Node.js简介

第 1 章

Node.js，或者 Node，是一个可以让 JavaScript 运行在服务器端的平台。它可以让 JavaScript 脱离浏览器的束缚运行在一般的服务器环境下，就像运行 Python、Perl、PHP、Ruby 程序一样。你可以用 Node.js 轻松地进行服务器端应用开发，Python、Perl、PHP、Ruby 能做的事情 Node.js 几乎都能做，而且可以做得更好。

Node.js 是一个为实时Web（Real-time Web）应用开发而诞生的平台，它从诞生之初就充分考虑了在实时响应、超大规模数据要求下架构的可扩展性。这使得它摒弃了传统平台依靠多线程来实现高并发的设计思路，而采用了单线程、异步式I/O、事件驱动式的程序设计模型。这些特性不仅带来了巨大的性能提升，还减少了多线程程序设计的复杂性，进而提高了开发效率。

Node.js 最初是由 Ryan Dahl 发起的开源项目，后来被 Joyent 公司注意到。Joyent 公司将 Ryan Dahl 招入旗下，因此现在的 Node.js 由 Joyent 公司管理并维护。尽管它诞生的时间（2009 年）还不长，但它的周围已经形成了一个庞大的生态系统。Node.js 有着强大而灵活的包管理器（node package manager，npm），目前已经有上万个第三方模块，其中有网站开发框架，有 MySQL、PostgreSQL、MongoDB 数据库接口，有模板语言解析、CSS 生成工具、邮件、加密、图形、调试支持，甚至还有图形用户界面和操作系统 API工具。由 VMware 公司建立的云计算平台 Cloud Foundry 率先支持了 Node.js。2011年6月，微软宣布与 Joyent 公司合作，将 Node.js 移植到 Windows，同时 Windows Azure 云计算平台也支持 Node.js。Node.js 目前还处在迅速发展阶段，相信在不久的未来它一定会成为流行的Web应用开发平台。让我们从

现在开始，一同探索 Node.js 的美妙世界吧！

1.1 Node.js 是什么

Node.js 不是一种独立的语言，与 PHP、Python、Perl、Ruby 的“既是语言也是平台”不同。Node.js 也不是一个 JavaScript 框架，不同于 CakePHP、Django、Rails。Node.js 更不是浏览器端的库，不能与 jQuery、ExtJS 相提并论。Node.js 是一个让 JavaScript 运行在服务端的开发平台，它让 JavaScript 成为脚本语言世界的一等公民，在服务端堪与 PHP、Python、Perl、Ruby 平起平坐。

Node.js 是一个划时代的技术，它在原有的 Web 前端和后端技术的基础上总结并提炼出了许多新的概念和方法，堪称是十多年来 Web 开发经验的集大成者。Node.js 可以作为服务器向用户提供服务，与 PHP、Python、Ruby on Rails 相比，它跳过了 Apache、Nginx 等 HTTP 服务器，直接面向前端开发。Node.js 的许多设计理念与经典架构（如 LAMP）有着很大的不同，可提供强大的伸缩能力，以适应21世纪10年代以后规模越来越庞大的互联网环境。

Node.js 与 JavaScript

说起 JavaScript，不得不让人想到浏览器。传统意义上，JavaScript 是由 ECMAScript、文档对象模型（DOM）和浏览器对象模型（BOM）组成的，而 Mozilla 则指出 JavaScript 由 Core JavaScript 和 Client JavaScript 组成。之所以会有这种分歧，是因为 JavaScript 和浏览器

之间复杂的历史渊源，以及其命途多舛的发展历程所共同造成的，我们会在后面详述。我们可以认为，Node.js 中所谓的 JavaScript 只是 Core JavaScript，或者说是 ECMAScript 的一个实现，不包含 DOM、BOM 或者 Client JavaScript。这是因为 Node.js 不运行在浏览器中，所以不需要使用浏览器中的许多特性。

Node.js 是一个让 JavaScript 运行在浏览器之外的平台。它实现了诸如文件系统、模块、包、操作系统 API、网络通信等 Core JavaScript 没有或者不完善的功能。历史上将 JavaScript 移植到浏览器外的计划不止一个，但Node.js 是最出色的一个。随着 Node.js 的成功，各种浏览器外的 JavaScript 实现逐步兴起，因此产生了 CommonJS 规范。CommonJS 试图拟定一套完整的 JavaScript 规范，以弥补普通应用程序所需的 API，譬如文件系统访问、命令行、模块管理、函数库集成等功能。CommonJS 制定者希望众多服务端 JavaScript 实现遵循 CommonJS 规范，以便相互兼容和代码复用。Node.js 的部份实现遵循了CommonJS规范，但由于两者还都处于诞生之初的快速变化期，也会有不一致的地方。

Node.js 的 JavaScript 引擎是 V8，来自 Google Chrome 项目。V8 号称是目前世界上最快的 JavaScript 引擎，经历了数次引擎革命，它的 JIT（Just-in-time Compilation，即时编译）执行速度已经快到了接近本地代码的执行速度。Node.js 不运行在浏览器中，所以也就不存在 JavaScript 的浏览器兼容性问题，你可以放心地使用 JavaScript 语言的所有特性。

1.2 Node.js 能做什么

正如 JavaScript 为客户端而生，Node.js 为网络而生。Node.js 能做的远不止开发一个网站那么简单，使用 Node.js，你可以轻松地开发：

- 具有复杂逻辑的网站；
- 基于社交网络的大规模 Web 应用；
- Web Socket 服务器；
- TCP/UDP 套接字应用程序；
- 命令行工具；
- 交互式终端程序；
- 带有图形用户界面的本地应用程序；
- 单元测试工具；
- 客户端 JavaScript 编译器。

Node.js 内建了 HTTP 服务器支持，也就是说你可以轻而易举地实现一个网站和服务器的组合。这和 PHP、Perl 不一样，因为在使用 PHP 的时候，必须先搭建一个 Apache 之类的 HTTP 服务器，然后通过 HTTP 服务器的模块加载或 CGI 调用，才能将 PHP 脚本的执行结果呈现给用户。而当你使用 Node.js 时，不用额外搭建一个 HTTP 服务器，因为 Node.js 本身就内建了一个。这个服务器不仅可以用来调试代码，而且它本身就可以部署到产品环境，它

的性能足以满足要求。

Node.js 还可以部署到非网络应用的环境下，比如一个命令行工具。Node.js 还可以调用 C/C++ 的代码，这样可以充分利用已有的诸多函数库，也可以将对性能要求非常高的部分用 C/C++ 来实现。

1.3　异步式 I/O 与事件驱动

Node.js 最大的特点就是采用异步式 I/O 与事件驱动的架构设计。对于高并发的解决方案，传统的架构是多线程模型，也就是为每个业务逻辑提供一个系统线程，通过系统线程切换来弥补同步式 I/O 调用时的时间开销。Node.js 使用的是单线程模型，对于所有 I/O 都采用异步式的请求方式，避免了频繁的上下文切换。Node.js 在执行的过程中会维护一个事件队列，程序在执行时进入事件循环等待下一个事件到来，每个异步式 I/O 请求完成后会被推送到事件队列，等待程序进程进行处理。

例如，对于简单而常见的数据库查询操作，按照传统方式实现的代码如下：

```
res = db.query('SELECT * from some_table');
res.output();
```

以上代码在执行到第一行的时候，线程会阻塞，等待数据库返回查询结果，然后再继续处理。然而，由于数据库查询可能涉及磁盘读写和网络通信，其延时可能相当大（长达几个到几百毫秒，相比CPU的时钟差了好几个数量级），线程会在这里阻塞等待结果返回。对于

高并发的访问，一方面线程长期阻塞等待，另一方面为了应付新请求而不断增加线程，因此会浪费大量系统资源，同时线程的增多也会占用大量的 CPU 时间来处理内存上下文切换，而且还容易遭受低速连接攻击。

看看Node.js是如何解决这个问题的：

```
db.query('SELECT * from some_table', function(res) {
  res.output();
});
```

这段代码中 db.query 的第二个参数是一个函数，我们称为回调函数。进程在执行到 db.query 的时候，不会等待结果返回，而是直接继续执行后面的语句，直到进入事件循环。当数据库查询结果返回时，会将事件发送到事件队列，等到线程进入事件循环以后，才会调用之前的回调函数继续执行后面的逻辑。

Node.js 的异步机制是基于事件的，所有的磁盘 I/O、网络通信、数据库查询都以非阻塞的方式请求，返回的结果由事件循环来处理。图1-1 描述了这个机制。Node.js 进程在同一时刻只会处理一个事件，完成后立即进入事件循环检查并处理后面的事件。这样做的好处是，CPU 和内存在同一时间集中处理一件事，同时尽可能让耗时的 I/O 操作并行执行。对于低速连接攻击，Node.js 只是在事件队列中增加请求，等待操作系统的回应，因而不会有任何多线程开销，很大程度上可以提高 Web 应用的健壮性，防止恶意攻击。

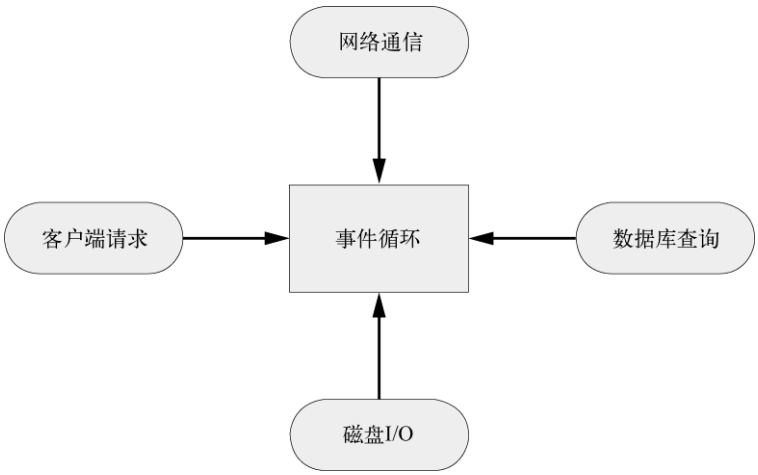

图1-1　事件循环

这种异步事件模式的弊端也是显而易见的，因为它不符合开发者的常规线性思路，往往需要把一个完整的逻辑拆分为一个个事件，增加了开发和调试难度。针对这个问题，Node.js 第三方模块提出了很多解决方案，我们会在第6章中详细讨论。

1.4　Node.js 的性能

1.4.1　Node.js 架构简介

Node.js 用异步式 I/O 和事件驱动代替多线程，带来了可观的性能提升。Node.js 除了使用 V8 作为JavaScript引擎以外，还使用了高效的 libev 和 libeio 库支持事件驱动和异步式 I/O。图1-2 是 Node.js 架构的示意图。

Node.js 的开发者在 libev 和 libeio 的基础上还抽象出了层 libuv。对于 POSIX[①]操作系统，

① POSIX（Portable Operating System Interface）是一套操作系统 API 规范。一般而言，遵守 POSIX 规范的操作系统指的是 UNIX、Linux、Mac OS X 等。

libuv 通过封装 libev 和 libeio 来利用 epoll 或 kqueue。而在 Windows 下，libuv 使用了 Windows 的 IOCP（Input/Output Completion Port，输入输出完成端口）机制，以在不同平台下实现同样的高性能。

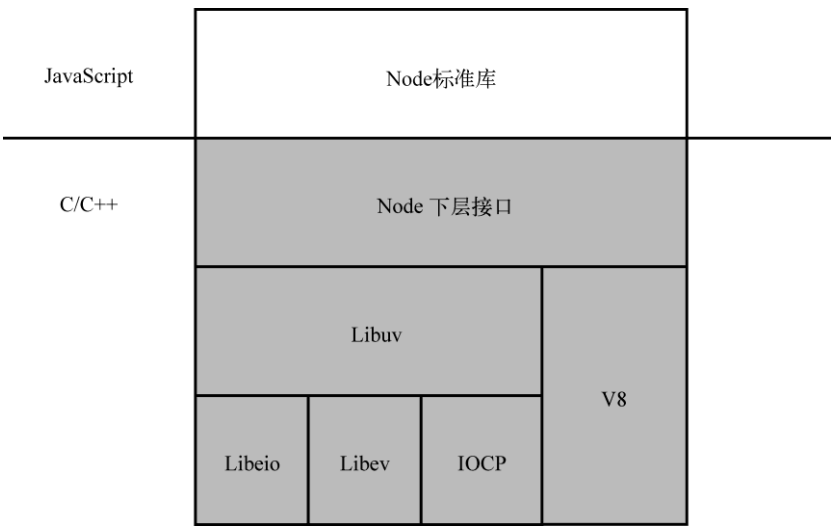

图1-2 Node.js 的架构

1.4.2 Node.js 与 PHP + Nginx

Snoopyxd 详细对比了 Node.js 与 PHP+Nginx 组合，结果显示在3000并发连接、30秒的测试下，输出“ hello world” 请求：

- PHP 每秒响应请求数为3624，平均每个请求响应时间为0.39秒；
- Node.js 每秒响应请求数为7677，平均每个请求响应时间为0.13秒。

而同样的测试，对MySQL查询操作：

- PHP 每秒响应请求数为1293，平均每个请求响应时间为0.82秒；
- Node.js 每秒响应请求数为2999，平均每个请求响应时间为0.33秒。

关于 Node.js 的性能优化及生产部署，我们会在第6章详细讨论。

1.5　JavaScript 简史

作为 Node.js 的基础，JavaScript 是一个完全为网络而诞生的语言。在今天看来，JavaScript 具有其他诸多语言不具备的优势，例如速度快、开销小、容易学习等，但在一开始它却并不是这样。多年以来，JavaScript 因为其低效和兼容性差而广受诟病，一直是一个被人嘲笑的“丑小鸭”，它在成熟之前经历了无数困难和坎坷，个中究竟，还要从它的诞生讲起。

1.5.1　Netscape 与 LiveScript

JavaScript 首次出现在1995年，正如现在的 Node.js 一样，当年 JavaScript 的诞生决不是偶然的。在1992年，一个叫 Nombas 的公司开发了“C减减”（C minus minus，Cmm）语言，后来改名为 ScriptEase。ScriptEase 最初的设计是将一种微型脚本语言与一个叫做 Espresso Page 的工具配合，使脚本能够在浏览器中运行，因此 ScriptEase 成为了第一个客户端脚本语言。

网景公司也想独立开发一种与 ScriptEase 相似的客户端脚本语言，Brendan Eich[①]接受了这一任务。起初这个语言的目标是为非专业的开发人员（如网站设计者），提供一个方便的

① Brendan Eich 被人称为 JavaScript 之父，他完全没想到自己当年无心设计的一个语言会成为今天最流行的网络脚本语言。

工具。大多数网站设计者没有任何编程背景，因此这个语言应该尽可能简单、易学，最终一个弱类型的动态解释语言 LiveWire 就此诞生。LiveWire 没过多久就改名为 LiveScript 了，直到现在，在一些古老的 Web 页面中还能看到这个名字。

1.5.2 Java 与 Javascript

在JavaScript 诞生之前，Java applet[①]曾经被热炒。之前 Sun 公司一直在不遗余力地推广 Java，宣称 Java applet 将会改变人们浏览网页的方式。然而市场并没有像 Sun 公司预期的那样好，这很大程度上是因为 Java applet 速度慢而且操作不便。网景公司的市场部门抓住了这个机遇，与 Sun 合作完成了 LiveScript 实现，并在网景的Navigator 2.0 发布前，将 LiveScript 更名为 JavaScript。网景公司为了取得 Sun 公司的支持，把 JavaScript 称为 Java applet 和 HTML 的补充工具，目的之一就是为了帮助开发者更好地操纵 Java applet。

Netscape 决不会预料到当年那个市场策略带来的副作用有多大。多年来，到处都有人混淆 Java 和 JavaScript 这两个不相干的语言。两者除了名字相似和历史渊源之外，几乎没有任何关系。现在看来，从论坛到邮件列表，从网站到图书馆，能把 Java 和 JavaScript 区分开的倒是少数[②]。图1-3 是百度知道上的“ Java 相关”分类。

---

① applet 的意思是“ 小程序”，它是 Java 的一个客户端组件，需要在“ 容器”中运行，通常浏览器会充当这个容器。

② Brendan Eich 为此抱憾不已，他后来在一个名为“ JavaScript at Ten Years”（JavaScript 这10年）的演讲稿中写道：“ Don’t let marketing name your language.”（不要为了营销决定语言名称）。

图1-3　百度知道上的“ Java 相关” 分类

1.5.3　微软的加入—— JScript

就在网景公司如日中天之时，微软的 Internet Explorer 3 随 Windows 95 OSR2 捆绑销售的策略堪称一颗重磅炸弹，轻松击败了强劲的对手——网景公司的Navigator。尽管这个做法致使微软后来声名狼藉（以及一系列的反垄断诉讼），但 Internet Explorer3 的成功却有目共睹，其成功不仅仅在于市场营销策略，也源于产品本身。Internet Explorer 3 是一个划时代产品，因为它也实现了类似于 JavaScript 的客户端语言—— JScript，除此之外还有微软的“ 老本行” VBScript。JScript 的诞生成为 JavaScript 发展的一个重要里程碑，标志了动态网页时代的全面到来。图1-4 是 Windows 95 上的 Internet Explorer 3。

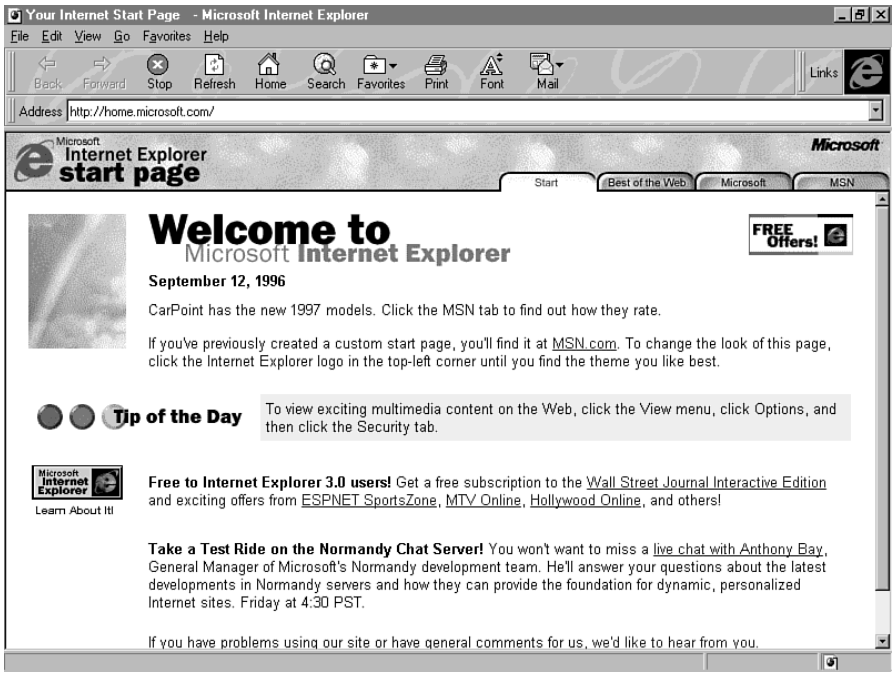

图1-4 Windows 95 上的 Internet Explorer 3

1.5.4 标准化—— ECMAScript

最初 JavaScript 并没有一个标准，因此在不同浏览器间有各种各样的兼容性的问题。Internet Explorer 占领市场以后这个问题变得更加尖锐，因此 JavaScript 的标准化势在必行。在1996年，JavaScript 标准由诸多软件厂商共同提交给ECMA（欧洲计算机制造商协会）。ECMA 通过了标准 ECMA-262，也就是 ECMAScript。紧接着国际标准化组织也采纳了 ECMAScript 标准（ISO-16262）。在接下来的几年里，浏览器开发者们就开始以 ECMAScript 作为规范来实现 JavaScript 解析引擎。

ECMAScript 诞生至今已经有了多个版本，最新的版本是在2009年12月发布的

ECMAScript 5，而到2012年为止，业界普遍支持的仍是 ECMAScript 3，只有新版的 Chrome 和 Firefox 实现了 ECMAScript 5。

ECMAScript 仅仅是一个标准，而不是一个语言的具体实现，而且这个标准不像 C++ 语言规范那样严格而详细。除了 JavaScript 之外，ActionScript[①]、QtScript[②]、WMLScript[③]也是 ECMAScript 的实现。

1.5.5　浏览器兼容性问题

尽管有 ECMAScript 作为 JavaScript 的语法和语言特性标准，但是关于 JavaScript 其他方面的规范还是不明确，同时不同浏览器又加入了各自特有的对象、函数。这也就是为什么这么多年来同样的 JavaScript 代码会在不同的浏览器中呈现出不同的效果，甚至在一个浏览器中可以执行，而在另一个浏览器中却不可以。

要注意的是，浏览器的兼容性问题并不只是由 JavaScript 的兼容性造成的，而是 DOM、BOM、CSS 解析等不同的行为导致的。万维网联盟（World Wide Web Consortium，W3C）针对这个问题提出了很多标准建议，目前已经几乎被所有厂商和社区接受，浏览器的兼容性问题迅速得到了改善。

1.5.6　引擎效率革命和 JavaScript 的未来

① ActionScript 最初是 Adobe 公司 Flash 的一部分，用于控制动画效果，现在已经被广泛应用在 Adobe 的各项产品中。

② QtScript 是 Qt 4.3.0 以后引入的专用脚本工具。

③ WMLScript 是 WAP 协议的一部分，用于扩展 WML（Wireless Markup Language）页面。

第一款 JavaScript 引擎是由 Brendan Eich 在网景的 Navigator 中开发的，它的名字叫做 SpiderMonkey。SpiderMonkey 在这之后还用作 Mozilla Firefox 1.0~3.0版本的引擎，而从 Firefox 3.5 开始换为 TraceMonkey，4.0版本以后又换为 JaegerMonkey。Google Chrome 的 JavaScript 引擎是 V8，同时 V8 也是 Node.js 的引擎。微软从 Internet Explorer 9 开始使用其新的 JavaScript 引擎 Chakra。[①]

过去，JavaScript 一直不被人重视，很大程度上是因为它效率不高——不仅速度慢，还占用大量内存。但如今JavaScript的效率却令人刮目相看。历史总是如此相似，正如没有 Shockley 发明晶体管就没有电子科技革命一样，如果没有2008年以来的 JavaScript 引擎革命，Node.js 也不会这么快诞生。

2008年 Mozilla Firefox 的一次改动，使 Firefox 3.0的 JavaScript 性能大幅提升，从而引发了 JavaScript 引擎之间的效率竞赛。紧接着 WebKit[②]开发团队宣告了 Safari 4 新的 JavaScript 引擎 SquirrelFish（后来改名 Nitro）可以大幅度提升脚本执行速度。Google Chrome 刚刚诞生就因它的 JavaScript 性能而备受称赞，但随着 WebKit 的 Squirrelfish Extreme 和 Mozilla 的 TraceMonkey 技术的出现，Chrome 的 JavaScript 引擎速度被超越了，于是 Chrome 2 发布时使用了更快速的 V8 引擎。V8 一出场就以其一骑绝尘般的速度打败了所有对手，一度成为

① 除此以外还有 KJS（用于 Konqueror）、Nitro（用于 Safari）、Carakan（用于Opera）等 JavaScript 引擎。

② WebKit 是苹果公司在设计 Safari 时开发的浏览器引擎，起源于 KHTML 和 KJS 项目的分支。WebKit 包含了一个网页引擎 WebCore 和一个脚本引擎 JavaScriptCore，但由于 JavaScript 引擎越来越独立，WebKit 逐渐成为了

JavaScript 引擎的速度之王。于是其他浏览器的开发者开始奋力追赶，与以往不同的是，Internet Explorer 也加入了这次竞赛，并取得了不俗的成绩。

时至今日，各个 JavaScript 引擎的效率已经不相上下，通过不同引擎根据不同测试基准测得的结果各有千秋。更有趣的是，JavaScript 的效率在不知不觉中已经超越了其他所有传统的脚本语言，并带动了解释器的革新运动。JavaScript 已经成为了当今速度最快的脚本语言之一，昔日“ 丑小鸭” 终于成了惊艳绝俗的“ 白天鹅”。

尽管如此，我们不能否认 JavaScript 还有很多不完美之处，譬如一些违反直觉的特性，这几乎成了 JavaScript 遭受批评和攻击的焦点。如今 JavaScript 还在继续发展，ECMAScript 6 也正在起草中，更有像 CoffeeScript 这样专门为了弥补 JavaScript 语言特性的不足而诞生的语言。Google 也专门针对客户端 JavaScript 不完美的地方推出了 Dart 语言。随着大规模的应用推广，我们有理由相信 JavaScript 会变得越来越好。

1.6　CommonJS

1.6.1　服务端 JavaScript 的重生

Node.js 并不是第一个尝试使 JavaScript 运行在浏览器之外的项目。追根溯源，在 JavaScript 诞生之初，网景公司就实现了服务端的 JavaScript，但由于需要支付一大笔授权费用才能使用，服务端 JavaScript 在当年并没有像客户端 JavaScript 一样流行开来。真正使大

WebCore 的代名词。

多数人见识到 JavaScript 在服务器开发威力的，是微软的 ASP。

2000年左右，也就是 ASP 蒸蒸日上的年代，很多开发者开始学习 JScript。然而 JScript 在当时并不是很受欢迎，一方面是早期的 JScript 和 JavaScript 兼容较差，另一方面微软大力推广的是 VBScript，而不是 JScript。随着后来 LAMP 的兴起，以及Web 2.0 时代的到来，Ajax 等一系列概念的提出，JavaScript 成了前端开发的代名词，同时服务端 JavaScript 也逐渐被人遗忘。

直至几年前，JavaScript 的种种优势才被重新提起，JavaScript 又具备了在服务端流行的条件，Node.js 应运而生。与此同时，RingoJS 也基于 Rhino 实现了类似的服务端 JavaScript 平台，还有像 CouchDB、MongoDB 等新型非关系型数据库也开始用 JavaScript 和 JSON 作为其数据操纵语言，基于 JavaScript 的服务端实现开始遍地开花。

1.6.2 CommonJS 规范与实现

正如当年为了统一 JavaScript 语言标准，人们制定了 ECMAScript 规范一样，如今为了统一 JavaScript 在浏览器之外的实现，CommonJS 诞生了。CommonJS 试图定义一套普通应用程序使用的API，从而填补 JavaScript 标准库过于简单的不足。CommonJS 的终极目标是制定一个像 C++ 标准库一样的规范，使得基于 CommonJS API 的应用程序可以在不同的环境下运行，就像用 C++ 编写的应用程序可以使用不同的编译器和运行时函数库一样。为了保持中立，CommonJS 不参与标准库实现，其实现交给像 Node.js 之类的项目来完成。图1-5

是 CommonJS 的各种实现。

图1-5　CommonJS 的实现

CommonJS 规范包括了模块(modules)、包(packages)、系统(system)、二进制(binary)、控制台（ console ）、编码（ encodings ）、文件系统（ filesystems ）、套接字（ sockets ）、单元测试（ unit testing ）等部分。目前大部分标准都在拟定和讨论之中，已经发布的标准有 Modules/1.0、Modules/1.1、Modules/1.1.1、Packages/1.0、System/1.0。

Node.js 是目前 CommonJS 规范最热门的一个实现，它基于 CommonJS 的 Modules/1.0 规范实现了 Node.js 的模块，同时随着 CommonJS 规范的更新，Node.js 也在不断跟进。由于目前 CommonJS 大部分规范还在起草阶段，Node.js 已经率先实现了一些功能，并将其反馈给

CommonJS 规范制定组织，但 Node.js 并不完全遵循 CommonJS 规范。这是所有规范制定者都会遇到的尴尬局面，因为规范的制定总是滞后于技术的发展。

1.7 参考资料

- Node.js: http://nodejs.org/。
- “再谈select、iocp、epoll、kqueue及各种I/O复用机制”: http://blog.csdn.net/shallwake/article/details/5265287。
- “巅峰对决：node.js和php性能测试”: http://snoopyxdy.blog.163.com/blog/static/60117440201183101319257/。
- “RingoJS vs. Node.js: Runtime Values”: http://hns.github.com/2010/09/21/benchmark. html。
- “Update on my Node.js Memory and GC Benchmark”: http://hns.github.com/2010/ 09/29/benchmark2.html。
- “JavaScript at Ten Years”: http://dl.acm.org/citation.cfm?id=1086382。
- QtScript： http://qt-project.org/doc/qt-4.8/qtscript.html。
- WebKit Open Source Project： http://www.webkit.org/。
- CommonJS API Specifications： http://www.commonjs.org/specs/。
- RingoJS： http://ringojs.org/。
- MongoDB : http://www.mongodb.org/。

- CouchDB : http://couchdb.apache.org/。
- Persevere : http://www.persvr.org/。
- 《JavaScript 语言精髓与编程实践》 周爱民著，电子工业出版社出版。
- 《JavaScript 高级程序设计（第3版）》 Nicholas C. Zakas 著，人民邮电出版社出版。
- 《JavaScript 权威指南（第5版）》 Flanagan David 著，机械工业出版社出版。