

# 架构师

ARCHITECT



## 热点 | Hot

Java 10正式发布,带来了这些新特性

伯克利推出世界最快的KVS数据库

Anna : 秒杀Redis和Cassandra

Spring Boot 2.0新特性解读

## 理论派 | Theory

Kubernetes效应

## 专题 | Topic

360开源项目大盘点



# CONTENTS / 目录

## 热点 | Hot

伯克利推出世界最快的 KVS 数据库 Anna：秒杀 Redis 和 Cassandra

Spring Boot 2.0 正式发布，新特性解读

这次没跳票！Java 10 正式发布，带来了这些新特性

## 理论派 | Theory

Apache Kylin 在绿城客户画像系统中的实践

## 专题 | Topic

360 开源项目大盘点

## 特别专栏 | Column

VPC 的 4 种典型应用场景



## 架构师 2018 年 4 月刊

本期主编 覃云

提供反馈 [feedback@cn.infoq.com](mailto:feedback@cn.infoq.com)

流程编辑 丁晓昀

商务合作 [hezuo@geekbang.org](mailto:hezuo@geekbang.org)

发行人 霍泰稳

内容合作 [editors@cn.infoq.com](mailto:editors@cn.infoq.com)

# 卷首语

## 在 AI 横行的时代，你为什么还要固守大前端？

作者 狼叔

我对自己的定位是 Node 全栈，以大前端为主，Node 辅助，囊括所有和用户直接相关的开发。我认为这是趋势，我也确实在自身的经历中体验到了好处。目前大前端还没有形成固定模式，还在混乱发展，所以前景是非常被看好的。当收到邀请让我写卷首语的时候，我特别想聊聊在 AI 横行的时代，你为什么还要固守大前端？

### 1) 大前端还没有天花板

从 web1.0 到 web 2.0，我们其实没太多感知，除了出现了很多 Prototype、jQuery 等框架，帮我们抹平了浏览器兼容性外，真的没有太多惊喜。至于 ria，估计早已经没多少人知道。至于 gwt 这个 Google 惊艳的尝试，也慢慢地淹没在记忆里。可能在企业开发领域，对 Ext.js 还有些许记忆，它自己实现了面向对象机制，它丰富的组件系统，快速开发经历和较好的用户体验，还是非常不错的，但对于组件化，这种探索是远远不够的。它只是面向对象层面对代码的复用。

2014 年，前端开始出现 Backbone，它是第一个知名的前端 mvc 框架，

从此开始前端开启了划时代的篇章。随后 Angular 横空出世，从 mvvm、ioc，到指令等各种概念，点燃了很多后端开发对前端的幻想。诚然，前端集成了更多的后端思想，这是好事，但从另一个角度考虑，这也是今天大前端发展过快的导火索。

随后 React 的诞生，借助于 Virtual DOM 的抽象，真正地实现了组件化，再次将组件化这个概念推向高潮。再说 Vue.js，在 Angular 火的时候，它借鉴了一部分，在 React 火的时候，它又借鉴了一些，在大家都抱怨前端越来越复杂的时候，它站出来，你想要的特性我都有，而且更简单，它的流行也是可以理解的。

移动端也面临同样的难题，既想要 H5 的灵活性，又想要有原生 App 的良好体验。这是鱼和熊掌的博弈，从 Native 到 Hybrid（还是不能满足复杂交互），到基于组件的各种方案 React Native/Weex（组件层面跨平台，写法统一，执行最终是原生代码，是折衷方案），它们丰富了端上的开发，也从另一个角度，是前端开发同学让这些创新方案落地。

今天，泛义的前端是涵盖那些熟悉 React 在做 React Native 的前端开发的。在很多大公司也都已经使用这种模式很久了，除了降低开发成本外，也拓宽了前端的更多涵盖领域。在未来，应该有更多领域被纳入到大前端概念里。

## 2) 你需要了解更多的架构知识

前端的爆发，说来也就是最近 3、4 年的事情，其最根本的创造力根源在 Node.js 的助力。Node.js 让更多人看到了前端的潜力，从服务器端开发，到各种脚手架、开发工具，前端开始沉浸在早轮子的世界里无法自拔。组件化后，比如 SSR、PWA 等辅助前端开发的快速开发实践你几乎躲不过去，再到 API 中间层、代理层，到专业的后端开发都有非常成熟的经验。

我亲历了从 Node 0.10 到 iojs，从 Node4 到目前的 Node9，写了很多文章，参加很多技术大会，做过很多次演讲，有机会和业内更多高手交流的机会。当然我也从 qunar 到阿里经历了各种 Node 应用场景，对于

Node 的前景我是非常笃定的。善于使用 Node 有无数好处，想快速出成绩，想性能调优，想优化团队结构，想人员招聘，诸多利好都让我坚定的守护 Node.js。

作为前端开发，你不能只会 Web 开发，你需要掌握 Node，你需要了解移动端开发方式，你需要对后端有更多了解。拥有更多的 Node.js 和架构知识，能够让你如鱼得水，开启大前端更多的可能性。

### 3) 不只是“端”的概念，而是用户体验

感谢苹果，将用户体验提升到了前无古人的位置。移动互联网兴起后，PC Web 日渐没落。我个人非常欣赏玉伯，在当年无线 ALL IN 战略中，他还是选择留下来继续做 PC Web 的前端。虽然公司重点转向无线，但 PC 业务一直没停。这是很多公司的现状，也是客观事实。那么，PC 端这样的“老古董”的出路到底在哪里呢？

在 AI 时代，没有“端”的支持可以么？明显是不可以的。

- 我们可以利用pc/h5快速发版本的优势，快速验证AI算法，继而为移动端提供更好的模型和数据上的支撑。
- 多端对齐，打好组合拳。既然不能在移动端有更大的突破，大家只能在细节上血拼。

今天的大前端，除了 Web 外，还包括各种端，比如移动端，ott、甚至是一些新的物联网设备。我们有理由相信 Chrome OS 当年的远见；“给我一个浏览器，我就能给你一个世界”。

当然，今天大前端还处在快速发展中，对所有程序员来说，既是机遇，也是挑战。只有站到更高的层面去架构前端，你才能收获更好的未来。前端变化快，变化多，除了拥抱变化外，狼叔最喜欢讲的一句也同样适用：“少抱怨，多思考，未来更美好”，与大家共勉。

# QCon

## 全球软件开发大会

北京·国际会议中心

演讲：2018年4月20-22日

培训：2018年4月23-24日

### 大会部分日程

时间	专题	议题	讲师
主题演讲	编程语言	Shaping the future with Java, Faster	Georges Saab Oracle / Java平台事业群VP
	架构设计	拥抱变化：演进式架构	Neal Ford ThoughtWorks 总监， 《卓有成效的程序员》作者
	流处理	Apache Kafka的过去，现在，和未来	Jun Rao Confluent / 联合创始人， Kafka作者之一
	职业成长	产品经理的发现和成长	俞军 滴滴 / 产品高级副总裁

[了解更多主题演讲 >>](#)

4月20日	数据库	MySQL的Docker容器化大规模实践	王晓波 同程艺龙 / 机票事业群CTO, TGO会员
	Java	GraalVM及其生态系统	郑雨迪 Oracle Labs / 高级研究员
	DevOps	从标准到落地：数据驱动的风险防范体系建设	华明 滴滴出行 / 运维架构师
	性能优化	分布式计算系统的性能优化	张建伟 百度 / 技术经理
	开源	OpenResty十年开源的历程和思考	温铭 OpenResty Inc. / 合伙人
	质量建设	手Q性能优化的大数据实战	谭力 腾讯科技 / 测试开发高级工程师
	管理创新	培育创新生态系统	吴穹 Agilean咨询公司 首席咨询顾问, 软件工程专家

以上仅为大会首日部分演讲，识别二维码了解更多内容

**9折** 报名倒计时，立减680元  
团购享受更多优惠

访问官网获取更多前沿技术趋势

2018.qconbeijing.com

如有任何问题，欢迎咨询

电话：15110019061，微信：qcon-0410



# GMTC 2018

## 全球大前端技术大会

—— 大 前 端 的 下 一 站 ——

聚集12大热门专题

UI与动画

PWA

终端AI

工程化

语言专场

Weex

性能优化

WebRTC

Android专场

iOS专场

Node专场

Web框架

时间：6月21-22日 地点：北京·国际会议中心

票务咨询

微信：18514549229  
Q Q：209463896

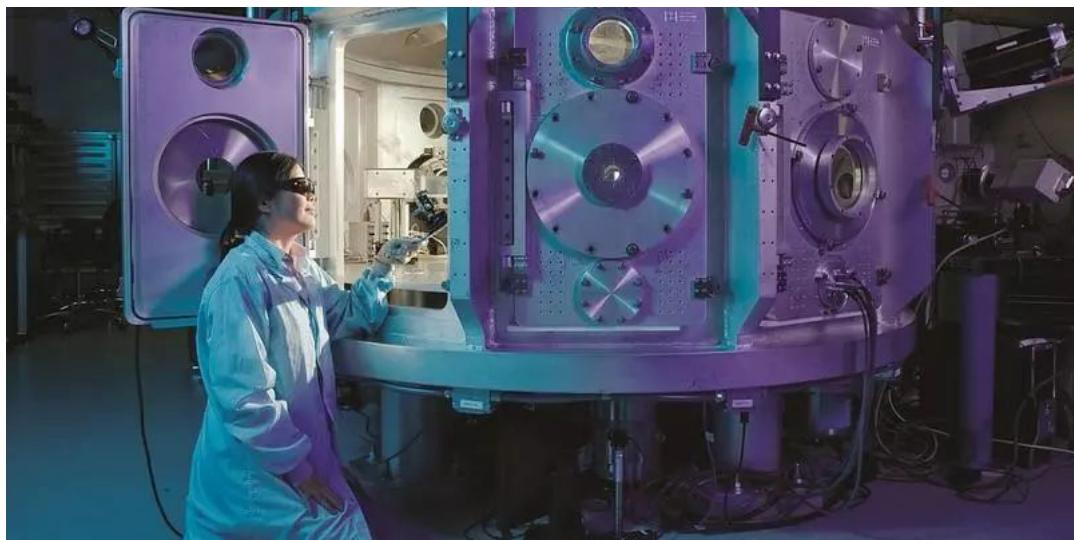
5月27日前购票，**8折**优惠  
(团购享更多优惠)

扫码了解更多信息



# 伯克利推出世界最快的 KVS 数据库 Anna：秒杀 Redis 和 Cassandra

编辑 蔡芳芳



题外话：RISE 实验室的前身是赫赫有名的伯克利 AMP 实验室，该实验室曾开发出了一大批大获成功的分布式技术，这些技术对高性能计算产生了深远的影响，包括 Spark、Mesos、Tachyon 等。如今，原 AMP 实验室博士生，同时也是 Spark 和 Mesos 核心作者之一的 Matei 已经转身去了斯坦福，并于去年年底推出了以普及机器学习实践为目的的开源项目 DAWN（详见 AI 前线报道），而 RISE 实验室也在没多久后推出了志在取代 Spark 的新型分布式执行框架 Ray（详见 AI 前线报道）。

过去几年，RISE 实验室把研究重点放在如何设计一个无需协调的分布式系统上。他们提出了 CALM 基础理论 (<http://db.cs.berkeley.edu/>)

[papers/sigrec10-declimperative.pdf](http://papers.sigrec10-declimperative.pdf)），设计出了新编程语言 Bloom (<http://bloom-lang.net/>)，开发出了跨平台程序分析框架 Blazes (<https://arxiv.org/pdf/1309.3324.pdf>)，发布了事务协议 HATs (<http://www.vldb.org/pvldb/vol8/p185-bailis.pdf>)。但在推出 Anna 之前，他们还未就这些理论、语言、框架或协议在多核环境下或云环境中能够提供怎样的性能有过任何测试或评估。

而 Anna 的推出正好印证了他们之前的研究成果。Anna 的论文显示，在单个 AWS 实例上，Anna 的速度是 Redis 的 10 倍。而在一个标准的交互式基准测试中，也以 10 倍的速度打败了 Cassandra。为了获得更多的比较结果，他们还拿 Anna 与其他主流的键值系统进行了性能对比：比 Masstree 快 700 倍，比英特尔的“无锁”TBB 哈希表快 800 倍。当然，Anna 并没有提供类似其他键值系统那样的线性一致性。不过，Anna 使用了本地缓存存放私有状态，仍然提供了极佳的无协调一致性，比“hogwild”风格的 C++ 哈希表要快上 126 倍。而且一旦到了云端，Anna 更是独领风骚，其他的系统无法真正提供线性伸缩，但 Anna 却可以。

Anna 的性能和伸缩性主要归功于它的完全无协调机制，节点工作进程有 90% 的工作负载是在处理请求，而其他大部分系统（如 Masstree 和 英特尔的 TBB）只有不到 10% 的时间在处理请求，它们其余的 90% 时间花在了等待协调上。不仅如此，其他系统因为使用了共享内存，还会出现处理器缓存击穿问题。

Anna 不仅速度快，在一致性方面也达到了很高的水准。多年前，他们发布的事务协议 HATs 就已表明，无协调的分布式一致性和事务隔离性存在很大的提升空间，包括级联一致性和读提交事务级别。Anna 将 Bloom 的单格子组合设计模式移植到了 C++ 中，是第一个实现了上述所有级别一致性的系统。当然，也是因为设计上的简洁，才能达到如此快的速度。

RISE 的研究员们在设计 Anna 的过程中学到了很多，它们已经远远超出了一个键值数据库的范畴，可以被应用在任何一个分布式系统上。他

们正基于 Anna 开发一个新的扩展系统，叫作 Bedrock。Bedrock 运行在云端，提供了无需人工干预、低成本的键值存储方案，而且是开源的。

## Anna 架构简析

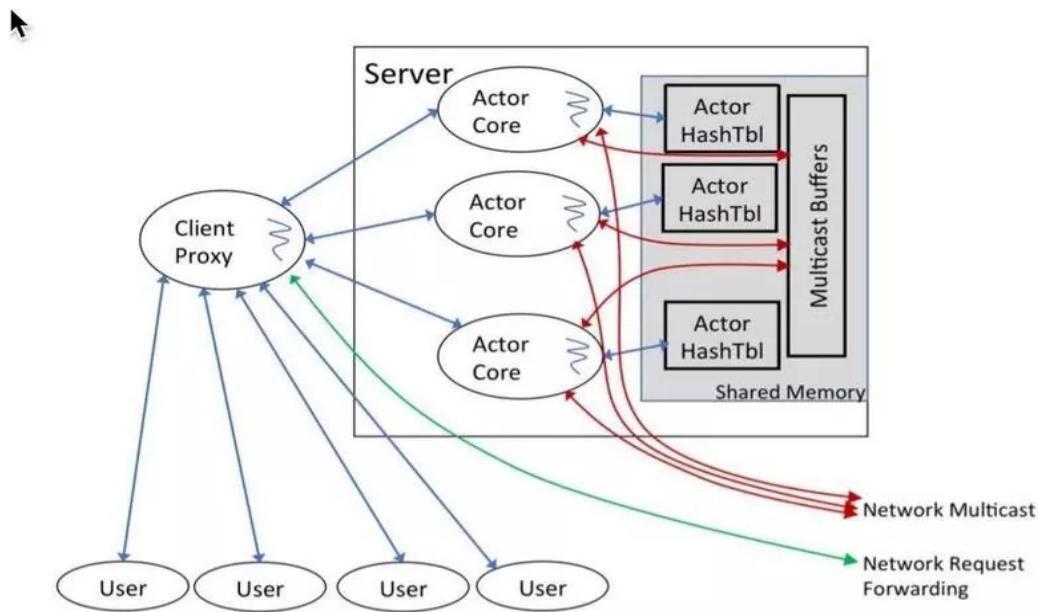


图 1：Anna 架构

上图是 Anna 单节点的架构图。Anna 服务器由一系列独立的线程组成，每个线程运行无协调的 actor。每个线程对应一个 CPU 核心，线程数量不超过 CPU 的总核数。客户端代理负责将远程请求分发给 actor，每个 actor 都有一个私有的哈希表，这些哈希表存放在共享内存中。线程间的变更通过内存广播进行交换，而服务器间的变更则通过 protobuf 进行交换。

这种线程和 CPU 核心一对一的模型避免了上下文切换开销。actor 之间不共享键值状态，通过一致性哈希对键空间进行分区，并使用多主复制机制在 actor 之间复制数据分区，而且复制系数是可配置的。actor 基于时间戳将键的更新通知给其他 actor 副本，每个 actor 有自己的私有状态，这个状态保存在一个叫作“格子”的数据结构中，确保在消息发生延迟、重排或重复时仍然能够保证一致性。

## Anna 性能评测

下面就 Anna 的无协调 actor 模型在多核 CPU 上的并行能力、多服务器伸缩能力进行评测，并将它与其他流行的键值数据库进行对比。

### 无协调 actor 模型的伸缩性

在无协调 actor 模型中，每个 actor 对应一个线程，对任何一个共享状态都有自己的一份私有拷贝，并通过异步广播将更新通知给其他 actor。在多核服务器上，这种模型比传统的共享内存模型的性能要高出一个数量级。

为此，RISE 研究员设计了一个对比实验，将 Anna 与其他基于共享内存的 TBB、Masstree 和自己实现的一个键值存储系统（姑且把它叫作“Ideal”）进行了对比。他们在 AWS 的 m4.16xlarge 实例上运行实验，每个实例配备了 32 核的 CPU。实验中使用了 1 百万个键值对，键的大小为 8 字节，值的大小为 1KB。在实验过程中，他们基于 zipfian 分布和各种系数生成不同的压力负载，模拟不同层次的冲突。

在第一次实验中，他们对比了 Anna 与 TBB、Masstree 和 Ideal 在单台服务器上的吞吐量。他们逐渐增加线程数量，直到达到服务器 CPU 核数的上限，并观察它们的吞吐量。

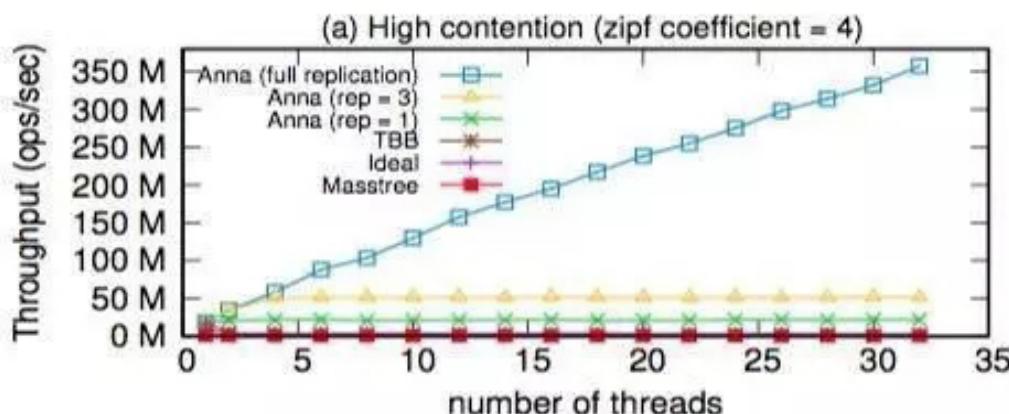


图 2

KVS	RH	AI	LM	M	O	CM
Anna (full)	90%	0%	4%	4%	2%	1.1
Anna (rep=3)	91%	0%	5%	2%	2%	1
Anna (rep=1)	94%	0%	5%	0%	1%	1
Ideal	97%	0%	0%	0%	3%	17
TBB	4%	95%	0%	0%	1%	19
Masstree	7%	92%	0%	0%	1%	16

(a) High Contention

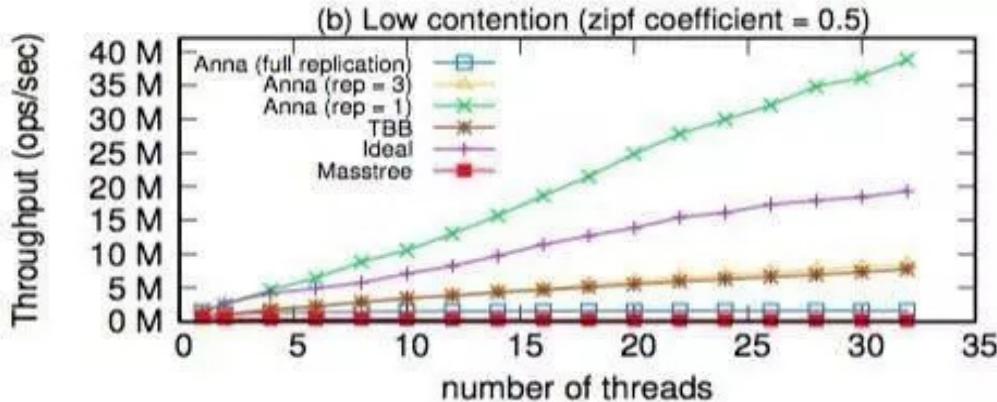
图 3

图 2 是在高并发情况下，线程数与吞吐量的变化关系，其中 zipf 系数为 4。图 3 是在高并发情况下，CPU 时间的使用情况。CPU 时间被分为 5 类：处理请求 (RH)、原子指令 (AI)、合并格子 (LM)、广播 (M) 和其他。最右边一栏是 L1 缓存击穿数量 (CM)。

从图中可以看出，在这样的负载压力下，TBB 和 Masstree 几乎失去了并行能力。因为大部分是更新操作，针对同一个键值的并行更新操作会被串行化，它们需要同步机制来防止多个线程同时更新同一个键值。因此，随着线程数量的增加，它们性能也只能趋于平缓。Ideal 虽然比 TBB 和 Masstree 的性能高出 6 倍，但相比 Anna，还是差了很多。尽管它没有使用同步机制，但在多个线程修改共享内存地址时，仍然存在缓存一致性方面的开销。

相反，在 Anna 中，更新操作是针对本地状态进行的，不需要进行同步，并定时通过广播进行变更交换。在高并发情况下，尽管它的性能仍然受限于其复制系数，但比基于共享内存的系统要好很多。Anna 有 90% 的 CPU 时间用于处理请求，花在其他方面的时间则很少。可见，Anna 的无协调 actor 模型解决了键值存储系统在多核环境里的伸缩性难题。

图 4 是在低并发情况下，线程数与吞吐量的变化关系，其中 zipf 系数为 0.5。图 5 是在低并发情况下，CPU 时间的使用情况，其中最右边一列表示内存占用 (MF)。



KVS	RH	AI	LM	M	O	MF
Anna (full)	3%	0%	3%	92%	2%	32
Anna (rep=3)	25%	0%	4%	69%	2%	3
Anna (rep=1)	93%	0%	5%	0%	2%	1
Ideal	97%	0%	0%	0%	3%	32
TBB	70%	26%	0%	0%	4%	32
Masstree	20%	78%	0%	0%	2%	32

(b) Low Contention

图 4 | 图 5

当复制系数为 1 时，Anna 因为内存占用率极低而获得了更好的伸缩性。不过，随着复制系数的增加（增加到 3），吞吐量出现了明显下降（下降了四分之三）。这里有两个原因。首先，增加复制系数会占用更多的内存，而且在低并发的情况下，唯一键的更新操作大量增加，所以无法通过合并的方式进行变更交换。图 5 显示，当复制系数为 3 时，Anna 有 69% 的 CPU 时间用于处理广播变更。而在使用完整的复制系数时，Anna 也停止了伸缩，因为此时相当于每个线程只能处理一个请求。不过，尽管 TBB 和 Masstree 没有广播开销，但在内存占用和同步操作方面仍然存在大量开销。因此，从这个实验中可以得出这样的结论：对于一个支持多主复制的系统来说，在低并发量情况下使用高复制系数对性能是一种伤害。

图 6 是在多个服务器上增加线程数时的吞吐量变化情况。Anna 的复制系数设置为 3，先是启动第一台服务器的 32 个线程，然后是第二台服

务器的 32 个线程，最后是第三台服务器的所有剩余线程。从图中可以看出，Anna 的吞吐量随着线程数量的增加呈线性增长。在启动第 33 个线程时吞吐量有轻微下降，不过那是因为第 33 个线程是属于第二台服务器的。但从整体来看，吞吐量的增长是很稳定的。可见，借助 Anna 的无协调 actor 模型，是可以实现吞吐量线性增长的。

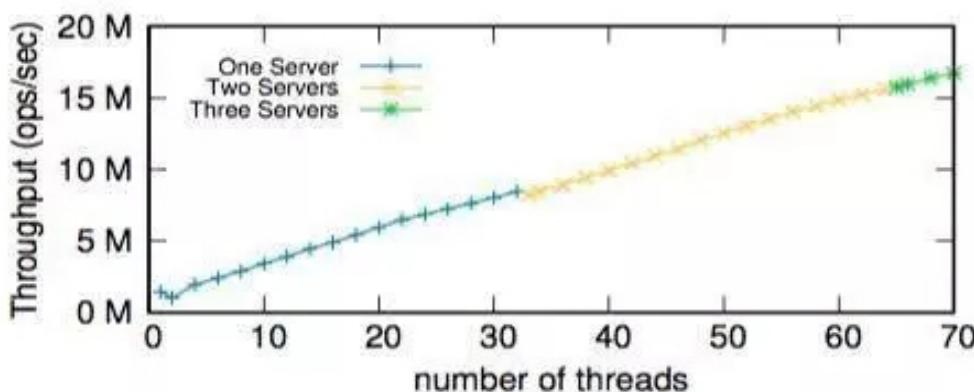


图 6

## 与其他系统的比较

为对比 Anna 与其他流行键值系统之间的性能差异，RISE 研究员设计了两次实验，第一次在单节点上与 Redis 进行对比，第二次在一个大型的分布式系统上与 Cassandra 进行对比。

Anna 具有多线程并行能力，而 Redis 使用的是单线程模型。所以在第一次实验中，他们在同一台服务器上搭建了一个 Redis 集群，让 Anna 与这个集群进行比较。实验是在 AWS 的 EC2 实例上运行的，其中 Redis 集群使用了尽可能多的线程。

从图 7(a) 可以看出，在高并发情况下，Redis 集群的整体吞吐量几乎保持不变，而 Anna 可以在副本之间分散热键。Anna 的吞吐量随着复制系数的增加而增长，直到达到平缓。如果热键完全被复制，吞吐量还会随着线程的增加继续增长。从图 7(b) 可以看出，在低并发情况下，Anna 和 Redis 集群都获得了不错的并行能力，它们的吞吐量都随着线程数的增加而增长。

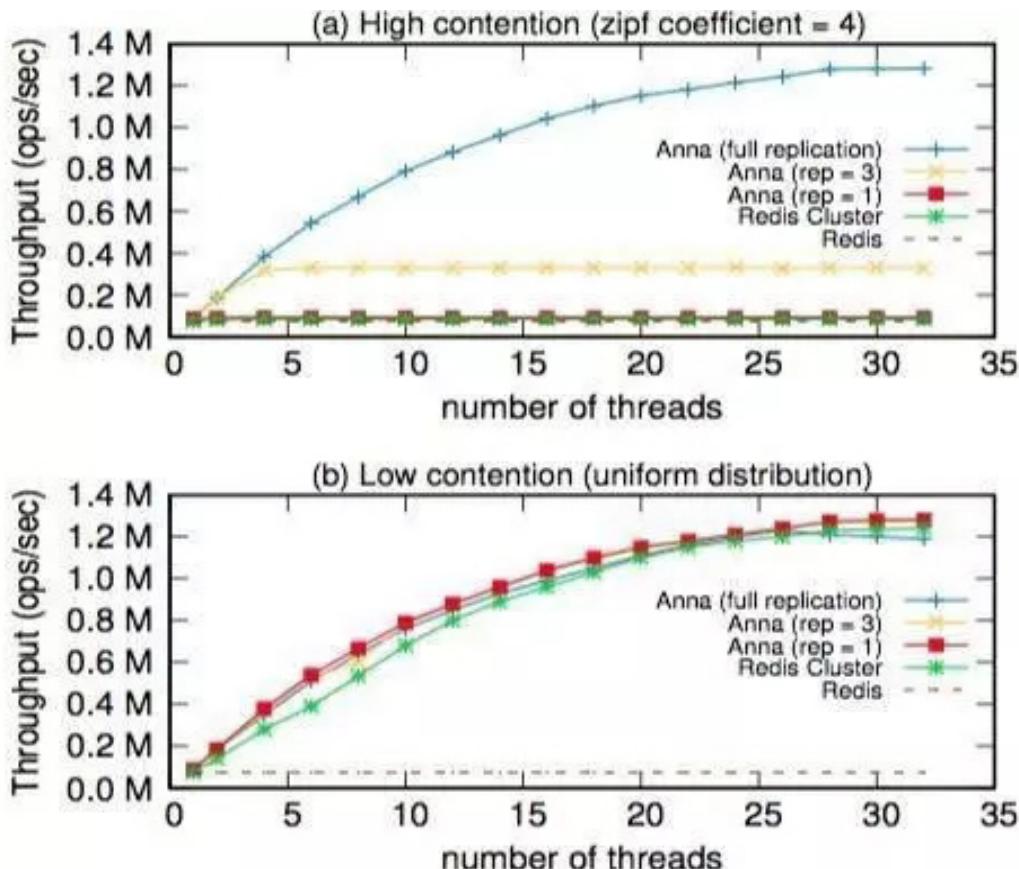


图 7

从这次实验可以看出，在高并发情况下，Anna 通过复制热键的方式在性能方面吊打 Redis 集群，而在低并发情况下，Anna 可以与 Redis 集群达到相似的性能。

在第二次实验中，RISE 研究员将 Cassandra 的一致性设置为最低级别（ONE），也就是说，只要一个节点确认就表示更新操作成功。他们在 AWS 的四个 EC2 可用区域（奥尔良、北弗吉尼亚、爱尔兰和东京）上运行该实验，并通过调整可用区域的节点数量来评测它们的伸缩性。它们的复制系数都被设置为 3。

从图 8 可以看出，随着节点的增加，Anna 和 Cassandra 的性能都呈现出线性增长。不过，Anna 比 Cassandra 的性能高出一大截。事实上，在每个节点使用 4 个线程时，Anna 就可以打败 Cassandra，而当把所有的线程都用上，Anna 比 Cassandra 的性能高出 10 倍以上。

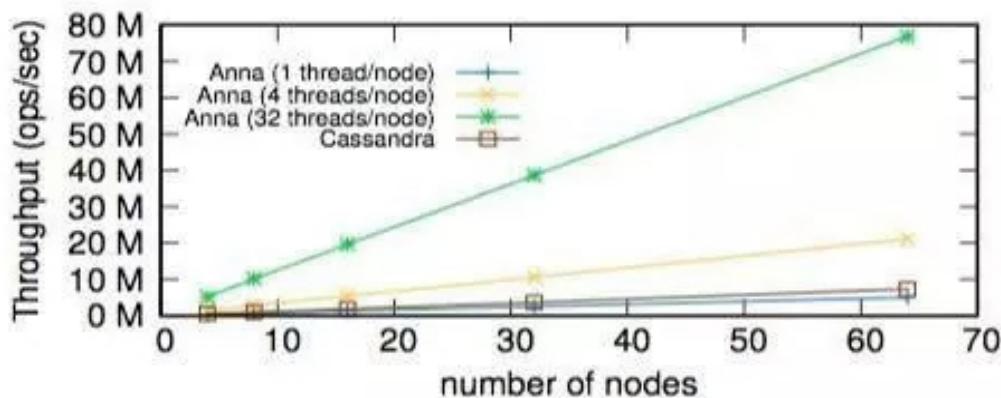


图 8

参考资料: <https://rise.cs.berkeley.edu/blog/anna-kvs/?twitter=@bigdata>

论文原文: [http://db.cs.berkeley.edu/jmh/papers/anna\\_ieee18.pdf](http://db.cs.berkeley.edu/jmh/papers/anna_ieee18.pdf)

# Spring Boot 2.0 正式发布，新特性解读

作者 翟永超

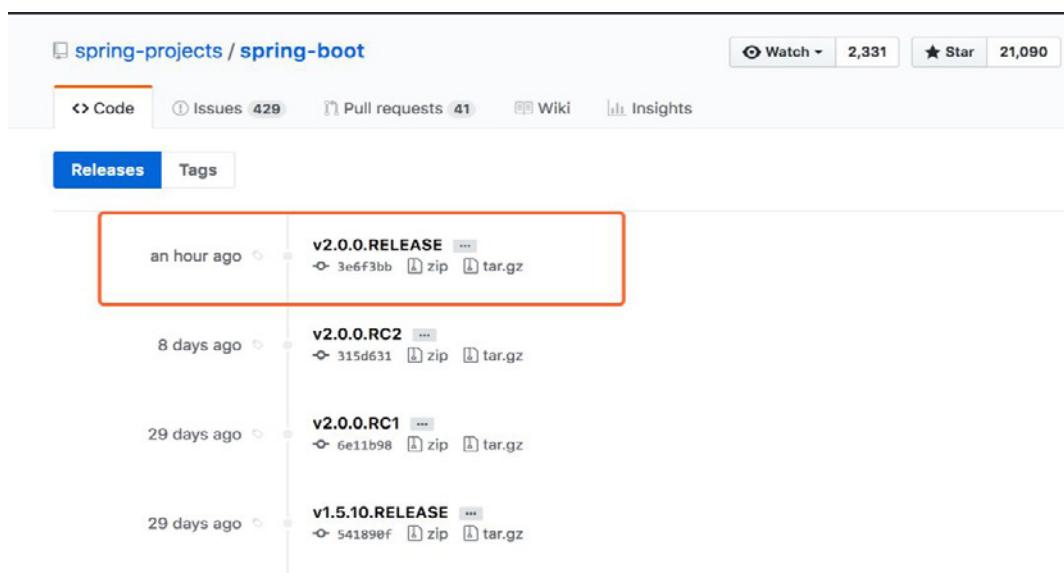


## 写在前面

北京时间 3 月 1 日，经过漫长的等待之后，Spring Boot 2.0 正式发布。作为 Spring 生态中的重要开源项目，Spring Boot 旨在简化创建产品级的 Spring 应用和服务。用户只需要 "run" 就能创建一个独立的，产品级别的 Spring 应用。

一经发布，Spring Boot 就迅速受到了开发者的青睐，到目前为止，它已经有超过 2 万个 Star，1.6 万个 fork（2017 年 GitHub 排名前十）。而 Spring Boot 2.0 的酝酿已有一段时间，从去年 5 月 16 日发布 M1 版本，再到后来的 RC 版本，也已有近 1 年时间。

Spring 2.0 中引入了众多令人激动的新特性，包括支持 Java 9、HTTP/2、基于 Spring 5 构建、强力集成 GSON 等等。为了了解 Spring Boot 的整体发展历史，以及 2.0 中的重要更新，InfoQ 特邀请到 Spring Boot 专家、永辉云创架构师翟永超撰文解读。



## Spring 帝国

Spring 几乎是每一位 Java 开发人员都耳熟能详的开发框架，不论你是一名初出茅庐的程序员还是经验丰富的老司机，都会对其有一定的了解或使用经验。在现代企业级应用架构中，Spring 技术栈几乎成为了 Java 语言的代名词，那么 Spring 为什么能够在众多开源框架中脱颖而出，成为业内一致认可的技术解决方案呢？我们不妨从最初的 Spring Framework 开始，看看它为什么能够横扫千军，一统江湖！

### 挑战权威，一战成名

2004 年 3 月，Spring 的第一个版本以及其创始人 Rod Johnson 的经典力作《Expert one-on-one J2EE Development without EJB》发布，打破了当时 Java 开发领域的传统思考模式，企业级应用开始走向“轻量化”发展的步伐。

最初的 Spring Framework 1.0 并不像如今的 Spring 那么复杂，但是在该版本中已经包含了 Spring 中最为核心的两大要素：依赖注入（IOC）和面向切面编程（AOP），这两个功能是 Spring 区别于其他优秀框架，并在企业级应用中建立核心地位的关键所在。

很多开发者在初涉 Java 应用的时候很可能会觉得这两个功能的意义并不大，因为不用它们我们依然可以很好的实现业务功能，事实也确实如此，但是随着业务的迭代和开发的深入，复杂多变的需求开始慢慢侵蚀原本“完美”的架构，开发与测试的难度逐步增大，往往在这个时候，我们才体会到了 Spring 的价值。

所以，即便在 Spring 的最初版本中也封装了诸多偏业务型的功能封装，如：邮件发送、事务管理等，但我们要知道真正让企业级应用离不开 Spring 的理由并不是这些与业务直接相关的功能，而是上面所提及的与业务实现毫不相关的两大核心。

由于在初期版本中 Spring 对很多功能性封装并没有今天的 Spring 那么强大，所以很长一段时间，我们都采用了 Spring 做工程管理来整合其他更优秀的功能型框架来完成系统开发的架构模式，比如曾经风靡一时的 Spring + Struts + Hibernate 架构，相信可以勾起一代人的回忆。

## 优雅灵活，吸粉无数

Spring 在发布并获得业界的普遍认可之后，Spring 开源社区变得异常活跃，除了社区自身不断对 Spring 进行增强之外，其他功能性框架也纷纷对 Spring 进行适配与支持。在随后发布的 Spring 2.x 和 3.x 中，先后支持了 Annotation 的优雅配置方式以及更为灵活的 Java 类的配置，这使得 Spring 在管理 Bean 的配置方式上变得更为多样化。

但是随着 Spring 的深入应用，繁琐的配置问题也开始显现，我们会发现每次在构建项目的时候总是在不断的复制黏贴着一些模版化的配置与代码，有时候我们只是想实现几个很简单的功能，结果配置内容远大于业务逻辑代码的编写；同时，在框架整合过程中，对于一些共同依赖的 Jar

包存在着潜在的冲突风险，使得一些复杂的整合任务变得困难起来。所以，Spring 的“轻量级”在其他动态语言面前就显得不那么轻了。

## 轮子大师，前途未卜

在之后的 Spring 4.x 中除了提供对 Java 8 的支持以及对依赖注入的增强之外，有很长一段时间，Spring 社区对其核心框架的创新就没有那么出彩了，社区更多的精力开始将矛头转向了曾经那些亲密无间的小伙伴们。于是，我们在 Spring 社区发现多出了各种功能性的兄弟项目，比如：简化数据访问的 Spring Data、提供批处理能力的 Spring Batch、用于保护应用安全的 Spring Security 等。

虽然这些框架从个体来说都有一定的优势和先进的理念，但是对于很多既有系统来说，在功能性框架上很难做出改变，对于这些新生的轮子项目就很难得到应用，除了一些从零开始的系统会做一些尝试之外，鉴于学习成本和踩坑风险的考虑，中小团队对这些新项目很少有愿意去尝试的。所以，一些老牌的功能性框架除非有严重的性能或安全问题出现，不然很难被这些轮子所替代。

在这段时间里，虽然 Spring 社区推出了那么多的轮子项目，但是真正在国内得到广泛应用的并不多，很多开发团队依然只是使用最核心的 IOC 和 AOP，并根据自己团队的技术栈情况整合出更适合自身的脚手架来进行系统开发。

## 神兵出世，再创辉煌

2014 年 4 月 1 日，Spring Boot 发布了第一个正式版本。该项目旨在帮助开发者更容易地创建基于 Spring 的应用程序和服务，使得现有的和新的 Spring 开发者能够最快速地获得所需要的 Spring 功能。一直到今天发布 2.x 版本，共经历了近 4 年的发展，Spring Boot 已经是一个拥有了 21000 多 Star，15000 多次 Commits，贡献者超过 400 多名的超热门开源项目。

Spring Boot 为什么突然如此备受关注与推崇呢？主要有以下几点：

- 简化依赖管理：在 Spring Boot 中提供了一系列的 Starter POMs，将各种功能性模块进行了划分与封装，让我们可以更容易的引入和使用，有效的避免了用户在构建传统 Spring 应用时维护大量依赖关系而引发的 JAR 冲突等问题。
- 自动化配置：Spring Boot 为每一个 Starter 都提供了自动化的 Java 配置类，用来替代我们传统 Spring 应用在 XML 中繁琐且并不太变化的 Bean 配置；同时借助一系列的条件注解修饰，使得我们也能轻松的替换这些自动化配置的 Bean 来进行扩展。
- 嵌入式容器：除了代码组织上的优化之外，Spring Boot 中支持的嵌入式容器也是一个极大的亮点（此处仿佛又听到了 Josh Long 的那句：“Deploy as a Jar, not a War”），借助这个特性使得 Spring Boot 应用的打包运行变得非常的轻量级。
- 生产级的监控端点：spring-boot-starter-actuator的推出可以说是 Spring Boot 在 Spring 基础上的另一个重要创新，为 Spring 应用的工程化变得更加完美。该模块并不能帮助我们实现任何业务功能，但是却在架构运维层面给予我们更多的支持，通过该模块暴露的 HTTP 接口，我们可以轻松的了解和控制 Spring Boot 应用的运行情况。

Spring Boot 虽然是基于 Spring 构建的，但是通过上面这些特性的支持，改变了我们使用 Spring 的姿势，极大得简化了构建企业级应用的各种配置工作，尤其对于很多初学者来说，变得更加容易入门使用。

## 如约而至，升级与否？

万众期待的 Spring Boot 2.0 终于发布了第一个正式版本，为什么 Spring Boot 2.0 如此受期待呢？我认为主要有以下几个原因：

1. 支持最新的 Java 9；
2. 基于 Spring 5 构建，Spring 的新特性均可以在 Spring Boot 2.0 中使用；

3. 为各种组件的响应式编程提供了自动化配置，如：Reactive Spring Data、Reactive Spring Security 等；
4. 支持 Spring MVC 的非阻塞式替代方案 WebFlux 以及嵌入式 Netty Server；
5. Spring Boot 2.0 的发布，Spring Cloud Finchley 还会远吗？

上述列举的内容是笔者主要关心的重要内容，并非 Spring Boot 2.0 所有的新特性，对于不同的使用者来说相信会有不同的关注点。

除此之外，在 Spring Boot 2.0 中还有非常多其他令人振奋的新特性，比如：对 HTTP/2 的支持、新增了更灵活的属性绑定 API（可以不通过 @ConfigurationProperties 注解就能实现配置内容读取和使用）、对 Spring Security 整合的简化配置、Gradle 插件的增强、Actuator 模块的优化等等。

本文不对这些新特性做详细的介绍，下面主要说说，我们是否有必要将我们的 Spring Boot 1.x 升级到 Spring Boot 2.x，在这过程中，我们需要考虑和注意哪些问题。

## Java 版本要求的变化

我们在选择是否要升级 Spring Boot 的时候，最先需要考虑的是 Java 版本的选择。在 Spring Boot 2.0 中提高了对 Java 版本的要求，我们需要至少使用 Java 8 才能使用它，如果你的 Spring Boot 应用还运行在 Java 7 上，那就还得考虑 Java 的升级成本。

另外，在未来的一段时间内，你是否想要使用 Java 9 将是一个影响升级与否的重要决策依据，因为 Spring Boot 1.x 版本明确说明了没有对 Java 9 的支持计划；换言之，如果你想将 Spring Boot 运行在 Java 9 上，那么你必须升级到 Spring Boot 2.0。

Tips：当前版本的 Spring Boot 2.0 虽然支持 Java 9，但是依然还有一些问题。比如：JDK 的代理支持需要使用 AspectJ 1.9，但是该版本还处于 RC 版；还不支持 Apache Cassandra；对于 JSP TLDs 在嵌入式 Tomcat 中也无法支持等情况。对于这些问题的具体处理方法[参见这里](#)。

## 依赖组件的升级

Spring Boot 的 Starter 中整合了不少优秀的第三方组件，这些组件的升级也需要我们做好一定的考量，在这些组件的版本升级过程中，使用上是否有变化等问题。其中，最为关键的几个组件需要我们注意：

- Tomcat 升级至 8.5
- Flyway 升级至 5
- Hibernate 升级至 5.2
- Thymeleaf 升级至 3

Tips：前几日曝出的 Tomcat 漏洞问题。经查 Spring Boot 2.0 选用的版本为 8.5.28，属于安全版本，所以大家可以放心使用。

## 依赖重组和配置重定位

在 Spring Boot 2.0 的升级过程中，可能这部分内容将是大家要做出较多修改的地方，所以建议大家在这里留个心眼。由于 Spring Boot 在构建 Starter POMs 的时候并非是扁平的一层结构，一些功能模块 Starter 之间是存在包含引用关系的，比如：spring-boot-starter-thymeleaf 中包含了 spring-boot-starter-web，因为 thymeleaf 模版引擎之前肯定是在 Spring MVC 下使用的。

但是，在 Spring Boot 2.0 中，WebFlux 的出现对于 Web 应用的解决方案将不再唯一，因此 spring-boot-starter-thymeleaf 中的依赖就不在包含 spring-boot-starter-web，开发人员需要自己添加 spring-boot-starter-web 或 spring-boot-starter-webflux 来决定是使用哪个模块实现 Web 应用。

除了类似上面的依赖重组之后，在 Spring Boot 2.0 中对于配置属性的重定位也是比较多的，这将导致一些原有的配置将不再生效，需要我们手工的去修改这些配置的 Key 来完成升级适配。比如，一些与 servlet 相关的 server.\* 属性重定位到 server.servlet 前缀下：

Old property	New property
server.context-parameters.*	server.servlet.context-parameters.*
server.context-path	server.servlet.context-path
server.jsp.class-name	server.servlet.jsp.class-name

server.jsp.init-parameters.*	server.servlet.jsp.init-parameters.*
server.jsp.registered	server.servlet.jsp.registered
server.servlet-path	server.servlet.path

更多的依赖变化、配置重定位以及默认配置的变化，读者可自行查阅[官方升级手册](#)。

## 不必要的顾虑

之前有朋友在 spring4all 社区上问：如果 Spring Boot 升级 2.0，2.0 出了那么多新功能，我们的业务代码是否也需要随之修改，风险会不会很大？其实，这个问题大家完全不用太多的顾虑，Spring Boot 2.0 虽然新增了很多强大的新特性，但是对于原有功能的支持并没有抛弃。所以，就算我们不用任何类似 WebFlux 这样的新功能，将工程升级到了 Spring Boot 2.0 之后，继续使用 Spring MVC 开发我们的项目也是完全没有影响的。只是，就如上面所述的，我们可能需要做一些依赖和配置上的调整才能继续将应用正常的运行起来。

## 总结与展望

感谢大家能够读完上面我对 Spring Boot 2.0 的薄见，希望这些内容能够对你在 Spring Boot 2.0 的选择上有一定的参考价值。这个版本虽然不像 Spring Boot 1.0 那样颠覆我们对繁琐的 Spring 应用的认识，但是依然透露着很多时代前沿的气息。同时，Spring Boot 2.0 的发布，也意味着 Spring Cloud Finchley 离正式发布又近了一步，因为这个版本中同样的将会带来很多令人兴奋的内容，相信这一天的到来也不远了！

对于当前 Spring Boot 2.0 的迁移升级，作为一名 Spring Boot 与 Spring Cloud 的忠实拥护者，在时间允许的情况下，这是一件必然会去尝试的事情，在未来的时间里，我也尽可能的希望抽出时间继续分享一些其中的问题与收获，与大家共勉！

## 参考资料

Spring Boot 2.0 Release Notes: <https://github.com/spring-projects/>

[spring-boot/wiki/Spring-Boot-2.0-Release-Notes](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Release-Notes)

Spring Boot 2.0 Migration Guide: <https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide>

Running Spring Boot on Java 9: <https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-with-Java-9>

## 作者介绍

翟永超，供职于永辉云创，任架构师，负责 Spring Cloud 微服务架构的落地。《Spring Cloud 微服务实战》作者，spring4all 社区发起人。

# Java 10 正式发布，带来了这些新特性

作者 张建锋



北京时间 3 月 21 日，Oracle 官方宣布 Java 10 正式发布。这是 Java 大版本周期变化后的第一个正式发布版本（详见[这里](#)），非常值得关注。你可以即刻[下载](#)试用。

去年 9 月，Oracle 将 Java 大版本周期从原来的 2-3 年，调整成每半年发布一个大的版本。而版本号仍延续原来的序号，即 Java 8、Java 9、Java 10、Java 11.....

但和之前不一样的是，同时还有一个版本号来表示发布的时间和是否为 LTS（长期支持版本），比如 Java 10 对应 18.3。如下示例：

```
/jdk-10/bin$ ./java -version
```

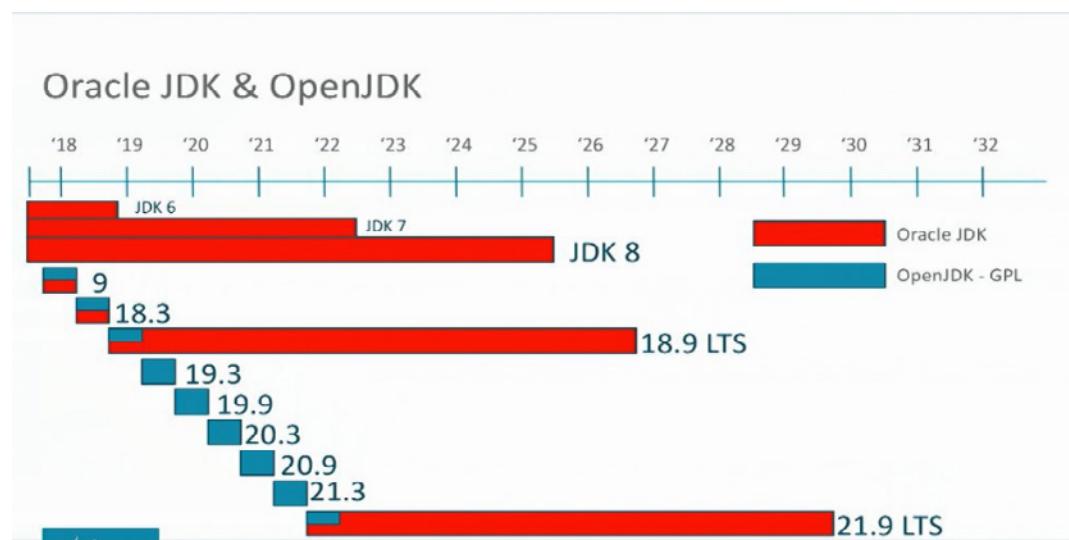
```
openjdk version "10" 2018-03-20
OpenJDK Runtime Environment 18.3 (build 10+46)
OpenJDK 64-Bit Server VM 18.3 (build 10+46, mixed mode)
```

需要注意的是 Java 9 和 Java 10 都不是 LTS 版本。和过去的 Java 大版本升级不同，这两个只有半年左右的开发和维护期。而未来的 Java 11，也就是 18.9 LTS，才是 Java 8 之后第一个 LTS 版本（得到 Oracle 等商业公司的长期支持服务）。

这种发布模式已经得到了广泛应用，一个成功的例子就是 Ubuntu Linux 操作系统，在偶数年 4 月的发行版本为 LTS，会有很长时间的支持。如 2014 年 4 月份发布的 14.04 LTS，Canonical 公司和社区支持到 2019 年。类似的，Node.js，Linux kernel，Firefox 也采用类似的发布方式。

Java 未来的发布周期，将每半年发布一个大版本，每个季度发布一个中间特性版本。这样可以把一些关键特性尽早合并入 JDK 之中，快速得到开发者反馈，可以在一定程度上避免 Java 9 两次被迫推迟发布日期的尴尬。

下图为 2017 年 JavaOne 大会时，Oracle 公开的未来 Java 版本发布和支持周期图。



## Java 10 新特性

这次发布的 Java 10，新带来的特性并不多。

根据官网公开资料，共有 12 个 JEP(JDK Enhancement Proposal 特性加强提议)，带来以下加强功能。

- JEP286，var 局部变量类型推断。
- JEP296，将原来用 Mercurial 管理的众多 JDK 仓库代码，合并到一个仓库中，简化开发和管理过程。
- JEP304，统一的垃圾回收接口。
- JEP307，G1 垃圾回收器的并行完整垃圾回收，实现并行性来改善最坏情况下的延迟。
- JEP310，应用程序类数据 (AppCDS) 共享，通过跨进程共享通用类元数据来减少内存占用空间，和减少启动时间。
- JEP312，ThreadLocal 握手交互。在不进入到全局 JVM 安全点 (Safepoint) 的情况下，对线程执行回调。优化可以只停止单个线程，而不是停全部线程或一个都不停。
- JEP313，移除 JDK 中附带的 javah 工具。可以使用 javac -h 代替。
- JEP314，使用附加的 Unicode 语言标记扩展。
- JEP317，能将堆内存占用分配给用户指定的备用内存设备。
- JEP317，使用 Graal 基于 Java 的编译器，可以预先把 Java 代码编译成本地代码来提升效能。
- JEP318，在 OpenJDK 中提供一组默认的根证书颁发机构证书。开源目前 Oracle 提供的 Java SE 的根证书，这样 OpenJDK 对开发人员使用起来更方便。
- JEP322，基于时间定义的发布版本，即上述提到的发布周期。版本号为 \\$FEATURE.\\$INTERIM.\\$UPDATE.\\$PATCH，分别是大版本，中间版本，升级包和补丁版本。

## 部分特性说明

## 1. var 类型推断

这个语言功能在其他一些语言 (C#、JavaScript) 和基于 JRE 的一些语言 (Scala 和 Kotlin) 中，早已被加入。

在 Java 语言很早就在考虑，早在 2016 年正式提交了 JEP286 提议。后来举行了一次公开的开发者调查，获得最多建议的是采用类似 Scala 的方案，“同时使用 val 和 var”，约占一半；第二多的是“只使用 var”，约占四分之一。后来 Oracle 公司经过慎重考虑，采用了只使用 var 关键字的方案。

有了这个功能，开发者在写这样的代码时：

```
ArrayList<String> myList = new ArrayList<String>()
```

可以省去前面的类型声明，而只需要

```
var list = new ArrayList<String>()
```

编译器会自动推断出 list 变量的类型。对于链式表达式来说，也会很方便：

```
var stream = blocks.stream();
...
int maxWeight = stream.filter(b -> b.getColor() == BLUE)
    .mapToInt(Block::getWeight).max();
```

开发者无须声明并且 import 引入 Stream 类型，只用 stream 作为中间变量，用 var 关键字使得开发效率提升。

不过 var 的使用有众多限制，包括不能用于推断方法参数类型，只能用于局部变量，如方法块中，而不能用于类变量的声明，等等。

另外，我个人认为，对于开发者而言，变量类型明显的声明会提供更加全面的程序语言信息，对于理解并维护代码有很大的帮助。一旦 var 被广泛运用，开发者阅读三方代码而没有 IDE 的支持下，会对程序的流程执行理解造成一定的障碍。所以我建议尽量写清楚变量类型，程序的易读维护性有时更重要一些。

## 2. 统一的 GC 接口

在 JDK10 的代码中，路径为 openjdk/src/hotspot/share/gc/，各个

GC 实现共享依赖 shared 代码，GC 包括目前默认的 G1，也有经典的 Serial、Parallel、CMS 等 GC 实现。

		Latest commit 48bd324 17 days ago
..		
■ cms	8195142: Refactor out card table from CardTableModRefBS to flatten th...	22 days ago
■ g1	Merge	17 days ago
■ parallel	8195142: Refactor out card table from CardTableModRefBS to flatten th...	22 days ago
■ serial	8195142: Refactor out card table from CardTableModRefBS to flatten th...	22 days ago
■ shared	8195142: Refactor out card table from CardTableModRefBS to flatten th...	22 days ago

### 3. 应用程序类数据 (AppCDS) 共享

CDS 特性在原来的 bootstrap 类基础之上，扩展加入了应用类的 CDS(Application Class-Data Sharing) 支持。

其原理为：在启动时记录加载类的过程，写入到文本文件中，再次启动时直接读取此启动文本并加载。设想如果应用环境没有大的变化，启动速度就会得到提升。

我们可以想像为类似于操作系统的休眠过程，合上电脑时把当前应用环境写入磁盘，再次使用时就可以快速恢复环境。

我在自己 PC 电脑上做以下应用启动实验。

首先部署 wildfly 12 应用服务器，采用 JDK10 预览版作为 Java 环境。另外需要用到一个工具 cl4cds<sup>[1]</sup>，作用是把加载类的日志记录，转换为 AppCDS 可以识别的格式。

A 安装好 wildfly 并部署一个应用，具有 Angularjs, rest, jpa 完整应用技术栈，预热后启动三次，并记录完成部署时间

分别为 6716ms, 6702ms, 6613ms，平均时间为 6677ms。

B 加入环境变量并启动，导出启动类日志

```
export PREPEND_JAVA_OPTS="-Xlog:class+load=debug:file=/tmp/wildfly.
classtrace"
```

C 使用 cl4cds 工具，生成 AppCDS 可以识别的 cls 格式

```
/jdk-10/bin/java -cp src/classes/ io.simonis.cl4cds /tmp/wildfly.
classtrace /tmp/wildfly.cls
```

打开文件可以看到内容为：

```

java/lang/Object id: 0x0000000100000eb0
java/io/Serializable id: 0x0000000100001090
java/lang/Comparable id: 0x0000000100001268
java/lang/CharSequence id: 0x0000000100001440
.....
org/hibernate/type/AssociationType id: 0x0000000100c61208 super:
0x0000000100000eb0 interfaces: 0x0000000100a00d10 source: /home/
shihang/work/jboss/wildfly/dist/target/wildfly-12.0.0.Final/modules/
system/layers/base/org/hibernate/main/hibernate-core-5.1.10.Final.
jar
org/hibernate/type/AbstractType id: 0x0000000100c613e0 super:
0x0000000100000eb0 interfaces: 0x0000000100a00d10 source: /home/
shihang/work/jboss/wildfly/dist/target/wildfly-12.0.0.Final/modules/
system/layers/base/org/hibernate/main/hibernate-core-5.1.10.Final.
jar
org/hibernate/type/AnyType id: 0x0000000100c61820 super:
0x0000000100c613e0 interfaces: 0x0000000100c61030 0x0000000100c61208
source: /home/shihang/work/jboss/wildfly/dist/target/wildfly-
12.0.0.Final/modules/system/layers/base/org/hibernate/main/
hibernate-core-5.1.10.Final.jar
....

```

这个文件用于标记类的加载信息。

D 使用环境变量启动 wildfly，模拟启动过程并导出 jsa 文件，就是记录了启动时类的信息。

```

export PREPEND_JAVA_OPTS="-Xshare:dump -XX:+UseAppCDS
-XX:SharedClassListFile=/tmp/wildfly.cls
-XX:+UnlockDiagnosticVMOptions -XX:SharedArchiveFile=/tmp/wildfly.
jsa"

```

查看产生的文件信息，jsa 文件有较大的体积。

```

/opt/work/cl4cds$ ls -l /tmp/wildfly.*
-rw-rw-r-- 1 shihang shihang 8413843 Mar 20 11:07 /tmp/wildfly.
classtrace
-rw-rw-r-- 1 shihang shihang 4132654 Mar 20 11:11 /tmp/wildfly.cls

```

```
-r--r--r-- 1 shihang shihang 177659904 Mar 20 11:13 /tmp/wildfly.jsa
```

E 使用 jsa 文件启动应用服务器

```
export PREPEND_JAVA_OPTS="-Xshare:on -XX:+UseAppCDS  
-XX:+UnlockDiagnosticVMOptions -XX:SharedArchiveFile=/tmp/wildfly.  
jsa"
```

启动完毕后记录时长，三次分别是 5535ms, 5333ms, 5225ms，平均为 5364ms，相比之前的 6677ms 可以算出启动时间提升了 20% 左右。

这个效率提升，对于云端应用部署很有价值。

以上实验方法参考于技术博客<sup>[2]</sup>。

#### 4. JEP314，使用附加的 Unicode 语言标记扩展

JDK10 对于 Unicode BCP 47 有了更多的支持，BCP 47 是 IETF 定义语言集的规范文档。使用扩展标记，可以更方便的获得所需要的语言地域环境。

如 JDK10 加入的一个方法，

```
java.time.format.DateTimeFormatter::localizedBy
```

通过这个方法，可以采用某种数字样式，区域定义或者时区来获得时间信息所需的语言地域本地环境信息。

附：从链接<sup>[3]</sup>可以看到 JDK10 所有的方法级别改动。

#### 5. 查看当前 JDK 管理根证书

自 JDK9 起在 keytool 中加入参数 -cacerts，可以查看当前 JDK 管理的根证书。而 OpenJDK9 中 cacerts 为空，这样就会给开发者带来很多不变。

EP318 就是利用 Oracle 开源出 Oracle JavaSE 中的 cacerts 信息，在 OpenJDK 中提供一组默认的根证书颁发机构证书，目前有 80 条记录。

```
/jdk-10/bin$ ./keytool -list -cacerts  
Enter keystore password:  Keystore type: JKS  
Keystore provider: SUN  
Your keystore contains 80 entries  
verisignclass2g2ca [jdk], Dec 2, 2017, trustedCertEntry,  
Certificate fingerprint (SHA-256): 3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21  
:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A1  
.....
```

## 下一版本展望

下一个 Java 大版本会是 Java 11，也是 Java 8 之后的 LTS 版本，预计会在今年的 9 月份发布。目前只有四个 JEP，更多加强提议会逐步加入。

这个版本会充分发挥模块化的能力，把当前 JDK 中的关于 JavaEE 和 Corba 的部分移除，变得更加紧凑。

虽然 JDK9 最大的亮点是模块化，但 Java 业界广泛接纳并且适应需要一个过程。当前已经有一些支持模块化的类库，如 log4j2，但大多数还未支持。

可以预见 JDK11 发布之后，模块化特性就成为长期支持特性，会有越来越多的类库提供对模块化的支持。

Java 依然会是最适合应用开发的语言和平台，庞大的社区和广泛的开发者，会不断促使 Java 不断完善优化，在各个编程领域继续发扬光大。

对文中引用文章原作者表示致谢！引用的图示，数据和方法都属于原作者。下一个 Java 大版本会是 Java 11，也是 Java 8 之后的 LTS 版本，预计会在今年的 9 月份发布。目前只有四个 JEP，更多加强提议会逐步加入。这个版本会充分发挥模块化的能力，把当前 JDK 中的关于 JavaEE 和 Corba 的部分移除，变得更加紧凑。虽然 JDK9 最大的亮点是模块化，但 Java 业界广泛接纳并且适应需要一个过程。当前已经有一些支持模块化的类库，如 log4j2，但大多数还未支持。可以预见 JDK11 发布之后，模块化特性就成为长期支持特性，会有越来越多的类库提供对模块化的支持。Java 依然会是最适合应用开发的语言和平台，庞大的社区和广泛的开发者，会不断促使 Java 不断完善优化，在各个编程领域继续发扬光大。对文中引用文章原作者表示致谢！引用的图示，数据和方法都属于原作者。

## 附录

[1] <https://simonis.github.io/cl4cds/>

[2] <https://marschall.github.io/2018/02/18/wildfly-appcds.html>

[3] <https://gunnarmorling.github.io/jdk-api-diff/jdk9-jdk10-api-diff.html#java.time.format.DateTimeFormatter>

## 作者介绍

张建锋，永源中间件共同创始人，原红帽公司 JBoss 应用服务器核心开发组成员。毕业于北京邮电大学和清华大学，曾供职于金山软件，IONA 科技公司和红帽软件。

对于 JavaEE 的各项规范比较熟悉；开源技术爱好者，喜欢接触各类开源项目，学习优秀之处并加以借鉴，认为阅读好的源码就和阅读一本好书一样让人感到愉悦；在分布式计算，企业应用设计，移动行业应用，DevOps 等技术领域有丰富的实战经验和自己的见解；愿意思考软件背后蕴涵的管理思想，认为软件技术是一种高效管理的实现方式，有志于将管理学和软件开发进行结合。

# Kubernetes 效应

作者 Bilgin Ibryam 译者 罗远航



Kubernetes (k8s) 在很短的一段时间内走过了很长的一段路。仅仅两年以前，它还需要与 CoreOS 的 Fleet、Docker Swarm、Cloud Foundry Diego、HashiCorp 的 Nomad、Kontena、Rancher 的 Cattle、Apache Mesos、Amazon ECS 等进行竞争，来证明自己比那些产品都要优秀。而现如今已经是完全不同的一幅景象了。其中的一些公司公开宣布了项目的终止并且开始加入到 Kubernetes 阵营中，还有一些公司没有公开宣布自己项目的失败，而是在战略上宣布了对 Kubernetes 的部分支持或者完全整合，这也就意味着他们的容器编排工具将会安静而缓慢地死掉。不论是哪一种情况，k8s 都是最后一个活下来的平台。除此之外，不仅仅是

用户和白金赞助商们，越来越多的大公司都将继续加入到 Kubernetes 的生态系统中，将自己的业务完全押注于 Kubernetes 的成功。我们首先能想到的有 Google 的 Kubernetes Engine、Red Hat 的 OpenShift、Microsoft 的 Azure Container Service、IBM 的 Cloud Container Service、Oracle 的 Container Engine。

但是这些意味着什么呢？首先，这意味着开发人员必须要掌握一个与 90% 的容器工作相关的容器编排平台。这是一个学习 Kubernetes 很好的理由。同时这还意味着我们已经深深地依赖于 Kubernetes，Kubernetes 就像容器领域中的 Amazon。在 Kubernetes 上进行设计、实现和运行应用程序可以让你在不同的云提供商、Kubernetes 发行版和服务提供商之间自由地对应用程序进行移动。它能让你有机会找到 [Kubernetes 认证](#) 的开发人员，让他们来开发一个项目并且在以后的运行过程中持续提供支持。Kubernetes 不是 VM，也不是 JVM，它是全新的应用程序移植层，它是大家共同的选择。

## 对 Kubernetes 的深度依赖

下图是一个容器化服务的图表，其中表明了该服务对 Kubernetes 的依赖程度：

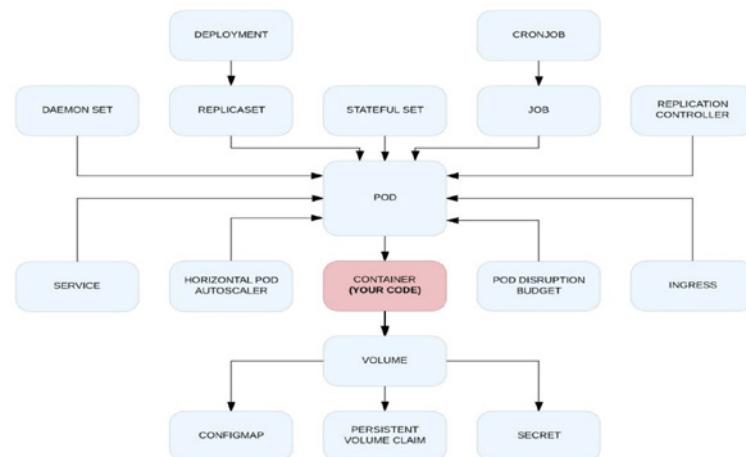


图 1 Kubernetes 上的应用程序依赖

请注意，我并没有写到 Kubernetes 是应用程序可移植性 API，它是一个层。上图向我们展示了只有显式创建的 k8s 对象才能够调用 k8s API。但事实上，我们与平台之间的关系更加紧密。k8s 提供了一组完整的分布式原语（如 Pod、服务、控制器等），它们能够满足我们的需求并且驱动我们对应用程序的设计。这些新的原语和平台容量决定了我们用来实现未来所有服务的指导设计原则和设计模式。反过来，它们也会影响我们用来应对日常挑战的技术，甚至会影响到我们所谓的“最佳实践”。因此，我们更倾向于将 Kubernetes 视为一个基本的范式，它在多个维度上都有意义，而不仅仅是一个能够与之交互的 API。

## Kubernetes 效应



图 2 软件开发生命周期中的 Kubernetes 效应

容器和容器编排工具特性提供了一组新的抽象和原语。为了获得这些原语的最佳价值并衡量它们的效果，我们需要一套新的设计原则来指导我们的实践。带来的结果就是，随着我们更多的使用这些新原语，我们就能更好地解决重复性的问题，能够重新发明轮子。这就是设计模式的用武之地。设计模式为我们提供了如何构建新原语以便更快地解决问题的方法。虽然原则更加抽象、更加基础、变化也更少，但是设计模式可能会受到原语行为变化的影响。平台上的一个新特性可能会使得一个设计模式成为反模式或者不那么相关。除此之外，还有我们每天使用的实践和技术也是如此。这些技术涵盖了从非常细微的技术技巧，到更有效地执行任务，再到更广泛的工作方法和实践方法。当我们发现了一种稍微好用一点的方法能够更快或者更简单地完成任务时，我们就会改进当前的技术和时间。这就是我们和平台互相促进的发展模式。我们为什么要这么做呢？因为，我们

这样做就能够利用这个新平台的优势和价值来满足我们自己的需求。从某种意义上来说。Kubernetes 效应是自我强化的，也是多方面的。

在本文的剩余部分，我们将探寻地更深，并且将对每个类别的示例进行研究。

## 分布式抽象和原语

为了能够阐释我所说的新的抽象和原语，我将把它们与大家众所周知的面向对象世界和 Java 进行比较。在面向对象编程（OOP）的世界里，我们有如下的概念：类、对象、包、继承、封装、多态等等。Java 运行时环境提供了一些特性并且它对对象的生命周期和整个应用程序进行管理。Java 语言和 JVM 运行时环境为应用程序的创建提供了本地的、进程内的构建块。Kubernetes 为这种众所周知的思维模式又添加了一个新的维度，它提供了一组新的分布式原语和运行时环境，用以创建分布于多个进程之间和不同节点上的分布式系统。有了 Kubernetes，我就不用依赖于 Java 原语来实现整个应用程序的行为了。我仍然需要使用面向对象的构建块来创建分布式应用程序的组件，但是我也可以使用 Kubernetes 原语来实现一些应用程序的行为。一些 [Kubernetes 分布式抽象和原语](#)的例子如下：

- Pod：相关容器集合的部署单元
- Service：服务发现和负载均衡原语
- Job：异步调度工作的原子单元
- CronJob：在未来进行调度或者定期进行调度的工作原子单元
- ConfigMap：一种跨服务实例的分发配置数据的机制
- Secret：一种管理敏感配置数据的机制
- Deployment：一种声明式应用程序发布机制
- Namespace：用于隔离资源池的控制单元

例如，我可以依赖于 Kubernetes 的健康监测（如预备检测或活性检测）来获取我的应用程序的一些可靠性信息。我可以使用 Kubernetes Service 来进行服务发现，而不是在应用程序中进行客户端服务的发现。我能使用

Kubernetes Jobs 来对异步原子工作进行调度。我还能使用 ConfigMap 来对配置进行管理。我还能使用 Kubernetes CronJob 对任务进行周期性调度，这样就不用使用基于 Java 的 Quartz 库或者去实现 ExecutorService 接口了。

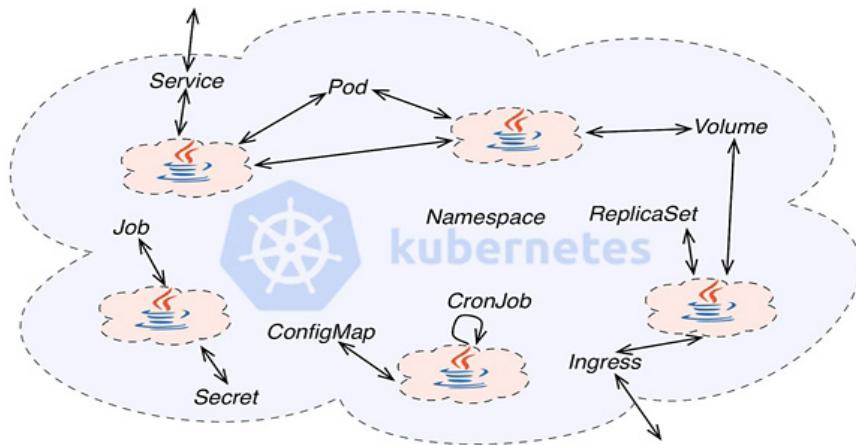


图 3 分布式系统中本地原语和分布式原语部分

进程原语和分布式原语有着相同之处，但是它们之间没有直接的可比性和可替换性。它们运行于不同的抽象级别，具有不同的先决条件和保证条件。一些原语可以在一起使用。例如，我们仍然需要使用类来创建对象并且将其放入容器镜像中。但是 Kubernetes 中一些像 CronJob 这样的原语就能对 Java 中的 ExecutorService 行为进行完全的替代。下边有一些概念的图表，是我在 JVM 和 Kubernetes 之间找到的共同点，但是并不是很深入。

Concern	Java & JVM	Kubernetes
Behaviour encapsulation	Class	Container Image
Behaviour instance	Object	Container
Unit of reuse	.jar	Container Image
Deployment unit	.jar/.war/.ear	Pod
Buildtime/Runtime isolation	Module, Package, Class	Container Image, Namespace
Initialization preconditions	Constructor	Init-container
Post initialization	init-method	PostStart
Pre destroy	destroy-method	PreStop
Cleanup procedure	finalize(), ShutdownHook	Defer-container*
Asynchronous & Parallel execution	ThreadPoolExecutor, ForkJoinPool	Job
Periodic task	Timer, ScheduledExecutorService	CronJob
Background task	Daemon Thread	DaemonSet
Configuration management	System.getenv(), Properties	ConfigMap, Secret

图 4 本地原语和分布式原语分类

我之前发布过关于分布式抽象和原语的[博文](#)。这里的要点是，作为一名开发人员，你可以使用一组更丰富的本地和全局原语来设计和实现分布式解决方案。随着时间的推移，这些新的原语产生了新的解决问题的方案，其中一些复杂的解决方案变成了设计模式。这就是我们将进一步进行探讨的问题。

## 容器设计原则

设计原则是编写高质量软件的基本规则和抽象准则。这些原则没有指定具体的规则，但是它们代表了许多开发人员所理解并且经常引用的一种语言和通用的智慧。和 Robert C. Martin 所引入的 [SOLID 原则](#)（该原则提供了如何能够更好地编写面向对象软件的指导）类似，也有一些设计原则能够用于创建更好的容器化应用程序。SOLID 原则使用了面向对象的原语和概念（例如，类、接口以及继承等等）来阐释面向对象设计。类似地，下面列出的这些创建容器化应用程序的原则使用了容器镜像来作为基础原语，使用容器编排工具平台作为目标容器的运行时环境。基于容器的应用程序设计原则如下。

### 构建时

- 单一关注原则（Single Concern Principle）：每一个容器都应该解决一个问题，并且把这一个问题解决好。
- 自我控制原则（Self-Containment Principle）：容器应该只依赖于 Linux 内核，在容器构建时才添加其它外部的库。
- 镜像不变性原则（Image Immutability Principle）：容器化的应用程序是不可变的，一旦构建完成，在不同的环境下也是不会产生变化的。

### 运行时

- 高可观测性原则（High Observability Principle）：每个容器都必须要实现所有必要的 API，用于帮助平台以最好的方式对应用程序进

行监测和管理。

- 生命周期一致性原则 (Lifecycle Conformance Principle)：容器应该有一种方法来读取来自平台的事件，并通过对这些事件的响应来保持一致。
- 过程一次性原则 (Process Disposable Principle)：应用程序容器化的过程应该尽可能的短暂，它们需要时刻准备着被另一个容器实例所替换。
- 运行时限制原则 (Runtime Confinement Principle)：每个容器都应该对其资源需求进行声明，并且保证应用程序受限制于指定的资源需求也是很重要的。

遵循这些原则能够帮助我们创建更适合于 Kubernetes 等云平台的容器化应用程序。

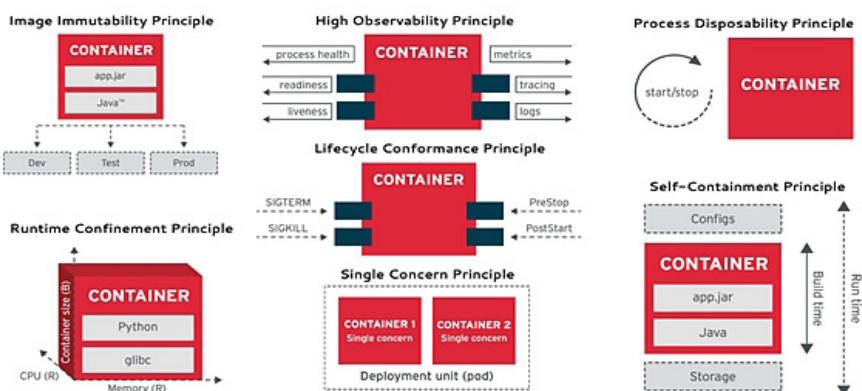


图 5 容器设计原则

这些原则都已经编纂成文，可以点击[这里](#)免费下载这份白皮书。上述图片就是这些原则的预览。

## 容器设计模式

新的原语需要新的原则来解释原语之间的作用。我们使用的原语越多，就能解决更复杂的问题，这就使得我们意识到可重用的解决方案，即设计模式。容器设计模式的重点是用能够解决当前面临挑战的最佳方式来构建

容器和其它分布式原语。以下简短的列表是与容器相关的设计模式：

- 跨斗模式（Sidecar Pattern）：跨斗模式的容器能够拓展和增强先前存在的容器的功能而不对其进行改变。
- 大使模式（Ambassador Pattern）：这种模式能够隐藏复杂性，并为容器提供一个统一的外部接口。
- 适配器模式（Adapter Pattern）：适配器模式是大使模式的反模式，它为来自外部世界的pod提供了一个统一的接口。
- 初始化器模式（Initializer Pattern）：初始化器模式的容器能够将初始化相关的任务与主应用程序逻辑相分离。
- 工作队列模式（Work Queue Pattern）：采用工作队列模式的容器能够将任意处理代码打包成容器或数据，并且构建成一个完整的工作队列系统。
- 自定义控制器模式（Custom Controller Pattern）：通过这种模式，控制器就能观察对象的变化，并对这些变化采取活动，从而将集群驱动至所需的状态。这种调解模式可用于实现自定义逻辑并拓展平台的功能。
- 自感知模式（Self Awareness Pattern）：这种模式描述了应用程序需要进行自身审查并且获得关于自身和运行环境元数据的情况。

这一领域的基础工作是由 [Brendan Burns](#) 和 Oppenheimer 在他们的容器设计模式的[论文](#)中完成的。之后，Brendan 出版了一本关于分布式系统设计模式[相关主题的书](#)。[Roland Huß](#) 和我也写了一本书叫做《[Kubernetes Patterns](#)》，里面涵盖了所有这些设计模式和基于容器的应用程序的使用例。下图是其中一些模式的可视化。

原语需要原则和精心设计的模式。接下来让我们看看使用 Kubernetes 的最佳实践和一些好处。

## 实践和技巧

除了原则和模式之外，创建良好的容器化应用程序还需要熟悉其他与

容器相关的最佳实践和技巧。原则和模式是抽象的、基础的观点，它们的变化很少。最佳实践和相关的技巧则更加具体，可能会频繁地发生变化。以下是一些常见的与容器相关的最佳实践：

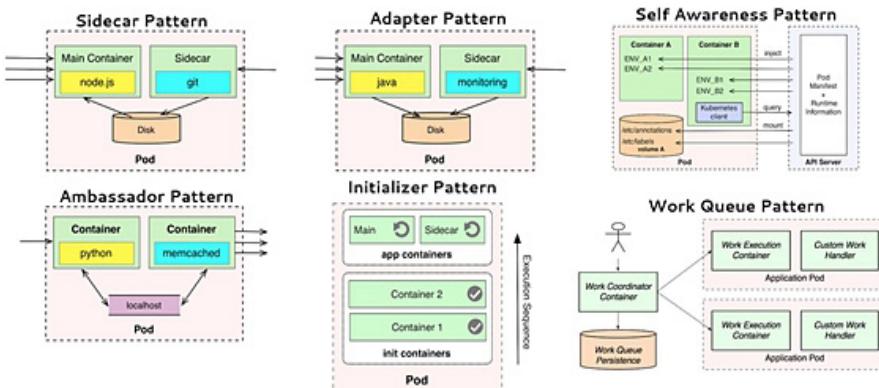


图 6 容器设计模式

- 专注于小镜像：这能减小容器的大小、构建时间和复制容器镜像时的网络时间。
- 支持任意用户id：避免使用sudo命令或需要一个特定的userid来运行您的容器。
- 标记重要端口：使用EXPOSE命令指定端口，这能使得其他人和软件能够更简单地使用你的镜像。
- 为持久数据使用卷（Volume）：在容器被销毁后，需要被保存的数据必须被写入到卷中。
- 设置映像元数据：镜像元数据以标记、标签和注释的形式出现，这能使您能够容易发现自己的容器镜像。
- 同步主机和镜像：一些容器化的应用程序要求容器在某些属性上与主机进行同步，例如时间和机器ID。
- STDOUT和STDERR日志：将日志记录到这些系统流中而不是文件里能够确保日志被正确地收集和聚合。

以下的这些链接是容器相关最佳实践的一些资源：

- [编写Dockerfiles的最佳实践](#)

- [OpenShift指导原则](#)
- [Kubernetes产品模式](#)
- [Kubernetes示例](#)

## Kubernetes优势

综上所述，Kubernetes 以一种基本的方式来指导分布式系统的设计、开发和操作。Kubernetes 的学习路线并不短，跨越 Kubernetes 技术鸿沟需要时间和耐心。这就是为什么我想在这里和读者谈谈 Kubernetes 给开发者带来的一系列好处。我希望这将有助于证明为什么学习 Kubernetes 是值得的，并且为什么要使用它来指导你的 IT 战略。

- 自服务环境：它使得团队和团队成员能够立即从集群中分离出隔离的环境，以便进行CICD和用于实验目的。
- 动态放置应用程序：它允许基于应用程序需求、可用资源和指导策略，将应用程序以可预测的方式放置到集群中。
- 声明式服务部署：这个抽象封装了一组容器的升级和回滚过程，并能够将其执行为可重复和可自动化的活动。

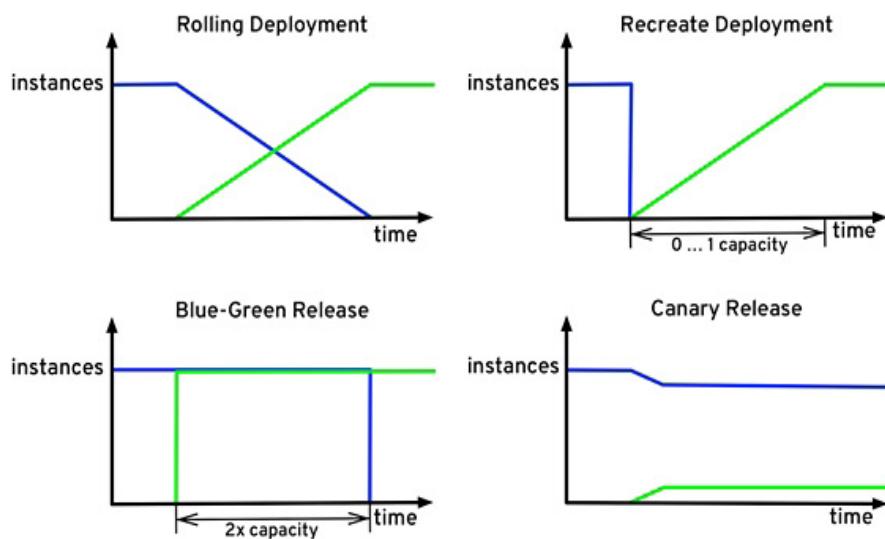


图 7 Kubernetes 部署和发布示例

- 应用程序弹性：容器和管理平台可以通过多种方式提高应用程序的弹性，例如：
  - 无限循环：CPU共享和配额
  - 内存泄露：自己内存不足
  - 磁盘占用：配额
  - Fork炸弹：进程限制
  - 断路器、超时、以跨斗模式重启
  - 以跨斗模式进行故障转移和服务发现
  - 使用容器进行线程隔离
  - 通过调度器进行硬件隔离
  - 容量自动伸缩和自我恢复
- 服务发现、负载均衡和断路器：平台允许服务在不使用应用程序代理的情况下发现和使用其他服务。此外，跨斗模式的容器和 Istio 框架等工具的使用，可以完全将应用程序之外的相关网络职责转移到平台级别之外。
- 声明性应用程序拓扑：使用 Kubernetes API 对象允许我们对如何部署我们的服务进行描述以及说明它们对其他服务和资源的依赖。将所有这些信息以可执行的格式提供给我们，使我们能够在开发的早期阶段对应用程序的部署进行测试，并将其作为可编程的应用程序基础架构。

关于 Kubernetes，还有更多我们为之感到兴奋的理由。我上边列出的是我感觉很有用的东西，因为它们来源于有开发背景的人们。

## 资源

我希望能描述我是如何看待 Kubernetes 对开发人员日常生活的影响的。如果你想了解更多关于这方面的信息，换句话说想从开发者的视角来了解 Kubernetes 的话，你可以看一下我的书并且关注下我的 Twitter 账号 [@bibryam](#)。

下面的这些链接是从开发人员的角度看待 Kubernetes 的一些资源：

- [基于容器的分布式系统设计模式](#)（白皮书）
- [基于容器的应用程序设计原则](#)（白皮书）
- [分布式系统设计](#)（电子书）
- [Kubernetes模式](#)（电子书）
- [Kubernetes实战](#)（电子书）

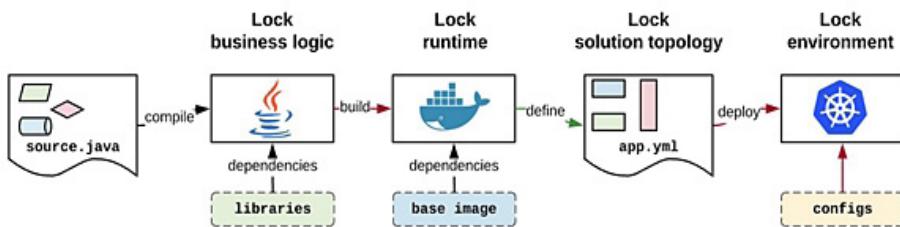


图 8 声明性应用程序拓扑对 Kubernetes 资源描述符文件的使用

## 关于作者

Bilgin Ibryam (@bibryam) 是 Red Hat 的首席架构师，也是 ASF 的成员和委员会成员。他是一位开源的布道者、博客作者，是《Camel Design Patterns》和《Kubernetes Patterns》的作者。在他的日常工作中，Bilgin 喜欢指导、编码和领导开发人员成功地构建云解决方案。他目前的工作重点是应用程序集成、分布式系统、消息传递、微服务、DevOps 和云原生的挑战。

# 360 开源项目大盘点

作者 徐川



自从 360 于 2016 年在美股退市以来，大家就一直猜测它何时会在国内上市，期间还发生了诸如借壳上市等话题，如今这只悬在空中的靴子终于落地了。

对于很多人来说，360 最为人所知的是它在网络安全方面的建树，不时国外得个奖，近年来席卷互联网的安全事件也都能及时给出警告和解决方案，受到广大网友信赖。

不过，除了安全，其实 360 还有很多开源技术值得称道，下面就让我们来盘点一下。

奇虎360重要开源项目盘点		
领域	项目	介绍
云计算	jepsen-io/jepsen	分布式系统验证框架
	Qihoo360/huststore	高性能分布式存储服务
	Qihoo360/poseidon	日志搜索引擎
人工智能	Qihoo360/Xlearning	深度学习调度平台
数据库	Qihoo360/zeppelin	高性能KV存储平台
	Qihoo360/Atlas	高性能MySQL代理
	Qihoo360/pika	类Redis存储系统
	Qihoo360/mysql-sniffer	基于MySQL协议的抓包工具
前端/Node	Chimeejs/chimee	组件化H5播放器框架
	thinkjs/thinkjs	企业级Node框架
	firekylin/firekylin	通用博客系统
移动开发	Qihoo360/RePlugin	Android占坑类插件化框架
	DroidPluginTeam/DroidPlugin	Android插件化框架
运维	Qihoo360/Qconf	分布式配置管理工具
	Qihoo360/phptrace	PHP执行跟踪工具
	360EntSecGroup-Skylar/ElasticHD	ElasticSearch可视化管理工具
开发工具	360EntSecGroup-Skylar/goreporter	代码质量检测工具
	360EntSecGroup-Skylar/excelize	Go语言Excel文档操作库

## 云计算

jepsen：分布式系统验证框架

<https://github.com/jepsen-io/jepsen>

Jepsen 是开源社区比较公认的分布式数据库的测试框架。Jepsen 验证过程包括 VoltDB、CockroachDB、Galera、MongoDB、etcd 在内的几乎所有的主流分布式数据库 / 系统。

### **huststore: 高性能分布式存储服务**

<https://github.com/Qihoo360/huststore>

huststore 是一个高性能的分布式存储服务，不但提供了 100 thousand QPS 级别的 kv 存储的功能，还提供了 hash、set、sort set 等一系列数据结构的支持，并且支持二进制的 kv 存储，可以替代 Redis 相关的功能。

### **Poseidon: 日志搜索引擎**

<https://github.com/Qihoo360/poseidon>

Poseidon 系统是一个日志搜索平台，能在数百万亿条规模的数据集中找出我们需要的数据，只需要花费几秒钟时间，大大提高工作效率；同时，数据不需要额外存储，节省了大量存储和计算资源。该系统可以应用于任何海量（从万亿到千万亿规模）的查询检索需求。

## **人工智能**

### **XLearning: 深度学习调度平台**

<https://github.com/Qihoo360/XLearning>

XLearning 由 360 系统部大数据团队与人工智能研究院联合开发，基于 Hadoop Yarn 完成了对 TensorFlow、MXNet、Caffe、Theano、PyTorch、Keras、XGBoost 等常用深度学习框架的集成。平台上线运行近一年时间，经多次版本迭代更新，为各学习框架的使用者提供了统一、稳定的作业提交平台，实现了资源共享，极大的提高了资源利用率，并且具有良好的扩展性和兼容性，在公司搜索、人工智能研究院、商业化、数据中心等业务部门得到广泛使用。

## **数据库**

### **Zeppelin: 高性能 KV 存储平台**

<https://github.com/Qihoo360/zeppelin>

Zeppelin 是奇虎 360 开源的一个高性能，高可用的分布式 Key-Value 存储平台，它以高性能、大集群为目标，并希望能在 Zeppelin 的基础上，不仅能够提供 KV 的访问，还可以通过简单的一层转换满足更复杂的协议需求。

**Atlas:** 高性能 MySQL 代理

<https://github.com/Qihoo360/Atlas>

Atlas 是由 Qihoo 360 公司 Web 平台部基础架构队开发维护的一个基于 MySQL 协议的数据中间层项目。它在 MySQL 官方推出的 MySQL-Proxy 0.8.2 版本的基础上，修改了大量 bug，添加了很多功能特性。

**pika:** 类 Redis 存储系统

<https://github.com/Qihoo360/pika>

Pika 是 360 DBA 和基础架构组联合开发的类 Redis 存储系统，完全支持 Redis 协议，用户不需要修改任何代码，就可以将服务迁移至 Pika。有维护 Redis 经验的 DBA 维护 Pika 不需要学习成本。

Pika 主要解决的是用户使用 Redis 的内存大小超过 50G、80G 等等这样的情况，会遇到启动恢复时间长，一主多从代价大，硬件成本贵，缓冲区容易写满等问题。Pika 就是针对这些场景的一个解决方案。

**MySQL Sniffer:** [MySQL 抓包工具](#)

<https://github.com/Qihoo360/mysql-sniffer>

MySQL Sniffer 是一个基于 MySQL 协议的抓包工具，实时抓取 MySQLServer 端的请求，并格式化输出。输出内容包括访问时间、访问用户、来源 IP、访问 Database、命令耗时、返回数据行数、执行语句等。有批量抓取多个端口，后台运行，日志分割等多种使用方式，操作便捷，输出友好。

**前端 /Node**

**chimee:** 浏览器视频播放框架

<https://github.com/Chimeejs/chimee>

Chimee 由奇舞团研制的 h5 播放器，它支持 mp4、m3u8、flv 等多种格式。通过插件式开发，能满足业务方快速迭代、灰度发布等要求。让开发者能够轻松快捷地完成视频场景的开发。

**ThinkJS：企业级 Node 框架**

<https://github.com/thinkjs/thinkjs>

ThinkJS 是一款面向未来开发的 Node.js 框架，整合了大量的项目最佳实践，让企业级开发变得如此简单、高效。从 3.0 开始，框架底层基于 Koa 2.x 实现，兼容 Koa 的所有功能。

**Firekylin：通用博客系统**

<https://github.com/firekylin/firekylin>

Firekylin 是一个高效简洁的动态博客系统，整体基于 ThinkJS 框架，后台采用了 React 技术栈。

## 移动开发

**RePlugin：Android 占坑类插件化方案**

<https://github.com/Qihoo360/RePlugin>

RePlugin 是一套完整的、稳定的、适合全面使用的，占坑类插件化方案，由 360 手机卫士的 RePlugin Team 研发，也是业内首个提出“全面插件化”（全面特性、全面兼容、全面使用）的方案。

**DroidPlugin：Android 插件化框架**

<https://github.com/DroidPluginTeam/DroidPlugin>

DroidPlugin 是在 Android 系统上实现了一种新的插件机制：它可以在无需安装、修改的情况下运行 APK 文件，此机制对改进大型 APP 的架构，实现多团队协作开发具有一定的好处。

## 运维监控

**QConf：分布式配置管理工具**

<https://github.com/Qihoo360/QConf>

QConf 是一个分布式配置管理工具。用来替代传统的配置文件，使得配置信息和程序代码分离，同时配置变化能够实时同步到客户端，而且保证用户高效读取配置，这使的工程师从琐碎的配置修改、代码提交、配置上线流程中解放出来，极大地简化了配置管理工作。

**phptrace**: PHP 执行跟踪工具

<https://github.com/Qihoo360/phptrace>

phptrace 是一个低开销的用于跟踪、分析 PHP 运行情况的工具。它可以跟踪 PHP 在运行时的函数调用、请求信息、执行流程，并且提供有过滤器、统计信息、当前状态等实用功能。在任何环境下，它都能很好的定位阻塞问题以及在高负载下 Debug，尤其是线上生产环境。

**ElasticHD**: ElasticSearch 可视化管理工具

<https://github.com/360EntSecGroup-Skylar/ElasticHD>

Elasticsearch 可视化 DashBoard，支持 Es 监控、实时搜索，Index template 快捷替换修改，索引列表信息查看，SQL converts to DSL 等。

## 开发工具

**GoReporter**: 代码质量检测工具

<https://github.com/360EntSecGroup-Skylar/goreporter>

Golang 开发工具，提供代码质量检测 /Golang 代码静态检测器 / Golang 项目单元测试，根据自定义模版自动生成 Golang 代码质量检测报告。

**Excelize**: Go 语言 Excel 文档操作库

<https://github.com/360EntSecGroup-Skylar/excelize>

这个项目是 Golang 编写的一个用来操作 Office Excel 文档的类库，基于 ECMA-376 Office OpenXML 标准。可以使用它来读取、写入带有复杂样式的 XLSX 文件。目前是开源项目中唯一支持读写带有图片（表）、透视表等复杂样式文档的类库。

# VPC 的 4 种典型使用场景

作者 吕昭波



互联网上各种软件、应用、手机 App 充斥着人们的每一天，这些应用也越来越易用、却也在开发商变得复杂。开发人员使用到的组件也更多了。稍微大些的应用开发将会使用组件解耦、系统分层，以便降低紧耦合带来的各种不良后果。

本文将介绍 VPC 如何实现各个组件分层和隔离，并将以 VPC 子网是否连通互联网、云平台 VPC 之间连通、本地数据中心与云端 VPC 的连通等介绍 VPC 的 4 种典型应用方式和解决需求的实现方法。

## VPC 有什么作用

VPC 通过子网将资源进行逻辑隔离为用户提供隔离的网络环，可以灵

活的定义子网网段，并支持随时在现有 VPC 中追加新的定义网段，保证地址取之不尽，解决传统子网带来的节点数量的限制。并可以使用 VPN 等方式连接本地数据中心后将业务平滑迁移到云端。

## 场景 1：VPC 内连接互联网

VPC 内使用子网将资源进行了隔离，初始情况下子网内资源无法连接到互联网，所有资源和服务仅可内网访问，起到了预想的与公网隔离的效果。但如果资源只能内网访问显然也不是我们想要的，我们创建的 Web 应用等服务需要暴露在公网上，也就需要将 VPC 子网内的资源具有访问互联网的能力。

### 问题

在 VPC 子网内的资源需要能够连接到互联网。

### 解决方法

- 为 VPC 子网内每一个云主机资源绑定 EIP；
- 通过 NAT 网关将 VPC 子网内资源路由至 NAT 网关，并通过其绑定的 EIP 连接互联网。

### 具体实现

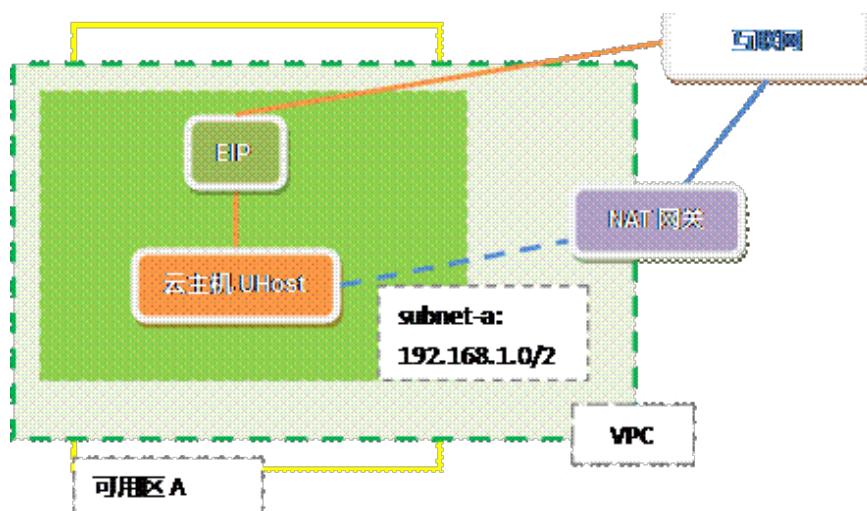


图 1 VPC 子网中云主机通过 EIP 或 NAT 网关连接互联网

1. 创建 VPC 及子网 subnet-a，并部署相关云主机等资源；
2. 选择使用 EIP 方式，申请 EIP 并绑定至云主机 UHost，则云主机 UHost 可以通过 EIP 与互联网进行连接；这种方式配置最简单，但是却要为每一台云主机 UHost 绑定 EIP，如果资源数量较多，不建议使用这种方式；
3. 选择使用 NAT 网关方式，在 VPC 中创建 NAT 网关并选择 subnet-a 连通，此时子网 subnet-a 中的所有资源将会路由至 NAT 网关，并通过绑定在 NAT 网关上的 EIP 连接互联网；这种方式通过一个配置即可满足一个子网内资源的连接互联网需求；
4. 云主机绑定 EIP 方式与 NAT 网关方式，两者只能取其一进行使用。



图 2 通过端口转发规则将流量转发至 NAT 网关绑定的 EIP 以连接互联网

## 应用方式

我们选择通过一台云主机 UHost 提供个人博客的 Web 服务，在创建云主机 UHost 时选择部署在 VPC1 的 192.168.1.0/24 子网内，申请一个 EIP 绑定到该云主机 UHost 上来提供外网访问能力，互联网上用户可以直接通过该 EIP 访问到该博客网站。

## 场景 2：通过 VPC 子网隔离内外网组件

一台云主机很难满足普通应用的需求，例如搭建具有更高可用性的企业博客，通常需要用到多台云主机、负载均衡、EIP，一些配置信息、博客数据还需要使用云数据库服务。用户可访问的 Web 服务需要连接公网，而用户数据、日志数据等只需在系统内部和 Web 服务器连接使用，因此需要根据是否对外网连接进行分层，部署在不同子网中。

## 需求

因业务需求将资源和组件划分为互联网可访问、互联网不可访问进行隔离。

## 解决方法

在 VPC 中创建子网 subnet-a（供连接互联网），创建 subnet-b（供内网使用），通过 NAT 网关只连接 subnet-a 并面向互联网开放；subnet-b 不连接到该 NAT 网关并且其中资源也不绑定 EIP。

## 具体实现

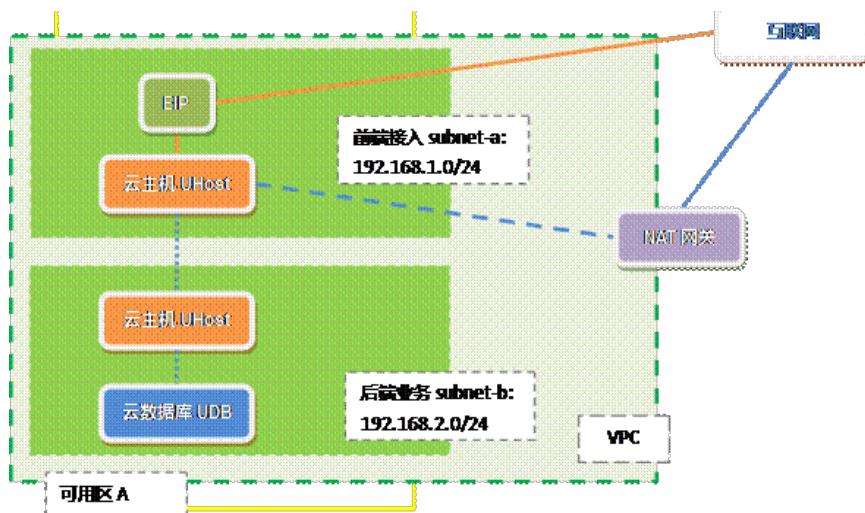


图 3 连接公网的子网和只供内网访问的子网

- 创建VPC;
- 创建两个子网subnet a和subnet b， subnet a为前端接入子网，部署云主机； subnet b为数据库子网部署云数据库；

- 配置NAT网关，并连接前端接入子网subnet-a，使其可以连接到互联网，对外提供前端接入功能；数据库子网subnet-b只能仅在VPC内访问，前端接入子网中的云主机UHost可以连接后端业务子网中的后端服务器和云数据库并实现业务支撑和数据操作。

通过如上配置，既实现了对互联网访问又保证了云数据库只供内网访问，为其安全增加了一道保障。

### 相关场景：后端业务子网临时连接互联网

如上所述，数据库私有子网中只能内网访问，但是仍会碰到云数据库进行版本更新、漏洞修复等需要访问互联网的情况。基于这种临时需求，我们可以通过NAT网关的白名单模式来连接互联网。在NAT网关配置中添加subnet-b，但是需要使用白名单模式，仅允许开放指定数据库的特定端口，尽可能避免全部暴露在互联网。



图4 使用NAT网关白名单并配置端口转发规则实现指定资源和端口对互联网开放

只需通过NAT网关连接需要外网访问能力的子网即可，通过白名单模式和端口转发规则配置可以实现细粒度的访问控制。

### 场景3：云平台多VPC之间互联

在构建业务时会按照生产环境、开发环境、测试环境等在不同的VPC中部署资源，偶尔需要打通不同环境的VPC。在VPC规划时可能因为对

可能的因素考虑不足，导致 VPC 设置过小，或者资源在 VPC 之间的分布不合理而又不便重新部署，这些可能都要将多个 VPC 进行连接。

## 需求

因生产环境与测试环境等分割环境、前期对 VPC 规划不足等原因而需要将云平台多个 VPC 进行连接。

## 解决方法

UCloud 云平台支持跨地域、跨项目的多个 VPC 连接，可在控制台直接操作配置。

## 具体实现

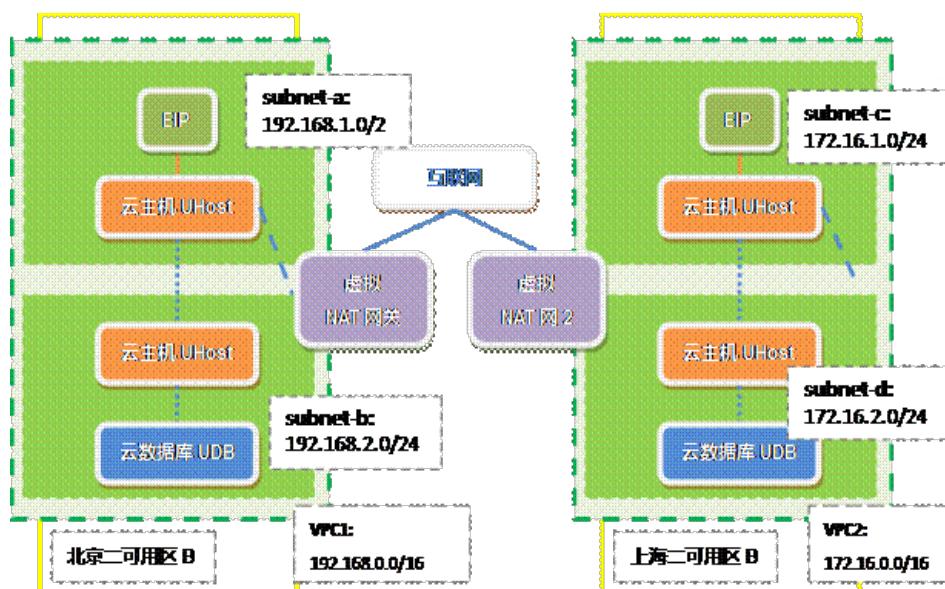


图 5 云平台中两个 VPC 互联

1. 在云平台 VPC 配置中直接选择多个 VPC，可直接实现 VPC 联通；
2. 在 VPC1 中所有流量会路由至 VPC 虚拟 NAT 网关，同样 VPC2 中所有流量会路由至虚拟 NAT 网关 2；
3. UCloud 云平台会自动将 VPC1 和 VPC2 进行连接，便可以实现两个 VPC 之间的流量传输。

此过程不再需要我们配置路由表，但为了方便了解数据流向，分析下后端实现的虚拟路由表。VPC1 中虚拟路由表为：

目的地	下一跳
192.168.0.0/16	Local
0.0.0.0	virtual-gw-1

VPC2 中虚拟路由表为：

目的地	下一跳
172.16.0.0/16	Local
0.0.0.0	virtual-gw-2

具体配置方式就比较简单了，如下图所示。



图 6 在云平台直接联通多个 VPC

## 场景 4：连接本地网络与云端 VPC

### 需求

用户业务部署在多个地域或者本地数据中心，需要将业务进行联通。

### 解决方法

- 使用VPN、专线接入UConnect连接本地数据中心VPC子网与UCloud云平台的VPC子网；

- 使用跨域通道UDPN连接UCloud云平台的多个VPC子网。

## 具体实现

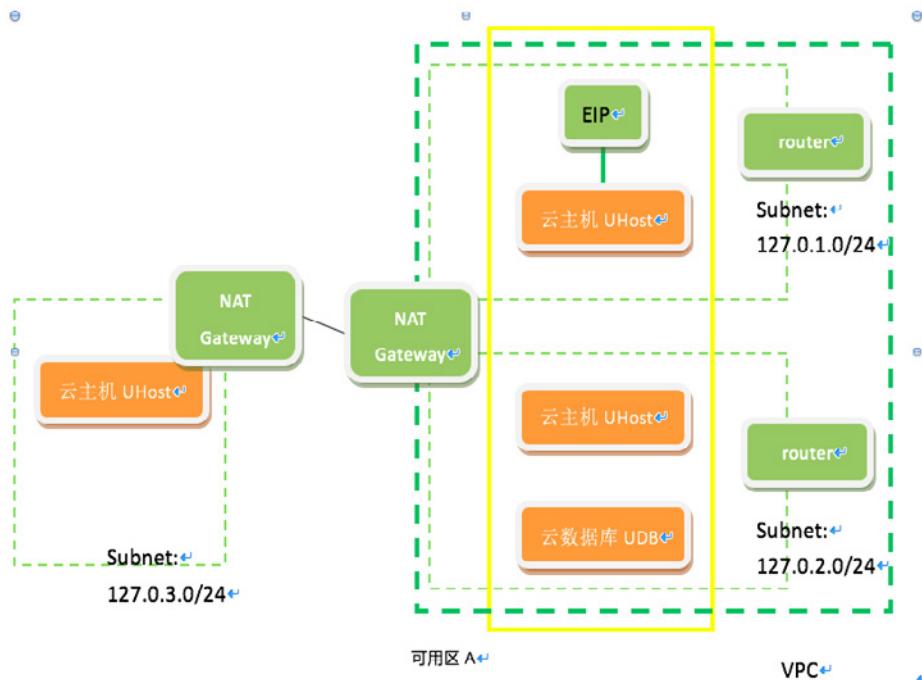


图 7 本地数据中心与云端 VPC 互联

- 在云端部署 VPC 公网子网、私网子网，并部署业务所需云平台资源；
- 在云端配置 IPSec VPN，其中配置云端网关地址、客户本地（对端）网关地址；
- 在客户本地安装 VPN 软件，并配置客户本地网关、云端网关；
- 在云端和客户本地 VPN 中配置 VPN 隧道 (tunnel) 并连通指定子网，测试流量正常。

## 相关场景

跨地域的 VPC 连通有多种类型，按照所属位置来说包括：

- 云平台间连通：多个VPC分布在UCloud云平台的多个地域中；
- 混合云架构（基础）：连通客户本地数据中心的VPC和UCloud云平台的VPC；
- 混合云架构：云平台上多个VPC，并且在本地数据中心也有VPC。

很多传统行业在本地数据中心部署有业务，在迁移上云的过程中可能将核心数据或业务保留在本地，产生了混合云的架构。连通本地数据中心和云端的方式也有几种方式：自建 VPN 方式、云端 IPSec VPN、专线接入 UConnect 等，具体差异在运维成本、通信效率等方面，VPC 子网连通配置上大同小异。

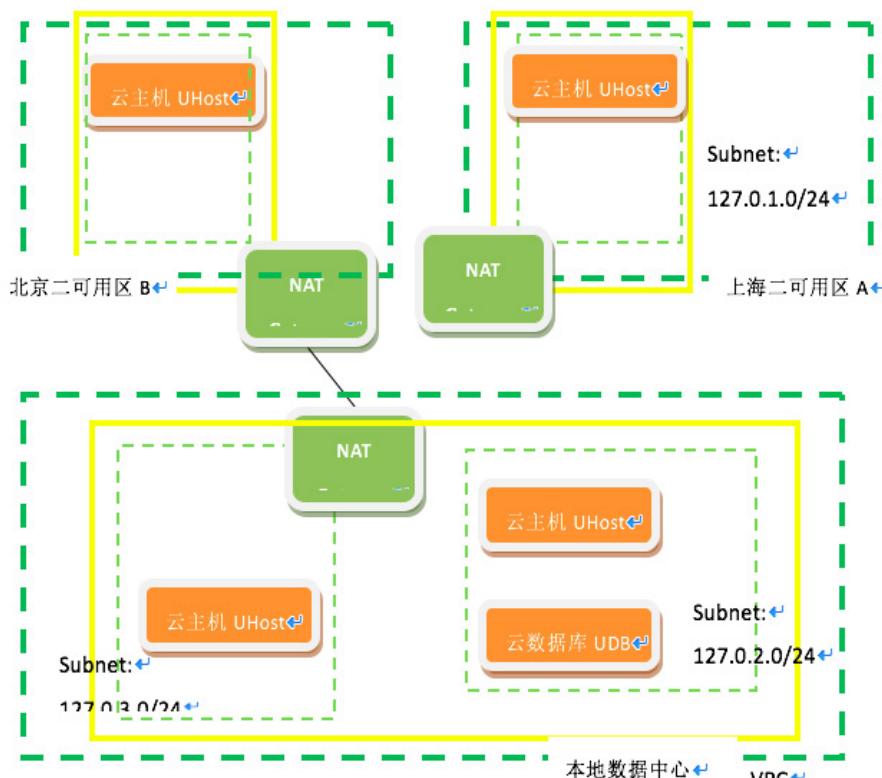


图 8 本地数据中心 VPC 以及多个云端 VPC 互联

云平台之间的连通可以选择跨域通道 UDPN，基于同城环网、专线等基础设施，实现了超远程专线网络。为云平台中跨地域的各个数据中心间，提供低延迟、高质量的跨地区内网数据传输的能力。

## 总结

VPC 无论是在传统服务器管理还是云端资源管理上都非常成熟，也是比较基础的服务。本文根据 VPC 子网是否需要连接到公网、云平台多



图 9 使用跨域通道 UDPN 便捷的连通多个地域

一个 VPC 互相连通以及本地网络与云端 VPC 的打通进行了概述和整理，回顾溯源主要是以上几种情况。现实情况下还要根据业务场景灵活的使用 VPC，难点还是在解决异构环境的打通、大型网络下复杂的规划和管理。本文抛砖引玉，期待与各位的深入探讨。

# GTLC GLOBAL TECH LEADERSHIP CONFERENCE

全球技术领导力峰会

BEIJING · 2018

聚变

加入 TGO 鲲鹏会与 500 位 CTO 共同成长

成为 TGO 会员，尊享 0 元购票 >>

## 大会简介

GTLC全球技术领导力峰会是由极客邦旗下TGO鲲鹏会主办的高端技术领导人盛会，大会倾力策划优质分享与活动，以社交为核心，为技术领导者提供学习交流、提升视野与拓展人脉的平台。

本届GTLC将邀请互联网及传统行业权威技术领袖，面向CTO、技术VP、技术团队LEADER、技术项目负责人等对领导力感兴趣的技术人，以大会演讲、深度培训、高端社交等多种形式，从不同领域、不同方向，分享他们关于技术、行业、商业、投资、领导力的实践与蜕变经验。

## 大会主题

技术影响力

技术规划与选型

新热技术落地评估

组织与管理

洞察力与决策力

商业运营与VC思维



扫码了解更多

# 极客邦企业服务

帮助企业和技术人成长

7000+

200

300

Growing Technicians  
Growing Companies

极客邦企业服务涵盖9大技术领域的200多名一线技术专家通过公开课、企业内训和技术咨询帮助企业了解新技术的应用、趋势及前景，分享业界领先的技术应用实践，通过行业对标，提升企业技术选型与应用能力。迄今已有300多家企业的7000多位学员通过极客邦企业服务提升技术能力和管理能力。



## 《产品与服务》



### 「公开课」

为来自不同企业的技术人员提供人工智能、大数据、架构、移动与前端、运维容器、技术管理等不同技术领域的实践驱动的系统培训课程，1-2天深度培训，线下公开课形式，全年在全国各地开展。



### 「企业内训」

根据企业研发团队技术能力培养需求，为企业定制培训课程，邀请特约讲师到企业内部与研发团队面对面进行系统化技术课程辅导，提升企业整体研发能力。



### 「技术咨询」

为企业提供具体技术问题的咨询服务，包含但不限于以下领域：架构优化、微服务架构、运维优化、容器化调度、大数据、人工智能技术、工程效率、技术管理。



架构优化



微服务架构



运维优化



容器化调度



大数据



人工智能



工程效率



技术管理



区块链



免费获取课程信息

企业培训顾问：Rodin（罗丹）

电话：15002200534

邮箱：rodin@geekbang.org



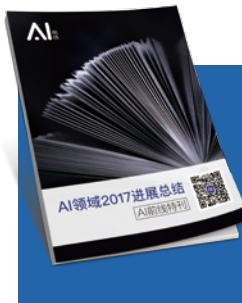
## 架构师 月刊 2018年3月

本期主要内容：Dubbo 正式进入 Apache 孵化器，开启开源新时代；批处理 ETL 已经消亡，Apache Kafka 才是数据处理的未来吗？Apache Kylin 在绿城客户画像系统中的实践；微服务架构技术栈选型手册。



## 深度学习器： TensorFlow程序设计

本书详细介绍了 TensorFlow 程序设计中的几个关键技术。



## AI前线特刊： AI领域2017进展总结

AI 前线在 2018 年之初为各位读者奉上这样一本迷你书，涵盖了来自全球 AI 和大数据领域技术专家的年终总结与趋势解读，同时还有世界知名技术大厂的年终技术总结与趋势预测。



## 架构师特刊 范式大学

构建商业 AI 能力的五大要素；判别 AI 改造企业的 70 个指标；用最小成本 验证 AI 可行性；企业技术人员如何向人工智能靠拢？人工智能的下一个技术风口与商业风口。