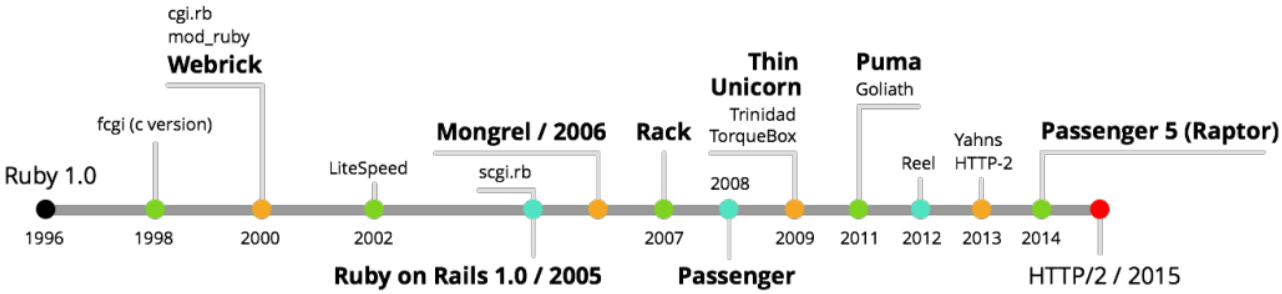
分类： 新兴技术

Ruby Web服务器：这十五年

坦率的说，作为一门年轻的计算机语言，Ruby在最近二十年里的发展并不算慢。但如果与坐拥豪门的明星语言们相比，Ruby就颇显平民范儿，表现始终不温不火，批评胜于褒奖，下行多过上扬。但总有一些至少曾经自称过Rubyist的程序员们，愉快地实践了这门语言，他们没有丝毫的歧视习惯，总是努力尝试各家之长，以语言表达思想，用基准评判高下，一不小心就影响了整个技术发展的进程。本文谨以Ruby Web服务器技术的发展为线索，回顾Ruby截至目前最为人所知的Web领域中，重要性数一数二的服务器技术的发展历程，试图帮助我们了解过去，预见未来。



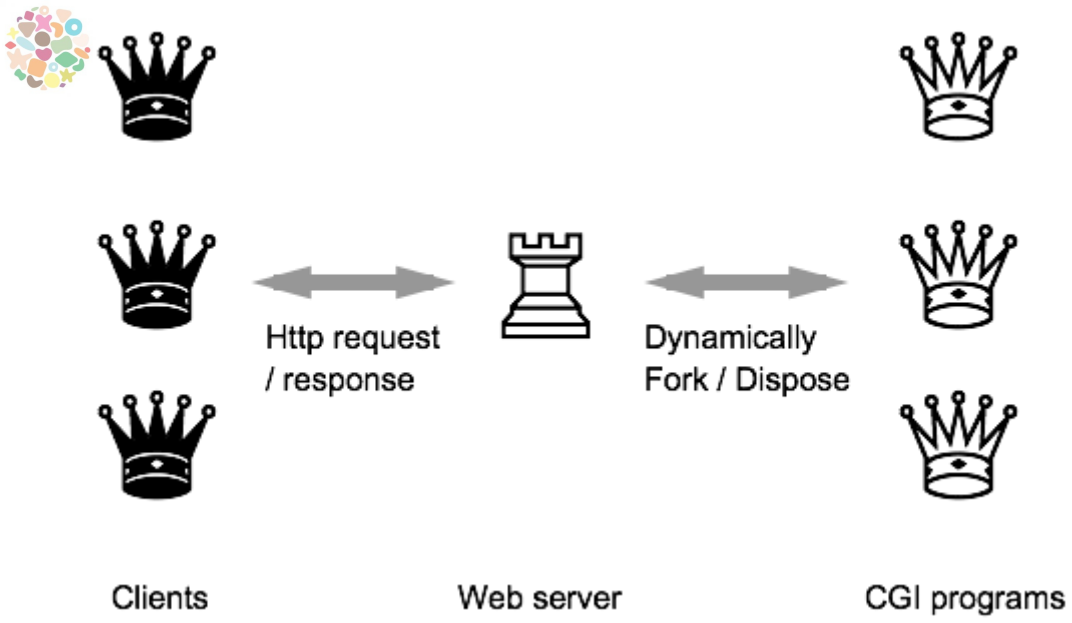
Ruby Web服务器发展时间轴

一、随波逐流

长久以来，任何Web服务器都具备的两项最重要的功能：一是根据RFC2616解析HTTP/1.1协议，二是接收、处理并响应客户端的HTTP请求。幸运的是Web技术的发展并不算太早，使得Ruby恰好能赶上这趟顺风车，但在前期也基本上受限于整个业界的进展。像Apache HTTP Server、Lighttpd和Nginx这些通用型Web服务器 + 合适的Web服务器接口即可完成大部分工作，而当时开发者的重心则是放在接口实现上。

cgi.rb

作为Web服务器接口的早期标准，CGI程序在调用过程中，通过环境变量（GET）或$stdin（POST）传递参数，然后将结果返回至$stdout，从而完成Web服务器和应用程序之间的通信。cgi.rb是Ruby官方的CGI协议标准库，发布于2000年的cgi.rb包含HTTP参数获取、Cookie/Session管理、以及生成HTML内容等基本功能。



Web服务器和CGI

当支持CGI应用的Web服务器接到HTTP请求时，需要先创建一个CGI应用进程，并传入相应的参数，当该请求被返回时再销毁该进程。因此CGI原生是单一进程/请求的，特别是每次请求时产生的进程创建/销毁操作消耗了大量系统资源，根本无法满足较高负载的HTTP请求。此外，CGI进程模型还限制了数据库连接池、内存缓存等资源的复用。

对于标准CGI应用存在的单一进程问题，各大厂商分别提出了兼容CGI协议的解决方案，包括网景的NSAPI、微软的ISAPI和后来的Apache API（ASAPI）。上述服务器API的特点是既支持在服务器进程内运行CGI程序，也支持在独立进程中运行CGI程序，但通常需要在服务器进程中嵌入一个插件以支持该API。

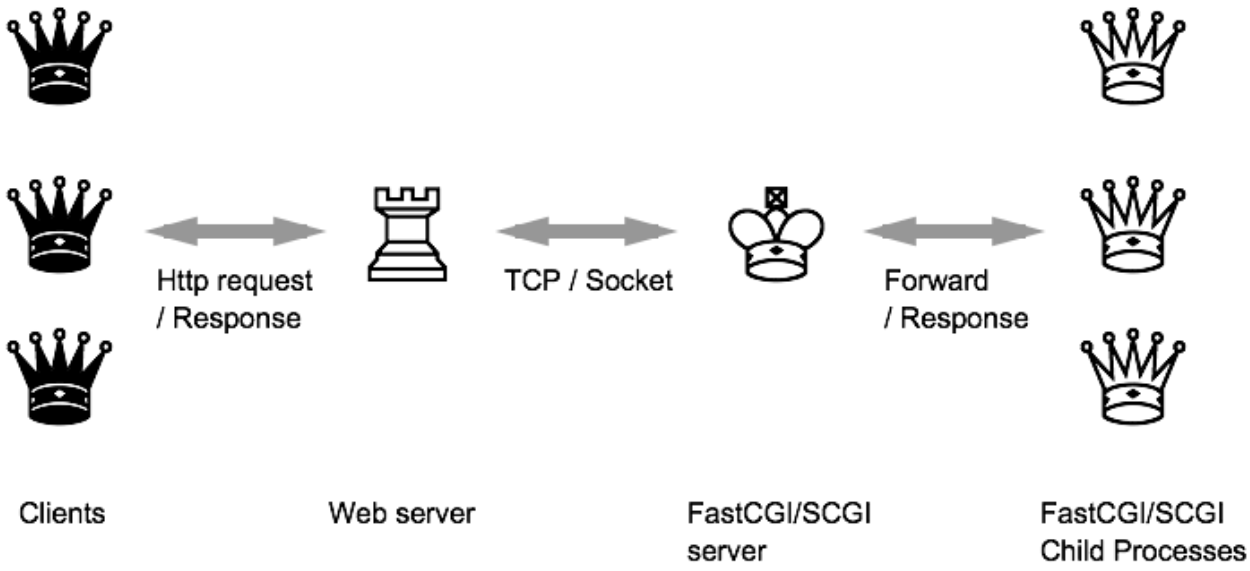
Webrick

作为最古老的Ruby Web服务器而不仅仅是一个接口，诞生于2000年的Webrick从Ruby 1.9.3（2011年10月正式发布）起被正式纳入标准库，成为Ruby的默认Web服务器API。Webrick支持HTTP/HTTPS、代理服务器、虚拟主机服务器，以及HTTP基础认证等RFC2617及以外的其它认证算法。同时，一个Webrick服务器还能由多个Webrick服务器或服务器小程序组合，提供类似虚拟主机或路由等功能：例如处理CGI脚本、ERb页面、Ruby块以及目录服务等。

Webrick曾被用于Rails核心团队的开发和测试中。但是，Webrick内置的HTTP Parser非常古老，文档缺失，性能低下且不易维护，功能单一且默认只支持单进程模式（但支持多线程，不过在Rails中默认关闭了对Webrick的多线程支持），根本无法满足产品环境中的并发和日常维护需求。目前一般只用于Web应用的本地开发和基准测试。

fcgi.rb

fcgi.rb是FastCGI协议的Ruby封装（latest版底层依赖libfcgi）。为了与当时的NSAPI竞争，FastCGI协议最初由Open Market提出和开发、并应用于自家Web服务器，延续了前者采用独立进程处理请求的做法：即维持一个FastCGI服务器。当Web服务器接收到HTTP请求时，请求内容和环境信息被通过Socket（本地）或TCP连接（远程）的方式传递至FastCGI服务器进行处理，再通过相反路径返回响应信息。分离进程的好处是Web服务器进程和FastCGI进程是永远保持的，只有相互之间的连接会被断开，避免了进程管理的开销。



Web服务器和FastCGI/SCGI服务器

进一步，FastCGI还支持同时响应多个请求。为了尽量减少资源浪费，若干请求可以复用同一个与Web服务器之间的连接，且支持扩展至多个FastCGI服务器进程。FastCGI降低了Web服务器和应用程序之间的耦合度，进而为解决安全、性能、管理等各方面问题提供新的思路，相比一些嵌入式方案如mod_perl和mod_php更具灵活性。

由于FastCGI协议的开放性，主流Web服务器产品基本都实现了各自的FastCGI插件，从而导致FastCGI方案被广泛使用。fcgi.rb最早开发于1998年，底层包含C和Ruby两种实现方式，早期曾被广泛应用于Rails应用的产品环境。

mod_ruby

mod_ruby是专门针对Apache HTTP Server的Ruby扩展插件，支持在Web服务器中直接运行Ruby CGI代码。由于mod_ruby在多个Apache进程中只能共享同一个Ruby解释器，

意味着当同时运行多个Web应用（如Rails）时会发生冲突，存在安全隐患。因此只在一些简单部署环境下被采用，实际上并没有普及。

LiteSpeed API/RubyRunner

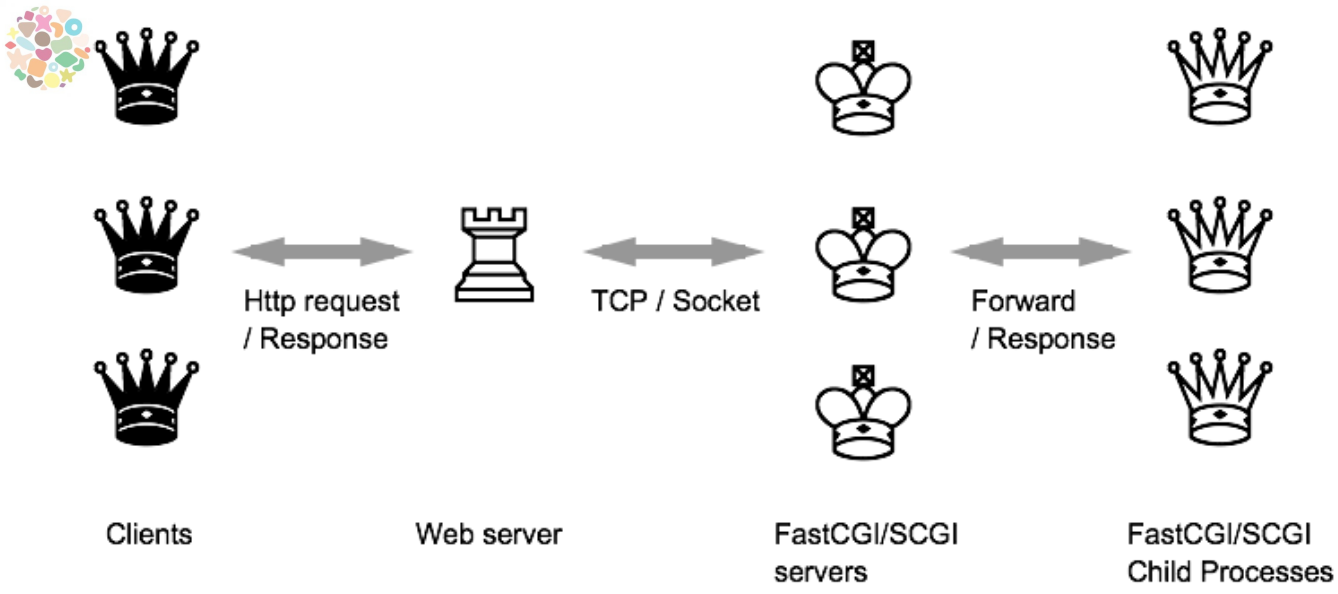
LiteSpeed是由LiteSpeed Tech公司最初于2002年发布的商用Web服务器，特点是与被广泛采用的Apache Web服务器的配置文件兼容，但因为采用了事件驱动架构而具有更好的性能。

LiteSpeed API（LSAPI）是LiteSpeed专有的服务器API，LSAPI具备深度优化的IPC协议以提升通信性能。类似其它Web服务器，LiteSpeed支持运行CGI、FastCGI、以及后来的Mongrel。同时在LSAPI的基础上开发了Ruby接口模块，支持运行基于Ruby的Web应用。此外，LiteSpeed还提供RubyRunner插件，允许采用第三方Ruby解释器运行Ruby应用，但综合性能不如直接基于LSAPI Ruby。

由于LiteSpeed是收费产品，其普及率并不高，一般会考虑采用LiteSpeed作为Web服务器的业务场景包括虚拟主机/VPS提供商、以及相关业务的cPanel产品。同时，LiteSpeed也会被用于一些业务需求比较特殊的场合，例如对Web服务器性能要求高，且应用程序及其部署需要兼容Apache服务器。LiteSpeed于2013年发布了开源的轻量Web服务器——OpenLiteSpeed（GPL v3），移除了商业版本中偏具体业务的功能如cPanel等，更倾向于成为通用Web服务器。

scgi.rb

scgi.rb是对SCGI协议的纯Ruby实现。从原理上来看，SCGI和FastCGI类似，二者的性能并无多大差别。但比起后者复杂的协议内容来说，SCGI移除了许多非必要的功能，看起来十分简洁，且实现复杂度更低。



Web服务器和多FastCGI/SCGI服务器

与FastCGI类似，一个SCGI服务器可以动态创建服务器子进程用于处理更多请求（处理完毕将转入睡眠），直至达到配置的子进程上限。当获得Web服务器请求时，SCGI服务器进程会将其转发至子进程，并由子进程运行CGI程序处理该请求。此外，SCGI还能自动销毁退出和崩溃的子进程，具有良好的稳定性。

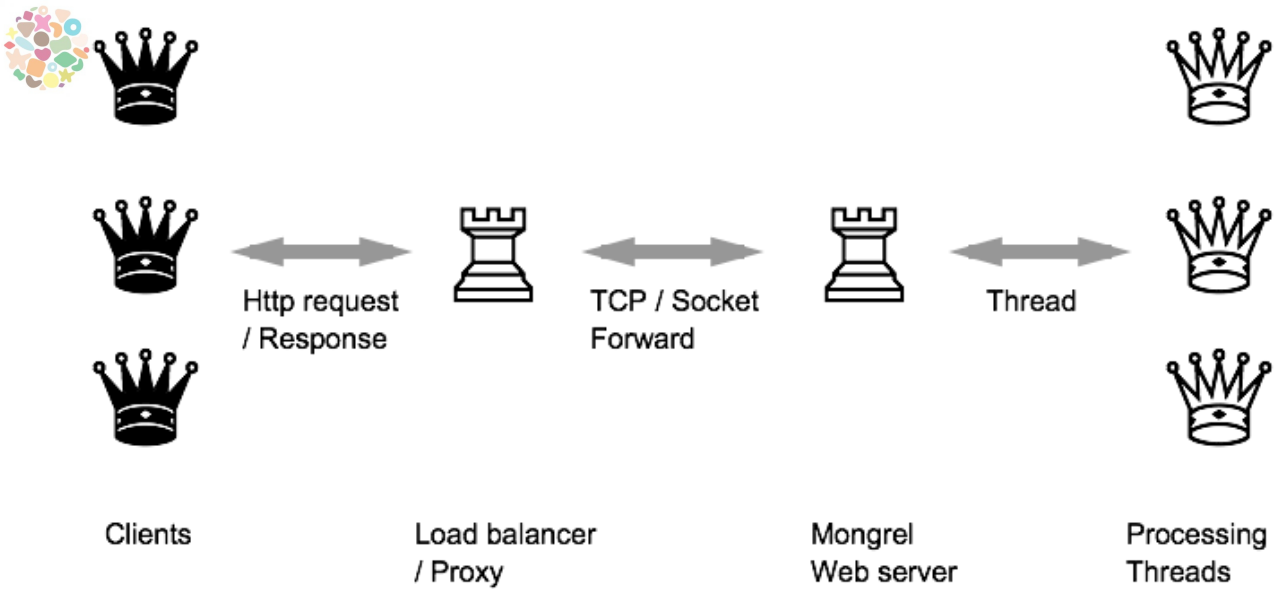
二、闻名天下

2005年，David Heinemeier Hansson（DHH）发布了基于Ruby的开发框架Ruby on Rails（Rails），聚光灯第一次聚焦在Ruby身上。但是业内普遍对Web服务器的方案感到棘手，本地环境Webrick/产品环境FastCGI + 通用Web服务器几乎成了标配，无论是开发、部署或维护都遇到不少困难，一些吃螃蟹的人遂把此视为Rails不如J2EE、PHP方案的证据。

Mongrel

2006年，Zed Shaw发布了划时代的Mongrel。Mongrel把自己定位成一个“应用服务器”，因为其不仅可以运行Ruby Web应用，也提供标准的HTTP接口，从而使Mongrel可以被放置在Web代理、Load Balancer等任意类型的转发器后面，而非像FastCGI、SCGI一样通过调用脚本实现Web服务器和CGI程序的通信。

Mongrel采用Ragel开发HTTP/1.1协议的Ruby parser，而后者是一个高性能有限自动机编译器，支持开发协议/数据parser、词法分析器和用户输入验证，支持编译成多种主流语言（包括Ruby）。采用Regel也使parser具有更好的可移植性。但是，Mongrel本身不支持任何应用程序框架，而需要由框架自身提供这种支持。



Mongrel Web服务器

Mongrel支持多线程运行（但对于当时非线程安全的Rails来说，仍然只能采用多进程的方式提高一部分并发能力），曾被Twitter作为其第一代Web服务器，还启发了Ryan Dahl发布于2009年的Node.JS。

但是当Mongrel发布后没过多久，Shaw就与Rails社区的核心成员不和（实际上Shaw对业界的许多技术和公司都表达过不满），随后就终止了Mongrel的开发。进而在其Parser的基础上开发了其后续——语言无关的Web服务器Mongrel2（与前续毫无关系）。

尽管Mongrel迅速衰落，却成功启发了随后更多优秀Ruby应用服务器的诞生，例如后文将介绍的Thin、Unicorn和Puma。

Rack

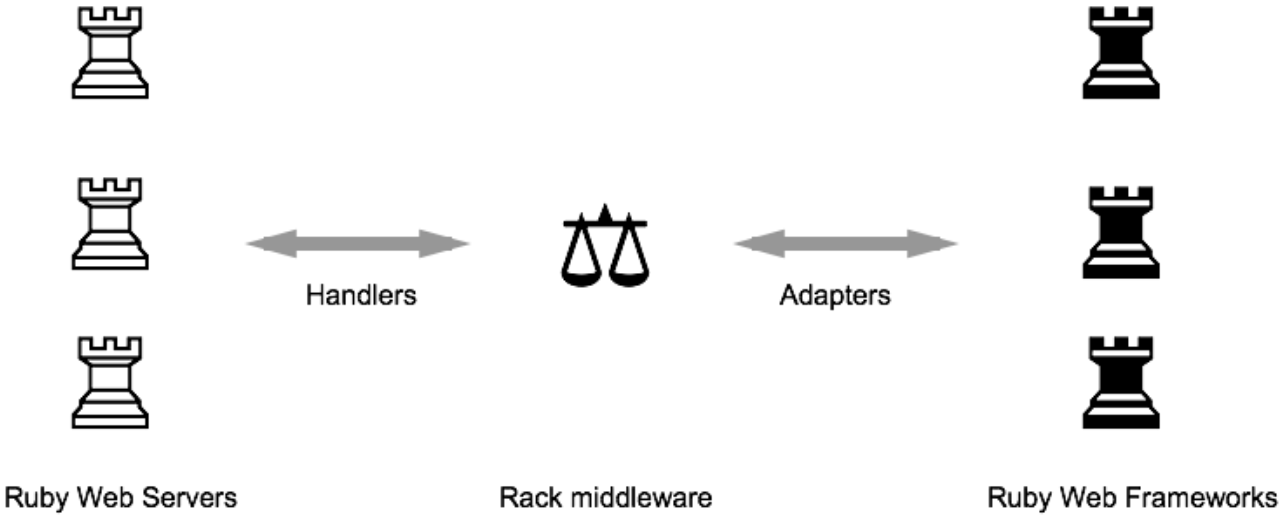
随着Web服务器接口技术的发展，从开始时作为一个module嵌入Web服务器，到维护独立的应用服务器进程，越来越多的应用服务器产品开始涌现，同时相互之间还产生了差异化以便适应不同的应用场景。但是，由于底层协议和API的差别，基于不同的应用服务器开发Web产品时，意味着要实现各自的通信接口，从而为Web应用开发带来更多工作量。特别是对于类似Django、Rails这些被广泛使用的Web框架来说，兼容主流应用服务器几乎是必须的。

2003年，Python界权威Phillip J. Eby发表了PEP 0333（Python Web Server Gateway Interface v1.0，即WSGI），提出一种Web服务器和应用程序之间的统一接口，该接口封装了包括CGI、FastCGI、mod_python等主流方案的API，使遵循WSGI的Python Web应

用能够直接部署在各类Web服务器上。与Python的发展轨迹相似，Ruby界也遇到了类似的挑战，并最终在2007年出现了与WSGI类似的Rack。

与WSGI最初只作为一份建议不同，Rack直接提供了模块化的框架实现，并由于良好的设计架构迅速统一了Ruby Web服务器和应用程序框架接口。

Rack被设计成一种中间件“框架”，接收到的HTTP请求会被rack放入不同的管线（中间件）进行处理，直到从应用程序获取响应。这种设计通过统一接口，把一般Web应用所需的底层依赖，包括Session处理、数据库操作、请求处理、渲染视图、路由/调度、以及表单处理等组件以中间件的形式“放入”rack的中间件管线中，并在HTTP请求/响应发生时依次通过上述管线传递至应用程序，从而实现Web应用程序对底层通信依赖的解绑。



Rack中间件

Rack接口部分包含两类组件：Handler，用于和Web服务器通信；Adapter，用于和应用程序通信。截至Rack 1.6，Rack内置的handlers包括WEBrick、FCGI、CGI、SCGI、LiteSpeed以及Thin，上述handlers用以兼容已有的常见应用服务器。而2008年后，随着rack逐渐成为事实标准，更新的Ruby Web服务器几乎都包含Rack提供的handler。包括Rails、Sinatra、Merb等等几乎所有主流框架都引入了Rack Adapters的支持。

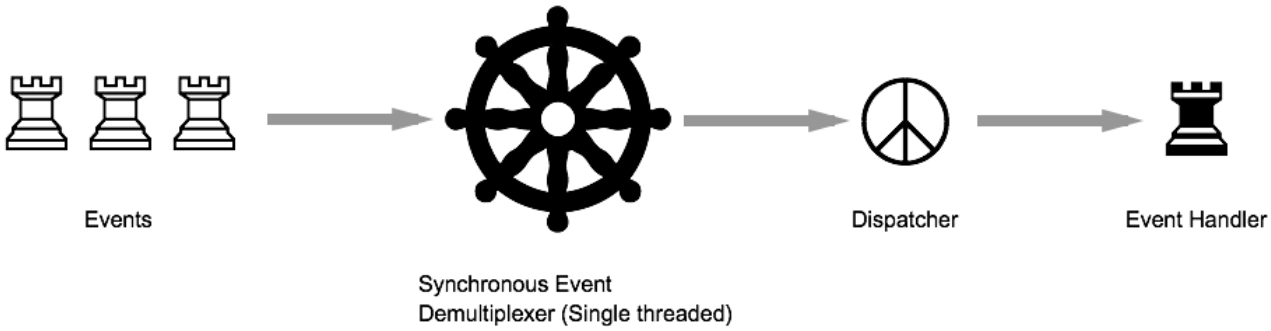
三、百花齐放

Mongrel和Rack的相继诞生，使Ruby Web服务器、乃至应用程序框架的发展有了一定意义上可以遵循的标准。Mongrel后相继派生出Thin、Unicorn和Puma；而Rack统一了Ruby Web服务器和应用程序框架接口，使应用开发不再需要考虑特定的部署平台。Ruby Web服务器开始依据特定需求深入发展。

Thin/Goliath

发布于2009年的Thin沿用了Mongrel的Parser，基于Rack和EventMachine开发，前者上文已有介绍，EventMachine是一个Ruby编写的、基于Reactor模式的轻量级事件驱动I/O（类似JBoss Netty、Apache MINA、Python Twisted、Node.js、libevent和libev等）和并发库，使Thin能够在面对慢客户端的同时支持高并发请求。

发表自1995年的Reactor模型的基本原理是采用一个单线程事件循环缓存所有系统事件，当事件发生时，以同步方式将该事件发送至处理模块，处理完成后返回结果。基于Reactor模型的EventMachine具备异步（非阻塞）I/O的能力，被广泛用于大部分基于Ruby的事件驱动服务器、异步客户端、网络代理以及监控工具中。



Reactor模型

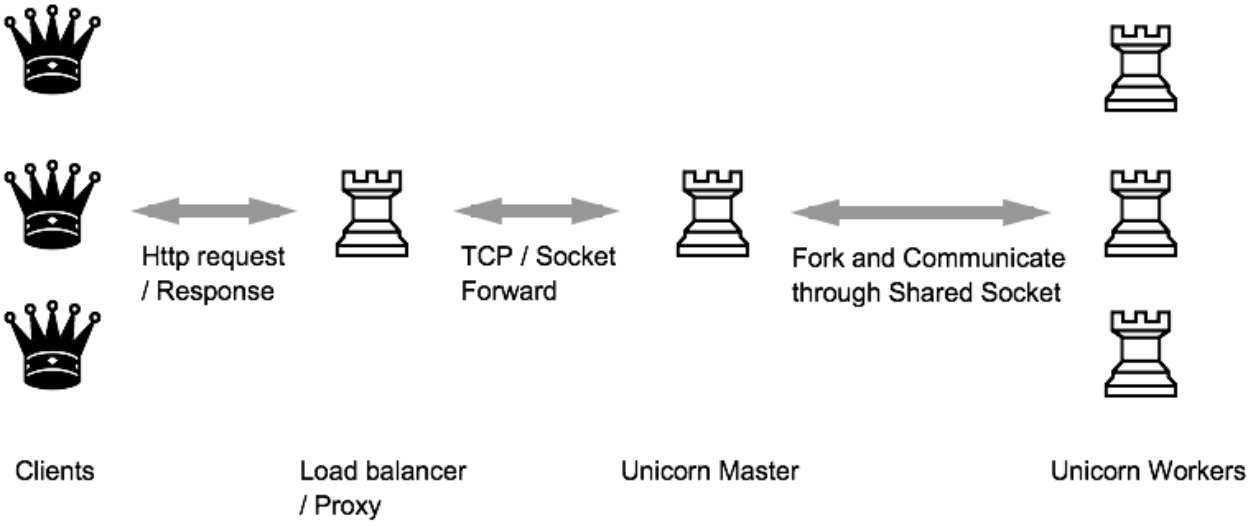
2011年，社交网络分析商PostRank开源了其Web服务器Goliath，与Thin相似（都采用了EventMachine）但又有很大不同，采用新的HTTP Parser，同时针对异步事件编程中的高复杂度回调函数问题，借助Ruby1.9+的纤程技术实现了线性编码，使程序具备更好的可维护性。Goliath支持MRI、JRuby和Rubinius等多平台。在附加功能方面，Goliath的目标不仅是作为Web服务器，更是一个快速构建WebServices/APIs的开发框架，但是随着之后PostRank被Google收购，Goliath项目也就不再活跃在开源界了。

Unicorn

2009年，Eric Wong在Mongrel 1.1.5版本的基础上开发了Unicorn。Unicorn是一个基于Unix/类Unix操作系统的、面向快客户端、低延迟和高带宽场景的Rack服务器，基于上述限制，任何情况下几乎都需要在Unicorn和客户端之间设置一个反向代理缓存请求和响应数据，这是Unicorn的设计特点所决定的，但也使得Unicorn的内部实现相对简洁、可靠。

尽管来源于Mongrel，但Unicorn只在进程级运行，且吸收和利用了一些Unix/类Unix系统内核的特性，如Prefork模型。

Unicorn由1个master进程和n个fork(2)子进程组成，子进程分别调用select(2)阻塞自己，直到出错或者超时时，才做一些写日志、处理信号以及维护与master的心跳链接等内置任务。子进程和master间通过一个共享socket实现通信，而由Unix/类Unix系统内核自身处理资源调度。



Unicorn的多进程模型

Unicorn的设计理念是“只专注一件事”：多进程阻塞I/O的方式令其无从接受慢客户端——但前置反向代理能解决这一问题；workers的负载均衡就直接交给操作系统处理。这种理念大大降低了实现复杂度，从而提高了自身可靠性。此外，类似Nginx的重加载机制，Unicorn也支持零宕机重新加载配置文件，使其允许在线部署Web应用而不用产生离线成本。

Phusion Passenger（mod_rails/mod_rack）

2008年初，一位叫赖洪礼的Ruby神童发布了mod_rails。尽管Mongrel在当时已经席卷Rails的Web服务器市场，但是面对部署共享主机或是集群的情况时还是缺少统一有效的解决方案，引起业内一些抱怨，包括DHH（也许Shaw就不认为这是个事儿）。

mod_rails最初被设计成一个Apache的module，与FastCGI的原理类似，但设置起来异常简单——只需要设置一个RailsBaseURI匹配转发至Rails服务器的URI串。mod_rails服务器会在启动时自动加载Web应用程序，然后按需创建子进程，并协调Web服务器和Rails服务器的通信，从而支持单一服务器同时部署多个应用，还允许按需自动重启应用服务器。

mod_rails遵循了Rails的设计原则，包括Convention over Configuration、Don't Repeat Yourself，使其面向部署非常友好，很快得到了业界青睐，并在正式release时改名Passenger。

在随后的发展中，Passenger逐渐成为独立的Ruby应用服务器、支持多平台的Web服务器。截至2015年6月，Phusion Passenger的版本号已经达到5.0.10（Raptor），核心采用C++编写，同时支持Ruby、Python和Node.js应用。支持Apache、Nginx和独立HTTP模式（推荐采用独立模式），支持Unix/类Unix系统，在统计网站Builtwith上排名Ruby Web服务器使用率第一。

值得一提的是，Phusion Passenger的开源版本支持多进程模式，但是其企业版同样支持多线程运行。本文撰写时，Phusion Passenger是最近一个号称"史上最快"的Ruby Web服务器（本文最后将进一步介绍Raptor）。

Trinidad/TorqueBox

Trinidad发布于2009年，基于JRuby::Rack和Apache Tomcat，使Rails的部署和世界上最流行的Web服务器之一Tomcat结合，支持集成Java代码、支持多线程的Resque和Delayed::Job等Worker，也支持除Tomcat以外的其它Servlet容器。

与Trinidad相比，同样发布于2009年的TorqueBox不仅仅是一个Web服务器，而且被设计成一个可移植的Ruby平台。基于JRuby::Rack和WildFly（JBoss AS），支持多线程阻塞I/O，内置对消息、调度、缓存和后台进程的支持。同时具有集群、负载均衡、高可用等多种附加功能。

Puma

Puma——Mongrel最年轻的后代于2011年发布，作者是Evan Phoenix。

由于Mongrel诞生于前Rack时期，而随着Rack统一了Web服务器接口，任何基于Rack的应用再与Mongrel配合就有许多不便。Puma继承了前者的Parser，并且基于Rack重写了底层通信部分。更重要的是，Puma部分依赖Ruby的其它两个流行实现：Rubinius和JRuby，与TorqueBox类似拥有多线程阻塞I/O的能力（MRI平台不支持真正意义上的多线程，但Puma依然具有良好并发能力），支持高并发。同时Puma还包含了一个事件I/O模块以缓冲HTTP请求，以降低慢客户端的影响。但是，从获得更高吞吐量的角度来说，Puma目前仍然需要采用Rubinius和JRuby这两个平台。

Reel

Reel是最初由Tony Arcieri发布于2012年的采用事件I/O的Web服务器。采用了不同于Eventmachine的Celluloid::IO，后者基于Celluloid——Actor并发模型的Ruby实现库，解决了EM只能在单一线程中运行事件循环程序的问题，从而同时支持多线程 + 事件I/O，在非阻塞I/O和多线程方案间实现了良好的融合。

与其它现代Ruby Web服务器不同的是，Reel并不是基于Rack创建，但通过Reel::Rack提供支持Rack的Adapter。尽管支持Rails，与Puma也有一定的相似性，但与Unicorn、Puma和Raptor相比，Reel在部署Rails/Rack应用方面缺少易用性。实际上基于Celluloid本身的普及程度和擅长领域，相比其它Web服务器而言，Reel更适合部署WebSocket/Stream应用。

Yahns

2013年，Eric Wong等人受Kqueue（源自FreeBSD，同时被Node.js作为基础事件I/O库）的启发启动了Yahns项目。其目标与Reel类似，同样是在非阻塞I/O设计中引入多线程。与Reel不同的是，Yahns原生支持Rack/HTTP应用。

Yahns被设计成具有良好的伸缩性和轻量化特性，当系统应用访问量较低或为零时，Yahns本身的资源消耗也会保持在较低水平。此外，yahns只支持GNU/Linux（并通过kqueue支持FreeBSD），并声称永远不会支持类似Unicorn或Passenger里的Watchdog技术，不会因为应用崩溃而自动销毁和创建进程/线程，因此对应用程序本身的可靠性有一定要求。

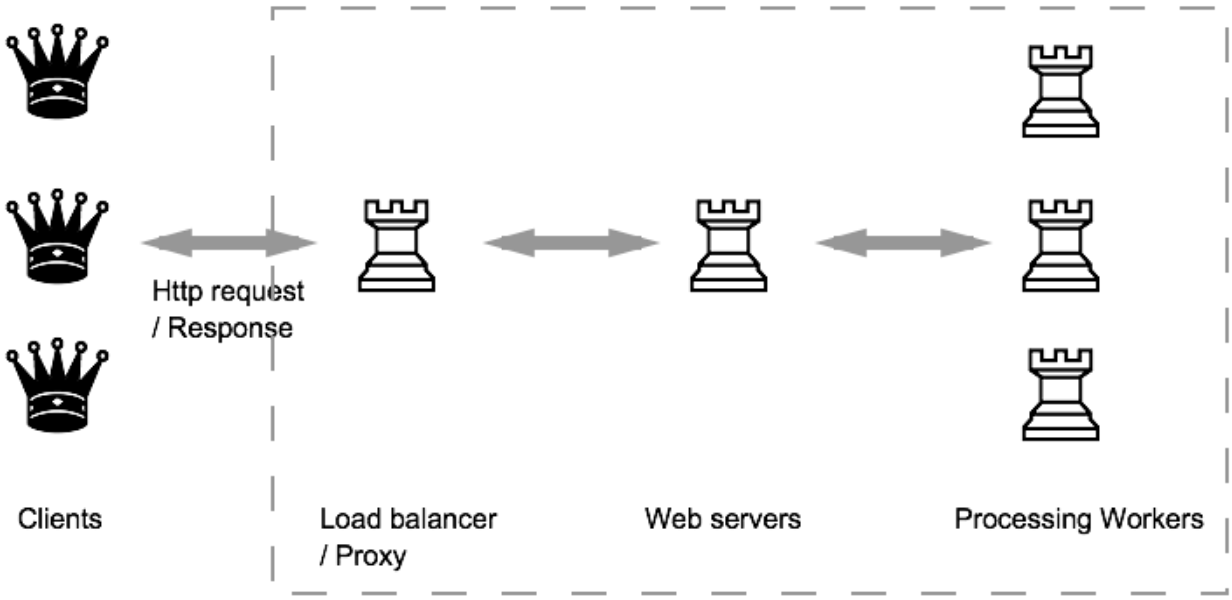
四、迈向未来

回顾过去，Ruby Web服务器在发展中先后解决了缺少部署方案、与Web应用程序不兼容、运维管理困难等问题，基础架构趋于成熟且稳定。而随着更多基准测试结果的出现，业界逐渐开始朝着更高性能和并发量发展，同时针对HTTP协议本身的优化和扩展引入的HTTP/2，以及HTML5的WebSocket/Stream等需求均成为未来Ruby Web服务器发展的方向。

高吞吐量

以最新的Raptor（上文提到的Phusion Passenger 5）为例，其在网络I/O模型的选择上融合了现有其它优秀产品的方案，包括Unicorn的多进程模型、内置基于多线程和事件I/O模型的反向代理缓冲（类似Nginx的功能，但对Raptor自身做了大量裁减和优化）、以及企业版具有的多线程模型（类似Puma和TorqueBox）；此外，Raptor采用的Node.js HTTP

Parser（基于Nginx的Parser）的性能超过了Mongrel；更进一步，Raptor甚至实现了Zero-copy和一般在大型游戏中使用的区域内存管理技术，使其对CPU和内存访问的优化达到了极致（感兴趣的话可以进一步查阅这里）。



Raptor的优化模型

另外也需要看到，当引入多线程运行方式，现有Web应用将不得不仔细检查自身及其依赖，是否是线程安全的，同时这也给构建Ruby Web应用带来更多新的挑战。这也是为什么更多人宁愿选择进程级应用服务器的方式——毕竟对大多数应用来说不需要用到太多横向扩展，引入反向代理即可解决慢客户端的问题，而采用Raptor甚至在独立模式能工作的更好（这样就不用花时间去学习Nginx）。

除非你已经开始考虑向支持大规模并发的架构迁移，并希望节省接下来的一大笔花费了。

HTTP/2

2015年5月，HTTP/2随着RFC7540正式发布。如今各主流服务器/浏览器厂商正在逐渐完成从HTTP/2测试模块到正式版本的过渡。而截至目前，主流Ruby Web服务器都还没有公开HTTP/2的开发信息。HTTP-2是在2013年由Ilya Grigorik发布的纯Ruby的HTTP/2协议实现，包括二进制帧的解析与编码、流传输的多路复用和优先级制定、连接和流传输的流量控制、报头压缩与服务器推送、连接和流传输管理等功能。随着HTTP/2的发布和普及，主流Ruby Web服务器将不可避免的引入对HTTP/2的支持。

WebSocket/流（Stream）/服务器推送事件（Server Sent Events，SSE）

2011年，RFC6455正式公布了WebSocket协议。WebSocket用于在一个TCP链接上实现全双工通信，其目的是实现客户端与服务器之间更高频次的交互，以完成实时互动业务。鉴于该特点，仅支持慢客户端的Web服务器就无法有效支撑WebSocket的并发需求，更何况后者对并发量更加严苛的要求了。而对于同样需要长连接的流服务器和服务器推送事件服务（SSE），都避免不了对长连接和高并发量的需求。尽管高性能的Ruby Web服务器都有足够的潜力完成这些任务，但是从原生设计的角度来看，更加年轻的Reel和Yahns无疑具有优势。

最近Planet ruby在Ruby邮件组发布了Awesome Webservers，该Github Repo旨在对主流Ruby Web服务器进行总结和对比，并且保持持续更新，推荐开发者关注。

发表评论

DeathKing

2015年8月14日

看完这十五年，不得不让我联想到太祖的诗：“江山代有才人出，各领风骚数百年”啊！

Ruby 在性能方面的路，任重而道远。

回复 DeathKing

标签： 韩翼