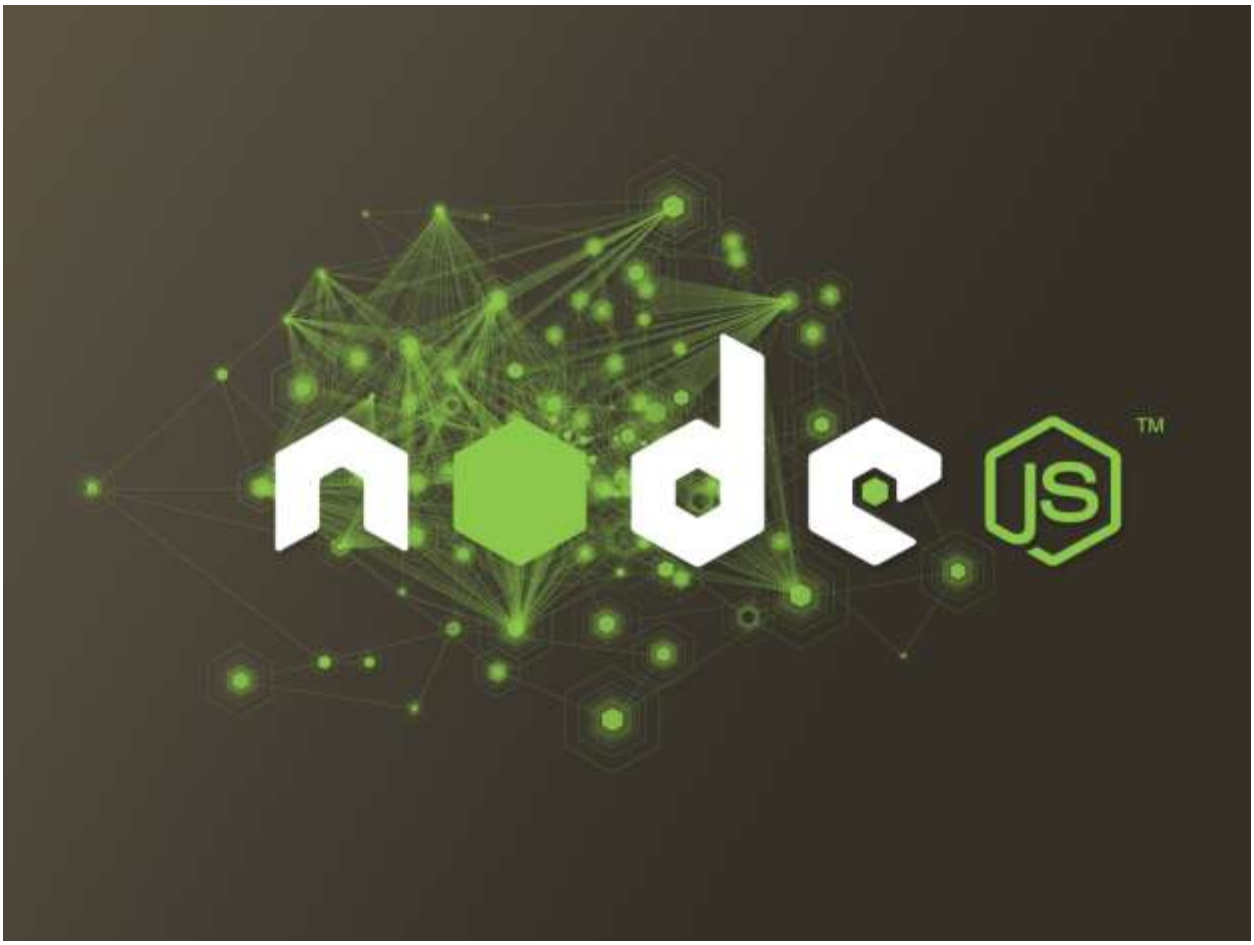

Node.js

برای تازه واردها

سعید علیدادی

آبان ۱۳۹۱

فهرست

مقدمه ٤

NODE.JS در مقابل PHP + APACHE ٤

نصب و راه اندازی ٥

ایست! ٥

سلام دنیا ٥

از مرورگر بخواهید ٥

دستور اول کجاست؟ ٦

راه اندازی یک سرور HTTP پایه ٦

ارسال توابع ٧

مدیریت رخداد با پاسخ برگشتی غیر همزمان ٨

سرور ما چگونه در خواست ها را مدیریت میکند ١٠

یافتن مکانی برای ماژول سرور ١٠

مسیر یابی درخواست ها ١٢

مسیریابی به مدیران درخواست حقیقی ١٥

مدیران درخواست را پاسخگو کنیم ١٨

چگونه این کار را نکنیم ١٩

بلاک یا بدون بلاک، مسئله این است ٢١

مدیران درخواست با عملیات بدون بلاک پاسخ میدهند ٢٤

یک خدمت رسانی مفید ٢٨

پازل ها را کامل میکنیم ۲۸

مدیریت آپلود فایل ۳٤

پایان ٤٤

منابع ٤٥

مقدمه

جاوا اسکریپت یک زبان برنامه نویسی به شیوه اسکریپت میباشد که در سال ۱۹۹۵ توسط شرکت نت اسکیپ برای طراحی صفحات داینامیک در سمت کاربر ساخته شد. جاوا اسکریپت یک زبان شی گرا و رخداد محور است همچنین این زبان خود با زبان C++ نوشته شده و به عنوان موتور جاوا اسکریپت در هسته مرورگر ها قرار گرفه است یکی از معروفترین موتورهای جاوا اسکریپت موتور V8 شرکت گوگل میباشد که در مرورگر کروم قرار گرفته و به عنوان سریعترین موتور جاوا اسکریپت معروف است.حال این زبان با همت شخصی به نام رایان دهل (Rayan dahl) در سال ۲۰۱۰ جای خود را با نام node.js در بین زبان های برنامه نویسی سمت سرور با همان موتور V8 پیدا کرد اما با سبکی جدید و قابلیت هایی بهتر. دو شرکت یاهو و جوی نت از اولین کاربران این زبان میباشند.رایان دهل هم اکنون در شرکت جوی نت مشغول میباشد. در سال ۲۰۱۰ شرکت یاهو یک کنفرانس را با سخنرانی رایان برای کارمندانش تدارک دید. میتوانید فیلم کنفرانس را از وبلاگ www.yui.zenfs.com با دو کیفیت دانلود کنید . در اینجا به یک آموزش مقدماتی و مقایسه اجمالی این زبان با دیگر رقبایش میپردازم. از آنجا که زمان زیادی از تولد Node.js نمیگذرد منابع زیادی برای یادگیری آن وجود ندارد. به ویژه در کشور ما که روند ترجمه تا انتشار پروسه ی طولانی مدتی میباشد. باید قبل از هر چیز این نکته را هم متذکر شوم، Node.js یک زبان برنامه نویسی کاملا مستقل نیست بلکه یک پیاده سازی از زبان جاوا اسکریپت است که رخدادهای سمت سرور را مدیریت میکند. امیدوارم این کتاب آغازی باشد برای آغازی دیگر و پایانی برای پایانی دیگر.

سعید علیدادی
۹۱ / ۷/ ۳۰

Node.js در مقابل PHP + Apache

آنهایی که با زبان پی اچ پی کار کرده اند به خوبی میدانند که برای هر درخواست HTTP از سمت کاربر یک بلاک جدیداز اسکریپت های این زبان اجرا میشود به معنی دیگر این زبان به صورت چند نخی به درخواست های کاربران پاسخ میدهد که مسلما نخ ها با مدیریت سرور وب از یکدیگر مجزا میباشند ولی نود روش خودش را دارد به این صورت که برای تمام در خواست ها یک بلاک ایجاد کرده ولی به شکل موازی به درخواست ها پاسخ میدهد. فرض کنید یک شرکت خدماتی به درخواست های مختلف مشتریان همزمان رسیدگی می کند حال برای اینکه به تمام در خواست ها به شکل موازی رسیدگی شود دو راه وجود دارد: یک تیم جدید استخدام کند یا در خواست را به مدیر تیم داده و بگوید متناسب با این درخواست یک عضو جدید به تیم اضافه کند.نود روش دوم را انتخاب کرده و آمارها در زمینه سرعت و دقت چشمگیر میباشند.این مهمترین تفاوت این دو زبان میباشد و در طول کتاب به بعضی از تفاوت های دیگر نود با دیگر زبان های سمت سرور با مثال اشاره خواهم کرد.برای دیدن نتایج حاصل از یک benchmark از این مقایسه میتوانید به آدرس زیر مراجعه کنید.

http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php

نصب و راه اندازی

نود را میتوانید متناسب با نوع سیستم عامل که دارید از سایت www. nodejs.orgدانلود، و بعد ان را نصب کنید یا بر روی گزینه نصب که در صفحه اول سایت قرار گرفته کلیک کنید تا بر روی سیستم شما نصب شود. همچنین نود یک زبان متن باز میباشد و میتوانید سورس آن را از سایت خودش دریافت کنید. بعد از نوشتن کد برنامه در یک ویرایشگر آن را با فرمت جاوا اسکریپت ذخیره کنید و سپس با برنامه ای که نصب کرده اید آن را اجرا کنید.خروجی برنامه را با توجه به کدی که نوشته اید به دو صورت میتوانید ببینید، در خود کنسول یا در صفحه مرورگتان.برای نود چندین محیط توسعه وجود دارد که webStorm ؛ محصول شرکت جتبینز و webMatrix از شرکت مایکروسافت از معروفترین ها میباشند.

ایست!

از آنجا که نود مفهوم پیشرفته ای از جاوا اسکریپت را ارائه میدهد آشنایی اولیه با جاوا اسکریپت و مفهوم شی گرایی و همچنین HTML و CSS جهت فهمیدن مطالب این کتاب ضروری میباشد.

سلام دنیا

نود با دو چیز همراه است، محیط اجرایی و کتابخانه،در این کتابخانه ها از فایلهایی DLL مانند کتابخانه های PHP خبری نیست همه چیز جاوا اسکریپت است.حالا کد زیر را در ویرایشگر مورد علاقه خود بنویسید و با پسوند js ذخیره کنید.در اینجا نود وارد عمل میشود، این فایل را با نود اجرا کنید.به دنیای نود خوش آمدید.

```
console.log("Hello World");
```

این دستور متن Hello World را در صفحه کنسول شما نمایش خواهد داد.اگر این دستور را مستقیما با نود اجرا کنید به سرعت بسته خواهد شد.این برنامه خیلی ساده بود حالا یک برنامه برای پاسخ گویی به درخواست های مرورگر مینویسیم.

از مرورگر بخواهید

در این قسمت قصد داریم یک برنامه برای آپلود کردن فایل های کاربر بنویسیم.سرویس دهی برنامه به شکل زیر میباشد.

- کاربر برای اجرای سایت ما آدرس http://domain را در مرورگر وارد میکند ودر خواستش را ارسال میکند.
- کاربر یک صفحه خوش آمد گویی با آدرس http://domain/start که یک فرم آپلود را نشان میدهد دریافت میکند.
- کاربر با انتخاب یک عکس و کلیک بر روی دکمه ارسال، فایل را در سایت آپلود میکند و یک صفحه با آدرس http://domain/upload دریافت مینماید.

دستور اول کجاست؟

حال بیایید مراحل اجرای بخش های مجزای برنامه را از شروع تا پایان مرور کنیم:

- **قصد داریم به عنوان یک سرویس دهنده وب عمل کنیم پس به یک سرور HTTP نیاز داریم.**
- **سرور ما باید به در خواست های متفاتی که در URL نهفته است پاسخ دهد پس بنابر این به نوعی مسیر یاب برای رساندن درخواست ها به مدیران درخواست نیاز داریم.**
- **برای رسیدگی به درخواست هایی که به سرور رسیده، و توسط مسیر یاب، مسیر یابی شده به مدیران درخواست نیاز داریم.**
- **مسیر یاب باید داده هایی که توسط POST از فرم گرفته شده اند را به مدیران در خواست برساند پس بنابر این به *مدیریت داده* های درخواست نیاز داریم.**
- **ما باید صفحه ای را که درخواست شده نمایش دهیم پس به نوعی *دستور نمایشی* نیاز داریم که مدیران درخواست از آنها برای نمایش محتوا در مرورگر کاربر استفاده کنند.**
- **کاربر باید بتواند عکس خود را آپلود کند پس به نوعی مدیر آپلود برای این کار نیاز داریم.**

به پیاده سازی این روند با PHP فکر کنیم.کار سختی نیست فقط کافیست حالت mod_php5 را در سرور آپاچی فعال کنید.پی اچ پی همه مسولیت ها را به عهده نمیگیرد.ولی با نود قضیه کمی متفاوت میباشدچرا که با نود فقط برنامه نیست که اجرا میشود بلکه وظیفه مدیریت سرور HTTP هم به عهده نود میباشد.در واقع برنامه ما و سرور وب هر دو یکی هستند. شاید در نگاه اول بار زیادی بر دوش نود به نظر برسد ولی در ادامه خواهیم دید که نود به خوبی این وظایف را انجام میدهد.

خوب حالا بیاید اولین بخش برنامه خود را با پیاده سازی سرور HTTP شروع کنیم. دستور اول اینجاست.

راه اندازی یک سرور HTTP پایه

شاید این نگرانی برایتان پیش آمده باشد که چگونه کدهای خود را مدیریت کنیم؟میتوانیم کدهای خود را در ماژولهای مجزا نوشته و مانند همیشه آن ها را در برنامه وارد کنیم.حالا در اینجا یک ماژول اصلی برای شروع برنامه مینویسیم و از ماژول HTTP که دستورات سرور در آن قرار دارد بهره میبریم.نام این فایل را server.js گذاشته و در ریشه اصلی پروژه خود ذخیره میکنیم. این ماژول به شکل زیر است.

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

حالا برنامه را بعد از ذخیره در ریشه اصلی پرژه با نود اجرا کنید و سپس در مرورگر خود این آدرس `http://localhost:88` را وارد کرده و ارسال کنید.باید یک صفحه با متن `Hello World` را ببینید.پس تا حالا یک وب سرور `HTTP` ساختیم.

در خط اول دستورات بالا با دستور `require` ماژول `http` را به برنامه اضافه نمودیم و آنرا در متغیری با همین نام قرا دادیم در خط بعد متد `createServer` متعلق به همین ماژول را با پارامتری که یک تابع بی نام میباشد صدا زدیم.این متد یک شی را برمیگرداند که دارای متدی با نام `listen` میباشد و پارامترش را شماره پورتی که سرور ما به آن گوش میدهد قرار دادیم.توجه کنید که هر عددی را نمیتوانستیم به جای پورت قرار دهید.میتوانیم کد بالا را به شکل زیر هم بنویسیم که یک سرور را برای ما فراهم میکند.

```
var http = require("http");

var server = http.createServer();
server.listen(88);
```

این دستورات همان کار را میکنند با این تفاوت که چیزی در مرورگر نمایش نمیدهد.

ارسال توابع

در جاوا اسکریپت میتوا ن تابعی را به عنوان پارامترتابع دیگر ارسال کرد.به کد زیر توجه کنید.

```
function say(word) {
  console.log(word);
}

function execute(someFunction, value) {
  someFunction(value);
}

execute(say, "Hello");
```

همچنین میتوان یک تابع را در پارامتر تابعی دیگر تعریف کرد و آن را فراخوانی نمود.به شکل زیر توجه کنید.

```
function execute(someFunction, value) {
  someFunction(value);
}

execute(function(word){ console.log(word) }, "Hello");
```

در این روش نیازی به انتخاب یک نام برای تابع داخلی نیست و به همین دلیل به آن تابع بینام میگوییم.با توجه به این دو روش میتوانیم دستوراتی را که برای راه اندازی سرور قبلا نوشتیم به شکل زیر پیاده سازی کنیم.

```
var http = require("http");

function onRequest(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);
```

مدیریت رخداد با پاسخ برگشتی غیر همزمان

برای فهمیدن اینکه چرا از این روش استفاده کردیم باید بدانیم نود چگونه برنامه ما را اجرا میکند.روش نود مختص به خودش نیست ولی با روش PHP، Python، Ruby یا جاوا فرق دارد. با مثال زیر بهتر متوجه میشویم.با این دستور یک کوئری را به پایگاه داده ارسال میکنیم و نتایج برگشتی را در یک متغیر ذخیره مینماییم.و سپس یک متن را در کنسول نشان میدهیم.

```
var result = database.query("SELECT * FROM hugetable");
console.log("Hello World");
```

در این قطعه کد مانند همیشه تا زمانی که خط اول اجرا نگردد برنامه به خط بعدی نمیرود.پس اگر اجرای کوئری روی پایگاه داده طول بکشد در واقع اجرای کل برنامه به تاخیر افتاده است و اگر این برنامه در اثر درخواست یک کاربر وب راه اندازی شده باشد، کاربر باید تا اتمام دستور اول منتظر متن `Hello World` بماند.در همه زبان ها این قضیه وجود دارد ولی از آنجا که در PHP برای هر درخواست HTTP یک نخ جدید باز میشود اگر برای یک درخواست اجرای این کوئری طول بکشد بر درخواست های دیگر تاثیری ندارد ولی در نود که فقط از یک چرخه پردازشی استفاده میکند این مشکل بزرگی محسوب میشود.نود روشی را به نام رخداد سازی غیر همزمان با پاسخ برگشتی با استفاده از حلقه رخداد پیاده سازی میکند.حالا قطعه کد قبل را به صورت زیر مینویسیم تا به این مشکل غلبه نماییم.

```
database.query("SELECT * FROM hugetable", function(rows) {
  var result = rows;
});

console.log("Hello World");
```

در اینجا به جای انتظار برای نتیجه حاصل از اجرای کوئری، یک تابع بی نام را به عنوان پارامتر دوم به `database.query()` ارسال کردیم.حالا نود جی اس میتواند درخواست پایگاه داده را به شکل غیر همزمان مدیریت کند.در واقع این یک متد غیر همزمان در کتابخانه غیر همزمان میباشد.کاری که نود انجام میدهد به این قرار است: کوئری را میگیرید و به دیتابیس ارسال میکند اما به جای انتظار برای پایان اجرای کوئری یک یادداشت در ذهن خود ثبت میکند که میگوید" هر زمان اجرای کوئری توسط دیتابیس کامل و ارسال شد باید تابع بی نامی را که به عنوان پارامتر دوم `database.qury()` فرستاده شده بود اجرا کنم".سپس به سرعت به دستور بعدی رفته و متن `Hello World` نمایش میدهد.و بعد از این وارد حلقه رخداد میشود.نود به طور پیوسته این چرخه را اجرا میکند و تا وقتی که کاری برای انجام دادن نباشد منتظر رخداد بعدی میماند رخدادی شبیه برگشت نتیجه حاصل از اجرای کوئری.

حالا میدانیم که چرا برای راه اندازی سرور HTTP از این روش استفاده کردیم.اگر نود سرور را بسازد وتارسیدن درخواست بعدی متوقف شود کارآمد نخواهد بود.اگر کاربر دومی درخواستش را در حالی که نود هنوز مشغول رسیدگی به درخواست اول است ارسال کند فقط زمانی که در خواست اول به پایان رسیده باشد جوابش را دریافت میکند.و از آنجا که این روند تکرار خواهد شد به هیچ وجه کارآمد نیست.

آیا برنامه ما بعد از پیاده سازی سرور در صورت عدم هیچ گونه درخواست HTTP یا پاسخ برگشتی از یک تابع هنوز در حال اجراست؟بیایید امتحان کنیم:

```
var http = require("http");

function onRequest(request, response) {
  console.log("Request received.");
  response.writeHead(200, {"Content-Type": "text/plain"});
```

```
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);

console.log("Server has started.");
```

در کد بالا از onsole.log() استفاده کردیم که بعد از راه اندازی سرور متن تایید را در کنسول نمایش میدهد.و یکی دیگر که بعد از شروع سرور HTTP تایید دریافت درخواست را نمایش میدهد.وقتی این کد را با نام server.js اجرا میکنیم server has started اولین خروجی خواهد بود.و هر وقت یک درخواست را از مرورگر خود با آدرس http://localhost:8888 ارسال کنیم request دومین خروجی خواهد بود.به این نکته توجه کنید که ممکن است request received دو بار در کنسول نشان داده شود چرا که اکثر مرورگرها درخواستی را جهت گرفتن آیکون اصلی سایت یا همان favicon به سرور ارسال میکنند.

سرور ما چگونه در خواست ها را مدیریت میکند

وقتی که تابع onRequest() توسط سرور راه می افتد دو پارامتر request و response به آن ارسال میشود.اینها اشیایی هستند که میتوانیم از متدهایشان برای مدیریت جزییات درخواست HTTP استفاده کنیم و نتیجه را به مرورگر کاربر ارسال نماییم.کد ما فقط این کار را میکند: هر زمان که یک درخواست از کاربر میرسد با افزودن ۲۰۰ به سرآیند HTTP و اعلام نوع محتوای ارسالی به این سرآیند آن را با محتوای Hello World پاسخ میدهد و با متد end() پاسخ قطع میشود. در اینجا نوع درخواست برای ما اهمیت ندارد و به این دلیل از شی request استفاده نکردیم.

یافتن مکانی برای ماژول سرور

حالا مایک ماژول پایه برای سرور ساخته ایم و مانند همیشه قصد اجرای این ماژول از ماژول اصلی برنامه سایت یعنی index.js که راه انداز سایت است را داریم.ببینیم چگونه ماژول server.js را به یک ماژول واقعی نود برای استفاده در آینده تبدیل کنیم.

همانطور که تا حالا متوجه شدید ما به صورت زیر از یک ماژول در کد های خود استفاده کردیم:

```
var http = require("http");

...

http.createServer(...);
```

ماژول `http` **در جایی از نود قرار گرفته و ما آنرا در اول برنامه با دستور** `require` **در متغیر محلی ذخیره میکنیم.حالا میتوانیم از متد های عمومی این ماژول مانند** `createServer()` **در برنامه خود استفاده کنیم.بهتر است از نام ماژول برای متغیرمحلی برنامه خود استفاده کنیم ولی میتوان هر نامی را برگزید:**

```
var foo = require("http");

...

foo.createServer(...);
```

برای تبدیل server.js **به یک ماژول جهت وارد کردن آن به شکل بالا نیاز به زحمت زیادی نیست. فقط باید کدی به آن اضافه نماییم که نشان دهد یک ماژول برای استفاده در جاهای دیگر است.کدهای اضافه شده به شکل زیر است.**

```
var http = require("http");

function start() {

 //our functions that will be used by external codes come here
}

exports.start = start;
```

حالا تابع `onRequest()` **خودمان را به همراه راه انداز آن جای توضیحات کد بالا قرار میدهیم:**

```
var http = require("http");


function start() {
  function onRequest(request, response) {
    console.log("Request received.");
```

```
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

حالا ماژول server.js ما آماده است.آنرا در پوشه پروژه خود با همین نام ذخیره میکنیم.

حال زمان استفاده از این ماژول در راه انداز یا bootstrap برنامه است. یک فایل با نام index.js بسازید و کد زیر را دران قرار دهید:

```
var server = require("./server");

server.start();
```

بله همین بود.حالا میتوانید با خیال راحت فایل `index.js` را اجرا کنید.در اینجا آدرس محل قرارگیری ماژول سرور را به تابع `require()` ارسال کردیم و از متد `start()` آن برای راه اندازی سرور استفاده کردیم.

ارسال درخواست های HTTP به قسمت های مختلفی از کدهای برنامه "routing" نامیده میشود.درادامه یک ماژول به نام router.js میسازیم.

مسیر یابی درخواست ها

در این قسمت باید داده های GET و POST موجود در آدرس URL را گرفته وبه مسیر یاب بدهیم.مسیر یاب بر اساس این مقدارها تصمیم میگیرد که کدام قسمت از کد برنامه باید اجرا شود.پس به URL ارسال شده میرویم و مقدارهای GET وPOST را از آن استخراج میکنیم.میتوان این قسمت را متعلق به سرور یا روتر قرار داد.ولی در اینجا اجازه دهید قسمتی از سرور باشد.

تمام چیزهایی که نیاز داریم در شی request قرار دارد که به عنوان پارامتر اول به `()` onRequest ارسال میشود. اما برای تفسراین اطلاعات به دو ماژول نود به نامهای url و querystring نیاز داریم.

ماژول url دارای متد هایی است به ما کمک میکند قسمت های مختلف یک آدرس URL را بیرون بکشیم.و querystring برای تجزیه کوئری های موجود در آدرس کاربرد دارد.

```
           url.parse(string).query
                                       |
           url.parse(string).pathname  |
                       |               |
                       |               |
                     ------ -------------------
http://localhost:8888/start?foo=bar&hello=world
                                  ---       -----
                                   |          |
                                   |          |
               querystring(string)["foo"]     |
                                              |
                          querystring(string)["hello"]
```

همچنین میتوانیم از qureystring برای تجزیه کردن POST هم استفاده کنیم که در ادامه خواهیم دید.

حالا به تابع برگشتی (تابع برگشتی با تابع بازگشتی تفاوت دارد تابع برگشتی callback نامیده میشود و تابعی است که به عنوان پارامتر یک تابع دیگر ارسال میشود) onRequest() کدی را که برای فهمیدن مسیر موجود در URL از سمت مرورگر درخواست شده است وارد میکنیم:

```
var http = require("http");
var url = require("url");

function start() {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }

  http.createServer(onRequest).listen(8888);
```

```
  console.log("Server has started.");
}

exports.start = start;
```

حالا برنامه ما میتواند درخواست ها را بر اساس مسیر موجود در URL فرستاده شده تشخیص دهد.این قابلیت به ما کمک میکند نقشه در خواست ها را با استفاده از روتری که بعدا خواهیم ساخت برای مدیران درخواست رسم نماییم.در مفهوم برنامه به این معنی است که امکان بررسی و مدیریت آدرسهای /start و /upload برای کدهای مختلف برنامه ما وجود دارد.

حالا زمان آن رسیده تا روتر یا مسیر یاب برنامه را بنویسیم. یک فایل با نام router.js در پوشه اصلی پروژه بسازید و کد زیر را در آن قرار دهید:

```
function route(pathname) {
  console.log("About to route a request for " + pathname);
}

exports.route = route;
```

البته این کد در حال حاظر کار زیادی انجام نمیدهد.بیایید ببینیم چگونه میتوان این روتر را به سرور متصل کرد قبل از آن که به منطقش چیزی را اضاف کنیم.

سرور HTTP ما نیاز دارد در مورد روتر بداند و از آن استفاده کند.ما میتوانستیم این نیازمندی را به سختی به سرور متصل کنیم ولی از آنجا که این راه سخت را از تجربه خودمان با زبان های دیگر یاد گرفته ایم قصد برقراری پیوند ضعیف سرور و روتر با تزریق این نیازمندی داریم.

بیایید اول تابع start() سرور را برای اینکه به ما اجازه دهد از پارامترهای ارسالی برای تابع route() استفاده کنیم گسترش دهیم.

```
var http = require("http");
var url = require("url");

function start(route) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");
```

```
    route(pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

حالا کدهای فایل index.js **را به شکل زیر تغییر دهید، با این کار تابع** `route()` **ماژول روتر را در ماژول سرور تزریق میکنیم:**

```
var server = require("./server");
var router = require("./router");

server.start(router.route);
```

برنامه را اجرا کنید(index.js **) و آدرس** localhost:8888/saeid **را وارد کنید اگر درست پیش رفته باشید متن زیر در کنسول نمایان میشود:**

```
Request for /saeid received.
About to route a request for /saeid
```

مسیریابی به مدیران درخواست حقیقی

حالا سرور HTTP **و مسیریاب درخواست به خوبی با یکدیگر ارتباط دارند ولی این کافی نیست.مسیر یابی به این معنیست که میخواهیم درخواست های** URL **های مختلف را به شکل های متفاوت مدیریت کنیم. شاید بخواهیم در خواستی که از** `/start` **می آید در یک تابع دیگری متفاوت از درخواستی که از** `/upload` **می آید مدیریت شود.**

مسیریابی در روتر به پایان میرسد ولی جایی نیست که به شکل واقعی به درخواستی رسیدگی شود چرا که وقتی برنامه پیچیده تر شد فهمیدن آن سخت میشود.

حالا بیایید مدیران درخواست را پیاده سازی کنیم، جایی که در خواست ها به آن مسیر یابی میشوند.یک فایل با نام requestHandlers.js ایجاد و کدهای زیر را در آن وارد کنید:

```
function start() {
  console.log("Request handler 'start' was called.");
}

function upload() {
  console.log("Request handler 'upload' was called.");
}

exports.start = start;
exports.upload = upload;
```

این ماژول به ما در اتصال مدیران در خواست به روتر کمک میکند.در این موقع باید یک تصمیم اتخاذ نماییم: آیا با کدنویسی سخت از requestHandlers در روتر بهره میگیریم، یا دوست داریم کمی بیشتر وابستگی را به برنامه تزریق کنیم؟هرچند تزریق وابستگی مانند تمام الگوهای دیگر فقط نباید به دلیل کاربردش استفاده شود، در این مورد به ما احساس یک پیوند ضعیف بین روتر و مدیران درخواستش را میدهد و بنابراین به روتر قابلیت استفاده دوباره میدهد.این به این معنیست که ما نیاز داریم مدیران درخواست را از سرور به روتر ارسال کنیم. اما این احساس خطای بیشتری به ما میدهد، که چرا باید کل مسیر را طی کنیم و همه چیز را از قسمت شروع برنامه(index.js) به سرور ارسال نماییم و آن را در روتر به جای دیگری.

حالا دو گیرنده یا مدیر داریم، اما در یک برنامه واقعی این تعداد افزایش میابد. مسلما نمیخواهیم هر زمان که یک مدیر درخواست برای یک URl جدید اضافه میشود با کار اضافی نقشه درخواست ها به مدیران را بکشیم. و داشته باشیم اگر request=x آنگاه مدیر y را صدا بزن.تعداد مختلفی از آیتم ها، هر کدام با یک رشته(URL درخواست شده) مشخص میشوند؟به نظر عالی میرسد.مثل یک آرایه انجمنی. کاملا مناسب کار ما.

ما تصمیم گرفتیم لیستی از requestHandlers را به عنوان یک شی ارسال نماییم و جهت رسیدن به یک پیوند ضعیف این شی را به route() تزریق کنیم.

بیایید با قرار دادن کد زیر در فایل اصلی برنامه یعنی index.js شروع کنیم:

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");
```

```
var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;

server.start(router.route, handle);
```

همنطور که میبینید کشیدن نقشه از درخواست مدیر درخواست با قرار دادن / و نام درخواست بعد از آن بسیار ساده است.و همچنین میتوان به یک مدیر بیش از چند درخواست را نسبت دهیم مانند "/" و "/start" که به `requestHandlers.start` متصل شده اند.بعد از تعریف شی `handle`، آن را به عنوان پارامتر دیگری به سرور ارسال میکنیم پس باید کد `server.js` را به صورت زیر اصلاح نماییم:

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(handle, pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

ما فقط پارامتر handle را به start() افزودیم و شی handle را به عنوان پارامتر اول تابع برگشتی route() ارسال کردیم.

حالا بیایید تابع route() را مطابقا در router.js به شکل زیر تغییر دهیم:

```
function route(handle, pathname) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname]();
  } else {
    console.log("No request handler found for " + pathname);
  }
}

exports.route = route;
```

کاری که در اینجا انجام دادیم این بود که بررسی کردیم اگر برای مسیر داده شده مدیر درخواستی وجود داشت تابع متناظر را صدا میزنیم در غیر این صورت یک پیام بر این مبنا که مدیری برای مسیر داده شده وجود ندارد در کنسول نمایش میدهیم.از آنجا که میتوانیم به مدیر درخواست ازدرون شی دسترسی داشته باشیم درست مانند دسترسی به اعضای یک آرایه انجمنی، پس به صورت handle[pathename]() عمل میکنیم.

حالا ما روتر، سرور و مدیران رخداد را متصل به هم در اختیار داریم.اگر در مرورگر آدرس http://localhost:8888/start را وارد کرده و ارسال نماییم متن زیر در کنسول نمایان میشود و این یعنی همه چیز تا حالا به خوبی انجام گرفته است.

```
Server has started.
Request for /start received.
About to route a request for /start
Request handler 'start' was called.
```

توجه کنید که در این جا از آوردن خروجی مربوط به favicon صرف نظر نموده ایم.

مدیران درخواست را پاسخگو کنیم

عالی بود.حالا اگر مدیران رخداد بتوانند چیزی را به مرورگر ارسال کنند بهتر میشود، نه؟

متن `Hello World` که تا حالا در مرورگر مشاهده میکردید از طرف تابع `onRequest()` در ماژول سرور بود."مدیریت درخواست" یعنی "پاسخگویی به درخواست" در پایان، پس باید مدیران رخداد را برای صحبت کردن با مرورگر آماده کنیم.درست مانند کاری که تابع `onRequest()` انجام میدهد.

چگونه این کار را نکنیم

راهی را که ممکن است – مانند آنچه برنامه نویسان PHP و Ruby در پیش میگیرند— به طور صریح دنبال کنیم بسیار فریبنده میباشد: درست مانند یک جادو، در نگاه اول خوب به نظر میرسد ولی در حالی که انتظارش را نداریم همه چیز را درهم میریزد.

منظور از راه صریح این است: مدیران رخداد را طوری بازنویسی کنیم که با استفاده از return() محتوایی که قرار است به کاربر نشان دهند را به onRequest() برگردانند و از آنجا به مرورگر ارسال کنیم.

بیایید این کار راانجام دهیم و ببینیم چرا ایده خوبی نیست.

با requestHandlers شروع میکنیم و آن را به شکل زیر تغییر میدهیم تا پاسخ ها را به مرورگر ارسال کند:

```
function start() {
  console.log("Request handler 'start' was called.");
  return "Hello Start";
}

function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello Upload";
}

exports.start = start;
exports.upload = upload;
```

خوب، شبیه کد بالا روتر هم چیزی را که مدیر درخواست به آن ارسال کرده به سرور برمیگرداند.پس فایل router.js را به صورت زیر تغییر میدهیم:

```
function route(handle, pathname) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
```

```
    return handle[pathname]();
  } else {
    console.log("No request handler found for " + pathname);
    return "404 Not found";
  }
}

exports.route = route;
```

و در آخر باید سرور را طوری پیاده سازی کنیم که به مرورگر پاسخگو باشد، با محتوایی که مدیران رخداد از طریق روتر ارسال مینمایند.فایل server.js را به صورت زیر در آورید:

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    response.writeHead(200, {"Content-Type": "text/plain"});
    var content = route(handle, pathname)
    response.write(content);
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

اگر برنامه را اجرا کنیم، با یک جادو روبرو میشویم: درخواست http://localhost:8888/start در مرورگر " Hello Start " را نشان میدهد، درخواست http://localhost:8888/upload ، " Hello Upload " و درخواست http://localhost:8888/foo متن "404 Not Found".

خوب، مشکل چه بود؟جواب کوتاه: ما با مشکل مواجه شدیم چرا که مدیران درخواست ها میخواهند در آینده از عمل های بدون بلوک استفاده کنند.با هم جواب اصلی را میخوانیم.

بلاک یا بدون بلاک، مسئله این است

همانطور که گفته شد،مشکل وقتی ایجاد میشود که از عملیات بدون بلاک در مدیران درخواست استفاده کنیم. اما اجازه دهید اول به بحث در باره عملیات بلوکی بپردازیم و بعد عملیات بدون بلوک.

به جای توضیح مفهوم "بلاک" و "بدون بلاک" بیایید با یک مثال ببینیم اگر یک عمل بلاکی را به مدیران درخواست اضافه کنیم چه مشکلی پیش خواهد آمد.

برای انجام این کار مدیر درخواست sart() به شکلی تغییر میدهیم که قبل از برگرداندن "Hello Start" به مدت ۱۰ ثانیه منتظر بماند. به این دلیل که چیزی شبیه تابع sleep() در جاوااسکریپت وجود ندارد از یک حقه جالب استفاده میکنیم.لطفا requestHandler.js را به شکل زیر تغییر دهید:

```
function start() {
  console.log("Request handler 'start' was called.");

  function sleep(milliSeconds) {
    var startTime = new Date().getTime();
    while (new Date().getTime() < startTime + milliSeconds);
  }

  sleep(10000);
  return "Hello Start";
}

function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello Upload";
}
```

```
exports.start = start;
exports.upload = upload;
```

چه اتفاقی می افتد: وقتی تابع `start()` فراخوانی میشود، Node.js بعد از ۱۰ ثانیه رشته `Hello World` را برمیگرداند. و وقتی `upload()` خوانده میشود مثل قبل نتیجه را سریعا بر میگرداند.حالا ببینیم این تغییر چه نتیجه ای خواهد داشت.

مثل همیشه باید سرور را دوباره راه اندازی کنیم.این بار از شما میخواهم برای دیدن نتیجه از این روش استفاده کنید: اول از هر چیز دو تب یا دو پنجره در مرورگر خود باز کنید. در تب یا پنجره اول مرورگر آدرس http://localhost:8888/start را وارد نموده ولی آن را به سرور ارسال نکنید. در تب یا پنجره دوم آدرس http://localhost:8888/upload را وارد کنید و مثل قبل آن را ارسال نکنید.حالا در تب یا پنجره اول کلید enter یا دکمه ارسال مرورگر را زده و سریعا به تب یا پنجره دوم رفته و همین کار را تکرار کنید.چیزی که میبینید برایتان عجیب است☺ اگر برنامه را درست پیاده سازی کرده باشید، همانطور که انتظارداشتیم دریافت نتیجه `/start`، ۱۰ ثانیه طول کشدولی نتیجه `/upload` هم بعد از ۱۰ ثانیه به مرورگر رسید، در صورتی که تابع `sleep()` درمدیر درخواست متناظرش تعریف نشده بود. چرا این اتافاق افتاد؟

برای اینکه تابع `start()` دارای یک عملیات بلوکی است. ما در مورد مدل اجرایی `Node.js` صحبت کردیم.اجرای عملیات پر هزینه قابل قبول است ولی باید مراقب باشیم که روند پردازشی نود را با آنها یکی نکنیم یا به عبارت دیگر بلوک نکنیم.به جای آن هر موقع که اجرای عملیات سنگین یا پر هزینه ضروری بود آنها را در پس زمینه قرار دهیم و رخدادهایشان را با حلقه رخداد نود مدیریت کنیم.

حالا بیایید یک بار دیگر برای تجربه مشکل اول کد برنامه را به صورت زیر تغییر دهیم.قصد داریم دوباره از مدیر درخواست `start()` استفاده کنیم.آن را به شکل زیر تغییر دهید: (فایل requestHandler.js)

```
var exec = require("child_process").exec;

function start() {
  console.log("Request handler 'start' was called.");
  var content = "empty";

  exec("ls -lah", function (error, stdout, stderr) {
    content = stdout;
  });

  return content;
}
```

```
function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello Upload";
}

exports.start = start;
exports.upload = upload;
```

در خط اول یکی دیگر از ماژولهای Node.js به نام `child_process` را وارد برنامه کردیم. متد `exec()` متعلق به این ماژول است که یک بدون بلاک میباشد وبرای اجرای عملیات بدون بلاک برایمان مفید خواهد بود.کاری که این متد انجام میدهد این است که یک دستور شل از درون Node.js اجرا میکند در این مثال با استفاده از آن لیست تمام فایلهای درون دایرکتوری جاری (`"ls -lah"`) را میگیریم. و به ما کمک میکند این لیست را برای در خواست مرورگر با آدرس http://localhost:8888/start ارسال نماییم.

به این صورت که یک متغیر جدید به نام content با مقدار اولیه `"empty"` تعیین میکنیم، اجرای "ls –lah"، ریختن نتیجه در متغیر و برگرداندن آن.

مثل همیشه برنامه را اجرا میکنیم و به آدرس http://localhost:8888/start میرویم.

یک صفحه با محتوای "empty" برای ما نمایش داده میشود.چه اشتباهی در اینجا رخ داده است؟

همانطور که احتمالا تا حالا حدس زده اید متد `exec()` در یک مد بدون بلاک اجرا شده است. این خیلی خوب میباشد، چرا که با این روش میتوانیم عملیات پر هزینه مربوط به شل را (مانند کپی فایل های بزرگ و هر چیزی شبیه به آن) بدون ترس از توقف برنامه به طور کامل، درست مانند کاری که تابع `sleep()` انجام میداد، اجرا کنیم.

ولی با برنامه بدون بلاک و زیبای خود، زمانی که مرورگر نتیجه مورد انتظار را نشان ندهد، خیلی خوشحال نخواهیم شد.درست است؟

خوب، پس بیایید آن را درست کنیم، و وقتی در حال انجام این کار هستیم سعی کنیم به عدم کار آمد بودن معماری آن پی ببریم.

مشکل `exec()` است، جهت اجرای بدون بلاک از یک تابع برگشتی استفاده میکند.

در مثال ما، یک تابع بی نام وجود دارد که به عنوان پارامتر دوم به متد `exec()` ارسال میشود:

```
function (error, stdout, stderr) {
  content = stdout;
}
```

ریشه مشکل ما در این کد نهفته است: کد ما به شکل همزمان اجرا میشود، به این معنی که بعد از صدا زدن تابع `exec()` فورا Node.js اقدام به برگرداندن مقدار`return` میکند. در این موقع `content` هنوز خالی است، به خاطر این حقیقت که آن تابع برگشتی ارسال شده به تابع `exec()` هنوز صدا نشده است، چرا که `exec()` غیر همزمان عمل میکند.

حالا "`ls -lah`" یک عمل کم هزینه و سریع است.(حتی اگر ملیونها فایل در دایرکتوری باشد). به همین دلیل تابع برگشتی نسبتا سریع صدا میشود در حالی که هنوز به شکل غیر همزمان اتفاق می افتد.

فکر کردن به عملیات پر هزینه تر موضوع را روشنتر میکند: "`find/`" در کامپیوتر من تقریبا یک دقیقه طول میکشد، ولی اگر در مدیر درخواست آن را با "`ls -lah`" عوض کنم، هنوز وقتی یک درخواست با آدرس http://localhost:8888/start ارسال میکنم سریعا یک پاسخ HTTP دریافت مینمایم.واضح است که `exec()` هنوز در پس زمینه کاری انجام میدهد، در حالی که Node.js به اجرای برنامه ادامه میدهد، و ممکن است فرض را بر این بگذاریم که تابع برگشتی ارسال شده به `exec()` فقط موقعی که اجرای دستور "`find/`" به اتمام برسد صدا میشود.

اما چطور میتوانیم به هدف خود برسیم، همان نمایش لیست دایرکتوری ها در مرورگر کاربر؟

خوب، حالا که فهمیدیم چگونه این کار نکنیم. بیایید در مورد راهی برای رساندن پاسخ در خواست ها از مدیر درخواست به مرورگر به شکل صحیح صحبت کنیم.

مدیران درخواست با عملیات بدن بلاک پاسخ میدهند

توجه داشته باشید که راه هایی زیادی برای این کار وجود دارد ولی در اینجا به یک راه حل ممکن که معمولا توسط Node.js زیاد استفاده میشود میپردازیم. یعنی ارسال توابع به اطراف.

تا حالا برنامه ما قادر است محتوا (که مدیران درخواست قصد دارند به کاربر نشان دهند) را از مدیران درخواست گرفته و با گذراندن آن از لایه های مختلف برنامه (requestHandlers->router->server) به سرور HTTP برساند.

روش ما برای رسیدن به هدف چنین است: به جای بردن محتوا به سرور، سرور را به محتوا میبریم. به بیان دقیق تر، شی `response` (از تابع برگشتی سرور یعنی `onRequest()`) را از روتر به مدیران درخواست تزریق میکنیم. با این کار، مدیران خود میتوانند با توابع این شی به در خواست ها پاسخ دهند.

توضیحات کافی بود. حالا دستورالعمل تغییر برنامه را قدم به قدم مرور میکنیم. بیایید با `server.js` شروع کنیم:

```
var http = require("http");
var url = require("url");
```

```
function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(handle, pathname, response);
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

به جای اینکه انتظار داشته باشیم تابع `route()` مقداری برگرداند، شی `response` را به عنوان پارامتر سوم آن اضافه میکنیم. بنابراین تمام متدهای `response` را از درون مدیر `onRequest()` حذف میکنیم. چرا که قصد داریم تابع `route()` مسئولیت های آنها را بپذیرد.

در مرحله بعد به سراغ router.js میرویم:

```
function route(handle, pathname, response) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}

exports.route = route;
```

همان طور که گفتیم: ارسال شی به عنوان پارامتر سوم. و ارسال آن به مدیر درخواست متناظر.

اگر هیچ مدیر درخواستی پیدا نشود یک پیام در محتوا با متن "404 Not Found" و یک خطای ۴۰۴ در هدر صفحه به مرورگر ترسال میکنیم.

در آخر فایل requestHandler.js را به شکل زیز تغییر میدهیم:

```
var exec = require("child_process").exec;

function start(response) {
  console.log("Request handler 'start' was called.");

  exec("ls -lah", function (error, stdout, stderr) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write(stdout);
    response.end();
  });
}

function upload(response) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Upload");
  response.end();
}

exports.start = start;
exports.upload = upload;
```

توابع مدیر باید شی response را به عنوان پارامتر خود بگیرند و از آن در جهت پاسخ مستقیم به در خواست استفاده کنند.مدیر start از درون تابع برگشتی و بی نام متعلق به exec() به درخواست پاسخ میدهد و مدیر upload هنوز متن "Hello World" را ولی این بار با استفاده از شی response در جواب درخواست ارسال میکند.

حالا اگر برنامه را اجرا کنیم (index.js) باید همان طور که انتظار داشتیم عمل کند.

اگر دوست دارید ثابت کنید که عمل پرهزینه در پشت /start پاسخ گویی سریع از /upload را بلاک نمیکند، کد خود را در فایل requestHandlers به صورت زیر تغیر دهید:

```
var exec = require("child_process").exec;

function start(response) {
  console.log("Request handler 'start' was called.");

  exec("find /",
    { timeout: 10000, maxBuffer: 20000*1024 },
    function (error, stdout, stderr) {
      response.writeHead(200, {"Content-Type": "text/plain"});
      response.write(stdout);
      response.end();
    });
}

function upload(response) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Upload");
  response.end();
}

exports.start = start;
exports.upload = upload;
```

حالا برنامه را اجرا کنید و دوباره دو درخواست http://localhost:8888/start و http://localhost:8888/upload را به روشی که قبلا گفتیم ارسال کنید. میبینید که ناخیر در دریافت نتیجه /start بر زمان دریافت /upload اثری ندارد. هر چند که درخواست /start هنوز در حال رسیدگی میباشد. برای اطلا عات بیشتر در مورد متد exec() میتوانید به مستندات نود در http://nodejs.org/api/child_process.html#child_process_child_process_exec_command_options_callback مرا جعه کنید. فقط به این نکته توجه کنیم که اگر مقدار خروجی یا خطا از maxBuffer بیشتر شود و یا زمان اجرای تابع برگشتی از exec() timeoute تجاوز کند نابود میشود.

یک خدمت رسانی مفید

همه کارهایی که تا حالا انجام دادیم جالب و مفید بودند.اما هیچ محتوا یا کار مفیدی را برای کاربر در مقام یک سایت مفید و قابل ستایش ارائه ندادیم.

سرور، روتر و مدیران درخواست ما آماده اند. پس میتوانیم محتوایی را به سایت خود اضافه کنیم تا کاربران بتوانند با آن ارتباط برقرار کرده، یک فایل را انتخاب نمایند، در سایت ما آپلود کنند و آن را مشاهده کنند. برای سادگی فرض میکنیم کاربران فقط قصد آپلود عکس و دیدن آن را از درون سایت دارند.

قدم به قدم پیش میرویم: اول بررسی میکنیم که چگونه درخواست هایی که در POST میرسند را مدیریت کنیم(اما نه فایل آپلود شده را). و درقدم بعد از یک ماژول خارجی `Node.js` برای مدیریت آپلود فایل استفاده میکنیم. به دو دلیل این روند را انتخاب کردیم.

اولا مدیریت در خواست های اصلی در POST با نود جی اس نسبتا راحت است ولی هنوز درس های خوبی به ما می آموزد که استفاده از آن بی ضرر نیست.
دوما، آپلود فایل (چندین درخواست POST) در نود کار راحتی نیست و از این نظر در این کتاب چند صفحه ای نمیگنجد. اما استفاده از یک ماژول خارجی برای ما که تازه با نود آشنا شده ایم درس خوبی است و کمک میکند به خواندن این کتاب ادامه دهیم.

پازل ها را کامل میکنیم

بیایید یک کار همیشگی را ساده کنیم: ما برای کاربر یک فیلد متنی را نمایش خواهیم داد که توسط کاربر پر شده و در یک درخواست POST به سرور ارسال میشود. به محض دریافت و مدیریت این درخواست، محتوای فیلد را نمایش میدهیم.

کدهای HTML مربوط به فیلد متنی قرار است در درخواست اول به مرورگر ارسال شود. پس آن را در در مدیر درخواست /start قرار میدهیم. فایل requestHandlers.js را به صورت زیر تغییر دهید:

```
function start(response) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" content="text/html; '+
    'charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<textarea name="text" rows="20" cols="60"></textarea>'+
    '<input type="submit" value="Submit text" />'+
```

```
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}

function upload(response) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Upload");
  response.end();
}

exports.start = start;
exports.upload = upload;
```

حالا اگر درخواست http//localhost:8888/start را در مرورگر خود ارسال کنید باید یک محل برای ورود متن و یک دکمه ارسال را دریافت نمایید در خیر این صورت احتمالا برنامه را دوباره اجرا نکرده اید.

شاید بپرسید چرا منطق نمایشی برنامه را به این شکل در مدیر درخواست نشان داده ام. حق با شماست ولی برای جلوگیری از طولانی شدن کتاب این کار را کردم. از طرفی روش MVC چیز مفیدی از نود را در اینجا به ما آموزش نمیدهد.

بیایید از بقیه صفحه برای رسیدگی به یک مسئله جالب دیگر استفاده کنیم، و آن مدیریت در خواست POST که قرار است در صورت در خواست `/upload` توسط کاربر ارسال گردد، میباشد.

ما که حالا در نود با تجربه ایم به زودی از فهمیدن این حقیقت که مدیریت داده های ارسالیPOST در مد بدون بلاک و با همکاری برگشتی های غیر همزمان انجام میگیرد متعجب خواهیم شد.

از این جهت که درخواست های POST از نظر اندازه پتانسیل بالایی دارند و هیچ چیز کاربران را از وارد کردن متن های با اندازه چند مگابایت باز نمیدارد، مدیریت حجم کلی داده ها در یک ارسال منجر به یک عمل بلاکی میشود.

برای تبدیل کل پردازش به شکل بدون بلاک، Node.js داده های POST را در قطعه های کوچکی به کد برنامه ما میدهد.برگشتی هایی که بعد از رخدادهای اولیه صدا میشوند. این رخدادها `data` (یک قطعه از داده POST میرسد) و `end` (همه قطعه ها رسیدند) هستند.

ما باید به Node.js بگوییم وقتی این رخدادها واقع میشوند کدام تابع به عقب برگردد. این با افزودن `listener` به شی `request` که به محض رسیدن یک درخواست `HTTP` به سمت تابع برگشتی `onRequest()` ارسال میشود انجام میگیرد.

شبیه کد زیر:

```
request.addListener("data", function(chunk) {
  // called when a new chunk of data was received
});

request.addListener("end", function() {
  // called when all chunks of data have been received
});
```

سوال این است که این منطق را در کجا پیاده کنیم. در حال حاظر به شی `request` فقط در سرور دسترسی داریم - ما آنرا به روتر و مدیران درخواست ارسال ارسال نمیکنیم، درست مانند کاری که با شی `response` انجام دادیم.

به عقیده من، این وظیفه سرور`HTTP` است که تمام داده ها را از درخواست گرفته و به برنامه تحویل دهد. پیشنهاد میکنم مدیریت پردازش داده های POST را در سرور انجام دهیم و داده های نهایی را برای روتر ارسال کنیم تا مدیران در خواست خودشان برای پاسخ آنها تصمیم بگیرند.

بنابراین، رخدادهای برگشتی `data` و `end` را در سرور قرار میدهیم، تمام قطعات داده POST را در برگشتی `data` جمع میکنیم و روتر را بعد از رخداد `end` صدا میزنیم، داده ها را به روتر ارسال میکنیم و روتر آن را به مدیران رخداد تحویل میدهد.

با server.js شروع میکنیم:

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var postData = "";
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    request.setEncoding("utf8");
```

```
    request.addListener("data", function(postDataChunk) {
      postData += postDataChunk;
      console.log("Received POST data chunk '"+
      postDataChunk + "'.");
    });

    request.addListener("end", function() {
      route(handle, pathname, response, postData);
    });

  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

در کد بالا سه کار را انجام دادیم: اول از همه مشخص نمودیم که داده های های دریافتی در قالب UTF-8 باشند، یک شنونده رخداد به رخداد data اضافه نمودیم که قدم به قدم متغیر postData را با رسیدن یک قطعه جدید از داده های POST پر میکند، و بعد در رخداد end به سمت روتر رفتیم تا مطمئن شویم فقط بع از رسیدن تمام قطعات داده POST صدا میشود. همچنین داده ها را به روتر فرستادیم تا به دست مدیران رخداد برساند.

نشان دادن یک متن در کنسول بعد ار رسیدن هر داده ایده خوبی نیست ولی کمک میکند رخدادها را ببینیم.

پیشنهاد میکنم کمی با آن تفریح کنید. از متن های با حجم کم شروع کنید و در جعبه متنی وارد کنید.اگر حجم متنی که وارد میکنید زیاد باشد در قسمت نوار وضعیت مرورگرتان درصد متن ارسالی را خواهید دید.

حالا بیایید کمی برنامه را جالبتر کنیم. در صفحه /upload محتوای ارسالی را نشان خواهیم داد. برای این کار باید مقدار postData را در router.js به مدیران در خواست ارسال کنیم:

```
function route(handle, pathname, response, postData) {
  console.log("About to route a request for " + pathname);
```

```
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, postData);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}

exports.route = route;
```

و در requestHandlers.js **داده ها را برای پاسخگویی به مدیر درخواست** upload **می دهیم:**

```
function start(response, postData) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" content="text/html; '+
    'charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<textarea name="text" rows="20" cols="60"></textarea>'+
    '<input type="submit" value="Submit text" />'+
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}
```

```
function upload(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("You've sent: " + postData);
  response.end();
}

exports.start = start;
exports.upload = upload;
```

حالا میتوانیم داده های post را دریافت و به مدیران درخواست ارسال کنیم.

آخرین کار برای این قسمت: چیزی که در اینجا به روتر و مدیران درخواست میدهیم تمام داده های POST است. شاید بخواهیم فقط فیلد خاصی از POST را ارسال کنیم مانند مقدار فیلد متنی در اینجا.

حالا در مورد ماژول `querystring` میخوانیم، که در اینجا به ما کمک میکند:

```
var querystring = require("querystring");

function start(response, postData) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" content="text/html; '+
    'charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<textarea name="text" rows="20" cols="60"></textarea>'+
    '<input type="submit" value="Submit text" />'+
    '</form>'+
    '</body>'+
    '</html>';
```

```
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}

function upload(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("You've sent the text: "+
  querystring.parse(postData).text);
  response.end();
}

exports.start = start;
exports.upload = upload;
```

برای یک مطلب مقدماتی توضیح در باره مدیریت داده های POST کافی بود.

مدیریت آپلود فایل

بیایید قسمت آخر سایت را هم کامل کنیم. برنامه ما این بود که کاربر بتواند یک عکس را آپلود کرده و نتیجه را در مرورگر خود ببیند.

در اینجا دو چیز را یاد خواهیم گرفت: اولا چگونه کتابخانه های خارجی نود را نصب کنیم؟ثانیا چگونه از آن در کدهای خود استفاده کنیم؟

ماژول خارجی که امروز میخواهیم از آن استفاده کنیم node-formidable **نام دارد. این ماژول به خوبی عملیات مربوط به تبدیل فایل های دریافتی را انجام میدهد. در اینجا منظور از فایل دریافتی فقط مدیریت داده های** POST **میباشد**

برای استفاده از این ماژول باید ابتدا آنرا به ماژول های نود اضافه کنیم. نود جی اس با مدیر بسته مخصوص به خودش ارائه میشود. این مدیر NPM **نام دارد که به ما کمک میکند ماژول های خارجی نود را به راحتی نصب کنیم. فقط باید دستور زیر را در خط فرمان اجرا کنیم:**

```
npm install formidable
```

قبل از اجرای دستور بالا نیاز به ساخت یک حساب کاربری در رجیستری سایت نود داریم.پس دستور زیر را در خط فرمان وارد کنید:

```
Npm   add-user
```

یک نام کاربری و رمز عبور انتخاب کنید و در پایان ایمیل خود را وارد نمایید.حالا میتوانید دستور نصب ماژول formidable را اجرا کنید تا این ماژول برای شما نصب گردد. اگر پیامی که حاوی متن زیر باشد را در خط فرمان دریافت نمودید نصب با موفقیت انجام گرفته است.به این نکته توجه کنید که ماژول ها از سایت نود دانلود میشوند و بعد از ارسال دستور نصب باید چند لحظه منتظر بمانید.

```
formidable@1.0.11 node_module\formidable
```

حالا ماژول formidable برای ما نصب شده است. تنها چیزی که نیاز داریم این است که آن را به شکل زیر وارد برنامه خود کنیم.درست مانند کاری که با ماژول های داخلی انجام میدادیم:

```
var formidable = require("formidable");
```

حالا نیاز داریم تا فایلی با نام incomingForm که جایی برای فرم ارسالی میباشد و به ما کمک میکند تا شئ request از سرور HTTP را به فیلدها ی مجزا تجزیه کنیم.

یک مثال ازپروژه non-formidable به ما نشان میدهد که بخشهای مجزا چگونه با هم کار میکنند:

```
var formidable = require('formidable'),
    http = require('http'),
    sys = require('sys');

http.createServer(function(req, res) {
  if (req.url == '/upload' && req.method.toLowerCase() == 'post') {
    // parse a file upload
    var form = new formidable.IncomingForm();
    form.parse(req, function(err, fields, files) {
      res.writeHead(200, {'content-type': 'text/plain'});
      res.write('received upload:\n\n');
      res.end(sys.inspect({fields: fields, files: files}));
    });
```

```
    return;
  }

  // show a file upload form
  res.writeHead(200, {'content-type': 'text/html'});
  res.end(
    '<form action="/upload" enctype="multipart/form-data" '+
    'method="post">'+
    '<input type="text" name="title"><br>'+
    '<input type="file" name="upload" multiple="multiple"><br>'+
    '<input type="submit" value="Upload">'+
    '</form>'
  );
}).listen(8888);
```

اگر این کد را در یک فایل قرار بدهیم و با نود انرا اجرا کنیم به ما امکان میدهد تا یک فایل را آپلود کنیم و نتیجه آن را به صورت زیر ببینیم:

```
received upload:

{ fields: { title: '' },
  files:
   { upload:
      { size: 42625,
        path:'C:\SAEID\AppData\Local\Temp\b1b8f52c71fac866e4bee89e099cc4bf',
        name: 'SP_Fa_tbl25.jpg',
       type: 'image/jpeg',
       hash: false,
    lastModifiedDate: Mon Oct 22 2012 17:46:11 GMT+0330 (Iran Standard Time),
     _writeStream: [Object],
     length: [Getter],
     filename: [Getter],
     mime: [Getter] } } }
```

بیایید مورد آخر را بررسی کنیم: اگر یک عکس در حافظه محلی کامپیوتر ما باشد چگونه آنرا جهت ارسال از مرورگر آماده کنیم؟

به وضوح قصد داریم محتوای این فایل را در سرور Node.js ذخیره کنیم و خوشبختانه برای این کار ماژول fs در نود مهیا شده است.

حالا بیایید یک مدیر رخداد دیگری به نام /show را به requestHandlers.js اضافه نماییم که عکس موجود در /temp/test.png را در مرورگر نشان میدهد. مسلما قبل از آن باید یک عکس را در این مکان ذخیره نماییم.

حالا requestHandlers.js را به شکل زیر تغییر میدهیم:

```
var querystring = require("querystring"),
    fs = require("fs");

function start(response, postData) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" '+
    'content="text/html; charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<textarea name="text" rows="20" cols="60"></textarea>'+
    '<input type="submit" value="Submit text" />'+
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}

function upload(response, postData) {
```

```
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("You've sent the text: "+
  querystring.parse(postData).text);
  response.end();
}

function show(response, postData) {
  console.log("Request handler 'show' was called.");
  fs.readFile("/tmp/test.png", "binary", function(error, file) {
    if(error) {
      response.writeHead(500, {"Content-Type": "text/plain"});
      response.write(error + "\n");
      response.end();
    } else {
      response.writeHead(200, {"Content-Type": "image/png"});
      response.write(file, "binary");
      response.end();
    }
  });
}

exports.start = start;
exports.upload = upload;
exports.show = show;
```

همچنین باید نقشه این مدیر درخواست جدید را به آدرس /show در index.js بدهیم:

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}
handle["/"] = requestHandlers.start;
```

```
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
handle["/show"] = requestHandlers.show;

server.start(router.route, handle);
```

با راه اندازی دوباره سرور و در خواست آدرس http://localhost:8888/show باید عکس ذخیره شده در temp/test.png نشان داده شود.

حالا کارهایی که باید انجام دهیم:

- **اضافه کردن یک بخش به فرم ارسال شده برای مرورگر در درخواست start/جهت آپلود فایل.**
- **سازگار کردن node-formidable با مدیر درخواست upload جهت ذخیره فایل ارسالی در مکان temp/ .**
- **درج کردن عکس آپلود شده در کد های HTML برای آدرس upload/.**

قدم اول راحت است، بیاید نوع کد گذاری فایل HTML برای فرم آپلود را به multipart/form-data تغییر دهیم ، جعبه ورود متن را خذف کنیم و متن دکمه ارسال را به "اپلود فایل" تغییر دهیم. بایید این مرحله را با تغییر requestHandlers.js به شکل زیر تمام کنیم:

```
var querystring = require("querystring"),
    fs = require("fs");

function start(response, postData) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" '+
    'content="text/html; charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" enctype="multipart/form-data" '+
    'method="post">'+
    '<input type="file" name="upload">'+
```

```
    '<input type="submit" value="Upload file" />'+
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}

function upload(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("You've sent the text: "+
  querystring.parse(postData).text);
  response.end();
}

function show(response, postData) {
  console.log("Request handler 'show' was called.");
  fs.readFile("/tmp/test.png", "binary", function(error, file) {
    if(error) {
      response.writeHead(500, {"Content-Type": "text/plain"});
      response.write(error + "\n");
      response.end();
    } else {
      response.writeHead(200, {"Content-Type": "image/png"});
      response.write(file, "binary");
      response.end();
    }
  });
}

exports.start = start;
```

```
exports.upload = upload;
exports.show = show;
```

عالی بود. قدم بعدی کمی پیچیده میباشد. مشکل اول این است: میخواهیم مدیریت آپلود فایل را در مدیر درخواست upload **انجام دهیم و در اینجا نیاز داریم شئ** request **را به** form.pars **از** node-formidable **ارسال کنیم.**

اما تمام چیزی که داریم شئ response **و آرایه** postData **میباشد. به نظر میرسد مجبور باشیم شی** request **را همیشه از سرور به روتر و از اینجا به مدیران درخواست ارسال کنیم. شاید راه بهتری برای این کار باشد. ولی در این جا این روش مشکل ما را حل میکند. بیایید تمام کدهای مرتبط با متغیر** postData **را از سرور و مدیران درخواست حذف کنیم. برای مدیریت آپلود فایل به آن نیازی نخواهیم داشت، حتی برای ما مشکل ایجاد میکند. هم اکنون تمام رخدادهای** data **متعلق به شئ** request **را در سرور مصرف میکنیم، به این معنی که نیاز داریم این رخدادها را در** form.pars **نیز مصرف نماییم چرا که در غیر این صورت داده زیادی را از آنها دریافت نخواهد کرد. چرا که** Node.js **هیچ داده ای را بافر نمیکند.**

با server.js **شروع میکنیم- مدیریت** postData **و خط** request.setEncoding **(که قرار است توسط خود** node-formidable **مدیریت شود) را حذف میکنیم. و به جای آن شئ** request **را به روتر ارسال میکنیم:**

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");
    route(handle, pathname, response, request);
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

حالا نوبت به router.js میرسد – دیگر نیاز نداریم postData را به آن ارسال کنیم، به جای آن شئ request را میفرستیم:

```
function route(handle, pathname, response, request) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, request);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/html"});
    response.write("404 Not found");
    response.end();
  }
}

exports.route = route;
```

حالا شئ request در مدیر درخواست upload قابل استفاده میباشد. ماژول node-formidable مدیریت جزئیات مربوط به ذخیره فایل آپلود شده را در دایرکتوری محلی /temp به عهده خواهد گرفت. اما باید اطمینان پیدا کنیم که فایل به test.png تغییر نام میدهد. برای سادگی کار فقط عکس های با فرمت یا قالب JPG را مدیریت میکنیم.

در منطق تغییر نام کمی پیچیدگی وجود دارد: پیاده سازی نود برای ویندوز اجازه نمیدهد دو فایل هم نام در یک مکان داشته باشیم دلیلی که باید قبل از اجرای برنامه آن فایل test.jpg را از /temp حذف کنیم.

حالا کد مربوط به مدیریت آپلود فایل را همزمان با تغییر نام آن در requestHandlers.js قرار میدهیم:

```
var querystring = require("querystring"),
    fs = require("fs"),
    formidable = require("formidable");

function start(response) {
  console.log("Request handler 'start' was called.");

  var body = '<html>'+
    '<head>'+
```

```
    '<meta http-equiv="Content-Type" '+
    'content="text/html; charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" enctype="multipart/form-data" '+
    'method="post">'+
    '<input type="file" name="upload" multiple="multiple">'+
    '<input type="submit" value="Upload file" />'+
    '</form>'+
    '</body>'+
    '</html>';

    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(body);
    response.end();
}

function upload(response, request) {
  console.log("Request handler 'upload' was called.");

  var form = new formidable.IncomingForm();
  console.log("about to parse");
  form.parse(request, function(error, fields, files) {
    console.log("parsing done");

    /* Possible error on Windows systems:
       tried to rename to an already existing file */
    fs.rename(files.upload.path, "/temp/test.jpg", function(err) {
      if (err) {
        fs.unlink("/temp/test.jpg");
        fs.rename(files.upload.path, "/temp/test.jpg");
      }
    });
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write("received image:<br/>");
```

```
    response.write("<img src='/show' />");
    response.end();
  });
}

function show(response) {
  console.log("Request handler 'show' was called.");
  fs.readFile("/temp/test.png", "binary", function(error, file) {
    if(error) {
      response.writeHead(500, {"Content-Type": "text/plain"});
      response.write(error + "\n");
      response.end();

    } else {
      response.writeHead(200, {"Content-Type": "image/jpg"});
      response.write(file, "binary");
      response.end();
    }
  });
}

exports.start = start;
exports.upload = upload;
exports.show = show;
```

همین بود، حالا سرور را راه اندازی کنید و یک فایل تصویری با فرمت JPG را انتخاب نموده و آن را آپلود کنید، این تصویر بعد از آپلود برای شما نشان داده میشود.

پایان....

ماموریت ما در اینجا پایان یافته است. یک برنامه ساده نوشتیم ولی برای شروع خوب است.هر چند به مفاهیمی مانند ارتباط با دیتابیس، ساخت ماژول هایی که از طریق NPM قابل نصب باشند و مدیریت درخواست های GET را یاد نگرفتیم. ولی خوشبختانه جامعه نود هر روز فعال تر و

بزرگترمیشود و منابع خوبی برای آموختن آن نگارش میگردد. اینجانب هم جهت پیشرفت موازی جامعه نود فارسی تصمیم گرفتم این اثرکه تلفیقی از ترجمه مطالب موجود در اینترنت و دانش اندکی از نود است را در اختیار جامعه وب فارسی زبان قرار دهم. امیدوارم مورد پسند واقع گردد و با نظرات خود به پیشرفت آن کمک کنید. هرچند این مطلب به پایان خود رسید ولی به شروعی دیگر و مطالبی بهتر در این زمینه فکر میکنم. alidadisaeid@gmail.com

منابع

http://www.nodebeginner.org

http://nodejs.org

http://debuggable.com

جاوا اسکریپت یک زبان برنامه نویسی به شیوه اسکریپت میباشد که در سال ۱۹۹۵ توسط شرکت اینترنتی نت اسکیپ و برای طراحی صفحت داینامیک در سمت کاربر ساخته شد. این زبان شی گرا و رخداد محور است. همچنین این زبان خود با زبان c++ نوشته شده و به عنوان موتور جاوا اسکریپت در هسته مرورگرها قرار گرفته است. یکی ازمعروفترین موتورهای جاوا اسکریپت موتور V8 شرکت گوگل میباشد که در مرورگر کروم استفاده شده و به عنوان سریعترین موتور جاوااسکریپت معروف است. حال این زبان با همت شخصی به نام رایان دهل (Rayan Dahl) در سال ۲۰۱۰ جای خود را با نام Node.js دربین زبان های برنامه نویسی سمت سرور با همان موتور V8 پیدا کرد اما با سبکی جدید و قابلیت هایی فراتر...

از آن جهت که زمان زیادی از تولد این زبان جدید نمیگذرد منابع کمی برای ان یافت میشود. کتابی که هم اکنون پیش روی شماست حاصل دانش اینجانب از منابع اینترنتی میباشد که با سبکی روان از همین منابع ترجمه و ویایش شده است. امیدوارم در راه یادگیری node.js برای شما شروعی تازه باشد.خوشحال میشوم با نظرات سازنده خودتان به رفع اشکالات آن کمک کنید.