促进软件开发领域知识与创新的传播

架构师 ARCHITECT

推荐文章 | Article

谈谈那令你永生难忘的编程之路

专题 | Topic

微服务与持续交付

观点 | Opinion

做云计算，如何才能超越AWS

不要让底层细节被上层打散

特别专栏 | Column

图像验证码和大规模图像识别技术

OpenResty 是什么

InfoQ Queue

2016年1月

锤子手机发布会提到的 OpenResty是什么

文章介绍了OpenResty基金会的运作方式以及OpenResty的未来发展。

谈谈那令你永生难忘的编程之路

InfoQ邮件采访了经验丰富的软件架构师、研发团队领导或自由职业者，现将他们早期如何开始编码的故事呈现给大家。

微服务与持续交付

本文节选自王磊著《微服务架构与实践》，介绍了持续交付是什么，以及微服务如何做到持续交付。

梁胜：做云计算，如何才能超越AWS

本文根据梁胜在ArchSummit北京2015大会上的主题演讲《容器时代的云计算》整理而成。

SpeedyCloud研发总监李孟：不要让底层细节被上层打散

为了进一步了解技术人在CDN服务设计研发工作背后的故事，InfoQ专门现场采访了李孟。

图像验证码和大规模图像识别技术

目前的图片自动识别技术到底有没有可能破解这种验证码呢？有没有更好的图像验证码方法，既安全又不影响真人的使用体验？

架构师 2016年1月刊

本期主编 董志南

流程编辑 丁晓昀

发行人 霍泰稳

联系我们

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

EGO EXTRA GEEKS' ORGANIZATION NETWORKS

高端技术人员
学习型社交网络

InfoQ queue

专注中高端技术
人员的社区媒体

StuQ queue

实践驱动的
IT职业学习和服务平台

GiT GEEKBANG INTERNATIONAL TRAINING 极客邦培训

一线专家驱动的
企业培训服务

地址：北京市朝阳区洛娃大厦C座6层1607室　　网址：www.geekbang.org

霍泰稳 极客邦科技创始人兼CEO

卷首语

技术社区的重度垂直

记得 2015 年《架构师》第一期的卷首语，我写的是“Dare to Dream”，大意是说做事情还是应该敢想，有了大的目标，并为之而努力，所做的成就也可能更大一些。2015 年过去了，回头看，我更加认为这个说法是正确的。

举个例子，在 Dare to Dream 那篇文章的末尾，我提到“请大家记住这个名字，因为它极有可能会影响和改变你的生活，如果你是一个技术人的话：极客邦（科技）”。经过一年的努力，我估计很多圈内的同学都看到了我们的努力，而极客邦科技 Geekbang 这个名字也在慢慢发酵。不用提我们一直在做的 InfoQ 网站以及 QCon、ArchSummit 这样的高端技术会议，2015 年极客邦科技研发的 StuQ、GiT、EGO 等新产品也已经走在深入人心的路上。

直到现在，还有很多人说技术社区是个很小的市场，很容易触摸到天花板。而我说，那是他们真的不懂技术社区，或者没有深入地去了解“技术圈”的诉求。社区可以做的很小，也可以做的很大，技术人从就业到学习到招聘到创业，企业从研发团队建立到能力提升到业务转型和突破，技术无一不扮演着重要的角色。回到开头的话，就看你的梦想有多大。在我们极客邦人看来，技术社区是如此的性感和丰饶，足够我们深耕细作 N 多年。而且我们还要培育更多的同行加入进来，一起玩。

但，显然，只有设想或者梦想还远远不够，如果停留至此，最终留下的只能是一个一个的半成品。很多社区倏然起于青萍之末，又倏然止于草莽之间的原因，大抵也是如此。这就是为什么在 2016 年，在我们有了一个大的梦想之后，

更加强调“重度垂直”的主要原因。简而言之，就是把梦想放在心底，埋头苦干，夯实基础，深入了解用户，进而优化产品，为技术人和企业提供更好的服务。

我希望在2016年，我们的内容（InfoQ）更加垂直，每个编辑都扎根于社区，从社区中来到社区中去，不能说是某个技术领域的专家，但要是某个技术领域的活动家；我们的会议也更加垂直，除了综合性的大会，在一些热点技术领域也能迅速纠集专家进行深入研讨，推动这些技术的快速普及和实践；我们的产品也更加垂直，不同人群的诉求不同，高端技术人有高端技术人的圈子（EGO），一线开发者有一线开发者的学习特征（StuQ、GiT），我们需要放下身段，和大家“在一起”……

另外，我们服务的对象，包括技术人，其实还有技术人背后的企业，这也是我们很多的业务是“To Business”的主要原因。在2016年，我也希望我们的营销人员把服务做得更垂直，让客户的费用能够花的更加有效率，站在客户的角度帮助大家省钱。在服务的意识上，也只有和客户“在一起”，才能赢得客户的信赖，共同将开发者社区的生态环境建设的更加完善、合理。

最后，希望在2017年元旦，我们回头看时，这些“垂直”已经让极客邦科技的梦想——“整合全球优质学习资源，帮助技术人和企业成长”——变得更加现实。

再次祝我们《架构师》杂志的读者朋友们新年快乐，学习更上一层楼。

Google确认下一个Android版本将不会使用Oracle的Java API，转而使用开源的OpenJDK替代

作者 百占辉

在下一个Android版本中Google将会把应用程序接口（APIs）的实现替换为OpenJDK，它是Oracle私有的Java开发工具包（JDK）的开源版本。Google确认了Android N将会仅依赖于OpenJDK，而非Android自身实现的Java APIs。一位Google的发言人说：“最为一个开源平台，Android的构建是基于开源社区的合作。在即将到来的Android的下一个版本Android N，我们计划将所有Android的Java语言开发包用OpenJDK实现，从而为开发人员在构建应用程序和服务时提供通用代码库。Google是OpenJDK社区的长期贡献者，并且我们期待在未来为OpenJDK作出更大的贡献。”

Android提供了一定的Java API库，以支持使用Java语言来开发Android apps，这些库分为两部分：API库和Google开发的API库的实现代码。Oracle开发的Java，其API库由两种实现：专有的JDK版本和开源的OpenJDK版本。Google决定全面使用OpenJDK，其实Android在一些地方早已开始使用了，使用OpenJDK意味着要开源这部分的实现代码。

这个code commit表明修改了8902个文件，明确表示了OpenJDK代码被加进了Android中：

```
Initial import of OpenJdk files.
Create new libcore/ojluni directory
with src/main/java and src/main/native
subdirectiories.
Build ojluni into core-oj jar.
Use openjdk classes from java.awt.font
```

package.
Copy all files from jdk/src/share/classes and jdk/src/solaris/classes directories in openjdk into libcore/ojluni/src/main/java.
Copy following native files from openjdk to libcore/ojluni/src/main/native: [long list of files]

Google 一直希望 Android 开发者能够接受这些改变，因为它在开发 apps 时有助于简化代码——使用单一共同的 Java API 代码库而非使用多代码库。这些原因可能是真实的，但并非完全转向 OpenJDK 的全部原因，如果是这样的话几年前 Google 早就这么干了。当 Google 发言人被问到为什么是现在，Google 指出是去年发布的 Java 8 和 Java 语言的一些新特性例如 lambdas。Google 想要为 OpenJDK 投入更多资源，这样团队就能对新特性和技术改进有更大的影响力和发言权。

当然这其中还涉及大量的版权问题，代码的提交是否意味着 Oracle 和 Google 之间关于 Java APIs 的法律诉讼是否已经庭外和解，由于 Oracle 的诉讼还在进行，Google 对于代码提交和诉讼是否有关拒绝作出评论。2010 年 1 月 Oracle 收购 Sun 之后，Oracle 在 2010 年 8 月起诉 Google 的版权和专利侵权，认为 Android 在未经授权的情况下使用了 Java API。Google 反驳称，APIs 不受版权保护，因为它对于软件开发、协作和创新是必不可少的。在 2012 年 5 月，一个陪审团认为 Java 的 API 不受版权保护，Google 对 Oracle 的专利侵犯不成立。2014 年 5 月，联邦巡回上诉法院部分逆转了区法院的判决，认定 Java API 受版权保护。就在 2015 年 6 月，美国最高法院拒绝审理此案，案件发回下级法院继续审理。在这些与 Oracle 的对决之后，Google 已经决定彻底拥抱 OpenJDK 了么？不管怎么样，结局是确定的：Android 未来的版本将基于 OpenJDK 而非 Oracle 专有的 JDK 版本。

不管怎么样，案件还未结束，Google 也无法改变现有的 Android 版本，业界人士对此案的裁决异常关注，因为这将对软件开发产生巨大的影响。如果 Oracle 胜诉，开发者基于现有应用和服务开发新的软件都将产生版权问题。如果 Google 胜了，APIs 的使用将不会受到版权保护。

谈谈那令你永生难忘的编程之路

作者 Abel Avram　译者 薛良

关于作者

Abel Avram，自从 2008 年起，就参与过多起由 InfoQ 组织的活动，热爱撰写关于移动、HTML、.NET、云计算、EA 和其他类型的新闻报道。他也是《Domain-Driven Design Quickly》联合作者。

在 2015 年 7 月份进行的一项名为“从个人特质、个人倾向和个人偏好的角度重新描述年轻时的自己”的调查中，有超过 2200 名程序员和开发者踊跃参与到其中，发起人说，主要是想了解这些人在年轻的时候，是什么因素引导他们最终选择了计算机科学专业。这项由 Code School 委托发起的活动最后得出的结论是：这些程序员和开发者平均在 16 岁的时候就已经对计算机产生了浓厚的兴趣了。除此之外，还有一些别的发现：

- 一般男性在 15 岁或之前就已经开始接触电脑了，而女性则在 16 岁或更晚。
- 被调查人群中，占比例最高的 83% 男性最爱仍然是电脑，其次是 61% 的人最爱是体育，59% 的男性最爱是音乐。女性调查者中 63% 的最爱是音乐，而喜欢计算机的只有 52%。

- 女性当中很少有人拖延，而男性中有 41% 的人会等到最后一分钟才做作业。
- 女性中只有 7% 的人会辍学，是男性的一半。此外，多数女性能够获得学士学位（51%）或研究生学位（30%），相比之下，男性只有 27% 和 42% 的人能获得上述学位。
- 谈到收入这个敏感话题，32% 的女性更倾向于 5-10 万美元的稳定年薪，只有 17% 的女性朝着 10+ 万美元年薪努力。而在男性方面则出现了两极分化现象，25% 的人年薪超过 10 万美元，而 20% 的人年薪少于 2.5 万美元。

随后，InfoQ 网站也做了一个类似的调查：您是如何开始编码的？这个调查主要是通过社会化媒体渠道在读者中进行推广，包含三个问题：什么时候开始编码的？你的第一台电脑是什么？你喜欢编码吗？最后根据 120 多名参与者的反馈得出了一些结论：

- InfoQ 读者群中开始编码的年龄段在 5-30 岁之间。
- 45% 的回复者开始编码是在 10-14 岁之间。
- 总体上的编码年龄段在 14 岁。
- InfoQ 的读者开始编码年龄 5 和 30 之间。
- 36% 的年轻读者刚开始工作都是在 Windows 上，19% 相对年纪大一点的程序员使用的是 Commogore，14% 的开发者使用的是 Spectrum，只有少数的 5% 在大型机上工作。
- 当谈及喜欢编程的原因时，可以精简为：实现自己的创造力（35%），解决实际问题（16%），被编程的神奇所吸引（15%）。有的人从事编程完全是因为兴趣，而有的人从 30 岁开始编程则是为了赚钱生活。

InfoQ 的读者中有很多都是经验丰富的软件架构师、研发团队领导或自由职业者。我们通过邮件采访的形式和他们进行了沟通，并将他们的在早期是如何开始编码的故事呈现给大家。

Ben Evans，jClarity 联合创始人

在我 8 岁生日的时候，父母给我买的生日礼物就是 ZX Spectrum（上世纪 80 年代一款标志性的计算机）。我父亲在上世纪 60 年代编写了 IBM S/360 系统的程序。也正是在我 8 岁的那一年，为了生活的更好，举家搬到了 Cornwall（英国康沃尔郡），父亲成了一名电视工程师。他认为，由英国广播公司、Commodore 和 Spectrum 这三家公司发起的“家庭电脑”潮流是一个非常重要的风向标，有可能引领接下来的科技走向，所以他和我妈妈凑钱给我买一台电脑。我用这台电脑学会了 Spectrum BASIC 和 Z80 assembler，此外还订阅了一个包含程序清单的电脑杂志。很荣欣，Spectrum 在转移到 PC 端之前的几代版本我都使用过，只是后来由于钱的问题退而求其次买了 286 英镑的没有显卡的电脑，搭配上相当便宜的 Hercules 监控适配器，用起来还是相当不错的，也是一段很好的回忆。

在大学里学习了 PASCAL 高级程序设计语言和一些基础的 C 语言，足够去写一些简单的“Light Tracer”/Snake 克隆。但是后来我对 PASCAL 有些沮丧，而且我不想以这种技术来开始我的职业生涯，所以，我最后选择了学习数学。

在大学里，我侵入了连接大学和 JANET 之间的基层接入网络，所以我可以轻而易举的连接到早期基于 telnet 的聊天服务，这重新点燃了我对 C 语言的兴趣，尤其是当时第一次听到 Perl 编程语言。我说服我的导师在一个 UNIX 帐户上签字，并且有一个可以打到家里的拨入号码，也就是在第一学期结束时，我发现我父亲的新工作是用远程方式来减少文书工作的，每天工作结束时都要在 Windows 3.1 版本的笔记本上下载字段调用。

后来我学会了 Dijkstra 算法并掌握了足够的图形理论知识，在班级里面保持领先水平。学

期末，Dijkstra 问我是否愿意到另一个班级听一种新的编程语言——Java。

首次品尝 Java 之后，研究生办公室问我为什么不选择更高薪水的编程工作，而到学校里任教。后来经过思考，我决定到公司里做一些跟 Perl 和 JavaScript 相关的网站工作。实际上，在那个时候我也开始与一些朋友搞在线音乐杂志的事情。在当时（1997-1998）这也是一件很新奇的东西，所以很多唱片公司乐意送我们黑胶唱片和音乐会门票。从运作杂志和运行网站程序当中获得了很多乐趣，尤其是在这其中，我的乐趣给我带来了经济收益。

Charles Humble，InfoQ 首席编辑

记得我 9 岁那年的一个暑假，我借一个朋友的 ZX Spectrum 开始编程，想想自己那个时候真是好奇心很重啊。于是在我 10 岁圣诞节的时候，父母给我买了一台 Commodore 64。虽然我当时特别想要一只小狗，但拿到这么先进的玩物后，还是很兴奋！不管怎样，这是我学的主要机器。

于是我开始学习 Commodore Basic，但刚开始的时候想用 Basic 语言在这个 Commodore 上写一个 Elite 克隆和一个高分辨率支持的克隆简直是太慢了，可以说是出奇的慢啊！所以我放弃了编写 assembler，坦率的说，Commodore Basic 汇编程序在移动编码组件上很稳定，基本上没什么大的跳跃。我的 Elite 克隆是最后废掉了，但它有一个很奇怪的声音轨道。Commodore 64 也有一个惊人的音效芯片，通过设计，振荡器的输出信号甚至从未停止过。这算得上是一个设计缺陷。

在学校学习的时候有一个 BBC 的模型，我经常在上面编写字节代码。后来有一个基于 RISC 系统的 Acorn Archimedes 计算机也为我的编程提供了很多便利，（RISC 是 Acorn Computers 为 Archimedes 和 RISC 计算机开发的操作系统，由于存储在 ROM 中所以启动只需几秒钟）。紧随其后，在学校里就能够接触到 Windows、C 和 C++ 语言了，而我的第一份编码工作就是为一家出版公司在 Excel 和 Access 里编写 VBA。Java 在银行里可谓会最好的归宿。

Abraham Marín Pérez，Java 程序员、敏捷开发拥趸者

我 8 岁的时候，父亲在办公室当会计，有的时候星期六早上也去工作，所以我有机会跟他一起去办公室用电脑。当时的电脑还用 5”1/4 软盘，其中有一些游戏。我当时最喜欢的程序是 AccuType，学习触摸式打字（touch-type），这意味着我在 9 岁之前就可以不看键盘打字了。

后来我的表弟有了一台 Amstrad 计算机，其实就是一块扁平的键盘，需要用一个看上去很古怪的软盘跟电视连接起来用。这里面有一个可以学习 BASIC 程序的书，只有 9 岁的我们就按照这本给专业程序员制定的书自学编程。我们只能一边看书一边拾取代码的随机样本，并复制下来看看都是什么功能：有时会画一个圆，有时会打印一系列数字……直到有一天我复制了一段在屏幕上绘制矩形的代码，然后写下书中的另一段代码，一个长方形出现了，这让我到，两个代码段基本相同，唯一的区别就是数字变化了。接着开始写三分之一代码块，只把数字改变，就能出现不同的形状。虽然那个时候根本不知道里面的深奥秘密，但是我心里清楚，不管要什么形状，都必须准确的告诉计算机。

最终我有了属于自己的电脑，一个 100MHz 顶配的 generic i486，400MB 硬盘和 4MB 内存。那个时候就在开始在学校学习 BASIC 编程，这是我当时最大的乐趣，我还记得写出来的第一块代码（或更确切的说是脚本）是一个批量脚本：我当时把一些游戏存储在硬盘上，然后我老爸说游戏太占内存，要删除。于是我自己写

脚本将这些游戏压缩、解压缩、运行。只记得这是在 MS-DOS 上完成的，现在想起来还是很有成就感的。

另外一件感觉很有意义的但是很简单的事情就是，帮我老爸的电脑设置个人权限。因为当时的电脑就像是一台电视或VCR，只能打开、使用，所有程序都是基于 MS-DOS，为了不让别的同事随意使用它，我就帮我老爸写了一个简单的程序，用 BASIC 编了一个密码，这样就只能给有密码的人使用。

奇怪的是，虽然我的第一份工作是跟 Flash 和 ActionScript 相关的编程工作，但我对这些工作内容没什么感觉。我不喜欢编程的那种感觉，更喜欢画一些框框，把代码片段嵌入进去。随后的工作是跟一个“将 COBOL 翻译成 Java 语言”相关的代码库，我称它为 Javol。也正是这份工作的需要，我不得不逼迫自己学习一些 COBOL 知识，这样才能正确理解运作原理。之后我到了一家银行工作，并施展了我的各种编程能力：将 Excel 表格与数据库相连；VBA 和手写的 XML 解析器连接；用 C# 将 VBA 代码嵌入 Word 文档等等。这份工作我一直做到现在。

Ralph Winzinger，Senacor Technologies 首席架构师

其实我小的时候也是一个乡村里土包子，我在 12 岁第一次看到电脑的时候，电脑已经可以放到桌子上了，一个朋友拿来一个相当经典的 Commodore C64。刚开始的时候只是为了打游戏，不久就开始自己写游戏了，虽然不知道怎么编写智能的软件程序，但是我们尝试写硬编码的文字冒险。

后来搬家弄丢了我的 Commdoore C64，却收获了一台老旧的 ZX81 Spectrum——一个带有塑料膜键盘的微型电脑。于是我照着维基百科上的讲解做出了一个乌龟图形和别的东西。

在我 15 岁的时候，我终于有了我自己电脑——一台带有 1571 个软盘驱动器的 Commodore C128。之后我就花了很对时间开始创建一个可以管理音频磁带的软件，这是我创建的第一个大型系统。

在上大学期间，我一边学习 Java 语言一边打理我的日常业务。和大多数学生一样，靠编程赚钱。而在当时，我主要是为掌上电脑编写程序，这也是我跟移动世界联系的收个入口。从那以后我就迷上了 WAP 手机，尤其是在我把移动方案应用在家乡公共交通规划中的时候。

事实上，无论开发者从什么年龄段开始从事编程，也不管开始学习编程之前是做什么工作的，只要他们敲下第一行代码，就证明这个人是很有激情的，至少他们愿意花大量的时间学习语法，锻炼编码技能，并为这个世界创造优秀的软件。

微服务与持续交付

作者 王磊

编者按：InfoQ 开设新栏目“品味书香”， 精选技术书籍的精彩章节，以及分享看完书留下的思考和收获，欢迎大家关注。本文节选自王磊著《微服务架构与实践》中的章节“微服务与持续交付”，介绍了持续交付是什么，以及微服务如何做到持续交付。

十年以前，软件在一年之内的交付次数屈指可数。

过去的十年间，交付的过程一直被不断地优化和改进。从早期的 RUP 模型、敏捷、XP、Scrum，再到近几年的精益创业、DevOps，都力求能更有效地降低交付过程所耗费的成本并提高效率，从而尽早实现软件的价值。

持续交付是一种软件开发策略，用于优化软件交付的流程，以尽快得到高质量、有价值的软件。这种方法能帮助组织更快地验证业务想法，并通过快速迭代的方式持续为用户提供价值。对于任何一个可交付的软件来说，必然要经历分析、设计、开发、测试、构建、部署、运维的过程。而从持续交付的角度来分析，对于任何一个可部署的独立单元，它都应该有一套独立的交付机制，来有效支撑其开发、测试、构建、部署与运维的整个过程。

如第 2 章所述，微服务将一个应用拆分成多个独立的服务，每个服务都具有业务属性，并且能独立地被开发、测试、构建、部署。换句话说，每个服务都是一个可交付的“系统”。那么在

这种细粒度的情况下，如何有效保障每个服务的交付效率，快速实现其业务价值呢？

本文，我们就来探讨微服务与持续交付。本文的内容主要包括：

- 什么是持续交付
- 持续交付的核心
- 微服务与持续交付

从技术上讲，持续交付是软件系统的构建、部署、测试、审核、发布过程的一种自动化实现，而其中的核心则是部署流水线。因为部署流水线能够将这几个环节有效地连接起来。当然，像探索性测试、易用性测试，以及管理人员的审批流程等还是需要一定的手工操作，如图 1 所示。

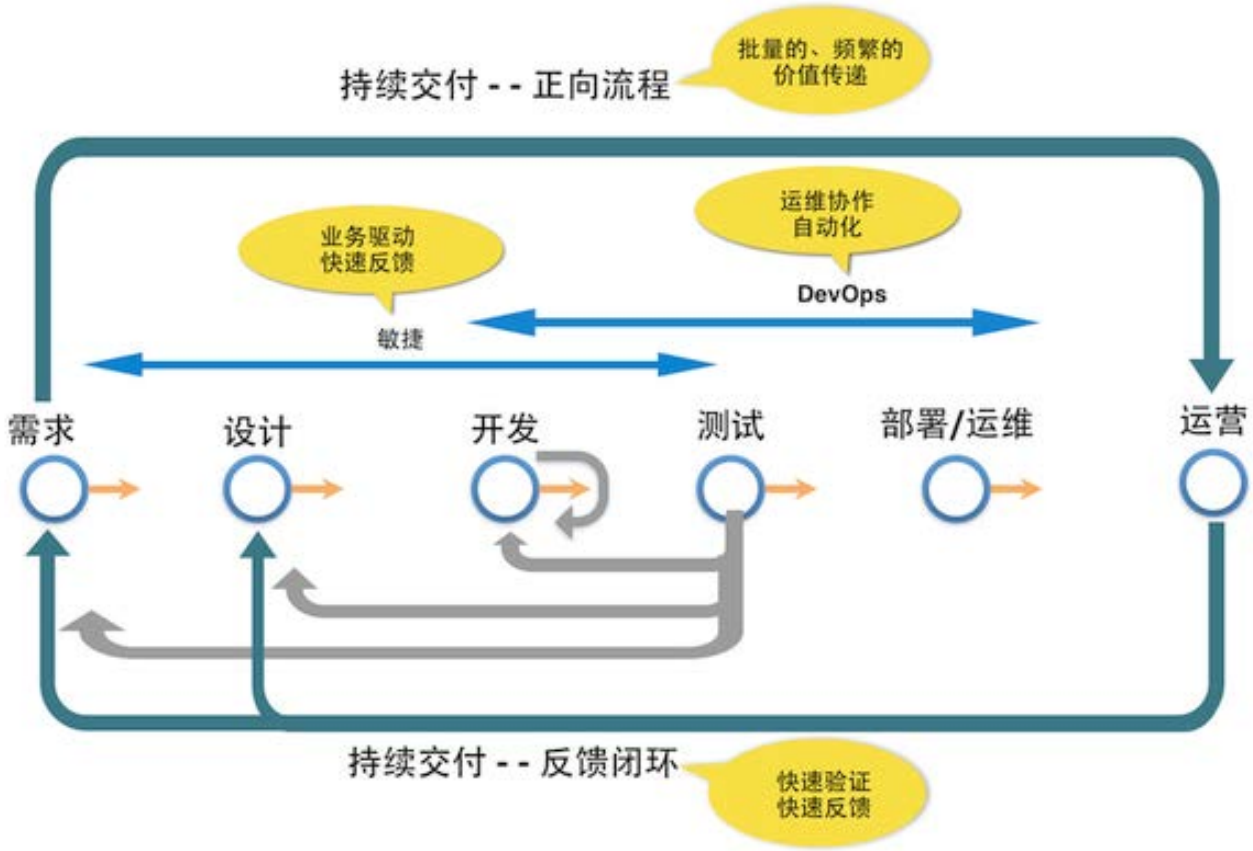


图 1　持续交付

1　持续交付的核心

在持续交付过程中，需求以小批量形式在团队的各个角色间顺畅流动，并以较短的周期完成小粒度的频繁发布。实际上，频繁的交付不仅能持续为用户提供价值，而且能产生快速的反馈，帮助业务人员制定更好的发布策略。

因此，持续交付的核心在于三个字：小、频、快。

- 小批量价值流动

通过建立自动化的构建及部署机制，将业务功能以小批量的方式，从需求产生端移动到用户端。

- 频繁可发布

通过建立自动化的构建及部署机制，将小批量的业务功能频繁地从需求产生端移动到用户端，持续地交付价值。

• 快速反馈

通过建立高效的反馈机制，快速验证需求是否有效。同时根据反馈，及时指导业务团队并调整策略，优先为用户交付高价值的功能。

持续交付让业务功能在整个软件交付过程中以小批量方式在各角色间顺畅流动，通过更频繁的、低风险的发布快速获得用户反馈，以此来持续达成业务目标。

2 微服务架构与持续交付

从交付的角度来分析，对于任何一个可部署的独立单元，它都应该有一套独立的部署流水线，来有效支撑其开发、测试、构建、部署与运维的整个过程。

在微服务架构中，由于每个服务都是一个独立的、可部署的业务单元，因此，每个服务也应该对应着一套独立的持续交付流水线，可谓是“麻雀虽小，五脏俱全”。

接下来，让我们看看在微服务的架构中，如果构建这样一套持续交付的流水线，各个环节需要做什么样的准备。

2.1 开发

对于微服务架构而言，如果希望构建独立的持续交付流水线，我们在开发阶段应该尽量做到如下几点。

• 独立代码库

对于每一个服务而言，其代码库和其他服务的代码库在物理上应该是隔离的。所谓物理隔离，是指代码库本身互不干扰，不同的服务有不同的代码库访问地址。譬如，对于我们平时使用的 SVN、GIT 等工具，每个服务都对应且只对应一个独立的代码库 URL。如下所示，分别表示产品信息服务和客户信息服务的代码库。

http://github.com/xxxxx/products-service
http://github.com/xxxxx/customers-service

除此之外，对不同服务隔离代码库的另一个好处在于，对某服务的代码进行修改，完全不用担心影响其他服务代码库中的代码，在很大程度上避免了修改一处，导致多处发生缺陷的情况。

• 服务说明文件

对于每一个服务而言，都应有一个清晰的服务说明，描述当前服务的信息，同时帮助团队更快地理解并快速上手。譬如，在笔者的微服务实践过程中，对于每一个代码库，其服务说明都包括如下几个部分。

1. 服务介绍

- 服务提供什么功能，譬如产品服务主要提供产品数据的获取或者存储。
- 谁是服务的消费者。譬如产品服务的消费者为电商的前端网站系统或者 CRM 系统。

2. 服务维护者

- 挑选 1~2 个团队的成员，作为服务的负责人，登记其姓名、电子邮件、电话等联系方式，以便其他团队遇到问题能及时找到服务的负责人。

3. 服务可用期

- 服务可用周期，如 7×24 小时，或周一 ~ 周五（7:00~19:00）等。
- 可用率，可用率是指服务可以正常访问的时间占总时间的百分比，如 99.9% 或者 99%。如果服务一天内都可以访问，则服务当天的可用率为 100%。如果服务有 3 分钟访问中断，而一天共有 1440 分钟，

那么服务的可用率为：((1440 - 3) / 1440) * 100%，也就是 99.79%。

- 响应时间，指服务返回数据的可接受响应时间。譬如为 0.5~1 秒。

4. 定义环境，描述服务运行的具体环境，通常包括：

- 生产环境
- 类生产环境
- 测试环境

5. 开发，描述开发相关的信息，通常包括：

- 如何搭建开发环境
- 如何运行服务
- 如何定位问题

6. 测试，描述测试相关的信息，通常包括：

- 测试策略
- 如何运行测试
- 如何查看测试的统计结果，譬如测试覆盖率、运行时间、性能等。

7. 构建，描述持续集成以及构建相关的信息，通常包括：

- 持续集成访问的 URL
- 持续集成的流程描述
- 构建后的部署包

8. 部署，描述部署相关的信息，通常包括：

- 如何部署到不同环境
- 部署后的功能验证

9. 运维，描述运维相关的信息，通常包括：

- 日志聚合的访问
- 告警信息的访问
- 监控信息的访问

- 代码所有权归团队

团队的任何成员都能向代码库提交代码，做到任何服务代码的所有权归团队。

代码所有权归团队，它表现的更多的是团队协同工作的观念，即集体工作的价值大于每个个体生产价值的总和。当所有权属于集体的时候，那么每个开发者就不应当出于个人原因来降低代码质量。代码质量上出现的问题应该在整个团队的努力下共同处理。

相反，如果某段代码背后的业务知识没有适当地分享给其他人，那么代码的演变逐渐变为依赖于具体的某个人，瓶颈也就由此而产生。

- 有效的代码版本管理工具

代码版本管理工具的使用早已经成为开发者必备的核心技能之一，譬如 Git、Mercurial，以及 CVCS（Centralized Version Control System）等。不过，团队最好能使用分布式版本控制工具（DVCS，Distributed Version Control System），它可以避免由于客户端不能连接服务器所带来的无法提交代码的问题。

- 代码静态检查工具

另外，团队也需要有代码静态检查工具，帮助完成代码的静态检查。譬如 Java 语言的 CheckStyle、Ruby 语言的 Rubocop 等。

另外，代码度量（Code Metrics）工具，譬如常用的 SonarQube、Ruby 的 Cane 等，能够保障团队内部代码的一致性和可维护性。

- 易于本地运行

作为团队的开发人员，当我们从代码库检出（Check out）某服务的代码后，应该花很短的时间、很低的成本就能在本地环境中将服务

运行起来。如果依赖于外部资源，并且构建、使用成本较高，就应该考虑采取其他打桩的机制来模拟这些外部资源。这类外部资源通常指数据库、云存储、缓存或者第三方系统等。

譬如，笔者最近参与的一个企业内部系统改造项目，使用了 OKTA 集成单点登录的功能。开发环境下当然也可以使用 OKTA，但由于网络、安全、审批等多种原因，极大影响了开发人员在本地环境访问 OKTA 的效率。最后，团队采用打桩的机制，构建了一套符合 OKTA 协议的模拟 OKTA，在本地使用。在开发环境下，通过加载这个模拟的 OKTA，有效地解决了本地访问 OKTA 时间较长的问题。

另外一个例子是，笔者在系统中使用了 AWS 的 S3 服务。由于权限、网络等多种因素的存在，本地开发时使用 S3 的成本非常高，因此就构建了一套模拟的 S3 环境。当服务运行在开发环境时，加载开发模式的环境变量，访问本地的 Mock S3 环境；而在生产环境，则使用生产模式的 S3 地址。在不改变任何代码的前提下，帮助团队快速在本地搭建运行环境并演示，极大地提高了开发效率，如图 2 所示。

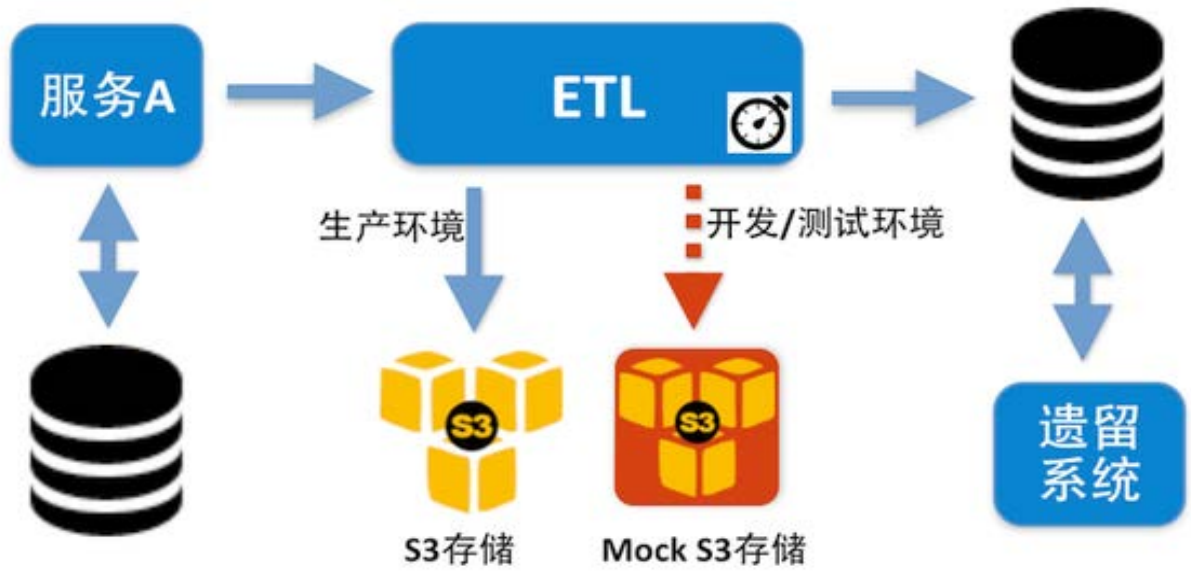


图 2　模拟 S3 存储

2.2　测试

对于微服务架构，如果希望构建独立的持续交付流水线，我们在测试方面应该注意以下几点。

• 集成测试的二义性

对于任何一个服务而言，单元测试必不可少。但是否需要集成测试，团队可以根据喜好自行决定。笔者个人建议明确定义集成测试的范围，因为“集成”这个词，很难有一个准确的度量机制。到底什么样的组合才叫集成？其可以是对外部不同系统之间的测试组合，也可以是系统内部实现逻辑、类与类之间调用的组合，因此“集成测试”这个术语，在团队和组织内部，容易在沟通过程中产生误解。

• Mock 与 Stub

对于单元测试而言，我们可以使用 Mock 框架帮助我们完成对依赖的模拟（Mock）或者打桩（Stub），譬如 Java 的 Mockito、Ruby 的 RSpec 等。当然，如果对象之间的依赖构建成本不高，也可以使用真实的调用关系而非 Mock

或者 Stub 机制。关于 Mock 和 Stub 的区别，有兴趣的读者可以参考 ThoughtWorks 首席科学家马丁·福勒的 Mocks Aren't Stubs 这篇文章。

• 接口测试

除了单元测试覆盖代码逻辑外，至少还应该有接口测试来覆盖服务的接口部分。注意，对于服务的接口测试而言，更关注的是接口部分。譬如，作为数据的生产者，接口测试需要确保其提供的数据能够符合消费者的要求。作为数据的消费者，接口测试需要确保，从生产者获取数据后，能够有效地被处理。另外，对于服务与服务之间的交互过程，最好能设计成无状态的。

• 测试的有效性

如果单元测试的覆盖率够高，接口测试能有效覆盖服务的接口，那么基本上测试机制就保障了服务所负责的业务逻辑以及和外部交互的正确性。

有些朋友可能会存在疑问，是否需要使用行为测试的框架，譬如像 Cucumber、JBehave 等的工具，基于不同的场景，做一些类似用户行为的测试呢？

实际上，这里并没有确定的答案。在笔者参与的项目中，通常都是在最外层做一部分行为测试，原因有以下几点。

1. 通常我们所说的服务，大多是不涉及用户体验部分的。也就是说，作为服务，更关注的是数据的改变，而不是同用户的交互过程。譬如，当我们从电商网站挑选某件商品、下单，一个新的订单就生成了。这时候，订单的状态可能是“新建”。随后，当完成付款时，那么订单的状态可能会被更新成“已支付”。如果忽略状态更新的实现流程，譬如同步更新、异步更新等，那么从服务的角度而言，其并不在意用户到底是从 PC 端的浏览器，还是手机上的 APP 来完成订单的支付，服务自身只完成了一件事：就是完成订单状态从“新建”到“已支付”的更新。

2. 服务作为整个应用的一部分，能够独立存在，那必然有其对应的边界条件。前面提到，单元测试保障内部逻辑，接口测试保障接口，在这样的前提下，服务的正确性和有效性在大部分情况下已经得到了验证。

3. 从经典的测试金字塔来看，越是偏向于用户场景、行为的测试，其成本越高，反馈的周期也越长；相反，越是接近代码级别的测试，成本越低，反馈周期也相对较短，如图 3 所示。

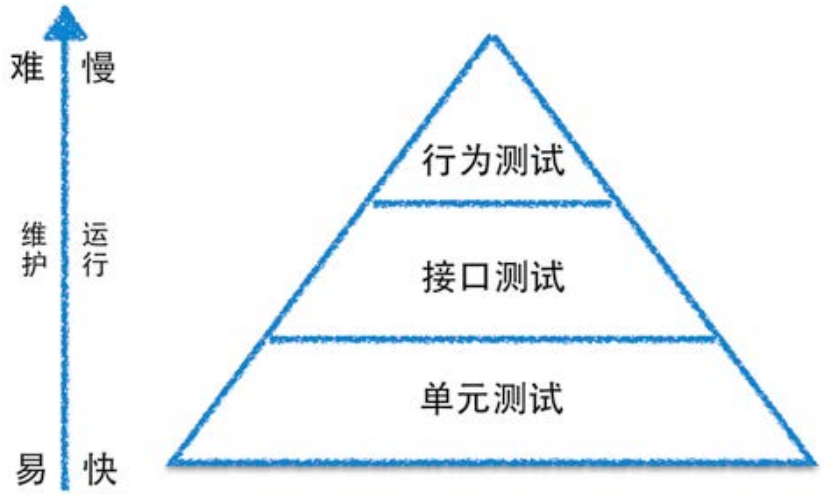


图 3　测试金字塔

2.3 持续集成

持续集成经过多年的发展，已成为系统构建过程中众所周知的最佳实践之一。对于每个独立的、可部署的服务而言，应为其建立一套持续集成的环境（Continuous Integration Project）。

当团队成员向服务的代码库提交代码后，配置好的持续集成工程会通过定期刷新或者WebHook的方式检测到代码变化，触发并执行之前开发阶段定义的静态检查、代码度量、测试以及完成构建的步骤，如图4所示。

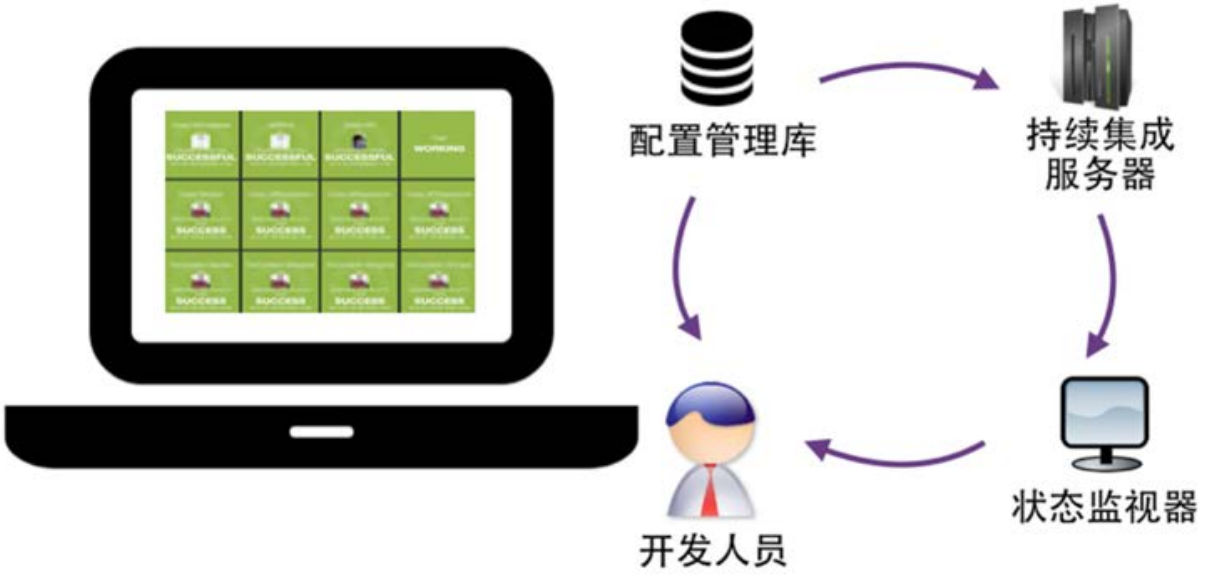


图4 持续集成

常用的企业级持续集成服务器有Jenkins、Bamboo以及GO等，在线的持续集成平台有Travis-CI、Snap-CI等。

更多关于持续集成的细节，请参考ThoughtWorks首席科学家马丁·福勒的这篇文章 http://www.martinfowler.com/articles/continuousIntegration.html。

2.4 构建

每个服务都是一个可独立部署的业务单元，经过静态检查、代码度量、单元测试、接口测试等阶段后，构建符合需求的部署包。

部署包存在的形式是多种多样的，可以是deb包、rpm包，能在不同UNIX操作系统平台直接安装；也可以是zip包、war包等，只需将其复制到指定的目录下，执行某些命令，就可以工作。当然，也有可能是基于某特定的IAAS平台，譬如亚马逊的AMI，我们称之为映像包（Image）。

另外，作为容器化虚拟技术的代表，Docker（一个开源的Linux容器）的出现，允许开发者将应用以及依赖包打包到一个可移植的Docker容器中，然后发布到任何装有Docker的Linux机器上。

通过使用Docker，我们可以方便地构建基于Docker的部署镜像包。

2.5 部署

对于每个独立的服务而言，如果希望构建独立的持续交付流水线，需要选择部署环境并制定合适的部署方式来完成部署。通常，我们可以从如下两个维度考虑如何进行部署。

1. 部署环境

• 基于云平台

我们知道，云平台是一个很广的概念，其中主要包括 IAAS、PAAS 和 SAAS 三层。当基于云平台部署的时候，要先分清楚部署的环境，即部署将发生在哪一层。由于 SAAS 层是相对于应用的使用者而言，软件即服务。即对使用者而言，不需要再去考虑本地安装、数据维护等因素，直接通过在线的方式享受服务，这和我们讨论的部署环境没有关系。因此，这里我们主要讨论 IAAS 层和 PAAS 层的部署。另外，笔者这里没有区分是公有云还是私有云。公有云指运行在 Internet 上的云服务，私有云则通常指运行在企业内部 Intranet 的云服务。

• 基于 IAAS 层

云平台的 IAAS 层，通常包括运行服务的基础资源，譬如计算节点、网络、负载均衡器、防火墙等。因此，对于该层的部署包而言，实际上应该是一个操作系统映像，映像里包含运行服务所需要的基本环境，譬如 JVM 环境、Tomcat 服务器、Ruby 环境或者 Passenger 配置等。当在 IAAS 层部署服务时，不仅可以使用映像创建新的节点，也可以创建其他系统相关的资源，譬如负载均衡器、自动伸缩监控器、防火墙、分布式缓存等。

• 基于 PAAS 层

PAAS 层并不关心基础资源的管理，它更关注的是服务或者应用本身。因此，对于该层的部署包而言，通常是能直接在 UNIX 操作系统安装的二进制包（譬如 deb 包或者 rpm 包等），或者是压缩包（譬如 zip 包、tar 包、jar 包或者 war 包等），将其复制到指定目录下解压缩，启动相关容器，就可以工作。

除此之外，也可以使用 PAAS 平台提供的工具或者 SDK，直接对当前的代码进行部署。譬如 Heroku 提供的命令行，就能很方便地将 Java、Ruby、NodeJS 等代码部署到指定的环境中。

• 基于数据中心

云平台已经成为大家公认的未来趋势之一，但是对于很多传统的企业，由于组织或者企业内部多年业务、数据的积累，以及组织架构、团队、流程固化等原因，无法从现有数据中心一步迁移到云端。而且，针对传统的数据中心，其对应的环境通常比较复杂，既没有 IAAS 那种按需创建资源的灵活性，也没有 PAAS 这种资源能够被自动化调配的可伸缩性。这时候，对于数据中心而言，部署就相对较麻烦，需要投入更多的成本构建环境以及调配资源。

很多企业也开始尝试在数据中心的节点上创建虚拟机（譬如 VMware、Xen 等），以帮助简化资源的创建以及调配。

• 基于容器技术

容器技术，是一种利用容器（Container）实现虚拟化的方式。同传统的虚拟化方式不同的是，容器技术并不是一套完全的硬件虚拟化方法，它无法归属到全虚拟化、部分虚拟化和半虚拟化中的任意一个，它是一个操作系统级的虚拟化方法，能为用户提供更多的资源。

过去两年里，Docker 的快速发展使其成为容器技术的典型代表。Docker 可以运行在任意平台上，包括物理机、虚拟机、公有云、私有云、服务器等，这种兼容性使我们不用担心生产环境的操作系统或者平台的差异性，能够很方便地将 Docker 的映像部署到任何运行 Docker 的环境中。

2. 部署方式

部署方式，是指通过什么样的方法将服务有效地部署到相应的环境。对于服务而言，由于部署环境的不同，采用的部署方式自然也不同，如图 5 所示。

图 5　部署方式的演变

• 手动部署

对于传统的数据中心环境，考虑到资源有限以及安全性等因素，通常的部署方式都是使用 SSH 工具，登录到目标机上，下载需要的部署包，然后复制到指定的位置，最后重启服务。

• 脚本部署

由于部署团队每次都要手动下载、复制，不仅效率低，而且人为出错的概率也大。因此，很多企业和组织使用 Shell 脚本将这些下载、复制、重启等过程逐渐实现自动化，大幅提升了效率。Shell 脚本的优势是兼容性好，但其弊端在于实现功能所需要的代码量大，可读性也较差，时间长了不易维护。

• 基础设施部署自动化

随着业务的发展，很多组织以及团队逐渐发现，环境的安装和配置、应用或者服务的部署所耗费的成本越来越高。“基础设施自动化”这个概念的提出，正好有效地解决了这一类问题。于是，越来越多的组织开始尝试使用 Chef、Puppet、Ansible 等工具，完成软件的安装和配置，以及应用或者服务的部署。

• 应用部署自动化

无须过多的人工干预、一键触发即可完成部署的自动化方式可以说是任何组织，从业务、开发到运维都希望达成的目标。但说起来容易，实现起来却很难，而且这也不是一蹴而就的过程。需要随着组织或者企业在业务的演进过程中、技术的积累过程中，逐渐实现自动化。通常，应用部署的自动化主要包括以下两部分。

• 映像部署

私有云、公有云的出现，使得部署方式发生了显著的变化。面对的环境不一样，因此部署包也不一样。以前的 war 包、rpm 包，在基于 IAAS 的云平台上，都可以变成映像。譬如，对于亚马逊的 AWS 云环境，可以方便地使用其提供的系统映像（AMI）来完成部署。

基于映像的最显著优势在于，能够在应用需要扩容的时候，更有效、迅速地扩容。原因在于，映像本身已经包括了操作系统和应用运行所需要的所有依赖，启动即可提供服务。而基于 war、rpm 包等的部署，扩容时通常需要先启动同构的节点，安装依赖，之后才能部署具体的

war 包或者 rpm 包。

• 容器部署

利用容器技术，譬如 Docker，构建出的部署包也可以是一个映像。该映像能运行在任何装有 Docker 的环境中，有效地解决了开发与部署环境不一致的问题。同时，由于 Docker 是基于 Linux 容器的虚拟化技术，能够在一台机器上构建多个容器，因此也大大提高了节点的利用率。

所以，就微服务架构本身而言，如何有效地基于部署环境选择合适的部署方式，并最终完成自动化部署，是一个值得团队或者组织不断探讨和实践的过程。

2.6 运维

由于每个服务都是一个可以独立运行的业务单元，同时每个服务都运行在不同的独立节点上。因此，需要为服务建立独立的监控、告警、快速分析和定位问题的机制，我们将它们统一归纳为服务的运维。

• 监控

监控是整个运维环节中非常重要的一环。监控通常分为两类：系统监控与应用监控。系统监控关注服务运行所在节点的健康状况，譬如 CPU、内存、磁盘、网络等。应用监控则关注应用本身及其相关依赖的健康状况，譬如服务本身是否可用、其依赖的服务是否能正常访问等。

关于监控，目前业界已经有很多成熟的产品，譬如 Zabbix、NewRelic、Nagios 以及国内的 OneAPM 等。对于笔者参与的项目，服务节点的运行环境大都基于 AWS（使用 EC2、ELB 以及 ASG 等），因此使用 AWS 的 CloudWatch 作为系统监控工具的情况比较多。关于应用监控，通常使用 NewRelic 和 Nagios 作为监控工具。

• 告警

告警是运维环节另外一个非常重要的部分。我们知道，当系统出现异常时，通过监控能发现异常。这时候，通过合适的告警机制，则能及时、有效地通知相关责任人，做到早发现问题，早分析问题，早修复问题。由于每个服务都是独立的个体，因此针对不同的服务，都应该能提供有效的告警机制，确保当该服务出现异常时，能够准确有效地通知到相关责任人，并及时解决问题。

对于告警工具，业界较有名的是 PagerDuty，它支持多种提醒方式，譬如屏幕显示、电话呼叫、短信通知、电邮通知等，而且在无人应答时还会自动将提醒级别提高。除此之外，之前提到的常用的监控产品也能提供告警机制。

• 日志聚合

除此之外，日志聚合也是运维部分必不可少的一环。由于微服务架构本质上是基于分布式系统之上的软件应用架构方式，随着服务的增多、节点的增多，登录节点查看日志、分析日志的工作将会耗费更高的成本。通过日志聚合的方式，能有效将不同节点的日志聚合到集中的地方，便于分析和可视化。

目前，业界最著名的日志聚合工具是 Splunk 和 LogStash，不仅提供了有效的日志转发机制，还提供了很方便的报表和定制化视图。更多关于 Splunk 与 LogStash 的信息，请参考其官方网站。

3 小结

本文首先讲述了持续交付的概念及其核心，接着讨论了在微服务架构的实施过程中，如果建立基于服务的细粒度的持续交付流水线，应该考虑的因素。虽然微服务中的服务只是整个应用程序的一个业务单元，但作为一个可以独立发布、独立部署的个体，它必然也要遵循持续

交付的机制和流程，包括开发、测试、集成、部署以及运维等，可谓是“麻雀虽小，五脏俱全”。通过搭建稳定的持续交付流水线，能够帮助团队频繁、稳定地交付服务。

书籍介绍

本书首先从理论出发，介绍了微服务架构的概念、诞生背景、本质特征以及优缺点；然后基于实践，探讨了如何从零开始构建 ** 个微服务，包括 Hello World API、Docker 映像构建与部署、日志聚合、监控告警、持续交付流水线等；最后，在进阶部分讨论了微服务的轻量级通信、消费者驱动的契约测试，并通过一个真实的案例描述了 如何使用微服务架构改造遗留系统。全书内容丰富，条理清晰，通俗易懂，是一本理论结合实践的微服务架构的实用书籍。

本书不仅适合架构师、开发人员、测试人员以及运维人员阅读，也适合正在尝试使用微服务架构解耦历史遗留系统的团队或者个人参考，希望本书能在实际工作中对读者有所帮助。

梁胜：做云计算，如何才能超越AWS？

作者　郭蕾

在 12 月 18 日举行的 ArchSummit 北京 2015 大会上，Rancher Labs 创始人兼全球 CEO 梁胜分享了题为《容器时代的云计算》的主题演讲，在演讲中，梁胜博士分享了他对目前公有云以及私有云格局的思考，以及如何再打造下一家“AWS”。梁胜博士提到，公有云厂商差距这几年一直是在加大，根本原因就是因为各个云计算平台有很大的差异性，如果全世界都一样，那价格就会成为决定性因素，然后可能导致恶性竞争。之所以一家独大或者是几家独大，是因为云计算有不同的技术上的创新，可以不断的取得新的业务用户。本文根据其演讲整理而成，点此下载演讲 PPT。

梁胜博士现任美国 Rancher Labs Inc. 公司联合创始人及公司 CEO，梁博士是位标准的技术梦想家，亦是一位具有开拓精神的优秀创业者和企业家。在此之前，从 2011 年至 2014 年间，梁博士担任 Citrix System Inc. 公司云平台首席技术官，也是 Citrix 公司首位华人 CTO。在加入 Citrix 公司之前，梁胜创立了 cloud.com 公司并担任首席执行官，直至 2011 年 7 月被 Citrix 以 2.3 亿美金购入旗下。

公有云的发展以及趋势

现在讲云计算和容器，势必要讲到数据中心。毫无疑问，数据中心在全世界范围都是飞速成长，不仅是中国，而且是美国以及任何一个国家。现在一些传统厂商，很多都是全球的知名厂商（HP、NetApp、Cisco、EMC、IBM），但是他们业务都在不同程度上受到了挑战，有些虽然在增长，但成长势头明显比整个工业界都要缓慢。究其原因，我觉得只有一个，就是云

计算。从亚马逊在 2006 年发布 S3，到现在的一大批的云计算服务厂商，他们从不同程度上都满足了这些数据中心成长的需求，而且是减弱了对传统数据中心构建的一些产品成长的机会。

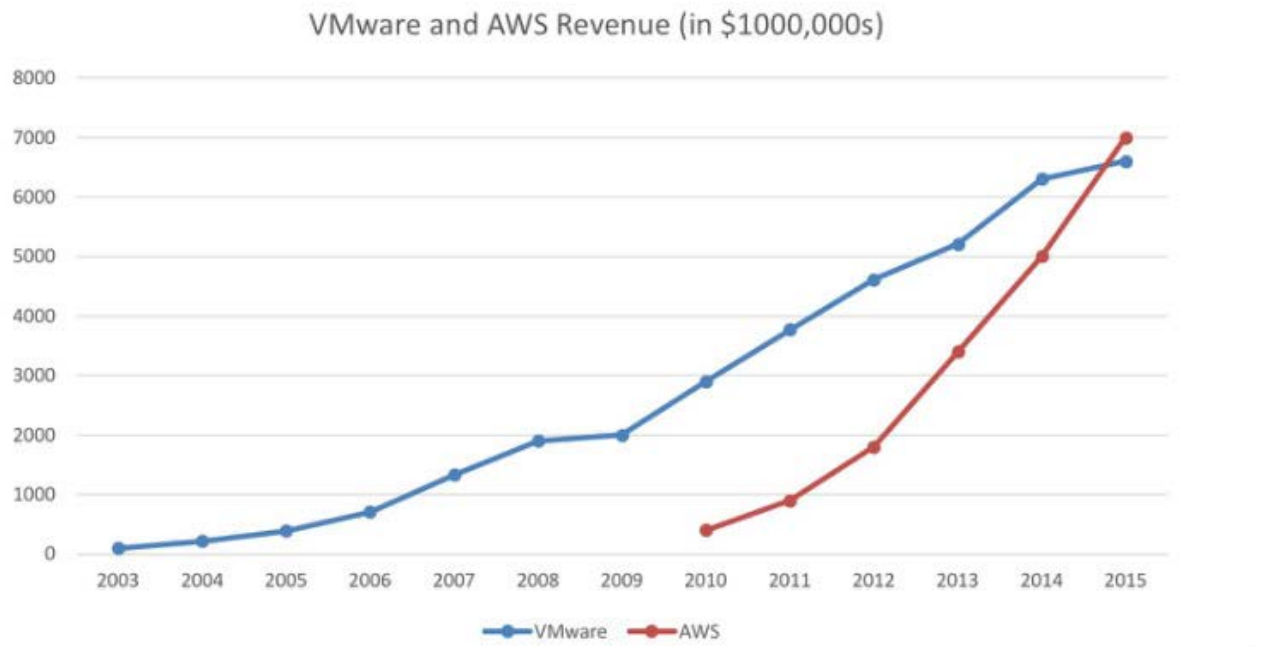
Company	Last report date	Quarterly revenue	YoY growth
HP	Oct 31 2015	$25.71B	-9.5%
NetApp	Nov 18 2015	$1.45B	-6.4%
Cisco	Nov 12 2015	$12.7	+4%
EMC	Oct 21 2015	$6.08B	+1%
IBM	Sep 20 2015	$19.29B	-14%

从 Gartner IaaS 魔力象限可以看到，全球最领先的是亚马逊，他们当之无愧，应该在很大程度上是创新最大的公司。还有一点值得注意，接下来在云计算领域会逐渐开始洗牌。这是什么原因呢？其实我们在国内外都可以看到，在云计算领域，市场领先的厂商和一般的厂商差距其实并不是缩小，而是在加大。2014 年 Gartner 公布的全世界 15 个最大的云计算厂商，亚马逊的体量比下面 14 个加起来 5 倍还要多，2015 年这个差距增长到了 10 倍。为什么会有这么大的差距？根本原因就是因为各个云计算平台有很大的差异性，当然如果全世界都一样，那价格就会成为决定性因素，然后可能导致恶性竞争。之所以一家独大或者是几家独大，是因为云计算有不同的技术上的创新，可以不断的取得新的业务用户。

私有云的现状

说起私有云，大家最先想到的就是 OpenStack，我记得从第一次参加 OpenStack 会议到现在，已经有 5 年半的时间。OpenStack 的发展也是经历了非常漫长的过程。外界对于 OpenStack 的评价，前几年过于乐观，现在有点过于悲观。但是毫无质疑，跟亚马逊比起来，OpenStack 确实不能算成功，我们想一下原因到底是什么，我们自己也做了很多反思，跟业界很多领袖也有很多交谈，特别是跟我们很多客户也有很多深入的讨论。用过 OpenStack 的人都知道，很多人认为 OpenStack 还不够成功，因为这个系统过于复杂，比较难用，这是大家比较普遍的认识。我觉得这确实有一定的道理。但 OpenStack 本身不是软件的问题，而是产品根本的问题。很多公司基于 OpenStack 做出来的产品，其实说到最后是用户不够，而不是说运维过于困难。我们做基础设施就像造房子，你可以造房子，容易造，你总是可以造得起来，造起来之后，房地产开发商还有另外一道最关键造房子得卖出去，卖出去后得有人搬进去，周边配套设施得起来。

云计算领域从全球来看，其实真正只有两个产品是达到了有人用，一个是刚刚说的亚马逊，另外一个是 vSphere。从下面的图中可以看出来，2015 年这两家公司的体量差不多。而现在 VMware 成长速度和亚马逊相比有些滞后，但亚马逊是服务型产品，所以 VMware 的利润还是比亚马逊高。



再说什么是 Docker

那如何与这两家公司竞争了？比较好的一个方案是我新造一个 VMware 和旧的 VMware 竞争，新造一个亚马逊跟亚马逊竞争。那新造的这家公司切入点应该是什么？我们看到最大的机会就是容器技术，就是 Docker。当然 Docker 也有竞争对手，但是从实际用户采纳角度来看，Docker 占用了很大的市场份额，它在过去两年的成长速度非常快。

我还是想讲一下什么是 Docker，因为最近我和国内一些朋友交流，我觉得这里面其实还是有一些误解，很多人还是把 Docker 与虚拟机相提并论，觉得它是轻量级的虚拟机，这一定程度上是对的。但是 Docker 并不是说要替代虚拟机，而是要替代进程。进程是可以跑在虚拟机或者物理机上的，所以是并存的。有的人问我说我是应该把 Docker 部署在虚拟机还是部署在物理机上，这问题和应该把 Java 程序部署在虚拟机还是物理机上一样。

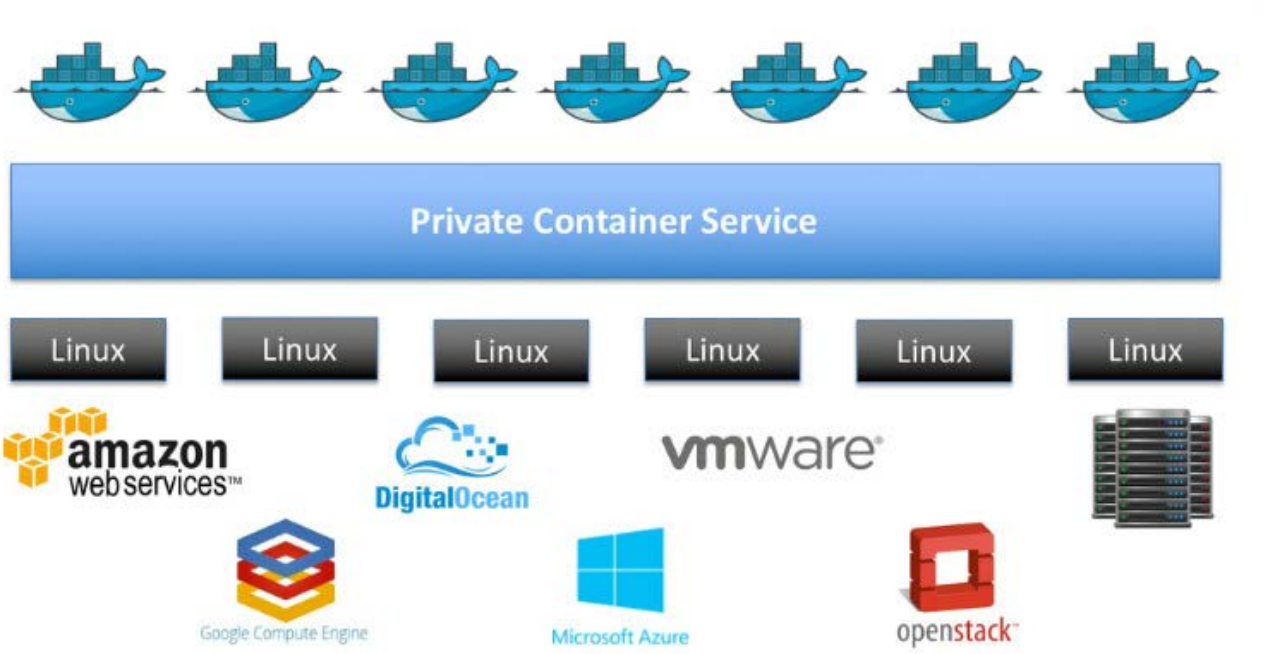
相比于进程，Docker 最关键的技术就是它的应用打包格式，类似 JAR、RPM 等。我看到绝大部分人用 Docker 都用的非常简单，这也是 Docker 厉害的地方，你不用采纳什么新的框架或者改变研发流程，只需要稍作调整就可以使用 Docker。比如说最近比较热的微服务架构，它非常好，我觉得也很适合和容器结合，但实际上我们看到，使用 Docker 的公司很多并不是采用的微服务架构。如果说必须是微服务架构才能用 Docker，那 Docker 的使用门槛就太高了。使用门槛低，这也是 Docker 的优势之一。

为什么是 Docker？

很多人都知道，容器的优点之一就是资源利用率高。但说到底，这并不是 Docker 容器最大的好处，也不是大部分企业采纳容器的主要原因。企业使用 Docker 最主要的原因还是研发人员。因为我刚刚说这是开发者最好的时代，研发人员现在在公司里面的地位确实是非常高，他们其实决定了公司用什么样的服务和用什么样的容器技术。从研发人员角度来讲，Docker 提升了他们的开发体验。

再就是基于 Docker 的私有容器云服务，这和原来的私有云不一样，你不一定要买机房买设

备，你可以运行在自己的机房设备上，也可以运行在公有云上。从公有云也好，私有云也好，拿到的就是资源，什么叫资源，资源就是Linux机器，有CPU，有存储，有网络，有磁盘，这是最好的资源。拿来之后，剩下的事情就是在私有容器云上自己做的。所以我刚刚讲为什么亚马逊一家独大，那是因为所有的基础设施云都不一样，但是从容器云角度来看所有基础设施都是一样，尽管亚马逊有一百一万个功能，但我也用不上。这也是我觉得容器云非常独特的地方。



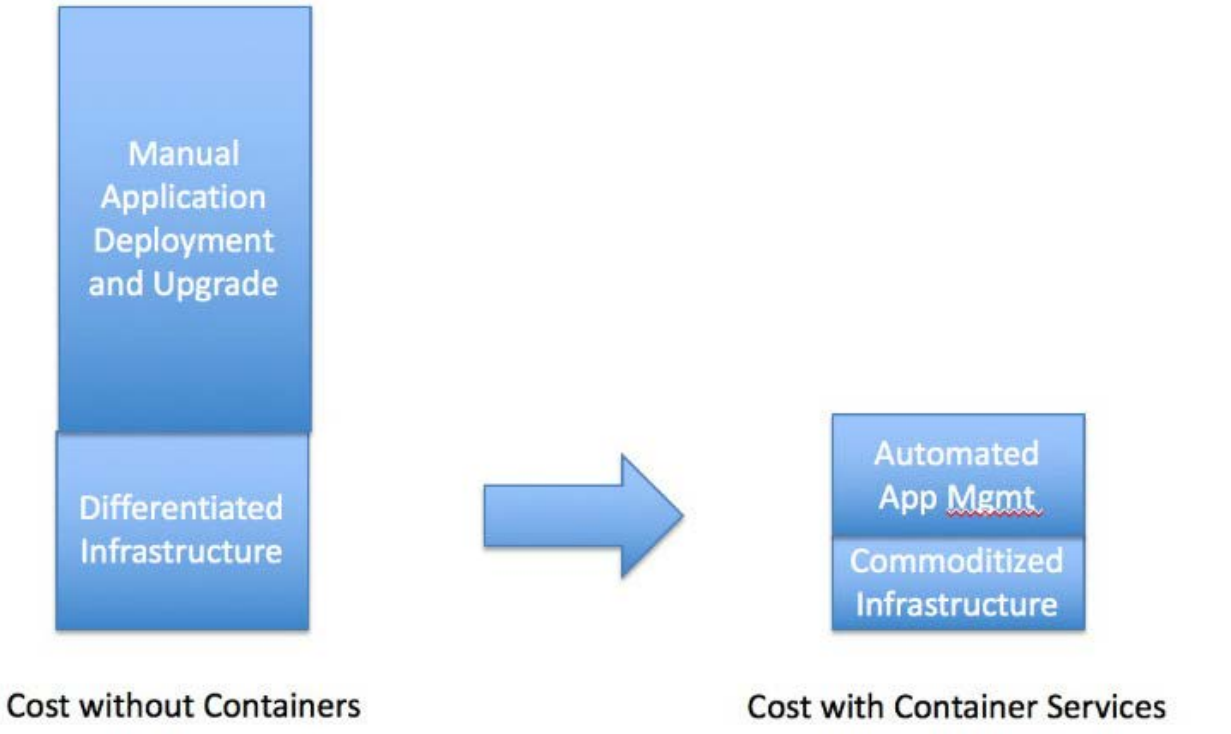
容器云

下面这张图是我总结的私有云容器技术栈：

Private Container Service	
Resource Mgmt	Docker Machine
Scheduling	Swarm, Kubernetes, Mesos
Orchestration	Compose, Kubernetes, Marathon
App Catalog	Helm, Rancher
Access Control	LDAP, AD, GitHub
Monitoring	Stats, Healthchecks, Sysdig, Datadog, etc.
Security	Docker, Kubernetes, Vault
Overlay network	VXLAN, IPSEC, etc.
Firewall	iptables
Load Balancer	HAProxy, ngnix, iptables
Block Storage	Ceph, EBS, SAN, etc.
Object Storage	Swift, S3, etc.
Distributed DB	etcd, consul, mongo, DynamoDB, etc.

还有一个比较有意思的事情，目前世界上第二大的主机服务公司是DigitalOcean，这个公司在最近几年的成长速度非常快，据说财富500强企业中，大约有100家公司使用了DigitalOcean的云技术服务。但DigitalOcean的功能非常简单，可以说砍掉亚马逊99%的功能剩下的就是DigitalOcean。它为什么能成长这么快？这个也没有什么答案或者说答案就在面前。但是就容器云的需求而言，DigitalOcean甚至都比AWS好，因为它更简单。

最后用一张图来总结下容器服务可以带给我们什么。



SpeedyCloud研发总监李孟：不要让底层细节被上层打散

作者　刘羽飞

关于受访者

李孟现担任 SpeedyCloud 研发总监，目前主要负责资源调度系统的设计与研发工作，为 SpeedyCloud 一站式云服务平台提供技术支撑。李孟曾在蓝汛就职 7 年，专注于 CDN GSLB 及其衍生领域研发与实践，是蓝汛自主研发 CDN DNS 的第一人，在研发与运营分析过程中积累了丰富的行业经验。

随着互联网的发展，对于现代企业来说，DNS 与 CDN 服务的作用正变得愈发重要，网络访问速度决定了前端客户体验，同时也影响着内部业务系统的运行。SpeedyCloud 作为一家新晋 IaaS 云服务供应商，在 DNS 与 CDN 方面同样拥有丰富的实践经验。今年 QCon 上海 2015 上，SpeedyCloud研发总监李孟作为大会演讲嘉宾，分享了 CDN 服务设计开发实践的心得体会。为了进一步了解技术人在这些研发工作背后的故事，InfoQ 专门现场采访了李孟。

InfoQ：您在 CDN 领域的经验已经超过 7 年，当时是什么原因进入这个领域的呢？

李孟：进入这个领域完全是因为巧合。刚入职的前两个月，还没有建立比较明确的目标，而公司也没有相应的软件基础，只能靠自己去摸索。当时的困难在于，自己只有一些运维经验，并没有相应研发经验，而且所使用的开源软件也比较复杂。CDN 与 DNS 业务的研发既要保证稳定性，同时又要保证兼容性，还要完成对整个业务线的不间断支撑。

InfoQ：当时在自行研发的过程当中是如何进行考虑与选择的？

李孟：那时选择了 BIND。DNS 毕竟是个轻量级的服务，其开发量并不是很大。但在当时 BIND 已经发布 8 年时间了，因为在开发过程中不断地打补丁，其可读性已经变得非常糟糕，另外为了照顾 DNS 所有业务的性能，那么代码的可读性就会变得更差。

如何让整个 DNS 系统上下游无缝融合在一起？这个问题上我考虑了很长时间，这需要理清 DNS 在实际应用场景中的使用过程，需要结合 CDN 的应用场景，明确应该怎样运作，所含的路径有哪些等问题。之后还要将互联网协议和真实的运行环境结合在一起，这样在业务最终上线的时候，才能够实现良好的效果。

InfoQ：在这次 QCon 上海中您的演讲主题是选型方向的，出于什么考虑而安排这样的演讲主题呢？

李孟：首先是让大家了解 CDN 的技术概念，另外一个原因是想要带领大家思考 CDN 对于云计算平台来说充当了什么样的角色。因为如果 CDN 流量调度不当的话，就会导致设备上的流量抖动异常，这对于承载 CDN 业务的云平台也是非常不利的。

流量调度说起来是个很宽泛的概念，如果能用 DNS 将流量调度表达好，就意味着后续的动态调度和静态调度也将会相对容易一些。因此希望能够借助 QCon 的机会分享 CDN 领域里的一些重要经验。

InfoQ：技术工具有理论性能，但是实际用到生产环境中就是另外一回事了。对此您有什么看法？

李孟：之所以进行定制开发，性能上的差异是一个重要原因。性能上不达标，那么实际的表现就会跟测试时的差距比较大了。

另外，开源 DNS 方案的功能相对比较可靠，但也会有一些差异，比如只能做地域性切割，只能做流量均分，因此某些场合下就没办法直接用开源软件。否则一旦流量出现大幅度抖动，后台承载业务的云主机，就将受到冲击。因为这意味着需要采购更多的云服务资源，或者配置要更多的硬件设备，但这些资源的有效利用率并不高，反而还提升了 IT 成本。

结合 CDN 实际应用环境来看，这说明很多时候用户如果只是按照最直接的方式去实施，就会发现实际数据相比于预期目标来说差距很大。

InfoQ：您认为影响 DNS 调度效果的因素都有哪些？同时又该如何评价 DNS 调度效果呢？

李孟：智能 DNS 实际上是 CDN GSLB 的表达，表达的好坏效果会直接影响到后续流量调度的难易程度。除了受到 Local DNS 用户规模差异悬殊的影响以外，DNS 流量调度过程中产生的流量抖动还由另外四个原因所决定着，首先部分终端改变了自身的 Local DNS；其次，Local DNS Cache 影响调度生效和失效；第三，Local DNS 涵盖的用户群大小不一；第四，Local DNS 择优行为影响数据对等。

评价 DNS 流量调度效果的指标主要为两个——精度和准度。较高的精度会让调度系统动态调节更容易，设备带宽利用率也会更高，但是在实际 DNS 中的流量比例配值会出现差异，在复杂情况下很难区分开具体的流量配比。而较高的准度则会降低调度系统预测规划的难度，但调度过程中的流量抖动会更严重。对于某些 DNS 流量调度方式来说，有时精度和准度是不可兼得的，因此要可以使用不同的 DNS 流量调度方式，以产生不同的调度精度与准度。

InfoQ：是否还有其他更多类似的误区？应该怎样去避开这些误区？

李孟：很多误区本身比较隐蔽。比如CDN使用者将流量均分给两个CDN公司时，会采用CNAME分成两支的做法，然而这种做法并不是DNS协议里的标准内容，导致的结果就是底层细节被上层打散，每次请求分配不均衡也无规律，流量飘忽不定。

这表明某些看似很直观、很可靠的方式，到了底层就会影响非常大。企业用户去定制特别的流量调度时，正常来说是要按照固定属性确定调度方式的，但有时候用户希望达成定量调度时，就会让调整渗透下去，把定量行为加入到DNS里。做DNS定量时，其信息是不对等的，DNS进行调配时，可能会因为访问不对称，而导致调度也不对称。DNS在调度表达上有时确实有些不合常规，而带来效果有时也是不可预估的。

从DNS调度的判断依据来看，不推荐直接使用DNS的统计计数数据作为DNS调度依据，因为这些解析数据并不可靠，它们都是经过Local DNS不稳定的择优选择后产生的，使用这些数据作为调度依据会让调度效果变得不可预估。比较常见的流量调度依据包括固有属性以及无状态属性，固有属性比如有Local DNS IP的地域属性，或者网络属性等，而无状态属性则比如有Local DNS IP的Hash值特征，或者随机值等。

常见的流量调度表达方式包括地区切分、默认等分、随机比例、以及按照Local DNS IP切分流量，其中默认等分的方式在调度准度与精度上均有很好的表现，而其余三种则都有各自不同的侧重点。

在决定采用哪种调度方式前，建议技术人员先去结合数据进行分析，在拿到一线的、真实的、完整的数据之后再做相应的调研，总结出相应的规律，通过数据分析明确什么能做，什么不能做，什么想做，什么不想做，最后决定采用哪种调度方式。

InfoQ：能否谈一谈您个人以及SpeedCloud在CDN与DNS方面的未来规划？

李孟：DNS这方面的业务目前仍然在规划当中，是否以CDN为重点，也是有待考虑的。现在只要是做DNS功能，就几乎必带CDN，希望在今后能够形成一个具体产品，为行业发展带来帮助。

图像验证码和大规模图像识别技术

作者　华先胜

为区分人和计算机，互联网上的很多服务都使用了验证码技术，例如电子邮箱申请，银行系统登录，电子商务系统的交易确认，等等。虽然字符识别仍然是最常用的验证码方法，但是基于图像语义识别的验证码逐渐出现在一些重要的互联网应用上，并引起了热议。一方面大家对其中的一些难题大力吐槽，一方面又有人号称能够破解，能够自动识别这些图像。那么，目前的图片自动识别技术到底有没有可能破解这种验证码呢？有没有更好的图像验证码方法，既安全又不影响真人的使用体验？今天我们一起来探讨一下这个问题（本文纯属笔者个人思考，因见识、能力有限，错漏之处难免，还望阅者见谅）。

验证码的由来

验证码的学名叫做CAPTCHA，是Completely Automated Public Turing test to tell Computers and Humans Apart缩写，意为“全自动区分计算机和人类的公开图灵测试”，也就是一种用来区分人类和计算机的方法。通常是由计算机生成一个对人类而言很容易而对电脑而言非常困难的问题，能回答者被判定为人。

验证码测试其实不是标准的图灵测试，因为标准图灵测试是人类来考计算机的。通常的人工智能研究者的目标是让他们设计的系统通过图灵测试，让人类无法区分对方是人还是机器，从而说明人工智能能够接近人类的智慧。常用的图灵测试是对话，当然不是面对面对话，而是“键盘对话”，不然当然知道对方是人是机器了。通过这个测试当然并非易事。直到2014年6月份，英国雷丁大学宣布来自俄国的Eugene Goostman聊天机器人在伦敦皇家学会举办的2014年图灵测试大赛中，首次通过图灵测试，骗过人类，让人类认为“他”是一个13岁的男孩。其判断标准是：5分钟内的一连

串键盘对话，让 30% 的人认为它是真人。当然，图灵测试还有其他的任务和方式，例如最近在《科学》上一篇论文讨论如何让计算机象人一样从一个手写的字，“只看一眼”，就能学会象人一样写这个字，而让其他人无法区分是人写的还是计算机写的 [1]。

与之相对，验证码是计算机出题来考人类（当然，这个题目怎么出，也是由人来设计，然后计算机自动产生的）。验证码的目的是不让计算机通过验证，从而阻止人们用计算机做其不应该做的事情。通常这些事情如果用计算机程序来做，可以做到数量很大，或者速度很快，或者兼而有之，从而破坏正常用户的使用秩序。例如注册大量免费电子邮箱来发广告邮件，在微博等社会化媒体中注册大量虚假用户作为某人的“僵尸粉”，以及自动抢票软件，等等。

早期的验证码大多用扭曲的文字来实现，用以避开 OCR（自动光学字符识别技术）的自动识别，例如 Yahoo 早期的验证码版本 EZ-Gimpy。后来，又有在扭曲字符上加曲线，以及将字符拥挤在一起，有时让真人也难以识别。

图 1：不同的字符验码（Credit: Wikipedia）

验证码的基本要求

一个好的验证码方案的特点很简单：人容易识别，而机器无法识别。严格来说，如同密码方案一样，验证码产生的算法应该公开，从而能够公开验证算法的安全性，但在实际使用中一般鲜有这样做的。在电子邮箱申请等这种典型的需要验证码的场景，还要求验证码的答案可能性（解空间）足够大，从而让猜测的成功率也极低。例如，如果有哪怕 1% 甚至更小的几率能猜对，也不能有效防止机器自动注册大量邮箱来发送垃圾广告邮件。例如，一个四个数字的验证码，10000 种可能的结果，在没有任何智能的情况下，计算机能猜中的概率是 0.01%，如果再加上一些智能，把猜中的几率提升到 1%，则不能有效防止机器自动注册电子邮箱。如果是 16 个英文字母，则解空间的大小是 52 的 16 次方，这时能猜中的概率就极低极低了。

当然，有些场景的验证码并不需要太大的解空间，特别是对实时性要求比较高和服务比较复杂的系统（例如在线交易），因为破解程序比较难以在短时间内产生高 QPS 的攻击。不过，解空间太小，可能会被事先离线穷举，而在线则通过匹配就可以找到答案。对此，我们后续

再进一步讨论如何设计更好的验证码，我们先看看为何用图像识别做验证码。

关于图像识别验证码

图像识别验证码在十几年前就有人尝试了。图像验证码的一个好处是不需要人键入识别出来的文字，只需要点击对应的图片就可以了。其关键还是要解空间足够大，并且机器算法基本无法完成。例如，在 2003 年，微软研究院提出一种利用人类对人脸区域的敏感性，将人脸进行变化，包括头部的旋转、变化的光线、复杂的背景等等，使得计算机人脸检测算法会失败，而人类可以轻松的指出人脸的位置 [2]。

图 2：人脸验证码（Credit: Yong Rui, etc.）

2009 年，有一个叫做 VidoopCAPTCHA 的图片验证码，与现在大家看到的图像验证码类似，其区别是每个图片中有个字母，要求用户按提示输入与提示词对应的图片的字母。不过，当年就被哥伦比亚大学的一个学生破解了，现在已经没有人使用了，这个叫做 Vidoop 的公司也不存在了。2010 年前后一个叫 Confident Technologies 的公司研发出了和现在流行的图像验证码更为相似的验证码。不同的是，它要求用户进行三次识别，每次只有一个答案是对的。目前互联网上有 139 家网站使用的是这个技术。

图 3：Confident Technologies 的图片识别验证码

通常图像验证码系统给的提示文字不想图 3 中“flower”那么简单好认，而是变形了的文字，也就是说，也是一个嵌入的文字识别验证码。换句话说，图像识别验证码中的提示文字是个文字识别“验证码”，尽管只需要人认出来而不需要人键入。文字识别验证码技术和可能的攻击超出了本文范围，因篇幅有限就不做讨论了。

那么计算机自动图像识别技术到底能否破解这种验证码呢？如果可以，是如何实现的呢？在讨论这个问题之前，我们看一下图像识别技术到底发展到了什么境地。

什么是自动图像识别技术

图像识别在学术界已经被研究了几十年，从大约 50 年前就有人提出让计算机外接相机

来识别相机看到的东西（图灵奖得主 Marvin Minsky）。这个见地在今天似乎没有什么，但是在 50 年前提出则相当不易 – 试想，如果让我们预测 50 年后的技术，甚至 5 年后的技术都不太容易的。

图像识别方法可以分为两大类，模型的方法和搜索的方法。模型的方法是在业界研究和使用最多的方法。模型的方法是试图通过一些已知“标签”（也就是这个图像是什么，例如图 3 中的第一幅图的标签是“小汽车”）的图像，通过机器学习的各种方法来学习一个描述这些标签的“模型”，从而，对于一个新的未知图像，经过这个模型判断出其应该具有的标签。

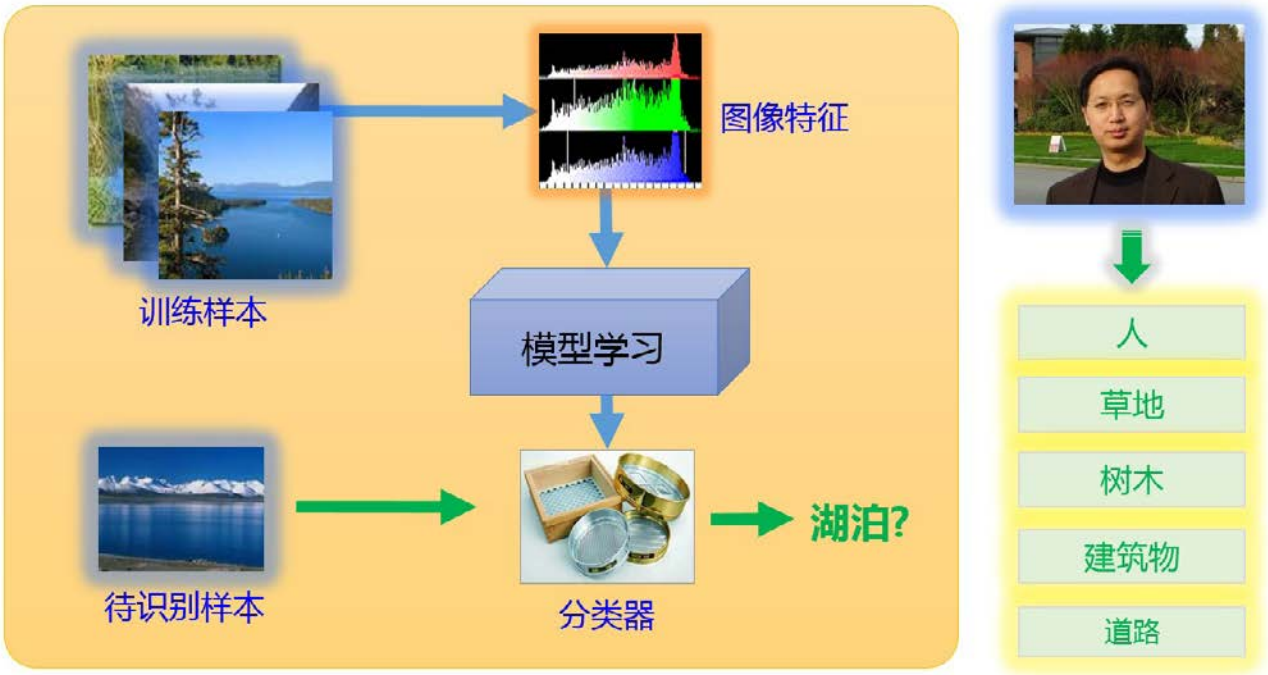


图 4：基于模型的图像识别示意图

基于搜索的方法是在大数据时代才出现的方法，其基础是将已知标签的图像数据建成一个可以进行高效率检索的数据库，称为图像索引。通常需要大量的图像来建索引，但图像的标签可以有少量的噪声。那么，对一副待测图像，我们到这个数据库中去找与其相同或者相似的若干图像，然后综合这些图像的标签来预测待测图像的标签。

当然，这两类方法究其本质并无差别，只是搜索的方法利用了大规模图像索引的技术，不去建立模型，而是直接用这些数据来进行匹配，所以我们可以认为这个大的索引就是一个特殊的模型。在大数据的时代，识别和搜索已经密不可分，精准的搜索离不开识别，广泛的识别也离不开搜索。笔者在很多报告中也经常把识别和搜索放在一起来讲。

自动图像识别有多牛

比较靠谱的图像识别方法是从上个世纪末 SIFT 图像特征的提出开始。在之后的十几年里，研究者们大多是从特征和模型来进攻这个难题。例如，特征上，SIFT 和 HOG 等，模型上 SVM，Boosting，DPM 等。2012 年前后，深度卷积神经网络在图像识别领域开始应用，则是同时去解决模型和特征的问题。也就是，既可以通过深度学习直接从图像像素开始训练图像识别模型，也可以通过同样的训练得到图像的更有效的特征描述，然后采用传统的机器学习模型来

训练识别模型。基本上，深度学习的方法击败了所有传统的方法，使得图像识别的准确率向前迈了很大一步。

ImageNet 图像识别（分类）是最有影响力的一个公开 PK 图像识别算法的比赛，其最基本的任务是训练一个 1000 类的图像识别器。在深度学习方式使用之前，最好的识别“前 5 准确率”（也就是给待测图像预测出的前 5 个标签中有一个是对的）只有 74% 左右，而深度学习第一次就将这个结果提升了将近 10 个百分点，到达 83.6%。而最新的 2015 年的最好结果达到了 96.3%。

但是对于一个希望能实际应用中使用的图像识别系统，还有三个方面：数据、系统和用户反馈。真实世界不只是 1000 类图像，如果每一类都需要大量的标注图像，这个工作量还是相当大的。例如，世界上大概有 900 多种不算很难见的花卉，有 500 种左右纯种的狗，大约 50000 个城市，淘宝上有超过 10000 种实物商品。因此，也有研究者开始着手如何高效甚至自动或半自动创建数据集。而系统方面的要求，是保证模型的训练在可控的时间内完成，并且图像识别则可以在非常快的时间（例如数十毫秒）内完成。用户反馈的使用，则是一个识别系统在使用的过程中，不断吸收用户的使用行为数据来逐步改进识别系统的准确率和覆盖率。

总结起来，衡量一个图像识别系统的性能可以从如下四个方面来看：

（1）准确率：衡量是否能正确识别图像中的内容，这也是多数人关注的目标，包括 ImageNet 比赛；

（2）覆盖率：衡量能识别多少种语义，这个在学术界比较少有人关注，却对一个识别系统能否在真实场景中应用起着很关键的作用。工业界，例如各大搜索引擎公司（微软、谷歌和百度）提供的识别 API 则能识别更多的内容（大概在万这个量级上）；

（3）效率：主要是衡量多快能够识别出结果，也包括多快能够训练或更新一个识别器；

（4）用户体验：在产品和用户界面上的考虑，如何弥补识别的准确率、覆盖率或效率的不足。

举个例子：笔者一年前曾经研发了一个称为 Prajna 的系统 [3]，用自动训练数据获取和清洗的方法，自动快速训练过很多的图像识别器，包括狗的品种，花卉，植物，地标，食物等等。其自动数据获取的方法比较有利于解决覆盖率的问题，而不需人来标注数据，使得产生一个新的识别器的周期变得很短，只要数小时到两三天的时间。有一次，笔者去参加朋友家里的一个聚会，主人家里有一株很漂亮的花，参与聚会的朋友们这都想知道花的名字，可惜主人也忘了。于是，我启动我研发的花卉识别器，成功地识别出了这株花（见下图）。当时很多图像识别程序只能识别出其为一种花，而不能知道其具体名称。这个也说明了实际应用系统中，“覆盖率”是相当重要的。

图 5：笔者的图像识别系统 Prajna 识别出这是“Amaryllis（孤挺花）”

那么，有了这些图像识别领域的最新进展，是不是就能破解现今一些网站的图像识别验证码呢？在讨论这个问题之前，我们看看图像识别除了可能可以破解验证码，以及在某些场合能够识别人不能识别的物种，还能干什么更有意义的事情呢？

图像识别还能做什么

笔者在图像识别和搜索领域摸爬滚打近二十年，以此经验，笔者认为目前最有实际用户需求的图像识别应用之一应该是商品的识别和搜索（如前所述，如今识别和搜索常常是密不可分）。其中典型的应用场景就是通过拍照的方式寻找需要的商品，例如亚马逊的 FireFly 和阿里巴巴的拍立淘。笔者是拍立淘算法的研发主管，在此简要介绍一下图像识别在拍照商品搜索中的应用。限于篇幅，只能非常简略地介绍一下，希望有机会另文介绍图像搜索技术的前世今生，以及拍立淘中的前沿技术和有趣应用。

在拍立淘中，对于用户拍的商品照片，我们首先要识别其大概的类别，例如，是上装，还是裤子，还是箱包，还是零食，等等。然后我们需要知道用户关注的商品在图像中的位置，通常称为“主体检测”。再然后，就是产生一个该主体的描述，也就是特征，通常是一个高位向量。这个向量的产生方法则是通过训练一个多类商品的识别器来实现的。最后，我们用这个特征到一个建好索引的大商品库中去找与此相同或相似的产品，返回给用户。这些关键步骤，包括商品识别，主题检测，和特征提取，都是基于深度学习的方法来完成的。

为什么说这个应用场景是个“刚需”场景呢？一方面，确实有很多情况下，用户用文字无法描述心中所需，只能通过样例的方式来查找；另一方面，这个仍在不断改进中的产品每天已经有数百万的活跃用户，每天在之上产生的交易量近千万。

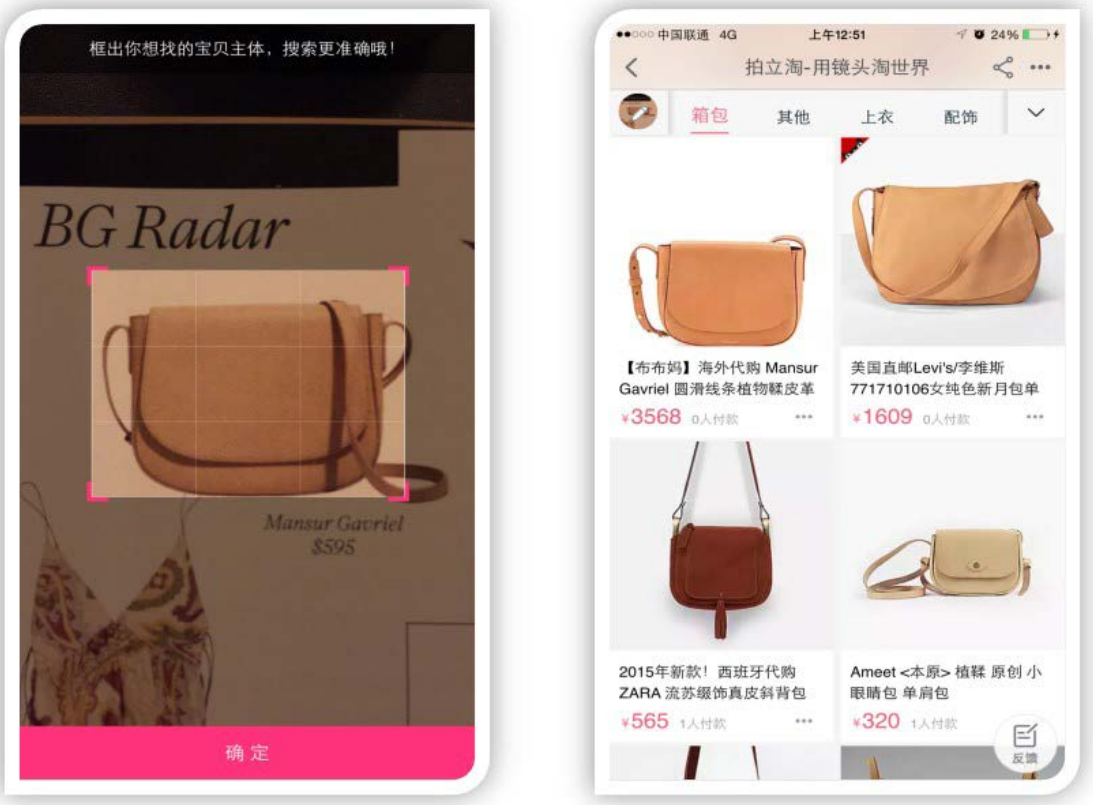


图 6：拍立淘商品搜索一例。左：用户拍的商品和自动框出的商品区域；右：识别结果（箱包）和搜索结果。

图像验证码真能被破解吗？

什么叫破解呢？是不是完全能识别所有的图片才能叫破解呢？其实不需要，如果能使识别准确率达到一定的数值就可以了，例如对于邮箱注册，一个很低的准确率就足以构成威胁，例如 10%，甚至更低。对于在线购票系统来说，大概 10% 左右也能构成很大威胁了（有网站统计表明，对于目前流行的图像识别验证码，真人的一次识别成功率只有 8%）。

目前流行的图像识别验证码的解空间有多大呢？通常见到的是识别 8 个图片，通常是 1 个到 3 个答案是对的（不知道有没有 4 个的），那么可能的答案组合一共只有种可能，如果任意回答，由 1/92（1.1%）的概率能答对（如果增加到 1 到 4 个答案可能对，则此概率降为 0.06%）。也就是说，如果破解程序没有任何智能，92 次中才能有一次机会通过认证。当然这种概率如果用来防止邮箱注册大概不太有效，但对于在线交易等一些秒杀场景，还是有效的。

如果有自动识别程序帮助呢？如果类目数不是太多（例如只有几百个），我们找到这些可能的物体类别后，训练一个对于图像的自动识别器并不是不可能（在此就不详述方法了）。当然，前提是能自动识别提示文字，不过这个问题通常比图像识别相对容易些，虽然提示文字也有些变形（如前述，字符验证码的技术不在本文讨论之列）。假定我们的“前 1 识别率”（就是第一个识别出来的物体名称就是对的）是 30%，那么 3 到 4 次就有一次能识别对，这个成功率足以对验证码系统造成威胁。用谷歌、百度或微软的识别 API，也是一种方法，但这些 API 并没有优化到这些类别上，应该不会有专门为这些类别训练的识别器准确率高。

有人提到过用图像比较的方法破解，也就是通过多次试用，将验证码后台的所有或大多数物体图像爬下来，人工标注，然后在线验证时直接比对即可。这种方法也是不错的思路，但也是需要由图像检索的专业人士才能真正实现，因为这里涉及到两件事情，一是合适图像的特征描述（如果直接用图像像素，图像加噪声后，比对方法就失效了），二是索引技术，以保证快速的查找（如果后台图像不太多，则此步可免）。

那么，图像验证码就没有出路了吗？当然，上面提到的方法只是对理想情况下的破解思路，一个验证码系统可以增加很多变数和限制，使得破解更为困难。不过，增加题目难度的方法是违背验证码初衷的，试想如果普通人也难以识别的东西如何用来区分计算机和人呢？

笔者凭经验给图像识别验证码支些招，基本的思路仍然是：（1）题目对人容易，对机器难；（2）解空间足够大；（3）当前的图像识别方法不能很好解决。可能的方法有：

（1）将待识别小图加复杂背景，并融合成一张无缝的图；

（2）多用识别算法容易混淆的相似物品；

（3）不断改变背景的模式和拼图的方法；

（4）物品前景加一些干扰；

（5）增加一些细分物体类别；

（6）除物品识别外，引入属性识别要素；

相信如果验证码采用上述几个大招，破解的难度会增加很多倍，即使是图像自动识别专家也要费一番周折才能解决，或者一段时间内根本无法解决，也就没有去攻关解决的价值。等技术进步了，就再更新换代，让破解者跟不上出题人的步伐。举个例子，只使用其中部分方法，大家可以看看，如果验证码长成下图这样，人应该还能识别（我找我 8 岁的儿子测试，大概 2 秒钟都能正确识别），反正机器大概是不容易搞定了（当然不是只是这一个情形，而是类似的大量情形）。

图 7：验证码问题：点击坐着的人的红色上衣。什么，不够难？那就点击图中看起来最小的那个水杯。

其实还有一个简单的大招，就是充分利用这个验证码系统，快速地增加和迭代系统的数据，使得破解者望尘莫及（笔者就卖个关子不详述方法了，内行者读到这里自能知晓）。

图像识别的未来

我们再回到图像识别的正向使用上来。虽然图像识别有了很大的进展，但是真实世界的图像识别仍然有很多困难。如前所述，在相对不大的集合上（例如 ImageNet 一千类识别），经过几年的努力，其识别准确率提升很快。但是，真实世界是很复杂的，在更多的场合下需要识别的覆盖率、准确率都要高，而且速度要快。笔者以为，只靠模型和特征的方法是不能完全解决的，未来的解决方法应该是模型、特征、数据、系统以及用户反馈这五个要素的综合作用，才能逐步达到理想的识别效果。

[1] BM Lake, et al. Human-level concept learning through probabilistic program induction. Science. Vol 350, no 6266, pp 1332-1338.Dec 11 2015.

[2] Yong Rui, Zicheng Liu. Excuse me, but are you human? ACM Mulitmedia 2003

[3] Xian-Sheng Hua, Jin Li. Prajna: Towards Recognizing Whatever You Want from Images without Image Labeling. AAAI 2015.

作者简介：华先胜，IEEE Fellow，ACM 杰出科学家，TR35 获得者，阿里巴巴研究员 / 资深总监。2015 年加入阿里巴巴搜索事业部，负责大数据多媒体内容分析及图像搜索算法团队。在此之前先后在微软中国研究院、微软总部 Bing 搜索引擎以及微软美国研究院任主管研究员、首席研发主管以及资深研究员。2001 年北京大学博士毕业，一直从事图像、视频内容分析和搜索方面的研究和开发工作。

EGO NETWORKS

EXTRA GEEKS' ORGANIZATION

我们的使命：

引导高端技术人学习和成长

我们的愿景：

全球最具影响力的高端技术人社交网络

关于EGO：

EGO是极客邦科技旗下高端技术人聚集和交流的组织，旨在打造全球最具影响力的高端技术人学习和成长平台，线上线下相结合，为会员提供专享服务。

EGO采用实名付费会员制，每一位申请加入的会员都必须经过EGO组织的严格审核，保证会员信息的真实性，让每一位会员在平等和相互信任的环境中分享交流，大家共同学习，共同成长。

微信公众号

即刻加入EGO

为什么加入EGO?

学习交流

独一无二的学习和交流体验，会员间平等分享各自的实战经验，从工作到生活

拓展人脉

认识国内外志同道合的技术同行，分享交流，拓展自己的人脉圈子

获取信息

获取最新最重要的行业信息及资讯，了解最新技术动态、新闻、社会热点，独家原创的专家观点及评论

解决问题

身边的智囊团，遇到问题时能够第一时间通过EGO平台找到答案

工作之外

在日益激烈的竞争中平衡工作与生活之间的关系，提高软技能

入会资格

1. 认同EGO理念，遵守EGO规则；
2. 热爱学习，乐于分享，追求成长，相信技术的力量
3. 公司/团队中的技术决策者，如CTO、首席架构师、技术总监等；
4. 在某一技术领域具有超过5年的工作经验。

EGO客服微信号：egowinner　EGO客服邮箱：service@egonetworks.org

了解更多信息请访问EGO网站：http://www.egonetworks.org

QINIU

产品 = 七牛云 . 服务(想法)

七大行业解决方案

娱乐　广电　教育　智能硬件　O2O　旅游　安防

七牛四大产品

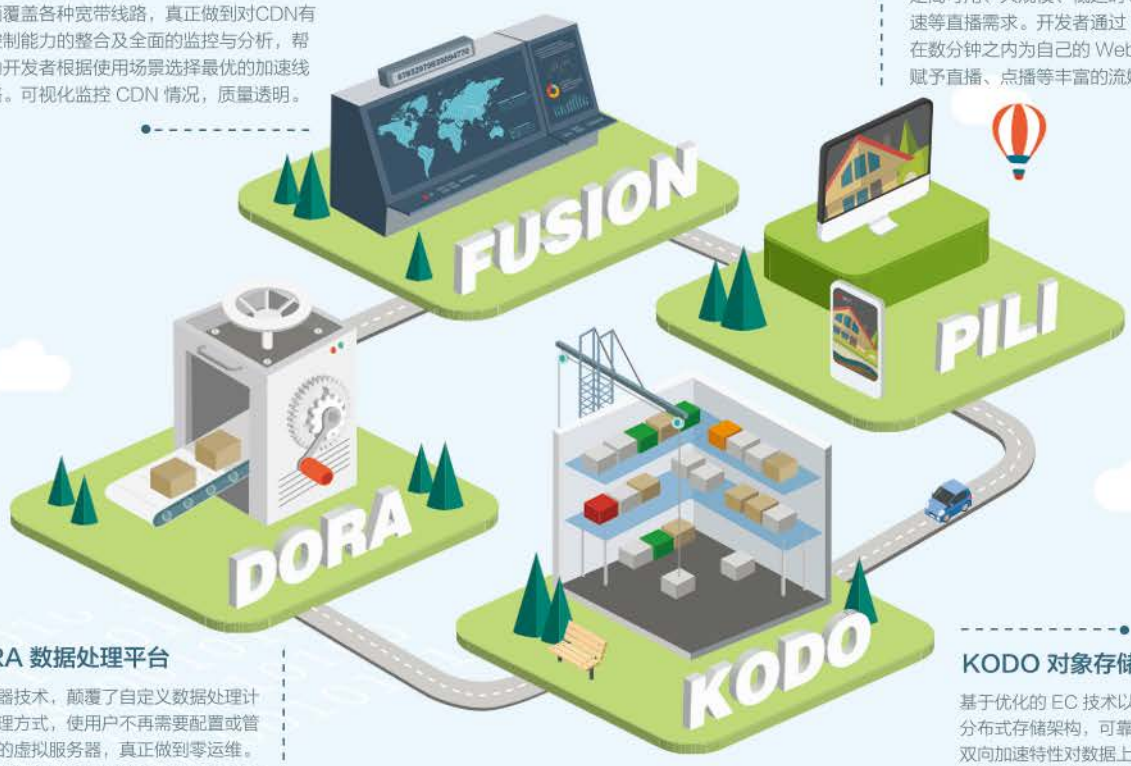
七牛重新定义了云存储

FUSION 融合 CDN 管理平台

持续引入国内外主流 CDN，双向加速，全面覆盖各种宽带线路，真正做到对CDN有控制能力的整合及全面的监控与分析，帮助开发者根据使用场景选择最优的加速线路。可视化监控 CDN 情况，质量透明。

PILI 直播云服务

全球首个用 Go 语言实现的企业级直播流媒体云服务，充分利用七牛的海量网络节点资源满足高可用、大规模、低延时、跨终端和全球加速等直播需求。开发者通过 RESTful API，能在数分钟之内为自己的 Web 和 Mobile App 赋予直播、点播等丰富的流媒体功能。



DORA 数据处理平台

基于容器技术，颠覆了自定义数据处理计算的管理方式，使用户不再需要配置或管理单一的虚拟服务器，真正做到零运维。为用户提供了按需弹性伸缩的计算力和高自由度的开发语言环境，能无缝衔接用户原有的业务技术栈。

KODO 对象存储服务

基于优化的 EC 技术以及七牛自主研发的全分布式存储架构，可靠性高达16个9。首创双向加速特性对数据上传下载均加速，镜像存储、客户端直传、断点续上传等功能，最大程度减少了服务器资源浪费。

锤子手机发布会提到的 OpenResty 是什么？

作者　OpenResty社区

编者按

在 12 月 29 日晚上的锤子科技冬季新品发布会上，罗永浩宣布把此次发布会的门票收入全部捐赠给正在筹备当中的 OpenResty 软件基金会。随后，有关 OpenResty 的讨论，在整个技术圈子开始发酵。我随即联系了 OpenResty 的温铭，因为之前有和他聊过基金会的运作方式以及 OpenResty 的发展，所以没有寒暄，开门见山，问他有没有时间写篇关于 OpenResty 的文章。

温铭很给力地秒回，说没问题。于是我草拟了一个提纲，给他发了过去。他很快向我发送了一个石墨的链接，我打开一看，很是震撼，他已经根据提纲，把内容分发给了几个社区的同学，大家很快就开始撰写。大概在晚上 11 点半，我看到稿子基本快搞定，他们已经有专门的同学在 Review。看了他们整个的协作流程，我非常兴奋，春哥（OpenResty 创始人）和温铭已经把开源的协作理念带到了文章的写作中，这也让我对这个社区顿生敬意。

早上起来，看到温铭的微信，他说初稿已经完成，我打开一看，洋洋洒洒几千字，几个人共同撰写的文章，但看起来却像是一气呵成。祝贺 OpenResty 在 2015 年的最后一周能有这么重要一个里程碑，正如温铭所说，2015 年是 OpenResty 发展的重要一年，确实。也非常感谢锤子科技给予国内开源项目的大力支持，期待越来越多的企业能参与进来，帮助国产的开源项目更快地发展。

写在前面的话

在刚刚结束的 Smartisan T2 发布会上，锤子科技宣布把发布会的门票收入捐助给 OpenResty 软件基金会，用来推动该开源项目的发展，听到这个消息非常的开心。首先要感谢锤子科技给予国内开源项目的大力支持，这次捐助，不仅是 OpenResty 的一件大事，也是国内开源社区的发展的一个重要里程碑。

关于 OpenResty，最初我们的想法很简单，希望通过《OpenResty 最佳实践》开源书籍这样的方式将我们在使用 OpenResty 过程中的一些经验分享给大家，并在 stuQ 上面录制本书的视频教程，后来为了降低 OpenResty 的使用门槛发起了汉化 OpenResty 官方文档的项目，并以社区的名义举办了全球第一届 OpenResty 技术大会，这一届大会将国内 OpenResty 的使用者和开发者聚集到一起，形成了活跃的社区氛围。

2015 年是 OpenResty 最重要的一年，关于 OpenResty 我们还有很多的想法和计划，谢谢锤子科技的支持和鼓励，我们相信美好的事情会再次发生。

OpenResty 是个什么样的项目？

OpenResty 是中国人章亦春发起的一个开源项目，它的核心是基于 NGINX 的一个 C 模块，该模块将 Lua 语言嵌入到 NGINX 服务器中，并对外提供一套完整 Lua Web 应用开发 API，透明地支持非阻塞 I/O，提供了“轻量级线程”、定时器等等高级抽象，同时围绕这个模块构建了一套完备的测试框架、调试技术以及由 Lua 实现的周边功能库；这个项目的**意义在于极大的降低了高性能服务端的开发难度和开发周期**，在快节奏的互联网时代这一点极为重要。

OpenResty 的起步和发展经历了一个漫长的时期，从 09 年基于 NGINX 用 C 语言重新改写的版本，到 11 年开始有很多公司开始小范围内尝试使用，再到今天，OpenResty 知名的 CDN 行业，各大电商，手游领域都有应用，我们对 OpenResty 源码发布包和 Windows 二进制发布包的下载做了一个统计，这些流量分别来自 135 个国家的 3072 个城市，特别是今年下半年以来，每个月都有 5 万次以上的对发布包的下载请求。其中有近一半的用户来自中国。

OpenResty 的发展

OpenResty 最早是雅虎中国的一个公司项目，起步于 2007 年 10 月。当时兴起了 OpenAPI 的热潮，于是公司领导也想做一个类似的东西，可以支持各种 Web Service 的需求。在部门领导的支持下，最早的 OpenResty 实现从一开始就开源了。最初的定位是服务于公司外的开发者，像其他的 OpenAPI 那样，但后来越来越多地是为雅虎中国的搜索产品提供内部服务。这是第一代的 OpenResty，当时的想法是，提供一套抽象的 Web Service，能够让用户利用这些 Web Service 构造出新的符合他们具体业务需求的 Web Service 出来，所以有些“meta web servie”的意味，包括数据模型、查询、安全策略都可以通过这种 Meta Web Service 来表达和配置。同时这种 Web Service 也有意保持 REST 风格。

与这种概念相对应的是纯 AJAX 的 web 应用，即 Web 应用几乎都使用客户端 JavaScript 来编写，然后完全由 Web Service 让 Web 应用“活”起来。用户把 HTML、JavaScript、CSS、图片等静态资源下载到网络浏览器中，然后客户端 JavaScript 代码就开始运行，跨域请求雅虎提供的经过站长定制过的 Web Service，最后应用就可以运行起来。

不过随着后来的发展，公司外的用户毕竟还是少数，于是应用的重点是为公司内部的其他团队提供 Web Service，比如雅虎中国的全能搜索产品，及其外围的一些产品。从那以后，开发的重点便放在了性能优化上面。章亦春在加入淘宝数据平台与产品部的量子恒道统计团队之后，决定对 OpenResty 进行重新设计和彻底重写，并把应用重点放在提供高性能的 Web Service 平台，以很好地支持像量子恒道统计这样复杂和繁忙的 Web 应用上面，所以量子统计 3.0 开始也几乎完全是 Web Service 驱动的纯 AJAX 应用。

第一代 OpenResty 是基于 Perl、Haskell 和一点点 C 实现的。在王晓哲的提议下，第二代 OpenResty 选择使用 C 语言基于 NGINX 进行开发，同时选择 Lua 作为最主要的用户语言。

OpenResty 的架构以及应用场景

OpenResty 与其他常规开发语言或框架截然不通，可以说是另辟蹊径的把两个极其优秀的组件 NGINX 和 Lua 进行糅合，充分利用各自的优势相互弥补。不仅保留了 NGINX 的高性能 web 服务特征，Lua 更是在近乎不损失性能的前提下可以快速、简单的进行业务功能开发，同时享有代码动态装载和卸载的特性。在运行速度、可伸缩性、灵活度以及开发效率之间寻找一个平衡点。

目前 OpenResty 的主要应用场景有：HTTP Proxy、API Server、Web Application。

- HTTP Proxy：这个在 CDN 行业用的比较多的，请求改写与路由调度、缓存控制、Web 应用防火墙（WAF） 等。
- API Server：各种智能设备 APP、广告拉取等请求比较密集，并发、QPS 比较高的环境。
- Web Application： OpenResty 创建初衷就是为了做这个，新浪移动已经在所有产品线使用 OpenResty，核心业务也在不断从 PHP 迁移到 Lua，京东已经开始使用这套方案解决高并发环境下的页面应用。

当然，我们也看到一些大的生产用户的另类用法，比如基于 OpenResty 来实现比较完整的分布式存储的后端。

OpenResty 社区的主旋律

每个做技术的人，或多或少都会用到一些开源技术，顺其自然也就会参与到各种技术社区的活动中，或是问题讨论、或是技术交流又或是“PHP 是世界上最好的语言”这样的激辩。每个社区都带有自己的基因有自己的特点。这些多少与开源软件的作者有一定的关系。因为 OpenResty 的作者章亦春（人称春哥）是一个热爱编码对自己要求特别高，对代码质量追求近乎于完美、为人谦和、热心，甚至是儒雅的一个人，目前 90% 以上的 OpenResty 相关问题基本都会指向 OpenResty Google 邮件组，在那里你会看到不管是大牛还是小白的问题都会得到一个 agentzh 的账号认真细致的解答。**随着这些年 OpenResty 社区的积累，慢慢涌入的同学们也秉承春哥这种作风，大家互助共进。**

OpenResty 社区一直秉持兼容并包的思想，基于 NGINX 在整个 Web Stack 中所处的特殊位置，用户可以很方便的将 OpenResty 与现有的技术进行融合，比如 PHP、Java、Python、Ruby、Go、Nodejs 等。OpenResty 在网关这个层面，所以可以同时和其他后端应用并存，方便工程师将其他技术方案实现的系统有选择地迁移到 OpenResty。 当然，出于对极致性能的追求，OpenResty 社区还是更倾向于比较纯净的方案。

另外很重要的一点，就是开源工作者很看重的:

有趣。我们假设开源是一个编译器，它有优化选项，-Ofun，我们是针对乐趣进行优化。这一点看上去是和实用主义原则冲突，其实不然。因为对于一个工程师来说，最有意思的莫过于自己的技术，自己搭建的系统，自己设计的方案，能够在线上跑的非常好，能够服务越来越多的用户，这是非常大的一个乐趣。对开源工作者来讲，他也希望自己的代码能够跑在尽可能多的公司的服务器上，能够收到尽可能多的用户的感谢信。在一天干活最痛苦的时候，突然收到一封来自世界另一个角落的用户的感谢信，字里行间洋溢着一种感激，一种欣喜，那你这一天立马就会变得非常美好。所以，要让乐趣变成我们工作的主旋律，而不要让工作变成一种负担。

国外成功的开源项目比较多，或许跟许多发达国家的程序员们的精神状态有关系。比如我们发现一些国外的黑客非常心思单纯、热情似火。他们在精神上的束缚非常少，做起事来多是不拘一格。有的人即便长期没有工作单纯靠抵押和捐赠过活，也会不遗余力地投身于开源项目。而国内许多程序员的精神负担一般比较重，经济上的压力也比较大，自然难有“玩开源”的心思。

不过，国内也是有一些程序员拥有国外优秀黑客的素质的，而且他们通过网络和全球的黑客紧密联系在一起，所以我们完全可以期待他们未来有振奋人心的产出。在互联网时代的今天，或许按国界的划分来讨论这样的问题会变得越来越不合时宜。

未来的规划

未来我们会尝试在 OpenResty 的基础之上提供更接近各种典型的互联网业务的抽象，包括领域内小语言和相关的编译器、运行时的支持。在这种模型下，OpenResty 就是一个虚拟机，而 Lua 语言是这个虚拟机上的“机器语言”。我们希望开发者能在更高的抽象层面上思考和表达业务问题，而不必纠缠于实现细节，同时能通过各种上层小语言的优化编译器，享受到接近手写 Lua 代码的运行时效率，无论是空间还是时间。我们在过去几年中已经在这个方向上做过一些有趣的成功尝试，包括在雅虎中国基于第一代 OpenResty 做的 Meta Web Service 平台，以及淘宝量子恒道统计所使用的数据 API 平台，在优美、简洁和高性能之间取得了多赢。我们希望能通过自己的实践，让业界越来越多地关注“编译型” Web 框架所使用的优美抽象和优化编译技术，而不仅仅是传统的“解释型” Web 框架所使用的不断地叠加运行时封装，无论是类还是函数封装的方式。

在 NGINX 官方服务器前不久引入全新的 stream 子系统之后，我们也希望 OpenResty 能支持使用 Lua 和同一套 Lua API 来实现任意 TCP 协议的服务器，而不仅仅是 HTTP 服务器。类似地，我们也将支持用同样的方式来构造高性能的 UDP 服务器。这些变化将会极大地拓展 OpenResty 的应用范围。届时我们应该会看到更多有意思的应用。

我们很期待看到有越来越多优秀的青年乃至少年加入到我们社区，加入到我们的开发团队。再没有什么比新鲜血液更能激发一个开源项目的活力了。或许未来我们能以 OpenResty 软件基金会为依托，开展类似 Google Summer of Code 这样的活动，同时赞助和支持有兴趣的学生和“开源导师”在学校放假期间为 OpenResty 社区贡献代码、文档和教程。

当然，我们也希望能够以 OpenResty 软件基金会的名义，积极地奖励和赞助那些有想法的资深工程师，帮助实现 OpenResty 核心及周边那些富有挑战的项目，或者指导新加入的开发者（包括在校学生）完成较为简单的项目，充分发挥自己的专业技能和理论知识。

具体的一些工作如下，包括软件本身以及社区方面：

1、自身功能和周边生态的搭建

- 网站服务器和域名、官网的改版
- LuaJIT 核心和官方测试集的维护，性能优化，和新功能开发
- OpenResty 核心功能点的开发
- lua-resty-* 周边库的开源和开发激励
- iresty.org 包管理网站设计、服务器资源
- 更完善的文档，以及 demo
- 文档翻译和维护

2、扩大影响力和促进交流

- 国内外线下 meetup，多城市
- 成功使用公司列表和采访
- 大学的技术启蒙、在线教育视频
- 编写和出版官方学习书籍

3、OpenResty 软件基金会

- 做开源社区，我们是希望走国外成熟的软件基金会的模式，成立一个合法的非盈利性组织，用各个公司的赞助来维持乃至扩展开发工作，这样可以吸引更多的开发者加入，同时也给予用户信心。另外，我们成立组织和制定社区计划来长期维护 OpenResty 项目，对于赞助公司也是保障，并且所有费用的使用都是透明公开，并且被依法审计的。

InfoQ 中文站　迷你书

架构师 月刊
2015年12月

本期主要内容：微信斑马系统：微信朋友圈广告背后的利器，Spark和Hadoop孰优孰劣，通过大量测试来构建测试系统，实施微服务需要哪些基础框架，重建还是重构，Medium开发团队谈架构设计，被误解的MVC和被神化的MVVM，深入了解IAM和访问控制

开源启示录
第二季

开源软件的未来在于建立一个良性循环，以参与促进繁荣，以繁荣促进参与。在这里，我们为大家呈现本期迷你书，在揭示些许开源软件规律的之外，更希望看到有更多人和企业参与到开源软件中来。

顶尖技术团队访谈录
第四季

本次的《中国顶尖技术团队访谈录》·第四季挑选的六个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。

云生态专刊

《云生态专刊》是InfoQ为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”。