MỤC LỤC

[MỞ ĐẦU 3](#_Toc519435588)

[CHƯƠNG 1 . TÌM HIỂU VỀ NOSQL DATABASE 4](#_Toc519435589)

[1.1 Thời đại của NoSQL 4](#_Toc519435590)

[1.2 NoSQL là gì ? 5](#_Toc519435591)

[1.2.1 Định nghĩa 6](#_Toc519435592)

[1.2.2 Một số thuật ngữ liên quan. 7](#_Toc519435593)

[1.3 Phân loại cơ sở dữ liệu NoSQL 9](#_Toc519435594)

[1.3.1 Key – Value Store 9](#_Toc519435595)

[1.3.2 Document Databases 11](#_Toc519435596)

[1.3.3 Column Family Stores 12](#_Toc519435597)

[1.3.4 Graph Database 14](#_Toc519435598)

[1.4 So Sánh NoSQL và SQL 15](#_Toc519435599)

[1.5 Tạo sao lại có nhiều NoSQL Database ? 19](#_Toc519435600)

[6. Lợi ích Của việc dùng NoSQL 20](#_Toc519435601)

[1.7 Nhược Điểm Của NoSQL 21](#_Toc519435602)

[1.8 Tổng kết 22](#_Toc519435603)

[CHƯƠNG 2 TÌM HIỂU VỀ NODE.JS 24](#_Toc519435604)

[2.1 Tổng Quan Về Node.JS 24](#_Toc519435605)

[2.1.1 Node.Js Là Gì ? 24](#_Toc519435606)

[2.1.2 Đặc điểm của Node.Js 26](#_Toc519435607)

[2.1.3 Với Node.js Bạn phải làm tất cả. 26](#_Toc519435608)

[2.1.4 Ai sử dụng node.js 26](#_Toc519435609)

[2.1.5 Các Thành phần quan trọng của Node.js 27](#_Toc519435610)

[2.1.6. Cộng Đồng Phát Triển 30](#_Toc519435611)

[2.2 Ưa Điểm Và Nhược Điểm 30](#_Toc519435612)

[2.2.1 Ưa Điểm 30](#_Toc519435613)

[**2.3 Installtion** 31](#_Toc519435614)

[2.3.1 Installing Node.js 31](#_Toc519435615)

[2.3.2 Installing New Modules 32](#_Toc519435616)

[2.4. Các Object Cơ Bản 32](#_Toc519435617)

[2.4.1 Global Objects (Đối tượng toàn cục) 32](#_Toc519435618)

[2.4.2 Sự kiện (Event). 44](#_Toc519435619)

[2.4.3.Luồng (Streams) 47](#_Toc519435620)

[2.4.4 File System 51](#_Toc519435621)

[2.4.5.HTTP 55](#_Toc519435622)

[CHƯƠNG 3. TÌM HIỂU CÁCH TẤN CÔNG VÀ NGĂN CHẶN INJECTION VÀO NOSQL QUA ỨNG DỤNG WEB NODE.JS 63](#_Toc519435623)

[3.1.TỔNG QUAN VỀ INJECTION NOSQL 63](#_Toc519435624)

[3.1.1 Cơ chế hoạt động 63](#_Toc519435625)

[3.1.2 Các loại tấn công 64](#_Toc519435626)

# MỞ ĐẦU

Chúng ta đang sống trong thời đại công nghệ 4.0 ,các mạng dịch vụ dữ liệu. cho phép mọi người chia sẻ hàng tỷ nội dung trên nền web. Khối lượng xử lý dữ liệu đang là rất lớn và nhanh chóng vượt qua giới hạn phần cứng cần để giải quyết . Tuy các hệ cơ sở dữ liệu quan hê ( RDBMS) vẫn được sự dụng để giải quyết lượng dữ liệu này nhưng nó tỏ ra có nhiều mối nguy hại cho những xử lý của nó.

Hầu hết các dữ liệu được lưa trữ phân tán trên nhiều máy chủ khác nhau. Các RDBMS hiện vẫn xử lý khá tốt nhiệm vụ lưa trữ dự liệu tập trung, nhưng chính vì tính tập trung này lại tạo ra khó khăn cho hệ thống muốn mở rộng. Việc người sử dụng thường xuyên tìm cách phi chuẩn hóa dữ liệu( bằng cách giảm số join cần thiết giữa các bảng.) Vô hình chung đã phá vỡ thiết kế cơ sở dữ liệu ban đầu . Hơn nữa, giao dịch phân tán RDBMS thường dễ sinh tượng nút cổ chai, việc xử lý chúng gặp tương đối nhiều khó khăn RDBMS hỗ trợ trực tiếp cho tình này. Rất nhiều giải pháp đưa ra, NoSQL giải pháp đó. NoSQL hệ sở liệu mới, giảm thiểu tối đa phép tính toán, tác vụ đọc-ghi liên quan kết hợp với xử lý theo lô đảm bảo yêu cầu xử lý liệu khối lượng lớn lớn. Hệ CSDL lưu trữ, xử lý từ lượng nhỏ đến hàng petabytes liệu với khả chịu tải tốt, khả chịu lỗi cao lại không đòi hỏi tài nguyên phần cứng cao.

Hiện nay, có khoảng 150 hệ sở liệu phân tán NoSQL (2) tiếp tục tăng. Một số hệ quản trị sở liệu đại diện cho NoSQL gồm có: Cassandra, MongoDB, SimpleDB, BigTable, CouchDB…

# CHƯƠNG 1 . TÌM HIỂU VỀ NOSQL DATABASE

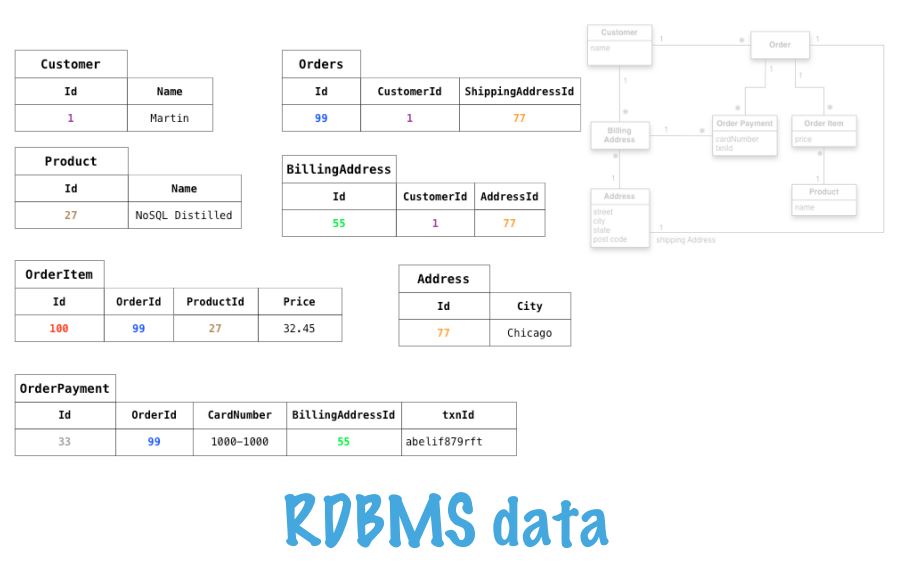
## 1.1 Thời Đại Của NoSQL

* Nhắc lại một chút, database(DB) là một cơ sở dữ liệu, gồm các bảng, hàng, cột.
* Còn MySQL, Microsoft SQL server, MongoDB,.. là hệ quản trị cơ sở dữ liệu (DBMS).
* SQL không phải là DB ,cũng không phải là DBMS . Nó là viết tắt của Structure Query ( Ngôn ngữ truy vấn cấu trúc). Ngôn ngữ này truy vấn trên nền một RDBMS( hệ quản trị CSDL quan hệ ) . Đây là thứ chúng ta đã được học trong môn “cơ sở dữ liệu “.
* Trong RDBMS , dự liệu được lưa thành nhiều bảng. Mỗi bảng sẽ có nhiều cột, nhiều hàng, ta sử dụng SQL để truy vấn như sau:
* **SELECT Name, Age FROM Students Where Score &gt;8**
* RDBMS (MySQL , Microsoft SQL Server , Oracle…) được sử dụng rất rộng rãi, trong hầu hết các ứng dụng , vì một số lý do sau :

1. Tính ACID ( Atomicity, Consistency, Isolation, Durability) của một transaction được đảo bảo.
2. Với databbase chuẩn 3, dữ liệu được đảm bảo tính đồng nhất và toàn vẹn ( Consistency)
3. Có rất nhiều driver cho mọi ngôn ngữ : Java , C# , PHP.
4. Số lượng lập trình viên biết dùng SQL rất nhiều

Tuy nhiên, RDBMS vẫn còn một số khuyết điểm :

* Việc mapping giữa các bảng trong database với các object trong code khá rắc rối và rất phức tạp. ( Mặc dù một số ORM như Entity Framework, Hibernate đã đơn giản hóa chuyện này ).
* Performance sẽ rất chậm khi join nhiều bảng để lấy dữ liệu ( Đó là lý do ta sử dụng “ giảm chuẩn “ để tăng hiệu suất cho RDBMS)
* Việc thay đổi cấu trúc dữ liệu ( Thêm/ xóa bảng) rất mệt moit , kéo theo vô số thay đổi trên code.
* Không làm việc được với dữ liệu không có cấu trúc ( un-structure).
* RDBMS được thiết kế để chạy trên máy chủ. Khi muốn mở rộng , nó khó chạy trên được nhiều máy ( clustering)



Dữ liệu được lưa thành nhiều bảng trong RDBMS , khi query ta phải join lại rất khó khăn

Do đó để giải quyết được những khuyết điểm của RDBMS , thế hệ database tiếp theo ra đời là Nosql Database.

* Đây là thế hệ database không rằng buộc, phân tán , mã nguồn mở , có khả năng mở rộng theo chiều ngang, có thể lưa trữ và xử lý từ một lượng rất nhỏ cho đến hàng petabytes dữ liệu trong hệ thông có độ chịu tải và chịu lỗi cao với những đòi hỏi về tài nguyên phần cứng thấp.
* Đặc điểm nhận dạng : Có schema-free , hộ trợ mở rộng dễ dàng, API đơn giản , eventual consistency ( Nhất quán cuối) và/hoặc transactions hạn chế trên các thành phần dữ liệu đơn lẻ, không giới hạn không gian dữ liệu,…

NoSQL storage đặc biệt phổ dụng cho phép người dùng tạo hàng tỷ nội dụng trên web. Do đó , dữ liệu lớn rất nhanh vượt qua giới hạn phần cứng và cần phải giải quyết bằng bài toán phân tán.

## 1.2 NoSQL Là Gì ?

### 1.2.1 Định Nghĩa

NoSQL là một xu hướng cơ sở dữ liệu mà không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mền. NoSQL có nghĩa là No-Relational (NoRel) – Không rằng buộc. Tuy nhiên thuật ngữ đó ít phổ dụng hơn và ngày ngày người ta thường dịch NoSQL thành Not Only SQL – Không chỉ SQL .

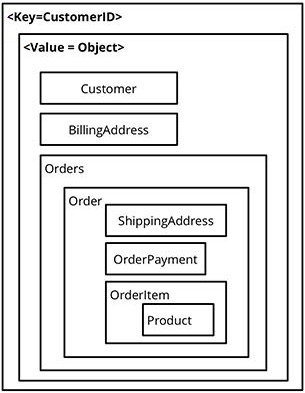
Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên chung cho các lightweight open source relational database( cơ sở dữ liệu quan hệ nguồn mở nhỏ ) nhưng không sử dụng SQL cho truy vấn

Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL trong một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc).

Là một công nghê cơ sở dữ liệu được thiết kế hộ trợ các yêu cầu của các ứng dụng điện toán đám mây và kiến trúc vượt qua quy mô, hiệu quả , mô hình dữ liệu, và các giới hạn phân phối dữ liệu của cơ sở dữ liệu quan hệ (RDBMS)

Công nghệ NoSQL ban đầu được tạo ra và được sử dụng bới các nhà lãnh đạo trên Internet như Facebook, Google ,Amazon, và những người yêu cầu hệ thống quản lý cơ sở dữ liệu có thể viết và đọc dữ liệu bất cứ nơi nào trên thế giới, trong khi mở rộng quy mô và cung cấp hiệu suất trên toàn bộ dữ liệu lớn và hằng triệu người sử dụng

Dữ liệu được lưa dưới dạng object . mặc dù bị trùng lặp nhưng truy vấn rất nhanh và đơn giản



### 1.2.2 Một Số Thuật Ngữ Liên Quan.

Relational (Rằng buộc) thuật ngữ sử dụng đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBMs) sử dụng mô hình khóa gồm 2 loại khóa: khóa chính và khóa phụ( primary key + foreign key) để rằng buộc dữ liệu nhàm thể hiện tính nhất quán dữ liệu từ các bẳng khác nhau.

Non- Relational (Không rằng buộc ) là khái niệm không sử dụng các rằng buộc dữ liệu cho nhất quán dữ liệu ở NoSQL database.

High Availability (Tính sẵn sàng) : Do chấp nhân sự trùng lặp trong lưa trữ nên nếu một node( commodity machine) nào đó bị chết cũng không ảnh hưởng tới toàn bộ hệ thống.

High Scalability ( khả năng mở rộng ) : Gần như không có một giới hạn cho dữ liệu và người dùng trên hệ thống.

Eventual consistency (Nhất quán cuối) : Tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khác sau mỗi phép write. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thơi gian ( không phải ngay tức khắc), thay đổi sẽ di đến mọi điểm trong hệ thống, tức là cuối cùng ( eventually) dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán

Vertical scalable ( khả năng mở rộng theo chiều dọc ): Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưa trữ và xử lý bằng việc cải tiến phần mền và cải thiện phần cứng trên một máy tính đơn lẻ được gọi là khả năng mở rộng chiều dọc. Ví dụ tăng cường CPUs, cải thiện đĩa cứng , bộ nhớ trong một máy tính,... cho DBMs nằm trong phạm trù này. Khả năng mở rộng chiều dọc còn có một thuật ngữ khác scale up.

Horizontal scalable (khả năng mở rộng chiều ngang) : Khi dữ liệu lớn về lượng , phương pháp tăng cường khả năng lưa trữ và xử lý dùng nhiều máy tính phân tán . Phân tán dữ liệu được hộ trợ bởi các phần mền tức cơ sở dữ liệu.

Distributed Data ( phân tán dữ liệu ): Mô hình lưa trữ phân tán các file hoặc dữ liệu ra nhiều máy tính khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mền .

Deployment Flexibility ( Triển khai linh hoạt ): Việc bổ sung thêm/loại bỏ các node , hệ thống sẽ tự động nhận biết để lưa trữ mà không cần phải can thiệp bằng tay. Hệ thống cũng không đòi hỏi cấu hình mạnh , đồng nhất .

Durability ( Lưa trữ tốt) : Dữ liệu có thể tồn tại trong bộ nhớ máy tính nhưng đồng thời cũng được lưa trữ lại đĩa cứng.

- Trong khi giá thành phần cứng ngày càng giảm. tốc độ xử lý, bộ nhớ ngày tăng thì horizontal scalable là một sự lựa chọn đúng đắn . Hằng trăm máy tính nhỏ được chập lại tạo thành một hệ thống tính toán mạnh hơn nhiều sơ với vi xử lý RISC truyền thống đơn lẻ. Mô hình này tiếp tục được hộ trợ bới các công nghệ kết nối Myrinet và InfiniBand. Từ đó chúng ta có thể quản lý , bảo trì từ xa, xây dụng batch procession( Xử lý đồng loạt tập lệnh ) tốt hơn. Do những đòi hỏi về tấc độ xử lý I/O cao , lượng cực lớn dữ liệu …scale horizontally sẽ thúc đẩy các công nghệ lưa trữ mới phát triển giống như object storage devices (OSD)

## 1.3 Phân loại cơ sở dữ liệu NoSQL

Tính đến thời điểm hiện tại thì có hơn 255 cơ sở dữ liệu NoSQL (<http://nosql-database.org/>) . Dựa theo đặc điểm và thuộc tính của một số loại cơ sở dữ liệu NoSQL có thể chưa ra chúng làm bốn loại. Thực chất thì việc chia chúng ra làm bốn 4 loại như vậy chỉ mang tính chất tương đối, dựa trên những đặc điểm rất khác biệt vì trên tổng thể , chúng vẫn mang tính chất khá giống nhau

### 1.3.1 Key – Value Store

* Key –value stores là kiểu lưa trữ đơn giản nhất trong các loại cơ sở dữ liệu NoSQL đồng thời nó cũng là kiểu lưa trữ cho tất các hệ quản trị cơ sở dữ liệu NoSQL. Thông thường các các hệ quản trị cơ sở dữ liệu Key-value lưa trữ dữ liệu dưới dạng key( là một chuỗi duy nhât) liên kết với value có thể ở dạng chuỗi văn bản đơn giản hoặc các tập, danh sách dữ liệu phức tạp hơn. Quá trình tìm kiếm dữ liệu thường sẽ được thực hiện thông qua key, điều này đẫn đến thiếu độ chính xác .

|  |  |
| --- | --- |
| Car | |
| Key | Attributes |
| 1 | Make: Honda  Model: SH  Color: Blue  Year: 2018 |
| 2 | Make: Yamaha  Model: Exciter  Color : Green  Year: 2017 |

Key – Value Store

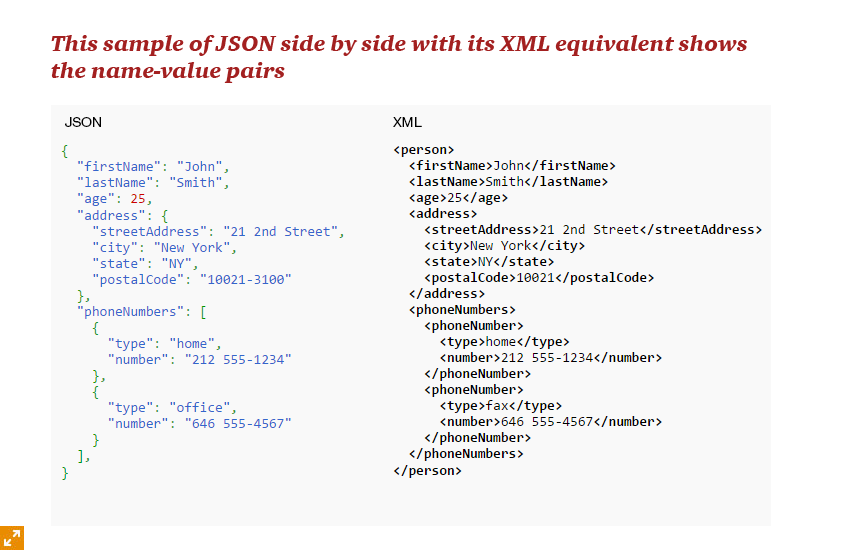
* Các API được cung cấp cho việc truy vấn dữ liệu của các cơ sở dữ liệu NoSQL thường cũng rất đơn giản , về cơ bản hầu hết các cơ sở dữ liệu NoSQL sẽ có các API sau:

**Void Put(string key , byte[] data); byte[] Get(string key) ; void Remove(string key);**

* Tuy nhiên việc triên khai thực tế của các cơ sở dữ liệu Key –Value thường rất phức tập do cấu trúc lưa trữ quá đơn giản. Vi dụ để lưa trữ thông tin một người:
* “ 1234” => “ BaLam, [nguyenbalam190@gmail.com](mailto:nguyenbalam190@gmail.com)”
* “1235” => “ Trung Nghĩa , [trungnghia@gmail.com](mailto:trungnghia@gmail.com)”
* “1236” => “ Văn Thao , [vanthao@gmail.com](mailto:vanthao@gmail.com), Tp,HCM”
* “1237” => “ Văn Thắng”
* Với kiểu lưa trữ này, ta sẽ rất dễ dàng và nhanh chóng truy xuất được thông tin của một người thông qua key, nhưng không hề đơn giản cho việc xử lý những dữ liệu phức tạp. Dễ thấy rằng hệ quản trị cơ sở dữ liệu Key-Value là đơn giản hóa việc lưa trữ dữ liệu , nghĩa là không cần quan tâm tới nội dung là gì .
* Nói cách khác , chúng ta lưa trữ thông tin mà không cần xác điịnh lược đồ. Điều này làm cho phương pháp lưa trữ dữ liệu với Key-Value trở nên đơn giản hơn rất nhiều trong việc xây dựng cũng như khả năng mở rộng là cự kì linh động và hiệu suất cho các thao tác truy vấn dữ liệu cũng nhanh.
* Với sự đơn giản của cách lưa trữ Key-Value làm cho các cơ sở dữ liệu loại này rất phù hợp với các ứng dụng cần truy xuất nhanh chóng và khả năng mở rộng cao, chẳng hạn như quản lý các phiên giao dịch (session) hoặc quản lý các thông tin về giỏ hàng vì trong trường hợp này biết được các ID của phiên giao dịch hoặc ID của khách hàng là rất cần thiết. Hay việc quản lý thông tin của sản phẩm bảo gồm các thông tin liên quan , đánh giá… sẽ được trữ dưới dạng key là mã sản phẩm chẳng hạn và value là các thông tin còn lại của sản phẩm cần lưa trữ. Điều này cho phép truy xuất được tất cả các thông tin về một sản phẩm chỉ thông qua mã sản phẩm cực kì nhanh.
* Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này: Aerospike, CouchDB, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB

### 1.3.2 Document Databases

Cơ sở dữ liệu Document được thiết kế để quản lý và lưa trữ dữ liệu ở dạng document. Nhưng document này được mã hóa về các dạng chuẩn như XML , JSON ( Javascript Option Notation) hay BSON( Binary JSON) . Khác với kiểu lưa trữ dạng Key- Value , giá trị của cột tring các cơ sở dữ liệu document chứa dữ liệu bán cấu trúc ( Semi-Structured Data) , đặc biệt là cặp thuộc tính name(key )- value . Một cột có thế chưa hàng trăm các thuộc tính như vậy, số lượng , loại thuộc tính được lưa lại có thể khác nhau giữa các dòng. Một điểm khác nữa so với kiểu lưa trữ dữ liệu Key-Value đơn giản là cả key và value đều có thể tìm kiếm trong cơ sở dữ liệu Document.



* Cơ sở dữ liệu Document phù hợp cho việc lưa trữ và quản lý dữ liệu có kích thước lớn như tài liệu văn bản, tin nhắn, cũng như biểu diễn một thực thể cơ sở dữ liệu là Product hay Customer
* Các cơ sở dữ liệu tiêu biểu cho cơ sở dữ liệu Document : CouchDB (JSON) , MongoDB (BSON) ,… đều là mã nguồn mở ( open source ), hướng document( document oriented) và có lược đồ tự do ( shema free).
* Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này : Apache Jackrabbit, CouchDB, IBM Lotus Notes Storage Format (NSF), MongoDB, Terrastore, ThruDB, OrientDB, RavenDB,...

MongoDB

* API : BSON
* Protocol : C
* Query Method : Dynamic object – based language & MapReduce,
* Replication : Master Slave & Auto –Sharding
* Writen in: C++.
* Concurrency : Update in place . Misc : Indexing, GridFS,

CouchDB

* API : JSON
* Protocol :REST
* Query Method :MapReduce of JavaScript Funcs.
* Replication : Master Master
* Writen in:Erlang.
* Concurrency : MVCC , Misc.

### 1.3.3 Column Family Stores

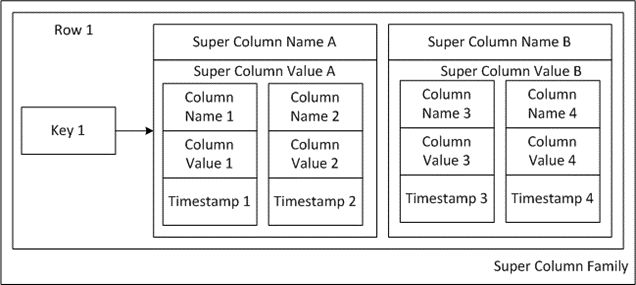
- Column Family được biết đến rộng rãi nhất qua sự triển khai BigTable của Google , nhìn bề ngoài , chúng khá giống với cơ sở dữ liệu quan hệ nhưng thực tế là hoàn toàn khác . Một số sự khác biệt dễ thấy nhất là việc lưa trữ dữ liệu theo dòng đối với các hệ quản trị cơ sở dữ liệu quan hệ với việc lưa trữ theo cột của các hệ quản trị cơ sở dữ liệu Column Family . Và các tư tưởng của cả hai loại cơ sở dữ liệu này cùng hoàn toàn khác nhau. Chúng ta không thể áp dụng cùng một giải pháp mà chúng ta đã sử dụng trong cơ sở dữ liệu quan hệ vào cơ sở dữ liệu Conlumn Family. Bởi vì cơ sở dữ liệu Column Family là các cơ sở dữ liệu phi quan hệ.

Với các cơ sở dữ liệu Column Family , chúng ta cần quan tâm đến các khái niệm sau :

Column Family ( Họ cột) : Một column family là cách thức dữ liệu được lưa trữ trên đĩa. Tất cả dữ liệu trong một cột sẽ được lưa trên cùng một file. Một column family có thể chứa super column hoặc column.

Super column ( Siêu cột) : Một super column có thể được dùng như một dictionnary ( kiểu từ điển ) . Nó là một column có thể chứa những column khác ( mà không phải là super column ).

-Column và super column trong column family database dùng thay thế nhau, có nghĩa là chúng sẽ là 0 byte nếu chúng không có chứa dữ liệu . không giống như một bảng, thứ duy nhất chúng ta cần xác đinh trong column farmily database tên cột và các tùy chọn chính ( không có lược đồ cố định).



* Cơ sở dữ liệu Column Family được thế kế để chạy trên một số lượng lớn các máy , và lưa trữ một lượng dữ liệu cực lớn. chúng ta không thể lưa trữ một lượng dữ liệu lớn dữ liệu trong cơ sở dữ liệu quan hệ vì chắc chắn chúng sẽ nhanh chóng bị sụp đổ hoặc là chết rất nhanh về kích thước của dữ liệu và những truy vấn đó được các cơ sở dữ liệu Column Family xử lý một cách dễ dàng. Các cơ sở dữ liệu Column Family loại bỏ các khái niệm trừa tượng , những thứ làm cho nó cứng nhắt khi chạy trên một cụm máy.

Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này :Hadoop/HBase – Apache, BigTable – Google, Cassandra - Facebook/Apache, Hypertable - Zvents Inc/Baidu, Cloudera, SciDB, Mnesia, Tablets,…

**Hadoop / Hbase**

* API :Java / any write
* Protocol : any write call
* Query Method : MapReduce Java /Any exec
* Replication : HDFS Replication
* Writen in: Java

Cassandra

Massively scable , partitioned row store ,architecture , linear scale performance , no single points of failure , read/write support across multiple data centers & cloud availability zones

API /Query Method : CQL and Thrift,

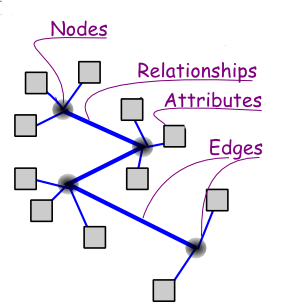
Replication : peer –to-peer

Written in : java

Concurrency : tunable consistency

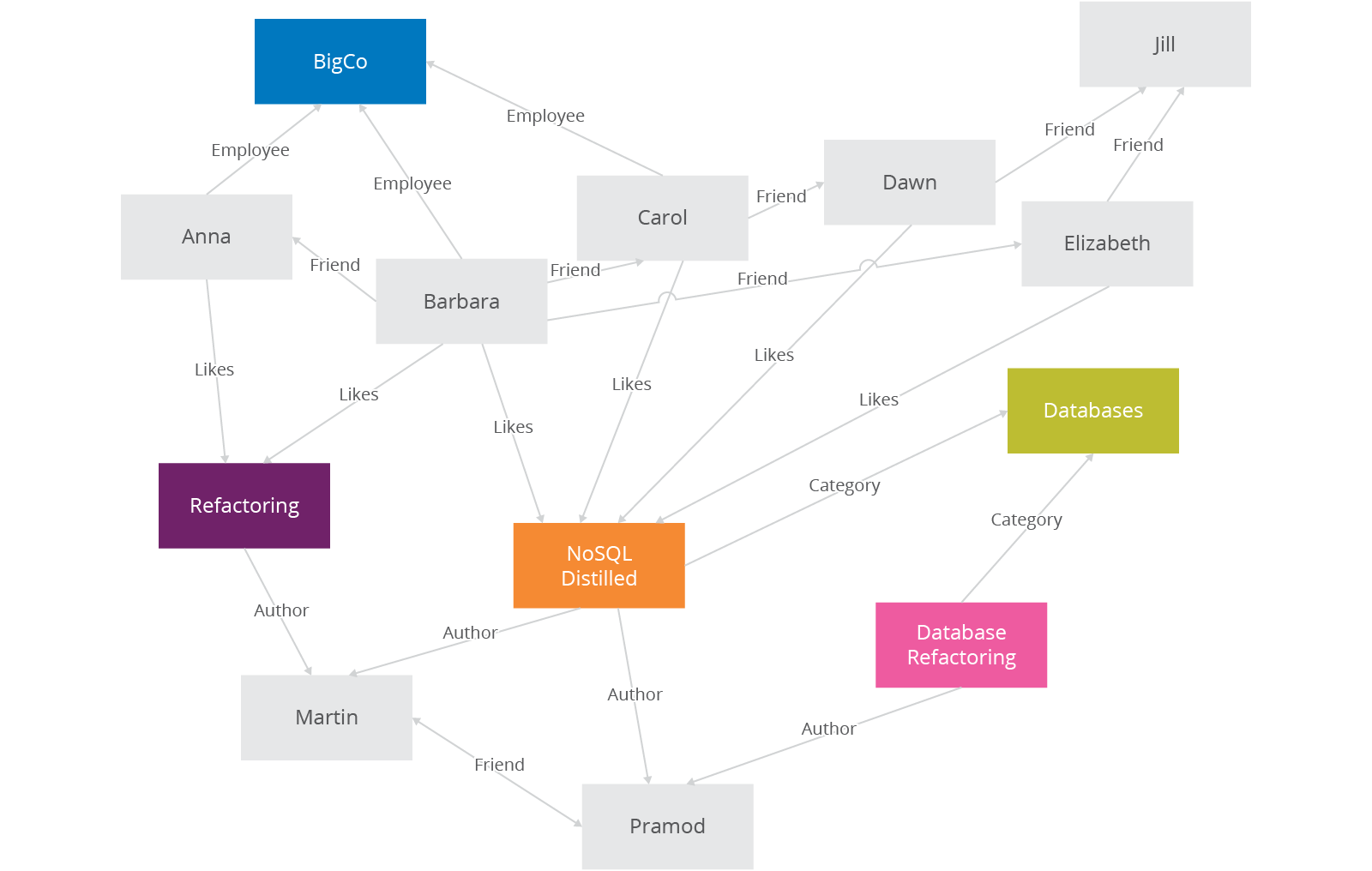
### 1.3.4 Graph Database

Cơ sở dữ liệu đồ thị là một cơ sở dữ liệu dạng đồ thị sử dụng các cấu trúc đồ thị với các nút ( nodes) các mối quan hệ ( relationships) , các thuộc tính (properties) để mô tả và lưa trữ dữ liệu



Chúng ta có thể thực hiện những truy vấn phưc tạp hơn như lọc trên các thuộc tính quan hệ , xem xét trọng số của người đó..

Graph Database thường được sử dụng để giải quyết các vấn đề về mạng. Trong thực tế , hầu hết các trang web mạng xã hội đều sử dụng một số hình thức của garph database để làm những việc mà chúng ta đã biết như :kết bạn, bạn của bạn.



Một vấn đề đối với việc mở rộng Graph Database là rất khó khăn để tìm thấy một đồ thị con độc lập, có nghĩ là rất khó khăn để ta phân tán Graph Database thành nhiều mảnh . có rất nhiều nỗ lực nghiên cứa cho việc này nhưng vẫn chưa có bất kì giải pháp nào đáng tin cậy được đưa ra .

Một số sản phẩm tiêu biểu của Graph database là : Neo4j , Sones , AllegroGraph , Core Data, Dex, FlockDB, InfoGrid, OpenLink Virtuoso…

## 1.4 So Sánh NoSQL và SQL

Tóm lại , NoSQL database chỉ là một kiểu database có các lưa trữ , truy vấn dữ liệu hoàn toàn khác so với RDBMS và SQL.

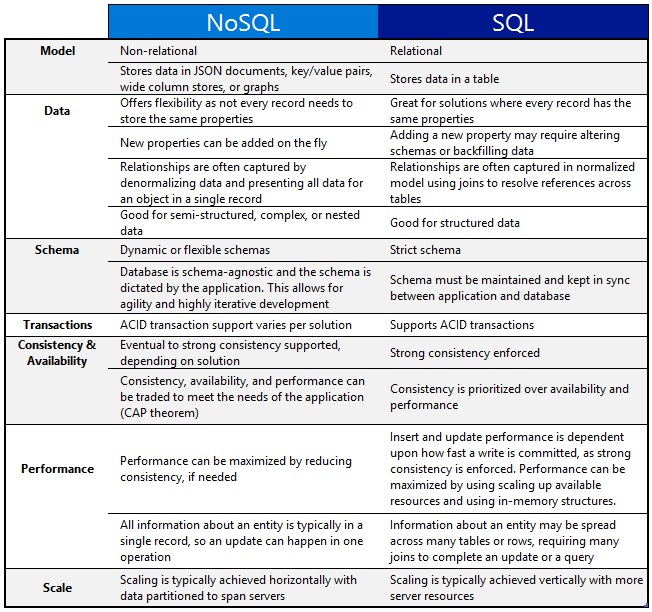
NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng (Scalability ). Chính những ưa điểm trên mà NoSQL được sử dụng nhiều trong các dự án Big Data, các dự án Real-time, số lượng dữ liệu nhiều.

Liệu NoSQL có thay thế được hoàn toàn RDBMS và NoSQL được không ? Câu trả lời là KHÔNG. Trong tương lai, RDBMS vẫn sẽ giữ nguyên chỗ đúng của mình. Một ứng dụng không chỉ có một database duy nhất, và có thể kết hợp cả SQL lẫn NoSQL.

So sánh SQL và NoSQL

|  |  |  |
| --- | --- | --- |
| Tính Năng | SQL | NoSQL |
| Hiệu xuất | Kém hơn NoSQL vì khi truy vấn nó phải tính toán, kiểm tra và xử lý các mối quan hệ các bảng | Tốt hơn SQL vì nó bỏ qua các rằng buộc |
| Mở rộng theo chiều ngang | Có thể thực hiện được nhưng quá trình mở rộng sẽ rất phức tạp nếu đã tồn tại dữ liệu trong database | Mở rộng dễ dàng |
| Tấc độ read/write | Kém hơn NoSQL vì phảm đảm bảo tính rằng buộc dữ liệu giữa các bảng  Nếu sử dụng nhiều server thì phải bảo toàn tính nhất quán về dữ liệu ở các server với nhau | Tấc độ nhanh hơn SQL vì bỏ qua các cơ chế rằng buộc bảng  Vì dữ liệu được lưa trong RAM , sau đó mới đẩy xuống HDD và nó tính nhất quán cuối |
| Phần cứng | Đòi hỏi phần cứng cao | Không đòi hỏi quá cao về phần cứng |
| Thay đổi số node  Trong hệ thống | Vì tính nhất quán về dữ liệu nên khi thêm hay xóa một node cần phải shutdown hệ thống trong một khoảng thời gian | Vì tính nhất quán cuối nên sẽ không cần phải shutdown hệ thống |
| Truy vấn báo cáo | Dễ dàng sử dụng ngôn ngữ SQL query để truy vấn trực tiếp dữ liệu từ database hoặc dùng công cụ hộ trợ để lấy báo cáo | Việc báo dữ dữ liệu trực tiếp NoSQL chưa được hộ trợ tốt , thực hiện chủ yếu thông qua giao diện ứng dụng |
| Mở rộng dữ liệu | Khi muốn bổ sung thêm cột cho một bảng ,chúng ta phải khai báo trước. | Không cần khai báo trước |
| Ứng dụng | Sử dụng để xây dựng những hệ thống có quan hệ chặt chẽ và cần tính đồng nhất về dữ liệu như : tài chính, ngân hàng ,chứng khoán | Sử dụng xây dựng những hệ thống lưa trữ thông tin lớn, không quá quan trọng vấn đề đồng nhất dữ liệu trong một thời gian nhất định. Vd như : báo trí, mạng xã hội, diễn đàn, shopping.. |

Bảng so sánh NoSQL và SQL của Microsoft . Link tham khảo bên dưới



* NoSQL và RDBMS được thiết kế để hỗ trợ các yêu cầu ứng dụng khác nhau và thường thì họ cùng tồn tại trong hầu hết các doanh nghiệp. Những điểm quyết định quan trọng khi sử dụng gồm :

|  |  |
| --- | --- |
| RDBMS | NoSQL |
| Phát triển các ứng dụng tập trung (ví dụ ERP) | Phát triển các ứng dụng phi tập trung( vi dụ web , điện thoại di động và IOT |
| Các ứng dụng vừa và nhỏ để có tính sẵn sàng phục vụ cao | Tính sẵn sàng liên tục, không có thời gian chết. |
| Truy cập vào dữ liệu với tấc độ vừa phải | Truy cập vào dữ liệu tấc độ cao.( thiết bị, cảm biến , vv) |
| Dữ liệu đến từ một hoặc vài địa điểm | Dữ liệu đến từ nhiều địa điểm |
| Chủ yếu là dữ liệu có cấu trúc | Dữ liệu có cấu trúc, bán cấu trúc, phi cấu trúc. |
| Các giao dịch phức tạp, lồng nhau | Các giao dịch đơn giản |
| Mối quan tâm chính là mở rộng việc đọc, việc đọc dễ dàng | Quan tâm cả việc đọc và ghi phải dễ dàng |
| Duy trì dữ liệu vừa phải với việc có thể xóa bỏ dễ dàng | Duy tri khối lượng dữ liệu cao , có thể giữ lại mãi mãi |

## 1.5 Tạo sao lại có nhiều NoSQL Database ?

- Theo thống kê của trang : <http://nosql-database.org/> hiện tại lơn hơn 225 loại NoSQL Database . Vậy tại sao lại có nhiều đến vậy ? Sự khác là do :

* Architecture : Một vài NoSQL databases như MongoDB được kiến trúc theo mô hình master/slave gần giống như RDBMS’s . Còn NoSQL Database khác ( như Cassandra) được thiết kế theo mô hình ‘ masterless’ nơi mà tất cả các node trong database cluster là giống nhau. Kiến trúc của NoSQL database có tác động rất lớn tới việc database hộ trợ các yêu cầu như thơi gian hoạt động liên tục , dự đoán hiệu suất…
* Data Model: NoSQL database thường được phân lớp dựa vào data model mà chúng được hộ trợ. Một vài hộ trợ lưa trữ winde-row tabular , trong khi đó một vài lại hộ trợ model kiểu document-oriented , key –value hoặc graph.
* Data Distribution Model : Bởi chúng khác nhau về architecture , NoSQL databases khác nhau về việc chúng được hộ trợ việc đọc, ghi và phân phối dữ liệu. Một vài nền tảng NoSQK như Cassandra hộ trợ việc ghi và đọc trên mọi node trong một cluster và có thể nhân rộng / đồng bộ hóa dữ liệu giữa nhiều trung tâm dữ liệu và các nhà cung cấp điện toán đám mây.
* Development Model : NoSQL databases khác nhau trên API cho nhà phát triển với vài hộ trợ ngôn ngữ SQL ( vd như Cassadra’s CQL).

## 6. Lợi ích Của việc dùng NoSQL

Có rất nhiều lợi ích khi dùng NoSQL Database ví dụ như :

* Nó là nguồn mở : Các sản phẩm nguồn mở được đưa ra cho những người phát triển với một vài lợi ích lơn lao, trong khi đó có tình trạng không có chi phí của chúng. Những lợi ích khác : phần mền nguồn mở có xu hướng sẽ là tin cậy hơn ,an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữa độc quyền.
* Việc mở rộng phạm vi là mền dẻo : NoSQL thây thế câu thần chú cũ của của các nhà quản trị cơ sở dữ liệu về ‘ mở rộng phạm vi ‘ với một thứ mới ‘ mở rộng ra ngoài ‘. Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển tải dữ liệu hơn, thì cơ sở dữ liệu NoSQL cho phép một công ty phân tán tải qua nhiều máy chủ khi mà tải gia tăng.
* Các cơ sở dữ liệu NoSQL khác nhau cho những dự án khác nhau :
  + MongoDB và Redis là những lựa chọn tốt cho việc lưa trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên , như một số đếm truy cập web chẳng hạn.
  + Hadoop , một cơ sở dữ liệu dạng tự do , phân tán tốt công việc lưa trữ các dự liệu lơn như các con số thống kê thời tiết hoặc công việc phân tích nghiệp vụ.
  + Memcache , một cơ sở dữ liệu nhất thời chóng tàn, tuyệt vời trong lưa trữ các phiên làm việc web , các khóa , và các con số thống kê ngắn hạn.
  + Cassandra và Riak ( các lưa trữ dư thừa, tự động tạo bó cluster) làm tốt trong các môi trường với các ứng dụng có tính sẵn sàng cao, khi thời gian sống tối đa là sống còn.

NoSQL được các hãng lơn sử dụng : Các công ty như Amazon , BBC , Facebook và Google dựa vào các cơ sở dữ liệu NoSQL.

NoSQL phù hợp với công nghệ đám mây : Các máy chủ ngày nay là không đắt và có thể dễ dàng mở rộng phạm vi được theo yêu cầu có sử dụng một dịch vụ như là Amzon EC2 . Giống như tất các công nghệ đám mây, EC2 dựa vào ảo hóa. Liên kết yếu của ảo hóa là sự thực thi của I/O , với bộ nhớ và CPU các kết nối mạnh.

Các cơ sở dữ liệu NoSQL hầu như sử dụng bộ nhớ qua địa như vị trí ghi đầu tiên – vì thế ngăn chặn được sự thực thi không ổn định của I/O . Và vì NoSQL lưa trữ dữ liệu thường thúc đẩy tính mở rộng phạm vi theo chiều ngang thông qua việc ngăn chia, chúng có khả năng tận dụng được việc cung cấp mền dẻo của đám mây.

## 1.7 Nhược Điểm Của NoSQL

*- Nguồn mở có thể có nghĩa là sự hộ trợ không đồng đều cho các doanh nghiệp :*

* Trong khi các nhà cung cấp chủ chốt của RMBMS như Oracle , IBM hay Sybase đưa ra sự hộ trợ tốt nổi tiếng cho khách hàng doanh nghiệp cỡ vừa , thì các doanh nghiệp nhỏ hơn, thường là các nhà cung cấp nguồn mở mới thành lập không thể được mong đợi cung cấp được sự hộ trợ có thể so sánh được – ngoại trừ một nhúm khách hàng blue chip
* Nhà cung cấp nguồn mở trung bình thiếu sự tiếp cận toàn cầu, các dịch vụ hộ trợ và sự tin cậy của một Oracle hay IBM

-*Chưa đủ “chín” cho các doanh nghiệp* : Dù chúng được triển khai tại một số công ty lơn thì các cơ sở dữ liệu NoSQL vẫn phải đối mắt với một số vấn đề vế sự tin cậy chính với nhiều doanh nghiệp. Điểm sống còn đối với sự thiếu của NoSQL về độ chín muồi và các vấn đề tính không ổn định có thể, trong khi trích ra tính chính muồi, giàu chứng năng và ổn định của các RDBMS được thiết lập từ lâu.

- *Những hạn chết về tri thức nghiệp vụ:*

Có một vài câu hỏi xung quang những khả năng về tri thức nghiệp vụ (BI) của các cơ sở dữ liệu NoSQL : Liệu cơ sở dữ liệu này có thể cung cấp dạng phân tích dữ liệu lớn và mạnh mà các doanh nghiệp đã quen với RDBMS? Cần bao nhiêu sự tinh thông về lập trình để có để tiến hành những truy vấn và phân tích hiện đại ?

Các câu trả là không tích cực. Các cơ sở dữ liệu NoSQL không có nhiều sự đeo bám tới các công cụ BI thường được sử dụng , trong khi những yêu cầu và phân tích hiện đại đơn giản nhất có liên quan khác tới sự tinh thông về lập trình. Tuy vậy , các giải pháp là sẵn sàng .Quest Software , ví dụ , đã tạo ra Toad cho các cơ sở dữ liệu đám mây, mà nó phân phối các khả năng truy vấn hiện đại tới một số cơ sở dữ liệu NoSQL.

* *Thiếu sự tính thông*

Tính rất mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị mà biết công nghệ này – làm cho các công ty khó khăn tìm người với sự tinh thông phù hợp. Đối lại thế giới của RDBMS có hàng ngàn người đủ tư cách.

* *Những vấn đề về tính tương thích* :

Không giống với các cơ sở dữ liệu quan hệ , các cơ sở dữ liệu NoSQL chia sẻ ít theo cách thức của tiêu chuẩn . Mỗi cơ sở dữ liệu NoSQL có các giao diện lập trình ứng dụng API riêng của mình, các giao diện truy vấn độc nhất vô nhị, và những sự riêng biệt. Sự thiếu hụt các tiêu chuẩn có nghĩ là nó không có khả năng để chuyển một các đơn giản từ một nhà cung cấp sang một nhà cung cấp khác, nếu bạn trở nên không hạnh phúc với dịch vụ

## 1.8 Tổng kết

**-** Tuy cùng mang những đặc điểm chung của NoSQL nhưng mỗi loại cơ sở dữ liệu NoSQL cũng có những đặc điểm riêng. Vì thế chúng thường được dùng cụ thể cho những dự án khác nhau . Ví dụ

- MongoDB và Redis là những lựa chọn tốt cho việc lưa trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên

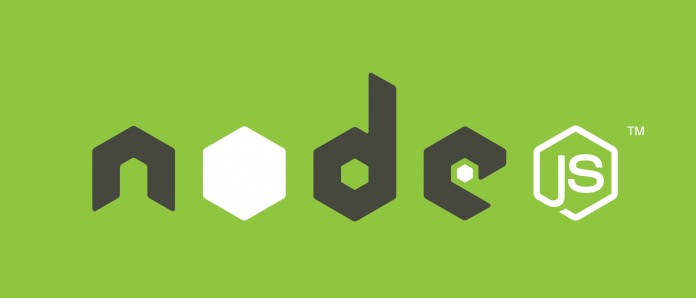
- Hadoop, một cơ sở dữ liệu dạng tự do, phán tán làm tốt công việc lưa trữ các dữ liệu lớn như các con số thống kê thời tiết, hoặc công việc phân tích nghiệp vụ.

Memcachedb một cơ sở dữ liệu nhất thời chóng tàn, tuyệt vời trong lưa trữ các phiên làm việc web, các khóa, và các con số thống kê ngắn hạn.

Canssandra va Riak ( các lưa trữ dư thừa, tự động tạo bó cluster) làm tốt trong các môi trường với các ứng dụng có tính sẵn sàng cao, khi thời gian sống tối đa là sống còn

# CHƯƠNG 2 TÌM HIỂU VỀ NODE.JS

## 2.1 Tổng Quan Về Node.JS



Hình 1. Ryan Lienhart Dahl & Node.js

### 2.1.1 Node.Js Là Gì ?

Nodejs được Ryan Lienhart Dahl giới thiệu vào năm 2009 cho phép lập trình JavaScript phía Server. Trước đây JavScript chỉ được coi là một ngôn ngữ đơn giản tạo hiệu ứng hoạt hình, tương tác phía trình duyệt. Cho đến khi Google Chrome sử dụng V8 engine , một bộ dịch , thực thi mã JavaScript nhanh nhất thể giới tính đến nay. V8 được viết bằng C/ C++ , mã nguồn mở, cho phép bên thứ 3 toàn quyền sử dụng , triển khai hệ điều hành khác nhau: Windows , Linux , Unix , MasOS . Node .js được xây dựng trên V8 và thư viện LibUV.

Định nghĩa Node.js bởi tài liệu chính thức như sau :

“ Node.js là một nền tảng dựa vào Chrome JavaScript runtime để xây dựng các ứng dụng nhanh , có độ lớn. Node.js sử dụng các phát sinh các sự kiện ( event-driven) , mô hình non-blocking I/O để tạo ra các ứng dụng nhẹ và hiệu quả cho các ứng dụng về dữ liệu thời gian thực chạy trên các thiết bị phân tán ’’

Node.js là một mã nguồn mở, đa nền tảng cho phát triển các ứng dụng phía server và các ứng dụng liên quan đến mạng. ứng dụng node.js được viết bằng JavaScript và có thể chạy trong môi trường node.js trên hệ điều hành Windows, Linux …

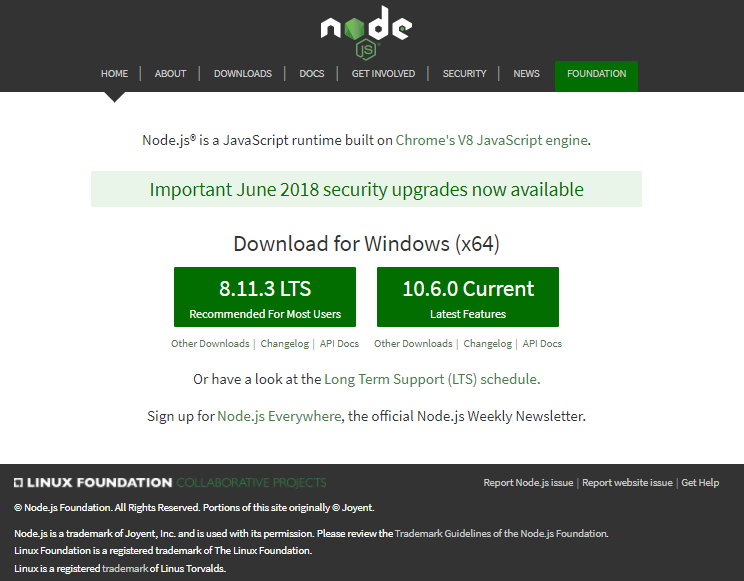
Node.js cũng cung cấp cho chúng ta các module JavaScript đa dạng, có thể đơn giản hóa, sự phát triển các ứng dụng web.

Node.js = Môi trường runtime + Các thư viện JavaScript

Nodejs phát triển nhanh đến chóng mặt. Đến nay các website lớn trên thế giới đã sử dụng Node.js gồm Microsoft , LinkedIn, Twitter, Walmart, Paypal…

Node.js liên tục được InfoWorld bình chọn là “ Technology Of the Year” từ năm 2012 đến nay.

Trang chủ : <https://nodejs.org/en/>



Phiên bản mới nhất tính đến ngày 07/8/ 2018 là v10.6.0 .Phiên bản ổn định nhất của Node.js hiện tại là :v8.11.3

### 2.1.2 Đặc điểm của Node.Js

Dưới đây là vài đặc điểm quan trọng biến Node.js thành sự lựa chọn hàng đầu trong phát triển phần mền :

**Không đồng bộ và phát sinh sự kiện ( Event Driven):** Tất cả các APIs của thư viện Node.js đều không đồng bộ, nghĩa là không blocking (khóa). Nó rất cần thiết vì Node.js không bao giờ đợi một API trả về dữ liệu. Server chuyển sang một API sau khi gọi nó và có cơ chế thông báo về sự kiện của Node.js giúp Server nhận được phản hồi từ các API gọi trước đó.

**Chạy rất nhanh:** Dựa trên V8 JavaScript Engine của Google Chrome , thư viện Node.js rất nhanh trong quá trình thực hiện code.

**Các tiến trình đơn giản nhưng hiệu năng cao :** Node.js sử dụng một mô hình luồng đơn (single thread) với các sự kiện lặp. Các cơ chế sự kiện giúp Server trả lại các phản hồi một cách không khóa và tạo cho server hiệu quả cao ngược lại với các cách truyền thống tạo ra một số lượng luồng hữa hạn để quản lý request . Node.js sử dụng các chương trình đơn luồng và các chương trình này cung cấp các dịch vụ cho số lượng request nhiều hơn so với các server truyền thống như Apache HTTP Server.

**Không đệm :** ứng dụng Node.js không lưa trữ các dữ liệu buffer

**Có giấy phép:** Node.js được phát hành dựa trên giấy phép MIT License.

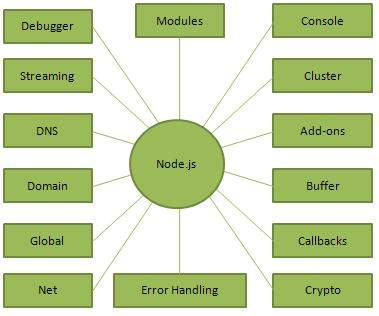
### 2.1.3 Với Node.js Bạn phải làm tất cả.

Node.js chỉ một môi trường - đều này có nghĩa bạn phải tự làm mọi thứ. Sẽ chảng có bất kì máy chủ mặc định nào cả !!!. Một đoạn script xử lý tất cả các kết nối với Client .Diều này làm giảm đáng kể số lượng tài nguyên được sử dụng ứng dụng.

### 2.1.4 Ai sử dụng node.js

<https://goo.gl/IZn6jX> chứa các danh sách các dự án , ứng dụng và các công ty sử dụng Node.js .Trong danh sách này bao gồm : eBay, GE, GoDaddy, Microsoft, Paypal, Uber..

### 2.1.5 Các Thành phần quan trọng của Node.js



**a. Luồng ( Threading)**

Node.js hoạt động trên một luồng duy nhất , sử dụng các lệnh gọi Non-blocking I/O cho phép hộ trỡ hàng chục nghìn kết nối đồng thời. Thiết kế chia sẻ một luồng duy nhất giữa tất cả các yêu cầu sử dụng mô hình quan sát được sử dụng để xây dựng ứng dụng đồng thời, trong đó bất kì chức năng nào thực hiện I/O phải sử dụng một callback. Để thức ứng với vòng lặp sự kiện đơn luồng. Node.js dùng thư viện libuv để sử dụng một threadpool có kích thước cố định. Nó chịu trách nhiệm cho một số hoạt động I/O không đồng bộ không bị ngăn chặn. Một nhược điểm của cách tiếp cận đơn luồng này là Node.js không cho phép nhân rộng theo chiều dọc bằng cách tăng số lỗi CPU của máy nó đang cahyj mà không cần sử dụng một module bổ sung. Chẳng hạn như cluster, StrongLoop Process Mannager hay pm2 . Tuy nhiên , các nhà phát triển có thể tăng số lượng mặc định của các luồng trong Threadpool libuv . các luồng này có thể được phân phối qua nhiều lõi bởi hệ điều hành máy chủ. Thực hiện các tác vụ song song trong Node.js được xử lý bằng một threadpool .Luồng chính gọi các function gửi các nhiệm vụ cho hàng đợi công việc mà luồng trong threadpool kéo và thực hiện. Các chức năng hệ thống không chặn non-blocking như việc chuyển đổi mạng sang các socket non-blocking phía kernel , trong khi đó các chức năng hệ thống như I/O chạy theo luồng riêng của nó. Khi một luồng trong threadpool hoàn thành một nhiệm vụ, nó sẽ thông báo cho thread chủ chốt của việc này lần lượt đánh thức và thực hiện callback đã đăng kí. Vì Callback được xử lý nối tiếp trên luồng chính, các phép tính kéo dài và nhiệm vụ khác rằng buộc với CPU sẽ đóng băng toàn bộ vòng lặp sự kiện cho đến khi hoàn thành.

**b. V8**

V8 là công cụ xử lý JavaScript được xây dụng cho Google Chrome và có mã nguồn mở từ google trong năm 2008. Viết bằng C++ , V8 biên dịch mã nguồn JavaScript thành mã máy thay vì interprete nó trong thời gian thực .Node.js sử dụng libuv để xử lý sự kiện không đồng bộ . Libuv là một lớp trừa tượng cho các chức năng hệ thống mạng và tệp tin trên cả Windows và các hệ thống dựa tren POSIX như linux ,MacOS, OSS trên NonStop Và Unix. Chức năng chính của Node.js nằm trong thư viện JavaScript . Các rằng buộc Node.js được viết bằng C++ , kết nối các công nghệ này với nhau và với hệ điều hành.

**c.Quản lý Package**

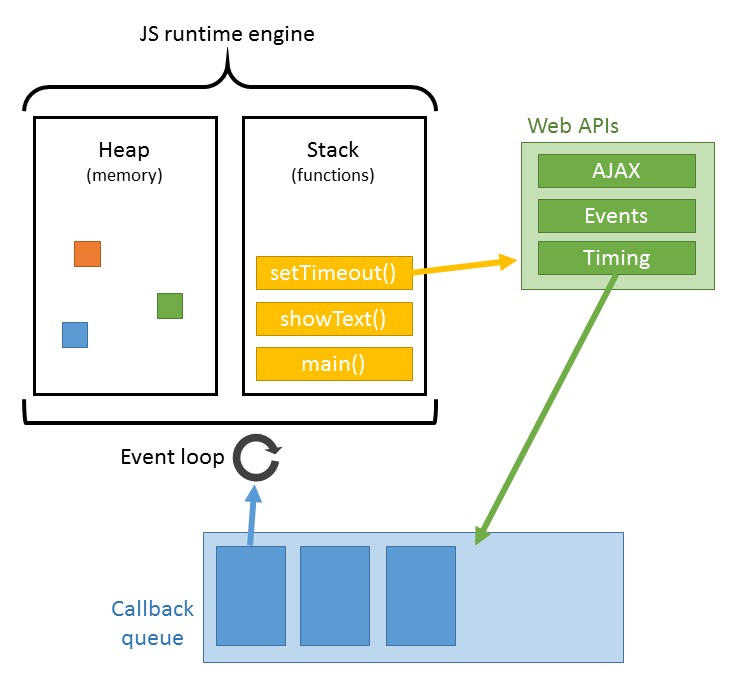
Nmp là trình quản lý gói cài đặt sẵn cho nền tảng máy chủ Node.js . Nó được sử dụng để cài đặt các chương trình Node.js từ đăng kí nmp, tổ chức cài đặt và quản lý các chương trình Node.js của bên thứ ba. NPM không được nhầm lẫn với câu lệnh yêu cầu CommonJS (). Nó không được sử dụng để nạp mã. Thay vào đó , nó được sử dụng để cài đặt mã và quản lý mã phụ thuộc từ dòng lệnh. Các gói được tìm thấy trong đăng kí nmp có thể bao gồm từ các thư viện trợ giúp đơn giản như Underscore.js cho các chương trình chạy như Grunt.

**d. Unified API**

Node.js có thể được kết hợp với một trình duyệt , một cơ sở dữ liệu hộ trợ dữ liệu JSON ( như Postgres, MongoDB hoặc CouchDB) và JSON cho một ngăn xếp phát triển thống nhất của JavaScript . Với sự thích nghi của các mô hình phát triển phía máy chủ như MVC, MVP, MVVM,… Node.js cho phép tái sử dụng cùng mô hình và giao diện dịch vụ giữa máy khách và phía máy chủ.

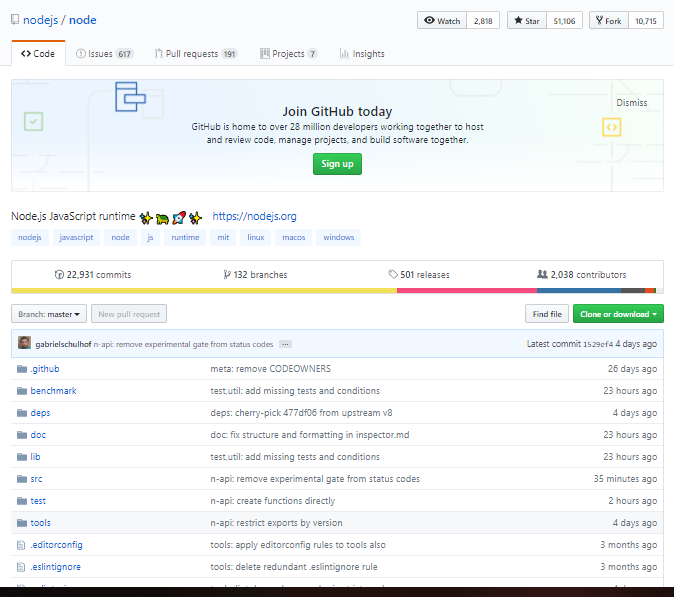
**e. Vòng lặp sự kiện( Event Loop)**

Node.js đăng kí chính nó với hệ điều hành để nó được thông báo khi một kết nối được thực hiện , và hệ điều hành sẽ phát hành một Callback Trong thời gian chạy Node.js mỗi kết nối là một sự phân bổ heap nhỏ . Thông thường, các tiến trình hoặc luồng tương đối nặng của hệ điều hành xử lý mỗi kết nối. Node.js sử dụng vòng lặp sự kiện cho khả năng mở rộng , thay vì các tiến trình hoặc các luồng. Trái ngược với các máy chủ điều khiển sự kiện khác, vòng lặp sự kiện của Node.js không cần phải được gọi rõ ràng. Thay vào đó, gọi lại được xác đinh, và máy chủ tự động nhập vào các vòng lặp sự kiện vào cuối định nghĩa gọi lại. Node.js thoát khỏi vòng lặp sự kiện khi không có callback khác được gọi thực hiện



### 2.1.6. Cộng Đồng Phát Triển

Địa chỉ mã nguồn mở cho dự án Node.js <https://github.com/nodejs/node>



Cộng đồng phát triển Node.js chủ yếu tập trung ở hai nhóm google : Node.js Và Node.js-dev, một kênh IRC là #node.js trên mạng Freenode.Có một hội thảo về Node.js cà NodeConf được tổ chức thường xuyên.

Hiện tại , cộng động Node.js việt nam hoạt động tại <https://nodejs.vn/>

## 2.2 Ưa Điểm Và Nhược Điểm

### 2.2.1 Ưa Điểm

Đặc điểm nổi bật của Node.js là nó nhận và xử lý nhiều kết nối chỉ với một single-thread. Điều này giúp hệ thống tốn ít RAM nhất và chạy nhanh nhất khi không phải tạo thread mới cho mỗi truy vấn giống PHP. Ngoài ra, tận dụng ưu điểm non-blocking I/O của Javascript mà Node.js tận dụng tối đa tài nguyên của server mà không tạo ra độ trễ như PHP

JSON APIs Với cơ chế event-driven, non-blocking I/O(Input/Output) và mô hình kết hợp với Javascript là sự lựa chọn tuyệt vời cho các dịch vụ Webs làm bằng JSON.

Ứng dụng trên 1 trang( Single page Application) Nếu bạn định viết 1 ứng dụng thể hiện trên 1 trang (Gmail?) NodeJS rất phù hợp để làm. Với khả năng xử lý nhiều Request/s đồng thời thời gian phản hồi nhanh. Các ứng dụng bạn định viết không muốn nó tải lại trang, gồm rất nhiều request từ người dùng cần sự hoạt động nhanh để thể hiện sự chuyên nghiệp thì NodeJS sẽ là sự lựa chọn của bạn.

Shelling tools unix NodeJS sẽ tận dụng tối đa Unix để hoạt động. Tức là NodeJS có thể xử lý hàng nghìn Process và trả ra 1 luồng khiến cho hiệu xuất hoạt động đạt mức tối đa nhất và tuyệt vời nhất.

Streamming Data (Luồng dữ liệu) Các web thông thường gửi HTTP request và nhận phản hồi lại (Luồng dữ liệu). Giả xử sẽ cần xử lý 1 luồng giữ liệu cực lớn, NodeJS sẽ xây dựng các Proxy phân vùng các luồng dữ liệu để đảm bảo tối đa hoạt động cho các luồng dữ liệu khác.

Ứng dụng Web thời gian thực Với sự ra đời của các ứng dụng di động & HTML 5 nên Node.js rất hiệu quả khi xây dựng những ứng dụng thời gian thực (real-time applications) như ứng dụng chat, các dịch vụ mạng xã hội như Facebook, Twitter,…

**2.3 Installtion**

### 2.3.1 Installing Node.js

Hiển nhiên là các bạn phải học cách cài đặt node trước khi muốn viết và chạy bất cứ ứng dụng nào trên nền node. Cài đặt node thì rất đơn giản, bạn là người sử dụng window hay linux thì trên website Node.js đều có những bộ cài đặt tương ứng , bạn chỉ cần download về cài đặt như bình thường.

Với Linux thì bạn sử dụng package Manager , bật cửa sổ terminal và Type:

Sudo apt-get update

Sude apt-get install node

Hoặc:

Sudo aptiude update

Sudo aptitude install node

Bạn chỉ cần thêm Node.js vào danh sách mã nguồn bằng lệnh sau.

Sudo echo deb http://fpt.us.debian.org/debian/ sid main > /etc/apt/sources.list.d/sid.list

Cần trọng khi cài sid packgare trên những hệ thống cũ hơn có thể làm hệ thống của bạn ảnh hưởng , hãy cận thận và remove /etc/apt/source.list.d/sid.list sau khi bạn cài xong Node.

### 2.3.2 Installing New Modules

Node.js có một ứng ựng quản lý packgare , đó là Node Packgate Manager (NMP) .ứng dụng này tự động được cài đặt khi bạn cài Node.js và bạn dùng NPM để cài đặt các module khác. Đề cài đặt một module , bạn mở của sổ command line của node.js ra , vào đường dẫn tường ứng và nhập lệnh.

Nmp install module\_name

Không phụ thuộc vào hệ điều hành bạn dùng , lệnh trên sẽ cài module mà bạn mong muốn chỉ định.

## 2.4. Các Object Cơ Bản

### 2.4.1 Global Objects (Đối tượng toàn cục)

Như chúng ta đã biết, hệ thống mô-dun của node không khuyến khích việc sự dụng biến toàn cục, tuy nhiên node cung cấp một globals quan trọng để sử dụng. Việc đầu tiên và quan trọng nhất là tiến trình Global , cho thấy nhiều thao tác như quá trình truyền tín hiệu, xuất cảnh, proccess(pid), và nhiều hơn thế nữa. Globals khác, chẳng hạn như console objects được cung cấp cho người sử dụng để viết JavaScript cho trình duyệt web.

1. Console

Các console objects sử dụng một số lệnh được sử dụng để xuất thông tin để stdout hoặc stdout và stderr . chúng là các lệnh như :

**Console.log ( [data], […]**

Phương pháp console objects được sử dụng thường xuyên nhất là *Console.log()* , mà chỉ đơn giản là viết cho stdout và gắn một nguồn cung cấp dữ liệu dòng (\n *console.log ( wahoo ‘);*

*// => wahoo*

*Console.log ({foo: ‘bar’});*

*console.log ({foo: 'bar'});*

*// => [Object Object]*

Còn một lệnh có chức năng như *console.log()* đó là *console.info().*

**console.error ([data], [...])**

Giống hệt nhau để *console.log(),* tuy nhiên viết cho stderr.

console.error ('kết nối cơ sở dữ liệu không thành công ');

Còn một lệnh có chức năng như console.error()đó là console.warn().

**console.dir (obj)**

Sử dụng phương pháp *inspect()* của mô-đun sys khá-in các đối tượng đến stdout.

console.dir ({foo: 'bar'});

/ / => {Foo: 'bar'}

**console.assert (expression, [message])**

Nếu *expression* bị đánh giá là có giá trị là false thì AssertionError sẽ đưa ra message được cho.

*console.assert (connected, 'Cơ sở dữ liệu kết nối không thành công* ');

**console.time(label)**

Đánh dấu thời gian bắt đầu.

**console.timeEnd (label)**

Thời gian kết thúc, được ghi vào đầu ra.Ví dụ:

*console.time('100-elements');*

*for (var i = 0; i < 100; i++) {*

*;*

*}*

*console.timeEnd('100-elements');*

**console.trace(label)**

In một tập stack các dấu vết stderr của vị trí hiện tại.

1. **Process**

Các process object gắn liền với goodies. Trước tiên, chúng ta sẽ có một cái nhìn tại một số

thuộc tính cung cấp thông tin về node process đó:

**process.version**

Chuỗi phiên bản nút, ví dụ:

*console.log (' Version:' + process.version);*

**process.execPath**

Đường dẫn đến thư mục thực thi chính của chương trình *"/ usr / local / bin / node".*

**process.platform**

Các nền tảng bạn đang sử dụng. Ví dụ, "darwin".

**process.pid**

Các process ID.

**process.stdout ()**

Một luồng có thể ghi được đến stdout.

Ví dụ: Định nghĩa về *console.log().*

*console.log = function (d) {*

*process.stdout.write (d + '\ n');*

*};*

**process.stderr ()**

Tương tự như *process.stdout()* nhưng ở đây là ghi đến stderr.

*process.stderr() và process.stdout()* là không giống như luồng khác trong Node,

khi viết chúng thường bị blocking. Chúng bị blocking trong trường hợp mà chúng liên quan

đến các tập tin thường xuyên hoặc mô tả tập tin TTY. Trong trường hợp chúng liên quan đến

các pipes, chúng không bị blocking như những luồng khác.

**process.stdin ()**

Một luồng có thể đọc được cho stdin. Các dòng stdin bị tạm dừng theo mặc định, do đó,

người ta phải gọi process.stdin.resume () để đọc từ nó.

Ví dụ mở đầu vào chuẩn và lắng nghe cả hai sự kiện:

*process.stdin.resume ();*

*process.stdin.setEncoding ('utf8');*

*process.stdin.on ('data', function (chunk) {*

*process.stdout.write ('data: ' + chunk);*

*});*

*process.stdin.on (end', function () {*

*process.stdout.write (' end');*

*});*

***process.cwd ( )***

*Trả về thư mục làm việc hiện tại. Ví dụ:*

*cd ~ && node*

*node> process.cwd() "/Users/tj"*

***process.chdir ( )***

*Thay đổi thư mục làm việc hiện tại.*

*process.chdir('/ foo');*

***process.getuid ( )***

*Trả về số user ID của process đang chạy.*

***process.setuid ( )***

*Thiết lập user ID có hiệu lực cho quá trình đang chạy. Phương pháp này chấp nhận cả một*

*số ID, cũng như một chuỗi. Ví dụ cả hai process.setuid(501), và*

*process.setuid('tj') đều hợp lệ.*

***process.getgid ( )***

*Trả về số group ID của process đang chạy.*

***process.setgid ( )***

*Tương tự như process.setuid() tuy nhiên được sử dụng trong group, cũng chấp nhận*

*một số giá trị hoặc chuỗi đại diện. Ví dụ, process.setgid(20) hoặc*

*process.setgid('www').*

*process.chdir (directory)*

*Thay đổi thư mục làm việc hiện tại của process hoặc đưa một ngoại lệ nếu thất bại.*

*console.log ('Starting directory: '+ process.cwd ());*

*try {*

*process.chdir ('/ tmp');*

*console.log ('New directory:' + process.cwd ());*

*}*

*catch (err) { console.log ('chdir: '+ err);*

*}*

*process.env*

*Một đối tượng có chứa các biến môi trường của người sử dụng. Ví dụ:*

*{ PATH:*

*'/Users/tj/.gem/ruby/1.8/bin:/Users/tj/.nvm/current/bin:/usr/bin:/bin:/u*

*s*

*/sbin:/sbin:/usr/local/bin:/usr/X1*

*, PWD: '/Users/tj/ebooks/masteringnode'*

*, EDITOR: 'mate'*

*, LANG: 'en\_CA.UTF-8'*

*, SHLVL: '1'*

*, HOME: '/Users/tj'*

*, LOGNAME: 'tj'*

*, DISPLAY: '/tmp/launch-YCkT03/org.x:0'*

*, \_: '/usr/local/bin/node'*

*, OLDPWD: '/Users/tj'*

*}*

*process.argv*

*Khi thực hiện một tập tin với các nút thực thi process.argv cung cấp truy cập vào các*

*vector đối số, giá trị đầu tiên là nút thực thi, thứ hai là tên tập tin, và giá trị còn lại là các đối*

*số được thông qua.*

*Ví dụ, tập tin nguồn của chúng ta ./src/process/misc.js có thể được thực hiện bằng cách*

*chạy:*

*$ node src/process/misc.js foo bar baz*

*mà chúng ta gọi console.dir(process.argv), xuất ra như sau:*

*[ 'node'*

*, '/Users/tj/EBooks/masteringnode/src/process/misc.js'*

*, 'foo'*

*, 'bar'*

*, 'baz'*

*]*

**process.exit ([code])**

Lệnh process.exit() là đồng nghĩa với hàm C exit(), trong đó code > 0 được biết là

thất bại, hoặc 0 được biết là thành công. Khi được gọi, việc exit được phát ra, cho phép một

thời gian ngắnđể chế biến tùy ý để xảy ra trước khi process.reallyExit() được gọi với

mã trạng thái nhất định.

**process.on ( )**

Process tự nó đã là một EventEmitter, cho phép bạn làm những điều như lắng nghe cho

các trường hợp ngoại lệ chưa bị bắt thông qua sự kiện uncaughtException:

process.on('uncaughtException', function(err){

console.log('got an error: %s', err.message);

process.exit(1);

});

setTimeout(function(){

throw new Error('fail');

*}, 100);*

**process.kill (pid, [signal])**

Lệnh process.kill() gửi tín hiệu thông qua pid nhất định, mặc định cho SIGINT. Trong

ví dụ dưới đây, chúng ta gửi tín hiệu SIGTERM đến quá trình cùng một nút để minh họa bẫy

tín hiệu, sau đó chúng ta ra " terminating " và thoát ra. Lưu ý rằng thời gian chờ thứ hai của

1000 mili giây là không bao giờ đạt được.

process.on ('SIGTERM', function () {

console.log('terminating');

process.exit(1);

});

setTimeout(function(){

console.log('sending SIGTERM to process %d', process.pid);

process.kill(process.pid, 'SIGTERM');

}, 500);

setTimeout(function(){

console.log('never called');

}, 1000);

});

**Errno**

Process object này lưu trữ của các con số báo hiệu lỗi tại máy chủ, tham khảo những gì

bạn sẽ tìm thấy trong C-land. Ví dụ, process.EPERM đại diện cho một lỗi dựa trên sự cho

phép, trong khi process.ENOENT đại diện cho một tập tin hoặc thư mục bị thiếu. Thông

thường đây là những được sử dụng trong các ràng buộc để thu hẹp khoảng cách giữa C++ và

JavaScript, nhưng chúng hữu ích cho việc xử lý các trường hợp ngoại lệ như:

if (err.errno === process.ENOENT) {

// Display a 404 "Not Found" page

} else {

// Display a 500 "Internal Server Error" page

}

**c. Buffers**

Cơ bản JavaScript là Unicode thân thiện, nhưng không phải với dữ liệu nhị phân. Khi giao.

tiếp với luồng TCP hoặc hệ thống tập tin, liệu nhị phân cần thiết để xử lý các luồng octet.

Node cung cấp một số phương pháp cho việc khai thác, tạo và sử dụng luồng octet.

Để xử lý các dữ liệu nhị phân, node cung cấp cho chúng ta với các đối tượng toàn cục.

Buffer là tương tự như một mảng các số nguyên, nhưng tương ứng với việc cấp phát bộ nhớ

thô bên ngoài V8 heap. Buffer không thể được thay đổi kích cỡ. Có một số cách để xây dựng

một trường hợp bộ đệm, và nhiều cách bạn có thể thao tác dữ liệu của nó.

Chuyển đổi giữa Buffers và các đối tượng chuỗi JavaScript đòi hỏi một phương pháp mã

hóa rõ ràng. Dưới đây là chuỗi các bảng mã khác nhau.

• 'ascii' - 7 bit dữ liệu ASCII duy nhất. Phương pháp mã hóa này là rất nhanh chóng,

và sẽ loại bỏ các bit cao nếu thiết lập. Lưu ý rằng việc mã hóa này chuyển đổi một ký

tự null ('\ 0' hoặc '\ u0000') vào 0x20 (mã ký tự của một không gian). Nếu bạn muốn

chuyển đổi một ký tự null vào 0x00, bạn nên sử dụng 'utf8'.

• 'utf8' - Nhiều byte mã hóa ký tự Unicode. Nhiều trang web và các định dạng tài liệu

khác sử dụng UTF-8.

• 'utf16le - 2 hoặc 4 byte, ký tự Unicode mã hóa ít về cuối. Các cặp đại diện (U

10.000 FFFF 10 U) được hỗ trợ.

• 'UCS2' - Tương tự 'utf16le'.

• 'base64' – Mã hóa chuỗi Base64

'binary' - Một cách mã hóa dữ liệu nhị phân thành chuỗi bằng cách sử dụng 8 bit

đầu tiên của mỗi ký tự. Phương pháp mã hóa này bị phản đối và nên tránh sử dụng

các đối tượng bộ đệm nếu có thể. Mã hóa này sẽ được loại bỏ trong các phiên bản

tương lai của Node.

• 'hex' - Mã hóa mỗi byte là hai ký tự thập lục phân.

Buffer cũng có thể được sử dụng xem mảng kiểu và DataViews.

var buff = new Buffer(4);

var ui16 = new Uint16Array(buff);

var view = new DataView(buff);

ui16[0] = 1;

ui16[1] = 2;

console.log(buff);

view.setInt16(0, 1); // set big-endian int16 at byte offset 0

view.setInt16(2, 2, true); // set little-endian int16 at byte offset 2

console.log(buff);

// <Buffer 01 00 02 00>

// <Buffer 00 01 02 00>

Cách đơn giản nhất để xây dựng một bộ đệm từ một chuỗi chỉ đơn giản là sử dụng chuỗi

như là tham số đầu tiên. Như bạn có thể nhìn thấy trong đăng nhập, bây giờ chúng ta có một

đối tượng bộ đệm có chứa 5 byte dữ liệu được đại diện trong hệ thập lục phân.

var hello = new Buffer ('Hello'); console.log (hello);

// => <Buffer 48 65 6c 6c 6f>

console.log (hello.toString ());

// => "Hello"

Theo mặc định, mã hóa là "utf8", nhưng điều này có thể được thay đổi bằng cách đi qua

một chuỗi như là đối số thứ hai. Ví dụ, dấu chấm lửng dưới đây sẽ được in stdout như kí tự

"&" khi trong bảng mã " ascii ".

var buf = new Buffer('â—¦');

console.log(buf.toString());

// => â—¦

var buf = new Buffer('â—¦', 'ascii');

console.log(buf.toString());

Một sự thay thế câu lệnh (nhưng trong trường hợp này hàm tương đương) để vượt qua

một mảng các số nguyên đại diện cho dòng octet.

var hello = new Buffer ([0x48, 0x65, 0x6c, 0x6c, 0x6f);

Bộ đệm cũng có thể được tạo ra với một số nguyên đại diện cho số lượng các byte được

phân bổ, sau đó chúng ta có thể gọi lệnh write(), cung cấp một offset và mã hóa. Dưới đây,

chúng ta cung cấp một offset của 2 byte cuộc gọi thứ hai của chúng ta để write() (buffering

"Hel") và sau đó viết 2 byte với offset của 3 (completing "Hello")

var buf = new Buffer(5);

buf.write('He');

buf.write('l', 2);

buf.write('lo', 3);

console.log(buf.toString());

// => "Hello"

The .length của một trường hợp bộ đệm chứa byte chiều dài của luồng (stream), trái

ngược với chuỗi cục bộ, chỉ đơn giản là trả lại số ký tự. Ví dụ, kí tự ellipsis 'â-¦' bao gồm ba

byte, do đó, các bộ đệm sẽ phản ứng với chiều dài byte (3), và không phải là chiều dài ký tự

(1).

var ellipsis = new Buffer('â—¦', 'utf8');

console.log('â—¦ string length: %d', 'â—¦'.length);

// => â—¦ string length: 1

console.log('â—¦ byte length: %d', ellipsis.length);

// => â—¦ byte length: 3

console.log(ellipsis);

// => <Buffer e2 80 a6>

Để xác định độ dài byte của một chuỗi cục bộ, truyền đến lệnh Buffer.byteLength().

API được viết bằng một cách như vậy mà nó giống như là String. Ví dụ, chúng ta có thể

làm việc với các " slices" của bộ đệm bằng cách truyền offset cho câu lệnh Slice():

var chunk =buf.slice( 4,9);

console.log (chunk.toString());

Ngoài ra , khi đợi một chuỗi , chúng ta có thể truyền offset đến Buffer#toString ();

var buf = new Buffer('just some data');

console.log(buf.toString('ascii', 4, 9));

// => "some"

### 2.4.2 Sự kiện (Event).

Khái niệm về một "sự kiện" là rất quan trọng trong node, và được sử dụng rất nhiều trong

suốt module chính của chương trình và module của bên thứ 3. Module sự kiện chính của

Node cung cấp cho chúng ta với một hàm tạo, EventEmitter.

1. **EventEmitter**

Thông thường một đối tượng kế thừa từ EventEmitter, tuy nhiên ví dụ nhỏ dưới đây minh

họa API. Đầu tiên chúng ta tạo ra một emitter, sau đó chúng ta có thể xác định bất kỳ số

lượng callbacks sử dụng emitter.on() phương pháp, mà chấp nhận tên của các sự kiện và

các đối tượng tùy ý thông qua như là dữ liệu. Khi emitter.emit() được gọi, chúng ta chỉ

required để truyền các tên sự kiện, theo sau bởi bất kỳ số lượng tham số (trong trường hợp

này các chuỗi tên đầu tiên và cuối cùng).

var EventEmitter = require('events').EventEmitter;

var emitter = new EventEmitter;

emitter.on('name', function(first, last){

console.log(first ', ' last);

});

emitter.emit('name', 'tj', 'holowaychuk');

emitter.emit('name', 'simon', 'holowaychuk');

**b. Kế thừa từ EventEmitter ( Inheriting From EventEmitter ).**

Được sử dụng phổ biến và thiết thực của EventEmitter là tính kế thừa từ nó. Điều này có

nghĩa là chúng ta có thể giữ nguyên EventEmitter nguyên mẫu mà không bị ảnh hưởng trong

khi sử dụng API của nó đối với phương tiện riêng của chúng ta.

Để làm như vậy, chúng ta bắt đầu bằng cách xác định các hàm khởi tạo Dog, trong đó tất

nhiên sẽ bark từ thời gian đến thời gian (còn được biết đến như một sự kiện).

var Eventemitter =require (‘events’).EventEmitter :

function Dog (nam){ this.name =name;}

Ở đây chúng ta kế thừa từ EventEmitter vì vậy chúng ta có thể sử dụng các phương thức

mà nó cung cấp, chẳng hạn như EventEmitter#on() và EventEmitter#emit (). Nếu

thuộc tính proto bị loại trừ, đừng lo lắng, chúng sẽ được trở lại này sau.

Dog.prototype. proto = EventEmitter.prototype;

Bây giờ chúng ta có Dog được thành lập, chúng ta có thể tạo ra ... Simon! Khi Simon bark,

chúng ta có thể cho stdout biết bằng cách gọi console.log() với callback. Callback chính

nó được gọi là trong ngữ cảnh của các đối tượng

var simon = new Dog('simon');

simon.on('bark', function(){

console.log(this.name ' barked');

});

Bark hai lần mỗi giây:

setInterval(function(){

simon.emit('bark');

}, 500);

c. **Loại bỏ các sự kiện lắng nghe (Removing Event Listeners).**

Như chúng ta đã biết, lắng nghe sự kiện chỉ đơn giản là hàm đó được gọi khi chúng ta

emit() một sự kiện. Chúng ta có thể loại bỏ những người nghe bằng cách dùng lệnh

removeListener(type, callback), mặc dù điều này không được dùng thường xuyên.

Trong ví dụ dưới đây, chúng ta phát ra thông báo "foo bar" mỗi 300 mili giây, trong đó có

một cuộc gọi lại của console.log(). Sau 1000 mili giây, chúng ta gọi removeListener()

với các đối số tương tự mà tôi đã thông qua on() ban đầu. Chúng ta cũng có thể sử dụng

removeAllListeners(type), trong đó loại bỏ tất cả các người nghe được đăng ký type nhất

định.

var EventEmitter = require('events').EventEmitter; var emitter = new

EventEmitter; emitter.on('message', console.log);

setInterval(function(){

emitter.emit('message', 'foo bar');

},300):

setTimeout(function(){ emitter.removeListener (‘message’, console.log);

},1000);

### 2.4.3.Luồng (Streams)

Streams là một khái niệm quan trọng trong nút. Các luồng API là một cách duy nhất để xử lý luồng giống như dữ liệu. Ví dụ, dữ liệu có thể được xem trực tiếp một tập tin, trực tiếp vào một socket để đáp ứng một HTTP request, hoặc trực tiếp từ một nguồn chỉ cho đọc như stdin.

Để bây giờ, chúng ta sẽ tập trung vào các API, để lại các chi tiết cụ thể luồng chương sau.

**a. Readable Streams**

Readable Streams được xem như một HTTP request kế thừa từ EventEmitter để lộ dữ liệu đến qua các sự kiện. Việc đầu tiên của những sự kiện này là sự kiện dữ liệu, là một đoạn tùy ý của các dữ liệu được truyền đi để xử lý sự kiện như là một trường hợp đệm ( Buffer instance ).

req.on('data', function(buf){

// Làm gì đó với Buffer

});

Một sự kiện quan trọng khác là kết thúc, đại diện cho sự kết thúc của dữ liệu sự kiện. Ví dụ, đây là một HTTP echo server, chỉ đơn giản là "simply" các request body data thông qua các response. Vì vậy, nếu chúng ta POST "hello world", response của chúng ta sẽ là " hello world ".

var http = require('http');

http.createServer(function(req, res){

res.writeHead(200);

req.on('data', function(data){

res.write(data);

});

req.on('end', function(){

res.end();

});

}).listen(3000);

Module sys thực sự có một chức năng được thiết kế đặc biệt cho hành động "simply" này,aptly tên sys.pump(). Nó chấp nhận một luồng đọc như là đối số đầu tiên, và viết dòng thứ hai.

var http = require('http'),

sys = require('sys');

http.createServer(function(req, res){

res.writeHead(200);

sys.pump(req, res);

}).listen(3000);

stream.readable

Giá trị boolean được mặc định là true, nhưng sẽ thành false sau khi xảy ra một lỗi, luồng đến một 'kết thúc', hoặc destroy() được gọi.

stream.setEncoding ([encoding])

Làm cho 'data' sự kiện phát ra một chuỗi thay vì một bộ đệm. encoding có thể là 'utf8',

'utf16le'('UCS2'),'ascii', hoặc 'hex'. Mặc định là 'utf8'. Như chúng ta biết, chúng

ta có thể gọi toString() trên một bộ đệm để trả về một chuỗi đại diện của dữ liệu nhị phân.

Tương tự như vậy, chúng ta có thể gọi setEncoding() trên một luồng, sau đó dữ liệu sự

kiện sẽ phát ra chuỗi.

req.setEncoding('utf8'); req.on('data', function(str){

// Làm gì đó với String

});

stream.pause ()

Vấn đề là một tín hiệu tư vấn cho các lớp giao tiếp cơ bản, yêu cầu không có thêm dữ liệu

được gửi cho đến khi resume() được gọi.

Lưu ý rằng, do tính chất tư vấn, luồng nhất định sẽ không được tạm dừng ngay lập tức, và

do đó, sự kiện 'data' có thể được phát ra cho một khoảng thời gian không xác định, ngay cả

sau khi pause() được gọi.

stream.resume ()

Tiếp tục lại sự kiện 'data' sau khi pause().

stream.destroy ()

Đóng tập tin mô tả cơ bản. Stream là không còn có thể ghi và cũng không thể đọc được.

Các dòng sẽ không phát ra bất kỳ chi tiết 'data', hoặc sự kiện 'kết thúc'. Bất kỳ dữ liệu ghi

xếp hàng sẽ không được gửi đi. Các luồng được tải sự kiện 'close' khi nguồn lực của mình đã được xử lý.

stream.pipe (destination, [options])

Đây là một phương pháp Stream.prototype có sẵn trên tất cả các luồng.

Kết nối này đọc dòng để WriteStream điểm đến. Dữ liệu đến trên luồng này được ghi đến

đích. Các luồng đích và nguồn được giữ đồng bộ bằng cách tạm dừng và khôi phục khi cần

thiết. Chức năng này trả về các luồng đích.

Theo mặc định end() được gọi là điểm đến khi đi qua luồng nguồn phát ra cuối cùng, do

đó, điểm đến mà không thể ghi được. {end: false} là tùy chọn để giữ cho luồng đích mở.

Điều này giữ process.stdout mở rằng "Goodbye" có thể được viết ở cuối.

process.stdin.resume();

process.stdin.pipe (process.stdout, {end: false});

process.stdin.on ("end", function () {

process.stdout.write ("Goodbye \ n");});

b. Writable Stream.

Một lớp cơ sở cho việc tạo ra các Writable Stream. Tương tự như Readable Stream, bạn

có có thể tạo ra các lớp con bằng cách ghi đè không đồng bộ khi sử dụng câu lệnh

\_write(chunk, cb).

stream.writable

Giá trị boolean được mặc định đó là true, nhưng sẽ thành false sau khi lỗi xảy ra hoặc

end() /destroy() được gọi.

stream.Write (string, [encoding])

Viết chuỗi với encoding cho luồng. Trả về true nếu chuỗi đã được bỏ vào bộ đệm kernel.

Trả về false để cho biết rằng bộ đệm kernel đã đầy, và dữ liệu sẽ được gửi đi trong tương lai.

Sự kiện 'drain' sẽ cho biết khi nào bộ đệm kernel là rỗng một lần nữa. Việc mã hóa mặc định

'utf8'.

stream.Write (buffer)

Tương tự như trên, ngoại trừ với một bộ đệm thô.

stream.end ()

Kết thúc dòng với EOF hoặc FIN. Cuộc gọi này sẽ cho phép hàng đợi ghi dữ liệu được

gửi trước khi đóng luồng.

stream.end (string, encoding)

Gửi chuỗi với mã hóa nhất định và chấm dứt dòng với EOF hoặc FIN. Điều này rất hữu

ích để giảm số lượng các gói tin gửi đi.

stream.end (buffer)

Tương tự như trên, nhưng với một bộ đệm.

stream.destroy ()

Đóng mô tả tập tin cơ bản. Stream là không còn có thể ghi và cũng không thể đọc được.

các luồng sẽ không phát ra bất kỳ chi tiết 'data', hoặc sự kiện 'kết thúc'. Bất kỳ hàng đợi dữ

liệu ghi sẽ không được gửi đi. các luồng được tải sự kiện 'close' khi tài nguyên của mình đã. được xử lý.

stream.destroySoon ()

Sau khi ghi hàng đợi được giải phóng, đóng tập tin mô tả. destroySoon() vẫn có thể hủy

ngay lập tức, miễn là không có dữ liệu còn lại trong hàng đợi để viết.

### 2.4.4 File System

Để làm việc với hệ thống tập tin, node cung cấp module "fs". Các lệnh thực thi các hoạt

động POSIX, và hầu hết các phương pháp làm việc đồng bộ hoặc không đồng bộ. Chúng ta

sẽ xem xét làm thế nào để sử dụng cả hai, sau đó thiết lập lựa chọn tốt hơn.

**a. Làm việc với tập tin hệ thống.**

Cho phép bắt đầu với một ví dụ cơ bản làm việc với tập tin hệ thống. Ví dụ này tạo một

thư mục, tạo ra một tập tin bên trong nó, sau đó viết nội dung của tập tin đến console:

var fs = require('fs');

fs.mkdir('./helloDir',0777, function (err) {

if (err) throw err;

fs.writeFile('./helloDir/message.txt', 'Hello Node', function (err)

{ if (err) throw err;

console.log('file created with contents:');

fs.readFile('./helloDir/message.txt','UTF-8' ,function (err, data)

{ if (err) throw err;

console.log(data);

});

});

});

Rõ ràng trong ví dụ trên, ứng với mỗi callback đều đặt trong callback trước đây được gọi

là callbacks chainable. Mô hình này cần được theo sau khi sử dụng phương pháp không đồng

bộ, không có đảm bảo rằng các hoạt động sẽ được hoàn thành theo thứ tự họ đang tạo ra.

Điều này có thể dẫn đến hành vi không thể đoán trước.

Ví dụ có thể được viết lại để sử dụng một cách tiếp cận đồng bộ:

fs.mkdirSync('./helloDirSync',0777);

fs.writeFileSync('./helloDirSync/message.txt', 'Hello Node');

var data = fs.readFileSync('./helloDirSync/message.txt','UTF-8');

console.log('file created with contents:');

console.log(data);

Nó là tốt hơn để sử dụng phương pháp không đồng bộ trên các máy chủ với một tải cao,

như các phương pháp đồng bộ sẽ làm cho toàn bộ quá trình để ngăn chặn và chờ cho các hoạt

động để hoàn thành. Điều này sẽ ngăn chặn bất kỳ kết nối đến hoặc các sự kiện khác.

1. **File thông tin**

Các đối tượng fs.Stats có chứa thông tin về một tập tin hoặc thư mục cụ thể. Điều này có

thể được sử dụng để xác định loại đối tượng mà chúng ta đang làm việc. Trong ví dụ này,

chúng ta đang nhận được tất cả các đối tượng tập tin trong một thư mục và hiển thị cho dù

chúng là một tập tin hoặc một đối tượng thư mục.

var fs = require('fs');

fs.readdir('/etc/', function (err, files)

{ if (err) throw err;

files.forEach( function (file) {

fs.stat('/etc/' + file, function (err, stats)

{ if (err) throw err;

if (stats.isFile()) {

console.log("%s is file", file);

}

else if (stats.isDirectory ()) {

console.log("%s is a directory", file);

}

console.log('stats: %s',JSON.stringify(stats));

});

});

});

**c. Xem các tập tin**

Phương pháp fs.watchfile theo dõi một tập tin và thay đổi một sự kiện bất cứ khi nào tập

tin được thay đổi.

var fs = require('fs');

fs.watchFile('./testFile.txt', function (curr, prev) {

console.log('the current mtime is: ' + curr.mtime);

console.log('the previous mtime was: ' + prev.mtime);

});

fs.writeFile('./testFile.txt', "changed", function (err) {

if (err) throw err;

console.log("file write complete");

});

Một tập tin cũng có thể được unwatched bằng cách sử dụng phương pháp gọi

fs.unwatchFile. Cách này chỉ nên sử dụng một lần khi tập tin không còn cần được giám sát.

### 2.4.5.HTTP

Để sử dụng HTTP server và client phải dùng lệnh require('http').

Các giao diện HTTP trong Node được thiết kế để hỗ trợ nhiều tính năng của các giao thức

truyền thống khó sử dụng. Trong đó, có thể là đoạn mã hóa, tin nhắn.

HTTP headers được biểu diễn bởi một đối tượng như thế này:

{ 'content-length': '123',

'content-type': 'text/plain',

'connection': 'keep-alive',

'accept': '\*/\*' }

Key được lowercased và giá trị không được sửa đổi.

http.STATUS\_CODES

Một bộ sưu tập của tất cả các mã trạng thái tiêu chuẩn của HTTP response, và mô tả ngắn

gọn cho từng cái. Ví dụ, http.STATUS\_CODES [404] === 'Not Found'.

http.createServer ([requestListener])

Trả về một đối tượng web server mới. RequestListener là một chức năng được tự động

thêm vào sự kiện 'request'.

http.createClient ([port], [host])

Hàm này bị phản đối sử dụng, hãy sử dụng http.request() để thay thế. Xây dựng một

HTTP client mới. Port và host tham chiếu đến máy chủ để được kết

a. Clas: http.Server.

server.listen(port, [hostname], [backlog], [callback])

Bắt đầu chấp nhận các kết nối trên port chỉ định và hostname. Nếu hostname được bỏ qua,

các server sẽ chấp nhận các kết nối trực tiếp đến bất kỳ địa chỉ IPv4 (INADDR\_ANY). Để

nghe một socket unix, cung cấp một tên tập tin thay vì port và hostname.

Backlog là chiều dài tối đa của hàng đợi kết nối đang chờ. Chiều dài thực tế sẽ được xác

định bởi hệ điều hành thông qua các thiết lập sysctl như tcp\_max\_syn\_backlog và

somaxconn trên Linux. Giá trị mặc định của tham số này là 511 (không phải 512).

Hàm này là không đồng bộ. Gọi lại tham số cuối cùng sẽ được thêm vào như là một

listener cho sự kiện 'listen'. Xem thêm net.Server.listen(port).

server.listen (path, [callback])

Bắt đầu một máy chủ socket UNIX lắng nghe cho các kết nối trên đường dẫn nhất định.

Hàm này là không đồng bộ. Gọi lại tham số cuối cùng sẽ được thêm vào như là một người

biết lắng nghe cho sự kiện'listen'. Xem thêm net.Server.listen (path).

server.listen (handle, [callback])

Các đối tượng có thể được thiết lập để xử lý một máy chủ hoặc socket, hoặc đối tượng

{fd: <n>}.

Lắng nghe trên một mô tả tập tin không được hỗ trợ trên Windows.

Chức năng này là không đồng bộ. Gọi lại tham số cuối cùng sẽ được thêm vào như là một

người biết lắng nghe cho sự kiện 'listen'. Xem thêm net.Server.listen().

server.close ([callback])

Dừng server và chấp nhận các kết nối mới. Xem net.Server.close().

server.maxHeadersCount

Giới hạn tối đa headers count đến, là 1000 theo mặc định. Nếu thiết lập là 0 - không có

giới hạn được áp dụng.

**b. Class: http.ServerRequest**.

Đối tượng này được tạo ra trong nội bộ của một máy chủ HTTP - không phải bởi người sử

dụng - và thông qua như là đối số đầu tiên để một listener 'request'. Request thực hiện các

giao diện Readable Streams.

request.method

Yêu cầu một chuỗi. Chỉ đọc. Ví dụ: 'GET', 'DELETE'.

request.url

Yêu cầu chuỗi URL. Câu lệnh này chỉ chứa các URL theo thực tế trong HTTP request.

Nếu request là:

GET /status?name=ryan HTTP/1.1\r\n

Accept: text/plain\r\n

\r\n

Thì request.url sẽ là:

'/status?name=ryan'

Nếu bạn muốn để phân tích các URL thành các phần của nó, bạn có thể sử dụng

require('url').parse(request.url). Ví dụ:

node> require('url').parse('/status?name=ryan')

{ href: '/status?name=ryan',

search: '?name=ryan',

query: 'name=ryan',

pathname: '/status' }

Nếu bạn muốn trích xuất các params từ chuỗi truy vấn, bạn có thể sử dụng

require('querystring').parse, hoặc thông qua các đối số thứ hai để

require('url').parse. Phân tích cú pháp. Ví dụ:

node> require('url').parse('/status?name=ryan', true)

{ href: '/status?name=ryan',

search: '?name=ryan',

query: { name: 'ryan' },

pathname: '/status' }

request.headers

Đọc bản đồ duy nhất của tên header và giá trị. Tên header là lower-cased. Ví dụ:

// In một cái gì đó như:

// {User-agent ':' curl/7.22.0 '

// Host: '127 .0.0.1:8000 ',

// accept: '\*/\*' }

console.log(request.headers);

request.trailers

Chỉ đọc HTTP trailer (nếu có).

request.httpVersion

Phiên bản giao thức HTTP như một chuỗi. Chỉ đọc. Ví dụ: '1.1 ', '1.0'. Ngoài ra

request.httpVersionMajor là số nguyên đầu tiên và request.httpVersionMinor là thứ

hai.

request.setEncoding ([encoding])

Thiết lập mã hóa cho cơ chế request. Xem stream.setEncoding() để biết thêm thông

tin.

request.pause ()

Tạm dừng request từ các sự kiện phát ra. Rất hữu ích để tăng tốc tải lên.

request.resume ()

Tiếp tục lại request tạm dừng.

request.connection

Các đối tượng net.Socket kết hợp với kết nối. Với hỗ trợ HTTPS, sử dụng

request.connection.verifyPeer() và request.connection.getPeerCertificate()

để có được thông tin xác thực của client.

1. Class: http.ServerResponse.

Đối tượng này được tạo ra trong nội bộ của một HTTP server - không phải bởi người sử

dụng. Nó được thông qua như là tham số thứ hai của sự kiện 'request'. Response thực hiện

các giao diện Writable Stream.

response.writeHead (statusCode [reasonPhrase], [headers])

Gửi một response headers để đáp ứng yêu cầu. statusCode là một trạng thái HTTP có 3

chữ số mã, giống như 404. Tùy chọn có thể đưa ra một reasonPhrase như là đối số thứ hai.

Ví dụ:

var body = 'hello world';

response.writeHead(200, {

'Content-Length': body.length,

'Content-Type': 'text/plain' });

Câu lệnh này chỉ được gọi là một lần trên một tin nhắn và nó phải được gọi trước khi

response.end() được gọi.

Nếu bạn gọi response.write() hoặc response.end() trước khi gọi, header ngầm định

sẽ được tính toán và gọi hàm này cho bạn.

Lưu ý: rằng Content-Length được đưa ra trong các byte không phải ký tự. Ví dụ trên làm

việc vì chuỗi 'hello world' chứa các byte ký tự duy nhất. Nếu nó chứa các ký tự được mã hóa

cao hơn thì sau đó Buffer.byteLength() nên được sử dụng để xác định số byte trong một

đoạn mã hóa nhất định. Và Node không kiểm tra Content-Length và độ dài của thân đã

được truyền đi bằng nhau hay không.

response.statusCode

Khi sử dụng các header ẩn (không gọi response.writeHead() một cách rõ ràng), hàm

điều khiển các mã trạng thái đó sẽ được gửi đến cho client khi các header nhận được.

Ví dụ:

response.statusCode = 404;

Sau khi response header đã được gửi cho client, hàm này chỉ ra các mã trạng thái được

gửi.

response.setHeader (name, value)

Thiết lập một giá trị header duy nhất cho header ẩn. Nếu header này đã tồn tại trong phần

header được gửi, giá trị của nó sẽ được thay thế. Sử dụng một mảng các chuỗi ở đây nếu bạn

cần phải gửi nhiều header với cùng một tên.

Ví dụ:

response.setHeader ("Content-Type", "text / html");

hoặc

response.setHeader ("Set-Cookie", "loại = ninja", "ngôn ngữ = javascript

"]);

response.sendDate

Khi giá trị là true, date header sẽ được tự động tạo ra và gửi response nếu nó không phải là

đã có trong các header. Mặc định là true.

response.getHeader (name)

Đọc ra một header đó đã được xếp hàng đợi nhưng không gửi cho client. Điều này chỉ có

thể được gọi là trước khi header nhận được.

Ví dụ:

var contentType = response.getHeader ('content-type');

response.removeHeader (name)

Loại bỏ một header xếp hàng đợi để gửi ngầm.

Ví dụ:

response.removeHeader ("Content-Encoding");

response.write (chunk, [encoding])

Nếu hàm này được gọi và response.writeHead() đã được gọi, nó sẽ chuyển sang chế độ

header ẩn và luôn xuất ra các header ẩn. Phương pháp này có thể được gọi là nhiều lần để

cung cấp các bộ phận liên tiếp nhau.

chuck có thể là một chuỗi hoặc một bộ đệm. Nếu chuck là một chuỗi, tham số thứ hai xác

định làm thế nào để mã hóa nó thành một dòng byte. Theo mặc định, mã hóa là 'utf8'.

Khi response.write() được gọi lần đầu tiên, nó sẽ gửi các thông tin header đệm và thân

đầu tiên cho client. Khi response.write() được gọi lần thứ hai, Node giả định bạn sẽ được

dữ liệu, và gửi riêng dữ liệu.

Trả về true nếu toàn bộ dữ liệu được xuất thành công đến kernel buffer. Trả về false nếu

tất cả hoặc một phần của dữ liệu được xếp hàng đợi trong bộ nhớ của người sử dụng. 'drain'

sẽ được phát ra khi bộ đệm trống một lần nữa.

response.end ([data], [encoding])

Với hàm này, tín hiệu đến server mà tất cả các response headers và phần thân đã được gửi,

server nên xem xét hoàn thành tin nhắn này. response.end() phải được gọi là trên mỗi câu

trả lời. Nếu dữ liệu được quy định cụ thể, đó là tương đương với gọi response.write

(data, encoding) tiếp theo response.end().

**d. http.ClientResponse.**

Đối tượng này được tạo ra khi thực hiện một yêu cầu với http.request(). Nó đượcthông qua với sự kiện 'response' của đối tượng yêu cầu. response thực hiện các giao diện Readable Stream.response.statusCode

3 chữ số mã trạng thái của HTTP response. VÍ DỤ: 404.

response.httpVersion

Các phiên bản HTTP của máy chủ để kết nối. Có thể hoặc là '1.1 'hoặc '1.0'. Ngoài ra

response.httpVersionMajor là số nguyên đầu tiên và response.httpVersionMinor là thứ hai.

response.headers

Đối tượng response headers .

response.trailers

Trailer đối tượng response.

response.setEncoding ([encoding])

Thiết lập mã hóa cho cơ thể phản ứng. Xem stream.setEncoding () để biết thêm thông

tin.

response.pause ()

Tạm dừng response từ các sự kiện phát ra. Hữu ích để tăng tốc tải về.

response.resume ()

Tiếp tục lại một response bị tạm dừng.

# CHƯƠNG 3. TÌM HIỂU CÁCH TẤN CÔNG VÀ NGĂN CHẶN INJECTION VÀO NOSQL QUA ỨNG DỤNG WEB NODE.JS

## 3.1.TỔNG QUAN VỀ INJECTION NOSQL

Chăc hẳn chúng ta khá quên thuộc với khái niệm SQL Injection đây là lỗ hổng phổ biến cũng như nguy hiểm với các trang web trên Internet. Mới đây với sự phát triển của những loại database mới như Graph Database và NoSql đã làm phong phú hơn lựa chọn database của những dự án phần mền. Với những ưa điểm của mình:

* Mã nguồn mở
* Khả năng mở rộng linh hoạt
* Phù hợp với điện toán đám mây

Các cơ sở dữ liệu như MongoDB hay redis là những lựa chọn đáng cân nhắc với những hệ thống lớn, một khía cạnh có lẽ không được cân nhắc đến nhất là ở việt nam đó chính là tính bảo mật của cơ sở dữ liệu này.

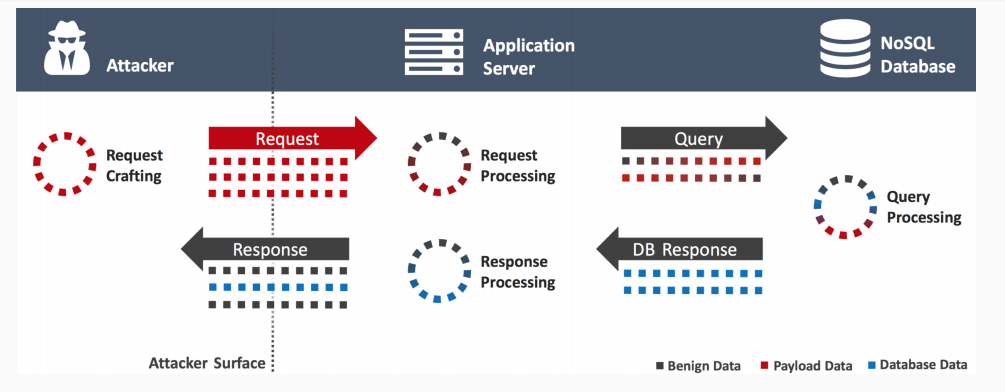
Trong bài viết này mình sẽ tìm hiểu về một số lỗ hổng thường gặp trong hệ quản trị cơ sở dữ liệu NoSQL, mà cụ thể là MongoDb hệ quản trị cơ sở dữ liệu Nosql phổ biến nhất hiện nay.

### 3.1.1 Cơ chế hoạt động

Theo như trang Mongo DB Developer FAQ có nói :

"... with MongoDB we are not building queries from strings, so traditional SQL injection attacks are not a problem “

Tức là MongoDB có thể tránh được tất cả các loại SQL injection truyền thống, . Nhưng thay vào đó các hacker lại sự dụng một kĩ thuật đặc thù cho NoSQL đó là NoSQL



**C**ũng như SQL Injection để có thể thực hiện Nosql Injection hacker cũng phải truy vấn lên server ,dựa vào request từ client server sẽ truy vấn đến Database và nhận lại kết quả từ database để thực hiện trả cho client. Nhiệm vụ của hacker là phải làm sao để hiểu sai request dẫn đến một thực hiện một câu truy vấn không mong muốn đến database cuối cùng hacker sẽ lấy được thông tin mong muốn hoặc nghiêm trọng hơn là xóa dữ liệu trong cơ sở dữ liệu.

Điểm khác nhau giữa Nosql Injection và SQL Injection truyền thống

* câu truy vấn cho khối dữ liệu không cấu trúc.
* các HQTCSDL rất đa dạng và khác biệ nhau khá nhiều trong cachs tổ chức dữ liệu cũng như truy vấn.
* cho phép truy cập trực tiếp client-database thông qua RESTfull API.

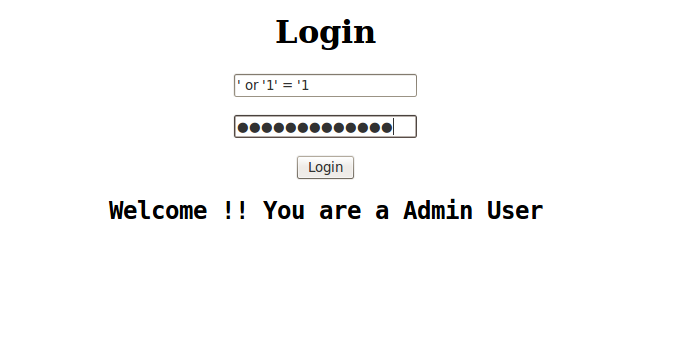
### 3.1.2 Các loại tấn công

Có 3 loại tấn công Injection NoSQL chủ yếu là .

Login Bypass For MongoDb Node Js

Cách tấn công :

Đây là một loại tấn công rất nguy hiểm vì hacker có thể bỏ qua quy trình đăng nhặp và giành quền một cách không khó khăn.



Xét Đoạn chương trình

app.post('/user', function (req, res) {

var query = {

username: req.body.username,

password: req.body.password

}

db.collection('users').findOne(query, function (err, user) {

console.log(user);

});

});

ở đây ta sử dụng luôn tham số trong request của client để thực hiện câu truy vấn đến database khi đó tham số mà người dụng gửi lên sẽ ở dạng :

<https://example.org/login?user=balam&password=1234>

Trong trường hợp này nếu hacker gửi lên một request dạng :

POST http://www.example.com/user HTTP/1.1

Content-Type: application/json

{

"username": {"$ne": null},

"password": {"$ne": null}

}

Toán tử $ne là toán tử không bằng nhau trường password luôn là true. Như vậy yêu cầu sẽ trả về người đầu tiên .Như vậy hacker đã pypass thành công

* 1. Giai pháp .

Trong trường hợp này khử trùng đầu vào trước khi sử dụng chúng. Một lựa chọn tốt nhất mongo-sanitize:

Nó sẽ loại bỏ bất kì khóa nào bằng $ trong đầu vào

var sanitize = require('mongo-sanitize');

app.post('/user', function (req, res) {

var query = {

username: sanitize(req.body.username),

password: sanitize(req.body.password)

}

db.collection('users').findOne(query, function (err, user) {

console.log(user);

});

});