循红绿红绿小步前进（baby step）的节奏，而是在没有失败的测试的时候就洋洋洒洒写下去。那样的做法其实就不完全是 TDD 了。TDD 的不足是它不是万能的，不应该是强制的。不是所有的任务都需要 TDD，那些临时的可抛弃的代码（如技术可行性试验）不需要 TDD。强制的 TDD 和强制的单元测试一样，因为设计优略不容易量化，TDD 也不能用 TestCase 的数量去量化，没法量化的实践不应该是强制的，否则会流于形式。TDD 无法量化也是它被大规模推广的时候的一个不足。

“**段念**：把 TDD 等同于单元测试，认为 TDD 只是“提前写单元测试”这种想法应该是很多不太了解 TDD 的人容易犯的错误吧。如果把 TDD 放到敏捷开发的大背景下，我倒不觉得 TDD 有什么明显的不足，但如果单独考量 TDD 在企业中的实践，TDD 技术本身不关注代码的质量应该是一个明显的问题。应用 TDD 的企业通常需要采用持续的 Code Review 和 Refactory 方法保证通过 TDD 产生的代码的质量。

InfoQ：一般开发者需要多久能掌握 TDD 呢？请向读者推荐一下 TDD 的学习资料吧。

“**熊节**：到他们掌握该怎么做设计时——which never happens to most people，请参阅《程序员的思维修炼》。 至于学习资料么，问豆瓣都比问我好。这事情也很悖论：能学会的人，读这些文字的价值约等于 0，因为他只需要豆瓣搜索 1 分钟+阅读 1 小时——这是他 anyway 会做的事——就能得到同样的信息；读了这些文字觉得很有用的人，有鉴于 Kent Beck 那本书出版已经 7 年了，基本上，他已经没希望了。

“**鲍央舟**：半年！网上一搜可以一大把资料，不过我觉得 TDD 光看是没用的，一定要和有经验的人 pair！

“**滕振宇**：多久才能掌握 TDD 呢？这不是一个“敏捷”的问题，因为每个人的学习能力，和对学习的投入程度是不一样的，因此学习的“速度”也不一样，因此很难说需要多场时间。在这里我采用敏捷计划与估计的做法，我们先确定一下范围，然后每个读者根据自己的学习速度，自然就可以算出需要多场时间。

做好TDD需要了解很多的技巧、工具、原则，我想用Ron Jeffries的“敏捷技能的七个支柱”来说明需要掌握的东西。在这七个支柱中

- 需要掌握“技术卓越”(Technical Excellence)
- “业务价值”中的“用户故事”及“递增迭带式发布”
- “自信”中的“持续集成”
- “产品”中的“易学易用”

所有这些背后都隐含着很多的学习要点，这些要点需要通过读很多书、资料、与专家交流才能掌握。

TDD 的学习资料，书：

- Kent Beck 的“测试驱动开发”
- Martin Fowler 的“重构”
- Bob 大叔的“敏捷软件开发原则、模式、实践”和“代码整洁之道”
- Michael Feathers 的“修改代码的艺术”
- Gerard Meszaros 的“XUnit 模式”
- Roy Osherove 的“The Art of Test Driven Development”
- Steve Freeman 的“Growing Object Oriented Software Guided by Tests”
- Eric Evans 的“领域驱动设计”等。

视频

- 到 Google 去查 Kata
- James Shore 的“Let’ s Play TDD”系列

> **田乐**：不知道多久可以掌握。学习资料的话 Kent Beck 的 TDD 就很好，还有我觉得 TDD 很适合与设计的方法论一起看 ,如领域模型驱动设计等(《测试驱动的面向对象软件开发》就结合了这两个知识)。

> **段念**：我们组织中有一些用于帮助开发工程师熟悉 TDD 的 program，根据我的了解，一般的开发工程师可以在 1-2 周内掌握 TDD 工作方式 ,但一般需要更长时间来达到对 TDD 的熟练掌握和灵活应用。

原文链接：http://www.infoq.com/cn/articles/virtual-panel-tdd

超百万大赛奖金 数亿元成长基金

开放平台 由你做主

直接收益

上线大搜索，传播大品牌

所开发的APP审核通过后，不仅在大赛官网展示传播，而且直接上线百度大搜索，即搜即展现，奠定人气与品牌基础。

特别收益

百万大赛奖金，超低门槛参与

丰厚的大赛奖金，应用审核上线即有资格参与，最终获奖者还将直接登上百度世界等影响力大会舞台。

长期收益

百度应用成长基金，长期稳定参与收益

应用审核上线后，即可长期参与百度应用成长基金收益，以及用户免费捐赠、应用增值性收入等收益支持。

增值收益

直接加入VIP俱乐部，典型案例重点传播

大赛期间，凡参赛应用审核上线的开发者（包括企业、工作室和个人等）均可自动加入开发者俱乐部并享受相关权益。

我要参赛 ▸

平台优势

海量用户 精准需求

门槛最低 合作最广

最佳保障 收益模式

品牌形象 立体传播

数据分享 技术支持

版权归属 绿色保护

Baidu 百度 | 2011 百度 WEB APP 应用创新大赛

2011 Web App应用创新大赛由百度主办，旨在激励开发者通过百度应用开放平台提交充满创造力的优质Web App应用，满足网民搜索应用的需求，让搜索更便捷、选择更丰富。还等什么？快来展现你的开发能力，百度开放平台，由你做主！

我要加入 ▸

推荐文章 | Articles

Java 深度历险（八）--Java I/O

作者 成富

在应用程序中，通常会涉及到两种类型的计算：CPU 计算和 I/O 计算。对于大多数应用来说，花费在等待 I/O 上的时间是占较大比重的。通常需要等待速度较 慢的磁盘或是网络连接完成 I/O 请求，才能继续后面的 CPU 计算任务。因此提高 I/O 操作的效率对应用的性能有较大的帮助。本文将介绍 Java 语言中与 I /O 操作相关的内容，包括基本的 Java I/O 和 Java NIO，着重于基本概念和最佳实践。

流

Java语言提供了多个层次不同的概念来对I/O操作进行抽象。Java I/O中最早的概念是流，包括输入流和输出流，早在JDK 1.0 中就存在了。简单的来说，流是一个连续的字节的序列。输入流是用来读取这个序列，而输出流则构建这个序列。InputStream和 OutputStream所操纵的基本单元就是字节。每次读取和写入单个字节或是字节数组。如果从字节的层次来处理数据类型的话，操作会非常繁琐。可以用更易使用的流实现来包装基本的字节流。如果想读取或输出Java的基本数据类型，可以使用 DataInputStream和 DataOutputStream。它们所提供的类似readFloat和writeDouble这样的方法，会让处理基本数据类型变得很简单。如果希望读取或写入的是Java中的对象的话，可以使用 ObjectInputStream和 ObjectOutputStream。它们与对象的 序列化 机制一起，可以实现Java对象状态的持久化和数据传递。基本流所提供的对于输入和输出的控制比较弱。InputStream只提供了顺序读取、跳过部分字节和标记/重置的支持，而OutputStream则只能顺序输出。

流的使用

由于I/O操作所对应的实体在系统中都是有限的资源，需要妥善的进行管理。每个打开的流都需要被正确的关闭以释放资源。所遵循的原则是谁打开谁释放。如果一个流只在某个方法体内使用，则通过finally语句或是JDK 7 中的 try-with-resources语 句来确保在方法返回之前，流被正确的关闭。如果一个方法只是作为流的使用者，就不需要考虑流的关闭问题。典型的

情况是在servlet实现中并不需要关闭 HttpServletResponse中的输出流。如果你的代码需要负责打开一个流，并且需要在不同的对象之间进行传递的话，可以考虑使用 Execute Around Method模式。如下面的代码所示：

```
public void use(StreamUser user) {
    InputStream input = null;
    try {
        input = open();
        user.use(input);
    } catch(IOException e) {
        user.onError(e);
    } finally {
        if (input != null) {
            try {
                input.close();
            } catch (IOException e) {
                user.onError(e);
            }
        }
    }
 }
```

如上述代码中所看到的一样，由专门的类负责流的打开和关闭。流的使用者 StreamUser 并不需要关心资源释放的细节，只需要对流进行操作即可。

在使用输入流的过程中，经常会遇到需要复用一个输入流的情况，即多次读取一个输入流中的内容。比如通过URL.openConnection方法打开了一 个远端站点连接的输入流，希望对其中的内容进行多次处理。这就需要把一个InputStream对象在多个对象中传递。为了保证每个使用流的对象都能获取 到正确的内容，需要对流进行一定的处理。通常有两种解决的办法，一种是利用InputStream的标记支持。如果一个流支持标记的话（通过 markSupported方法判断），就可以在流开始的地方通过 mark方法添加一个标记，当完成一次对流的使用之后，通过 reset方法就可以把流的读取位置重置到上次标记的位置，即流开始的地方。如此反复，就可以复用这个输入流。大部分输入流的实现是不支持标记的。可以通过 BufferedInputStream进行包装来支持标记。

```
private InputStream prepareStream(InputStream ins) {
    BufferedInputStream buffered = new BufferedInputStream(ins);
    buffered.mark(Integer.MAX_VALUE);
    return buffered;
}
private void resetStream(InputStream ins) throws IOException {
    ins.reset();
    ins.mark(Integer.MAX_VALUE);
```

```
}
```

如上面的代码所示，通过 prepareStream 方法可以用一个 BufferedInputStream 来包装基本的 InputStream。通过 mark 方法在流开始的时候添加一个标记，允许读入 Integer.MAX_VALUE 个字节。每次流使用完成之后，通过 resetStream 方法重置即可。

另外一种做法是把输入流的内容转换成字节数组，进而转换成输入流的另外一个实现 ByteArrayInputStream。 这样做的好处是使用字节数组作为参数传递的格式要比输入流简单很多，可以不需要考虑资源相关的问题。另外也可以尽早的关闭原始的输入流，而无需等待所有使 用流的操作完成。这两种做法的思路其实是相似的。BufferedInputStream在内部也创建了一个字节数组来保存从原始输入流中读入的内容。

```
private byte[] saveStream(InputStream input) throws IOException {
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    ReadableByteChannel readChannel = Channels.newChannel(input);
    ByteArrayOutputStream output = new ByteArrayOutputStream(32 * 1024);
    WritableByteChannel writeChannel = Channels.newChannel(output);
    while ((readChannel.read(buffer)) > 0 || buffer.position() != 0) {
        buffer.flip();
        writeChannel.write(buffer);
        buffer.compact();
    }
    return output.toByteArray();
}
```

上面的代码中 saveStream 方法把一个 InputStream 保存为字节数组。

缓冲区

由于流背后的数据有可能比较大，在实际的操作中，通常会使用缓冲区来提高性能。传统的缓冲区的实现是使用数组来完成。比如经典的从InputStream到OutputStream的复制的 实现 ，就是使用一个字节数组作为中间的缓冲区。NIO中引入的 Buffer类及其子类，可以很方便的用来创建各种基本数据类型的缓冲区。相对于数组而言，Buffer类及其子类提供了更加丰富的方法来对其中的数据进行操作。后面会提到的通道也使用Buffer类进行数据传递。

在Buffer上进行的元素添加和删除操作，都围绕 3 个属性 position、limit和 capacity展 开，分别表示Buffer当前的读写位置、可用的读写范围和容量限制。容量限制是在创建的时候指定的。Buffer提供的get/put方法都有相对和绝 对两种形式。相对读写时的位置是相对于position的值，而绝对读写则需要指定起始的序号。在使用Buffer的常见错误就是在读写操作时没有考虑到 这 3 个元素的值，因为大多数时候都是使用的是相对读写操作，而position的值可能早就发生了变化。一些应该注意的地方包括：将数据读入缓冲区之前， 需要调用 clear方法；

将缓冲区中的数据输出之前，需要调用 flip方法。

```
ByteBuffer buffer = ByteBuffer.allocate(32);
CharBuffer charBuffer = buffer.asCharBuffer();
String content = charBuffer.put("Hello ").put("World").flip().toString();
System.out.println(content);
```

上面的代码展示了 Buffer 子类的使用。首先可以在已有的 ByteBuffer 上面创建出其它数据类型的缓冲区视图，其次 Buffer 子类的很多方法是可以级联的，最后是要注意 flip 方法的使用。

字符与编码

在程序中，总是免不了与字符打交道，毕竟字符是用户直接可见的信息。而与字符处理直接相关的就是编码。相信不少人都曾经为了程序中的乱码问题而困扰。要弄清楚这个问题，就需要理解字符集和编码的概念。字符集，顾名思义，就是字符的集合。一个字符集中所包含的字符通常与地区和语言有关。字符集中的每个字符通常会有一个整数编码与其对应。常见的字符集有 ASCII、ISO-8859-1 和 Unicode 等。对于字符集中的每个字符，为了在计算机中表示，都需要转换某种字节的序列，即该字符的编码。同一个字符集可以有不同的编码方式。如果某种编码格式产生的字节序列，用另外一种编码格式来解码的话，就可能会得到错误的字符，从而产生乱码的情况。所以将一个字节序列转换成字符串的时候，需要知道正确的编码格式。

NIO中的 java.nio.charset包提供了与字符集相关的类，可以用来进行编码和解码。其中的 CharsetEncoder和 CharsetDecoder允 许对编码和解码过程进行精细的控制，如处理非法的输入以及字符集中无法识别的字符等。通过这两个类可以实现字符内容的过滤。比如应用程序在设计的时候就只 支持某种字符集，如果用户输入了其它字符集中的内容，在界面显示的时候就是乱码。对于这种情况，可以在解码的时候忽略掉无法识别的内容。

```
String input = "你123好";
Charset charset = Charset.forName("ISO-8859-1");
CharsetEncoder encoder = charset.newEncoder();
encoder.onUnmappableCharacter(CodingErrorAction.IGNORE);
CharsetDecoder decoder = charset.newDecoder();
CharBuffer buffer = CharBuffer.allocate(32);
buffer.put(input);
buffer.flip();
try {
    ByteBuffer byteBuffer = encoder.encode(buffer);
    CharBuffer cbuf = decoder.decode(byteBuffer);
    System.out.println(cbuf);  //输出123
} catch (CharacterCodingException e) {
```

```
    e.printStackTrace();
}
```

上面的代码中，通过使用 ISO-8859-1 字符集的编码和解码器，就可以过滤掉字符串中不在此字符集中的字符。

Java I/O在处理字节流字之外，还提供了处理字符流的类，即 Reader/Writer类 及其子类，它们所操纵的基本单位是char类型。在字节和字符之间的桥梁就是编码格式。通过编码器来完成这两者之间的转换。在创建 Reader/Writer子类实例的时候，总是应该使用两个参数的构造方法，即显式指定使用的字符集或编码解码器。如果不显式指定，使用的是JVM的默 认字符集，有可能在其它平台上产生错误。

通道

通道作为 NIO 中的核心概念，在设计上比之前的流要好不少。通道相关的很多实现都是接口而不是抽象类。通道本身的抽象层次也更加合理。通道表示的是对支持 I/O 操作的实体的一个连接。一旦通道被打开之后，就可以执行读取和写入操作，而不需要像流那样由输入流或输出流来分别进行处理。与流相比，通道的操作使用的是 Buffer 而不是数组，使用更加方便灵活。通道的引入提升了 I/O 操作的灵活性和性能，主要体现在文件操作和网络操作上。

文件通道

对文件操作方面，文件通道 FileChannel提供了与其它通道之间高效传输数据的能力，比传统的基于流和字节数组作为缓冲区的做法，要来得简单和快速。比如下面的把一个网页的内容保存到本地文件的实现。

```
FileOutputStream output = new FileOutputStream("baidu.txt");
FileChannel channel = output.getChannel();
URL url = new URL("http://www.baidu.com");
InputStream input = url.openStream();
ReadableByteChannel readChannel = Channels.newChannel(input);
channel.transferFrom(readChannel, 0, Integer.MAX_VALUE);
```

文件通道的另外一个功能是对文件的部分片段进行加锁。当在一个文件上的某个片段加上了排它锁之后，其它进程必须等待这个锁释放之后，才能访问该文件的这个片段。文件通道上的锁是由 JVM 所持有的，因此适合于与其它应用程序协同时使用。比如当多个应用程序共享某个配置文件的时候，如果 Java 程序需要更新此文件，则可以首先获取该文件上的一个排它锁，接着进行更新操作，再释放锁即可。这样可以保证文件更新过程中不会受到其它程序的影响。

另外一个在性能方面有很大提升的功能是 内存映射文件 的支持。通过FileChannel的 map方法可以创建出一个 MappedByteBuffer对象，对这个缓冲区的操作都会直接反映到文件内容上。这点尤其适合对大文件进行读写操作。

套接字通道

在套接字通道方面的改进是提供了对非阻塞I/O和多路复用I/O的支持。传统的流的I/O操作是阻塞式的。在进行I/O操作的时候，线程会处于阻塞状态等待操作完成。NIO中引入了非阻塞I/O的支持，不过只限于套接字I/O操作。所有继承自 SelectableChannel的通道类都可以通过 configureBlocking方法来设置是否采用非阻塞模式。在非阻塞模式下，程序可以在适当的时候查询是否有数据可供读取。一般是通过定期的轮询来实现的。

多路复用I/O是一种新的I/O编程模型。传统的套接字服务器的处理方式是对于每一个客户端套接字连接，都新创建一个线程来进行处理。创建线程是很耗时的 操作，而有的实现会采用线程池。不过一个请求一个线程的处理模型并不是很理想。原因在于耗费时间创建的线程，在大部分时间可能处于等待的状态。而多路复用 I/O的基本做法是由一个线程来管理多个套接字连接。该线程会负责根据连接的状态，来进行相应的处理。多路复用I/O依靠操作系统提供的 select或相似系统调用的支持，选择那些已经就绪的套接字连接来处理。可以把多个非阻塞I/O通道注册在某个 Selector上，并声明所感兴趣的操作类型。每次调用Selector的 select方法，就可以选择到某些感兴趣的操作已经就绪的通道的集合，从而可以进行相应的处理。如果要执行的处理比较复杂，可以把处理转发给其它的线程来执行。

下面是一个简单的使用多路复用 I/O 的服务器实现。当有客户端连接上的时候，服务器会返回一个 Hello World 作为响应。

```
private static class IOWorker implements Runnable {
    public void run() {
        try {
            Selector selector = Selector.open();
            ServerSocketChannel channel = ServerSocketChannel.open();
            channel.configureBlocking(false);
            ServerSocket socket = channel.socket();
            socket.bind(new InetSocketAddress("localhost", 10800));
            channel.register(selector, channel.validOps());
            while (true) {
                selector.select();
                Iterator iterator = selector.selectedKeys().iterator();
                while (iterator.hasNext()) {
                    SelectionKey key = iterator.next();
```

```
                    iterator.remove();
                    if (!key.isValid()) {
                        continue;
                    }
                    if (key.isAcceptable()) {
                        ServerSocketChannel    ssc    =    (ServerSocketChannel)
key.channel();
                        SocketChannel sc = ssc.accept();
                        sc.configureBlocking(false);
                        sc.register(selector, sc.validOps());
                    }
                    if (key.isWritable()) {
                        SocketChannel client = (SocketChannel) key.channel();
                        Charset charset = Charset.forName("UTF-8");
                        CharsetEncoder encoder = charset.newEncoder();
                        CharBuffer charBuffer = CharBuffer.allocate(32);
                        charBuffer.put("Hello World");
                        charBuffer.flip();
                        ByteBuffer content = encoder.encode(charBuffer);
                        client.write(content);
                        key.cancel();
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

上面的代码给出的只是非常简单的示例程序，只是展示了多路复用I/O的基本使用方式。在开发复杂网络应用程序的时候，使用一些Java NIO网络应用框架会让你事半功倍。目前来说最流行的两个框架是 Apache MINA和 Netty。在使用了Netty之后，Twitter的 搜索功能速度提升达到了 3 倍之多 。网络应用开发人员都可以使用这两个开源的优秀框架。

参考资料

- Java 6 I/O-related APIs & Developer Guides
- Top Ten New Things You Can Do with NIO
- Building Highly Scalable Servers with Java NIO

原文链接：http://www.infoq.com/cn/articles/cf-java-i-o

相关内容：System.IO的新特性和性能改进 、 快讯：淘宝自主研发文件系统TFS正式开源

推荐文章 | Articles

对象已死？

作者 徐昊

最近常有一种说法，就是我们如今面临着另外一场编程模型的变革，面向对象技术已经处在被淘汰的边缘，函数式语言会取代面向对象技术成为主流方式，甚至出现 了面向对象已死的言论。作为一个硬核函数语言的狂热者，我个人当然希望函数式语言可以一统天下，成为主流之选。但是不是应该把对象技术和函数技术对立起来，说式后者取前者而代之，我个人认为，这和如何看待面向对象技术有关。

做为工程实践的对象技术

在这个年代，大家有一种神圣化面向对象技术的倾向，很多人都把对象技术奉为高深的思想和理论。但实际上，面向对象技术仅仅一种工程实践而已，它是依托于其他技术而存在的一种实践，本身并不是一种完备的计算模型。

在计算机科学发展的早期，对于计算机的非数值计算应用的讨论，以及对于可计算性问题的研究和发展，大抵确立了几种的计算模型：递归函数类、图灵机、Lambda 演算、Horn 子句、Post 系统等等。其中递归函数类是可计算性问题的数学解释；Horn 子句是 prolog 这类逻辑语言的理论基础；lambda 演算成为了函数式语言的理论基础；图灵机是图灵解决可计算问题的时候所设计的装置，其后成为计算机的装置模型，与图灵机相关的自动机以及冯诺依曼结构，成为了命令式语言的理论基础。

因此当我们谈及函数语言和命令式语言优劣的时候，我们实际上是在讨论其背后的计算模型——也就是 lambda 演算和冯结构装置操作——在执行效率和抽象层次上的优劣。

而面向对象技术则比较尴尬了，其背后没有一个对应的计算模型（80 年代的时候曾有人研究过，Pi 演算是个备选，但是这个模型更多的是在并发对象领域的语义，而不是通常意义上的计算模型）。它有点类似于“最佳实践”，在不同的计算模型上有着完全不同实现方式和含义。因此对比对象技术和其他技术的时候，搞清楚到底是哪一种面向对象就变得格外重要起来。

两种不同的面向对象

目前流行的对象技术，实际上有两个截然不同的源头。它们分别在两个完全不同的计算模型上发展起来，但是都顶着“面向对象”这个帽子。

第一种对象技术出现的较晚，在 1979 年以后。它是以抽象数据类型（ADT，Abstract Data Type）为源起，发展出来的面向对象技术。也就是首先被C++所采用的面向对象技术。

C++作为“更好的 C”，继承了 C 语言对于程序的看法，也就是数据抽象（Data Abstraction）和过程。面向对象技术在 C++中，是作为一种更好的数据抽象的方式而存在的。

数据抽象在这类面向对象语言中是一种关键的抽象方式。所谓数据抽象，在计算机发展的早期是一种非常关键的技术。众所周知，计算机在装置模型上是一个存储和一组指令集，而二进制的存储实际上是没有任何类型表示的。整数，浮点这些操作必须通过相应的约定，再以指令集的形式进行支持。而随着计算机的发展，简单的数据类型显然已经不能满足应用的需要。这时候一种灵活且有效的类型系统，就成了一种自然的追求（直到 80 年代初，类型系统都是计算机科学研究的重要方向之一）。

在 C++中（以及后来的 Java 和 C#），对象是一种构造数据类型的方式，把每个“类”看作一段存储（状态）和操作（方法）的集合。“类”作为已经存在的类型系统的一种扩展（这一点在 C++中体现得尤其强烈）。在这类语言种，“类”（class）实际上代替了“对象”（object）成为了头等公民。构造一个更好的类型系统，是这种面向对象技术所要解决的问题。与其说是面向对象，不如说是面向类或面向类型的。

从计算语义上说，这类对象技术仍然是装置的操作语义，和面向过程的没有实质上的区别。唯一的不同是，被这种对象语言操作的机器，可以借由对象技术扩展机器所支持的类型。这种面向对象技术是过程技术的一种发展，虽然在抽象层次上没有什么太大的提高，但在实践上已经是巨大的进步。

另一种对象技术出现的很早，大概在 60 年代末就出现了，直到 80 年代初还有发展。但是很长一段时间内并不是太主流的做法，反而并不太为人所知。

在函数式语言里，因为高阶函数（High Order Function）的存在，数据可由函数来表达。这就是函数语言里一个非常重要的观点：Data as Procedure。在函数语言中，可以构造一种非常类似于对象的高阶函数：

```
(define (make-user name age sex)
  (define (dispatch message)
    (cond ((eq? message 'getName) name)
```

```
        ((eq? message 'getAge) age)
        ((eq? message 'getSex) sex))
        (else (error 'messageNotUnderstand))))
  dispatch)
(define vincent (make-user 'Vincent 30 'Male))
(vincent 'getName)
```

如上面的 Lisp 代码所示，可以借由返回一个 dispatch 函数，将基本数据组合成一个更复杂的数据对象，而通过高阶函数的后续调用，可以使用相应的选择器（selector）与数据对象交互。这种风格的数据抽象被称作“消息传递”（Message Passing），是早期面向对象技术的雏形，无论是 Smalltalk 还是 CLOS 都是以这种技术为蓝本，设计的对象系统，包括后来的 Ruby，实际上也是这种模型的一个发展。

因此实际上，就算在函数式语言上面，我们仍然可以通过引入这种对象的形式，对函数进行相应的模块化和局部化。这种形式的对象与函数本身没有任何差别，因此这种类型的对象系统，被称作“方便的接口”，用于简化对象的函数的访问和调用。

在函数式语言里，另一个非常重要的概念就是“副作用”（Side effect，即函数可以修改某个存在的状态）。像 Lisp 并不是纯函数语言，因此是允许状态修改的。因此对象技术除了可以被看作函数局部化和模块化的方法之外，还可以看作副作用局部化的一种方式。采用这类面向对象技术的语言，通常被称作动态面向对象语言。

这类对象语言通常都会保持一些函数式语言的特性，比如 lambda 的各种变体，比如较容易的函数组合，比如 curry，比如高阶函数。而且由于这类对象系统是从函数式发展出来的，也更加推崇一些副作用小的，利用高阶函数的对象设计方法。比如，不变体（Immutable object）回调等等。

计算语义上，无副作用的对象系统实际上和 Lambda 演算享有同样的计算语义。而带副作用的本身只能被看作一种坏的实现，在函数上都没有明确语义。仅仅能够看作对于副作用的局部化和模块化。

以上，我们简单地看了一下两种不同的“面向对象”技术。其中一种是用来解决如何构造更好的类型系统的，另一种是用来对函数和副作用进行有效模块化和局部化的。如果单以这两种面向对象技术和函数式语言去比较，实在不是一个层次的东西。那么为什么我们最近能够听到这么多函数和对象的讨论呢？

新的发展

静态类型函数语言

最早的函数语言是不太在意类型的，因为有 Data as Procedure 的存在，lambda 演算可以通过把参数类型抽象成另一个高阶函数来绕过函数参数类型问题（把参数也变成 lambda，每个函数都看作参数和函数体的高阶）。然而随着形式化类型系统在理论上的发展，把 lambda 演算扩展为 typed lambda 演算自然就是一种很自然的推论。

随着在此基础上发展出来的 ML 族和 Haskell 语言的日渐成熟，以及代数数据类型（algebraic data type）的引入，这些语言可以较为容易地构造出非常复杂的类型系统。而且伴随着类型推演和类型计算的引入，类型间复杂的关系也可以较为容易表达。由此，静态类型函数式语言也开始挑战以对象为基础的类型系统构造方法。

实际上这里函数语言的挑战是类型系统之争，而非面向对象和函数语言之争。因此，消息传递类的对象语言根本不在讨论之列，而对于静态类型面向对象语言而言，除了 C++外（而对于 C++，面向对象仅仅是构造类型系统的一种方式，另一种则是著名的范型编程。我仍然相信，在语义上静态类型函数语言会胜过 C++很多，但是弹性和表现力 C++并不会差太多），其他主流语言如 Java 和 C#，类型系统的已经被限制在一个相对简单的范畴内，说完败也不为过。

主流平台也为需要处理复杂类型系统的开发者提供了不同的选择，比如.NET 平台上的 F#。以及 JVM 上的 Scala。都是在主流平台上引入静态类型函数语言的一些特征，来简化复杂类型系统的构造。

并发编程 / 并行计算 / 多核编程

Lisp 并不是一个纯函数语言，它允许有副作用存在。后来发展了一些严格的纯函数语言，严格禁止副作用。也就是所有变量都和数学中的变量具有相同的语义，不能修改。然而计算机程序终归是要处理状态变化、输入输出这些不具有函数语义的操作的。一些纯函数语言开始引入了更精巧的方式来管理状态，比如 Monad。Monad 的传递性使得副作用的扩散在函数中变得更明确可见。

这种方式本来是用来解决纯函数语言内副作用处理的一种技巧，但是恰好赶上 Intel 受制于生产技术，无法再通过提高单核频率以追赶摩尔定律，必须通过集成多核的方式来制造更快的 CPU。多核 CPU 作为一种新的事物，给计算机界带来了新的恐慌，大家觉得有必要使用一种新的编程模型以充分利用多核的优势。

而第一个尝试的方案就是将计算分布到多个 CPU 上，也就是利用多核进行并行计算。这时候，纯函数式语言对于副作用的处理，恰好给多核编译器提供了一个理想的优化方式：即所有无作用的函数皆可以随意分布到多核上，而带副作用的函数则无法分布。通过对于类型系

统的简单识别和标注，就可以自动地将纯函数式程序编译为支持多核的程序。这在一段时间内，形成一种函数式语言是自动适应多核的，而面向对象程序则需要重写的印象。一时间内，函数与对象之间的选择实际上变成了多核和单核的选择。

好在还有 Amdahl's law存 在，事实也证明除去一些特定的应用场景，自动编译为支持多核并行的函数式程序并不快多少，而转化为纯函数程序的成本却高出不少，同时大多数纯函数语言都带 有学术性质，对于团队开发并不友好。在加上JVM和.NET CLR对于多核都做出了一些回应。因此除去一些计算密集型应用，纯函数语言并没比面向对象好多少。

峰回路转的是，由消息传递风格发展出来的 actor 模型，利用操作系统的进程/线程特性，在一个合理的粒度上很好地利用了多核的能力，简化了并发编程。虽然第一个著名的实现是 Erlang 的 actor 系统，但是由于消息传递风格和面向对象模型相去不远，很快就在各种面向对象语言中有了类库支持。虽然利用当代函数语言的语法特性，actor 可以实现得更简洁，但是对象对于副作用和状态的封装，更好地解决了在并发环境下对于共享状态的操作，反而有了更好的发展。

以上，我们看了函数式语言中两个新的发展，以及围绕这些发展涉及的一些“对象 v.s.函数”的讨论。正如本文一开始所说，对象技术作为一种工程实践，其发展总是依托于其他更基本的计算模型的演化的。函数语言的发展，使得我们对于对象的认识和理解有了更深更好的认识。而对象作为函数的“方便的接口”总会在新的发展中，让我们更加便利的享有函数式和其他计算模型发展的成果。

回到本文最开始的讨论，函数的发展会的确会促使一些对象技术的消亡，但也会产生新的对象技术。或许更好的理解和掌握函数，类型系统才是真正掌握对象技术的捷径，也未可知。

关于作者

徐昊，ThoughtWorks 中国区首席技术专家，ThoughtWorks 全球技术策略顾问（TAB），TW 中国首席咨询师。BJUG（Beijing Java User Group）和 AgileChina 创始人。从 2003 年起开始实践极限编程等敏捷方法，2005 年开始，多次以敏捷教练的角色帮助国内外多个团队实施极限编程，Scrum 和 FDD 等敏捷方法，敏捷交付和敏捷项目管理经验极为丰富。目前主要致力于大规模团队（300-500 人）内的敏捷实践和管理再造，以及对企业级技术应用趋势和技术战略的研究。

原文链接：http://www.infoq.com/cn/articles/object-have-dead

相关内容：面向对象编程——走错了路？、F#中的并行和异步编程

推荐文章 | Articles

正则表达式（五）：浅谈两种匹配操作

作者 余晟

在正则表达式中，匹配是最最基本的操作。使用正则表达式，换种说法就是“用正则表达式去匹配文本”。但这只是广义的“匹配”，细说起来，广义的“匹配”又可以分为两类：提取和验证。所以，本篇文章就来专门讲讲提取和验证。

提取

提取可以理解为“用正则表达式遍历整个字符串，找出能够匹配的文本”，它主要用来提取需要的数据，常见的任务有：找出文本中的电子邮件地址，找出 HTML 代码中的图片地址、超链接地址……提取数据时，首先要注意的，就是准确性。

准确

准确性分为两方面：完整和精确。前者是要提取出需要的所有文本，不能漏过；后者是要保证提取的结果中没有不需要的文本，不可出错。

为保证完整，我们需要考虑足够多的变体，覆盖所有情况。一般来说，要提取的数据都只有概念的描述（比如，提取一个电子邮件地址，提取一个身份证号），如果没有拿到完整规范的特征描述，可能只能凭经验总结出几条特征，然后逐步完善，也就是不断考虑新的情况，照顾到各种情况。

拿“提取文本中的浮点数字符串”为例。最容易想到的情况，就是 3.14、3999.2、0.36 之类，也就是“数字字符串 + 小数点 + 数字字符串”，所以用表达式『\d+\.\d+』，按照我们上一篇文章说过的“与或非”，三个部分都是必须出现的，所以这个表达式似乎是没问题了。

```
\d+\.\d+
```

但是有些时候，0.7 是写作.7 的，上面的表达式无法照顾这种情况，所以必须修改表达式：整数部分是可能出现也可能不出现的，所以小数点之前的\d+应该改为\d*，就成了『\d*\.\d+』。

```
\d*\.\d+
```

但是且慢，浮点数还包括负数，比如-0.7，但现在这个表达式无法匹配最开始的符号，所以还应该改成『-?\d*\.\d+』。

```
-?\d*\.\d+
```

但仅仅保证完整性还不够，提取的另一方面是精确，就是排除掉那些“能够由正则表达式匹配，但其实并非期望”的字符串，所以我们还需要仔细观察目前的正则表达式，适当添加限制条件。

仍然用上面的正则表达式作例子，『-?\d*\.\d+』中，『-?』和『\d*』都是可能出现的元素，所以它们可能都不出现，这时候表达式能匹配.7 之类，没有错；如果只出现了『\d*』能匹配的文本，可以匹配 3.14 之类，也没有错；但是，如果只出现『-?』呢？-.7，通常来说，负的浮点数是应该写作-0.7 的，而-.7 显然是不合法的。所以，这个表达式应该修改为『(-?\d+|\d*)\.\d+』。

```
(-?\d+|\d*)\.\d+
```

事情到这里就完整了吗？似乎还不是。我们知道有些地方，日期字符串是“2010.12.22”的形式，如果你要处理的文本中不包含这种日期字符串还好，否则，上面的表达式会错误匹配 2010.12.22 或者 2010.12.22。为了避免这种情况，我们需要给表达式加上更多的限制。最直接想法就是，限定表达式两端不能出现点号.，变成『(?!<.)(-?\d+|\d*)\.\d+(?!.)』。

```
(?!<.)(-?\d+|\d*)\.\d+(?!.)
```

这样确实避免了 2010.12.22 的错误匹配，但它也造成了新的问题，比如“...the value of π is 3.14. Therefore...”，3.14 本来是我们需要提取的浮点数，但加上这个限制之后，因为 3.14 之后的有一个作为英文句号使用的点号，所以 3.14 无法匹配。仔细观察我们要排除的 2010.12.22 这类字符串，我们发现点号.的另一端仍然是数字，而用作句号的点号，另一端必定不是数字（一般是空白字符，或者就是字符串的开头/末尾），所以应当把限制条件表达的更精确些，变为『(?!<\d.)(-?\d+|\d*)\.\d+(?!.\d)』。

```
(?!<\d.)(-?\d+|\d*)\.\d+(?!.\d)
```

好了，关于浮点数的匹配就讲到这里。回过头想想得到最后的这个表达式，我们发现，如果要用正则表达式匹配，必须兼顾完整和精确，通常的做法就像这个例子中的一样：先逐步放宽限制，保证完整；再添加若干限制，保证精确。

\d+\.\d+ \d*\.\d+ -?\d+\.\d+ (-?\d+|\d*)\.\d+	逐步放宽条件，保证完整
(?!<.)(-?\d+|\d*)\.\d+(?!.) (?!<\d.)(-?\d+|\d*)\.\d+(?!.\d)	逐步收拢条件，保证准确

效率

提取数据时还有一点需要注意，就是效率。有时要处理的文本非常长，即便进行简单的字符串查找都很费力，更不用说可能出现各种变体的正则表达式了。这时候就 应当尽量减少“变化”的范围。比如知道文本中只包含一个双引号字符串，希望将它提取出来，正则表达式写成了『".*"』。在文本不长时这样还可以接受，如 果文本很长，『.*』这类子表达式就会导致大量的回溯，因为『.*』的匹配过程是这样的：

```
正则表达式          字符串                .*匹配的字符串
 ".*"            "quoted string"
 ^               ^
 ".*"            "quoted string"
  ^               ^
 ".*"            "quoted string"        q
   ^               ^
 ".*"            "quoted string"        qu
   ^                 ^
  …                   …                  …
 ".*"            "quoted string"        quoted string"
   ^                              ^
 ".*"            "quoted string"        quoted string
    ^                            ^
 ".*"            "quoted string"        quoted string
     ^                            ^
```

观察匹配过程就会发现，如果字符串很长，而引号字符串又出现在比较靠前的位置，比如 "quoted string" and long long long text…，匹配时就需要进行大量的回溯操作，严重影响效率。如果这种问题并不是任何情况下都可能发生，但效率确实非常重要的，如果正则表达式编写不当， 可以产生极为严重的影响，比如ReDos（正则表达式拒绝服务），具体情况可以参考 http://en.wikipedia.org/wiki/ReDoS。

另一方面，正则表达式提取的效率，不仅与正则表达式本身有关，也与调用的 API 有关。如

果文本很大，要提取出的结果很多，集中到一次操作进行，就可能影响 性能，所以条件容许（比如只需要逐步提取出来，依次处理），就可以“逐步进行”，下面的表格列出了常用语言中的提取操作。

语言	方法	备注
Java	Matcher.find()	只能逐步进行
PHP	preg_match(regex, string, result)	逐步进行
	preg_match_all(regex, string, result)	一次性进行
.NET	Regex.match(string)	逐次进行
	Regex.matches(string, regex)	一次性进行
Python	re.find(regex, string)	逐步进行
	re.finditer(regex, string)	逐步进行
	re.findall(regex, string)	一次性进行
Ruby	Regexp.match(text)	只能找到第一次匹配
	string.index(Regexp, int)	逐步进行
	string.scan(Regexp)	一次性进行
JavaScript	RegExp.exec(string)	一次性进行
	string.match(RegExp)	一次性进行

一次性提取所有匹配结果的操作这里不多说，我们要补充讲解的是，在“逐步进行”时，如何真正保证“逐步”？或者说，在第二次调用匹配时，如何保证是“承接”第一次调用，找

到下一个匹配结果。通常的做法有几种，以下分别介绍。例子统一使用字符串为"123 45 6"，查找其中的数字字符串，依次输出 123、45、6。

如果采用的是面向对象式处理，表示匹配结果的对象，可能可以“记住”匹配的位置，下次调用时自动“继续”，Java 就是这样，循环调用 Matcher.find()方法，就可以逐个获得所有匹配，在.NET 中，是循环调用 Match.NextMatch()。

代码（以 Java 为例）

```
String str = "123 45 6";
Pattern p = Pattern.compile("\\d+");
Matcher m = p.matcher(str);
while (m.find()) {
    System.out.println(m.group());
}
```

如果不是面向对象式处理，无法记录匹配的状态信息，则可以手动指定偏移值。多数语言都有办法在匹配时指定偏移值，也就是“从字符串的 offset 位置开始尝试匹配”。如果要逐一获得所有匹配，每次将偏移值指定为上一次匹配的结束位置即可。注意，字符串处理时可能有人习惯将偏移值指定为“上一次匹配的起始位置+1”，但正则表达式处理时这样是不对的，比如正则表达式是『\d+』，而字符串是"123 45 6"，第一次匹配的结果是 123，如果把偏移值设定为“上一次匹配的起始位置+1”，之后的匹配结果就是 23，3……。在 PHP、JavaScript、Ruby 中，通常采用这种办法。

代码（以 PHP 为例）

```
$string="123 45 6";
$regex="/\\d+/";
$matched = 1;
$oneMatch=array();
$lastOffset = 0;
$matched = preg_match($regex, $string, $oneMatch, PREG_OFFSET_CAPTURE,
$lastOffset);
while ($matched == 1) {
    $lastOffset = $oneMatch[0][1] + strlen($oneMatch[0][0]);
    echo $oneMatch[0][0]."<br />";
    $matched = preg_match($regex, $string, $oneMatch, PREG_OFFSET_CAPTURE,
$lastOffset);
}
```

第 3 种办法是使用迭代器，Python 的 re.finditer()会得到一个迭代器，每次调用 next()，就会获得下一次匹配的结果。这种办法目前只有 Python 提供，其它语言尚不具备。

代码（以 Python 为例）

```
for match in re.finditer("\\d+", "123 45 6")
print match.group(0)
```

验证

另一类“匹配”是数据验证，也就是“检查字符串能否完全由正则表达式匹配”，它主要用来测试和保证数据的合法性。比如有些网站要求你设定密码，密码只能由数字或小写字母构成，长度在 6 到 12 个字符之间，如果输入的密码不符合条件，则会提示你修改，这个任务，一般使用 JavaScript 的正则表达式来完成。

初看起来，这也是用正则表达式在字符串中查找匹配文本。但仔细想想，两者又不一样：一般来说，提取时正则表达式匹配的开始/结束位置都是不确定的，需要逐次试错，才能决定；验证时，同样需要考虑准确性，但效率并不是重点考虑的因素（一把验证的文本是用户名、手机号、密码之类，不会太长），虽然也要求准确性，但匹配的开始/结束位置都是确定的，只要从文本的开头验证即可，不用反复推进-尝试；而且只要发现任何一个“硬性”条件无法满足（比如长度、锚点），即可失败退出。

正因为验证操作有这些特点，有些语言中提供了专门的方法进行正则表达式验证。如果没有，我们也可以使用简单的查找功能，只是在正则表达式的首尾加上匹配字符串起始/结束位置的锚点来定位，这样既保证表达式匹配的是整个字符串，也可以在无法匹配时尽早判断失败退出。

常见语言中的验证方法

语言	验证方法	备注
Java	String.matches(regex)	专用于验证，返回 boolean 值，不需要『^』和『$』
PHP	preg_match(regex, string) != 0	preg_match 返回匹配成功的次数，需要『^』和『$』
.NET	Regex.IsMatch(string, regex)	专用于验证，返回 boolean 值，不需要『^』和『$』

Python	re.search(regex, string) != None	成功则返回 True，否则返回 False，需要『^』和『$』
	re.match(regex, string) != None	成功则返回 True，否则返回 False，需要『$』
Ruby	Regexp.match(text) != nil	Regexp.match(text)返回匹配成功的起始位置，若无法匹配则返回 nil，需要『^』和『$』
JavaScript	Regexp.test(string)	专用于验证 ,返回 boolean 值, 需要『^』和『$』

前面说过，在验证时，文本的开始/结束位置是预先知道的，所以验证的表达式编写起来更加简单。比如之前匹配浮点数的表达式，我们首先得到的是『(-?\d+|\d*)\.\d+』，在进行数据提取时，需要在两端加上环视，防止错误匹配其它字符；但是如果是验证浮点数，就不需要考虑两端的环视，应该/不应该出现什么字符，直接在首尾加上『^』和『$』即可，所以验证用的表达式是『^(-?\d+|\d*)\.\d+$』。

我们甚至可以简单将各个条件叠加起来，直接得到最后的表达式，比如下面这个例子：

需要验证密码字符串，前期的分析总结出 5 条明确的规则：

1. 密码的长度在 6-12 个字符之间
2. 只能由小写字母、阿拉伯数字、横线组成
3. 开头和结尾不能是横线
4. 不能全部是数字
5. 不容许有连续（2 个及以上）的横线

下面依次列出对应 5 条规则的表达式：

1. 密码长度在 6-12 个字符之间：其形式类似『.{6, 12}』
2. 只能由小写字母、阿拉伯数字、横线组成：所有的字符都只能由『[0-9A-Za-z-]』匹配
3. 开头和结尾不能是横线：开头『^(?!-)』，结尾『(?<!-)$』
4. 不能全部是数字，也就是说必须出现一个『[^0-9]』或者『\D』
5. 不容许有连续（2 个及以上）的横线，也就是说不能出现『--』

如果用来提取数据，就必须把这 5 条规则糅合到一起。前 3 条规则比较好办，可以合并为『^(?!-)[0-9A-Za-z-]{6,12}(?<!-)$』，但它与第 4 和第 5 个条件合并都不简单。

与第 4 条规则合并的难点在于，我们无法确定这个『[^0-9]』出现的位置，如果简单改为『^(?!-)[0-9A-Za-z-]{6,12}[^0-9][0-9A-Za-z-]{6,12}(?<!-)$』，看似正确，却无法保证整个字符串的长度在 6-12 之间——目前这个表达式的长度在 13（6+1+6）到 25（12+1+12）之间。这显然有问题，但照这个方式也确实无法保证整个字符串的长度，因为我们无法跨越『[^0-9]』，为两端『[0-9A-Za-z-]』的量词建立关联，让它们的和为 5-11 之间。同样，与第 5 条规则的合并也存在这类问题，因为我们无法确认『--』的出现位置。

看起来，把这 5 条规则糅合成一个正则表达式，找到能够匹配的文本，真不是件容易的事情。不过，如果我们要做的只是验证，不妨换个思路：我们要匹配的并不是所有的文本，而是文本的开始位置，它后面的文本满足 5 个条件，而每个条件都可以不用实际匹配任何文本，而用环视来满足。

对应 5 条规则的环视表达式依次是：

1. 密码长度在 6-12 个字符之间：『^(?=.{6, 12}$)』
2. 只能由小写字母、阿拉伯数字、横线组成：『^(?=[0-9A-Za-z-]*$)』
3. 开头和结尾不能是横线：『^(?!-).*(?<!-)$』
4. 不能全部是数字：『^(?=.*[^0-9])』（这里不需要出现$，只要出现了非数字字符就可以）
5. 不容许有连续（2 个及以上）的横线：『^(?!.*--)』

下面就是寻找这样一个文本起始位置，它后面的文本同时满足这 5 个条件。实际上，因为锚点并不真正匹配文本，所以多个锚点可以重叠在一起，因此我们完全可以寻找 5 个锚点，把它们串联起来：

『(^(?=.{6, 12}$))(^(?=[0-9A-Za-z-]*$))(^((?!-).*(?<!-)$))(^(?=.*[^0-9])(^(?!.*--))』

意思就是：先寻找这样一个字符串起始位置，它之后的字符串满足条件 1；然后寻找这样一个字符串其实位置，它之后的字符串满足条件 2；…… 如果能找到 5 个这样的字符串起始位置（实际上，因为只有一个字符串起始位置，所以这 5 个位置是重叠的），就算验证成功。

其实我们也可以不用那么多的括号，只用一个『^』即可：

```
『^(?=.{6, 12}$)(?=[0-9A-Za-z-]*$)(?=(?!-).*(?<!-)$)(?=.*[^0-9])(?!.*--)』
```

总结

虽然“匹配”是正则表达式的常见操作，但细分起来，“匹配”又可分为提取和验证两种操作。

提取时需要照顾准确性和效率，因为此时字符串的起始/结束位置是不确定的，应当添加适当的环视结构，避免匹配了不期望的数据。

验证时对效率的要求并不高，因为验证的字符串一般都很短，而且验证的起始/结束位置都是确定的，直接在字符串两端添加^和$即可。而且验证有时候要比提取简单得多，我们可以改换思路，改“查找文本”为“查找位置”，针对验证时容许/不容许出现的每一个条件，写出对应的环视功能，作为一个将它们并列在一起。

关于作者

余晟，程序员，曾任抓虾网高级顾问，现就职于盛大创新院，感兴趣的方向包括搜索和分布式算法等。翻译爱好者，译有《精通正则表达式》（第三版）和《技术领导之路》，目前正在写作《正则表达式傻瓜书》（暂定名），希望为国内开发同行贡献一本实用的正则表达式教程。

原文链接：http://www.infoq.com/cn/articles/regular-expressions-5-match

相关内容：

- .NET 4 中的模式匹配
- 正则表达式（一）：纠结的转义
- 正则表达式（二）：Unicode诸问题（上）
- 正则表达式（三）：Unicode诸问题（下）
- 正则表达式（四）：正则表达式的与或非

推荐文章 | Articles

Guardian.co.uk 从 Java 切换到 Scala

作者 Charles Humble 译者 李剑

用卫报（guardian.co.uk）编辑的话说，这家报纸拥有世界上第二大的读者群，仅次于纽约时报；而它的开发团队正在逐渐从Java迁移到Scala。迁移工作是从Content API开始的，使用这套API可以查找并聚合卫报的内容。

Guardian.co.uk 网站大概有 10 万行代码。它用的是很典型的 Java 开源技术栈：Spring、Apache Velocity、Hibernate，数据库用的是 Oracle。它的 Content API 和网站一样，都是用 Java 做的，但开发团队决定把它换成另一套基于 JVM 的语言——Scala。Web 平台开发团队的主管 Graham Tackley 在采访中说：

> 我们过去几年都在用 Java，也一直做的挺好。不过作为一家新闻网站来看，我们需要对各种事件做出快速响应。我们现在用的 Java 平台可以每两周就发布一次 www.guardian.co.uk。比起很多企业级 Java 应用来，这已经很好了，但是跟其他网站相比，就差的没法说了。
>
> 所以我们早就开始寻找各种工具、流程、语言，希望能交付的更快一些。包括使用轻量级的 Java 框架，如 Google Guice；完全不一样的 Java 开发方式，如 Play 框架；也用过其他框架，如 Python+Django。在尝试的过程中，我们用了一段时间 Scala，但是跟其他的实践不同，我们在尝试期间，没有用 Scala 写过任何产品代码。
>
> 我们迫切期望着 Content API 的第一个非 beta 版本能够成为一个标志，它的发布将意味着这个 API 走进了持续演进的第一个迭代，从此以后，每当我们发现从前没有想到的有意义的功能，这套 API 都可以快速进化。但我们得保证 API 的客户端不会受到影响，所以我们需要一系列面面俱到的集成测试。我们一开始用 Java 来写集成测试，写了一段时间以后，决定都换成用 Scala 写。主要原因有三点：
>
> 1. ScalaTest 提供了很灵活的测试 DSL。
> 2. 我们希望写集成测试的过程充满愉悦，而不是令人厌倦。

3. 只用 Scala 写测试，就不直接影响产品代码，我们还能忍着怒气继续用 Java。

但是，用 Scala 写了 4 周测试代码以后，我们再也忍受不了只用 Java 来写产品代码了，于是决定把大部分代码都换成 Scala。

InfoQ: 总的来说，你们是怎么做移植的呢？是一次性的把所有 Java 代码都用 Scala 重写，还是这两种语言共存了一段时间？

> Content API的beta版本用的是一个私有的搜索引擎。当前版本已经换成了Apache Solr这款很优秀的搜索平台（这儿可以看到 卫报是如何使用Solr的），而且跟beta版的风格差异很大——beta版在一件事情上做的很好，它让我们看到了API不应该变成什么样子。在真正开始用Scala之前，我们决定先把API重新实现一遍，而不是重用原先的代码库。
>
> 这项工作用了三个人六周的时间，然后才开始引入 Scala，这样就有了一个稍微干净点的起点。但说到要把当前的项目停上几星期用来切换到 Scala，我们当时还没做好这个准备，所以还是先一点点移植现有的集成测试。当时我们用的 Maven 做构建，引入 Scala 就很简单了，maven-scala- plugin 可以构建混合 Java/Scala 的项目，只要按照它的说明来就行。这个插件可以允许 Java 和 Scala 代码共存，还能相互依赖。我们一个类一个类的重写，情况比我们预想的要好得多：它直接就能工作了。
>
> 在重写产品代码的时候，我们用了同样的办法：在几周的时间里，每碰到一点代码，就重写一点。到最后又用了几天来扫尾。

InfoQ: 你们用到了哪些类库/框架？

> 因为这门语言我们都没接触过，所以我们打算控制一下，不引入太多的新东西。我们依然用了普通的 servlet，Google Guice 做依赖注入，这也是我们现在构建 Java 应用的方式。还用了 SolrJ，这是跟 Apache Solr 通信的 Java 客户端。用 Joda-time 来做时间处理，用 Mockito 做 mock（这个类库也能跟 Scala 代码协同工作）。
>
> 有时候，为了保证定期交付，我们会有意保留一些我们熟悉的东西：比如它是用 javax.xml.stream.XMLStreamWrite 生成 XML 格式的端点（endpoint），而不是 Scala 那套很强大的 XML API。因为我们在换用 Scala 之前已经把这些给做完了，这些代码可以工作，容易理解，就留下了。不过生成 Json 格式的端点的时候，我们就换成了 Lift 用的 JSON 库——lift-json，因为它写出来的代码比 Java 的 JSON 库要干净很多。

InfoQ: 你们用的是什么 IDE？Scala 对 IDE 的支持怎么样？

我们用的是 Jetbrains IntelliJ IDEA 10。有些用的社区版，有些用的收费版。代码补全、检查调用点、类和方法跳转这些功能基本上一直都挺好用。但是它不能够很完美的识别出无效语法并且加上红色标记。查找 ScalaTest 的测试方法也有问题。不过比起我们之前所熟悉的环境来，这些不便之处换来的是一门强大而卓越的语言。

InfoQ: 你们项目里面是不是大多数人都是 Java 程序员？他们学 Scala 容易么？

是啊，我们所有人都有很丰富的 Java 开发经验。最开始的四人小组学 Scala 学的很开心：经常是一个人发现了一个 Scala 的新特性，然后就激动不已，跟整个团队分享。那种冲动已经很久没有在用 Java 的时候出现了。我们在共同学习，所以这个过程还不错。最开始的几周里，偶尔出现这种情况，我们打算用很体面的 Scala 的编程方式来实现一些功能，但就是搞不明白该怎么写。而且最关键的是，这儿要是用 Java 的话早就写完了，这就更让人泄气了。我们有几次受不了了，回家的时候说，“明天就换成 Java 吧。”但每天早上都会迎来一个新的开始，我们继续往前走。

然后又有十个 Java 程序员来写 Scala 代码。每个人的学习方式和速度都不一样，这也司空见惯了。但他们都走过来了，到现在要是碰到非得写 Java 不可的情况，每个人都会抓狂。

我们曾经拿学习 Scala 跟迁移到新平台上（比如 Python/Django 和 Ruby on Rails）做过对比。用 Scala 的话，原先 Java 的知识大概有 75%还能用得上。同样的库、同样的 IDE、同样打 jar 包和 war 包的方式，运行时环境和特征都一样。优秀的 Java 程序员可以用一天实现学会用 Scala 编写 Java 风格的代码，然后再学会闭包和隐式类型转换的强大之处，很快就能比用 Java 更有效率了。

InfoQ: 我听到人们对 Scala 最为诟病的是，这门语言太过复杂了。我觉得这些声音大多应该是由于 Scala 的可读性造成的：同样的代码，如果用更为严格一点的语言（比如 Java）来写的话，就会比 Scala 更容易理解。你觉得这种批评合理么？你是怎么想的？

没错，可读性应该是代码最重要的品质。我不在乎代码到底是命令式的还是函数式的，我也不在乎它到底是合乎 Scala 的规范，还是只把它当作无分号的 Java 来用，我只在乎它是不是好理解。在我们学到 Scala 新特性的时候，我们会根据代码是不是能够更明确的体现出实现意图来决定用不用。比如，我们试过 Scala 的 Either 类来消除一些 if 表达式，但团队一致认为，if 表达式更容易读懂，所以我们就没用 Either 类。

我承认，由于 Java 语法的严格，那些几行几行的代码看起来是容易懂，但是对于真正能用起来的代码来说，我觉得这根本不是问题。因为我不需要去理解实现细节，我想要理

解的是代码意图。优秀的设计和 OO 技术可以增强 Java 代码的表现性，但我读 Java 代码的时候，仍然常常见木不见林。在表现代码意图的层次上，Scala 突破了 Java 的很多限制。

举个例子来看，Content API 需要判断到底是要以 XML 或 JSON 格式返回结果，还是重定向到 HTML 浏览器上。它可以接收一个字符串，其格式为 format=query，也可以接受一个.xml 或是.json 的扩展名，也可以从 http header 的 Accept 字段里面读取扩展名类型。下面是实现代码，我觉得这可以很好的证明 Scala 强大的表现力（它会一步步调到 Scala 的 Option 类）：

```
def negotiateFormatParameter =getParam("format").
    orElse(getExtension).
    orElse(getExtensionFromAcceptHeader).
    getOrElse("html")
```

这同样也可以很好的证明，要说可读性这回事，最起码也得看看要读完多少代码才能搞明白一个功能。换成 Java 的话，就得写上很多跟问题域完全无关的代码，比如非空检查、getter/setter、用来做依赖注入的构造函数、操作容器。这些地方用 Scala 都可以写的很精炼。当然，你也可以说这类代码大部分都可以用 IDE 生成，但我还是得去读你写的构造函数和 getter/setter 啊，要不然我怎么知道你有没有在里面做什么其他处理

找个经典的例子来比较一下看看：

Java:

```
public class WelcomeClass {
       private String name;
       public WelcomeClass(String name) {
           this.name = name;
       }
       public String sayHello() {
           return "Hello " + name;
       }
   }
```

Scala:

```
class WelcomeClass(name: String) {
       def sayHello = "Hello " + name
    }
```

Java 版的代码三次告诉我“name“是 String 类型，并且五次提到“name”。Scala 版的代码只提到“name”两次，也只说了一遍“name“是 String 类型。这虽然是个很简单的例子，但也能说明 Scala 的优势：少了格式约束，少了重复，也就少了树木多了森林，

换句话说，读代码的人更容易看到实现意图而不是细节。

我也注意到，一行孤零零的 Scala 代码要稍微多花上一点时间才能明白它干了啥，但它急剧减少了代码行数，够本还有余呢。

InfoQ: 说到 Scala 的复杂性，你觉得这门语言的某些特性——我觉得这里主要是符号命名和隐式转换——会给真实应用带来麻烦么？

实际上隐式转换帮了我们很大忙。我前面说过，我们用 SolrJ 跟 Solr 通信，这是个很优秀的类库，但是它有 Java 库的通病——太喜欢返回 null 了。为了不让大量的非空检查污染了代码库，我们就把一些核心的类隐式转换成了别的类，而在新类里面就拥有一些 Scala 风格的方法。所以这个特性非但没有给我们在实际应用中带来麻烦，还帮我们解决了问题。另外值得一提的是，IntelliJ Scala 插件现在几乎可以理解所有的隐式转换了，所以如果你不明白到底发生了什么，control + 左键单击就能带你到发生调用的地方。

我们其实也在避免使用重量级的符号库，避免用符号来做方法名，但我觉得这是 Scala 的一个很重要的特性，也跟其他特性一样，都有可能被过度使用。有的时候这个特性很有用：我们有个方法从 http 请求中把用来做查询的字符串提取出来，这个方法名就是" ?"，放在代码里面很好理解。用 Scala 的时候，你得比用 Java 投入更多的精力来让别人更容易理解你的意图。能力越强，责任越大，说的就是这个意思。你不能因为这种力量容易被人误用就不想要它。

InfoQ: 人们对 Scala 企业应用的另一个关注点是，每一次 Scala 发行新版本总会破坏向后的兼容性，比如在 Scala 2.8 上编译的应用程序就没办法跟在之前的版本上编译出来的二进制文件一起用。你觉得这个问题严重么？你是怎么处理 Scala 项目之间兼容性问题的呢？

我们一开始用的是 Scala 2.7.7，等 2.8.0 和 2.8.1 发布以后，就马上迁移过去了。这个过程很顺利，到 2.8.0 版本用了不到一天（这还只是因为我打算消除所有 deprecation 的警告信息），然后又换到了 2.8.1。我们用到的所有库都针对 Scala 的各个版本相应的发布过多个版本，所以这方面没出现问题。

唯一出问题的地方是我自己的项目，当时我用的是 2.8 预发行版。但人家已经明确表明这是预发行版了，我还要用，那就是我自己的事了。

我们现在打算用 simple-build-tool 来构建 Scala，换掉 Maven。前者能够更方便的针对 Scala 的不同版本发布内部使用的库。

我宁愿面对升级带来的不兼容，也不愿意看到在 Java 里面有些东西永远不曾变过，也永

远无法改变。我们常用的 HttpServletRequest.getHeader 方法到现在还是返回 java.util.Enumeration，可 Enumeration 早在 Java 1.2 的时候就废除了。

InfoQ：你现在招人做 Scala 开发的情况怎么样啊？目前找优秀的 Scala 程序员跟找 Java 程序员一样容易么？

> 我们的主要关注点在于招的是优秀的程序员，而不是专门招 Scala 程序员。我们希望找到热衷于通晓多门语言的 web 开发者，至少用过 Groovy，或者是 Scala、Clojure、Ruby、Python。这种人往往都会喜欢 Scala 开发的工作机会。

InfoQ：现在 guardian.co.uk 一共有多少行 Scala 产品代码了？

> guardian.co.uk 每两周发布一次，它的核心代码在两周前还只有一个 Scala 类。我们之前一直有规定，在 guardian.co.uk 的代码库上不许写 Scala 代码。这是为了确保我们所有人都为迎接 Scala 做好了准备（也免得我总是动出重写的心思来）。
>
> 不过我们的 microapps服务已经用Scala实现了，在它的驱动下，网站的很多组件也换成了Scala，包括 Search（用Lift写的）、 Most Viewed、Punctuated Equilibrium Mystery Bird，在每个页面上显示相关内容的组件也是Scala实现的。
>
> 另外，我们新开发的 身份校验平台 也是用Scala写的，它还在开发阶段，不过第一个版本已经上线了。

InfoQ：你们打算将来继续用 Scala 么？

> 我们发现 Scala 可以以较少的代码量提升交付速度。这让我们团队重新焕发了生机。我们会继续挑选合适的工具工作，无论是 Scala，还是 Python、.NET、PHP、Bash。
>
> 过去的六个月里，我们新开的基于 JVM 的项目都用的是 Scala，没有一个用 Java。我们接下来的新项目肯定也不会用 Java 了，而且 Java 7 也没什么让人满意的新特性，发布时间又一拖再拖。

Tackley为想学习Scala的开发人员推荐了Martin Odersky的书《Programming in Scala》，该书的第二版包括了Scala 2.8 的内容。（译者注：去年国内出版了一本Scala中文图书，书名为“Scala程序设计：Java虚拟机多核编程实战”。由译者与同事郑晔翻译完成。只是该书中的Scala版本为 2.7.4。）他还提到：

> Scala REPL（命令行）可以用来试验 Scala 代码，效果很不错。而且，不管别人怎么说，在刚开始的日子里，只管大胆的把 Scala 当作无分号的 Java 用， 用上几天、几星期，或是几个月。如果你就到这一步为止了，虽然会错过很多精彩的故事，但这也算得上是不错的体验。你可以循序渐进的来不断学习和接受 Scala 的新特性。我觉得，正是因为这种学习方式的存在，才使得 Scala 成为 Java 程序员进阶的不二选择。

抛开技术方面不谈，从业务的角度来看，guardian.co.uk的Content API和开放平台服务栈（Content API是其中的一部分）也是很有意思的，因为在UK和其他国家，越来越多的优秀报纸把自己的内容放在墙内而不是共享，卫报则提出了与众不同的方案。迄今为止，金融时报，新闻国际旗下的泰晤士报和星期日泰晤士报都走的封闭路线，而最近纽约时报还推出了付费阅读模式。BBC的资深记者John Humphrys在太阳报（新闻国际旗下的一个小开型日报）上跟人争论说，“优秀的记者必须要得到报酬，就像我们付钱给水管工修水管一样，不然这事就没人干了。”但Tackley对此有不同观点：

> 我们坚信，数字出版的未来一定是走出去跟互联网集成、协作，而不是退缩到网后。
>
> Content API 可以把卫报的影响力和品牌扩展到我们自己力所不能及的范围。这都是使用 API 的第三方和合作伙伴帮我们做到的，他们会投资某个领域，然后使用相关的卫报内容。
>
> Content API 的访问权限分为若干等级：非注册用户可以访问内容元数据，但不能访问具体内容，每秒查询次数也有限制；注册以后，可以看到文章内容——包括嵌入式广告，每秒查询次数也有限制；顶级用户是那些卫报的合作伙伴，我们会签署一份合适的商业协议。
>
> 我们有一家不错的合作伙伴，叫做 WhatCouldICook.com，它是个个人开发的网站。它调用API来解析获取我们发布的所有食谱，然后换成大众喜闻乐见的形式展示出来。与此同时，我们的读者也会从中受益，因为我们把whatcouldIcook.com的一些功能也聚合了进来。比如 在卫报网站右边就 有搜索食谱的功能。
>
> 我们基于API还做了 wordpress插件 ,所有wordpress用户都可以在他们的博客上引用相关的卫报内容。
>
> 像这样在特定领域内的创新，没有Content API就是做不到的。我们还把Content API用在卫报自己的项目上，推动了内部革新；比如 search、zeitgeist这些功能，以及 手机版站点 ，iPhone应用都是Content API驱动出来的。

原文链接：http://www.infoq.com/cn/articles/guardian_scala

推荐文章 | Articles

上线：准备和部署软件包时开发和运维的角色

作者 Andrew Phillips 译者 姚九强

按照发布流程正确的部署软件——二进制代码和与之相关的配置文件——到你的开发、测试、 验收或产品环境（DTAP）是一项复杂的任务，涉及到众多部门和团队。不幸的是，在许多组织中这项关键的流程还是费时并容易出错的。

这篇文章里，我们会探讨开发团队、运维团队和其它相关方如何通过协作来准备一个“好”的部署软件包。“好”的软件包能减少部署中出错的可能，并在需要自定义环境时提高部署的透明性。

此外，我们还会检视为何一个结构良好的部署包更易于转为自动化部署，提升生产率和可靠性，同时减少软件开发和维护生命周期中的错误和等待时间。

区分部署过程中的担忧：为什么 vs. 如何做

显而易见，部署包需要包含应用程序的所有组件。而不仅仅是你自己的二进制包——EARs，WARs 之类——通常这些包由集成构建产生，还应包含静态内容、配置文件、共享库、防火墙配置等等。特别还应包括应用服务器的参数和设置，比如消息队列、连接工厂、数据源或类环境变量。实际上，软件包应包含软件生命周期中的所有内容，也就是那些需要一起被部署、升级和取消部署的内容。

确保软件包是“完备的可部署单元”对于一次可靠的部署来说是至关重要的，特别是在大规模环境中。指望目标中间件栈（middleware stack）已经用“正确值”设置好是导致部署失败的一个常见原因，这通常导致耗费很多时间来找出不同环境中的细微差异，这往往是令人沮丧的过程。

部署包中只包含配置信息，比如共享“服务”，例如为多个应用所共享的消息队列或数据源，这种情形不鲜见。这些配置显然应该按照有版本管理的、可部署的对象来处理，意味着这些配置文件按照和“普通”应用程序一样严格的流程来发布。实际上，平稳、可靠的部署共享服务通常更为关键。

除了确保软件包最完整的描述了包含什么使特定版本的应用程序或配置正确工作，软件包还需要指出它们能够运行所需的其它东西。换句话说，部署软件包需要明确界定他们的先决条件和依赖，以便在匹配的环境上部署。

准确的说，哪个部门负责部署和组织结构有很大关系。多数情况下是开发团队，他们通常能使用持续构建工具使某些发布流程变为自动化，但发布过程还可能涉及其它团队。举例来说，让 DBA 在发布前确认 SQL 脚本，或要中间件竞争中心（Center of Competence）检查中间件配置的情况并不少见。

请不要增量式发布（Delta Release）

通常，开发的一个目标就是尽量减小对目标环境的影响。显然，如果你的应用需要“不间断运行”或多个应用程序共享一个集群，不必要的重启服务器是要尽量避免的。

为了达到这个目标，一个通常的做法是要求开发团队提交一个“增量”发布包，其中值包含新的或修改过的系统模块，并且只部署这些模块。经验显示，其实这是一个冒险的策略，应尽量避免，原因如下：

- 准备一个增量发布通常都是手工任务，完全依赖开发人员判断哪些是修改过的模块而哪些不是。这是一个耗时的又容易犯错的过程，因而这个过程几乎不可重复 。
- 某个应用程序部署包的版本不适合实际部署——在最坏的情况下；可能所有的组件都需要上一版本。部署到一个“干净”的环境（设想需要快速设置几个干净的镜像来重现一个压力问题）则更加复杂，失败的风险也因所有早期版本的包必须还要保留、并按照顺序保留而增大。这是因为，当然，简单来说就是数据库增量备份的问题。
- 除非应用程序的每个版本都部署到了每个环境，否则在升级到相同版本时会产生问题，因为需要从不同的版本升级。这很快会在选择了错误的增量发布包时导致混乱和产生错误。
- 不包括完整的包，而只是一些碎片的发布仓库不是一个好的最终软件库的候选。

上述各点都无法让减小部署影响的目标失效，这是显而易见的。但部署包不能解决问题 。实际上，部署流程才能应该通过在部署时推知哪些组件应增加、哪些被修改和哪些应移除，并据此实施来满足这个需求。

在时限压力下手工实现上述部署流程是非常困难的（开发人员总是首先要做增量发布包），而一个好的自动部署系统能够很容易地实现上述目标。实际上，这是引入自动化部署的一个主要好处。

在上述情况下，通常会设立专门的发布管理小组负责协调各种交付物，部署包按照便于恰当的人更新、修改和批准部署包的内容来组织是非常重要的。

构建部署包是一个需要多个步骤的工作流，需要很多部门参与并耗时很长，也会产生准备“不完整”部署包的情况。这种情况下，发布管理系统不产生这种“不完整（draft）”发布包非常重要。

我们建议由恰当的人员对已批准和已发布的包进行数字化签名，这样所有的包在部署前都通过验证，为部署安全提供更进一步保障。

深入到比特和字节

选择部署包格式主要是从方便起见。当然，部署包应该：

- 便于移动——最好是个单一文件
- 便于检查
- 可压缩，因为文本文件，如 SQL 或配置文件压缩率很高
- 支持某种错误修正
- 可移植（比如开发人员在 Windows 环境开发，生产环境是 UNIX）
- 可进行数字化签名，目的是可以验证待部署的包是通过审批的包。

所以通常来说，选择一种压缩文件格式，比如 ZIP、TGZ 或 RPM，作为发布包的格式，但具体选择哪种完全取决于你自己。

目前关于 build 文件格式没有多少可说，因为除了 WAR 就是 WAR。然而对于配置文件，情况则大不相同。目前，如果消息队列和数据源的定义，甚至包本身（在发给运维团队的成员的电子邮件中碰见这种情况很常见）通常都只是由人们可读的文档来描述的，比如发布注记

（release notes）或按模板写的 Word 文档，这些文档非常容易引起歧义并难于检查修改。

基于上述原因，配置文件也应该由机器可识别的格式来定义，比如 XML。不幸的是，在这个方面目前还没有什么标准可推荐。事实上，这方面也是我们非常希望和用户及供应商合作的。

最好避免使用某种特定中间件定义的格式，比如 WebSphere 的 XML 配置。有时直接从已有的配置文件中抽取配置项看上去很方便，但这样会限定于某个中间件供应商，在部署到不同的服务器环境时可能会遇到困难。通常，这是完全可以避免的，因为应用程序通常不会用到特定中间件的某个特定特性，因此可以在自己的配置文件中很容易的定义更“一般”的 JMS 队列，数据源等等。

最后， 声明文件或清单应该能描述包的内容。应该在这里列出包的依赖关系，给操作者的描述和任何其它不容易找到的信息，比如开发人员的联系信息，项目名称，等等。声明文件还需指出包内项目内容的类型，这些内容往往不容易通过名字判断出来：EAR 文件显然是 EAR 但是一个 ZIP 文件可能包含 HTTP 服务器的的静态内容、应用程序配置文件等等。

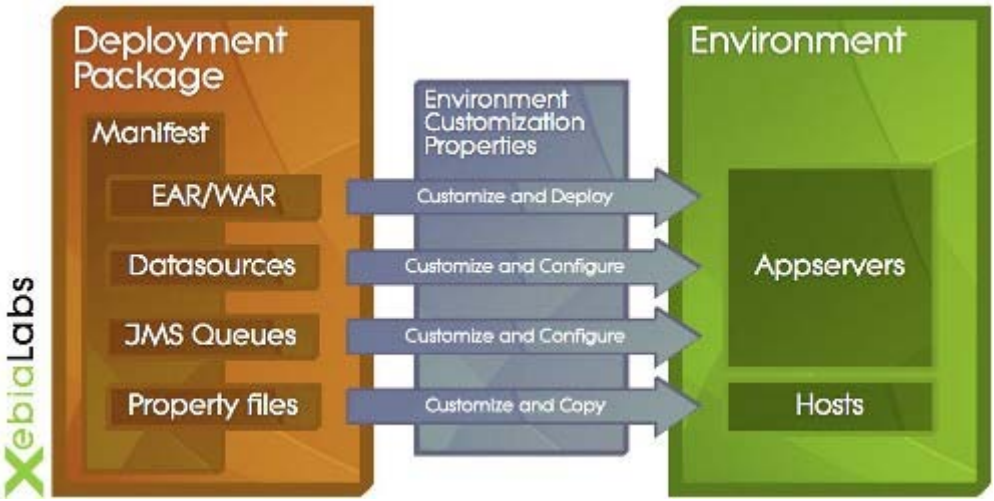
再次指出，文件格式应该是机器可读并可验证的，同时也应该是容易自动生成的（比如从构建系统生成）。

我们现在有了方便的、可移植的并能自动部署的软件包，这个软件包也是待部署应用程序要包含什么内容的完整描述。到现在，我们还没说任何如何部署的内容。发布注记在哪儿？部署和回滚计划呢？

理想情况下，应该没有这些东西。更准确的说，标准的 Java EE 应用程序的部署流程应该，以部署包内容为基础，是足以完成正确部署的。应用程序特定的部署步骤意味着满足该应用程序的些特殊要求，这些特殊要求在之前只有开发人员了解。当凌晨 2 点钟时需要进行一次紧急部署，而无法联系上开发人员，运维人员犯错几乎是不可避免的。简单来说，开发人员应该知道什么应该被部署，他们不应该为如何部署而操心。

当然，组织内有标准的 Java EE 部署流程是实现上述目标的先决条件，当你的组织有几类不同的应用时可能有几个流程。这个流程要足够复杂，以便能覆盖在部署时所需的各个组件和配置，但限制可选项的数量以保证流程的可维护性、测试性和可靠性也同样重要。

这必然会给开发技术加上一些限制条件。然而，相比在当前的情况下维护成本是运行应用程序中最大的日常开支这一事实，减小犯错可能性、更可靠的部署流程带来的好处胜过最大化开发灵活性。当然，在每个组织都需要依据自身的要求来平衡这两者。



标准的部署流程不只是更容易维护。它也使运维人员在压力下更可靠的执行操作，相比于依照大量针对不同应用程序的不同版本的发布注记手工操作容易得多。标准部署流程也会让自动化变得更简单，让运维人员从重复任务中解脱出来，同合适的访问控制相结合，可以让开发人员自己执行部署。如果有适合的集成点，将构建和发布系统连接起来实行持续部署以达到真正的端到端自动化也是有可能的。

此外，实现更多的自动化部署流程来支持更复杂的场景，让开发团队在避免临时流程时更加灵活性更容易。不过随着自动化流程变得越来越复杂而难于实现和测试，是否采取复杂的自动化流程需要仔细考虑。但是相对手工流程明显的优势是，一旦自动化流程通过测试和校验，你可以确保流程的所有步骤是执行结果是可预测的。对人们来说，不断成功的执行复杂的、步骤众多的流程的目标是非常难以达到的。

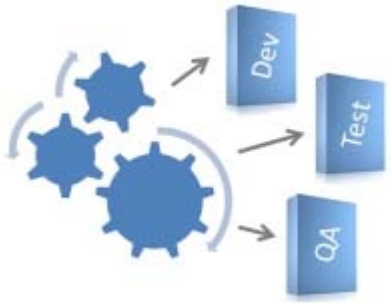
从部署包到运行中的应用软件：为目标环境定制化软件包

目前为止，我们探讨了将所有内容打为一个自描述的包，这是发布管理中的一个好实践。这样做，我们默认为一个软件包可以“原封不动”地部署到不同环境。然而在当前的部署、测试、验证和生产过程中，我们其实还须针对不同的目标环境“调整”应用程序、相关配置和资源——设想一下配置文件或者数据源、用户名及口令等配置项。

有些公司在虚拟化和虚拟设备的道路上更进一步，这会让上述情况有所改变，但当前构建部署包和部署流程应简单、可靠并在部署时透明定制应用程序组件和配置文件。

在缺少已有标准，甚至是指导原则的情况下，为了解决上述问题人们尝试了很多解决方案，从如 JMX 等非常优雅的方法到简陋的方案如字符串搜索替代 。

此外，不同的中间件平台有各种不同程度的定制化支持：通常，门户、ESB 和过程服务器提供某些解决这些问题的“本地化”解决方案，但是应用服务器倾向于让用户自己弄好这些配置。



时常，结果定制化方法因不同项目、目标平台和部门而异变成大杂烩。像以前一样，这会导致维护开支增加和提高潜在的混乱和错误。

那什么是“标准“定制化解决方案应提供的呢？当比较不同的方案时，下面这几件事应牢记在心：

1. **方便**：定制化过程应便于设置和快速实现。对有时需要在部署到开发环境时进行定制化操作的开发人员来说这很重要。
2. **可视化**：便于授权用户查看特定部署环境上的定制项（应用程序中可定制的部分）和赋给这些定制项的值。
3. **安全的**：当缺少定制项的值或设置了错误值（比如忘记设置超时值或在生产环境上设置为测试环境上的值）时，应该能够快速检测到。宁可在部署时检查到定制值缺少或错误也要比在运行时出现模棱两可的错误好。
4. **有版本控制和访问控制**：只有授权用户才能查看和修改应用程序定制项的值。最好能记录这些值的变更历史。这有助于比较同一应用程序不同版本的定制值，也允许用户比较同一应用程序在不同部署目标环境上的定制值。

无论使用那种技术，定制化方案主要分为两类：基于指针的定制化和基于符号的定制化。

基于指针的替换只是“搜索并替换”的一种时髦的说法，它稍微先进一些的同类，基于 XPath 的替换，通常能在门户或 ESB 环境中见到。这种方法是脆弱的，因为部署包包含有效的定制值，因此存在隐蔽错误的风险很大。更进一步，部署包的定制值基本上不可见。讽刺的是，这种方法在为那些不能定制化的包打“补丁”时很有用。

总的来说，准备基于指针方式定制化的包的确更容易——简单来说就是将开发环境或其它“创建（authoring）”环境上的设置导出，或者做一个快照。当然，这个导出过程可以是“符号化（tokenized）”的（用符号来替代需要定制的值），但这是易出错的并且引起高昂的手工开支，特别在开发过程中经常需要导出时是个问题。

显然，基于指针的定制化仅在需定制的设置或定义用某种方式结构化的情况下才可行——否则不可能构造一个指向待修改项的指针。XML 和资源定义（比如 properties 文件）归为这一类，但类似纯文本文件则不属于这一类。

基于符号的替换，是另一种方式，主要指占位符——部署包包含（构件、资源定义 等）特殊的符号，在部署时这些符号被给定的值替代。

${replace.me}

基于符号的定制化要求部署包提供者，通常是开发人员，专门准备部署构件。

这意味着，首先，所有待替换的符号在交付时要清楚，应用程序有了一个正确定义的定制项集合，这是额外的好处。符号的特殊语法也让验证替换值是否都给出提供了可能。

即使校验失败，应用程序通常因符号的语法错误而终止——符号一般来说不是合法值——这也时额外的安全机制。

当然，在开发人员来看这种错误保护机制也会是麻烦的。因为除非符号被替换否则应用程序无法正确工作，这要求必须建立构建流程来实现它，同时这个流程还要是快速的，当然是在开发环境。

基于指针的替换方案对于那些没设计成可定制化的应用程序也适用，然而基于符号的替换方案在错误保护和可视度方面有明显优势。另外，基于符号的替换方案便于传播通过特定环境的只有运维人员才能访问的的定制值（想一下如生产环境数据库口令）来得知应用程序在哪些地方需要定制（开发人员应该知道并详细列出）的知识。

因为部署通常都是关键任务，这些实实在在的好处应使基于符号替换的方案在任何可能的时候作为首选。

事实上，应用程序的定制项应列在包的声明文件中。基于特定的符号语法，通常能够（并推荐）依据这个定制项列表对包内容进行检查。在部署时，也可用同样的方式验证，为安全的部署提供了额外的保障。

> **请不要区分开发、测试、验收和生产环境的部署包**
>
> 任何尝试过，在开发过程中，按含糊的说明来判断要替换什么、这些定制项在哪个配置文件中以便让应用程序运行起来的人，可能会纳闷为什么不能用不易出错的方法做事。如果没有合适的自动化部署方案，一个常用的替代方法是尝试将定制作为构建和发布流程的一个环节。
>
> 从技术角度看，一个容易想到的选择是：目前有许多好的持续构建和发布产品，开源的和商业软件都有，并且大多数组织已经有了。很多工具都支持“profile”这个概念或类

似区分 build 细节的机制，并且如果没提供这类机制的话也提供了让你加入自己的搜索替换功能的钩子。

然而，有些严重的缺陷导致这个方法作为合适的定制化方案并不合适：

- 在构建过程中需要访问环境相关的细节。这不应只是觉得“错误”，有些信息可能是高度敏感的——比如生产环境数据库的口令——让这个方案从安全角度来看不可行。
- 持续构建工具一般不能提供恰当的有版本控制的、有安全机制的仓库存储环境相关的定制值，很容易犯臭名昭著的复制粘贴 env.properties 文件的错误和其它错误。
- 有了环境相关的 build，几乎不可避免的是，在某些时候你会不小心在生产环境上运行测试 build。

结论

顺畅和可靠的部署流程从简单的第一步开始：以一种自动的、可检查校验的方法整合并发布一个结构化的，完整的部署包，该部署包定义了特定版本的应用程序所有的组件、配置和依赖关系。这能显著的减少因不合法或缺少定制值、组件或必须的服务而产生的错误。

计算机可读的包也是引入部署自动化的第一步，这里提到的自动化不仅是检查校验自动化，还包括按照自定义的标准流程部署这些包，并按照环境自动定制。除了确保一致性和提高可靠性之外，这可显著提高效率并可以免除开发和维护生命周期内的一个最常见的瓶颈。

关于作者

作为一位早期就相信 Java 有能力去交付“企业级”软件的人，Andrew Philips 很快关注于高产能、有弹性和可扩展的 J2EE 应用软件开发。特别是在并发和高性能开发方面，Andrew 通过不断在多个国家从事复杂的并有挑战性的企业应用环境工作积累了丰富的经验。他一直关注有效的集成有前途的新 Java 开发到公司软件开发，Andrew 于 2009 年 3 月加入 XebiaLabs，在那儿他作为公司部署自动化产品 Deployit 开发团队的一员。和其他人一起，他也为开源 Java STM 实现 Multiverse 和领先的 Java 云计算库 jclouds 贡献代码。

原文链接：http://www.infoq.com/cn/articles/dev-op-xebia

相关内容：敏捷经济学：故事点成本、SOA治理：实现并维持业务及IT的敏捷性

新品推荐 | New Products

Colm 语言简介

作者 Jean-Jacques Dubray 译者 侯伯薇

Colm是一种新的编程语言，设计它是为了对计算机语言进行分析和转换。Colm的主要贡献在于解析方法。Colm是由Adrian Thruston设计的，那也是他的博士研究工作的一部分。

原文链接：http://www.infoq.com/cn/news/2011/05/colm

Entity Framework Power Tools 简介

作者 Jonathan Allen 译者 侯伯薇

在 Entity Framework 中引入 Code First 的做法已经获得了很多赞誉。尽管这项技术未来还有很长的路要走，但是它已经动摇了 ORM、nHibernate 的统治地位，微软所做出的进展非常惊人。为了保持这种势头，微软又发布了一系列强大的工具，它们能够获取现存数据库的信息，并创建 Code First 的代码。

原文链接：http://www.infoq.com/cn/news/2011/05/EF-CodeFirst-PowerTools

Siwpas 1.0——针对 EE6 Web Profile 基于 Tomcat 的服务器

作者 Kostis Kapelonis 译者 侯伯薇

最近 MechSoft公司 发布了Siwpas，它是基于Apache Tomcat的轻量级应用程序服务器，特别专注于Web应用程序。

原文链接：http://www.infoq.com/cn/news/2011/05/siwpas10

JetBrains 发布 IntelliJ IDEA 10.5，全面支持 Java 7

作者 Charles Humble **译者** 王丽娟

JetBrains发布了 IDEA 10.5，对Java 7 提供了全面的支持，并支持代码完成、检查和快速修复。

原文链接：http://www.infoq.com/cn/news/2011/05/idea-105

下一个版本的 Windows Phone Mango 包含了大量新的特性

作者 Abel Avram **译者** 李永伦

微软宣布了 Mango，Windows Phone 即将到来的版本，这个移动操作系统希望通过提供大量新的特性追上竞争对手：支持移动硬件加速的 IE、多任务、整合通信、Silverlight 4 和 XNA 的支持、更多的传感器、VB.NET 的支持等等。

原文链接：http://www.infoq.com/cn/news/2011/05/Mango-Windows-Phone-7-1

Akka 1.1 发布，对 Futures 和性能做出改进，降低了依赖性

作者 Werner Schuster **译者** 侯伯薇

Akka 1.1 是在 5 月 12 日发布的，它在 Akka 1.0的基础上做出了一些改进。另外，Scala 2.9也在当天发布（Akka 1.1 需要Scala 2.9）；Scala和Akka现在都是 由Martin Odersky和Jonas Bonér创建的新公司Typesafe来管理的，该公司提供针对Akka、Scala以及构建在这两种技术上的商业产品的培训和支持。Typesafe还提供了一个 完整的框架。

原文链接：http://www.infoq.com/cn/news/2011/05/akka11

Spec Explorer 2010 3.5 版改进了更改检测和批处理

作者 Jenni Ripley **译者** 李永伦

微软 最近发布 了Spec Explorer 2010 3.5 版。这个Visual Studio扩展允许用户生成可重用的测试代码，以及创建、分析和生成软件行为的图形模型。Spec Explorer的模型可以通过使用C#写的规则来创建，也可以通过使用正则表达式的操作模式来定义。

原文链接：http://www.infoq.com/cn/news/2011/05/Spec-Explorer-35

Essential: 新语言工作台

作者 Jean-Jacques Dubray **译者** 金明

语言工作台正变得 日益流行 。在这个领域内 ,甚至有着重于比较 不同产品 的 大赛 。Pedro Molina最近发布了 他的新语言工作台Essential的公众beta版。

原文链接：http://www.infoq.com/cn/news/2011/05/essential-wl

Microsoft 发布 iOS 版 Windows Azure 工具包

作者 Simon Guest **译者** 杨晨

继近期发布了 Windows Phone 7 版的Windows Azure工具包之后 ,微软 宣布 将在 5 月 9 日发布iOS版的Azure工具包，并且计划在下个月发布Android版本。

原文链接：http://www.infoq.com/cn/news/2011/05/watoolkit-ios

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

Java社区：企业Java社区的变化与创新

.NET社区：.NET相关的企业软件开发解决方案

Ruby社区：面向Web和企业开发的Ruby，主要关注Ruby和Rails

SOA社区：关于企业内面向服务架构的一切

敏捷社区：面向敏捷软件开发的项目经理

架构社区：设计、技术趋势及架构师所感兴趣的话题

运维社区：关注IT运维中的存储、监控和设计

官方微博：http://weibo.com/infoqchina
豆瓣小组：http://www.douban.com/group/infoq
讨论组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
商务合作：sales@cn.infoq.com

推荐编辑 | 翻译团队编辑 金毅

霍总让我写点编辑感悟，着实让我犯难了。我也是个说起来还算年轻的 IT 人，一来感悟就不免沉重：IT 人到底为了什么？二十年以后我们能干点什么？我们真的就是一辈子技术人了？……

我想，很多时候，我们总会有很多的感悟。感悟到生命在飞逝，感悟到生活的真谛，感悟到人生的意义，感悟到希望在前方…。可转念一想，所谓的感悟往往是一种比时间飞逝的还要快的感觉！明天一觉醒来，或许我们又会义无反顾的投入到虚度中去。十多年的美好时光，很多次的一而再，再而三的感受到了这种矛盾和落寞，可人心的感悟竟然是这般烟云易过不留痕迹？

也许，我们可以说我们是自己最大的叛徒。我们都在努力、都在奔跑，这是自我的寻找还是救赎，或许还是彷徨：在希望和困惑之间，在坚定和迷茫之间，说着一套套的理由，却更像永恒的悖论，没有答案？

我们就在这个浮躁的年代飘着、飘着、飘着……但我们都知道需要坚定、需要坚持、需要实干。

我想，这是我加入 InfoQ 的一个原因，我来学习、我来分享、我也愿意努力为建立一个优秀的社区而服务。我们需要为我们自己做点什么。我很高兴，我看到 InfoQ 的网站上，有那么多的“干货”，当中多少还有点我的努力！我很高兴，InfoQ 的同事能够直言不讳指正错误、追求完美、共同进步，这种严谨的治学态度在工作中同样重要！我也很高兴，我能参与到各种活动之中，第一时间聆听接受大师牛人们的思想见解，乃至推翻我之前的认识，并且与人分享……

一杯绿茶，几许清香。IT 人永远在不断地快速学习中前进，在命运选择你之前，选择命运！

我在 InfoQ，大家也在！

封面植物

紫斑牡丹

紫斑牡丹，野生种属于渐危种。紫斑牡丹野生种星散分布于中国陕西、甘肃和河南西部。由于根皮入药，长期遭受过度采挖，资源不断被破坏，又因天然繁殖力弱，分布区逐渐缩小，应采取有效措施，加以保护。

生长分布：分布于山坡次生夏绿阔叶林下或灌丛中以及岩石缝中。分布区的气候夏季温暖而干燥，冬春干冷，最低温在 0℃以下，年平均温 13℃左右；年降水量 650--950 毫米，多集中在 7--9 月。土壤为山地灰化土及山地棕壤和黄壤。

植物形态：紫斑牡丹为落叶小灌木，高可达 180 厘米；茎直立，圆柱形，微具棱，无毛。茎下部叶为二回羽状复叶，具长柄；其下部的一回羽片为卵状长圆形，具二回羽片 2-7 片；最下部的二回羽片为宽卵形，长 2.5-3 厘米，宽 2-3 厘米，三深裂或全缘，裂片卵状椭圆形，或长圆状披针形；叶正面无毛或近无毛，深绿色；叶 背面粉绿色，疏生柔毛，叶脉上较多。花大、美丽，两性，单生枝顶。萼片通常 4 枚，近圆形，长约 3 厘米，宽 2-2.8 厘米，先端短尾状尖；花瓣通常 10 枚， 白色，腹面基部具紫色大斑纹，宽倒卵形，长 6-10 厘米，宽 4-8.2 厘米，基部楔形，先端截圆形，微有蚀状浅齿；雄蕊多数，长 1.8-2.5 厘米，花药 长圆形，黄色，长 6-8 毫米；花盘革质，鞘状，包被子房，果期开裂成瓣；心皮 5-8 个，子房密被黄色短硬毛，花柱短，柱头扁平；蓇葖果长 2-3.5 厘米， 粗约 1.5 厘米，被黄毛，尖端具喙。

保护价值：紫斑牡丹是中国特有植物，为珍贵的花卉种质资源，根皮供药用。对研究牡丹属的系统发育和培育牡丹新品种，都具有一定意义。陕西太白山已建立自然保护区，应采取有效的保护措施。建议在分布集中的地区建立保护点，严禁采挖，并积极开展繁殖试验，推广栽培。北京、东三省、青海同仁县等已引种栽培。

1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 6 月刊

每月 8 日出版

本期主编：霍泰稳

总编辑：霍泰稳

美术编辑：胡伟红

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：
http://t.sina.com.cn/infoqchina

商务合作：sales@cn.infoq.com 13581658359

本期主编：霍泰稳，InfoQ 中文站创始人兼总编辑

霍泰稳，有多年的软件开发经验和媒体从业经历，以技术传播为己任，关注企业软件开发领域的变化与创新。大学毕业后先是在杭州从事软件开发工作，基于多种语言和技术开发过 MIS、GIS 等系统，但现在大多技术已经遗忘。2004 年步入技术媒体行业，并渐渐沉溺其中不能自拔，在 CSDN 工作期间曾先后参与《程序员》杂志、《MSDN 开发精选》杂志、《开源大本营》图书和《开源技术选型手册》2008 版图书的策划编辑工作。后参与创建 InfoQ 中文站，并主导运营至今！

《架构师》月刊由 InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ neue
Enterprise Software Development Community

商务合作：sales@cn.infoq.com
读者反馈/投稿：editors@cn.infoq.com