Computer Science and Application 计算机科学与应用, 2013, 3, 191-194
doi:10.12677/csa.2013.33033 Published Online June 2013 (http://www.hanspub.org/journal/csa.html)

Study of Asynchronous Non-Blocking Server Based on Nodejs

Rujia Li, Jing Hu

Luojia College, Wuhan University, Wuhan
Email: lirujia00@live.cn, 50049625@qq.com

Received: Apr. 19th, 2013; revised: May 2nd, 2013; accepted: May 14th, 2013

Abstract: The old blocking multi-threaded server architecture always encounter the limitations of performance when face these problems such as large data, long connection, high concurrent requests. Nodejs will change the maximum number of connections to the flow of a single system to solve the bottleneck. This article aims to explore the difference between asynchronous non-blocking server and blocking multi-threaded server from work and running performance. Then it explains the superiority of Nodejs in processing large data, long connection and high concurrent requests.

Keywords: Server; Nodejs; High Concurrent; Performance

基于 Nodejs 的异步非阻塞服务器研究

李汝佳，胡　婧

武汉大学珞珈学院，武汉
Email: lirujia00@live.cn, 50049625@qq.com

收稿日期：2013 年 4 月 19 日；修回日期：2013 年 5 月 2 日；录用日期：2013 年 5 月 14 日

摘　要：传统的阻塞式多线程服务器体系在面对大数据量、长连接、高并发请求时常常遇到性能方面的局限，而新兴的 Nodejs 通过将最大连接数量更改到单个系统的流量来解决以上的瓶颈。本文旨在从工作方式和运行性能上探究 Nodejs 搭建的异步非阻塞服务器与传统的阻塞多线程服务器的区别。进而说明 Nodejs 在处理大数据量、长连接、高并发请求时的优越性。

关键词：服务器；Nodejs；高并发；性能

1. 引言

Nodejs 是一个 Javascript 运行环境，其核心是一个独立的 Javascript 虚拟机，这个虚拟机实际上是对谷歌 GoogleV8 引擎进行了封装，谷歌 GoogleV8 引擎本身使用了一些最新的编译技术，采用一系列的非阻塞库来支持事件循环的方式，将谷歌 GoogleV8 引擎提供的非阻塞 I/O 栈与 Javascript 提供的闭包和匿名函数相结合[1]，是编写高吞吐量网络的优秀平台，它可以创建快速、可扩展的网络应用程序。

2. Nodejs 技术综述

在 Web 体系中传统技术如 Java，C#等技术的最终目的是为生成网页，进而在页面上展示数据，这些高级语言的编译解析需要一定时间，再加上线程之间切换需要高昂的代价。在面对大数据量、长连接、高并发请求时常常遇到性能方面的局限，在解决这些问题时，通常使用高性能的 Web 容器，高性能的服务器等，这些措施有一定效果但不能完全消除这种瓶颈，Nodejs 从根本上改变了这一现状，一方面浏览器可以

很好的解析 Javascript，而 Javascript 又可以完全操纵 html[1]；当浏览器与服务器的语言同属于 Javascript 时，没必要进行二次解析，可以迅速生成 html，这也就大大减少了系统开销；另一方面 Nodejs 通过将最大连接数量更改到单个系统的流量来解决以上的瓶颈。下面将对 Nodejs 技术进行分解，分解为以下三点核心技术，并进行探究。

2.1. GoogleV8 引擎

GoogleV8 引擎是谷歌的开放源代码，由 C++语言编写，谷歌 GoogleV8 引擎中实现了 ECMAScript 中指定的 ECMA-262，可以独立运行，也可以嵌入到任何 C++应用程序[2]。谷歌 GoogleV8 引擎在执行 Javascript 代码之前将其编译成了机器码，而非字节码或者是直译它，因此提升效能。而且谷歌 GoogleV8 引擎使用了如内联缓存等方法来提高性能。由于这些优化，Javascript 程序与谷歌 GoogleV8 引擎的速度媲美二进制编译[3]。同时它还具有管理内存，负责垃圾回收，与宿主语言的交互等功能，这一系列的功能使之成为世界上最快的 Javascript 解析器之一，Nodejs 开发团队在此基础上添加了一些模块如 HTTP、TCP、DNS 等从而创造了 Nodejs。

2.2. Nodejs 组成结构

Nodejs 有 80%的 C/C++代码与 20%的 Javascript 组成，Nodejs 除了使用谷歌 GoogleV8 引擎作为 Javascript 引擎外，还使用了高效的 libev 和 libeio 库支持事件驱动和异步 I/O[4]，其组织结构如图 1 所示。

其中 C/C++的库主要负责运行 Javascript(通过谷歌的 V8 引擎)和为 HTTP、TCP 和 DNS 等提供支持，小部分的 Javascript 提供了 Nodejs 的原生库和模块如网络包装和文件读写，可以更好的方便开发者使用。

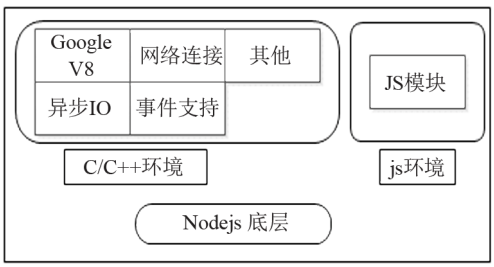

Figure 1. The structure of Nodejs
图 1. Nodejs 组织结构图

2.3. Nodejs 特性分析

Nodejs 采用事件驱动、异步编程、单线程、为网络服务而设计，在某些传统语言网络编程中会用到回调函数，比如当 socket 资源达到某种状态时，注册的回调函数就会执行。Nodejs 采用一系列非阻塞库来支持事件循环的方式，本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无需等待硬盘寻址并检索文件，硬盘准备好的时候非阻塞接口会通知 Nodejs 主线程进行回调[5]。

Nodejs 的设计思想中采用单线程，非严格意义单线程，因为基于 C/C++那部分是多线程，以事件驱动为核心，当某个请求到来时，主线程会触发响应时间执行，然后继续响应下一个请求，当上一个请求完成时，因为 Javascript 回调函数的机制[6]，会自动在事件循环列表中等待，直到主线在此唤醒它，在整个过程中始终保持了单线程，网络的响应时间包括请求时间、网络延迟时间、本地磁盘响应时间等，在同步体制下，网络的响应时间等于与所有响应时间之和，然而异步时网络的响应时间取决于响应时间的最大值，Nodejs 采用异步的编码风格在面临高并发时依然可以很好加快网络的响应时间，在所有的延迟中本地磁盘读写无疑是最耗时，Nodejs 事件循环在面对大数据量读写时不会阻碍主线程的响应，很好的解决了摘要中提到的问题。

3. Nodejs 服务器与普通 http 服务对比

Nodejs 服务器与普通 HTTP 服务，在多个方面存在差异，在这里以实际应用中问题为导向，引出工作原理和性能方面的不同，同时也表明本次探讨的意义。为了更好的引证本文的观点，我取了它们最重要的两点不同做对比，反映在这里就是一个单线程基于事件驱动非阻塞 I/O 服务器与一个多线程阻塞 I/O 的服务器的比较，下面将从工作原理和性能方面进行对比。

3.1. 实际应用中的差异

需求是从数据库表中查出某个字段，并打印出来，采用多线程阻塞 I/O 服务器的方式通常的做法是：

```
name = query("select name from db");
output(name);
```

数据库层把这个查询发送到数据库，查询结果返回数据库之间，进程就会阻塞对于高并发，I/O 密集行的网络应用中，一方面进程很长时间处于等待状态，一方面为了应付新的请求不断的增加新的进程[7]，每个进程执行栈都要占用内存，计算机的资源很快将会耗尽。若采用 Nodejs 基于事件驱动的非阻塞服务器的方式，以代码如下：

```
query("select name from db " function(name){
output(name);});
```

此时查询的结果不是调用函数的结果，而是提供给一个稍后将调用的回调函数，控制会立刻返回事件循环，而进程能够继续处理其他请求。

3.2. 工作原理对比

实际应用的差异的本质就是其工作原理的不同，Nodejs 构建的异步非阻塞服务器有两部分组成。在上层是谷歌 GoogleV8 引擎(单线程)，事件循环和 C/C++的库运行在其的基础上，同时负责监听 HTTP/TCP 请求，在服务器的底层有 libuv(包含 libio)和其他的 C++库。他们的主要作用是提供异步的 I/O。Nodejs 事件驱动非阻塞 I/O 服务器模型如图 2 所示。

图 2 描述了 Nodejs 服务器的工作原理，当一个请求从浏览器发出，运行在谷歌 GoogleV8 引擎中的主线程会检查它是否含有 I/O 请求，如果他是 I/O 请求会立刻委托给底层所在的服务器一个 POSIX 线程池来执行异步的 I/O[8]，主线是空闲的，所以可以接收其他请求，当与数据库或者文件交换数据完成后，事件触发会向主线程发出一个响应，告诉主线程已经读取完毕，当谷歌 GoogleV8 引擎执行完正在做的事情空闲时，它将获取这个结果，然后返回到浏览器。

线程阻塞式服务器与异步非阻塞服务器在工作原理上有着较大差异，图 3 是多线程阻塞式服务器的模型图，其中 T 表示一个线程。其中的核心模块是线程池，线程池的容量直接决定了服务器的并发数量。

图 3 模拟了有四个用户登录到多线程服务器，其中两个用户不停的点击刷新按钮导致服务器开启了许多线程(每一次请求开启一个线程)，当一个请求到来时，线程池中一个线程执行它的阻塞 I/O 操作，去数据库或者文件中进行数据的交换，线程池控制着线程的切换和线程上下文的执行，当 I/O 操作结束后，操作系统上下文切换回到先前的线程返回结果。请求直接面对这种阻塞的 I/O 要求更多的内存和 CPU 资源。

3.3. 性能对比

因为同为这种异步非阻塞体系结构，Nodejs 和 Nginx 性能上表现的同样出色，因为 Nginx 与 Nodejs 都是采用事件循环，非阻塞 I/O 体系，我们将用 Nginx 与传统 HTTP 服务的代表 Apache(多线程体系结构)进行数据上的对比来侧面证明 Nodejs 的性能[9,10]。

从图 4 可以看出，随着并发用户的增加，Apache 能及时响应的客户骤减，而 Nginx 减少的较为缓慢，Apache 每一次连接都会开辟一个新线程，当线程达到连接池的上限时便会出现以上症状，而 Nginx 采用的是事件循环机制则不会出现以上情况。

Nginx 与 Apache 并发时内存使用情况如图 5 所示，因为 Apache 这种模式开辟了更多的线程，开辟线程与线程之间的切换会消耗大量内存和 CPU 资源，显然面对大量请求同时来临时系统的资源会迅速耗尽，这也是引言中提到那些问题的根源，Nginx 和 Nodejs 采用事件机制，Nodejs 有效的利用了 Javascript 闭包和作用域，单线程的模式将内存的消耗降到最低。

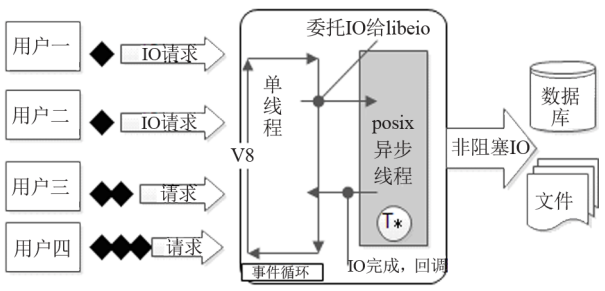

Figure 2. Chart: Node event-driven non-blocking I/O server model
图 2. Nodejs 事件驱动非阻塞 I/O 服务器模型图

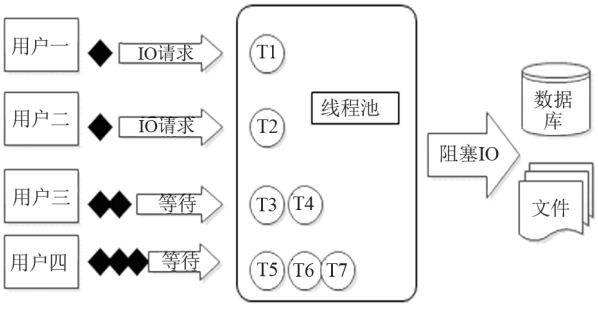

Figure 3. Chart: Multi-threaded blocking I/O server model
图 3. 多线程阻塞 I/O 的服务器模型图

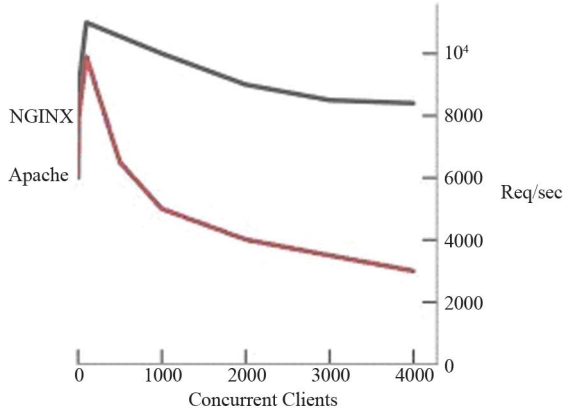

Figure 4. The relationship curve of concurrent clients and request-response for Nginx and Apache
图 4. Nginx 与 Apache 并发时请求对比图

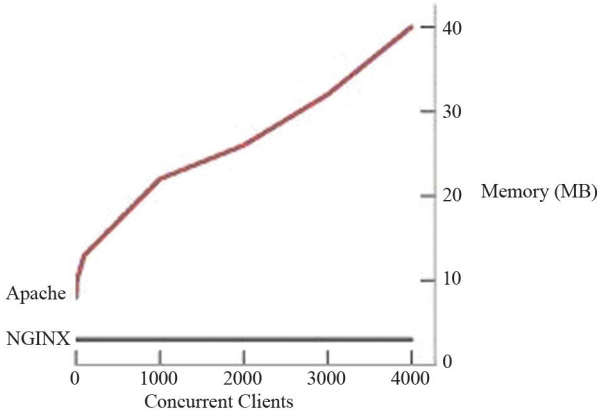

Figure 5. The relationship curve of concurrent clients and memory for Nginx and Apache
图 5. Nginx 与 Apache 并发时内存对比图

4. 总结

多线程服务器支持同步、阻塞 I/O 模型来提供了更简单的方式执行 I/O[11]，但是在面对高并发，大负荷时，多线程服务器会使用更多的线程，因为为了保证数据的安全性，每次请求将开辟一个线程，为了支持更多的线程和更多的线程上下文切换会消耗大量内存和 CPU 利用率，事件驱动非阻塞 I/O 服务器的体系结构顶层采用事件循环，底层再用内核级异步 I/O，连接没有和线程直接关联，这种模式需要主线程不停循环和少量的内核级线程执行 I/O，因为有更少的线程，因此更少的上下文切换，它使用更少的内存和 CPU 也少。从而很高效率利用硬件资源，在面对高并发，大数据量时可以表现更加出色，以上从理论与实际数据的引证证明了这一点。

5. 致谢

本章节为作者提供“致谢”的示例。首先感谢我的导师给予的热切指导，是她一步一步的引导才有这篇论文的总体设想。同时感谢同学在资料提供上给予的帮助，最后感谢文中引用的文献及著作的原创作者，正是他们的辛勤无私的奉献，才能使本次文章的中心表达更加流畅。

参考文献 (References)

[1] D. Herron. Node Web 开发[M]. 北京: 人民邮电出版社, 2012: 2-7.
[2] L. Bakerhamer. V8 JavaScript Engine, 2012. https://code.google.com/p/v8/
[3] 高原. 服务器端 Javascript 技术研究[J]. 信息与电脑, 2012, 1: 78-80.
[4] 郭家宝. Node.js 开发指南[M]. 北京: 人民邮电出版社, 2012: 21-32.
[5] 黄明恩. 初始 Nodejs 基于 GoogleV8 的 Javascript 运行环境[URL], 2013. http://www.cnblogs.com/CodeGuy/archive/2013/04/19/3030567.html
[6] 傅强, 陈宗斌. Nodejs 入门经典[M]. 北京: 人民邮电出版社, 2013: 33-34.
[7] M. Kiessling. The node beginner book. lulu.com, 2012: 23-25.
[8] 郑鹏, 曾平, 李蓉蓉. 计算机操作系统[M]. 武汉: 武汉大学出版社, 2009: 18-23.
[9] B. Erb. Concurrent programming for scalable web architectures. Ulm University, 2011.
[10] R. Dahl. Nodejs. 2010. http://nodejs.org/jsconf2010.pdf
[11] T. Hughes-Croucher, M. Wilson. Scalable server-side code with JavaScript. New York: O’Reilly Media Inc., 2012: 68-69.