NodeJS 的异步非阻塞 I/O 研究

Asynchronous and Non-blocking I/O of NodeJS

许会元　何利力　（浙江理工大学信息学院，浙江 杭州 310018）

摘　要

NodeJS 算得上是如今最火热的技术了，它是一个服务器端 JavaScript 执行环境，它将改变服务器应该如何工作的概念。它的目标是帮助程序员构建高度可伸缩的应用程序，编写能够处理数万条并发请求的连接代码。除了 Web 应用外，NodeJS 也被应用在许多方面，这些项目涉及到应用程序监控、媒体流、远程控制、桌面和移动应用等等。主要内容包括 NodeJS 的简单介绍以及对其核心特性异步 I/O 的研究。

关键词：web 前端，NodeJS，异步 I/O

Abstract

now NodeJS maybe the hottest technology.It is a server-side JavaScript execution environment and it will change the concept of how the server should work.It´s goal is to help programmers to build highly scalable applications,to write code which can handle tens of thousands of concurrent connection requests.The main content of this paper includes a brief description ofNodeJS and the research it´s core properties about asynchronous I/O.

Keywords:web front,NodeJS,asynchronousI/O

Jeff Atwood 在 2007 年就曾提出阿特伍德定律，大意就是：“任何可以用 JavaScript 来写的应用，最终都将用 JavaScript 来写”。目前看来，他的预言已经有所显现，NodeJS 的出现，把 JavaScript 带入了一个新的领域，打破了过去 JavaScript 只能在浏览器中运行的局面。NodeJS 的出现把 JavaScript 推到了一个从未到过的高度，不谈其他原因，仅仅因为好奇，就值得去关注和研究。

1　NodeJS 简介

NodeJS 是一个基于 Chrome V8 引擎的 JavaScript 执行平台，它可以快速构建网络服务及相关应用。借助事件驱动，异步非阻塞 I/O 等特性，对于数据密集型，I/O 密集型的应用场景有着优秀的处理能力。

1.1 NodeJS 的发展史

2009 年 3 月，Ryan Dahl 在其博客上宣布准备基于 V8 创建一个轻量级的 Web 服务器并提供一套库。2009 年 5 月，Ryan Dahl 在 GitHub 上发布了最初的版本。2009 年 12 月和 2010 年 4 月，两届 JSConf 大会都安排了 Node 的讲座。2010 年年底，Node 获得硅谷云计算服务商 Joyent 公司的资助，其创始人 Ryan Dahl 加入 Joyent 公司全职负责 Node 的发展。2011 年 7 月，Node 在微软的支持下发布了其 Windows 版本。2011 年 11 月，Node 超越 Ruby on Rails，成为 GitHub 上关注度最高的项目（随后被 Bootstrap 项目超越，目前仍居第二）。2012 年 1 月底，Ryan Dahl 在对 Node 架构设计满意的情况下，将掌门人的身份转交给 Isaac Z. Schlueter，自己转向一些研究项目。Isaac Z. Schlueter 是 Node 的包管理器 NPM 的作者，之后 Node 的版本发布和 bug 修复等工作由他接手。2013 年 7 月，发布的 Node 稳定版为 v0.10.13，非稳定版为 v0.11.4，NPM 的官方模块数达到 34 943 个，模块的周下载量为 1479 万次。

1.2 NodeJS 的结构

从本质上说 JavaScript 可以分为三部分：ECMAScript、DOM、BOM。从语言的角度讲，ECMAScript 才是真正的核心语言基础，DOM 和 BOM 是由宿主环境提供实现的。在 NodeJS 中我们提到的 JavaScript 实际上指的是脱离了浏览器宿主环境的 ECMAScript。NodeJS 是兼容 ECMAScript 的，并在其基础上借鉴 CommonJS 规范实现了一套非常易用的模块系统，完成了对服务端各项职能的封装，比如模块化、二进制、I/O 流、文件系统、套接字等等。对于 NodeJS，浏览器端 JavaScript 以及 ECMAScript 和一些标准之间的关系如图 1：

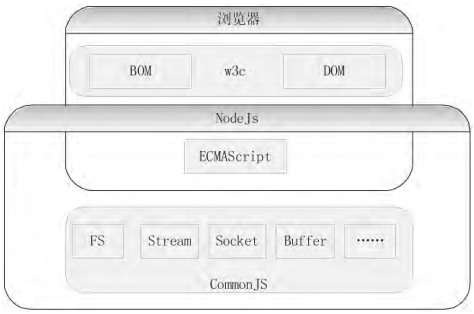

图 1　NodeJS 与浏览器、ECMA、CommonJS、W3C 关联图

ECMA 标准为核心，W3C 掌控着 BOM 和 DOM 标准，BOM、DOM、ECMAScript 共同组成了前端浏览器的执行环境。而后端同样基于 ECMA，按照 CommonJS 的规范对后端各方面内容进行实现，共同组成了 NodeJS 环境。

1.3 NodeJS 的主要特点

NodeJS 作为后端 JavaScript 的运行平台，所以并没有改变语言本身的任何特性，依旧是弱类型，基于作用域和原型链的。同时又将 Web 前端中大家熟悉的思想，比如事件机制等迁移到了服务端的环境中。相较于其他的服务端编程语言，NodeJS 的主要特点如下：

1）异步非阻塞 I/O。这是 NodeJS 的主要特性，是其处理高并发请求的秘诀所在。NodeJS 在底层构建了很多异步 I/O 的 API，我们可以很自然，很方便的调用这些 API 进行异步操作，每个调用之间无需等待，操作结束后通过回调进行数据处理。相对于同步 I/O，异步编程模型可以提升效率。

2)单线程。NodeJS 和浏览器端的 JavaScript 一样保持了单线程的特点。单线程的好处是无需考虑多线程下的状态同步，上下文切换，死锁，线程安全等方方面面的问题。

3)事件机制。事件的编程方式已经在前端得到了广泛的应用和肯定，因此 NodeJS 将浏览器中常见且成熟的事件引入后端，配合异步 I/O 操作，具有轻量级，松耦合，只关注事务点等优势。

4）跨平台。NodeJS 最开始只能在 Linux 运行，但是随着 NodeJS 的发展，微软投入了一个团队帮助其实现了 Windows 平台的兼容。在 v0.6.0 版本之后，NodeJS 已经可以直接运行在 Windows 平台上。

2 异步非阻塞 I/O

2.1 什么是异步和非阻塞

异步和非阻塞从意义上来讲实际上是一个意思，都能够达到并行处理的结果。但是为了区分，从操作系统内核对 I/O 的处理以及应用程序本身的执行来讲，异步和非阻塞实际上是两个问题。

阻塞与非阻塞指的的是操作系统内核对于 I/O 的处理方式只有两种：阻塞和非阻塞。异步和同步指的是应用程序的 API 调用是否是需要等待的。

2.2 非阻塞 I/O

操作系统对计算机进行了抽象，将所有输入输出设备抽象为文件。系统内核在进行 I/O 操作时，是通过文件描述符进行管理的。阻塞 I/O 指的是一个 I/O 调用必须等到系统内核中的所有操作结束，返回数据，才算完成。这个时候 CPU 将等待 I/O 操作，无法进行其它操作，会极大的降低 CPU 的利用率，因此操作系统内核一般都提供了非阻塞 I/O，即一个 I/O 操作调用后立即返回得到一个状态，CPU 继续执行后续作业。这样 CPU 得到了充分利用，性能会得到显著提升。但是因为 I/O 操作并没有结束，所以如果想要真正得到数据，对于非阻塞 I/O，需要一种方式来确认 I/O 的结束，并且获得最终数据。这种技术叫做“轮询”，也就是通过循环，重复判断 I/O 操作是否完成。epoll 方案是 Linux 下效率最高的 I/O 事件通知机制，其示意图如图 2。

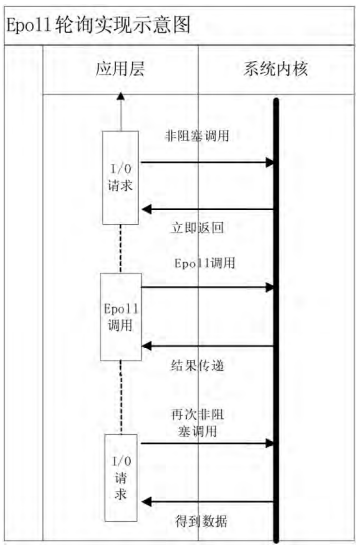

图 2 epoll 执行过程

这里的实现只是系统内核层面上对非阻塞 I/O 的实现，并没有涉及应用层的异步。

2.3 异步 I/O

通过操作系统的非阻塞 I/O 已经可以降低 CPU 的浪费，但是从 epoll 的执行过程来看，应用层上的方法在等待 I/O 的过程中仍然是休眠的。我们真正需要的是异步 I/O 调用后，无需轮询，接着向下执行，I/O 结束后通过信号或者回调将数据传递给用应用程序进行处理。实现这种异步 I/O 需要用到多线程的技术。在一个线程池中通过让部分线程进行 I/O 操作，运用轮询技术完成数据获取，让一个主线程进行计算处理，通过线程通信完成数据传递。工作模型如图 3。

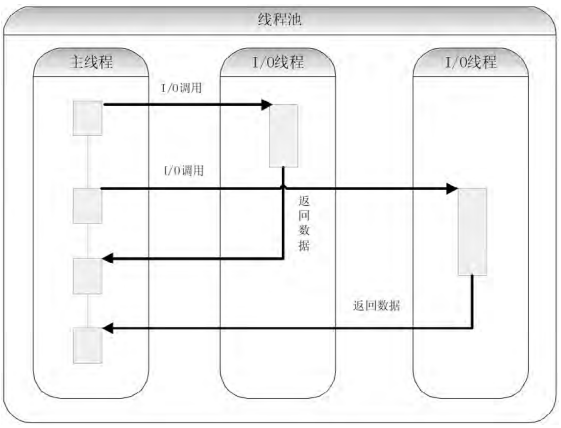

图 3 异步 I/O 线程池

通过这种方式就能够进行异步 I/O 的模拟，甚至每一个 I/O 线程的处理都无需关心是阻塞的还是非阻塞的，应用层的调用时无需等待的，即异步调用。

3 NodeJS 的异步非阻塞 I/O 运行原理

NodeJS 作为一个跨平台的环境，其底层异步非阻塞 I/O 的实现是通过两套方案解决的。首先在 Linux 系统下，v0.9.3 版本以后，是通过自行实现的线程池来完成的。在 Windows 平台下，是直接利用 Windows 下 IOCP 实现的，IOCP 的内部其实仍然是线程池的原理，只不过这些线程池有系统内核管理，无需我们自己手动实现管理。同时由于这种跨平台的差异性，NodeJS 构建了一个平台层架构，libuv。所有平台兼容性问题都由这一层来完成。结构图如图 4：

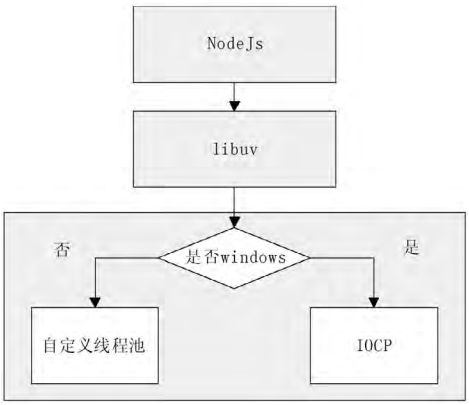

图 4 NodeJS 异步执行结构图

下面介绍 NodeJS 的整个异步 I/O 执行模型，主要包括三个方面，事件循环，观察者和请求对象。

3.1 事件循环

事件循环是整个异步模型的容器，其中包含着对观察者和请求对象的应用。在进程启动的时候，node 会创建一个无限循环，每执行一次循环体，我们称之为一个 tick。每一个 tick 的内部执行过程是：查看是否有事件待处理，如果有，取出事件。如果事件有相关回调函数，取出回调函数并执行。然后进入下一个循环，如果不再有事件处理，就退出进程。

3.2 观察者

在事件循环中，需要判断是否有事件需要处理，判断的方式就是加入观察者。不同的事件对应有不同的观察者，比如文件系统 I/O 处理有文件观察者，网络请求有网络 I/O 观察者。这是一种典型的发布/订阅模式，就像是前端浏览器事件系统的运行模式一样，我们通过代码在 DOM 元素上注册各种事件，算是一个种订阅，发布的方式是用户的交互，可能是单击鼠标，可能是按

下键盘的某个键，可能是放缩浏览器。后端当中我们同样可以订阅各种各样的事件，并将其绑定在对应的观察者上，发布的方式可能是某个网络请求，可能是文件读取，可能是数据库连接。事件循环就是不断地从观察者中取出订阅的事件进行处理。

3.3 请求对象

请求对象是真正和底层线程池交互的核心，是异步 I/O 过程中的重要中间产物。事件的相关信息，参数，结果，回调函数等等都在请求对象当中，所以事件循环从观察者当中得到的事件实体实际上就是请求对象。请求对象是 JavaScript 层面和 NodeJS 的底层交互的接口，每一个异步 I/O 操作都在 JavaScript 封装好请求对象交给底层线程池进行异步非阻塞的 I/O 处理，处理结束后再将请求对象交给 I/O 观察者由事件循环捕获。

3.4 整体流程

结合事件循环，观察者和请求对象，Windows 平台下 NodeJS 的异步 I/O 处理的整体流程如图 5：

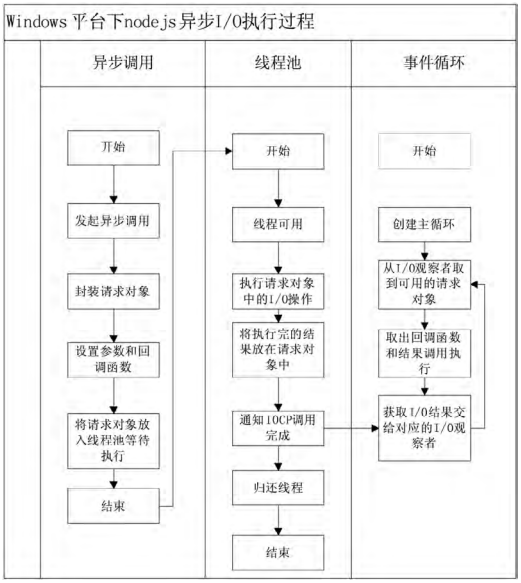

图 5 NodeJS 异步 I/O 整体流程图

应用层的每一次异步调用都会对请求对象进行封装，设置好参数和回调函数，然后将请求对象加入线程池中等待 I/O 执行，此时应用层处理结束，无需等待，可以直接向下执行代码，可能会继续发起另一个 I/O 请求。NodeJS 底层的线程池会依次处理请求对象中的 I/O 操作，并将直接结果放在请求对象中。Windows 下会在 IOCP 的协作性完成非阻塞的 I/O 操作并在完成时将请求对象交给 I/O 观察者。事件循环会在 NodeJS 启动时创建，不断的捕获 I/O 观察者中的请求对象，获取其中的执行结果和回调函数进行执行。以上就是 NodeJS 的整个异步非阻塞 I/O 的处理过程。

4 结束语

NodeJS 作为一个新兴技术，必然还存在着很多的不足。尤其是其异步操作的编程方式与传统同步编程习惯有很大的差异，让人一开始很难适应。同时异步编程还存在着嵌套过深的“恶魔金字塔”问题，异常捕获问题，无多线程问题，无法真正实现代码阻塞等问题。当然有问题，必然也会有着解决方案，比如基于事件模型的 EventProxy，比如基于 promise/deferred 的 Whenjs，比如像 async 这样的成熟的流程控制库等等。总之，NodeJS 在快速地发展，异步编程的方式也越来越受欢迎，在未来的开发中，其在 I/O 密集型场景下的优秀处理能力一定会为在构建快速可伸缩的高性能网络应用领域取得一席之地。

参考文献

[1]Tom Hughes-Croucher.Node.O´Reilly Media[M].2011
[2]Mike Cantelon,TJ Holowaychuk.Node.js in Action[M]. Manning.
[3]Mick Thompson.Getting Started with GEO, CouchDB, and Node.js. O´Reilly Media[M].2011
[4]Brett McLaughlin. What is Node. O´Reilly Media[M]. 2011-7
[5]David Herron. Node Web Development [M].Packt Publishing. 2011
[6]taobao ued. node.js 调研与服务性能测试[EB/OL].http://www.tbdata.org/archives/1285
[7]ibm developer.Node.js 究竟是什么[EB/OL].http://www.ibm.com/developerworks/cn/opensource/os-NodeJS
[8]朴灵.深入浅出 NodeJS[M].北京：人民邮电出版社，2013

[收稿日期：2014.10.8]

（上接第 126 页）

身是非稳态的，则强制分解成稳态后，可能会对 IMF 分量有很强的污染。例如图 6，IMF1 中 0.2s 处并不能完美提取 50Hz 分量。因此，若要对 EMD 方法完美使用，仍需进一步改进前期去除非稳态噪声的滤波方法。

3 结束语

EMD 算法提取的信号是各模态的平稳信号，只要有流量存在，必将有一模态代表纯净的涡街信号。因此，只需利用涡街信号与频率平方成正比这一关系，对各模态的幅值进行筛选，即可达到滤波目的。本文应用 MATLAB 软件对在线数据进行分析，结果表明，该信号处理方法具有较强的滤除噪声的能力，算法误差小于 0.3%。

参考文献

[1]谢宇怀，黄显元，张宏建.压电式涡街流量计噪声处理方法研究[J].机电工程，1999，12(5)：194-195
[2]Wade Matter, Method and System for Characterizing Pulsatile Flow in a Vortex Flowmeter[J],US Patent 6386046 B1, May. 14, 2002
[3]蒙建波，朱林章.自适应频率测量方法(AMF)在涡街流量计中的应用[J].自动化学报，1992，18(3)：362-366
[4]张涛，段瑞峰，孙宏军.基于双核技术的数字涡街流量计信号处理系统[J].化工自动化及仪表，2004，31(6)：71-74
[5]徐科军，汪枫.涡街流量计信号处理的软件方法[J].仪表技术与传感器，1995，5：22-25
[6]徐科军，汪安民.涡街流量计信号估计的自适应陷波方法[J].仪器仪表学报，2000，21(2)：222-224
[7]徐科军，汪安民.基于小波变换的涡街流量计信号处理方法[J].仪器仪表学报，2001，22(6)：636-639
[8]李斌，陈洁，宋磊.涡街流量计信号的阈值自适应处理方法及装置.中国，200410084746.1[P]
[9] 李斌，陈洁，张伟卿. 涡街流量计的信号处理方法及系统. 中国，200610029218.5[P]
[10] 李斌，陈洁，张伟卿，等. 一种双通道涡街流量计系统. 中国，200710042749.2[P]
[11]R.C.Baker, An Introductory Guide to Industrial Flow[M],Mechanical Engineering Publications, London, UK, 1996

[收稿日期：2014.11.25]