TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÀI TẬP CHUYÊN ĐỀ
WEB VÀ DỊCH VỤ TRỰC TUYẾN

Tìm hiểu và lập trình NodeJS

Bạch Văn Hải- CNTT1 20101464
Nguyễn Đức Tuấn-CNTT2 20102430

Giáo viên hướng dẫn :

Tạ Tuấn Anh

HÀ NỘI
Ngày 17 tháng 3 năm 2013

Mục lục

Chương 1

Lời nói đầu

Công nghệ web đang có những bước phát triển mạnh mẽ. Sự phát triển bùng nổ của mạng xã hội và các dịch vụ thời gian thực thúc đẩy nhiều công nghệ mới ra đời, giúp tăng cường khả năng tương tác. Trong phần nghiên cứu này, chúng em xin đề cập đến một công nghệ rất mới, NodeJS. Được giới thiệu từ năm 2010,đây là một công nghệ mạnh mẽ giúp xây dựng các ứng dụng hiệu năng cao, có khả năng tương tác người dùng mạnh mẽ.

Chương 2

Công nhệ NodeJS

:

2.1 Mô hình WebServer truyền thống

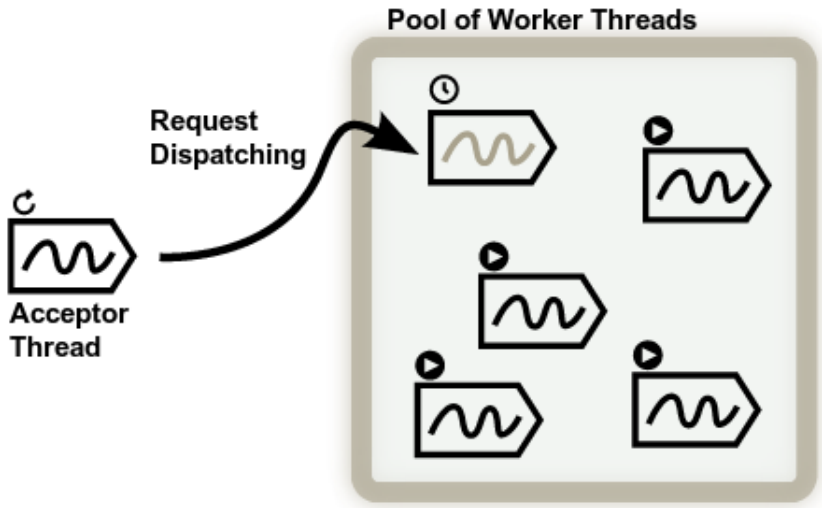
Trước hết ta tìm hiểu hoạt động của server. Server truyền thống hoạt động theo cơ chế `synchronize IO`. Khi client gửi đến server một yêu cầu, server sẽ dựa trên url yêu cầu để tìm tài nguyên tương ứng. Server sẽ đợi vào ra dữ liệu.Tùy vào tài nguyên đó, server có thể gọi một chương trình khác để tiền xử lý (ví dụ php) và trả lại kết quả cho client. Để đáp ứng được yêu cầu cho nhiều client, server sẽ sử dụng thread để quản lý mỗi yêu cầu của client. Để tăng cường hiệu quả sử dụng thread, các server thường sử dụng cơ chế spool thread. Mô hình này có thể mô hình bởi hình vẽ dưới đây.
Như có thể thấy trong hình 2.1, khi có một yêu cầu được chấp nhận. Bộ phân phối yêu cầu sẽ tìm thread rảnh rỗi và phân phối yêu cầu cho thread đó. Tuy nhiên, mô hình này vẫn chứa những hạn chế nhất định. Đó là:

- Hạn chế của việc sử dụng luồng: Việc sử dụng luồng có nhưng hạn chế nhất định. Trước hết, số lượng luồng tối đa có thể tạo ra bị giới hạn bởi khả năng của CPU. Tiếp đến là chi phí quản lý luồng cực kỳ tốn kém đặc biệt khi số lượng luồng lớn(khởi tạo luồng, lập lịch và chuyển đổi luồng, hủy bỏ luồng).
- Lãng phí CPU do thời gian vào ra IO. Khi một yêu cầu từ client gửi tới, server sẽ phải đọc dữ liệu được định danh trong url, và chuyển nó tới bộ tiền xử lý. Trong thời gian vào ra, thread đó hoàn toàn không có tính toán. Do vậy hiệu xuất sử dụng CPU rất thấp.
- Việc đồng bộ giữa các thread: Trong trường hợp muốn đồng bộ giữa các thread, việc này cực kỳ khó.

2.2 NodeJS, mô hình server mới

2.2.1 Mô hình NodeJS

Khác với cơ chế truyền thống, NodeJS hoạt động theo cơ chế `asynchronous IO`. Trong NodeJS, có duy nhất một tiến trình hoạt động. Khi yêu cầu gửi từ client đến, server sẽ gửi yêu cầu cho bộ phận vào ra về yêu cầu đó và tiếp tục lắng nghe các yêu cầu khác từ client. Khi đã kết thúc vào ra, bộ phận vào ra gửi tín hiệu ngắt tới server. Server sẽ nhận dữ liệu



Hình 2.1: Mô hình Server truyền thống

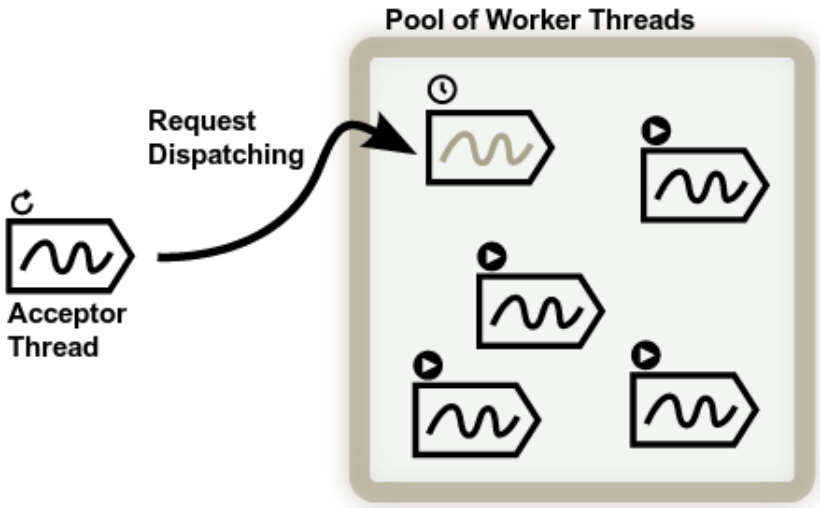
này và tiếp tục xử lý. Như vậy, hiệu xuất sử dụng CPU đã tăng nên. Hơn nữa, trong trường hợp này, do chỉ có một tiến trình duy nhất, ta không còn gặp phải vấn đề về thread và đồng bộ giữa các Thread như trong mô hình cũ gặp phải nữa. Mô hình hoạt động của NodeJS được theer hiện trong hình dưới đây: Theo cơ chế này, khi có ngắt xuất hiện, server sẽ tiếp tục thực hiện xử lý với dữ liệu được đọc. Vậy làm thế nào server có thể xác định được công việc cần phải làm? Giả sử đây là một công việc cần thực hiện của server:

```
var post = db.query('SELECT * FROM posts where id = 1');

// processing from this line onward cannot execute

// until the line above completes

doSomethingWithPost(post);

doSomethingElse();
```

Trong ví dụ này server phải thực hiện truy vấn cơ sở dữ liệu. Sau khi thực hiện song truy vấn, ta muốn server thực hiện hàm `doSomethingWithPost(post)` Trong mô hình `Asynchronous IO`, làm thế nào để khi truy vấn thực hiện xong, server biết và thực hiện hàm `ioSomethingWithPost(post)` NodeJS quản lý theo cơ chế `Event callback`. `Event callback` là khái niệm chỉ hàm sẽ được gọi khi có một sự kiện xảy ra. Code trên có thể được viết lại như sau:

```
callback = function(post) {
        doSomethingWithPost(post);
};

db.query('SELECT * FROM posts where id = 1', callback);
doSomethingElse();
```

Cách tiếp cận này khá đơn giản và hợp lý. Ta sẽ truyền hàm cần thực thi như một tham số của truy vấn. Khi quá trình IO kết thúc, callback sẽ được gọi. Vấn đề còn lại duy nhất là làm



Hình 2.2: Mô hình hoạt động của NodeJS

sao quản lý ngữ cảnh của hàm. Khi hàm trên được gọi, các biến môi trường đang ở trạng thái nào? Ryan Dahl , tác giả của NodeJS ban đầu viết sử dụng ngôn ngữ C, tuy nhiên do việc quản lý ngữ cảnh giữa các callback rất phức tạp, ông đã chuyển sang dùng ngôn ngữ Lua. Lua có cả các thư viện non IO blocking và I/O blocking có thể gây phức tạp cho các nhà lập trình vì vậ Lua không phải là một giải pháp hay. Sau đó, ông nghĩ tới javascript. Trong javascript, có một đặc trưng thú vị là closure function. Khi một hàm được gọi, trước hết bộ thông dịch sẽ tìm các biến cục bộ của hàm sẽ được tham chiếu. Nếu biến đó không tồn tại, bộ thông dịch tiếp tục tìm kiếm ra toàn cục. Ngữ cảnh của hàm được xác định khi hàm được khai báo. Để hiểu kỹ vấn đề này hơn, ta cùng xem ví dụ đơn giản sau:

```
var clickCount = 0;

document.getElementById('mybutton').onclick = function() {
        clickCount ++;
}
```

Trong ví dụ này, hàm callback ở đây thực hiện đếm số lần click vào một nút. Biến clickCount ở đây không được khai báo trong hàm. Do vậy nó chính là biến toàn cục. Do tại thời điểm khai báo, biến clickCount đã tồn tại, nên đây chính là biến được tham chiếu đến khi callback được gọi.

2.2.2 Một kết quả benchmark

Trong phần này ta cùng làm một ví dụ benchmark đơn giản để so sánh hiệu năng của 2 mô hình

2.2.3 Lập trình hướng sự kiện trên server

Với việc ra đời NodeJS, cũng dẫn đến một mô hình lập trình mới trên server. Đó là lập trình hướng sự kiện. Phía client, khái niệm này có thể không quá mới mẻ. Trong ngôn ngữ javascript, việc lập trình gắn liền với các sự kiện. Các lập trình viên sẽ tạo các sự kiện DOM

và bắt các sự kiện này và sử lý. Với NodeJS, cách lập trình cũng như vậy. Ta cũng bắt các sự kiện từ phía client hoặc các sự kiện từ hệ thống và phản hồi. So với cách lập trình truyền thống, đây là một cách lập trình khá thú vị, mới mẻ.Để hiểu rõ vấn đề này hơn ta sẽ tìm hiểu các module được trình bày trong chương 3 của bài báo

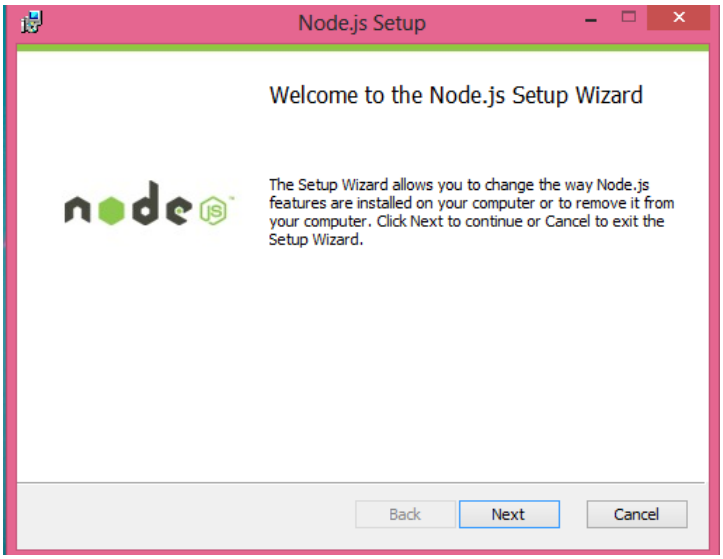
Chương 3

Cài đặt và quản lý module trong NodeJS

3.1 Cài đặt NodeJS

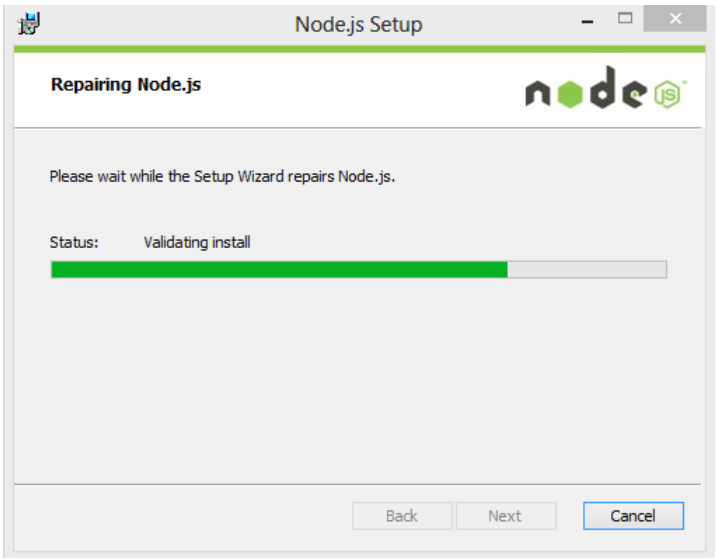
3.1.1 Cài đặt NodeJS

Để cài đặt NodeJS bạn có thể truy cập và trang http://nodejs.org/download/ để tải phiên bản phù hợp với hệ điều hành của bạn xuống. **Với Windows:**
Bạn chỉ cần chạy file *node-v0.8.18-x84.msi* hoặc *node-v0.8.18-x64.msi* tùy theo hệ điều hành đang sử dụng và tiến hành cài đặt bình thường.



Với Linux:
Sau khi tải về gói package của NodeJs ta tiến hành thực hiện các bước sau:

1. $ `xzf node-v0.8.18.tar.gz`
2. $ `cd node -v0.8.18`
3. $ `./configure`
4. $ `make`



5. `$ sudo make install`

Ngoài ra một số bản phân phối linux cho phép bản tải NodeJS thông qua package. Để thực hiện điều đó, bạn chỉ cần vào terminal và gõ lệnh:

```
$ apt-get install nodejs
```

Để kiểm tra máy đã cài nodeJS chưa ta thực hiện lệnh:

```
$ node -v
```

3.1.2 Sử dụng NodeJs

Sau khi tiến hành cài đặt xong NodeJs chúng ta bắt đầu sử dụng Node. Để khởi động giao diện dòng lệnh của Node ta sử dụng câu lệnh:

```
$ node
```

Tiến hành kiểm tra việc cài đặt và quan sát Node đang làm gì, ta sử dụng:

```
> console.log('Hello World!')
Hello World!
> undefined
```

Cũng có thể chạy một JavaScript từ một file. Nếu tạo chúng ta tạo một file `hello_world.js` với nội dung.

```
console.log('Hello World!')
```

Tiến hành file trên với câu lệnh:

```
$ node hello_world.js
Hello World
```

3.2 Quản lý module trong NodeJS

3.2.1 Quản lý module trong NodejS

Với NodeJs ta có thể mở rộng các chức năng hông qua các module. Để quản lý chúng, ta sử dụng một phần mềm của bên thứ 3 là *npm* (Node Package control management). Thông tin về *npm* có thể tìm thấy ở trang chủ : http://npmjs.org. Npm được cài sẵn khi cài Nodejs.

1. Để liệt kê các gói ta sử dụng lệnh
   `$ node ls [filter]`
   - Liệt kê tất cả các gói
     `$ node ls`
   - Liệt kê tất cả các gói được cài đặt
     `$ node ls installed`
   - Liệt kê tất cả các gói đang ở trạng thái ổn định(stable)
     `$ node ls stable`
   - Liệt kê tất cả các gói theo tên
     `$ node ls "tên hoặc pattern"`
2. Để cài đặt gói ta sử dụng lệnh:
   `$ npm install package[@filters]`
   - Cài gói express
     `$ npm install express`
   - Cài gói express phiên bản 2.0.0beta
     `$ npm install express@2.0.0beta`
   - Cài gói phiên bản lớn hơn 0.1.0
     `$ npm install express@">=0.1.0`
3. Để gỡ một gói , ta dùng lệnh
   `$ npm rm [-g] <package name>[@version]`
4. Để xem thông tin về module nào đó ta sử dụng lệnh.
   `$ npm view [@] [[.]...]`
5. Để update các module bằng việc sử dụng câu lệnh:
   `$ npm update [-g] <package name>`

3.3 Xây dựng và sử dụng các module

Trong phần này, ta sẽ tìm hiểu cách xây dựng và sử dụng một module cho nodeJS.

```
var module = require(path_to_module| module_name);
```

Tham số truyền vào là tên module nếu đó là một core module được quản lý qua npm. Trong trường hợp module đó là một module custom của người dùng, ta chỉ cần chi ra đường dẫn đến file đó.

Ngoài ra ta cũng có thể load các module trong một thư mục. Đầu tiên Nodejs sẽ kiểm tra xem đó có phải là một packge hay không thông qua kiểm tra một file package.json. Cấu trúc file json sẽ chứa thông tin về file chính của package. Ví dụ:

```
{
    "name": "myModule",
    "main": "./lib/myModule.js"
}
```

Khi đó, Node sẽ thử load file path_to_module/lib/myModule.js

Một điểm quan trọng cần lưu tâm là khi thực hiện thao tác này, ta đã cache module đó. Do vậy khi viết nhiều lần nạp module, thực chất chỉ có một hàm đã được thực hiện. Để truy cập vào các thành phần của module, module cần phải xuất thông tin đó ra thông qua lệnh module.exports.

Ví dụ:

```
1  //test.js
2  var level = 0;
3
4  function loadGame(){
5           //code in here
6  }
7
8  function playGame(){
9          //code in here
10 }
11
12 module.exports.loadGame = loadGame;
13 module.exports.playGame = payGame;
14 module.exports.level = level;
15
16 //app.js
17 var test = require('./test.js');
18 test.loadGame();
19 test.playGame();
20 console.log(test.level);
```

Trong ví dụ này, trong module test, ta đã xuất ra thông tin gồm 2 hàm và một biến. Chính vì vậy khi load module này vào file app.js, ta có thể sử dụng chúng.

Lập trình cơ bản với NodeJS

3.4 Module trong NodeJS

3.5 Một số module làm việc vào ra

3.6 Một số module về mạng

3.7 Một số module công cụ

Chương 4

Ví dụ minh họa với NodeJS