**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

------------------------------



**BÁO CÁO THỰC TẬP TỐT NGHIỆP ĐẠI HỌC**

**Đề tài :**

**“NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL”**

**Người hướng dẫn : ThS. HUỲNH TRỌNG THƯA**

**Sinh viên thực hiện :**  **NGUYỄN BÁ LÂM**

**Mã số sinh viên : N14DCA141**

**Lớp : D14CQAT01-N**

**Khóa : 2014**

**Hệ : CHÍNH QUY**

**TP.HCM, tháng 8/2018**

# LỜI CẢM ƠN

Kính thưa quý thầy cô !

Trong quá trình thực hiện đề tài “ **Ngăn chặn tấn công Injection vào NoSQL**” em đã nhận được nhiều sự giúp đỡ, tạo điều kiện của tập thể Ban giám Hiệu, Khoa Công Nghệ Thông Tin, các thầy cô, cán bộ các phòng ban chức năng Học Viện Công Nghệ Bưa Chính Viễn Thông tại TP.HCM .Em xin bày tỏ làm cảm ơn chân thành về sự giúp đỡ đó.

Đặc biệt ,em xin bày tỏ lòng biết ơn sâu sắc tới ThS.Huỳnh Trọng Thưa. Thầy đã hướng dẫn tận tình, truyền đạt kiến thức, chỉ bảo em hoàn thành đề tài này.

Sau cùng , em xin kính chúc các quý thầy cô Khoa Công Nghệ Thông Tin và ThS. Huỳnh Trọng Thưa thật dồi dào sức khỏe , niềm tin để tiếp tục ương mần thế hệ mai sau.

Trân trọng !

TP.Hồ Chí Minh,Ngày…tháng…năm 2018

Sinh viên thực hiện

**Nguyễn Bá Lâm**

MỤC LỤC

[DANH MỤC CÁC HÌNH, SƠ ĐỒ, BẢNG 5](#_Toc521055521)

[MỞ ĐẦU 6](#_Toc521055522)

[CHƯƠNG I : TÌM HIỂU VỀ NOSQL DATABASE 7](#_Toc521055523)

[1.1 Thời đại của NoSQL 7](#_Toc521055524)

[1.2 NoSQL là gì ? 8](#_Toc521055526)

[1.2.1 Định nghĩa 9](#_Toc521055527)

[1.2.2 Một số thuật ngữ liên quan. 10](#_Toc521055529)

[1.3 Phân loại cơ cở dữ liệu NoSQL 11](#_Toc521055530)

[1.3.1 Key – Value Store 11](#_Toc521055531)

[1.3.2 Document Databases 13](#_Toc521055532)

[1.3.3 Column Family Stores 14](#_Toc521055533)

[1.3.4 Graph Database 16](#_Toc521055534)

[1.4 So sánh NoSQL và SQL 17](#_Toc521055535)

[1.5 Tạo sao lại có nhiều NoSQL Database ? 21](#_Toc521055536)

[1.6 Lợi ích của việc dùng NoSQL 21](#_Toc521055537)

[1.7 Nhược điểm của NoSQL 22](#_Toc521055538)

[1.8 Tổng kết 23](#_Toc521055539)

[CHƯƠNG II : TÌM HIỂU VỀ NODE.JS 25](#_Toc521055540)

[2.1 Tổng quan về Node.Js 25](#_Toc521055541)

[2.1.1 Node.Js là gì ? 25](#_Toc521055542)

[2.1.2 Đặc điểm của Node.js 26](#_Toc521055543)

[2.1.3 Với Node.js bạn phải làm tất cả. 27](#_Toc521055544)

[2.1.4 Ai sử dụng Node.js 27](#_Toc521055545)

[2.1.5 Các thành phần quan trọng của Node.js 27](#_Toc521055546)

[2.1.6. Cộng đồng phát triển 29](#_Toc521055547)

[2.2 Ưa điểm và nhược điểm 30](#_Toc521055548)

[2.2.1 Ưa điểm 30](#_Toc521055549)

[2.3.1 Installing Node.js 31](#_Toc521055550)

[2.3.2 Installing New Modules 35](#_Toc521055551)

[2.4. Các Object cơ bản 35](#_Toc521055552)

[2.4.1 Global Objects (Đối tượng toàn cục) 35](#_Toc521055553)

[2.4.2 Sự kiện (Event). 44](#_Toc521055554)

[2.4.3.Luồng (Streams) 46](#_Toc521055555)

[2.4.4 File System 50](#_Toc521055556)

[2.4.5.HTTP 52](#_Toc521055557)

[CHƯƠNG III :TÌM HIỂU CÁCH TẤN CÔNG VÀ NGĂN CHẶN INJECTION VÀO NOSQL QUA ỨNG DỤNG WEB NODE.JS 58](#_Toc521055558)

[3.1.TỔNG QUAN VỀ INJECTION NOSQL 58](#_Toc521055559)

[3.1.1 Cơ chế hoạt động 58](#_Toc521055560)

[3.1.2 Các loại tấn công 59](#_Toc521055561)

[CHƯƠNG IV: XÂY DỰNG ỨNG DỤNG WEB ĐƠN GIẢN BẰNG NODE.JS NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL 61](#_Toc521055562)

[4.1 Tổng quan về ứng dụng Facebook Massenger 61](#_Toc521055563)

[4.1.1 Các công nghệ sử dụng trong việc xây dựng ứng dụng 62](#_Toc521055564)

[4.1.2 Một số tính năng của ứng dụng Facebook Massenger 62](#_Toc521055565)

[4.2 Cài đặt một số tính năng tiêu biểu 62](#_Toc521055566)

[4.2.1 Đăng nhập và Đăng kí 62](#_Toc521055567)

[4.2.2 Lưa trữ và kết nối MongoDB 64](#_Toc521055568)

[4.2.3 Quản lý user và Group chat bằng session 65](#_Toc521055569)

[4.2.4 Ngăn chặn tấn công Injection vào NoSQL 65](#_Toc521055570)

[CHƯƠNG V: KẾT LUẬN 66](#_Toc521055571)

[5.1 Kết quả đạt được 66](#_Toc521055572)

[5.2 Hạn Chế 66](#_Toc521055573)

[DANH MỤC TÀI LIỆU THAM KHẢO 67](#_Toc521055574)

# 

DANH MỤC CÁC HÌNH, BẢNG, SƠ ĐỒ

## Hình 1.1.

## Hình 1.2. Key/ Value Database

## Hình 1.3. Document Store

## Hình 1.4. Super Column Family

## Hình 1.5. Graph Database Store

## Hình 1.6. NoSQL Graph Database

## Hình 1.7 . So sánh NoSQL Và SQL của Microsoft

## Hình 2.1. Ryan Lienhart Dahl & Node.js

## Hình 2.2 . Trang chủ Node.js

## Hình 2.3 . Thành phần quan trọng của Node.js

## Hình 2.4 .Even Loop

## Hình 2.5 .Cộng đồng phát triển

## Hình 3.1. Cơ chế hoạt động

## Hình 4.1 Cấu trúc ứng dụng Facebook. Messenger

## Bảng 1. Key /Value Store

## Bảng 2. So sánh SQL và NoSQL

## Bảng 3. So sánh RDBMS và NoSQL

# MỞ ĐẦU

# 

## 

## CHƯƠNG I : TÌM HIỂU VỀ NOSQL DATABASE

## NoSQL là gì ?

### Định nghĩa

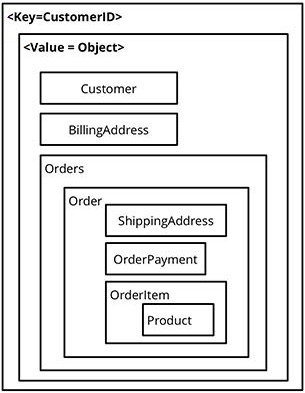
NoSQL là một xu hướng cơ sở dữ liệu mà không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mền. NoSQL có nghĩa là No-Relational (NoRel) – Không rằng buộc. Tuy nhiên thuật ngữ đó ít phổ dụng hơn và ngày ngày người ta thường dịch NoSQL thành Not Only SQL – Không chỉ SQL .

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)

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



## Hình 1.2. Key/ Value Database

### 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.
* 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.

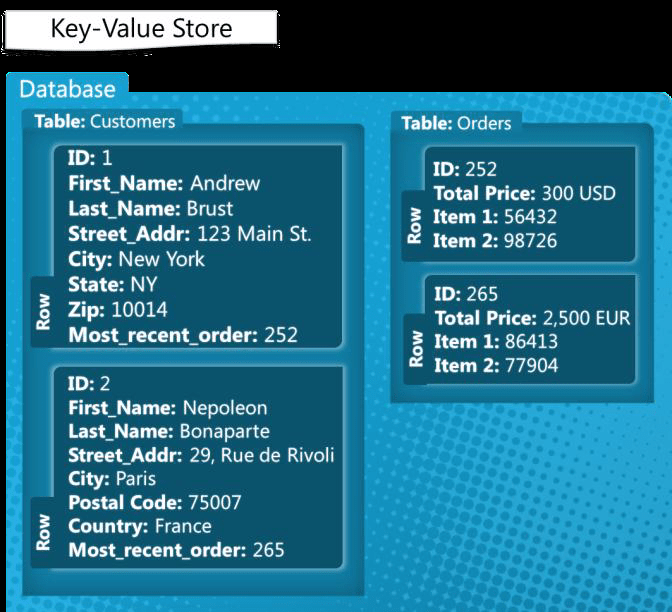
## Phân loại cơ sở dữ liệu NoSQL

Tính đến thời điểm hiện tại thì có hơn 255 cơ sở dữ liệu NoSQL (<http://nosql-database.org/>) . Các loại cơ sở dữ liệu này không giống nhau về các lưa trữ. Và chúng được chia thành những nhóm sau.

### 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.

Ý tưởng chính của kiểu Key – Value là sử dụng bảng băm ( hash-table) , nới có một khóa duy nhất và một con trỏ chỉ đến một mục dữ liệu cụ thể. Các mô hình key/value thì đơn giản và dễ thực hiện nhất. Nhưng nó sẽ không hiệu quả khi bạn chỉ truy vấn hay update đến một phần dữ liệu trong phần giá trị.



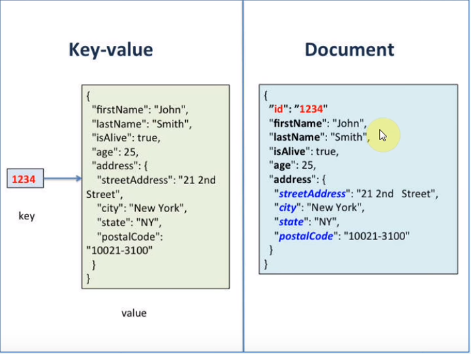
Bảng 1. Key – Value Store

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.

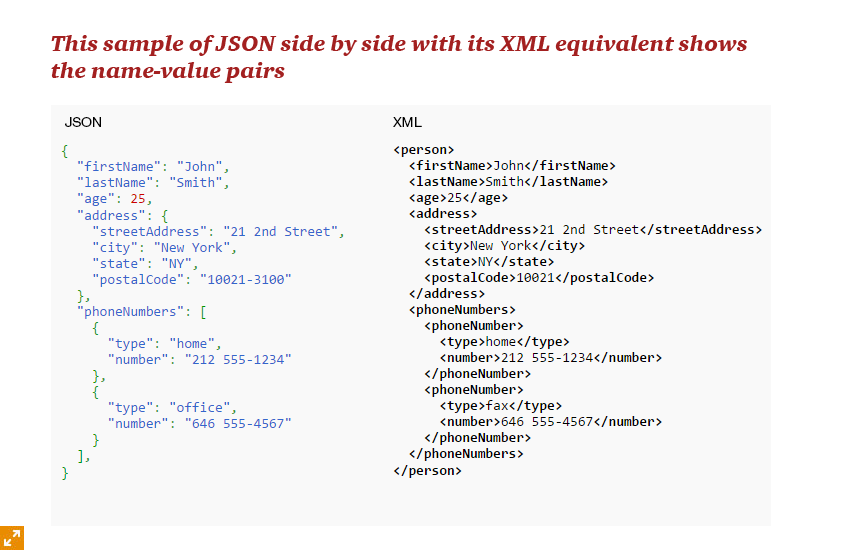
Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này: Aerospike, CouchDB, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB

### Document Databases

Cơ sở dữ liệu Document được thiết kế để quản lý và lưa trữ dữ liệu ở dạng document. Nó được lấy cảm hứng tử Lotus Noted và nó cũng có kiểu lưa trữ tương tự key –value stores. Mô hình Document database nó giống như tập hợp các tài liệu , trong các tài liệu nó có các key - value. Nhưng document này được lưa ở định dạng JSON. Trong document có nhiểu cặp key-value và các cặp key-value này được phép lồng nhau. Mỗi document sẽ có một key, kiểu cơ sở dữ liệu document này hộ trợ truy vấn hiệu quả hơn.



Hình . Key-Value và Document



Hình 1.3 . Document Store

− 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

− 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,...

### Column Family Stores

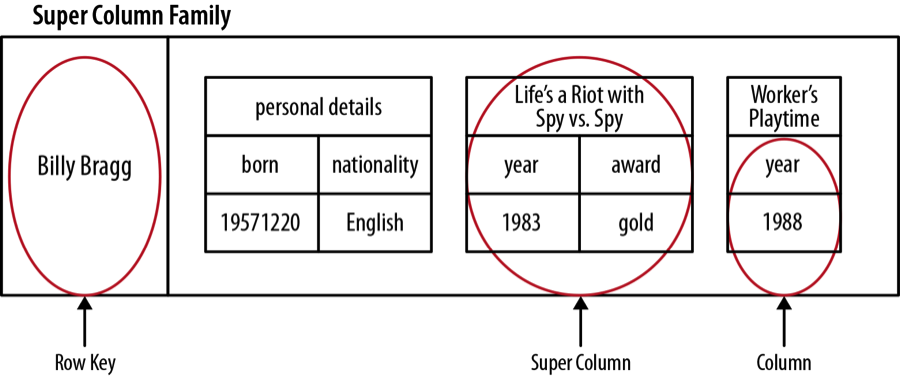
Cơ sở dữ liệu Column Family Stores được tạo ra để lưa và xử lý một lượng dữ liệu lớn trên nhiều server. Cơ sở dữ liệu Column-family lưa trữ dữ liệu trong nhiều cột trong mỗi dòng với key cho từng dòng. Column families là một nhóm dữ liệu các dữ liệu liên quan được truy cập cùng nhau.

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 .



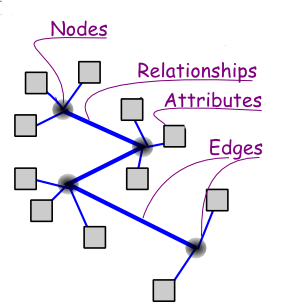
Hình 1.4. Super Column Family

.

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,…

### Graph Database

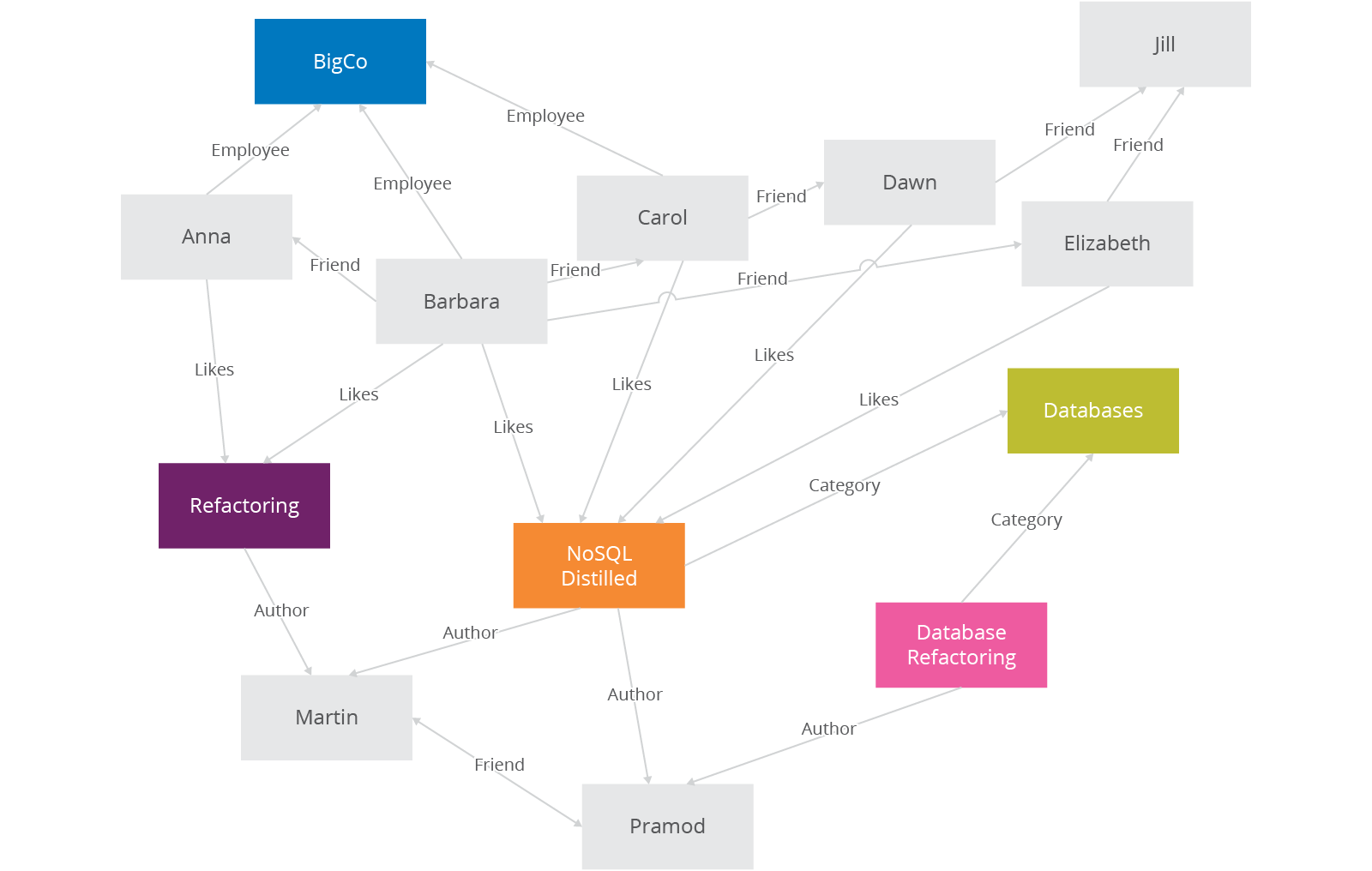
Cơ sở dữ liệu đồ thị là một cơ sở dữ liệu dạng đồ thị sử dụng các cấu trúc đồ thị với các nút ( nodes) các mối quan hệ ( relationships) , các thuộc tính (properties) để mô tả và lưa trữ dữ liệu



Hình 1.5. Graph Database Store

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.



Hình 1.6. NoSQL Graph Database

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 .

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

## So sánh NoSQL và SQL.

|  |  |  |
| --- | --- | --- |
| 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 2. So sánh SQL và NoSQL

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 |

Bảng 3. So sánh RDBMS và NoSQL

## Ưa điểm và nhược điểm của Nosql Database

## Ưa Điểm

## Open source : Hầu hết các sản phẩm nguồn mở được đưa ra cho người phát triển với nhiều lợi ích, sử dụng miễn phí

## Khả năng mở rộng linh hoạt : do không bị rằng buộc chặt chẽ mối quan hệ, cấu trúc lưa trữ nên khả năng mở rộng rất linh động

## Các cơ sở dữ liệu khác nhau cho những dự án khác nhau: Mỗi loại NoSQL cụ thể sẽ là giải pháp phục vụ cho một vấn đề cụ thể

## NoSQL được các hãng lớn 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 : Những yêu cầu về lưa trữ của công nghệ đám mây với NoSQL là sự trùng khớp .

## Nhược điểm

## − Hộ trợ không đồng đều cho doanh nghiệp: trong khi các nhà cung cấp RDBMS thường hộ trợ tốt cho khách hàng. Thì các nhà cung cấp nguồn mở mới thành lập không thể được mong đợi sẽ cung cấp hỗ trợ tốt hơn.

## − Chưa đủ “ chín “ cho các doanh nghiệp: Vấn đề lớn của NoSQL là thiếu độ chín muồi và các vấn đề tính không ổn định .

## − Những hạn chế về tri thức nghiệp vụ : Cơ sở dữ liệu NoSQL không có nhiều sự đeo bám tới các công cụ BI , trong khi những yêu cầu và phân tích hiện đại đơn giản thì cũng liên quan nhiều tới sự tinh thông về lập trình.

## − Thiếu sự tinh thông: Tính mới mẻ của NoSQL khiến nhiều lập trình viên và người quản trị biết tới công nghệ này. Như vậy sẽ gặp nhiều khó khăn trong việc tìm người phù hợp

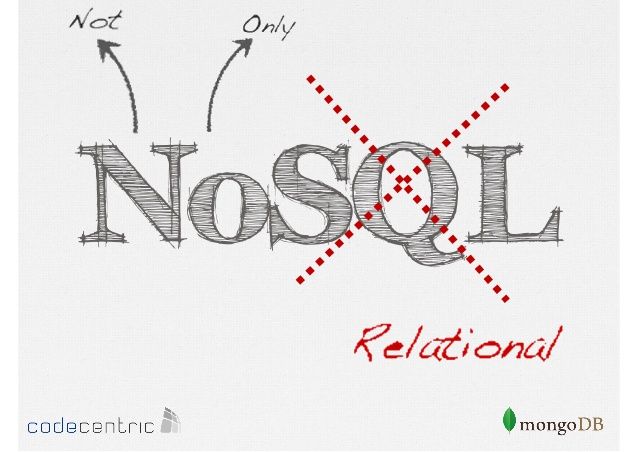
## − Những vẫn đề về tương thích: Mỗi cơ sở dữ liệu NoSQL có các API riêng các giao diện truy vấn riêng, và sự thiếu hụt tiêu chuẩn sẽ gây ra nhiều khó khăn trong việc chuyển giao giữa các công ty với nhau.

## MongoDb và một số Thuật ngữ

## Khái niệm

MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.

MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng



* + 1. Một số Thuật ngữ hay sử dụng trong MongoDB
* \_id - Là trường bắt buộc có trong mỗi document. Trường \_id đại diện cho một giá trị duy nhất trong document MongoDB. Trường \_id cũng có thể được hiểu là khóa chính trong document.
* Collection - Là nhóm của nhiều document trong MongoDB. Collection có thể được hiểu là một bảng tương ứng trong cơ sở dữ liệu RDBMS (Relational Database Management System). Collection nằm trong một cơ sở dữ liệu duy nhất. Các collection không phải định nghĩa các cột, các hàng hay kiểu dữ liệu trước
* Cursor - Đây là một con trỏ đến tập kết quả của một truy vấn. Máy khách có thể lặp qua một contrỏ để lấy kết quả
* Database - Nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* Document - Nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* Field - Là một cặp name – value trong một document. Một document có thể có không hoặc nhiều trường. Các trường giống các cột ở cơ sở dữ liệu quan hệ
* JSON -. Con người có thể đọc được ở định dạng văn bản đơn giản thể hiện cho các dữ liệu có cấu trúc. Hiện tại JSON đang hỗ trợ rất nhiều ngôn ngữ lập trình.
* Index - Là những cấu trúc dữ liệu đặc biệt, dùng để chứa một phần nhỏ của các tập dữ liệu một cách dễ dàng để quét. Chỉ số lưu trữ giá trị của một fields cụ thể hoặc thiết lập các fields, sắp xếp theo giá trị của các fields này
  1. **Các kiểu dữ liệu trong MongoDB**

MongoDb hộ trợ nhiều kiểu dữ liệu :

|  |  |  |
| --- | --- | --- |
| Type | Number | Alias |
| Double | 1 | “ double” |
| String | 2 | “ string” |
| Object | 3 | “object” |
| Array | 4 | “array” |
| Binary data | 5 | “binData” |
| Undefined | 6 | “undefined” |
| Boolen | 8 | “bool” |
| Data | 9 | “date” |
| Null | 10 | “null |

Bảng : Một số kiểu dữ liệu phổ biến

* Chuỗi : Đây là kiểu dữ liệu được sử dụng phổ biến nhất để lưu giữ dữ liệu
* Số nguyên : Kiểu dữ liệu này được sử dụng để lưu một giá trị số
* Boolean : Kiểu dữ liệu này được sử dụng để lưu giữ một giá trị Boolean (true/false).
* Double : Kiểu dữ liệu này được sử dụng để lưu các giá trị số thực dấu chấm động.
* Mảng : Kiểu dữ liệu này được sử dụng để lưu giữ các mảng hoặc danh sách hoặc nhiều giá trị vào trong một key
* Object : Kiểu dữ liệu này được sử dụng cho các Document được nhúng vào
* Null : Kiểu dữ liệu này được sử dụng để lưu một giá trị Null
* Date: Kiểu dữ liệu này được sử dụng để lưu giữ date và time hiện tại trong định dạng UNIX time
* Code : Kiểu dữ liệu này được sử dụng để lưu giữ JavaScrip code vào trong Document
  1. **Một số câu lệnh dùng trong MongoDb**
     1. **MongoDB – Tạo Database**

Lệnh use trong MongoDB

Lệnh **use DATABASE\_NAME** trong MongoDB được sử dụng để tạo cơ sở dữ liệu. Lệnh này sẽ tạo một cơ sở dữ liệu mới, nếu nó chưa tồn tài, nếu không thì, lệnh này sẽ trả về cơ sở dữ liệu đang tồn tại.

Cú pháp : use **DATABASE\_NAME**

Ví dụ : tạo một cơ sở dữ liệu tên **mydb**

**use mydb**

* + 1. **MongoDB – Xóa Database**

Phương thức **dropDatabase()** trong MogoDB

Lệnh **db.dropDatabase()** trong MongoDB được dùng để xóa một cơ sở dữ liệu đang tồn tại.

Vd: Nếu bạn muốn xóa 1 cơ sở dữ liệu **mydb** có sẵn

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

* + 1. **MongoDB – Tạo Collection**

Phương thức createCollection() trong MongoDB

Phương thức **db. createCollection(name, options)** trong MongoDB được sử dụng để tạo Collection. Với name kiểu chuỗi là tên Collection, options là một Document

>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )

{ "ok" : 1 }

>

* + 1. **MogoDB - Xóa Collection**

Phương thức drop() trong MongoDB

Phương **thức db. Collection.drop()** được sử dụng để xóa một Collection từ cơ sở dữ liệu.

db.COLLECTION\_NAME.drop()

Phương thức drop() sẽ trả về true , nếu Collection đã chọn xóa thành công, nếu không sẽ trả về false.

>db.mycollection.drop()

true

>

* + 1. **MongoDB – Chèn Document**

Để chèn dữ liệu vào trong Collection trong MongoDB sử dụng phương thức **insert()** hoặc **save()**

>db.COLLECTION\_NAME.insert(document)

**Ví dụ :**

>db.mycol.insert({

\_id: ObjectId(7df78ad8902c),

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

})

* + 1. **MongoDB – Truy vấn Document**

Để truy vấn dữ liệu từ Collection trong MongoDB ta sử dụng phương thức find()

Cú pháp:

>db.COLLECTION\_NAME.find()

Để hiển thị các kết quả theo một cách đã được định dạng, bạn có thể sử dụng phương thức **pretty()**.

Cú pháp :

>db.mycol.find().pretty()

Ngoài phương thức find(), trong MongoDB còn có phương thức findOne() sẽ chỉ trả về một Document

* + 1. **MongoDB- Cập nhập Document**

Phương thức **update()** hoặc **save()** trong MongoDB được sử dụng để cập nhật Document vào trong một Collection. Phương thức update() cập nhật các giá trị trong Document đang tồn tại trong khi phương thức save() thay thế Document đang tồn tại với Document đã truyền trong phương thức save() đó.

* Phương thức update() cập nhật các giá trị trong Document đang tồn tại

Cú pháp :

>db.COLLECTION\_NAME.update(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

* Phương thức save() thay thế Document đang tồn tại với Document mới đã được truyền trong phương thức save() này.

Cú pháp :

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

* + 1. **MongoDB - Xóa Document**

Phương thức remove() trong MongoDB được sử dụng để xóa Document từ Collection. Phương thức remove() nhận hai tham số. Tham số đầu tiên deletion criteria xác định Document để xóa, và tham số thứ hai là justOne.

* **deletion criteria :** (Tùy ý) Xác định Document để xóa.
* **justOne :** (Tùy ý) Nếu được thiết lập là true hoặc 1, thì chỉ xóa một Document.

Cú Pháp :

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

* Nếu có nhiều bản ghi và bạn chỉ muốn xóa bản ghi đầu tiên, thì thiết lập tham số **justOne** trong phương thức remove()

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

* Xóa tất cả Document

>db.mycol.remove()

>db.mycol.find()

# CHƯƠNG II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

# Tổng quan về Node.js

### Node.js là gì ?

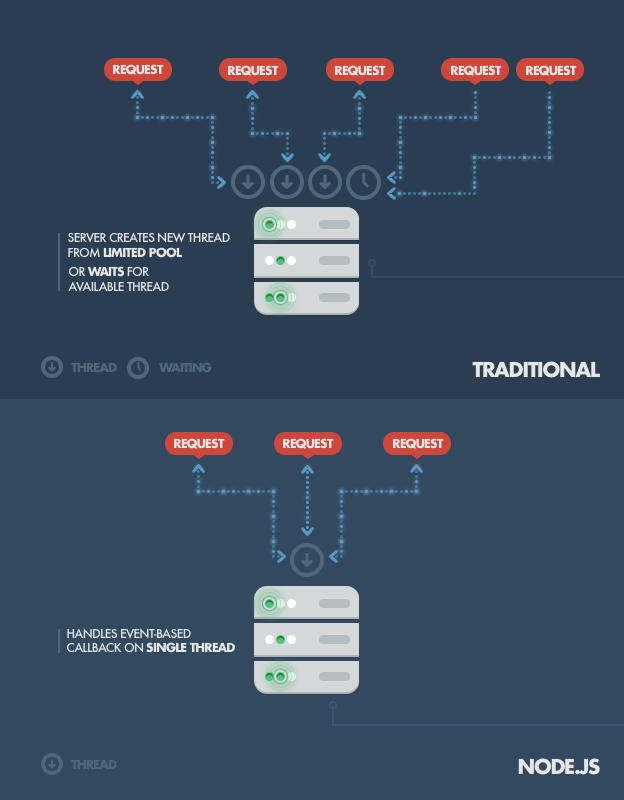
NodeJS là một mã nguồn được xây dựng dựa trên nền tảng Javascript V8 Engine, nó được sử dụng để xây dựng các ứng dụng web như các trang video clip, các forum và đặc biệt là trang mạng xã hội phạm vi hẹp. NodeJS là một mã nguồn mở được sử dụng rộng bởi hàng ngàn lập trình viên trên toàn thế giới. NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ WIndow cho tới Linux, OS X nên đó cũng là một lợi thế. NodeJS cung cấp các thư viện phong phú ở dạng Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất

### Các đặc tính của Node.js

* **Không đồng bộ**: Tất cả các API của NodeJS đều không đồng bộ (none-blocking), nó chủ yếu dựa trên nền của NodeJS Server và chờ đợi Server trả dữ liệu về. Việc di chuyển máy chủ đến các API tiếp theo sau khi gọi và cơ chế thông báo các sự kiện của Node.js giúp máy chủ để có được một phản ứng từ các cuộc gọi API trước (Realtime).
* **Chạy rất nhanh**: NodeJ được xây dựng dựa vào nền tảng V8 Javascript Engine nên việc thực thi chương trình rất nhanh.
* **Đơn luồng nhưng khả năng mở rộng cao**: Node.js sử dụng một mô hình luồng duy nhất với sự kiện lặp. cơ chế tổ chức sự kiện giúp các máy chủ để đáp ứng một cách không ngăn chặn và làm cho máy chủ cao khả năng mở rộng như trái ngược với các máy chủ truyền thống mà tạo đề hạn chế để xử lý yêu cầu. Node.js sử dụng một chương trình đơn luồng và các chương trình tương tự có thể cung cấp dịch vụ cho một số lượng lớn hơn nhiều so với yêu cầu máy chủ truyền thống như Apache HTTP Server.
* **Không đệm**: NodeJS không đệm bất kì một dữ liệu nào và các ứng dụng này chủ yếu là đầu ra dữ liệu.
* **Có giấy phép**: NodeJS đã được cấp giấy phép bởi MIT License..

### Node.js làm việc như thế nào

Ý tưởng chính của Node js là sử dụng non-blocking, hướng sự vào ra dữ liệu thông qua các tác vụ thời gian thực một cách nhanh chóng. Bởi vì, Node js có khả năng mở rộng nhanh chóng, khả năng xử lý một số lượng lớn các kết nối đồng thời bằng thông lượng cao. Nếu như các ứng dụng web truyền thống, các request tạo ra một luồng xử lý yêu cầu mới và chiếm RAM của hệ thống thì việc tài nguyên của hệ thống sẽ được sử dụng không hiệu quả. Chính vì lẽ đó giải pháp mà Node js đưa ra là sử dụng luồng đơn (Single-Threaded), kết hợp với non-blocking I/O để thực thi các request, cho phép hỗ trợ hàng chục ngàn kết nối đồng thời.



* + 1. NPM : The Node Package Manager

Một số các module NPM phổ biến nhất hiện nay là:

* [expressjs.com/](http://expressjs.com/) - Express.js, một Sinatra-inspired web framework khá phát triển của Node.js, chứa rất nhiều các ứng dụng chuẩn của Node.js ngày nay.
* [connect](https://github.com/senchalabs/connect#readme) - Connect là một mở rộng của HTTP server framework cho Node.js, cung cấp một bộ sưu tập của hiệu suất cao "plugins" được biết đến như là trung gian; phục vụ như một nền tảng cơ sở cho Express
* [socket.io](http://socket.io/) and [sockjs](https://github.com/sockjs) - Hai thành phần Server-side websockets components nổi tiếng nhất hiện nay.
* [Jade](http://jade-lang.com/) - Một trong những engines mẫu, lấy cảm hứng từ HAML, một phần mặc định trong Express.js.
* [mongo](https://www.npmjs.com/package/mongodb) and [mongojs](https://github.com/mafintosh/mongojs) - MongoDB hàm bao để cung cấp các API cho cơ sở dữ liệu đối tượng trong MongoDB Node.js
* [redis](https://github.com/NodeRedis/node_redis) - thư viện Redis client.
* [coffee-script](https://www.npmjs.com/package/coffee-script) - CoffeeScript trình biên dịch cho phép developers viết các chương trình Node.js của họ dùng Coffee.
* [underscore](https://www.npmjs.com/package/underscore) ([lodash](https://lodash.com/), [lazy](https://www.npmjs.com/package/lazy)) - Thư viện tiện ích phổ biến nhất trong JavaScript, package được sử dụng với Node.js, cũng như hai đối tác của mình, hứa hẹn hiệu suất tốt hơn bằng cách lấy một cách tiếp cận thực hiện hơi khác nhau.
* [forever](https://www.npmjs.com/package/forever) - Có lẽ là tiện ích phổ biến nhất để đảm bảo rằng một kịch bản nút cho chạy liên tục. Giữ quá trình Node.js của bạn lên trong sản xuất đối mặt với bất kỳ thất bại không ngờ tới.

## Ưa điểm và nhược điểm

### 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 & HTML5 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,…

* + 1. Nhược điểm

Ứng dụng nặng tốn tài nguyên Nếu bạn cần xử lý các ứng dụng tốn tài nguyên CPU như encoding video, convert file, decoding encryption… hoặc các ứng dụng tương tự như vậy thì không nên dùng NodeJS (Lý do: NodeJS được viết bằng C++ & Javascript, nên phải thông qua thêm 1 trình biên dịch của NodeJS sẽ lâu hơn 1 chút ). Trường hợp này bạn hãy viết 1 Addon C++ để tích hợp với NodeJS để tăng hiệu suất tối đa !

NodeJS và ngôn ngữ khác NodeJS, PHP, Ruby, Python .NET …thì việc cuối cùng là phát triển các App Web. NodeJS mới sơ khai như các ngôn ngữ lập trình khác. Vậy nên bạn đừng hi vọng NodeJS sẽ không hơn PHP,Ruby,Python… ở thời điểm này. Nhưng với NodeJS bạn có thể có 1 ứng dụng như mong đợi, điều đó là chắc chắn !

* 1. **Installtion**

### 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 Ubuntu/CentOS thì bạn sử dụng package Manager , bật cửa sổ terminal và Type:

1. **Cài đặt từ Packege**

Tải package Linux Binaries 64bit từ [NodeJS Download](https://nodejs.org/en/download/)

# wget https://nodejs.org/dist/v8.9.3/node-v8.9.3-linux-x64.tar.xz

Giải nén nội dung bên trong vào /usr/local

# tar --strip-components 1 -xJvf node-v8.9.3-linux-x64.tar.xz -C /usr/local

Kiểm tra lại phiên bản NodeJS

# node –version

v8.9.3

1. **Cài đặt từ Source Code**

Cài đặt trình biên dịch

# apt-get update  
# apt-get update

# apt-get install make g++ libssl-dev git

– Tải source code từ [NodeJS Download](https://nodejs.org/en/download/).

# wget https://nodejs.org/dist/v8.9.3/node-v8.9.3.tar.gz

# tar -xzvf node-v8.9.3.tar.gz

# cd node-v8.9.3/

– Cấu hình và biên soạn mã nguồn (tốn 10-20 phút tùy cấu hình VPS)

# ./configure

# make

– Biên soạn thành công, tiến hành cài đặt

# make install

– Kiểm tra lại phiên bản NodeJS

# node --version

v8.9.3

1. **Cài đặt từ Repository**

− Cài đặt NodeJS và công cụ NPM

# apt-get update

# apt-get install nodejs npm

– Kiểm tra lại phiên bản NodeJS

# nodejs --version

v4.2.6

# npm --version

3.5.2

1. **Cài đặt sử dụng Node Version**

Nếu bạn muốn cài đặt NodeJS một cách linh hoạt, hãy sử dụng Node Version Manager(NVM). Phần mềm này cho phép cài đặt và sử dụng độc lập cùng lúc nhiều phiên bản khác nhau của NodeJS cùng các package liên quan.  
– Truy cập [NVM Github](https://github.com/creationix/nvm) và copy lệnh chạy

# curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.8/install.sh | bash

– Để sử dụng, bạn cần source phần .bash\_profile:

# source /root/.bashrc

### 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.

$ npm install <Module Name>

Ví dụ: Dưới đây là lệnh để cài đặt một module rất phổ biến trong Node.js, đó là **express** Module:

$ npm install express

Bây giờ để sử dụng môđun này trong file js của bạn, sử dụng cú pháp sau đây:

var express = require('express');

## Khái niệm về Callback trong Node.js

Callback có tính chất không đồng bộ tương đương cho một hàm. Một hàm callback được gọi khi hoàn thành một nhiệm vụ cụ thể. Tất cả các API của Node đều được viết theo các cách của hàm callback.

Ví dụ, một hàm để đọc file bắt đầu với việc đọc file và trả về phần điều khiển để môi trường thực hiện quyết định lệnh tiếp theo có thể thực thi. Khi phần I/O (đọc/ghi) file được hoàn thành, nó sẽ gọi về một hàm callback, với nội dung của file là tham số. Do đó sẽ không có blocking hoặc chờ khi đọc/ghi File. Nó làm cho Node.js có hiệu năng cao hơn, như có số lượng request cao hơn mà không cần phải chờ kết quả trả về.

* Ví dụ Blocking Code

Tạo một dòng text với input.txt với nội dung

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

Tạo một file main.js với nội dụng

var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());

console.log("Ket thuc chuong trinh");

Kết quả chạy chương trình

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

Ket thuc chuong trinh

* Ví dụ Non-Blocking Code

Tạo 1 file với tên input.txt với nội dung

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

* Hàm main.js

ar fs = require("fs");

fs.readFile('input.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("Ket thuc chuong trinh");

Kết quả khi chạy chương trình :

Ket thuc chuong trinh

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

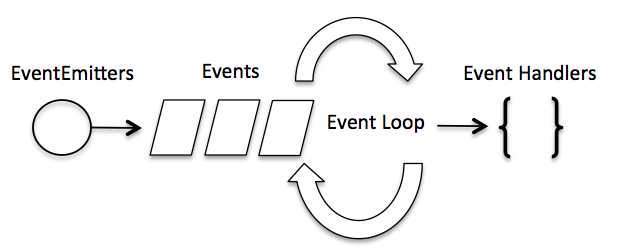
Hai ví dụ trên giải thích định nghĩa cách gọi blocking và non-blocking. Ví dụ đầu tiên chỉ ra rằng chương trình khóa cho đến khi nó đọc file và chỉ tiếp tục chạy vài giây sau đó, chương trình thứ 2 không đợi cho việc đọc file và tiếp tục in "Ket thuc chuong trinh" cùng thời điểm thực hiện chương trình.

### Khái niệm Event Loop

Node.js là ứng dụng đơn luồng nhưng có hỗ trợ việc xử lí đồng thời thông qua các định nghĩa về sự kiện và callback. Như tất cả các API của Node.js có tính chất không đồng bộ và được xử lí đơn luồng, nó sử dụng hàm **async** để duy trì sự đồng thời. Node.js sử dụng Observer Pattern. Các Thread trong Node.js giữ một Event Loop và bất cứ khi nào có tác vụ nào hoàn thành, nó sẽ kích hoạt sự kiện tương ứng để báo cho Event Listener sẵn sàng thực hiện..

Node.js sử dụng rất nhiều sự kiện, đó là lý do tại sao Node.js khá nhanh so với các sản phẩm công nghệ khác. Ngay khi Node khởi động Server của nó, nó sẽ nhanh chóng khởi tạo các biến, khai báo các hàm và sau đó đơn giản là chờ đợi các sự kiện xảy ra.

Trong ứng dụng xử lý sự kiện, nhìn chung vòng lặp chính lắng nghe các sự kiện, và sau đó trigger đến hàm callback khi một trong những sự kiện được phát hiện.



Trong khi các Event là khá tương tự như các hàm callback . Điểm khác nhau nằm ở chỗ, hàm callback gọi khi một hàm không đồng bộ và trả về kết quả của nó trong khi phần xử lí sự kiện làm việc trên Observer Pattern. Hàm này sẽ lắng nghe các sự kiện, đóng vai trò như một Observers (Người quan sát). Bất cứ khi nào một sự kiện phát sinh, các hàm Listener của nó sẽ bắt đầu thực thi. Node.js có nhiều sự kiện có sẵn thông qua **events** Module và lớp **EventEmitter**có thể dựa vào để bind sự kiện và lắng nghe sự kiện.

Trước khi sử dụng event Module, bạn sử dụng phương thức require() để khai báo như sau:

// Khai bao events module

var events = require('events');

// Tao mot doi tuong eventEmitter

var eventEmitter = new events.EventEmitter();

Sau đó, để gắn kết Event Handler với một sự kiện, bạn sử dụng cú pháp sau:

// Gan ket event voi Event Handler nhu sau:

eventEmitter.on('eventName', eventHandler);

Bạn có thể kích hoạt một sự kiện bởi sử dụng phương thức emit() của EventEmitter:

// Kich hoat mot event

eventEmitter.emit('eventName');

Ví dụ minh họa Event Loop trong Node.js

// Khai bao events module

var events = require('events');

// Tao mot doi tuong eventEmitter

var eventEmitter = new events.EventEmitter();

// Tao mot Event Handler nhu sau:

var connectHandler = function connected() {

console.log(' ket noi thanh cong!');

// Kich hoat su kien data\_received

eventEmitter.emit('data\_received');

}

// Gan ket su kien connection voi Event Handler

eventEmitter.on('connection', connectHandler);

// Gan ket su kien data\_received voi mot ham an danh

eventEmitter.on('data\_received', function(){

console.log('Du lieu duoc tiep nhan thanh cong.');

});

// Kich hoat su kien connection

eventEmitter.emit('connection');

console.log("Ket thuc chuong trinh.");

Kết quả của chạy chương trình :

ket noi thanh cong!

Du lieu duoc tiep nhan thanh cong.

Ket thuc chuong trinh.

* 1. **Lớp Event Emitter**

Nhiều đối tượng trong Node.js sinh ra các sự kiện, ví dụ **net.Server** sinh ra một sự kiện mỗi khi có một kết nối ngang hàng đến nó, hay **fs.readStream** sinh ra sự kiện khi một file được mở. Tất cả các đối tượng này đều là sự thể hiện của lớp **events.EventEmitter** trong Node.js.

* Lớp EventEmitter trong Node.js

Lớp EventEmitter nằm trong events Module. Lớp này được truy cập qua cú pháp sau:

// Khai bao events Module

var events = require('events');

// Tao mot doi tuong eventEmitter

var eventEmitter = new events.EventEmitter();

Khi một EventEmitter gặp bất kì lỗi nào, nó sẽ sinh ra một Error Event. Khi một Listener mới được thêm, sự kiện 'newListener' sẽ được kích hoạt và một Listener sẽ bị loại bỏ, sự kiện 'removeListener' sẽ được kích hoạt.

Event Emitter cung cấp nhiều thuộc tính như **on** hay **emit**. Thuộc tính *on* được sử dụng để gắn kết một hàm với sự kiện, và *emit* dược sử dụng để kích hoạt một sự kiện.

## Các phương thức của lớp EventEmitter trong Node.js

|  |  |
| --- | --- |
| Stt | Phương thức & Miêu tả |
| 1 | **addListener(event, listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 2 | **on(event, listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 3 | **once(event, listener)** Thêm một One-Time Listener cho sự kiện. Listener dạng này sẽ chỉ được gọi khi sự kiện được kích hoạt, sau đó nó sẽ bị xóa |
| 4 | **removeListener(event, listener)** Xóa một Listener ra khỏi mảng các Listener cho một sự kiện nào đó. |
| 5 | **removeAllListeners([event])** Xóa tất cả Listener của một sự kiện |
| 6 | **setMaxListeners(n)** Theo mặc định, lớp EventEmitters sẽ in một lời cảnh báo nếu bạn thêm nhiều hơn 10 Listener cho một sự kiện cụ thể. Việc này khá hữu ích, bởi vì nó sẽ giúp tìm ra các lỗi gây rò rỉ bộ nhớ. Tất nhiên, không phải tất cả các Emitters đều cần được giới hạn với con số là 10. Hàm này cho phép bạn tăng con số đó. Thiết lập nó về 0 để không giới hạn lượng Listener cần thêm |
| 7 | **listeners(event)** Trả về một mảng bao gồm các Listener cho một sự kiện cụ thể nào đó |
| 8 | **emit(event, [arg1], [arg2], [...])** Thực thi từng Listener với các tham số đã cho. Trả về true nếu sự kiện có các Listener, và false nếu không có |

## Các sự kiện của lớp EventEmitter trong Node.js

|  |  |
| --- | --- |
| Stt | Sự kiện & Miêu tả |
| 1 | **newListener**   * **event** - Dạng chuỗi, biểu diễn tên sự kiện * **listener** - Tên hàm xử lý sự kiện   Sự kiện này được sinh bất cứ khi nào bạn thêm một Listener. Khi sự kiện này được kích hoạt, Listener có thể sẽ chưa được thêm vào mảng Listener của sự kiện |
| 2 | **removeListener** **event** - Dạng chuỗi, biểu diễn tên sự kiện   * **listener** - Tên hàm xử lý sự kiện   Sự kiện này xảy ra bất cứ khi nào có ai đó xóa một Listener. Khi một sự kiện được kích hoạt, Listener này chưa được xóa khỏi mảng Listener của sự kiện |

## Buffer Trong Node.js

Javascript thuần được mã hóa Unicode một cách tiện lợi nhưng không thật sự tốt với các dữ liệu nhị phân. Khi làm việc với các luồng TCP hoặc hệ thống file, cần thiết phải xử lý các luồng dữ liệu bát phân. Node.js cung cấp các lớp Buffer cho phép lưu trữ các dữ liệu thô như một mảng các số nguyên tương ứng với phần cấp phát bộ nhớ thô bên ngoài V8 heap.

Các lớp Buffer trong Node.js là các lớp toàn cục và có thể được truy cập trong ứng dụng mà không cần khai báo các Buffer Module bởi phương thức require() như các Module khác.

Buffer trong Node.js có thể được xây dựng theo nhiều cách khác nhau.

Cách 1 : Cú pháp cho việc tạo một Buffer cỡ 10:

var buf = new Buffer(10);

cách 2 : Cú pháp tạo một Buffer từ một mảng cho trước

var buf = new Buffer([10, 20, 30, 40, 50]);

Cách 3 : Cú pháp tạo một Buffer từ một chuỗi cho trước và với kiểu mã hóa tùy ý:

var buf = new Buffer("nguyen ba lam "utf-8");

Ghi dữ liệu vào Buffer trong node.js

Cú pháp để ghi một Buffer trong node.js

buf.write(string[, offset][, length][, encoding])

Phương thức sẽ trả về số lượng các byte được ghi. Nếu bộ nhớ trong buffer là không đủ để đáp ứng cho toàn bộ chuỗi, nó sẽ ghi một phần của chuỗi đó.

Đọc dữ liệu từ Buffer trong Node.js

Cú pháp : đọc dữ liệu từ Buffer

buf.toString([encoding][, start][, end])

Phương thức này giải mã và trả về một chuỗi từ dữ liệu đã được mã hóa trong Buffer bởi sử dụng bộ mã hóa cụ thể.

Chuyển đổi Buffer thành JSON trong Node.js

Cú pháp :

buf.toJSON()

Phương thức này trả về một biểu diễn JSON cho đối tượng Buffer đã cho.

Ghép nối các Buffer trong node.js

Cú pháp : Để nối ghép hai hoặc nhiều Buffer thành một Buffer trong Node.js, bạn sử dụng phương thức concat() như sau:

Buffer.concat(list[, totalLength])

Phương thức này trả về một Buffer mới.

So sánh các Buffer trong Node.js

Cú pháp : Để so sánh hai Buffer trong Node.js, bạn sử dụng phương thức compare() như sau:

buf.compare(otherBuffer);

Trả về một giá trị dạng số thể hiện Buffer này là đứng trước, sau hay cùng thứ tự với Buffer kia

Sao chép Buffer trong Node.js

Cú pháp : Để sao chép Buffer trong Node.js, bạn sử dụng phương thức copy() như sau:

buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

## chia nhỏ Buffer trong node.js

## Cú pháp :

buf.slice([start][, end])

## Stream Trong Node.js

Stream là các đối tượng cho phép bạn đọc dữ liệu từ một nguồn và ghi dự liệu đến một đích. Trong Node.js, có 4 loại Stream.

**Readable** - Là Stream được sử dụng để cho hoạt động đọc

**Writable** - Là Stream được sử dụng cho hoạt động ghi

**Duplex** - Là Stream được sử dụng cho cả mục đích ghi và đọc

**Transform** - Đây là một kiểu Duplex Stream, khác ở chỗ là kết quả đầu ra được tính toán dựa trên dữ liệu bạn đã nhập vào.

Mỗi loại Stream là một sự thể hiện của đối tượng **EventEmitter** và ném một vài sự kiện tại các thời điểm khác nhau. Danh sách sau liệt kê một số sự kiện thường được sử dụng:

**data** - Sự kiện này được kích hoạt khi dữ liệu là có sẵn cho hoạt động đọc.

**end** - Sự kiện này được kích hoạt khi không còn dữ liệu nào để đọc nữa.

**error** - Sự kiện này được kích hoạt khi xảy ra bất kỳ lỗi nào trong việc đọc và ghi dữ liệu.

**finish** - Sự kiện này được kích hoạt khi tất cả dữ liệu đã được chuyển hết tới vùng hệ thống cơ sở.

Đọc dữ liệu từ Stream trong Node.js

Tạo 1 file input.txt có nội dung

Báo cáo thực tập tốt nghiệp

Tạo 1 file main.js  Trong file này, đầu tiên bạn khai báo **fs** Module (đây là Module cho các hoạt động File I/O) bởi sử dụng phương thức require(). Sau đó sử dụng phương thức createReadStream() nhận tham số là tên của text file bạn đã tạo trước đó để đọc dữ liệu từ đó

var fs = require("fs");

var data = '';

// Tao mot Readable Stream

var readerStream = fs.createReadStream('input.txt');

// Thiet lap encoding la utf8.

readerStream.setEncoding('UTF8');

// Xu ly cac su kien lien quan toi Stream --> data, end, va error

readerStream.on('data', function(chunk) {

data += chunk;

});

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

console.log(data);

});

readerStream.on('error', function(err){

console.log(err.stack);

});

console.log("Ket thuc chuong trinh");

## Chạy chương trình kết quả thu được :

Ket thuc chuong trinh

Báo cáo thực tập tốt nghiệp

## Ghi dữ liệu tới Stream trong Node.js

var fs = require("fs");

var data = 'timhieuvelaptrinhnodejs';

// Tao mot Writable Stream

var writerStream = fs.createWriteStream('output.txt');

// Ghi du lieu toi Stream theo ma hoa utf8

writerStream.write(data,'UTF8');

// Danh dau diem cuoi cua file (end of file)

writerStream.end();

// Xu ly cac su kien lien quan toi Stream --> finish, va error

writerStream.on('finish', function() {

console.log("Ket thuc hoat dong ghi.");

});

writerStream.on('error', function(err){

console.log(err.stack);

});

console.log("Ket thuc chuong trinh");

## Chạy chương trình kết quả thu được :

Ket thuc chuong trinh

Ket thuc hoat dong ghi.

Bây giờ, bạn mở output.txt đã được tạo trong thư mục hiện tại và kiểm tra nội dung kết quả thu được:

timhieuvelaptrinhnodejs

* 1. Đọc ghi file trong Node.js

để khai báo **fs** Module để triển khai các hoạt động về File I/O trong Node.js. Cú pháp như sau:

var fs = require("fs")

Mỗi phương thức trong fs Module có các form đồng bộ và các form không đồng bộ. Các phương thức không đồng bộ nhận một tham số cuối cùng là một hàm callback thực thi khi kết thúc và nhận tham số đầu tiên là một hàm callback để xử lý lỗi. Việc sử dụng các phương thức không đồng bộ là tốt hơn các phương thức đồng bộ, bởi vì các phương thức không đồng bộ không bao giờ khóa trình thực thi chương trình trong khi phương thức đồng bộ thì có.

Mơ Một file trong node.js

Cú pháp :

fs.open(path, flags[, mode], callback)

Lấy thông tin file trong Node.js

Cú pháp :

fs.stat(path, callback)

Ghi dữ liệu file trong Node.js

fs.writeFile(filename, data[, options], callback)

Phương thức này sẽ ghi đè nếu file đã tồn tại.

Đọc dữ liệu từ file trong Node.js

Cú Pháp:

fs.read(fd, buffer, offset, length, position, callback)

Phương thức này sẽ sử dụng tham số fd (viết tắt của File Descriptor) để đọc file. Nếu bạn muốn đọc file bởi sử dụng trực tiếp tên file thì bạn nên sử dụng phương thức khác.

Đóng File trong Node.js

Cú pháp :

fs.close(fd, callback)

Xóa file trong Node.js

Cú pháp:

fs.unlink(path, callback)

Tạo thư mục trong Node.js

Cú pháp :

fs.mkdir(path[, mode], callback)

Đọc thư mục trong Node.js

Cú pháp :

fs.readdir(path, callback)

Xóa thư mục trong Node.js

Cú pháp:

fs.rmdir(path, callback)

2.10 Đối Tượng Toàn Cục

Khái niệm Toàn cục (Global) tức là mọi thứ đều có quyền truy cập đến. Trong Node.js cũng vậy, các đối tượng toàn cục là có sẵn cho tất cả Module. Chúng ta không cần khai báo hay import chúng bởi phương thức require() mà vẫn có thể sử dụng chúng một cách trực tiếp. Các đối tượng này có thể là các Module, các hàm, các chuỗi hoặc các đối tượng.

Trong Node.js, **\_\_filename** biểu diễn tên file của đoạn code đang được thực thi. Đây là đường dẫn tuyệt đối tới file chứa phần code này

Ví dụ : Để minh họa cho \_filenam trong Node.js, bạn tạo main.js có nội dung đơn giản sau:

// In gia tri cua \_\_filename trong Node.js

console.log( \_\_filename );

Kết quả : in ra chính là đường dẫn tới main.js chứa đoạn code trên:

Trong Node.js, **\_\_dirname** biểu diễn thư mục chứa đoạn code mà đang được thực thi.

Ví dụ : Để minh họa cho việc lấy thông tin của \_dirname trong Node.js, bạn tạo main.js chứa đoạn code đơn giản sau:

// In gia tri cua \_\_dirname trong Node.js

console.log( \_\_dirname );

Kết quả : in ra là thư mục hiện tại chứa main.js:

## Hàm setTimeout(cb, ms) trong Node.js

Hàm toàn cục **setTimeout(cb, ms)** được sử dụng để chạy hàm callback có tên là **cb** sau một khoảng thời gian **ms** milisecond.

Hàm toàn cục này trả về một giá trị biểu diễn Timer để có thể được xóa bởi hàm clearTimeout().

## Hàm clearTimeout(t) trong Node.js

## Hàm toàn cục clearTimeout( t ) được sử dụng để dừng một Timer đã được tạo bởi hàm setTimeout() trước đó. Tham số t chính là Timer được trả về từ hàm setTimeout().

## Hàm setInterval(cb, ms) trong Node.js

Hàm toàn cục **setInterval(cb, ms)** được sử dụng để chạy hàm callback có tên là **cb** một cách lặp đi lặp lại sau một khoảng thời gian **ms** millisecond.

Hàm toàn cục này trả về một giá trị biểu diễn Timer để có thể được xóa bởi hàm **clearInterval(t)**.

**console** trong Node.js là đối tượng toàn cục được sử dụng để in ra các mức độ thông báo khác nhau trên stdout và stderr. Đối tượng toàn cục này có sẵn nhiều phương thức được sử dụng để in thông tin mang tính cảnh báo, lỗi.

Bảng dưới trình bày các phương thức của đối tượng toàn cục console trong Node.js.

|  |  |
| --- | --- |
| Stt | Phương thức & Miêu tả |
| 1 | **console.log([data][, ...])** In một dòng mới tới stdout. |
| 2 | **console.info([data][, ...])** In một dòng mới tới stdout. |
| 3 | **console.error([data][, ...])** In một dòng mới tới stderr |
| 4 | **console.warn([data][, ...])** In một dòng mới tới stderr |
| 5 | **console.dir(obj[, options])** Sử dụng util.inspect trên obj và in chuỗi kết quả tới stdout. |
| 6 | **console.time(label)** Đánh dấu một mốc thời gian |
| 7 | **console.timeEnd(label)** Kết thúc Timer, ghi lại kết quả |
| 8 | **console.trace(message[, ...])** In tới stderr dòng thông báo 'Trace :', theo sau bởi thông báo đã được định dạng và stack trace tới vị trí hiện tại |
| 9 | **console.assert(value[, message][, ...])** Tương tự assert.ok(), nhưng thông báo lỗi ở đây được định dạng giống như phương thức util.format(message...). |

## process trong Node.js là đối tượng toàn cục và có thể được truy cập từ bất cứ nơi đâu. Đối tượng toàn cục process này cung cấp rất nhiều phương thức tiện lợi được trình bày dưới đây.

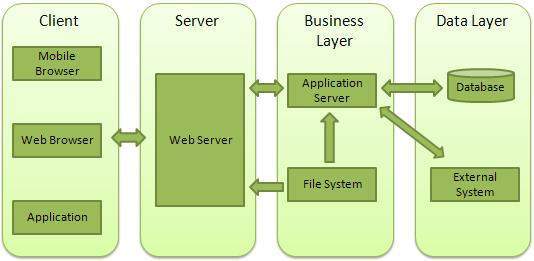
|  |  |
| --- | --- |
| Stt | Sự kiện & Miêu tả |
| 1 | **exit** Sự kiện xảy ra khi process chuẩn bị thoát. Không có cách nào để ngăn cản nó |
| 2 | **beforeExit** Sự kiện này xảy ra khi Node.js làm trống Event Loop và không còn cái gì khác để schedule nữa. |
| 3 | **uncaughtException** Sự kiện xảy ra khi xuất hiện một exception và không được bắt. Nếu một Listener được thêm vào exception này thì hành động mặc định sẽ không xảy ra |
| 4 | **Signal Events** Sự kiện xảy ra khi process nhận một tín hiệu giống như SIGINT, SIGHUP, ... |

2.10 Module Trong Node.js

Web Server là một ứng dụng phần mềm có thể xử lý các HTTP request được gửi bởi HTTP Client (ví dụ: trình duyệt web) và trả về một trang web trong phản hồi tới Client. Web Server thường gửi các tài liệu html bên cạnh các ảnh cũng như style sheet và các đoạn Javascript.

## Cấu trúc ứng dụng Web

Một ứng dụng web thường được chia thành 4 lớp như sau:



* **Client** - Lớp này bao gồm các trình duyệt web và các ứng dụng có thể tạo ra các HTTP request đến Web Server.
* **Server** - Lớp này bao gồm các Web Server có thể can thiệp các request được tạo bởi Client và trả về các phản hồi (response).
* **Business** - Lớp này bao gồm ứng dụng trên Server có thể được tận dụng bởi các Web Server để thực hiện các tiến trình xử lý cần thiết. Lớp này tương tác với lớp Data qua các chương trình bên ngoài.
* **Data** - Lớp này bao gồm các Database và bất kì các nguồn dữ liệu nào.

## Tạo Web Server bởi sử dụng Node.js

Node.js cung cấp **http** Module có thể được sử dụng để tạo các HTTP client và server. Dưới đây là phần kiến trúc thu nhỏ của HTTP Server được lắng nghe trên cổng 8081.

Đầu tiên, bạn tạo *server.js* có nội dung như sau:

var http = require('http');

var fs = require('fs');

var url = require('url');

// Tao mot Server

http.createServer( function (request, response) {

// Parse request co chua ten file

var pathname = url.parse(request.url).pathname;

// In thong tin ve ten file ma tu do Request duoc tao.

console.log("Request cho " + pathname + " da duoc nhan.");

// Doc noi dung tu File da duoc yeu cau boi Request

fs.readFile(pathname.substr(1), function (err, data) {

if (err) {

console.log(err);

// HTTP Status: 404 : NOT FOUND

// Content Type: text/plain

response.writeHead(404, {'Content-Type': 'text/html'});

}else{

// HTTP Status: 200 : OK

// Content Type: text/plain

response.writeHead(200, {'Content-Type': 'text/html'});

// Ghi noi dung cua File toi phan BODY cua Response

response.write(data.toString());

}

// Gui phan BODY cua Response

response.end();

});

}).listen(8081);

// In thong bao sau tren console

console.log('Server dang chay tai dia chi: http://127.0.0.1:8081/');

Kết quả khi chạy chương trình.:

Server dang chay tai dia chi: http://127.0.0.1:8081/

# 

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

## 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

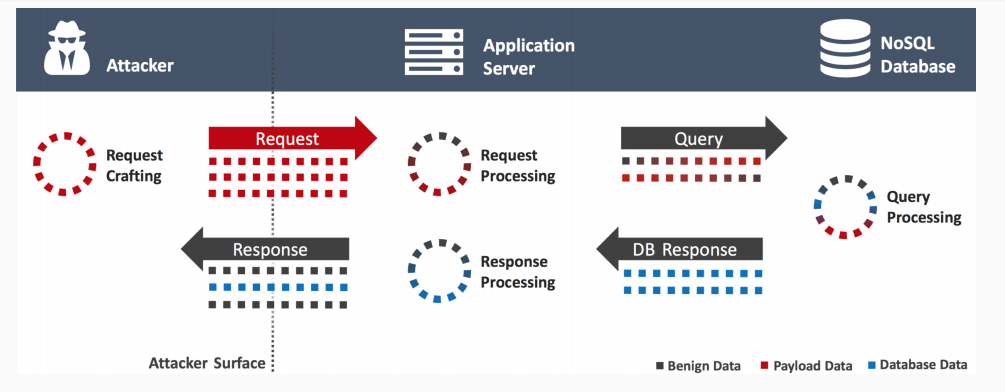
Các cơ sở dữ liệu như MongoDB hay redis là những lựa chọn đáng cân nhắc với những hệ thống lớn, một khía cạnh có lẽ không được cân nhắc đến nhất là ở việt nam đó chính là tính bảo mật của cơ sở dữ liệu này.

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.

* + 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



Hình 3.1 Cơ chế hoạt động

**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.

### Các Loại Tấn Công

1. **Chuyển định dạng string thành biểu thức điều kiện**

Bắt đầu với truy vấn trên MongoDB

db.usersdata.find({ username: "nguyenbalam", password: "123456"})

được xem như một truy vấn bình thường và tìm thấy kết quả.

Tuy nhiên, hacker có thể thay thế bằng một Biểu thức logic thay vì String, thường dùng nhất là "&ne" với ý nghĩa not equals.

Mongo.usersCollection.find({ username: {$ne: ""}, password: {$ne: ""}})

Truy vấn này luôn trả về tất cả các row trong collection mà có username và password lớn hơn rỗng. Điều này thật sự rất nguy hiểm!

Để ngăn chặn điều này, chúng ta cần:

* Đọc dữ liệu vào theo định dạng String. Framework Meteor mặc định tất cả các input text nhập vào đều ở định dạng String.
* Một lần nữa kiểm tra tất cả dữ liệu nhập vào từ người dùng. Framework Meteor cung cấp một module rất mạnh đó là check. Nó có thể kiểm tra xem dữ liệu bạn đưa vào có phải dự liệu bạn mong muốn hay không.

VD: check(username, String)

Function này sẽ xuất hiện Exception nếu như username không phải là String và không cho chạy đoạn code phía sau. Nếu username là String thì tiếp tục chạy như bình thường.

1. **Nối chuỗi trong query**

Nếu như dev viết một câu truy vấn dạng này:

var string query = “{ username: ‘“ + user + “’, password: ‘” + password + “’ }”

Đây rõ ràng có thể thực hiện tương tự như SQL Injection mà chúng ta đã được học. Nếu:

username = admin’, $or: [ {}, { ‘a’:’a

password = ’ } ], $comment:’hacked’

thì câu query sẽ trở thành

{ username: ‘admin’ , $or: [ {}, { ‘a’: ‘a’, password: ‘’ } ], $comment: ‘hacked’ }

Và hacker đã bypass thành công bước Authentication.

Tương tự như phần 1, chúng ta cần quy định cứng dữ liệu nhập vào & một lần nữa kiểm tra chúng. Ngoài ra, hãy sử dụng thư viện Publish/Subcribe mà Meteor cung cấp. Publish/Subcribe giúp bạn quản lý dữ liệu được truyền từ server & nhận tại client. Chỉ truyền những thứ cần thiết đến client, không truyền tất cả. Điều này giúp giảm thiểu được nguy cơ bị khai thác dữ liệu sau khi cuộc tấn công NoSQL Injection.

# 

# CHƯƠNG IV : XÂY DỰNG ỨNG DỤNG WEB ĐƠN GIẢN BẰNG NODE.JS NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL

## Tổng quan về ứng dụng Facebook Massenger



Hình 4.1 Cấu trúc ứng dụng Facebook Messenger

### Các công nghệ sử dụng trong việc xây dựng ứng dụng

* Môi trường : Node.js
* Cơ sở dữ liệu : MongoDB
* Framework: Meteor

### Một số tính năng của ứng dụng Facebook Massenger

* Đăng nhập/ Đăng kí
* Chat với nhau trong một group chat
* Kết nối và lưa trữ database bằng MongoDB
* Quản lý user và group hiện tại bằng session
* Phân biệt các dòng chat
* Ngăn chặn Injection Vào NoSQL

## Cài đặt một số tính năng tiêu biểu

### Đăng nhập và Đăng kí

Tiến hành cài đặt trong file App.js

class App extends Component {

username\_handleKeyPress(event) {

if (event.which === 13 && event.currentTarget.value.trim() != "") {

$('.password').focus();

}

}

password\_handleKeyPress(event) {

if (event.which === 13 && event.currentTarget.value.trim() != "") {

if ($('.username').val().trim() === ""){

$('.username').focus();

}

else{

var username =ReactDOM.findDOMNode(this.refs.userName).value.trim();

var password = ReactDOM.findDOMNode(this.refs.passWord).value.trim();

check(username, String);

check(password, String);

Meteor.call('my-login', {

username: username,

password: password

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else {

if (res === true){

Session.set('currentUser', username); }

else if (res === false){

Meteor.call('my-check-username', {

username: username

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else{

if (res === true){

alert("Password is wrong!");

}

else if (res === false){

var choose = confirm("Do you want to register?");

if (choose){

Meteor.call('register', {

username: username,

password: password

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else{

alert("Done! Please login again!");

}

});

}

}

}

});

}

}

});

}

}

}

### Lưa trữ và kết nối MongoDB

Tiến hành cài đặt trong file connect-db.js

import { Mongo } from 'meteor/mongo';

export const groupsCollection = new Mongo.Collection('groupsdata');

export const messagesCollection = new Mongo.Collection('messagesdata');

if (Meteor.isServer) {

Meteor.publish('groupsdata', function tasksPublication() {

return groupsCollection.find();

});

Meteor.publish('messagesdata', function tasksPublication(currentRoom) {

return messagesCollection.find({ room: currentRoom}, { sort: { time: 1 }});

});

}

### Quản lý user và Group chat bằng session

Tiến hành cài đặt trong file App.js

Khi người dùng đăng nhập thành công thì chúng ta tạo Session currentUser bằng cú pháp:

Session.set('currentUser', username);

Khi người dùng đăng xuất chúng ta xóa Session currentUser bằng cú pháp:

Session.set('currentUser', undefined);

### Ngăn chặn tấn công Injection vào NoSQL

Tiến hành cài đặt trong file App.js

Khi người dùng nhập dữ liệu vào trường Username Và Password chúng ta tiến hành kiểm tra dữ liệu nhập vào có đúng chuẩn hay không bằng hàm check theo cú pháp.

check(username, String);

check(password, String);

# 

# CHƯƠNG V: KẾT LUẬN

## Kết quả đạt được

Đã xây dựng được một ứng dụng chat nhóm an toàn phòng chống được Injection vào cơ sở dữ liệu cho phép các thành viên trong nhóm chat với nhau mà không phải lo sợ về tính bảo mật.

Trong quá trình thực tập , bản thân em đã học hỏi và tìm hiểu thêm được một ngôn ngữ lập trình mới và những cơ sở dữ liệu mới.

## Hạn chế

Tuy nhiên, do thời gian ngắn và khả năng lập trình của em không tốt lắm cộng với số lượng công việc cũng như tìm hiểu các công nghệ mới cho nên đề tài của em chưa được hoàn thiện về mặt thực hành. Các chức năng chưa được đầy đủ như dự kiến. Mong nhận được nhiều ý kiến từ thầy cô và giúp em hoàn thiện sản phẩm hơn.

# 

**DANH MỤC TÀI LIỆU THAM KHẢO**

**Danh Mục**

1. <https://www.owasp.org/images/e/ed/GOD16-NOSQL.pdf>
2. <https://scotch.io/@401/mongodb-injection-in-nodejs>
3. <https://zanon.io/posts/nosql-injection-in-mongodb>
4. <https://technopy.com/mongodb-injection-how-to-hack-mongodb-html/>
5. <https://viblo.asia/p/tim-hieu-ve-nosql-Zzb7vDNYMjKd>
6. <https://kipalog.com/posts/NOSQL-co-dong-nghia-voi-NO-Injection>
7. <https://viblo.asia/p/tim-hieu-ve-nosql-Zzb7vDNYMjKd>
8. <https://vietjack.com/nodejs/index.jsp>
9. <https://blog.sqreen.io/mongodb-will-not-prevent-nosql-injections-in-your-node-js-app/?utm_content=buffer25729&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer>