Tìm hiểu về NodeJS

**Lịch sử về NodeJS**

NodeJS được hoàn thiện vào năm 2009 bởi Ryan Dahl, khoảng 13 năm sau khi JavaScript ra đời. Phiên bản đầu của NodeJS chỉ hỗ trợ các hệ điều hành UNIX như Linux và MAC OS X. Sau đó Dahl tiếp tục xây dựng và duy trì công cụ này, sau đó được tài trợ thêm bởi Joyent.



Figure 1 Ryan Dahl Trong sự kiện JSConf

Dahl chỉ trích khả năng hạn chế của máy chủ web phổ biến lúc bấy giờ (2009) – Apache HTTP Server, để xử lí đồng thời nhiều kết nối ( khoảng 10,000 kết nối hoặc nhiều hơn thế nữa) trong cùng một thời điểm, và phong cách lập trình tuần tự.

Dahl đã giới thiệu NodeJs trong lễ khai mạc sự kiện JSConf Châu âu vào tháng 8 năm 2009. Trong sự kiện đó, Dahl giới thiệu rằng NodeJS vẫn chạy trên bộ Engine V8 JavaScript engine giống như JavaScript thông thường. Nhờ thế, NodeJS cũng có thể sử dụng Event Loop cũng như các Web Api do V8 cung cấp.

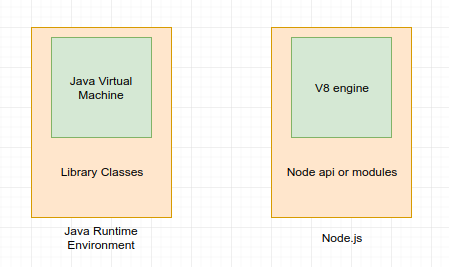


Figure 2 V8 Engine

Tháng 1 năm 2010, một bộ quản lí gói cài đặt của JavaScript được giới thiệu mang tên npm. Bộ quản lí gói cài đặt hỗ trợ nhà phát triển có thể dễ dàng trong việc sử dụng lại các đoạn code của người khác, dễ dàng hơn trong việc phát triển ứng dụng, sản phẩm. Bên cạnh đó sự có mặt của NPM phần nào hỗ trợ lập trình viên tối ưu dung lượng của ứng dụng, khả năng cập nhật ứng dụng.

Vào tháng 6 năm 2011, Microsoft và Joyent đã kết hợp xây dựng bản NodeJS đặc thù Windows, tháng 7 năm 2011 phiên bản NodeJS hỗ trợ hệ điều hành Windows chính thức được ra mắt.

Tháng 1 năm 2012, Dahl đã đề xuất trợ lí của mình , cũng là nhà sáng tạo ra công cụ quản lí gói NPM – Issac Schlueter trở thành quản lí chính của dự án NodeJs, sau đó 2 năm, vị trí này được đảm nhiệm bởi Timothy J. Fontaine.

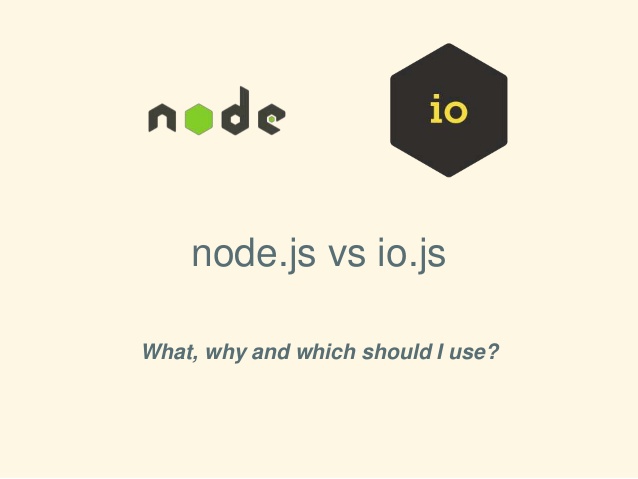


Figure 3 2 công cụ khác nhau trước khi cùng được sử dụng dưới tên NodeJs

Năm 2014, Fedor Indutny xây dựng IO.js , đây là một công cụ được xây dựng rẽ nhánh từ Nodejs, Mặc kệ các xung đột trong việc quản lí công nghệ đang xảy ra với NodeJs, IO.js được xây dựng tách biệt giữa ban quản lí và ban xây dựng công nghệ. Không giống như NodeJS, IO.js luôn được tác giả cập nhật liên tục để có thể phù hợp nhất với bộ compiler v8

Tháng 2 năm 2015, ý định thành lập Tổ chức Node.js trung lập đã được công bố. Tháng 6 năm 2015, hai cộng đồng Nodejs và IOjs đã đề xuất kết hợp và cùng phát triển dưới tên tổ chức NodeJS.

Vào tháng 9 năm 2015, Node.js v0.12 và io.js v3.3 đã được hợp nhất lại với nhau thành Node v4.0. [42] Việc hợp nhất này đã đưa các tính năng của V8 ES6 vào Node.js và chu kỳ phát hành hỗ trợ dài hạn. [43] Kể từ năm 2016, trang web io.js khuyến nghị các nhà phát triển chuyển trở lại Node.js và không có kế hoạch phát hành thêm io.js do hợp nhất.

**Sơ lược về NodeJS**

NodeJS được đưa vào thực tế thông qua cộng đồng người sử dụng JavaScript truyền thống. Sự phát triển này hướng tới việc đưa JavaScript từ một ngôn ngữ phải sử dụng trên các web browser đã có thể chạy độc lập trên ứng dụng nằm trên máy của bạn, thông qua đó, JavaScript có thể xử lí các tác vụ giống như các ngôn ngữ độc lập khác như Python, Java.

Cả JavaScript và Nodejs đều dùng bộ V8 JavaScript Runtime Engine để biên dịch code. Bộ Engine này hoạt động theo phong cách thông dịch ngôn ngữ ( interpreter). Việc thông dịch này được diễn ra tuần tự từ trên xuống dưới, từ trái sang phải, từng dòng một ngay trong thời gian chạy code thay vì đọc và chạy code trước khi đưa vào runtime như một số ngôn ngữ biên dịch khác ( Compiler ).

Mục đích của việc thông dịch là đưa các dòng code JavaScript được biên dịch thành mã máy ( machine code ). Mã máy là ngôn ngữ thấp nhất của ngôn ngữ lập trình, các đoạn mã này sẽ trực tiếp xử lí các thao tác với thanh ghi và đưa ra kết quả.

Là một JavaScript chạy theo hướng sự kiện không đồng bộ, Node.js được thiết kế để xây dựng các ứng dụng mạng có thể mở rộng. Trong ví dụ "hello world" sau, nhiều kết nối có thể được xử lý đồng thời. Sau mỗi kết nối, lệnh gọi lại được kích hoạt, nhưng nếu không có việc gì phải thực hiện, Node.js sẽ ngủ.



Có thể bạn đã đọc những câu này trước đây…

* Node.js là một trình chạy JavaScript được xây dựng trên công cụ JavaScript V8 của Chrome
* Node.js sử dụng mô hình I / O hướng sự kiện, không đồng bộ không chặn
* Node.js hoạt động trên một vòng lặp sự kiện luồng đơn ( single thread event loop )

… Và tự hỏi tất cả những điều này có nghĩa là gì. Hy vọng rằng đến cuối bài viết này, bạn sẽ hiểu rõ hơn về các thuật ngữ này cũng như về Node là gì, nó hoạt động như thế nào, tại sao và khi nào thì nên sử dụng nó.

Hãy bắt đầu bằng cách xem qua thuật ngữ.

**I/O (input/output)**

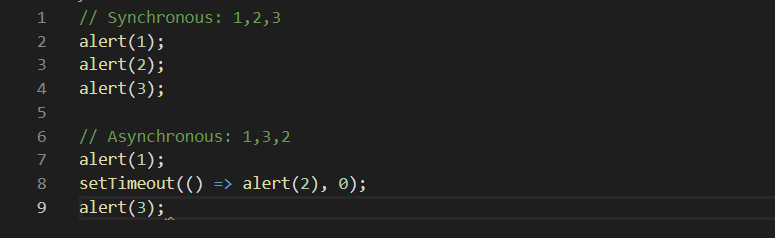
Viết tắt của input / output, I / O chủ yếu đề cập đến sự tương tác của chương trình với đĩa và mạng của hệ thống. Ví dụ về hoạt động I / O bao gồm đọc / ghi dữ liệu từ / vào đĩa, thực hiện các yêu cầu HTTP và nói chuyện với cơ sở dữ liệu. Chúng rất chậm so với việc truy cập bộ nhớ (RAM) hoặc thực hiện công việc trên CPU.

#### ****Synchronous vs Asynchronous****

Thực thi đồng bộ (Synchronous) (hoặc đồng bộ) thường đề cập đến mã thực thi theo trình tự. Trong lập trình đồng bộ, chương trình được thực hiện từng dòng, từng dòng một. Mỗi khi một hàm được gọi, việc thực thi chương trình sẽ đợi cho đến khi hàm đó trả về trước khi tiếp tục đến dòng mã tiếp theo.

Thực thi không đồng bộ (Asynchronous) (hoặc bất đồng bộ) đề cập đến việc thực thi không chạy theo trình tự mà nó xuất hiện trong mã. Trong lập trình không đồng bộ, chương trình không đợi tác vụ hoàn thành và có thể chuyển sang tác vụ tiếp theo.

Trong ví dụ sau, hoạt động đồng bộ hóa khiến alert() kích hoạt theo trình tự. Trong hoạt động không đồng bộ, trong khi alert(2) dường như thực thi thứ hai, nhưng không.



Hoạt động không đồng bộ thường liên quan đến I / O, mặc dù setTimeout là một ví dụ về một thứ không phải là I / O nhưng vẫn không đồng bộ. Nói chung, bất kỳ thứ gì liên quan đến tính toán đều là đồng bộ và bất kỳ thứ gì liên quan đến đầu vào / đầu ra / thời gian là không đồng bộ. Lý do khiến các hoạt động I / O được thực hiện không đồng bộ là chúng rất chậm và nếu không sẽ chặn tiếp tục thực thi mã.