50 Recipes for Programming

Volume One

Jamie Munro

50 Recipes for Programming Node.js

Volume 1

Organized by: Jamie Munro

This book is dedicated to the user's of StackOverFlow who have asked over 180,000 questions at the time this book was organized.

Preface

About the Book

Node.js® is built on Chrome's V8 JavaScript Engine. Node.js is designed to build and execute applications using an event-driven, non-blocking Input/Output model attempting to make it lightweight and efficient. Node.js contains one of the largest open source package ecosystem using npm to create, install, and manage a variety of useful JavaScript packages that can easily be embedded into any Node application.

This book is structured in a Cookbook format featuring recipes that contain problem statements and solutions. A detailed explanation follows each problem statement of the recipe. This is usually contained within the solution; however, an optional discussion section can often contain other useful information helping to demonstrate how the solution works.

Prerequisites

To use the examples in this book, you will want to have a Node.js server up-and-running so you will be able to leverage the many samples contained within this book. I would suggest by visiting the Installing Node.js page where you can download and install the latest version for your operating system of choice.

How to contact

To comment or ask technical questions about this book, send an email to: info@nodejsrecipes.com

For thousands of more recipes, visit our online version of the site at Node.js Recipes.

Find us on Facebook: https://www.facebook.com/nodejsrecipes/

Find us on Twitter: https://twitter.com/nodejs_recipes

The Recipes

1. Managing lots of callback recursion in Nodejs
2. Clientside going serverside with node.js
3. node.js callback getting unexpected value for variable
4. What is Node.js?
5. How do I do a deferred response from a node.js express action handler?
6. MySQL 1064 error, works in command line and phpMyAdmin; not in app
7. gzip without server support?
8. Best IDE for javascript server development
9. Stream data with Node.js
10. Need help with setting up comet code
11. Long polling getting timed out from browser
12. POST request and Node.js without Nerve
13. Node.js appears to be missing the multipart module
14. How is a functional programming-based JavaScript app laid out?
15. node.js real-time game
16. When to use TCP and HTTP in node.js?
17. Why Won't the WebSocket.onmessage Event Fire?
18. What does addListener do in node.js?
19. Recommendation for integrating nodejs with php application
20. Is it possible in .NET, using C#, to achieve event based asynchronous pattern without multithreading?
21. Node.js and wss://
22. How to create a streaming API with NodeJS
23. How do I include a JavaScript file in another JavaScript file?
24. How do I escape a string for a shell command in node?
25. How do I debug Node.js applications?
26. Auto-reload of files in Node.js
27. Best server-side javascript servers
28. How would MVC-like code work in Node.js?
29. Is there a solution that lets Node.js act as an HTTP reverse proxy?
30. Nodejs in-memory storage
31. Writing files in Node.js
32. NodeJS and HTTP Client - Are cookies supported?
33. How do you get a list of the names of all files present in a directory in

Node.js?

34. Uncatchable errors in node.js
35. How can I edit on my server files without restarting nodejs when i want to see the changes?
36. nodejs and database communication - how?
37. Is there a Thrift or Cassandra client for Node.js/JavaScript
38. Could someone give me an example of node.js application
39. Node.js for lua?
40. Node.js or Erlang
41. How do I get the path to the current script with Node.js?
42. JavaScript Standard Library for V8
43. node.js:: what does hostname do in `listen` function?
44. What is the benefit of using NginX for Node.js?
45. How to check the number of open connections in node.js?
46. Preferred DOM library for parsing html pages with node.js?
47. $.getJSON from jQuery (in Rails app) not reading JSON from cross-domain Node.js/Express app / jsonp?
48. Server side javascript on Google app engine
49. How can I share code between Node.js and the browser?
50. Parse PHP Session in Javascript

Managing lots of callback recursion in Nodejs

Problem

In Nodejs, there are virtually no blocking I/O operations. This means that almost all nodejs IO code involves many callbacks. This applies to reading and writing to/from databases, files, processes, etc. A typical example of this is the following:

```
var useFile = function(filename,callback){
posix.stat(filename).addCallback(function (stats) {
posix.open(filename, process.O_RDONLY, 0666).addCallback(function (fd) {
posix.read(fd, stats.size, 0).addCallback(function(contents){
callback(contents);
});
});
});
};

...

useFile("test.data",function(data){
// use data..
});
```

I am anticipating writing code that will make **many** IO operations, so I expect to be writing **many** callbacks. I'm quite comfortable with using callbacks, but I'm worried about all the recursion. Am I in danger of running into too much recursion and blowing through a stack somewhere? If I make thousands of individual writes to my key-value store with thousands of callbacks, will my program eventually crash?

Am I misunderstanding or underestimating the impact? If not, is there a way to get around this while still using Nodejs' callback coding style?

Problem courtesy of: Maciek

Solution

None of the code you show is using recursion. When you call useFile it calls posix.stat(), which returns, and useFile terminates as it has run to completion. At some later time, when the call to posix.stat() has completed *within the underlying system* and the results are available, the callback function you added for that will be executed. That calls posix.open(), and then terminates as it has run to completion. Once the file has been successfully opened, the callback function for *that* will execute, calling posix.read(), and will then terminate as it, too, has run to completion. Finally, when the results of the read are available, the innermost function will be executed.

The important point is that each function runs to completion, as the calls to the posix.*() functions are non-blocking: that is, they return immediately, having caused some magic to be started off in the underlying system. So each of your functions terminates, and later an event will cause the next function to execute; but at no point is there any recursion.

The nested structure of the code can give one the impression that the stuff inside will have to finish before the stuff outside can get to its own end point. But in this style of asynchronous event-driven programming it makes more sense to see the nesting in terms of *deeper => happens-later-than*.

EDIT: Try adding some logging statements immediately before the end of each nested function; this will help to illustrate that the order in which they complete is from the outside inwards.

Solution courtesy of: NickFitz

Discussion

As with any JavaScript, it's possible to make recursive calls with Node.js. If you do run into recursion depth problems (as NickFitz points out, you don't seem to be in danger of that), you can often rewrite your code to use an interval timer instead.

Discussion courtesy of: Nosredna

Your stuff is fine. I do recursive calls in Express to follow HTTP redirects, but what your doing is "traversal" and not recursion

Discussion courtesy of: TJ Holowaychuk

Same example, with debug output added (see below for output):

usefile.js:

```
var sys = require("sys"),
posix = require("posix");

var useFile = function(filename,callback){
posix.stat(filename).addCallback(function (stats) {
posix.open(filename, process.O_RDONLY, 0666).addCallback(function (fd) {
posix.read(fd, stats.size, 0).addCallback(function(contents){
callback(contents);
sys.debug("useFile callback returned");
});
sys.debug("read returned");
});
sys.debug("open returned");
});
sys.debug("stat returned");
};

useFile("usefile.js",function(){});
```

Output:

```
DEBUG: stat returned
DEBUG: open returned
DEBUG: read returned
DEBUG: useFile callback returned
```

Discussion courtesy of: Jeoff Wilks

You can try

http://github.com/creationix/do

or roll your own like I did. Never mind missing error handling for now (just ignore that) ;)

```
var sys = require('sys');
```

```
var Simplifier = exports.Simplifier = function() {}

Simplifier.prototype.execute = function(context, functions, finalFunction) {
this.functions = functions;
this.results = {};
this.finalFunction = finalFunction;
this.totalNumberOfCallbacks = 0
this.context = context;
var self = this;

functions.forEach(function(f) {
f(function() {
self.totalNumberOfCallbacks = self.totalNumberOfCallbacks + 1;
self.results[f] = Array.prototype.slice.call(arguments, 0);
if(self.totalNumberOfCallbacks >= self.functions.length) {
// Order the results by the calling order of the functions
var finalResults = [];
self.functions.forEach(function(f) {
finalResults.push(self.results[f][0]);
})
// Call the final function passing back all the collected results in the right
order
finalFunction.apply(self.context, finalResults);
}
});
});
}
```

And a simple example using it

```
// Execute
new simplifier.Simplifier().execute(
// Context of execution
self,
// Array of processes to execute before doing final handling
[function(callback) {
db.collection('githubusers', function(err, collection) {
collection.find({}, {limit:30}, function(err, cursor) {
cursor.toArray(function(err, users) { callback(users); })
});
});
},

function(callback) {
db.collection('githubprojects', function(err, collection) {
collection.find({}, {limit:45, sort:[['watchers', -1]]}, function(err, cursor)
{
cursor.toArray(function(err, projects) { callback(projects); })
});
});
}
],
// Handle the final result
function(users, projects) {
// Do something when ready
}
);
```

Also take a look at 'step' (http://github.com/creationix/step) or 'flow-js' on github. This lets you write callback flows in a more natural style. This will also make it clear that there's no recursion going on.

Discussion courtesy of: Ignacio Thayer

This content originated from StackOverFlow and has been re-organized into the above recipe.

Clientside going serverside with node.js

Problem

I`ve been looking for a serverside language for some time, and python got my attention somewhat. But as I already know and love javascript, I now want learn to code on the server with js and node.js.

Now, what books and what subjects do I need to learn to understand the serverside world better?

(let me know if Im to vague)

Problem courtesy of: Sveisvei

Solution

I'm not aware of any printed resources about node.js. Some good places to start are:

- The official node.js documentation
- The web site howtonode.org with a lot of tutorials
- The video of Ryan Dahl's (node.js creator) talk at the JSConf.eu
- A nice blog post about node.js by Simon Willison
- The mailing list archive
- The source code of node.js projects on github

Solution courtesy of: Fabian Jakobs

Discussion

The HTTP/1.1 spec is very informative.

Discussion courtesy of: postfuturist

Learn python, it'll make you a better node.js programmer because it has good examples for all the networking features.

- urllib
- socket
- file

JavaScript is my favorite language, but I've spent some time in python and the standard library is really good for learning the basics of sockets, file descriptors, and networking.

Discussion courtesy of: Fire Crow

node.js callback getting unexpected value for variable

Problem

I have a for loop, and inside it a variable is assigned with var. Also inside the loop a method is called which requires a callback. Inside the callback function I'm using the variable from the loop. I would expect that it's value, inside the callback function, would be the same as it was outside the callback during that iteration of the loop. However, it always seems to be the value from the *last* iteration of the loop.

Am I misunderstanding scope in JavaScript, or is there something else wrong?

The program in question here is a node.js app that will monitor a working directory for changes and restart the server when it finds one. I'll include all of the code for the curious, but the important bit is the parse_file_list function.

```
var posix = require('posix');
var sys = require('sys');
var server;
var child_js_file = process.ARGV[2];
var current_dir = __filename.split('/');
current_dir = current_dir.slice(0, current_dir.length-1).join('/');

var start_server = function(){
server = process.createChildProcess('node', [child_js_file]);
server.addListener("output", function(data){sys.puts(data);});
};

var restart_server = function(){
sys.puts('change discovered, restarting server');
server.close();
start_server();
};

var parse_file_list = function(dir, files){
for (var i=0;i<files.length;i++){
var file = dir+'/'+files[i];
sys.puts('file assigned: '+file);
posix.stat(file).addCallback(function(stats){
sys.puts('stats returned: '+file);
if (stats.isDirectory())
posix.readdir(file).addCallback(function(files){
parse_file_list(file, files);
});
else if (stats.isFile())
process.watchFile(file, restart_server);
});
}
};

posix.readdir(current_dir).addCallback(function(files){
parse_file_list(current_dir, files);
```

```
});

start_server();
```

The output from this is:

```
file assigned: /home/defrex/code/node/ejs.js
file assigned: /home/defrex/code/node/templates
file assigned: /home/defrex/code/node/web
file assigned: /home/defrex/code/node/server.js
file assigned: /home/defrex/code/node/settings.js
file assigned: /home/defrex/code/node/apps
file assigned: /home/defrex/code/node/dev_server.js
file assigned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
stats returned: /home/defrex/code/node/main_urls.js
```

For those from the future: node.devserver.js

Problem courtesy of: defrex

Solution

This is perfectly normal behaviour. Your callbacks are executed at some later point (asynchronously) but still reference the scope your *for* loop was running in. All callback references to *file* will therefore have the last value it was set to.

What you want to do is create a new function scope, assign the current value of *file* to a local variable **and** create the callback inside that scope.

```
for (var i=0;i<files.length;i++){
var file = dir+'/'+files[i];
(function() {
var file_on_callback = file;
sys.puts('file assigned: '+ file_on_callback);
posix.stat(file_on_callback).addCallback(function(stats){
sys.puts('stats returned: '+ file_on_callback);
if (stats.isDirectory())
posix.readdir(file_on_callback).addCallback(function(files){
parse_file_list(file_on_callback, files);
});
else if (stats.isFile())
process.watchFile(file_on_callback, restart_server);
});
})(); // This creates and executes a new function with its own scope.
}
```

Solution courtesy of: Michiel Kalkman

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

What is Node.js?

Problem

I don't fully get what Node.js is all about. Maybe it's because I am mainly a web based business application developer. What is it and what is the use of it?

My understanding so far is that:

1. The programming model is event driven, especially the way it handles I/O.
2. It uses JavaScript and the parser is V8.
3. It can be easily used to create concurrent server applications.

Are my understandings correct? If yes, then what are the benefits of evented I/O, is it just more for the concurrency stuff? Also, is the direction of Node.js to become a framework like, JavaScript based (V8 based) programming model?

Problem courtesy of: Jeff

Solution

I think the advantages are:

1. Web development in a dynamic language (JavaScript) on a VM that is incredibly fast (V8). It is much faster than Ruby, Python, or Perl.

2. Ability to handle thousands of concurrent connections with minimal overhead on a single process.

3. JavaScript is perfect for event loops with first class function objects and closures. People already know how to use it this way having used it in the browser to respond to user initiated events.

4. A lot of people already know JavaScript, even people who do not claim to be programmers. It is arguably the most popular programming language.

5. Using JavaScript on a web server as well as the browser reduces the impedance mismatch between the two programming environments which can communicate data structures via JSON that work the same on both sides of the equation. Duplicate form validation code can be shared between server and client, etc.

Solution courtesy of: postfuturist

Discussion

Well, I understand that

- Node's goal is to provide an easy way to build scalable network programs.
- Node is similar in design to and influenced by systems like Ruby's Event Machine or Python's Twisted.
- Evented I/O for V8 javascript.

For me that means that you were correct in all three assumptions. The library sure looks promising!

Discussion courtesy of: nes1983

V8 is an implementation of JavaScript. It lets you run standalone JavaScript applications (among other things).

Node.js is simply a library written for V8 which does evented I/O. This concept is a bit trickier to explain, and I'm sure someone will answer with a better explanation than I... The gist is that rather than doing some input or output and waiting for it to happen, you just *don't* wait for it to finish. So for example, ask for the last edited time of a file:

```
// Pseudo code
stat( 'somefile' )
```

That might take a couple of milliseconds, or it might take seconds. With evented I/O you simply fire off the request and instead of waiting around you attach a callback that gets run when the request finishes:

```
// Pseudo code
stat( 'somefile', function( result ) {
// Use the result here
} );
// ...more code here
```

This makes it a lot like JavaScript code in the browser (for example, with Ajax style functionality).

For more information, you should check out the article *Node.js is genuinely exciting* which was my introduction to the library/platform... I found it quite good.

Discussion courtesy of: rfunduk

The closures are a way to execute code in the context it was created in.

What this means for concurency is that you can define variables, then initiate a nonblocking I/O function, and send it an anonymous function for its callback.

When the task is complete, the callback function will execute in the context with the variables, this is the closure.

The reason closures are so good for writing applications with nonblocking I/O is that it's very easy to manage the context of functions executing asynchronously.

Discussion courtesy of: Fire Crow

Node.js is an open source command line tool built for the server side JavaScript code. You can download a tarball, compile and install the source. It lets you run JavaScript programs.

The JavaScript is executed by the V8, a JavaScript engine developed by Google which is used in Chrome browser. It uses a JavaScript API to access the network and file system.

It is popular for its performance and the ability to perform parallel operations.

Understanding node.js is the best explanation of node.js I have found so far.

Following are some good articles on the topic.

- Learning Server-Side JavaScript with Node.js
- This Time, You'll Learn Node.js

Discussion courtesy of: Asif Mushtag

Two good examples are regarding how you manage templates and use progressive enhancements with it. You just need a few lightweight pieces of JavaScript code to make it work perfectly.

I strongly recommend that you watch and read these articles:

- Video glass node
- Node.js YUI DOM manipulation

Pick up any language and try to remember how you would manage your HTML file templates and what you had to do to update a single CSS class name in your DOM structure (for instance, a user clicked on a menu item and you want that marked as "selected" and update the content of the page).

With Node.js it is as simple as doing it in client-side JavaScript code. Get your DOM node and apply your CSS class to that. Get your DOM node and innerHTML your content (you will need some additional JavaScript code to do this. Read the article to know more).

Another good example, is that you can make your web page compatible both with JavaScript turned on or off with the same piece of code. Imagine you have a date selection made in JavaScript that would allow your users to pick up any date using a calendar. You can write (or use) the same piece of JavaScript code to make it work with your JavaScript turned ON or OFF.

Discussion courtesy of: Renato

I use Node.js at work, and find it to be very powerful. Forced to choose one word to describe Node.js, I'd say "interesting" (which is not a purely positive adjective). The community is vibrant and growing. JavaScript, despite its oddities can be a great language to code in. And you will daily rethink your own understanding of "best practice" and the patterns of well-structured code. There's an enormous energy of ideas flowing into Node.js right now, and working in it exposes you to all this thinking - great mental weightlifting.

Node.js in production is definitely possible, but far from the "turn-key" deployment seemingly promised by the documentation. With Node.js v0.6.x, "cluster" has been integrated into the platform, providing one of the essential building blocks, but my "production.js" script is still ~150 lines of logic to handle stuff like creating the log directory, recycling dead workers, etc. For a "serious" production service, you also need to be prepared to throttle incoming connections and do all the stuff that Apache does for PHP. To be fair, Ruby on Rails has this *exact* problem. It is solved via two complementary mechanisms: 1) Putting Ruby on Rails/Node.js behind a dedicated webserver (written in C and tested to hell and back) like Nginx (or Apache / Lighttd). The webserver can efficiently serve static content, access logging, rewrite URLs, terminate SSL, enforce access rules, and manage multiple sub-services. For requests that hit the actual node service, the webserver proxies the request through. 2) Using a framework like Unicorn that will manage the worker processes, recycle them periodically, etc. I've yet to find a Node.js serving framework that seems fully baked; it may exist, but I haven't found it yet and still use ~150 lines in my hand-rolled "production.js".

Reading frameworks like Express makes it seem like the standard practice is to just serve everything through one jack-of-all-trades Node.js service ... "app.use(express.static(__dirname + '/public'))". For lower-load services and development, that's probably fine. But as soon as you try to put big time load on your service and have it run 24/7, you'll quickly discover the motivations that push big sites to have well baked, hardened C-code like Nginx fronting their site and handling all of the static content requests (...until you set up a CDN, like Amazon CloudFront)). For a somewhat humorous and unabashedly negative take on this, see this guy.

Node.js is also finding more and more non-service uses. Even if you are using something else to serve web content, you might still use Node.js as a build tool, using npm modules to organize your code, Browserify to stitch it into a single asset, and uglify-js to minify it for deployment. For dealing with the web, JavaScript is a perfect impedance match and frequently that makes it the easiest route of attack. For example, if you want to grovel through a bunch of JSON response payloads, you should use my underscore-CLI module, the utility-belt of structured data.

Pros / Cons:

- Pro: For a server guy, writing JavaScript on the backend has been a "gateway drug" to learning modern UI patterns. I no longer dread writing client code.
- Pro: Tends to encourage proper error checking (err is returned by virtually all callbacks, nagging the programmer to handle it; also, async.js and other libraries handle the "fail if any of these subtasks fails" paradigm much better than typical synchronous code)
- Pro: Some interesting and normally hard tasks become trivial - like getting status on tasks in flight, communicating between workers, or sharing cache state

- Pro: Huge community and tons of great libraries based on a solid package manager (npm)
- Con: JavaScript has no standard library. You get so used to importing functionality that it feels weird when you use JSON.parse or some other build in method that doesn't require adding an npm module. This means that there are five versions of everything. Even the modules included in the Node.js "core" have five more variants should you be unhappy with the default implementation. This leads to rapid evolution, but also some level of confusion.

Versus a simple one-process-per-request model (LAMP):

- Pro: Scalable to thousands of active connections. Very fast and very efficient. For a web fleet, this could mean a 10X reduction in the number of boxes required versus PHP or Ruby
- Pro: Writing parallel patterns is easy. Imagine that you need to fetch three (or N) blobs from Memcached. Do this in PHP ... did you just write code the fetches the first blob, then the second, then the third? Wow, that's slow. There's a special PECL module to fix that specific problem for Memcached, but what if you want to fetch some Memcached data in parallel with your database query? In Node.js, because the paradigm is asynchronous, having a web request do multiple things in parallel is very natural.
- Con: Asynchronous code is fundamentally more complex than synchronous code, and the up-front learning curve can be hard for developers without a solid understanding of what concurrent execution actually means. Still, it's vastly less difficult than writing any kind of multithreaded code with locking.
- Con: If a compute-intensive request runs for, for example, 100 ms, it will stall processing of other requests that are being handled in the same Node.js process ... AKA, cooperative-multitasking. This can be mitigated with the Web Workers pattern (spinning off a subprocess to deal with the expensive task). Alternatively, you could use a large number of Node.js workers and only let each one handle a single request concurrently (still fairly efficient because there is no process recycle).
- Con: Running a production system is MUCH more complicated than a CGI model like Apache + PHP, Perl, Ruby, etc. Unhandled exceptions will bring down the entire process, necessitating logic to restart failed workers (see *cluster*). Modules with buggy native code can hard-crash the process. Whenever a worker dies, any requests it was handling are dropped, so one buggy API can easily degrade service for other cohosted APIs.

Versus writing a "real" service in Java / C# / C (C? really?)

- Pro: Doing asynchronous in Node.js is easier than doing thread-safety anywhere else and arguably provides greater benefit. Node.js is by far the least painful asynchronous paradigm I've ever worked in. With good libraries, it is only slightly harder than writing synchronous code.
- Pro: No multithreading / locking bugs. True, you invest up front in writing

more verbose code that expresses a proper asynchronous workflow with no blocking operations. And you need to write some tests and get the thing to work (it is a scripting language and fat fingering variable names is only caught at unit-test time). BUT, once you get it to work, the surface area for heisenbugs -- strange problems that only manifest once in a million runs -- that surface area is just much much lower. The taxes writing Node.js code are heavily front-loaded into the coding phase. Then you tend to end up with stable code.

- Pro: JavaScript is much more lightweight for expressing functionality. It's hard to prove this with words, but JSON, dynamic typing, lambda notation, prototypal inheritance, lightweight modules, whatever ... it just tends to take less code to express the same ideas.
- Con: Maybe you really, really like coding services in Java?

For another perspective on JavaScript and Node.js, check out *From Java to Node.js*, a blog post on a Java developer's impressions and experiences learning Node.js.

Modules When considering node, keep in mind that your choice of JavaScript libraries will *DEFINE* your experience. Most people use at least two, an asynchronous pattern helper (Step, Futures, Async), and a JavaScript sugar module (Underscore.js).

Helper / JavaScript Sugar:

- Underscore.js - use this. Just do it. It makes your code nice and readable with stuff like _.isString(), and _.isArray(). I'm not really sure how you could write safe code otherwise. Also, for enhanced command-line-fu, check out my own Underscore-CLI.

Asynchronous Pattern Modules:

- Step - a very elegant way to express combinations of serial and parallel actions. My personal reccomendation. See my post on what Step code looks like.
- Futures - much more flexible (is that really a good thing?) way to express ordering through requirements. Can express things like "start a, b, c in parallel. When A, and B finish, start AB. When A, and C finish, start AC." Such flexibility requires more care to avoid bugs in your workflow (like never calling the callback, or calling it multiple times). See Raynos's post on using futures (this is the post that made me "get" futures).
- Async - more traditional library with one method for each pattern. I started with this before my religious conversion to step and subsequent realization that all patterns in Async could be expressed in Step with a single more readable paradigm.
- TameJS - Written by OKCupid, it's a precompiler that adds a new language primative "await" for elegantly writing serial and parallel workflows. The pattern looks amazing, but it does require pre-compilation. I'm still making up my mind on this one.

- StreamlineJS - competitor to TameJS. I'm leaning toward Tame, but you can make up your own mind.

  Or to read all about the asynchronous libraries, see this panel-interview with the authors.

  Web Framework:

- Express Great Ruby on Rails-esk framework for organizing web sites. It uses JADE as a XML/HTML templating engine, which makes building HTML far less painful, almost elegant even.
- jQuery While not technically a node module, jQuery is quickly becoming a de-facto standard for client-side user interface. jQuery provides CSS-like selectors to 'query' for sets of DOM elements that can then be operated on (set handlers, properties, styles, etc). Along the same vein, Twitter's Bootstrap CSS framework, Backbone.js for an MVC pattern, and Browserify.js to stitch all your JavaScript files into a single file. These modules are all becoming de-facto standards so you should at least check them out if you haven't heard of them.

  Testing:

- JSHint - Must use; I didn't use this at first which now seems incomprehensible. JSLint adds back a bunch of the basic verifications you get with a compiled language like Java. Mismatched parenthesis, undeclared variables, typeos of many shapes and sizes. You can also turn on various forms of what I call "anal mode" where you verify style of whitespace and whatnot, which is OK if that's your cup of tea -- but the real value comes from getting instant feedback on the exact line number where you forgot a closing ")" ... without having to run your code and hit the offending line. "JSHint" is a more-configurable variant of Douglas Crockford's JSLint.
- Mocha competitor to Vows which I'm starting to prefer. Both frameworks handle the basics well enough, but complex patterns tend to be easier to express in Mocha.
- Vows Vows is really quite elegant. And it prints out a lovely report (--spec) showing you which test cases passed / failed. Spend 30 minutes learning it, and you can create basic tests for your modules with minimal effort.
- Zombie - Headless testing for HTML and JavaScript using JSDom as a virtual "browser". Very powerful stuff. Combine it with Replay to get lightning fast deterministic tests of in-browser code.
- A comment on how to "think about" testing:
  - Testing is non-optional. With a dynamic language like JavaScript, there are *very* few static checks. For example, passing two parameters to a method that expects 4 won't break until the code is executed. Pretty low bar for creating bugs in JavaScript. Basic tests are essential to making up the verification gap with compiled languages.
  - Forget validation, just make your code execute. For every method, my first

validation case is "nothing breaks", and that's the case that fires most often. Proving that your code runs without throwing catches 80% of the bugs and will do so much to improve your code confidence that you'll find yourself going back and adding the nuanced validation cases you skipped.

- Start small and break the inertial barrier. We are all lazy, and pressed for time, and it's easy to see testing as "extra work". So start small. Write test case 0 - load your module and report success. If you force yourself to do just this much, then the inertial barrier to testing is broken. That's <30 min to do it your first time, including reading the documentation. Now write test case 1 - call one of your methods and verify "nothing breaks", that is, that you don't get an error back. Test case 1 should take you less than one minute. With the inertia gone, it becomes easy to incrementally expand your test coverage.
- Now evolve your tests with your code. Don't get intimidated by what the "correct" end-to-end test would look like with mock servers and all that. Code starts simple and evolves to handle new cases; tests should too. As you add new cases and new complexity to your code, add test cases to exercise the new code. As you find bugs, add verifications and / or new cases to cover the flawed code. When you are debugging and lose confidence in a piece of code, go back and add tests to prove that it is doing what you think it is. Capture strings of example data (from other services you call, websites you scrape, whatever) and feed them to your parsing code. A few cases here, improved validation there, and you will end up with highly reliable code.

Also, check out the official list of recommended Node.js modules. However, GitHub's Node Modules Wiki is much more complete and a good resource.

To understand Node, it's helpful to consider a few of the key design choices:

Node.js is **EVENT BASED** and **ASYNCHRONOUS** / **NON-BLOCKING**. Events, like an incoming HTTP connection will fire off a JavaScript function that does a little bit of work and kicks off other asynchronous tasks like connecting to a database or pulling content from another server. Once these tasks have been kicked off, the event function finishes and Node.js goes back to sleep. As soon as something else happens, like the database connection being established or the external server responding with content, the callback functions fire, and more JavaScript code executes, potentially kicking off even more asynchronous tasks (like a database query). In this way, Node.js will happily interleave activities for multiple parallel workflows, running whatever activities are unblocked at any point in time. This is why Node.js does such a great job managing thousands of simultaneous connections.

Why not just use one process/thread per connection like everyone else? In Node.js, a new connection is just a very small heap allocation. Spinning up a new process takes significantly more memory, a megabyte on some platforms. But the real cost is the overhead associated with context-switching. When you have 10^6 kernel threads, the kernel has to do a lot of work figuring out who should execute next. A bunch of work has gone into building an O(1) scheduler for Linux, but in the end, it's just way way more efficient to have a single event-driven process than 10^6 processes competing for CPU time. Also, under overload conditions, the multi-process model behaves very poorly, starving critical

administration and management services, especially SSHD (meaning you can't even log into the box to figure out how screwed it really is).

Node.js is **SINGLE THREADED** and **LOCK FREE**. Node.js, as a very deliberate design choice only has a single thread per process. Because of this, it's fundamentally impossible for multiple threads to access data simultaneously. Thus, no locks are needed. Threads are hard. Really really hard. If you don't believe that, you haven't done enough threaded programming. Getting locking right is hard and results in bugs that are really hard to track down. Eliminating locks and multi-threading makes one of the nastiest classes of bugs just go away. This might be the single biggest advantage of node.

But how do I take advantage of my 16 core box?

Two ways:

1. For big heavy compute tasks like image encoding, Node.js can fire up child processes or send messages to additional worker processes. In this design, you'd have one thread managing the flow of events and N processes doing heavy compute tasks and chewing up the other 15 CPUs.
2. For scaling throughput on a webservice, you should run multiple Node.js servers on one box, one per core, using cluster (With Node.js v0.6.x, the official "cluster" module linked here replaces the learnboost version which has a different API). These local Node.js servers can then compete on a socket to accept new connections, balancing load across them. Once a connection is accepted, it becomes tightly bound to a single one of these shared processes. In theory, this sounds bad, but in practice it works quite well and allows you to avoid the headache of writing thread-safe code. Also, this means that Node.js gets excellent CPU cache affinity, more effectively using memory bandwidth.

Node.js lets you do some really powerful things without breaking a sweat. Suppose you have a Node.js program that does a variety of tasks, listens on a TCP port for commands, encodes some images, whatever. With five lines of code, you can add in an HTTP based web management portal that shows the current status of active tasks. This is EASY to do:

```
var http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end(myJavascriptObject.getSomeStatusInfo());
}).listen(1337, "127.0.0.1");
```

Now you can hit a URL and check the status of your running process. Add a few buttons, and you have a "management portal". If you have a running Perl / Python / Ruby script, just "throwing in a management portal" isn't exactly simple.

But isn't JavaScript slow / bad / evil / spawn-of-the-devil? JavaScript has some weird oddities, but with "the good parts" there's a very powerful language there, and in any case, JavaScript is THE language on the client (browser). JavaScript is here to stay; other languages are targeting it as an IL, and world class talent is competing to produce the most advanced JavaScript engines. Because of JavaScript's role in the browser, an enormous amount of

engineering effort is being thrown at making JavaScript blazing fast. V8 is the latest and greatest javascript engine, at least for this month. It blows away the other scripting languages in both efficiency AND stability (looking at you, Ruby). And it's only going to get better with huge teams working on the problem at Microsoft, Google, and Mozilla, competing to build the best JavaScript engine (It's no longer a JavaScript "interpreter" as all the modern engines do tons of JIT compiling under the hood with interpretation only as a fallback for execute-once code). Yeah, we all wish we could fix a few of the odder JavaScript language choices, but it's really not that bad. And the language is so darn flexible that you really aren't coding JavaScript, you are coding Step or jQuery -- more than any other language, in JavaScript, the libraries define the experience. To build web applications, you pretty much have to know JavaScript anyway, so coding with it on the server has a sort of skill-set synergy. It has made me not dread writing client code.

Besides, if you REALLY hate JavaScript, you can use syntactic sugar like CoffeeScript. Or anything else that creates JavaScript code, like Google Web Toolkit (GWT).

Speaking of JavaScript, what's a "closure"? - Pretty much a fancy way of saying that you retain lexically scoped variables across call chains. ;) Like this:

```
var myData = "foo";
database.connect( 'user:pass', function myCallback( result ) {
database.query("SELECT * from Foo where id = " + myData);
} );
// Note that doSomethingElse() executes _BEFORE_ "database.query" which is inside a callback
doSomethingElse();
```

See how you can just use "myData" without doing anything awkward like stashing it into an object? And unlike in Java, the "myData" variable doesn't have to be read-only. This powerful language feature makes asynchronous-programming much less verbose and less painful.

Writing asynchronous code is always going to be more complex than writing a simple single-threaded script, but with Node.js, it's not that much harder and you get a lot of benefits in addition to the efficiency and scalability to thousands of concurrent connections...

Discussion courtesy of: Dave Dopson

Also, do not forget to mention that Google's V8 is VERY fast. It actually converts the JavaScript code to machine code with the matched performance of compiled binary. So along with all the other great things, it's INSANELY fast.

Discussion courtesy of: Quinton Pike

There is a very good fast food place analogy that best explains the event driven model of Node.js, see the full article, *Node.js, Doctor's Offices and Fast Food Restaurants – Understanding Event-driven Programming*

Here is a summary:

If the fast food joint followed a traditional thread-based model, you'd order your food and wait in line until you received it. The person behind you wouldn't be able to order until your order was done. In an event-driven model, you order your food and then get out of line to wait. Everyone else is then

free to order.

Node.js is event-driven, but most web servers are thread-based.York explains how Node.js works:

- You use your web browser to make a request for "/about.html" on a Node.js web server.
- The Node.js server accepts your request and calls a function to retrieve that file from disk.
- While the Node.js server is waiting for the file to be retrieved, it services the next web request.
- When the file is retrieved, there is a callback function that is inserted in the Node.js servers queue.
- The Node.js server executes that function which in this case would render the "/about.html" page and send it back to your web browser."

Discussion courtesy of: adeleinr

Q: The programming model is event driven, especially the way it handles I/O.

Correct. It uses call-backs, so any request to access the file system would cause a request to be sent to the file system and then Node.js would start processing its next request. It would only worry about the I/O request once it gets a response back from the file system, at which time it will run the callback code. However, it is possible to make synchronous I/O requests (that is, blocking requests). It is up to the developer to choose between asynchronous (callbacks) or synchronous (waiting).

Q: It uses JavaScript and the parser is V8.

Yes

Q: It can be easily used to create concurrent server applications.

Yes, although you'd need to hand-code quite a lot of JavaScript. It might be better to look at a framework, such as http://www.easynodejs.com/ - which comes with full online documentation and a sample application.

Discussion courtesy of: Charles

How do I do a deferred response from a node.js express action handler?

Problem

Using Express (for node.js) How do I write a response after a callback?

The following is a minimal example. posix.cat is a function that returns a promise, up does something to the result, and I want to send that as the response.

```
require.paths.unshift('lib');
require('express');
var posix = require('posix');

get('/', function () {
function up(s) {
return s.toUpperCase();
}
return posix.cat('/etc/motd').addCallback(up);
});

run(3001);
```

The client never gets a response.

I've also tried variations on this:

```
get('/2', function () {
var myRequest = this;
function up(s) {
myRequest.respond(s.toUpperCase());
}
return posix.cat('/etc/motd').addCallback(up);
});
```

but that tends to crash everything:

```
[object Object].emitSuccess (node.js:283:15)
[object Object].<anonymous> (node.js:695:21)
[object Object].emitSuccess (node.js:283:15)
node.js:552:29
node.js:1027:1
node.js:1031:1
```

Problem courtesy of: keturn

Solution

According to this, the problem with approach #2 that respond was called before a status code was set.

This works:

```
get('/2', function () {
var myRequest = this;
function up(s) {
myRequest.halt(200, s.toUpperCase());
}
return posix.cat('/etc/motd').addCallback(up);
});
```

Solution courtesy of: keturn

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

MySQL 1064 error, works in command line and phpMyAdmin; not in app

Problem

Here is my query:

```
select *
from (select *, 3956 * 2 * ASIN(SQRT(POWER(SIN(RADIANS(45.5200077 - lat)/ 2),
2)      +      COS(RADIANS(45.5200077))      *      COS(RADIANS(lat))      *
POWER(SIN(RADIANS(-122.6942014 - lng)/2),2))) AS distance
from stops
order by distance, route asc) as p
group by route, dir
order by distance asc
limit 10
```

This works fine at the command line and in PHPMyAdmin. I'm using Dbslayer to connect to MySQL via my JavaScript backend, and the request is returning a 1064 error.

Here is the encoded DBSlayer request string:

```
http://localhost:9090/db?
{%22SQL%22:%22select%20*%20from%20%28select%20*,%203956%20*%202%20*%20ASIN%28SQR
%20lat%29/%202%29,%202%29%20+%20COS%28RADIANS%2845.5200077%29%29%20*%20COS%28RAD
122.6942014%20-
%20lng%29/2%29,2%29%29%29%20AS%20distance%20from%20%60stops%60%20order%20by%20%6
```

And the response:

```
{"MYSQL_ERRNO" : 1064 , "MYSQL_ERROR" : "You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server version for the right
syntax   to   use   near   '(RADIANS(45.5200077))   *   COS(RADIANS(lat))   *
POWER(SIN(RADIANS(-122.6942014 - lng' at line 1" , "SERVER" : "trimet"}
```

Thanks!

Problem courtesy of: user128970

Solution

A possible immediate source of your problem is the URL encoding. I see the plus operator is transmitted as-is. That's dangerous, because + used to mean *space* in the traditional encoding. http://www.faqs.org/rfcs/rfc1738

Solution courtesy of: just somebody

Discussion

...distance%60%20asc%20...

How are you escaping the sql? Looks like you're missing the "route", so the above reads "distance, asc". Looks like it goes downhill from there.

Discussion courtesy of: Todd R

gzip without server support?

Problem

I have written a css server which does minimization and basic parsing/var replacement. The server is using node.js.

I am wanting to gzip my response from this server. As told in IRC, node.js does not currently have a gzip lib, so I am attempting to do it manually from the command line (as I am only gzipping when not in cache).

I am pushing the file data out to a temp file and then using exec to call 'gzip -c -9 -q ' + tempFile. I get the compressed data back correctly (it seems), and send the proper Content-Encoding header as 'gzip', but Chrome reports:

Error 330 (net::ERR_CONTENT_DECODING_FAILED): Unknown error.

Also some independant gzip testers online fail as well (not just Chrome).

I'm assuming this is something simple I do not know about generating gzip blocks for browsers, seeing as I have never tried to do it manually.

Any assistance would be helpful. The server is blazing fast, but I need to gzip the content to get best performance for end users.

Thanks.

UPDATE I have verified my Content-Length is correct

Problem courtesy of: Spot

Solution

Have you updated the Content-Length to match the gzipped size? It seems like that might screw up the decoding.

Solution courtesy of: Sionide21

Discussion

Node is still bleeding edge and seems not yet to have a good handling of binary data.

Node's string encodings are ascii, binary and utf8. [...] "binary" only look[s] at the **first 8 bits** of the 16bit JavaScript string characters. The problem is that strings according to ECMA are 16bit character strings. If you use UTF-8 (it's the default) there is some normalization when reading into the string, and this corrupts gzip. If you use ascii, it obviously won't work.

It will work if you use **binary encoding *both* reading and writing**. The upper 8 bits of a Javascript string character just are not being used. If not, try to send the files directly to client without any loading into Javascript strings, perhaps with the help of a proxy server in front of Node.

I myself hope that Google's V8 engine implements a true binary string datatype, something like this proposal http://groups.google.com/group/nodejs/browse_thread/thread/648a0f5ed2c95211/ef89lnk=gst&q=binary+type#ef89acfe538931a1

CommonJS is also proposing Binary/B, and since Node tries to follow CommonJS, there is some hope for the future.

Edit I just discovered the net2 branch of node which contains a binary buffer (see src/node_buffer.h). It is part of a complete overhaul of network it seems.

Discussion courtesy of: nalply

Best IDE for javascript server development

Problem

After reading http://www.pragprog.com/magazines/2010-03/javascript-its-not-just-for-browsers-any-more im wondering which is the best IDE to develop server-side javascript applications?

I want a nice development environment with commonjs and node etc. Preferebly windows but anything is interesting really. Is there any IDE with some nifty refactoring tools, maybe some intellisense-like function, etc etc.. Or is it notepad++ ftw?

Problem courtesy of: MatteS

Solution

I use Aptana 1.5 its very similar to eclipse which I use @ my work every day

Solution courtesy of: ant

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

Stream data with Node.js

Problem

I want to know if it is possible to stream data from the server to the client with Node.js. From what I can understand all over the internet is that this has to be possible, yet I fail to find a correct example or solution.

What I want is a single http post to node.js with AJAX. Than leave the connection open and continuously stream data to the client.

The client will receive this stream and update the page continuously.

Update 1

As an update to this answer: Stream data with Node.js

I tried this already and I tried it again after your anser. I can not get this to work. The response.write is not send before you call close. I have set up an example program that i use to achieve this.

Node.js code:

```
var sys = require('sys'),
http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
var currentTime = new Date();
setInterval(function(){
res.write(
currentTime.getHours()
+ ':' +
currentTime.getMinutes()
+ ':' +
currentTime.getSeconds()
);
},1000);
}).listen(8000);
```

Here the html code:

```
<html>
<head>
<title>Testnode</title>
</head>
<body>

<!-- This fields needs to be updated -->
Server time: <span id="time"> </span>

<!-- import jQuery from google -->
<script
type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>

<!-- import jQuery -->
<script type="text/javascript">
$(document).ready(function(){
// I call here node.localhost nginx ports this to port 8000
$('#time').load('http://node.localhost');
});
</script>
</body>
</html>
```

Using this method I don't get anything back until I call close(). is this possible or should I rather go with a long poll approach where I call the load function again as one comes in.

i really rather want to stream this. I hope someone can help.

Problem courtesy of: Saif Bechan

Solution

It is possible. Just use response.write() multiple times.

```
var body = ["hello world", "early morning", "richard stallman", "chunky bacon"];
// send headers
response.writeHead(200, {
"Content-Type": "text/plain"
});

// send data in chunks
for (piece in body) {
response.write(body[piece], "ascii");
}

// close connection
response.end();
```

You may have to close and reopen connection every 30 seconds or so.

EDIT: this is the code I actually tested:

```
var sys = require('sys'),
http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
var currentTime = new Date();
sys.puts('Starting sending time');
setInterval(function(){
res.write(
currentTime.getHours()
+ ':' +
currentTime.getMinutes()
+ ':' +
currentTime.getSeconds() + "\n"
);

setTimeout(function() {
res.end();
}, 10000);

},1000);
}).listen(8090, '192.168.175.128');
```

I connected to it by Telnet and its indeed gives out chunked response. But to use it in AJAX browser has to support XHR.readyState = 3 (partial response). Not all browsers support this, as far as I know. So you better use long polling (or Websockets for Chrome/Firefox).

EDIT2: Also, if you use nginx as reverse proxy to Node, it sometimes wants to gather all chunks and send it to user at once. You need to tweak it.

Solution courtesy of: Kuroki Kaze

Discussion

Look at Sockets.io. It provides HTTP/HTTPS streaming and uses various transports to do so:

- WebSocket
- WebSocket over Flash (+ XML security policy support)
- XHR Polling
- XHR Multipart Streaming
- Forever Iframe
- JSONP Polling (for cross domain)

And! It works seamlessly with Node.JS. It's also an NPM package.

https://github.com/LearnBoost/Socket.IO

https://github.com/LearnBoost/Socket.IO-node

Discussion courtesy of: BMiner

You can also abort the infinite loop:

```
app.get('/sse/events', function(req, res) {
res.header('Content-Type', 'text/event-stream');

var interval_id = setInterval(function() {
res.write("some data");
}, 50);

req.socket.on('close', function() {
clearInterval(interval_id);
});
});
```

This is an example of expressjs. I believe that without expressjs will be something like.

Discussion courtesy of: Andrey Antukh

Need help with setting up comet code

Problem

Does anyone know off a way or maybe think its possible to connect Node.js with Nginx http push module to maintain a persistent connection between client and browser.

I am new to comet so just don't understand the publishing etc maybe someone can help me with this.

What i have set up so far is the following. I downloaded the jQuery.comet plugin and set up the following basic code:

Client JavaScript

```
<script type="text/javascript">

function updateFeed(data) {
$('#time').text(data);
}

function catchAll(data, type) {
console.log(data);
console.log(type);
}

$.comet.connect('/broadcast/sub?channel=getIt');
$.comet.bind(updateFeed, 'feed');
$.comet.bind(catchAll);

$('#kill-button').click(function() {
$.comet.unbind(updateFeed, 'feed');
});
</script>
```

What I can understand from this is that the client will keep on listening to the url followed by /broadcast/sub=getIt. When there is a message it will fire updateFeed.

Pretty basic and understandable IMO.

Nginx http push module config

```
default_type application/octet-stream; sendfile on; keepalive_timeout 65; push_authorized_channels_only off;

server {
listen 80;
location /broadcast {
location = /broadcast/sub {
set $push_channel_id $arg_channel;
push_subscriber;
push_subscriber_concurrency broadcast;
push_channel_group broadcast;
}

location = /broadcast/pub {
set $push_channel_id $arg_channel;
push_publisher;
push_min_message_buffer_length 5;
push_max_message_buffer_length 20;
push_message_timeout 5s;
push_channel_group broadcast;
}
}
}
```

Ok now this tells nginx to listen at port 80 for any calls to /broadcast/sub and it will give back any responses sent to /broadcast/pub.

Pretty basic also. This part is not so hard to understand, and is well documented over the internet. Most of the time there is a ruby or a php file behind this that does the broadcasting.

My idea is to have node.js broadcasting /broadcast/pub. I think this will let me have persistent streaming data from the server to the client without breaking the connection. I tried the long-polling approach with looping the request but I think this will be more efficient.

Or is this not going to work.

Node.js file

Now to create the Node.js i'm lost. First off all I don't know how to have node.js to work in this way.

The setup I used for long polling is as follows:

```
var sys = require('sys'),
http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.write(new Date());
res.close();
seTimeout('',1000);
}).listen(8000);
```

This listens to port 8000 and just writes on the response variable.

For long polling my nginx.config looked something like this:

```
server {
listen 80;
server_name _;

location / {
proxy_pass http://mydomain.com:8080$request_uri;
include /etc/nginx/proxy.conf;
}
}
```

This just redirected the port 80 to 8000 and this worked fine.

Does anyone have an idea on how to have Node.js act in a way Comet understands it. Would be really nice and you will help me out a lot.

Recources

used

- An example where this is done with ruby instead of Node.js
- jQuery.comet
- Nginx HTTP push module homepage
- Faye: a Comet client and server for Node.js and Rack

To use faye I have to install the comet client, but I want to use the one supplied with Nginx. Thats why I don't just use faye. The one nginx uses is much more optimzed.

extra

- Persistant connections
- Going evented with Node.js

Problem courtesy of: Saif Bechan

Solution

Looking at your link it seems to me that all pub/sub work is done by Nginx, Ruby code is used only to test it and send messages. Clients are still using long polling:

In the script above, every five seconds a publisher emits a new event to our Nginx server, which in turn, pushes the data to two subscribers **which have long-polling connections open** and are waiting for data. Once the message is sent to each subscriber, Nginx closes their connections and the clients then immediately **re-establish them** to wait for the next available message.

Nginx serves as simple retranslator for messages (very clever setup BTW, thanks for the link).

To put it short: browsers aren't capable for this sort of connection you trying to do. This is what WebSockets were invented for.

Later i'll make some code in Node.js to use this setup with Nginx (I'm interested in it too).

Solution courtesy of: Kuroki Kaze

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

Long polling getting timed out from browser

Problem

I'm trying to serve long polling requests for 60 secs using node.js. The problem I'm facing is, the browser is getting timed out. The same setup is working for 30 secs. Can anybody suggest how to achieve this? Using JQuery as JS framework.

Thanks...

Problem courtesy of: intellidiot

Solution

By default, node.js has a 60 second timeout for TCP/IP connections. You can get around this by explicitly setting the timeout. Here's a quick example:

```
http.createServer(function (req, res) {
// Connection now times out after 120 seconds
req.connection.setTimeout(120000);
// ... TODO: server logic ...
}).listen(8000);
```

You can tell node to hold the connection open indefinitely by setting to the timeout to 0. Also, note that the default 60 second timeout applies to all socket connections in addition to TCP/IP.

Solution courtesy of: Xavi

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

POST request and Node.js without Nerve

Problem

Is there any way to accept POST type requests without using Nerve lib in Node.js?

Problem courtesy of: intellidiot

Solution

By default the **http.Server** class of Node.js accepts any http method.
You can get the method using request.method (api link).

Example:

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

http.createServer(function (request, response) {
response.writeHead(200, {'Content-Type': 'text/plain'});
response.write(request.method);
response.end();
}).listen(8000);

sys.puts('Server running at http://127.0.0.1:8000/');
```

This will create a simple http server on the port 8000 that will echo the method used in the request.

If you want to get a *POST* you should just check the request.method for the string "POST".

Update regarding response.end:

Since version 0.1.90, the function to close the response is response.end instead of response.close. Besides the name change, end can also send data and close the response after this data is sent unlike close. (api example)

Solution courtesy of: Diogo Gomes

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

Node.js appears to be missing the multipart module

Problem

I am trying to parse form data, including upload files with a node.js http server. All of the tutorial type articles I have found use a require("multipart"); to include the multipart module, but when I try the same I get:

```
Error: Cannot find module 'multipart'
```

I also can't find it in the current api docs (though it is in the google cached version). So, has this module been removed from the standard installation or is there something else that does the job?

Problem courtesy of: Brenton Alker

Solution

Multipart.js is now standalone module.

Solution courtesy of: Kuroki Kaze

Discussion

There is a better multipart parser now. You can find it at: http://github.com/felixge/node-formidable

Discussion courtesy of: kvz

Run a cmd window, there go to the folder that contains the files where you are trying to use the multipart module. Then, run the following command:

```
npm install multipart
```

That's all.

Discussion courtesy of: Matías Emanuel Toro

This content originated from StackOverFlow and has been re-organized into the above recipe.

How is a functional programming-based JavaScript app laid out?

Problem

I've been working with node.js for a while on a chat app (I know, very original, but I figured it'd be a good learning project). Underscore.js provides a lot of functional programming concepts which look interesting, so I'd like to understand how a functional program in JavaScript would be setup.

From my understanding of functional programming (which may be wrong), the whole idea is to avoid side effects, which are basically having a function which updates another variable outside of the function so something like

```
var external;
function foo() {
external = 'bar';
}
foo();
```

would be creating a side effect, correct? So as a general rule, you want to avoid disturbing variables in the global scope.

Ok, so how does that work when you're dealing with objects and what not? For example, a lot of times, I'll have a constructor and an init method that initializes the object, like so:

```
var Foo = function(initVars) {
this.init(initVars);
}

Foo.prototype.init = function(initVars) {
this.bar1 = initVars['bar1'];
this.bar2 = initVars['bar2'];
//....
}

var myFoo = new Foo({'bar1': '1', 'bar2': '2'});
```

So my init method is intentionally causing side effects, but what would be a functional way to handle the same sort of situation?

Also, if anyone could point me to either a Python or JavaScript source code of a program that tries to be as functional as possible, that would also be much appreciated. I feel like I'm close to "getting it", but I'm just not quite there. Mainly I'm interested in how functional programming works with traditional OOP classes concept (or does away with it for something different if that's the case).

Problem courtesy of: user126715

Solution

You should read this question:

Javascript as a functional language

There are lots of useful links, including:

- Use functional programming techniques to write elegant JavaScript
- The Little JavaScripter
- Higher-Order JavaScript
- Eloquent JavaScript, Chapter 6: Functional Programming

Now, for my opinion. A lot of people misunderstand JavaScript, possibly because its syntax looks like most other programming languages (where Lisp/Haskell/OCaml look completely different). JavaScript is *not* object-oriented, it is actually a prototype-based language. It doesn't have classes or classical inheritance so shouldn't really be compared to Java or C++.

JavaScript can be better compared to a Lisp; it has closures and first-class functions. Using them you can create other functional programming techniques, such as partial application (currying).

Let's take an example (using sys.puts from node.js):

```
var external;
function foo() {
external = Math.random() * 1000;
}
foo();

sys.puts(external);
```

To get rid of global side effects, we can wrap it in a closure:

```
(function() {
var external;
function foo() {
external = Math.random() * 1000;
}
foo();

sys.puts(external);
})();
```

Notice that we can't actually do anything with external or foo outside of the scope. They're completely wrapped up in their own closure, untouchable.

Now, to get rid of the external side-effect:

```
(function() {
function foo() {
return Math.random() * 1000;
}
```

```
sys.puts(foo());
})();
```

In the end, the example is not purely-functional because it **can't** be. Using a random number reads from the global state (to get a seed) and printing to the console is a side-effect.

I also want to point out that mixing functional programming with objects is perfectly fine. Take this for example:

```
var Square = function(x, y, w, h) {
this.x = x;
this.y = y;
this.w = w;
this.h = h;
};

function getArea(square) {
return square.w * square.h;
}

function sum(values) {
var total = 0;

values.forEach(function(value) {
total += value;
});

return total;
}

sys.puts(sum([new Square(0, 0, 10, 10), new Square(5, 2, 30, 50), new Square(100, 40, 20, 19)].map(function(square) {
return getArea(square);
})));
```

As you can see, using objects in a functional language can be just fine. Some Lisps even have things called property lists which can be thought of as objects.

The real trick to using objects in a functional style is to make sure that you don't rely on their side effects but instead treat them as immutable. An easy way is whenever you want to change a property, just create a *new* object with the new details and pass that one along, instead (this is the approach often used in Clojure and Haskell).

I strongly believe that functional aspects can be very useful in JavaScript but ultimately, you should use whatever makes the code more readable and what works for you.

Solution courtesy of: Brian McKenna

Discussion

So 2 things to point out ,

1. In your first example your variable would not be leaking into the global area and is the way it should be done , try to never use variables without declaring them i.e. test = 'data' would cause data to leak into the global area.

2. Your second example is correct as well , bar1 and bar2 would only be declared on the Foo object.

Things to keep in mind try not to overuse prototyping since it applies to every object that you create , this could be extremely memory intensive depending on how complex your objects are.

If you are looking for a app development framework , have a look at ExtJs. Personally I think it would fit perfectly into the model you are trying to develop against. Just keep in mind how their licensing model works before getting heavily invested in it.

Discussion courtesy of: RC1140

I think, http://documentcloud.github.com/underscore/ should be nice fit for what you need - it provides the most important higher-order functions for functional programming and does not has client-side functions for DOM manipulation which you don't need for server side. Though I don't have experience with it.

As a side note: IMHO primary feature of functional programming is Referential transparency of a function - function result depends only on its parameters - function does not depend on changes on other objects and does not introduce any change except its result value. It makes it easy to reason about program's correctness and very valuable for implementing of predictable multi-threading (if relevant). Though JavaScript is not the bet language for FP - I expect immutable data structures to be very expensive performance-wise to use.

Discussion courtesy of: Alexey

You have to understand that functional programming and object oriented programming are somewhat antithetical to each other. It's not possible to both be purely functional and purely object oriented.

Functional programming is all about stateless computations. Object oriented programming is all about state transitions. (Paraphasing this. Hopefully not too badly)

JavaScript is more object oriented than it is functional. Which means that if you want to program in a purely functional style, you have to forego large parts of the language. Specifically all the object orient parts.

If you are willing to be more pragmatic about it, there are some inspirations from the purely functional world that you could use.

I try to adhere to the following rules:

Functions that perform computations should not alter state. And functions that alter state should not perform computations. Also, functions that alter state should alter as little state as possible. The goal is to have lots of little functions that only do one thing. Then, if you need to do anything big, you compose a bunch of little functions to do what you need.

There are a number of benefits to be gained from following these rules:

1. Ease of reuse. The longer and more complex a function is, the more specialized it also is, and therefore the less likely it is that it can be reused. The reverse implication is that shorter functions tend to more generic and therefore easier to reuse.

2. Reliability of code. It is easier to reason about correctness of the code if it is less complex.

3. It is easier to test functions when they do only one thing. That way there are fewer special cases to test.

Update:

Incorporated suggestion from comment.

Update 2:

Added some useful links.

node.js real-time game

Problem

Is it possible to create a real time game with node.js that requires twitch reflexes. How high is the latency? How low can it realistically go?

Problem courtesy of: Mark

Solution

HTTP servers are typically optimised for throughput/bandwidth over latency. node.js is unlikely to be an exception, and HTTP is intrinsically poor for low latency anyway due to the structure of the protocol.

One informal benchmark using node.js supports this, showing latencies of hundreds of milliseconds. By comparison most twitch games support latencies of no more than 30 or 40ms, ideally less.

Therefore I'd recommend dropping the twitch aspect if you can't drop HTTP.

Solution courtesy of: Kylotan

Discussion

It is possible to make a real-time game in node.js as you could with any other language/framework.

The problem here would be what kind of server and client you would use.
Using the http server feature for such game would be a bad idea and very difficult, but you could use the TCP server (now called net server) as you would in any other language.

The client would be on some platform where you can use sockets, like Flash, Java applets or desktop software.

Please notice that even with using a TCP socket server you might have problems with latency for a twitch game, but this is outside the area related to this question and more about games and networking.

PS: You *could* use web sockets since they should theoretically work like TCP sockets but there isn't yet a good support for them in the current modern browsers.

EDIT:

It seems I haven't explained myself correctly, you *can* make a browser accessible game like you said, you just need to use a protocol that allows you to **quickly send data back and forth in real time**.

If you want a *"pure"* browser game without any third party plugins the only way, like I said before, is using ***JavaScript* with *websockets*** which is not well supported yet by the major browsers. (You could use a Flash bridge and still have your game in JavaScript though.)

Using a third party plugin you have **Flash** and **Java** (besides the numerous less known plugins like unity and so on). Both have TCP sockets (not sure about UDP) and can be made to connect to a node.js net server (with some security limitations). Most people would say for you to go with Flash since there is a bigger support but Apple doesn't like it so no Flash in iPhone/iPad/iPod Touch or on other miscellaneous mobile devices (that support Java instead).

So yeah... good luck with this.

EDIT 2:

Websocket support in browsers is now pretty decent so I recommend it for realtime games if you want to use the browser as a client.

Discussion courtesy of: Diogo Gomes

It's possible, but it depends on how much data must be transmitted between server and client and how fast (speaking of latency). Take a look at Sousaball by Creationix, for example.

Also, if you plan to use websockets, take a look at Socket.IO library by learnboost. It uses websockets when available and falls back to comet in other

cases.

Discussion courtesy of: Kuroki Kaze

This content originated from StackOverFlow and has been re-organized into the above recipe.

When to use TCP and HTTP in node.js?

Problem

Stupid question, but just making sure here:

When should I use TCP over HTTP? Are there any examples where one is better than the other?

Problem courtesy of: resopollution

Solution

TCP is full-duplex 2-way communication. HTTP uses request/response model. Let's see if you are writing a chat or messaging application. TCP will work much better because you can notify the client immediately. While with HTTP, you have to do some tricks like long-polling.

However, TCP is just byte stream. You have to find another protocol over it to define your messages. You can use Google's ProtoBuffer for that.

Solution courtesy of: ZZ Coder

Discussion

Use HTTP if you need the services it provides -- e.g., message framing, caching, redirection, content metadata, partial responses, content negotiation -- as well as a large number of well-understood tools, implementations, documentation, etc.

Use TCP if you can't work within those constraints. However, if you use TCP you'll be creating a new application protocol, which has a number of pitfalls.

Discussion courtesy of: Mark Nottingham

This content originated from StackOverFlow and has been re-organized into the above recipe.

Why Won't the WebSocket.onmessage Event Fire?

Problem

After toying around with this for hours, I simply cannot find a solution. I'm working on a WebSocket server using "node.js" for a canvas based online game I'm developing. My game can connect to the server just fine, it accepts the handshake and can even send messages to the server. However, when the server responds to the client, the client doesn't get the message. No errors, nothing, it just sits there peacefully. I've ripped apart my code, trying everything I could think of to fix this, but alas, nothing.

Here's a stripped copy of my server code. As I said before, the handshake works fine, the server receives data fine, but sending data back to the client does not.

```
var sys = require('sys'),
net = require('net');
var server = net.createServer(function (stream) {
stream.setEncoding('utf8');
var shaken = 0;
stream.addListener('connect', function () {
sys.puts("New connection from: "+stream.remoteAddress);
});
stream.addListener('data', function (data) {
if (!shaken) {
sys.puts("Handshaking...");
//Send handshake:
stream.write(
"HTTP/1.1 101 Web Socket Protocol Handshake\r\n"+
"Upgrade: WebSocket\r\n"+
"Connection: Upgrade\r\n"+
"WebSocket-Origin: http://192.168.1.113\r\n"+
"WebSocket-Location: ws://192.168.1.71:7070/\r\n\r\n");
shaken=1;
sys.puts("Handshaking complete.");
}
else {
//Message received, respond with 'testMessage'
var d = "testMessage";
var m = '\u0000' + d + '\uffff';
sys.puts("Sending '"+m+"' to client");
var result = stream.write(m, "utf8");
/*
Result is equal to true, meaning that it pushed the data out.
Why isn't the client seeing it?!?
*/
}
});
stream.addListener('end', function () {
sys.puts("Connection closed!");
stream.end();
});
});
server.listen(7070);
sys.puts("Server Started!");
```

Here's my client side code. It uses Chrome native WebSockets.

```
socket = new WebSocket("ws://192.168.1.71:7070");
socket.onopen = function(evt) {
socket.send("blah");
alert("Connected!");
};
socket.onmessage = function(evt) {
alert(evt.data);
};
```

Problem courtesy of: SumWon

Solution

```
var m = '\u0000' + d + '\uffff';
var result = stream.write(m, "utf8");
```

Ah, that seems not quite right. It should be a zero byte, followed by a UTF-8 encoded string, followed by a 0xFF byte.

\uFFFF does not encode in UTF-8 to a 0xFF byte: you get 0xC3 0xBF. In fact no character will ever produce an 0xFF in UTF-8 encoding: that's why it was picked as a terminator in WebSocket.

I'm not familiar with node.js, but I would guess you'd need to say something like:

```
stream.write('\u0000'+d, 'utf-8');
stream.write('\xFF', 'binary');
```

Or manually UTF-8 encode and send as binary.

The handshaking is also questionable, though I guess it's a simple-as-possible first draft that's probably working for you at the moment. You're relying on the first packet being the whole handshake header and nothing else, then the second packet being the payload data. This isn't at all guaranteed. You'll probably want to collect data in a buffer until you see the 0x0D 0x0A 0x0D 0x0A marker for the end of the headers, then check the header content and possibly return the 101, then continue from data after that marker.

Solution courtesy of: bobince

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

What does addListener do in node.js?

Problem

I am trying to understand the purpose of addListener in node.js. Can someone explain please? Thanks! A simple example would be:

```
var tcp = require('tcp');
var server = tcp.createServer(function (socket) {
socket.setEncoding("utf8");
socket.addListener("connect", function () {
socket.write("hello\r\n");
});
socket.addListener("data", function (data) {
socket.write(data);
});
socket.addListener("end", function () {
socket.write("goodbye\r\n");
socket.end();
});
});
server.listen(7000, "localhost");
```

Problem courtesy of: Jeff

Solution

Due to the fact that Node.js works event-driven and executes an event-loop, registering listeners allow you to define callbacks that will be executed every time the event is fired. Thus, it is also a form of async. code structuring.

It's comparable to GUI listener, that fire on user interaction. Like a mouse click, that triggers an execution of code in your GUI app, your listeners in your example will be run as soon as the event happens, i.e. a new client connects to the socket.

Solution courtesy of: b_erb

Discussion

it registers a listener for an "event". Events are identified by strings, such as "connect" and "data". the second argument is a function, a so called "callback", also refered to as "event handler". Whenever a specific event occurs within the object the listeners have been registered to, all handlers are invoked.

node.js uses this, because it employs an asynchronous execution model, that can best be handled with an event-driven approach.

greetz
back2dos

Discussion courtesy of: back2dos

Recommendation for integrating nodejs with php application

Problem

I have an existing app written in PHP (using Kohana framework) and I want to do long polling. From some things I read it seems that doing long polling with PHP is not advisable and using something like nodejs is a better choice. My question is what's the best way to integrate nodejs (or some other well suited tool for long polling) with an existing application?

For clarification my app basically is a browser plugin that you can use to send data to groups of other people. When that data is sent, I want the recipients, if they are online and also have the browser plugin, to instantly receive that data and be notified.

Problem courtesy of: jhchen

Solution

Possibly the best way is to let node.js listen to a port and to let PHP send messages to that port.

In Node.js you can just open a socket for listening and in PHP you can use cURL to send messages. The messages can be in JSON-format.

If the Node.js-part receives a message, it can forward it, possibly after some processing, directly to the long-polling browser.

Solution courtesy of: edwin

Discussion

I am creating a small hack that would allow you to do this with ease. It is in a very early stage but it has enough code for it to work: https://github.com/josebalius/NodePHP

I plan on updating the readme later today.

Discussion courtesy of: Jose

Is it possible in .NET, using C#, to achieve event based asynchronous pattern without multithreading?

Problem

I am amazed by the architectural design of Node.js and was wondering if C# is capable of such a design:

Asynchronous, event based / event loop, non-blocking I/O without multithreading.

Problem courtesy of: Jeff

Solution

I think that all the BeginXyz operations that implement the standard asynchronous programming model run the callback on a thread pool thread, which makes the application automatically multi-threaded.

However, you can achieve single-threaded asynchronous programming model by synchronizing all the operations through the single GUI thread that is maintained for windows applications using Control.Invoke or more generally, SynchronizationContext.

Each call to BeginXyz would have to be rewritten along these lines:

```
// Start asynchronous operation here (1)
var originalContext = SynchronizationContext.Current;
obj.BeginFoo(ar =>
// Switch to the original thread
originalContext.Post(ignored => {
var res = obj.EndFoo();
// Continue here (2)
}));
```

The code marked as (2) will continue running on the same thread as the code in (1), so you'll use the thread-pool thread only for forwarding the postback back to the original (single) thread.

As a side-note, this is more directly supported by asynchronous workflows in F# and it can be used for quite elegant style of GUI programming as described here. I don't know node.js, but I suppose that you may be also amazed by F# *asynchronous workflows* as they are really cool for asynchronous/event based/... style of programming :-)

Solution courtesy of: Tomas Petricek

Discussion

Not sure if this is what you're looking for. .NET can do asynchronous callbacks without explicit multi threading.

Discussion courtesy of: Jeremy

your example of node.js is not really applicable since the server that it's running on is performing all the necessary multithreading. If the events are executed based on the same external clock signal, then they're not asynchronous. You can get around this by running another application and that would create two processes on the system.

There is no way that you can make the same application running as a single process on a single system perform asynchronous events without having another thread.

For more details see this Asynchronous vs Multithreading question.

Discussion courtesy of: Lirik

The Reactive Extensions for .NET (Rx) is designed for asynchronous and parallel programming. It allows you to program in a reactive vs. interactive way, non-blocking. You use the LINQ query operators, and new ones for IObservable/IObserver interfaces, which are part of Rx. Rx provides the mathematical dual of IEnumerable/IEnumerator, in the form of IObservable/IObserver, which means you can use all of the LINQ standard query operators, in a declarative way, as opposed to using the multithreading APIs directly.

Discussion courtesy of: Richard Hein

Sure, it just requires an event loop. Something like:

```
class EventLoop {
List<Action> MyThingsToDo { get; set; }

public void WillYouDo(Action thing) {
this.MyThingsToDo.Add(thing);
}

public void Start(Action yourThing) {
while (true) {
Do(yourThing);

foreach (var myThing in this.MyThingsToDo) {
Do(myThing);
}
this.MyThingsToDo.Clear();
}
}

void Do(Action thing) {
thing();
```

```
}
}

class Program {
static readonly EventLoop e = new EventLoop();

static void Main() {
e.Start(DoSomething);
}

static int i = 0;
static void DoSomething() {
Console.WriteLine("Doing something...");
e.WillYouDo(() => {
results += (i++).ToString();
});
Console.WriteLine(results);
}

static string results = "!";
}
```

Pretty soon, you'll want to get rid of DoSomething and require all work to be registered with MyThingsToDo. Then, you'll want to pass an enum or something to each ThingToDo that tells it why it's doing something. At that point, you'll realize you have a message pump.

BTW, I'd say node.js is glossing over the fact that it's running on an OS and application that's multithreaded. Without that, each call to the network or disk would block.

Discussion courtesy of: Mark Brackett

You can use the WPF dispatcher, even if you aren't doing UI. Reference the assembly WindowsBase. Then, in your startup code execute:

```
Dispatcher.CurrentDispatcher.BeginInvoke(new Action(Initialize));
Dispatcher.Run();
```

From Initialize and elsewhere in your code, use Dispatcher.CurrentDispatcher.BeginInvoke to schedule execution of subsequent async methods.

Discussion courtesy of: Edward Brey

I'm working on just such a thing in .NET as a pet project. I call it ALE (Another Looping Event)... because beer.

It's *extremely* alpha right now, but it's all homebrew because, like you, I wanted to know if it can be done.

- It's an event loop architecture.
- It leverages .NET's non-blocking async I/O calls
- It uses callback-style calls to assist developers writing async code that is a little more readable.

- Async web sockets implementation
- Async http server implementation
- Async sql client implemenation

Unlike some of the other tries at this I've seen it's not just a web server with an event loop. You could write any sort of event loop based application off of this.

Example

The following code will:

- Start the Event Loop
- Start a web server on port 1337
- Start a web socket server on port 1338
- Then read a file and "DoSomething" with it:

```
EventLoop.Start(() => {

//create a web server
Server.Create((req, res) => {
res.Write("<h1>Hello World</h1>");
}).Listen("http://*:1337");


//start a web socket server
Net.CreateServer((socket) => {
socket.Receive((text) => {
socket.Send("Echo: " + text);
});
}).Listen("127.0.0.1", 1338, "http://origin.com");


//Read a file
File.ReadAllText(@"C:\Foo.txt", (text) => {
DoSomething(text);
});
});
```

So I guess my answer is "Yes", it can be done in C#... or almost any language for that matter. The real trick is being able to leverage native non-blocking I/O.

More information about the project will be posted here.

Discussion courtesy of: Ben Lesh

i developed a server based on HttpListener and an event loop, supporting MVC, WebApi and routing. For what i have seen the performances are far better than standard IIS+MVC, for the MVCMusicStore i moved from 100 requests per seconds and 100% CPU to 350 with 30% CPU. If anybody would give it a try i am struggling for feedbacks!

PS any suggestion or correction is welcome!

- Documentation
- MvcMusicStore sample port on Node.Cs
- Packages on Nuget

Discussion courtesy of: EDR

I believe it's possible, here is an open-source example written in VB.NET and C#:

https://github.com/perrybutler/dotnetsockets/

It uses Event-based Asynchronous Pattern (EAP), IAsyncResult Pattern and thread pool (IOCP). It will serialize/marshal the messages (messages can be any native object such as a class instance) into binary packets, transfer the packets over TCP, and then deserialize/unmarshal the packets at the receiving end so you get your native object to work with. This part is somewhat like Protobuf or RPC.

It was originally developed as a "netcode" for real-time multiplayer gaming, but it can serve many purposes. Unfortunately I never got around to using it. Maybe someone else will.

The source code has a lot of comments so it should be easy to follow. Enjoy!

EDIT: After stress tests and a few optimizations it was able to accept 16,357 clients before reaching OS limits. Here are the results:

```
Simulating 1000 client connections over 16 iterations...
1) 485.0278 ms
2) 452.0259 ms
3) 495.0283 ms
4) 476.0272 ms
5) 472.027 ms
6) 477.0273 ms
7) 522.0299 ms
8) 516.0295 ms
9) 457.0261 ms
10) 506.029 ms
11) 474.0271 ms
12) 496.0283 ms
13) 545.0312 ms
14) 516.0295 ms
15) 517.0296 ms
16) 540.0309 ms
All iterations complete. Total duration: 7949.4547 ms
```

Now that's with all clients running on localhost, and sending a small message to the server just after connecting. **In a subsequent test on a different system, the server was maxing out at just over 64,000 client connections (port limit reached!) at about 2000 per second, consuming 238 MB of RAM.**

Discussion courtesy of: perry

Here is my example of single-threaded EAP. There are several implementations of EAP in different application types: from simple console app to asp.net app, tcp server and more. They are all build on the tiny framework called SingleSand

which theoretically is pluggable to any type of .net application.

In contrast to the previous answers, my implementation acts as a mediator between existing technologies (asp.net, tcp sockets, RabbitMQ) from one side and tasks inside the event loop from other side. So the goal is not to make a pure single-threaded application but to integrate the event loop to existing application technologies. I fully agree with previous post that .net does not support pure single-threaded apps.

Discussion courtesy of: neleus

Node.js and wss://

Problem

I'm looking to start using javascript on the server, most likely with node.js, as well as use websockets to communicate with clients. However, there doesn't seem to be a lot of information about encrypted websocket communication using TLS and the wss:// handler. In fact the only server that I've seen explicitly support wss:// is Kaazing.

This TODO is the only reference I've been able to find in the various node implementations. Am I missing something or are the websocket js servers not ready for encrypted communication yet?

Another option could be using something like lighttpd or apache to proxy to a node listener, has anyone had success there?

Problem courtesy of: case nelson

Solution

TLS/SSL support works for this websocket implementation in Node.js, I just tested it: https://github.com/Worlize/WebSocket-Node/issues/29

Solution courtesy of: pors

Discussion

Well you have stream.setSecure() and server.setSecure().

I'm guessing you should be able to use one of those (specially the last one) to use TLS in websockets since in the end a websocket is just a normal http connection "upgraded" to websocket.

Using TLS in the normal http server object should theorically also secure the websocket, only by testing this can be confirmed.

Discussion courtesy of: Diogo Gomes

How to create a streaming API with NodeJS

Problem

How would you go to create a streaming API with Node? just like the Twitter streaming API.

What I want to do ultimately is get the first update from the FriendFeed api, and stream when a new one becomes available (if the id is different), and later on expose it as a web service so I can use it with WebSockets on my website :).

So far I have this:

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

var ff = http.createClient(80, 'friendfeed-api.com');
var request = ff.request('GET', '/v2/feed/igorgue?num=1',
{'host': 'friendfeed-api.com'});

request.addListener('response', function (response) {
response.setEncoding('utf8'); // this is *very* important!
response.addListener('data', function (chunk) {
var data = JSON.parse(chunk);
sys.puts(data.entries[0].body);
});
});
request.end();
```

Which only gets the data from FriendFeed, creating the Http server with node is easy but it can't return a stream (or I haven't yet found out how).

Problem courtesy of: igorgue

Solution

You would want to set up a system that keeps track of incoming requests and stores their response objects. Then when it's time to stream a new event from FriendFeed, iterate through their response objects and responses[i].write('something') out to them.

Check out LearnBoost's Socket.IO-Node, you may even just be able to use that project as your framework and not have to code it yourself.

From the Socket.IO-Node example app (for chat):

```
io.listen(server, {

onClientConnect: function(client){
client.send(json({ buffer: buffer }));
client.broadcast(json({ announcement: client.sessionId + ' connected' }));
},

onClientDisconnect: function(client){
client.broadcast(json({ announcement: client.sessionId + ' disconnected' }));
},

onClientMessage: function(message, client){
var msg = { message: [client.sessionId, message] };
buffer.push(msg);
if (buffer.length > 15) buffer.shift();
client.broadcast(json(msg));
}

});
```

Solution courtesy of: JasonWyatt

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

How do I include a JavaScript file in another JavaScript file?

Problem

Is there something in JavaScript similar to @import in CSS that allows you to include a JavaScript file inside another JavaScript file?

Problem courtesy of: Alec Smart

Solution

The old versions of JavaScript had no import, include, or require, so many different approaches to this problem have been developed.

But recent versions of JavaScript have standards like ES6 modules to import modules, although this is not supported yet by most browsers. Many people using modules with browser applications use build and/or transpilation tools to make it practical to use new syntax with features like modules.

ES6 Imports

Note that currently (unless you are using the latest MS Edge) this will require the use of build and/or transpilation tools.

```
// module.js
export function hello() {
return "Hello";
}

// main.js
import {hello} from 'module'; // or './module'
let val = hello(); // val is "Hello";
```

Node.js require

Node.js is currently using a module.exports/require system. You can use babel to transpile if you want the import syntax.

```
// mymodule.js
exports.hello = function() {
return "Hello";
}

// server.js
const myModule = require('./mymodule');
let val = myModule.hello(); // val is "Hello"
```

There are other ways for JavaScript to include external JavaScript contents in browsers that do not require preprocessing.

Ajax Loading

Load an additional script with an Ajax call and then use eval. This is the most straightforward way, but it is limited to your domain because of the JavaScript sandbox security model. Using eval also opens the door to bugs and hacks.

jQuery Loading

The jQuery library provides loading functionality in one line:

```
$.getScript("my_lovely_script.js", function(){

alert("Script loaded but not necessarily executed.");
```

```
});
```

Dynamic Script Loading

Add a script tag with the script URL in the HTML. To avoid the overhead of jQuery, this is an ideal solution.

The script can even reside on a different server. Furthermore, the browser evaluates the code. The <script> tag can be injected into either the web page <head>, or inserted just before the closing </body> tag.

Both of these solutions are discussed and illustrated in *JavaScript Madness: Dynamic Script Loading*.

Detecting when the script has been executed

Now, there is a big issue you must know about. Doing that implies that *you remotely load the code*. Modern web browsers will load the file and keep executing your current script because they load everything asynchronously to improve performance. (This applies to both the jQuery method and the manual dynamic script loading method.)

It means that if you use these tricks directly, *you won't be able to use your newly loaded code the next line after you asked it to be loaded*, because it will be still loading.

For example: my_lovely_script.js contains MySuperObject:

```
var js = document.createElement("script");

js.type = "text/javascript";
js.src = jsFilePath;

document.body.appendChild(js);

var s = new MySuperObject();

Error : MySuperObject is undefined
```

Then you reload the page hitting F5. And it works! Confusing...

So what to do about it ?

Well, you can use the hack the author suggests in the link I gave you. In summary, for people in a hurry, he uses an event to run a callback function when the script is loaded. So you can put all the code using the remote library in the callback function. For example:

```
function loadScript(url, callback)
{
// Adding the script tag to the head as suggested before
var head = document.getElementsByTagName('head')[0];
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = url;

// Then bind the event to the callback function.
// There are several events for cross browser compatibility.
script.onreadystatechange = callback;
script.onload = callback;

// Fire the loading
head.appendChild(script);
}
```

Then you write the code you want to use AFTER the script is loaded in a lambda function:

```
var myPrettyCode = function() {
```

```
// Here, do what ever you want
};

Then you run all that:

loadScript("my_lovely_script.js", myPrettyCode);
```

Note that the script may execute after the DOM has loaded, or before, depending on the browser and whether you included the line script.async = false;. There's a great article on Javascript loading in general which discusses this.

Source Code Merge/Preprocessing

As mentioned at the top of this answer, many developers now use build/transpilation tool(s) like WebPack, Babel, or Gulp in their projects, allowing them to use new syntax and support modules better, combine files, minify, etc.

Solution courtesy of: e-satis

Discussion

It is possible to dynamically generate a JavaScript tag and append it to HTML document from inside other JavaScript code. This will load targeted JavaScript file.

```
function includeJs(jsFilePath) {
var js = document.createElement("script");

js.type = "text/javascript";
js.src = jsFilePath;

document.body.appendChild(js);
}

includeJs("/path/to/some/file.js");
```

Discussion courtesy of: Svitlana Maksymchuk

Maybe you can use this function that I found on this page *How do I include a JavaScript file in a JavaScript file?*:

```
function include(filename)
{
var head = document.getElementsByTagName('head')[0];

var script = document.createElement('script');
script.src = filename;
script.type = 'text/javascript';

head.appendChild(script)
}
```

Discussion courtesy of: Arnaud Gouder de Beauregard

I just wrote this JavaScript code (using Prototype for DOM manipulation):

```
var require = (function() {
var _required = {};
return (function(url, callback) {
if (typeof url == 'object') {
// We've (hopefully) got an array: time to chain!
if (url.length > 1) {
// Load the nth file as soon as everything up to the
// n-1th one is done.
require(url.slice(0, url.length - 1), function() {
require(url[url.length - 1], callback);
});
} else if (url.length == 1) {
require(url[0], callback);
}
return;
}
if (typeof _required[url] == 'undefined') {
// Haven't loaded this URL yet; gogogo!
```

```
_required[url] = [];

var script = new Element('script', {
src: url,
type: 'text/javascript'
});
script.observe('load', function() {
console.log("script " + url + " loaded.");
_required[url].each(function(cb) {
cb.call(); // TODO: does this execute in the right context?
});
_required[url] = true;
});

$$('head')[0].insert(script);
} else if (typeof _required[url] == 'boolean') {
// We already loaded the thing, so go ahead.
if (callback) {
callback.call();
}
return;
}

if (callback) {
_required[url].push(callback);
}
});
})();
```

Usage:

```
<script src="prototype.js"></script>
<script src="require.js"></script>
<script>
require(['foo.js','bar.js'], function () {
/* Use foo.js and bar.js here */
});
</script>
```

Gist: http://gist.github.com/284442.

Discussion courtesy of: nornagon

You can also assemble your scripts using PHP:

File main.js.php:

```
<?php
header('Content-type:text/javascript; charset=utf-8');
include_once("foo.js.php");
include_once("bar.js.php");
?>

// Main JavaScript code goes here
```

Discussion courtesy of: Calmarius

There actually *is* a way to load a JavaScript file *not* asynchronously, so you

could use the functions included in your newly loaded file right after loading it, and I think it works in all browsers.

You need to use jQuery.append() on the <head> element of your page, that is:

```
$("head").append('<script type="text/javascript" src="' + script + '">
</script>');
```

However, this method also has a problem: if an error happens in the imported JavaScript file, Firebug (and also Firefox Error Console and Chrome Developer Tools as well) will report its place incorrectly, which is a big problem if you use Firebug to track JavaScript errors down a lot (I do). Firebug simply doesn't know about the newly loaded file for some reason, so if an error occurs in that file, it reports that it occurred in your main HTML file, and you will have trouble finding out the real reason for the error.

But if that is not a problem for you, then this method should work.

I have actually written a jQuery plugin called *$.import_js()* which uses this method:

```
(function($)
{
/*
* $.import_js() helper (for JavaScript importing within JavaScript code).
*/
var import_js_imported = [];

$.extend(true,
{
import_js : function(script)
{
var found = false;
for (var i = 0; i < import_js_imported.length; i++)
if (import_js_imported[i] == script) {
found = true;
break;
}

if (found == false) {
$("head").append('<script type="text/javascript" src="' + script + '">
</script>');
import_js_imported.push(script);
}
}
});

})(jQuery);
```

So all you would need to do to import JavaScript is:

```
$.import_js('/path_to_project/scripts/somefunctions.js');
```

I also made a simple test for this at http://www.kipras.com/dev/import_js_test/.

It includes a main.js file in the main HTML and then the script in main.js uses $.import_js() to import an additional file called included.js, which defines this function:

```
function hello()
{
alert("Hello world!");
}
```

And right after including included.js, the hello() function is called, and you get the alert.

(This answer is in response to e-satis' comment).

Discussion courtesy of: Kipras

Another way, that in my opinion is much cleaner, is to make a synchronous Ajax request instead of using a <script> tag. Which is also how Node.js handles includes.

Here's an example using jQuery:

```
function require(script) {
$.ajax({
url: script,
dataType: "script",
async: false, // <-- This is the key
success: function () {
// all good...
},
error: function () {
throw new Error("Could not load script " + script);
}
});
}
```

You can then use it in your code as you'd usually use an include:

```
require("/scripts/subscript.js");
```

And be able to call a function from the required script in the next line:

```
subscript.doSomethingCool();
```

Discussion courtesy of: Ariel

I came to this question because I was looking for a simple way to maintain a collection of useful JavaScript plugins. After seeing some of the solutions here, I came up with this:

1) Set up a file called "plugins.js" (or extentions.js or what have you). Keep your plugin files together with that one master file.

2) plugins.js will have an array called "pluginNames[]" that we will iterate over each(), then append a tag to the head for each plugin

```
//set array to be updated when we add or remove plugin files
var pluginNames = ["lettering", "fittext", "butterjam", etc.];
//one script tag for each plugin
$.each(pluginNames, function(){
$('head').append('<script src="js/plugins/' + this + '.js"></script>');
```

```
});
```

3) manually call just the one file in your head:
```
<script src="js/plugins/plugins.js"></script>
```

UPDATE: I found that even though all of the plugins were getting dropped into the head tag the way they ought to, they weren't always being run by the browser when you click into the page or refresh.

I found it's more reliable to just write the script tags in a PHP include. You only have to write it once and that's just as much work as calling the plugin using JavaScript.

Discussion courtesy of: rgb_life

I have created a function that will allow you to use similar verbiage to C#/Java to include a JavaScript file. I've tested it a little bit even from inside of *another* JavaScript file and it seems to work. It does require jQuery though for a bit of "magic" at the end.

I put this code in a file at the root of my script directory (I named it global.js, but you can use whatever you want. Unless I'm mistaken this and jQuery should be the only required scripts on a given page. Keep in mind this is largely untested beyond some basic usage, so there may or may not be any issues with the way I've done it; use at your own risk yadda yadda I am not responsible if you screw anything up yadda yadda:

```
/**
* @fileoverview This file stores global functions that are required by other libraries.
*/

if (typeof(jQuery) === 'undefined') {
throw 'jQuery is required.';
}

/** Defines the base script directory that all .js files are assumed to be organized under. */
var BASE_DIR = 'js/';

/**
* Loads the specified file, outputting it to the <head> HTMLElement.
*
* This method mimics the use of using in C# or import in Java, allowing
* JavaScript files to "load" other JavaScript files that they depend on
* using a familiar syntax.
*
* This method assumes all scripts are under a directory at the root and will
* append the .js file extension automatically.
*
* @param {string} file A file path to load using C#/Java "dot" syntax.
*
* Example Usage:
* imports('core.utils.extensions');
*       This       will       output:       <script       type="text/javascript" src="/js/core/utils/extensions.js"></script>
*/
function imports(file) {
var fileName = file.substr(file.lastIndexOf('.') + 1, file.length);
```

```
// Convert PascalCase name to underscore_separated_name
var regex = new RegExp(/([A-Z])/g);
if (regex.test(fileName)) {
var separated = fileName.replace(regex, ",$1").replace(',', '');
fileName = separated.replace(/[,]/g, '_');
}

// Remove the original JavaScript file name to replace with underscore version
file = file.substr(0, file.lastIndexOf('.'));

// Convert the dot syntax to directory syntax to actually load the file
if (file.indexOf('.') > 0) {
file = file.replace(/[.]/g, '/');
}

var src = BASE_DIR + file + '/' + fileName.toLowerCase() + '.js';
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = src;

$('head').find('script:last').append(script);
}
```

Discussion courtesy of: Wayne Molina

In a past project I had quite a bit of success using ajile to do imports of reusable JavaScript files. I always wished there was a feature for this built into JavaScript itself.

Discussion courtesy of: jpierson

This is perhaps the biggest weakness of JavaScript in my opinion. It's caused me no end of problems over the years with dependency tracing. Anyhow, it does appear that the only *practical* solution is to use script includes in the HTML file and thus horribly making your JavaScript code dependent upon the user including the source you need and making reuse unfriendly.

Sorry if this comes across as a lecture ;) It's a (bad) habit of mine, but I want to make this point.

The problem comes back to the same as everything else with the web, the history of JavaScript. It really wasn't designed to be used in the widespread manner it's used in today. Netscape made a language that would allow you to control a few things, but they didn't envisage its widespread use for so many things as it is put to now and for one reason or another it's expanded from there, without addressing some of the fundamental weaknesses of the original strategy.

It's not alone of course. HTML wasn't designed for the modern webpage; it was designed to be a way of expressing the logic of a document, so that readers (browsers in the modern world) could display this in an applicable form that was within the capabilities of the system, and it took years for a solution (other than the hacks of MS and Netscape) to come along. CSS solves this problem, but it was a long time coming and even longer to get people to use it rather than the established BAD techniques. It happened though, praise be.

Hopefully JavaScript (especially now it's part of the standard) will develop to take on board the concept of proper modularity (as well as some other things)

as every other (extant) programming language in the world does and this stupidity will go away. Until then you just have to not like it and lump it, I'm afraid.

Discussion courtesy of: Stephen Whipp

Or rather than including at run time, use a script to concatenate prior to upload.

I use Sprockets (I don't know if there are others). You build your JavaScript code in separate files and include comments that are processed by the Sprockets engine as includes. For development you can include files sequentially, then for production to merge them...

See also:

- *Introducing Sprockets: JavaScript dependency management and concatenation*

Discussion courtesy of: JMawer

If anyone is looking for something more advanced, try out RequireJS. You'll get added benefits such as dependency management, better concurrency, and avoid duplication (that is, retrieving a script more than once).

You can write your JavaScript files in "modules" and then reference them as dependencies in other scripts. Or you can use RequireJS as a simple "go get this script" solution.

Example:

Define dependencies as modules:

some-dependency.js

```
define(['lib/dependency1', 'lib/dependency2'], function (d1, d2) {

//Your actual script goes here.
//The dependent scripts will be fetched if necessary.

return libraryObject; //For example, jQuery object
});
```

implementation.js is your "main" JavaScript file that depends on **some-dependency.js**

```
require(['some-dependency'], function(dependency) {

//Your script goes here
//some-dependency.js is fetched.
//Then your script is executed
});
```

Excerpt from the GitHub README:

RequireJS loads plain JavaScript files as well as more defined modules. It is optimized for in-browser use, including in a Web Worker, but it can be used in

other JavaScript environments, like Rhino and Node. It implements the Asynchronous Module API.

RequireJS uses plain script tags to load modules/files, so it should allow for easy debugging. It can be used simply to load existing JavaScript files, so **you can add it to your existing project without having to re-write your JavaScript files.**

...

Discussion courtesy of: John Strickler

There is a good news for you. Very soon you will be able to load JavaScript code easily. It will become a standard way of importing modules of JavaScript code and will be part of core JavaScript itself.

You simply have to write import cond from 'cond.js'; to load a macro named cond from a file cond.js.

So you don't have to rely upon any JavaScript framework nor do you have to explicitly make Ajax calls.

Refer to:

- *Static module resolution*
- *Module loaders*

Discussion courtesy of: Imdad

```
var js = document.createElement("script");

js.type = "text/javascript";
js.src = jsFilePath;

document.body.appendChild(js);
```

Discussion courtesy of: Sam4Code

Better use the jQuery way. To delay the ready event, first call $.holdReady(true). Example (source):

```
$.holdReady(true);
$.getScript("myplugin.js", function() {
$.holdReady(false);
});
```

Discussion courtesy of: weageoo

Don't forget to check out LAB.js!

```
<script type="text/javascript">
$LAB
.script("jquery-1.8.3.js").wait()
```

```
.script("scripts/clientscript.js");
</script>
```

Discussion courtesy of: emolaus

Now, I may be totally misguided, but here's what I've recently started doing... Start and end your JavaScript files with a carriage return, place in the PHP script, followed by one more carriage return. The JavaScript comment "//" is ignored by PHP so the inclusion happens anyway. The purpose for the carriage returns is so that the first line of your included JavaScript isn't commented out.

Technically, you don't need the comment, but it posts errors in Dreamweaver that annoy me. If you're scripting in an IDE that doesn't post errors, you shouldn't need the comment or the carriage returns.

```
\n
//<?php require_once("path/to/javascript/dependency.js"); ?>

function myFunction(){
// stuff
}
\n
```

Discussion courtesy of: Duncan

```
var s=["Hscript.js","checkRobert.js","Hscript.js"];
for(i=0;i<s.length;i++){
var script=document.createElement("script");
script.type="text/javascript";
script.src=s[i];
document.getElementsByTagName("head")[0].appendChild(script)
};
```

Discussion courtesy of: robert

This should do:

```
xhr = new XMLHttpRequest();
xhr.open("GET", "/soap/ajax/11.0/connection.js", false);
xhr.send();
eval(xhr.responseText);
```

Discussion courtesy of: tggagne

I wrote a simple module that automates the job of importing/including module scripts in JavaScript. For detailed explanation of the code, refer to the blog post *JavaScript require / import / include modules*.

```
// ----- USAGE -----

require('ivar.util.string');
require('ivar.net.*');
require('ivar/util/array.js');
require('http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js');
```

```
ready(function(){
//Do something when required scripts are loaded
});

//--------------------

var _rmod = _rmod || {}; //Require module namespace
_rmod.LOADED = false;
_rmod.on_ready_fn_stack = [];
_rmod.libpath = '';
_rmod.imported = {};
_rmod.loading = {
scripts: {},
length: 0
};

_rmod.findScriptPath = function(script_name) {
var script_elems = document.getElementsByTagName('script');
for (var i = 0; i < script_elems.length; i++) {
if (script_elems[i].src.endsWith(script_name)) {
var href = window.location.href;
href = href.substring(0, href.lastIndexOf('/'));
var   url   =   script_elems[i].src.substring(0,   script_elems[i].length   -
script_name.length);
return url.substring(href.length+1, url.length);
}
}
return '';
};

_rmod.libpath = _rmod.findScriptPath('script.js'); //Path of your main script
used to mark
//the root directory of your library, any library.


_rmod.injectScript = function(script_name, uri, callback, prepare) {

if(!prepare)
prepare(script_name, uri);

var script_elem = document.createElement('script');
script_elem.type = 'text/javascript';
script_elem.title = script_name;
script_elem.src = uri;
script_elem.async = true;
script_elem.defer = false;

if(!callback)
script_elem.onload = function() {
callback(script_name, uri);
};
document.getElementsByTagName('head')[0].appendChild(script_elem);
};

_rmod.requirePrepare = function(script_name, uri) {
_rmod.loading.scripts[script_name] = uri;
_rmod.loading.length++;
};

_rmod.requireCallback = function(script_name, uri) {
```

```
_rmod.loading.length--;
delete _rmod.loading.scripts[script_name];
_rmod.imported[script_name] = uri;

if(_rmod.loading.length == 0)
_rmod.onReady();
};

_rmod.onReady = function() {
if (!_rmod.LOADED) {
for (var i = 0; i < _rmod.on_ready_fn_stack.length; i++){
_rmod.on_ready_fn_stack[i]();
});
_rmod.LOADED = true;
}
};

_.rmod = namespaceToUri = function(script_name, url) {
var np = script_name.split('.');
if (np.getLast() === '*') {
np.pop();
np.push('_all');
}

if(!url)
url = '';

script_name = np.join('.');
return url + np.join('/')+'.js';
};

//You can rename based on your liking. I chose require, but it
//can be called include or anything else that is easy for you
//to remember or write, except "import", because it is reserved
//for future use.
var require = function(script_name) {
var uri = '';
if (script_name.indexOf('/') > -1) {
uri = script_name;
var lastSlash = uri.lastIndexOf('/');
script_name = uri.substring(lastSlash+1, uri.length);
}
else {
uri = _rmod.namespaceToUri(script_name, ivar._private.libpath);
}

if (!_rmod.loading.scripts.hasOwnProperty(script_name)
&& !_rmod.imported.hasOwnProperty(script_name)) {
_rmod.injectScript(script_name, uri,
_rmod.requireCallback,
_rmod.requirePrepare);
}
};

var ready = function(fn) {
_rmod.on_ready_fn_stack.push(fn);
};
```

I did:

```
var require = function (src, cb) {
cb = cb || function () {};

var newScriptTag = document.createElement('script'),
firstScriptTag = document.getElementsByTagName('script')[0];
newScriptTag.src = src;
newScriptTag.async = true;
console.log(newScriptTag)
newScriptTag.onload = newScriptTag.onreadystatechange = function () {
(!this.readyState || this.readyState === 'loaded' || this.readyState === 'complete') && (cb());
};
firstScriptTag.parentNode.insertBefore(newScriptTag, firstScriptTag);
}
```

It works great and uses no page-reloads for me. I tried that Ajax thing, but it doesn't really work.

Discussion courtesy of: Christopher Dumas

Most of solutions shown here imply dynamical loading. I was searching instead for a compiler which assemble all the depended files into a single output file. The same as Less/Sass preprocessors deal with the CSS @import at-rule. Since I didn't find anything decent of this sort, I wrote a simple tool solving the issue.

So here is the compiler, https://github.com/dsheiko/jsic, which replaces $import("file-path") with the requested file content securely. Here is the corresponding Grunt plugin: https://github.com/dsheiko/grunt-jsic.

On the jQuery master branch, they simply concatenate atomic source files into a single one starting with intro.js and ending with outtro.js. That doesn't suits me as it provides no flexibility on the source code design. Check out how it works with jsic:

src/main.js

```
var foo = $import("./Form/Input/Tel");
```

src/Form/Input/Tel.js

```
function() {
return {
prop: "",
method: function(){}
}
}
```

Now we can run the compiler:

```
node jsic.js src/main.js build/mail.js
```

And get the combined file

build/main.js

```
var foo = function() {
return {
prop: "",
method: function(){}
}
};
```

Discussion courtesy of: Dmitry Sheiko

```
function include(js)
{
document.writeln("<script src=" + js + "><" + "/script>");
}
```

Discussion courtesy of: Amir Saniyan

This script will add a JavaScript file to the top of any other <script> tag:

```
(function () {
var li = document.createElement('script');
li.type = 'text/javascript';
li.src= "http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js";
li.async=true;
var s = document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(li, s);
})();
```

Discussion courtesy of: Vicky Gonsalves

I also wrote a JavaScript dependency manager for Java web applications: JS-Class-Loader.

Discussion courtesy of: Damon Smith

Here is a Grunt plugin allowing you to use @import "path/to/file.js"; syntax in any file including JavaScript files. It can be paired with uglify or watch or any other plugin.

It can be installed with npm install: https://npmjs.org/package/grunt-import

Discussion courtesy of: Marcin

If you want in pure JavaScript, you can use document.write.

```
document.write('<script src="myscript.js" type="text/javascript"></script>');
```

If you use the jQuery library, you can use the $.getScript method.

```
$.getScript("another_script.js");
```

Discussion courtesy of: Venu immadi

There is also Head.js. It is very easy to deal with:

```
head.load("js/jquery.min.js",
"js/jquery.someplugin.js",
"js/jquery.someplugin.css", function() {
alert("Everything is ok!");
});
```

As you see, it's easier than Require.js and as convenient as jQuery's $.getScript method. It also has some advanced features, like conditional loading, feature detection and much more.

Discussion courtesy of: Ale

Here is a **synchronous** version **without jQuery**:

```
function myRequire( url ) {
var ajax = new XMLHttpRequest();
ajax.open( 'GET', url, false ); // <-- the 'false' makes it synchronous
ajax.onreadystatechange = function () {
var script = ajax.response || ajax.responseText;
if (ajax.readyState === 4) {
switch( ajax.status) {
case 200:
eval.apply( window, [script] );
console.log("script loaded: ", url);
break;
default:
console.log("ERROR: script not loaded: ", url);
}
}
};
ajax.send(null);
}
```

Note that to get this working cross-domain, the server will need to set allow-origin header in its response.

Discussion courtesy of: heinob

The @import syntax for achieving CSS-like JavaScript importing is possible using a tool such as Mixture via their special .mix file type (see here). I imagine the application simply uses one of the aforementioned methods "under the hood," though I don't know.

From the Mixture documentation on .mix files:

Mix files are simply .js or .css files with .mix. in the file name. A mix file simply extends the functionality of a normal style or script file and allows you to import and combine.

Here's an example .mix file that combines multiple .js files into one:

```
// scripts-global.mix.js
// Plugins - Global

@import "global-plugins/headroom.js";
@import "global-plugins/retina-1.1.0.js";
@import "global-plugins/isotope.js";
@import "global-plugins/jquery.fitvids.js";
```

Mixture outputs this as scripts-global.js and also as a minified version (scripts-global.min.js).

Note: I'm not in any way affiliated with Mixture, other than using it as a front-end development tool. I came across this question upon seeing a .mix JavaScript file in action (in one of the Mixture boilerplates) and being a bit confused by it ("you can do this?" I thought to myself). Then I realized that it was an application-specific file type (somewhat disappointing, agreed). Nevertheless, figured the knowledge might be helpful for others.

UPDATE: Mixture is now free.

Discussion courtesy of: Isaac Gregson

In case you are using Web Workers and want to include additional scripts in the scope of the worker, the other answers provided about adding scripts to the head tag, etc. will not work for you.

Fortunately, Web Workers have their own importScripts function which is a global function in the scope of the Web Worker, native to the browser itself as it is part of the specification.

Alternatively, as the second highest voted answer to your question highlights, RequireJS can also handle including scripts inside a Web Worker (likely calling importScripts itself, but with a few other useful features).

Discussion courtesy of: Turnerj

There are a lot of potential answers for this question. My answer is obviously based on a number of them. Thank you for all the help. This is what I ended up with after reading through all the answers.

The problem with $.getScript and really any other solution that requires a callback when loading is complete is that if you have multiple files that use it and depend on each other you no longer have a way to know when all scripts have been loaded (once they are nested in multiple files).

Example:

file3.js

```
var f3obj = "file3";

// Define other stuff
```

file2.js:

```
var f2obj = "file2";
$.getScript("file3.js", function(){

alert(f3obj);

// Use anything defined in file3.
});
```

file1.js:

```
$.getScript("file2.js", function(){
alert(f3obj); //This will probably fail because file3 is only guaranteed to have loaded inside the callback in file2.
alert(f2obj);

// Use anything defined in the loaded script...
});
```

You are right when you say that you could specify Ajax to run synchronously or use XMLHttpRequest, but the current trend appears to be to deprecate synchronous requests, so you may not get full browser support now or in the future.

You could try to use $.when to check an array of deferred objects, but now you are doing this in every file and file2 will be considered loaded as soon as the $.when is executed not when the callback is executed, so file1 still continues execution before file3 is loaded. This really still has the same problem.

I decided to go backwards instead of forwards. Thank you document.writeln. I know it's taboo, but as long as it is used correctly this works well. You end up with code that can be debugged easily, shows in the DOM correctly and can ensure the order the dependencies are loaded correctly.

You can of course use $("body").append(), but then you can no longer debug correctly any more.

NOTE: You must use this only while the page is loading, otherwise you get a blank screen. In other words, **always place this before / outside of document.ready**. I have not tested using this after the page is loaded in a click event or anything like that, but I am pretty sure it'll fail.

I liked the idea of extending jQuery, but obviously you don't need to.

Before calling document.writeln, it checks to make sure the script has not already been loading by evaluating all the script elements.

I assume that a script is not fully executed until its document.ready event has been executed. (I know using document.ready is not required, but many people use it, and handling this is a safeguard.)

When the additional files are loaded the document.ready callbacks will get executed in the wrong order. To address this when a script is actually loaded, the script that imported it is re-imported itself and execution halted. This causes the originating file to now have its document.ready callback executed after any from any scripts that it imports.

Instead of this approach you could attempt to modify the jQuery readyList, but this seemed like a worse solution.

Solution:

```
$.extend(true,
{
import_js : function(scriptpath, reAddLast)
{
if (typeof reAddLast === "undefined" || reAddLast === null)
{
reAddLast = true; // Default this value to true. It is not used by the end user, only to facilitate recursion correctly.
```

```
}

var found = false;
if (reAddLast == true) // If we are re-adding the originating script we do not care if it has already been added.
{
found = $('script').filter(function () {
return ($(this).attr('src') == scriptpath);
}).length != 0; // jQuery to check if the script already exists. (replace it with straight JavaScript if you don't like jQuery.
}

if (found == false) {

var callingScriptPath = $('script').last().attr("src"); // Get the script that is currently loading. Again this creates a limitation where this should not be used in a button, and only before document.ready.

document.writeln("<script type='text/javascript' src='" + scriptpath + "'></script>"); // Add the script to the document using writeln

if (reAddLast)
{
$.import_js(callingScriptPath, false); // Call itself with the originating script to fix the order.
throw 'Readding script to correct order: ' + scriptpath + ' < ' + callingScriptPath; // This halts execution of the originating script since it is getting reloaded. If you put a try / catch around the call to $.import_js you results will vary.
}
return true;
}
return false;
}
});

Usage:

file3:

var f3obj = "file3";

// Define other stuff
$(function(){
f3obj = "file3docready";
});

file2:

$.import_js('js/file3.js');
var f2obj = "file2";
$(function(){
f2obj = "file2docready";
});

file1:

$.import_js('js/file2.js');
```

```
// Use objects from file2 or file3
alert(f3obj); // "file3"
alert(f2obj); // "file2"

$(function(){
// Use objects from file2 or file3 some more.
alert(f3obj); //"file3docready"
alert(f2obj); //"file2docready"
});
```

Discussion courtesy of: curlyhairedgenius

Keep it nice, short, simple, and maintainable! :]

```
// 3rd party plugins / script (don't forget the full path is necessary)
var FULL_PATH = '', s =
[
FULL_PATH + 'plugins/script.js' // Script example
FULL_PATH + 'plugins/jquery.1.2.js', // jQuery Library
FULL_PATH + 'plugins/crypto-js/hmac-sha1.js', // CryptoJS
FULL_PATH + 'plugins/crypto-js/enc-base64-min.js' // CryptoJS
];

function load(url)
{
var ajax = new XMLHttpRequest();
ajax.open('GET', url, false);
ajax.onreadystatechange = function ()
{
var script = ajax.response || ajax.responseText;
if (ajax.readyState === 4)
{
switch(ajax.status)
{
case 200:
eval.apply( window, [script] );
console.log("library loaded: ", url);
break;
default:
console.log("ERROR: library not loaded: ", url);
}
}
};
ajax.send(null);
}

// initialize a single load
load('plugins/script.js');

// initialize a full load of scripts
if (s.length > 0)
{
for (i = 0; i < s.length; i++)
{
load(s[i]);
}
}
```

This code is simply a short functional example that *could* require additional

feature functionality for full support on any (or given) platform.

Discussion courtesy of: tfont

Statement import is in ECMAScript 6.

Syntax

```
import name from "module-name";
import { member } from "module-name";
import { member as alias } from "module-name";
import { member1 , member2 } from "module-name";
import { member1 , member2 as alias2 , [...] } from "module-name";
import name , { member [ , [...] ] } from "module-name";
import "module-name" as name;
```

Discussion courtesy of: draupnie

Here's the generalized version of how Facebook does it for their ubiquitous Like button:

```
<script>
var firstScript = document.getElementsByTagName('script')[0],
js = document.createElement('script');
js.src = 'https://cdnjs.cloudflare.com/ajax/libs/Snowstorm/20131208/snowstorm-min.js';
js.onload = function () {
// do stuff with your dynamically loaded script
snowStorm.snowColor = '#99ccff';
};
firstScript.parentNode.insertBefore(js, firstScript);
</script>
```

If it works for Facebook, it will work for you.

The reason why we look for the first script element instead of head or body is because some browsers don't create one if missing, but we're guaranteed to have a script element - this one. Read more at http://www.jspatterns.com/the-ridiculous-case-of-adding-a-script-element/.

Discussion courtesy of: Dan Dascalescu

I had a simple issue, but I was baffled by responses to this question.

I had to use a variable (myVar1) defined in one JavaScript file (myvariables.js) in another JavaScript file (main.js).

For this I did as below:

Loaded the JavaScript code in the HTML file, in the correct order, myvariables.js first, then main.js:

```
<html>
<body onload="bodyReady();" >

<script src="myvariables.js" > </script>
```

```
<script src="main.js" > </script>

<!-- Some other code -->
</body>
</html>

File: myvariables.js

var myVar1 = "I am variable from myvariables.js";

File: main.js

// ...
function bodyReady() {
// ...
alert (myVar1); // This shows "I am variable from myvariables.js", which I needed
// ...
}
// ...
```

As you saw, I had use a variable in one JavaScript file in another JavaScript file, but I didn't need to include one in another. I just needed to ensure that the first JavaScript file loaded before the second JavaScript file, and, the first JavaScript file's variables are accessible in the second JavaScript file, automatically.

This saved my day. I hope this helps.

Discussion courtesy of: Manohar Reddy Poreddy

If your intention to load the JavaScript file is **using the functions from the imported/included file**, you can also define a global object and set the functions as object items. For instance:

global.js

```
A = {};
```

file1.js

```
A.func1 = function() {
console.log("func1");
}
```

file2.js

```
A.func2 = function() {
console.log("func2");
}
```

main.js

```
A.func1();
A.func2();
```

You just need to be careful when you are including scripts in an HTML file. The

order should be as in below:

```
<head>
<script type="text/javascript" src="global.js"></script>
<script type="text/javascript" src="file1.js"></script>
<script type="text/javascript" src="file2.js"></script>
<script type="text/javascript" src="main.js"></script>
</head>
```

Discussion courtesy of: Adem İlhan

I basically do it like the following, creating a new element and attach that to head:

```
var x = document.createElement('script');
x.src = 'http://example.com/test.js';
document.getElementsByTagName("head")[0].appendChild(x);
```

In jQuery:

```
// jQuery
$.getScript('/path/to/imported/script.js', function()
{
// Script is now loaded and executed.
// Put your dependent JavaScript code here.
});
```

Discussion courtesy of: Rahul Srivastava

I have the requirement to asynchronously load an array of JavaScript files and at the final make a callback. Basically my best approach is the following:

```
// Load a JavaScript file from other JavaScript file
function loadScript(urlPack, callback) {
var url = urlPack.shift();
var subCallback;

if (urlPack.length == 0) subCallback = callback;
else subCallback = function () {
console.log("Log script: " + new Date().getTime());
loadScript(urlPack, callback);
}

// Adding the script tag to the head as suggested before
var head = document.getElementsByTagName('head')[0];
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = url;

// Then bind the event to the callback function.
// There are several events for cross browser compatibility.
script.onreadystatechange = subCallback;
script.onload = subCallback;

// Fire the loading
head.appendChild(script);
}
```

Example:

```
loadScript(
[
"js/DataTable/jquery.dataTables.js",
"js/DataTable/dataTables.bootstrap.js",
"js/DataTable/dataTables.buttons.min.js",
"js/DataTable/dataTables.colReorder.min.js",
"js/DataTable/dataTables.fixedHeader.min.js",
"js/DataTable/buttons.bootstrap.min.js",
"js/DataTable/buttons.colVis.min.js",
"js/DataTable/buttons.html5.min.js"
], function() { gpLoad(params); });
```

The second script will not load until the first is completely loaded, and so...

Results:

```
Log script: 1470596913596
Log script: 1470596913604
Log script: 1470596913608
Log script: 1470596913613
Log script: 1470596913646
Log script: 1470596913658
Log script: 1470596913660
```

Discussion courtesy of: MiBol

You can't import, but you can reference.

PhpShtorm IDE. To reference, in one .js file to another .js, just add this to the top of the file:

```
<reference path="../js/file.js" />
```

Of course, you should use your own PATH to the JavaScript file.

I don't know if it will work in other IDEs. Probably yes, just try. It should work in Visual Studio too.

Discussion courtesy of: Evgeniy Miroshnichenko

Here is maybe another way! In Node.js you do that just like the following! http://requirejs.org/docs/node.html

sub.js

```
module.exports = {
log: function(string) {
if(console) console.log(string);
}
mylog: function(){
console.log('just for log test!');
}
}
```

main.js

```
var mylog =require('./sub');

mylog.log('Hurray, it works! :)');
mylog.mylog();
```

Discussion courtesy of: xqqfrms

```
var xxx = require("../lib/your-library.js")
```

or

```
import xxx from "../lib/your-library.js" //get default export
import {specificPart} from '../lib/your-library.js' //get named export
import * as _name from '../lib/your-library.js' //get full export to alias _name
```

Discussion courtesy of: Mesut Yiğit

Another approach is to use HTML imports. These can contain script references as well as stylesheet references.

You can just link an HTML file like

```
<link rel="import" href="vendorScripts.html"/>
```

Within the vendorScripts.html file you can include your script references like:

```
<script src="scripts/vendors/jquery.js"></script>
<script src="scripts/vendors/bootstrap.js"></script>
<script src="scripts/vendors/angular.js"></script>
<script src="scripts/vendors/angular-route.js"></script>
```

Look at https://www.html5rocks.com/en/tutorials/webcomponents/imports/ for more details.

Unfortunately this only works in Chrome.

Discussion courtesy of: gabriel211

If you use Angular, then you can a plugin module $ocLazyLoad. Here are some quotes from its documentation:

Load one or more modules & components with multiple files:

```
$ocLazyLoad.load(['testModule.js',                    'testModuleCtrl.js',
'testModuleService.js']);
```

Load one or more modules with multiple files and specify a type where necessary: Note: When using the requireJS style formatting (with js! at the beginning for example), do not specify a file extension. Use one or the other.

```
$ocLazyLoad.load([
'testModule.js',
{type: 'css', path: 'testModuleCtrl'},
```

```
{type: 'html', path: 'testModuleCtrl.html'},
{type: 'js', path: 'testModuleCtrl'},
'js!testModuleService',
'less!testModuleLessFile'
]);
```

You can load external libs (not angular):

```
$ocLazyLoad.load(['testModule.js',
'bower_components/bootstrap/dist/js/bootstrap.js', 'anotherModule.js']);
```

You can also load css and template files:

```
$ocLazyLoad.load([
'bower_components/bootstrap/dist/js/bootstrap.js',
'bower_components/bootstrap/dist/css/bootstrap.css',
'partials/template1.html'
]);
```

Discussion courtesy of: gm2008

Here's a workaround **for browsers** (not Node.js) using HTML imports.

First, all JavaScript classes and scripts are not in .js files, but in .js.html files (the **.js**.html is just to recognize between HTML pages and complete JavaScript script/classes), inside <script> tags, like this:

MyClass.js.html:

```
<script>
function MyClass(){}

// Your code here..
</script>
```

Then if you wish to import your class, you just need to use HTML imports:

```
<link rel="import" href="relative/path/to/MyClass.js.html"/>

<script>
var myClass = new MyClass();
// Your code here..
</script>
```

Discussion courtesy of: Jeremy

It's very simple. Suppose you want to import file A.js in file B.js.

Now it's sure you have linked B.js in an HTML file, then just link A.js before B.js in that HTML file. Then the public variables of A.js will be available inside the B.js

This does not require a complicated answer.

Discussion courtesy of: Akshay Vijay Jain

How do I escape a string for a shell command in node?

Problem

In nodejs, the only way to execute external commands is via sys.exec(cmd). I'd like to call an external command and give it data via stdin. In nodejs there does yet not appear to be a way to open a command and then push data to it (only to exec and receive its standard+error outputs), so it appears the only way I've got to do this right now is via a single string command such as:

```
var dangerStr = "bad stuff here";
sys.exec("echo '" + dangerStr + "' | somecommand");
```

Most answers to questions like this have focused on either regex which doesn't work for me in nodejs (which uses Google's V8 Javascript engine) or native features from other languages like Python.

I'd like to escape dangerStr so that it's safe to compose an exec string like the one above. If it helps, dangerStr will contain JSON data.

Problem courtesy of: Maciek

Solution

There is a way to write to an external command: process.createChildProcess (documentation) returns an object with a write method. createChildProcess isn't as convenient though, because it doesn't buffer stdout and stderr, so you will need event handlers to read the output in chunks.

```
var stdout = "", stderr = "";
var child = process.createChildProcess("someCommand");

child.addListener("output", function (data) {
if (data !== null) {
stdout += data;
}
});
child.addListener("error", function (data) {
if (data !== null) {
stderr += data;
}
});
child.addListener("exit", function (code) {
if (code === 0) {
sys.puts(stdout);
}
else {
// error
}
});

child.write("This goes to someCommand's stdin.");
```

Solution courtesy of: Matthew Crumley

Discussion

This is what I use:

```
var escapeShell = function(cmd) {
return '"'+cmd.replace(/(["\s'$`\\])/g,'\\$1')+'"';
};
```

Discussion courtesy of: Sylvain Zimmer

If you need simple solution you can use this:

```
function escapeShellArg (cmd) {
return '\'' + cmd.replace(/\'/g, "'\\''") + '\'';
}
```

So your string will be simply escaped with single quotes as Chris Johnsen mentioned.

```
echo 'John'\''s phone';
```

It works in bash because of strong quoting, feels like it also works in fish, but does not work in zsh and sh.

If you have bash your can run your script in sh or zsh with 'bash -c \'' + escape('all-the-rest-escaped') + '\''.

But actually... node.js will escape all needed characters for you:

```
var child = require('child_process')
.spawn('echo', ['`echo 1`;"echo $SSH_TTY;\'\\0{0..5}']);

child.stdout.on('data', function (data) {
console.log('stdout: ' + data);
});

child.stderr.on('data', function (data) {
console.log('stderr: ' + data);
});
```

this block of code will execute:

```
echo '`echo 1`;"echo $SSH_TTY;'\''\\0{0..5}'
```

and will output:

```
stdout: `echo 1`;"echo $SSH_TTY;\'\\0{0..5}
```

or some error.

Take a look at http://nodejs.org/api/child_process.html#child_process_child_process_spawn_comma

By the way simple solution to run a bunch of commands is:

```
require('child_process')
.spawn('sh', ['-c', [
'cd all/your/commands',
'ls here',
'echo "and even" > more'
].join('; ')]);
```

Have a nice day!

Discussion courtesy of: Alex Yaroshevich

If you also need to deal with special character (line-breaks etc.) you can do it this way:

```
str = JSON.stringify(str)
.replace(/^"|"$/g,'') //remove JSON-string double quotes
.replace(/'/g, '\'"\'"\'') //escape single quotes the ugly bash way
```

This assumes you use Bash's strong-quoting via single-quotes) and the receiver can understand JSON's C-like escaping.

Discussion courtesy of: MicMro

A variation on Sylvain's answer:

```
var escapeShell = function(string) {
// Sanitise all space (newlines etc.) to a single space
string.replace(/\s+/g, " ");
// Optionally remove leading and trailing space
string.replace(/^\s+|\s+$/g, " ");
// Quote with single quotes, escaping backslashes and single quotes
return "'" + string.replace(/(['\\])/g, '\\$1') + "'";
};
```

On the premise that $ does not need to be escaped in a single quoted string (as Alex pointed out) and the other quote types don't need to be either.

Discussion courtesy of: SuperDuperApps

How do I debug Node.js applications?

Problem

How do I debug a Node.js server application?

Right now I'm mostly using *alert debugging* with print statements like this:

```
sys.puts(sys.inspect(someVariable));
```

There must be a better way to debug. I know that Google Chrome has a command-line debugger. Is this debugger available for Node.js as well?

Problem courtesy of: Fabian Jakobs

Solution

The V8 debugger released as part of the Google Chrome Developer Tools can be used to debug Node.js scripts. A detailed explanation of how this works can be found in the Node.js GitHub wiki.

There is also ndb, a command line debugger written in Node.js itself.

Solution courtesy of: Fabian Jakobs

Discussion

node-inspector could save the day! Use it from any browser supporting WebSocket. Breakpoints, profiler, livecoding, etc... It is really awesome.

Install it with:

npm install -g node-inspector

Then run:

node-debug app.js

Discussion courtesy of: daralthus

Node.js version 0.3.4+ has built-in debugging support.

node debug script.js

Manual: http://nodejs.org/api/debugger.html

Discussion courtesy of: JulianW

There is built-in command line debugger client within Node.js. Cloud 9 IDE have also pretty nice (visual) debugger.

Discussion courtesy of: vojimbo87

If you need a powerful logging library for Node.js, Tracer https://github.com/baryon/tracer is a better choice.

It outputs log messages with a timestamp, file name, method name, line number, path or call stack, support color console, and support database, file, stream transport easily. I am the author.

Discussion courtesy of: Baryon Lee

I personally use JetBrains WebStorm as it's the only JavaScript IDE that I've found which is great for both frontend and backend JavaScript.

It works on multiple OS's and has Node.js debugging built-in (as well as a ton of other stuff](http://www.jetbrains.com/webstorm/features/index.html).

My only 'issues'/wishlist items ~~are~~ **were**:

1. ~~It seems to be more resource hungry on Mac than Windows~~ It no longer seems an issue in version 6.
2. ~~It would be nice if it had Snippet support (like those of Sublime Text 2 - i.e. type 'fun' and tap 'tab' to put in a function.~~ See @WickyNilliams comment below - With Live Templates you also have snippet support.

Discussion courtesy of: isNaN1247

Assuming you have node-inspector installed on your computer (if not, just type 'npm install -g node-inspector') you just have to run:

```
node-inspector & node --debug-brk scriptFileName.js
```

And paste the URI from the command line into a WebKit (Chrome / Safari) browser.

Discussion courtesy of: Shaheen Ghiassy

Debugging

- Joyent's Guide
- Debugger
- Node Inspector
- Visual Studio Code
- Cloud9
- Brackets

Profiling

1. `node --prof ./app.js`
2. `node --prof-process ./the-generated-log-file`

Heapdumps

- node-heapdump with Chrome Developer Tools

Flamegraphs

- 0x
- jam3/devtool then Chrome Developer Tools Flame Charts
- Dtrace and StackVis — Only supported on SmartOS

Tracing

- Interactive Stack Traces with TraceGL

Logging

Libraries that output debugging information

- Caterpillar
- Tracer

Libraries that enhance stack trace information

- Longjohn

Benchmarking

- Apache Bench: ab -n 100000 -c 1 http://127.0.0.1:9778/
- wrk

Other

- Trace
- Vantage
- Bugger
- Google Tracing Framework
- Paul Irish's Guide

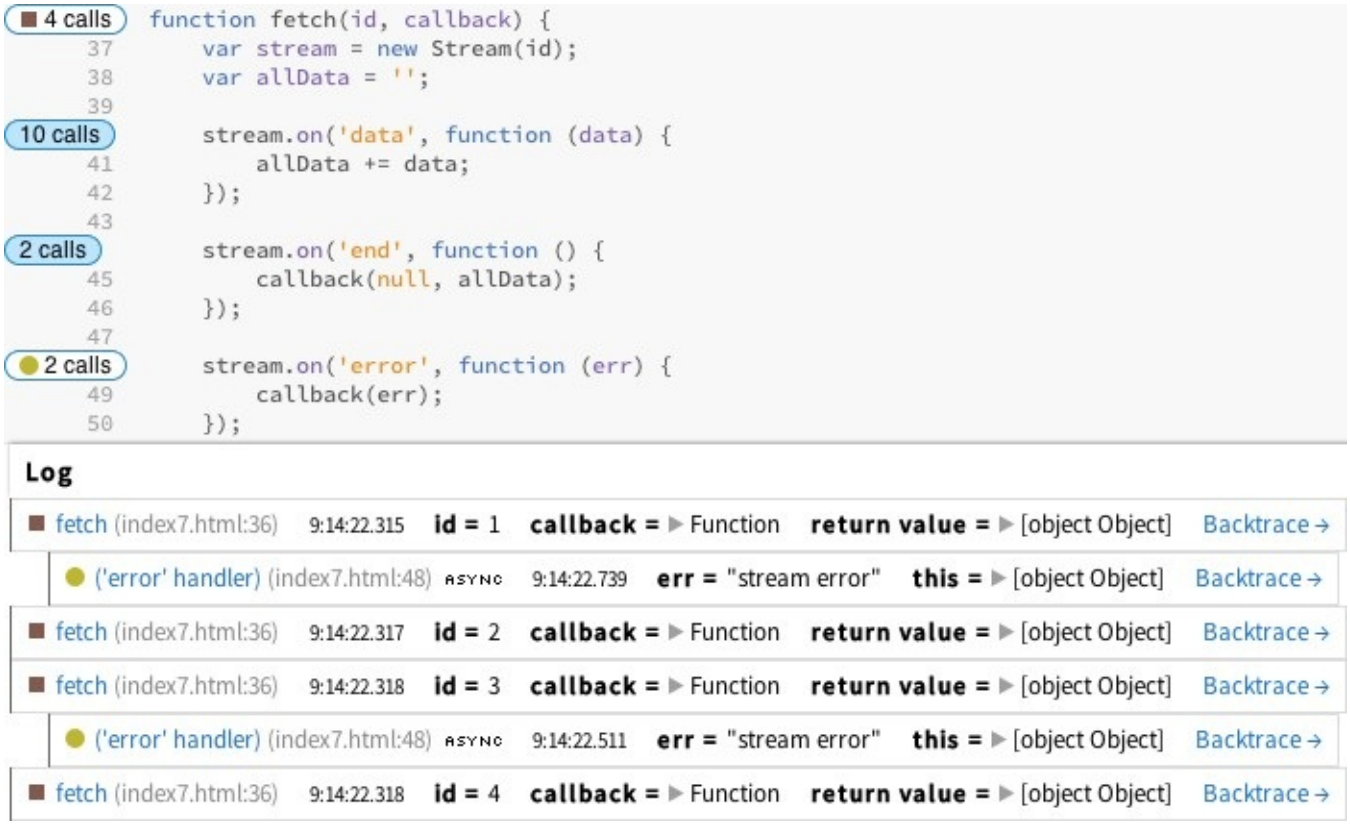
Legacy

These use to work but are no longer maintained or no longer applicable to modern node versions.

- https://github.com/bnoordhuis/node-profiler - replaced by built-in debugging
- https://github.com/c4milo/node-webkit-agent - replaced by node inspector
- https://nodetime.com/ - defunct

Discussion courtesy of: balupton

Theseus is a project by Adobe research which lets you debug your Node.js code in their Open Source editor Brackets. It has some interesting features like real-time code coverage, retroactive inspection, asynchronous call tree.



Discussion courtesy of: Sindre Sorhus

I put together a short Node.js debugging primer on using the node-inspector for those who aren't sure where to get started.

Discussion courtesy of: Josh H

There is the new open-source Nodeclipse project (as a Eclipse plugin or Enide Studio):

http://www.nodeclipse.org/img/Nodeclipse-1-debugging.png

Nodeclipse became #1 in Eclipse Top 10 NEW Plugins for 2013. It uses a modified

V8 debugger (from Google Chrome Developer Tools for Java).

Nodeclipse is free open-source software released at the start of every month.

Discussion courtesy of: Paul Verest

Just for completeness:

The PyCharm 3.0 + Node.js Plugin offers an awesome development + run + debug experience.

Discussion courtesy of: Alex

I would use GOOD by Walmart Labs. It will do the job, and it's very flexible:

```
var hapi = require('hapi');
var good = require('good');
var server = hapi.createServer('localhost', 5000,{});
server.route({SOME ROUTE HERE});
server.start();

var options = {
subscribers: {
'console': ['ops', 'request', 'log', 'error'],
'http://localhost/logs': ['log']
}
};
server.pack.require('good', options, function (err) {

if (!err) {
console.log('Plugin loaded successfully');
}
});
```

Discussion courtesy of: Doron Segal

Node.js Tools for Visual Studio 2012 or 2013 includes a debugger. The overview here states "Node.js Tools for Visual Studio includes complete support for debugging node apps.". Being new to Node.js, but having a background in .NET, I've found this add in to be a great way to debug Node.js applications.

Discussion courtesy of: John81

```
node-debug -p 8888 scriptFileName.js
```

Discussion courtesy of: matt burns

I created a neat little tool called pry.js that can help you out.

Put a simple statement somewhere in your code, run your script normally and node will halt the current thread giving you access to all your variables and functions. View/edit/delete them at will!

```
pry = require('pryjs')
```

```
class FizzBuzz

run: ->
for i in [1..100]
output = ''
eval(pry.it) # magic
output += "Fizz" if i % 3 is 0
output += "Buzz" if i % 5 is 0
console.log output || i

bar: ->
10

fizz = new FizzBuzz()
fizz.run()
```

Discussion courtesy of: BlaineSch

If you are using the Atom IDE, you can install the node-debugger package.

Discussion courtesy of: Uchiha Itachi

A quick-and-dirty way to debug small Node.js scripts with your favorite **browser debugger** would be to use **browserify**. Note that this approach doesn't work with any applications which require native I/O libraries, but it is good enough for most small scripts.

```
$ npm install -g browserify
```

Now move all your var x = requires('x') calls into a requires.js file and run:

```
$ browserify requires.js -s window -o bundle.js
```

(The downside here is that you either have to move or comment the requires in all your files.)

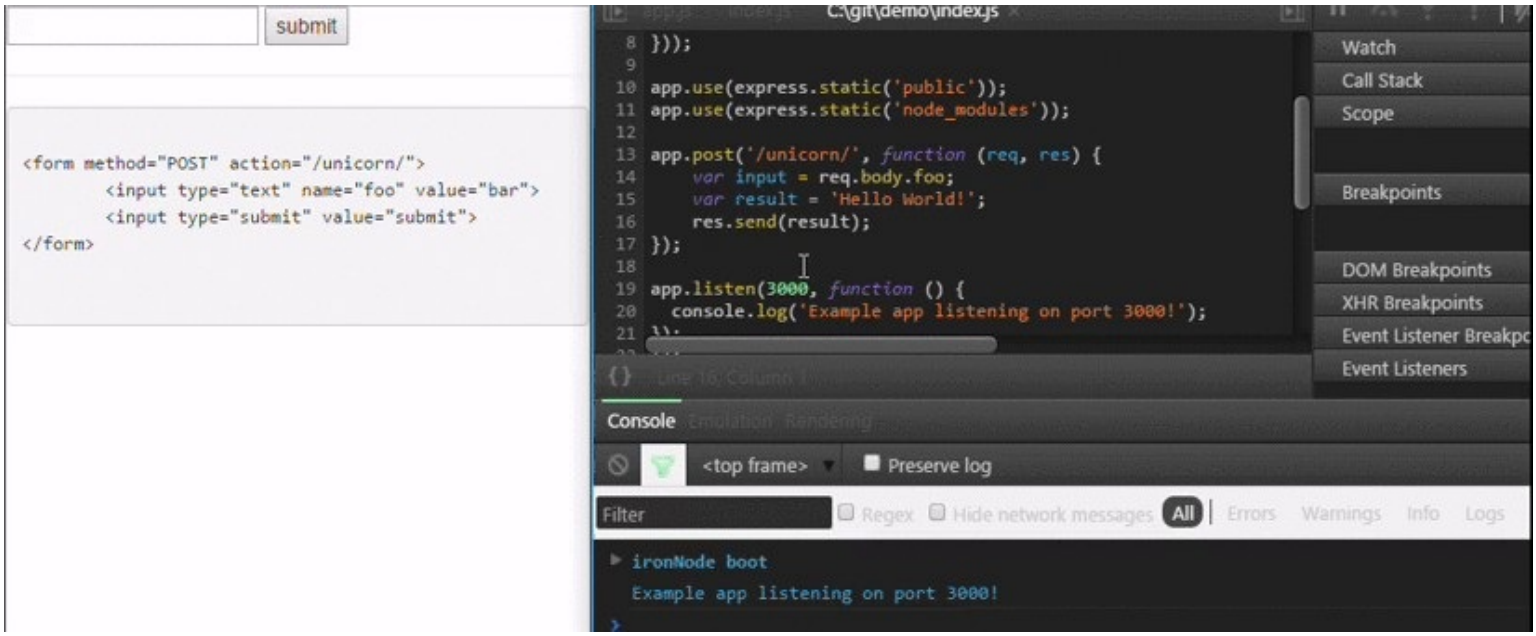
Include the bundle.js in an HTML file like so:

```
<script type="text/javascript" src="bundle.js"></script>
```

Now load the file in your browser and press F12 and viola: debug in browser.

Discussion courtesy of: Gerold Meisinger

I wrote a different approach to debug Node.js code which is stable and is extremely simple. It is available at https://github.com/s-a/iron-node.



An opensource cross-platform visual debugger.

Installation:

npm install iron-node -g;

Debug:

iron-node yourscript.js;

Discussion courtesy of: Stephan Ahlf

IntelliJ works wonderfully for Node.js.

In addition, IntelliJ supports 'Code Assistance' well.

Discussion courtesy of: Shengyuan Lu

Visual Studio Code has really nice Node.js debugging support. It is free, open source and cross-platform and runs on Linux, OS X and Windows.

You can even debug grunt and gulp tasks, should you need to...

Discussion courtesy of: hans

A lot of great answers here, but I'd like to add my view (based on how my approach evolved)

Debug Logs

Let's face it, we all love a good console.log('Uh oh, if you reached here, you better run.') and sometimes that works great, so if you're reticent to move too far away from it at least add some bling to your logs with Visionmedia's debug.

Interactive Debugging

As handy as console logging can be, to debug professionally you need to roll up your sleeves and get stuck in. Set breakpoints, step through your code, inspect scopes and variables to see what's causing that weird behaviour. As others have mentioned, node-inspector really is the bees-knees. It does everything you can do with the built-in debugger, but using that familiar Chrome DevTools interface. If, like me, you use **Webstorm**, then here is a handy guide to debugging from there.

Stack Traces

By default, we can't trace a series of operations across different cycles of the event loop (ticks). To get around this have a look at longjohn (but not in production!).

Memory Leaks

With Node.js we can have a server process expected to stay up for considerable time. What do you do if you think it has sprung some nasty leaks? Use heapdump and Chrome DevTools to compare some snapshots and see what's changing.

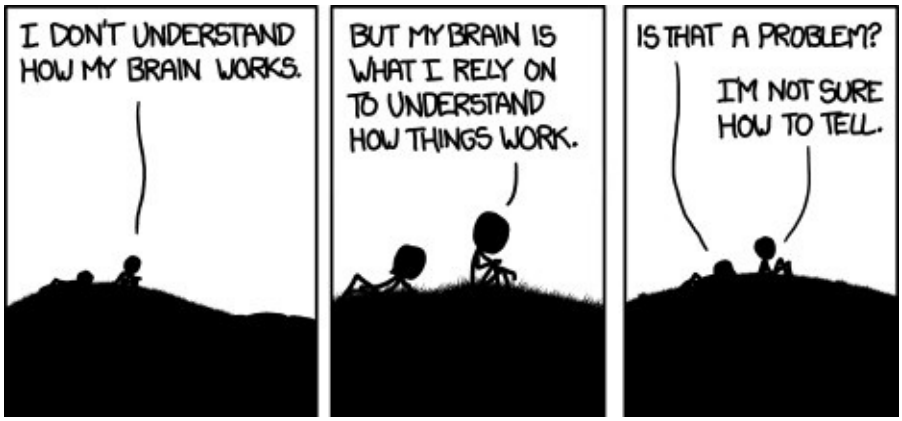
For some useful articles, check out

- RisingStack - Debugging Node.js Applications
- Excellent article by David Mark Clements of nearForm

If you feel like watching a video(s) then

- Netflix JS Talks - Debugging Node.js in Production
- Interesting video from the tracing working group on tracing and debugging node.js
- Really informative 15-minute video on node-inspector

Whatever path you choose, just be sure you understand how you are debugging



It is a painful thing
To look at your own trouble and know
That you yourself and no one else has made it

Sophocles, Ajax

Discussion courtesy of: Philip O'Brien

The NetBeans IDE has had Node.js support since version 8.1:

<...>

New Feature Highlights

Node.js Application Development

- New Node.js project wizard
- New Node.js Express wizard
- Enhanced JavaScript Editor
- New support for running Node.js applications
- New support for debugging Node.js applications.

<...>

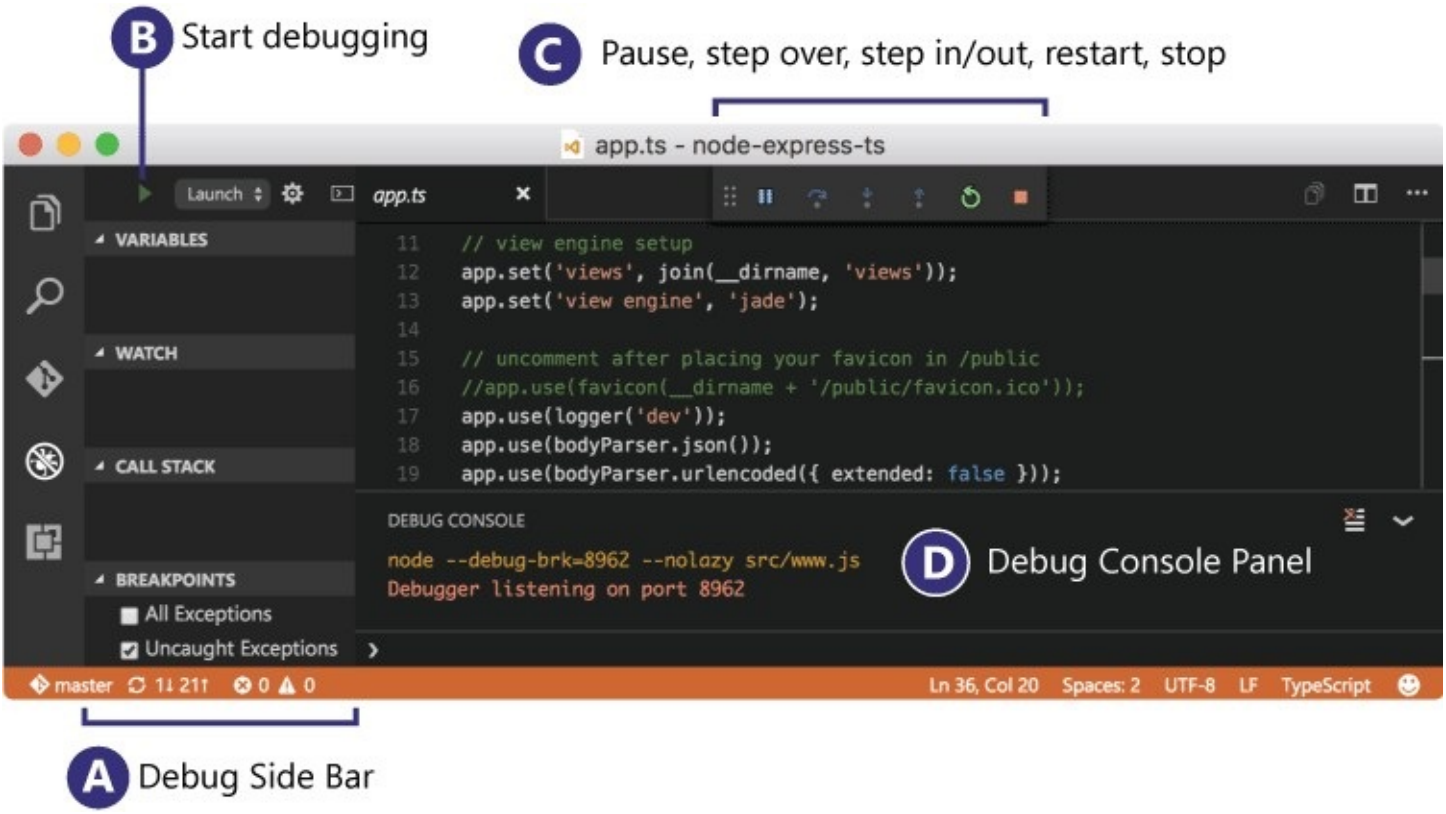
Additional references:

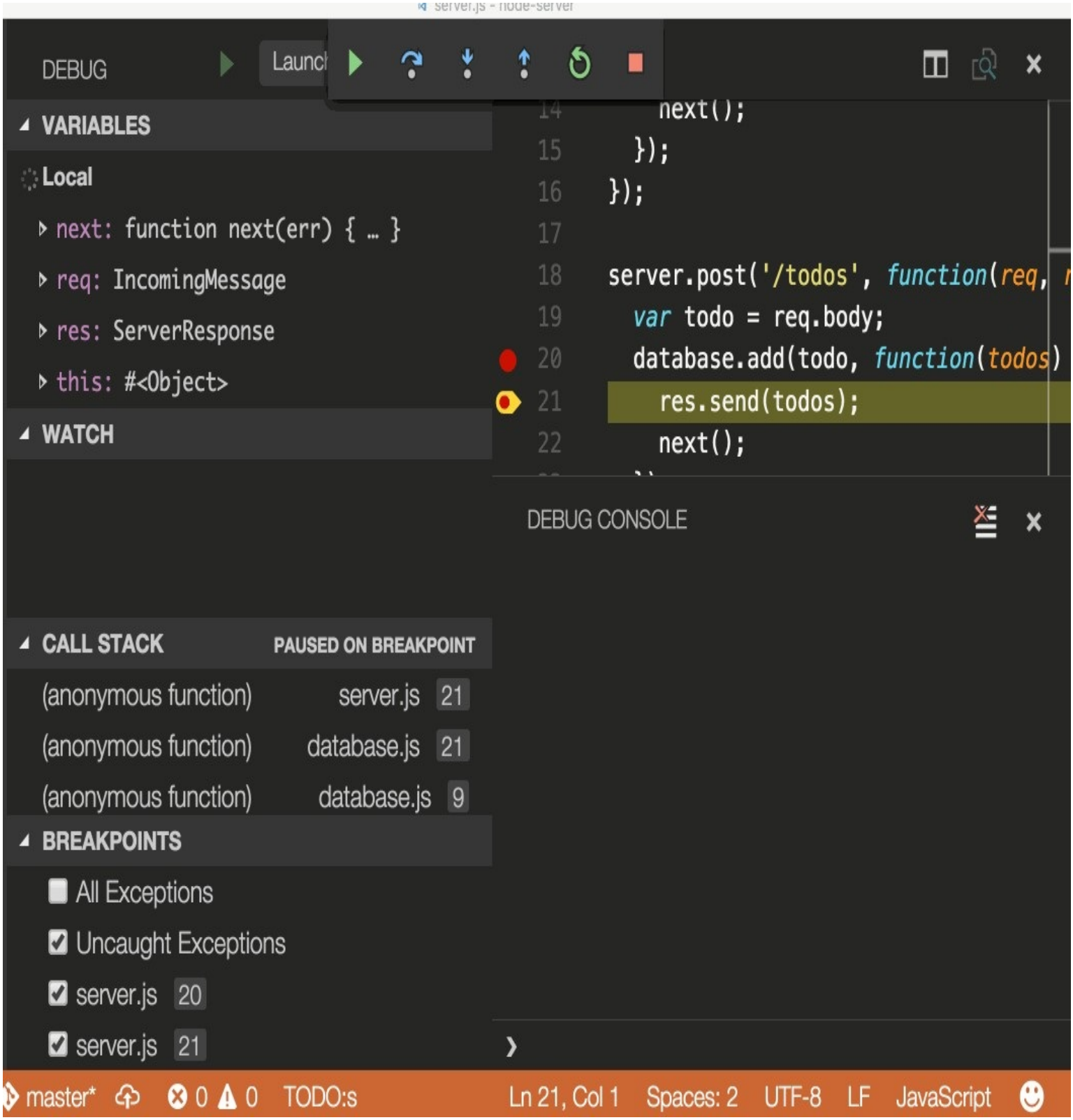
1. NetBeans Wiki / NewAndNoteworthyNB81.
2. Node.js Express App in NetBeans IDE, Geertjan-Oracle.

Discussion courtesy of: Sergey Brunov

Visual Studio Code will be my choice for debugging. No overhead of installing any tools or npm install stuff. Just set the starting point of your app in package.json and VSCode will automatically create a configuration file inside your solution. It's build on Electron, on which editors like Atom are built.

VS Code gives similar debugging experience as you might have had in other IDEs like VS, Eclipse, etc.



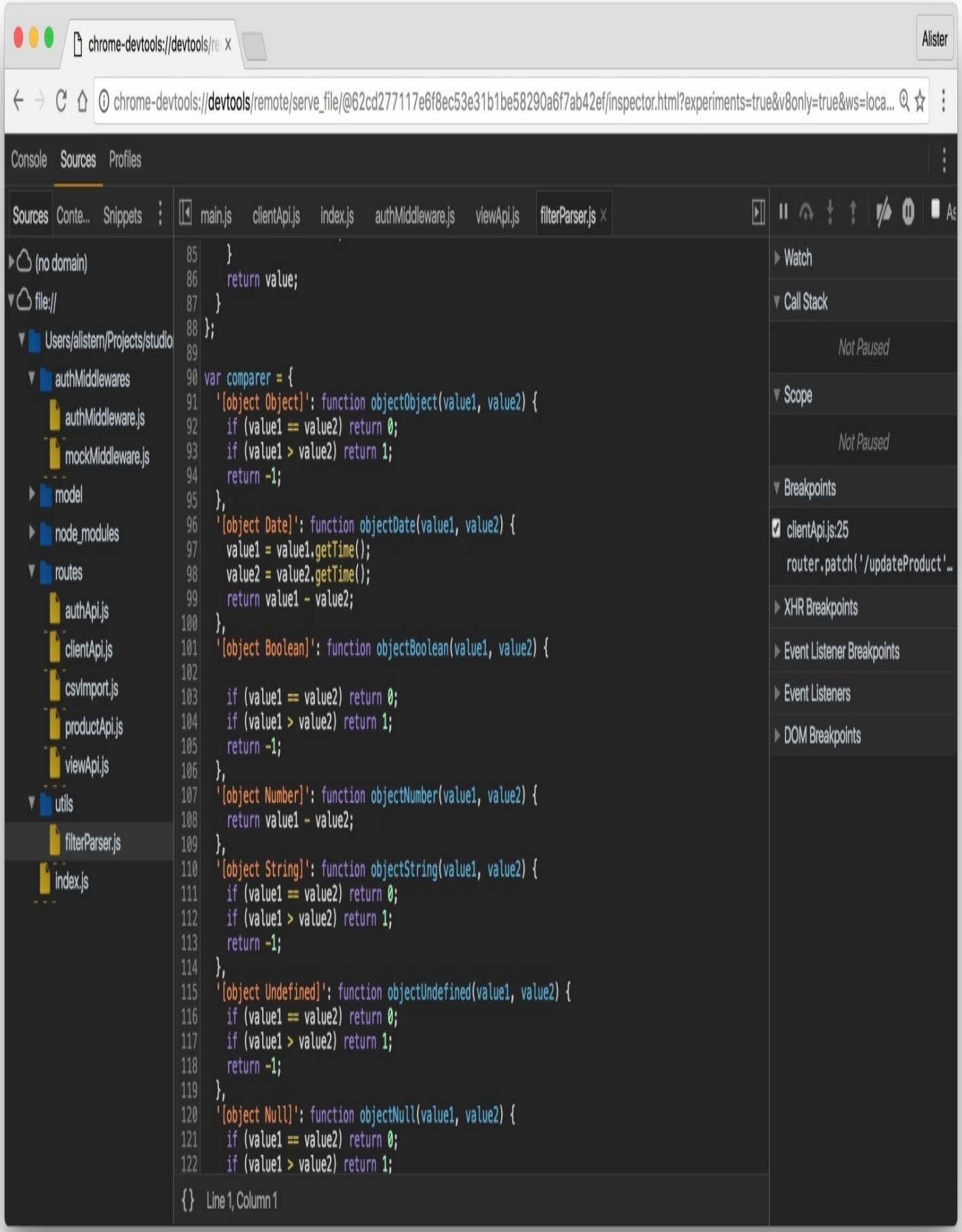


Discussion courtesy of: Shreyas

Use Webstorm! It's perfect for debugging Node.js applications. It has a built-in debugger. Check out the docs here: https://www.jetbrains.com/help/webstorm/2016.1/running-and-debugging-node-js.html

Discussion courtesy of: OneMoreQuestion

Node has its own built in GUI debugger as of version 6.3 (using Chrome's DevTools)


chrome-devtools://devtools/re
Alister
chrome-devtools://devtools/remote/serve_file/@62cd277117e6f8ec53e31b1be58290a6f7ab42ef/inspector.html?experiments=true&v8only=true&ws=loca...
Console
Sources
Profiles
Sources
Conte...
Snippets
main.js
clientApi.js
index.js
authMiddleware.js
viewApi.js
filterParser.js
(no domain)
file://
Users/alistern/Projects/studio
authMiddlewares
authMiddleware.js
mockMiddleware.js
model
node_modules
routes
authApi.js
clientApi.js
csvImport.js
productApi.js
viewApi.js
utils
filterParser.js
index.js
85 }
86 return value;
87 }
88 };
89
90 var comparer = {
91 '[object Object]': function objectObject(value1, value2) {
92 if (value1 == value2) return 0;
93 if (value1 > value2) return 1;
94 return -1;
95 },
96 '[object Date]': function objectDate(value1, value2) {
97 value1 = value1.getTime();
98 value2 = value2.getTime();
99 return value1 - value2;
100 },
101 '[object Boolean]': function objectBoolean(value1, value2) {
102
103 if (value1 == value2) return 0;
104 if (value1 > value2) return 1;
105 return -1;
106 },
107 '[object Number]': function objectNumber(value1, value2) {
108 return value1 - value2;
109 },
110 '[object String]': function objectString(value1, value2) {
111 if (value1 == value2) return 0;
112 if (value1 > value2) return 1;
113 return -1;
114 },
115 '[object Undefined]': function objectUndefined(value1, value2) {
116 if (value1 == value2) return 0;
117 if (value1 > value2) return 1;
118 return -1;
119 },
120 '[object Null]': function objectNull(value1, value2) {
121 if (value1 == value2) return 0;
122 if (value1 > value2) return 1;
{} Line 1, Column 1
Watch
Call Stack
Not Paused
Scope
Not Paused
Breakpoints
clientApi.js:25
router.patch('/updateProduct'...
XHR Breakpoints
Event Listener Breakpoints
Event Listeners
DOM Breakpoints

Simply pass the inspector flag and you'll be provided with a URL to the inspector:

```
node --inspect server.js
```

You can also break on the first line by passing --inspect-brk instead.

To open a Chrome window automatically, use the inspect-process module.

```
# install inspect-process globally
npm install -g inspect-process

# start the debugger with inspect
inspect script.js
```

Discussion courtesy of: Alister Norris

There are many possibilities...

- node includes a debugging utility
- node-inspector
- Code editors / IDEs (see debug instructions for one of the following)
  - Atom,
  - VSCode
  - Webstorm
  - and more

Debug support is often implemented using the v8 Debugging Protocol or the newer Chrome Debugging Protocol.

Discussion courtesy of: cmd

Auto-reload of files in Node.js

Problem

EDIT: Use http://github.com/isaacs/node-supervisor; This is an old question and the code example is made with an outdated Node.js API.

Any ideas on how I could implement an auto-reload of files in node.js. I'm tired of restarting the server every time I change a file. Apparently Node.js' require() function does not reload files if they already have been required, so I need to do something like this:

```
var sys = require('sys'),
http = require('http'),
posix = require('posix'),
json = require('./json');

var script_name = '/some/path/to/app.js';
this.app = require('./app').app;

process.watchFile(script_name, function(curr, prev){
posix.cat(script_name).addCallback(function(content){
process.compile( content, script_name );
});
});

http.createServer(this.app).listen( 8080 );
```

And in the *app.js* file I have:

```
var file = require('./file');
this.app = function(req, res) {
file.serveFile( req, res, 'file.js');
}
```

But this also isn't working - I get an error in the process.compile() statement saying that 'require' is not defined. process.compile is evaling the *app.js*, but has no clue about the node.js globals.

Problem courtesy of: disc0dancer

Solution

A good, up to date alternative to supervisor is **nodemon**:

Monitor for any changes in your node.js application and automatically restart the server - perfect for development

To use nodemon:

```
$ npm install nodemon -g
$ nodemon app.js
```

Solution courtesy of: Marius Butuc

Discussion

Edit: My answer is obsolete. Node.js is a very fast changing technology.

I also wondered about **reloading** modules. I modified node.js and have published the source at Github at nalply/node. The only difference is the function require. It has an optional second argument reload.

require(url, reload)

To reload app.js in current directory use

app = require("./app", true);

Write something like this, and you have **auto**-reload:

```
process.watchFile(script_name, function(curr, prev) {
module = reload(script_name, true);
});
```

The only problem I see is the variable module, but I am working at it now.

Discussion courtesy of: nalply

Take a look at require.hot() patch by Felix Geisendörfer. I use it with modified Nerve framework.

EDIT: link no longer works, node repo moved to nodejs/node, discarding all very old commits, it seems.

Discussion courtesy of: Kuroki Kaze

There was a recent thread about this subject on the node.js mailing list. The short answer is no, it's currently not possible auto-reload required files, but several people have developed patches that add this feature.

Discussion courtesy of: Xavi

There is Node-Supervisor that you can install by

npm install supervisor

see http://github.com/isaacs/node-supervisor

Discussion courtesy of: Richard Metzler

Here is a blog post about Hot Reloading for Node. It provides a github Node branch that you can use to replace your installation of Node to enable Hot Reloading.

From the blog:

```
var requestHandler = require('./myRequestHandler');
```

```
process.watchFile('./myRequestHandler', function () {
module.unCacheModule('./myRequestHandler');
requestHandler = require('./myRequestHandler');
}

var reqHandlerClosure = function (req, res) {
requestHandler.handle(req, res);
}

http.createServer(reqHandlerClosure).listen(8000);
```

Now, any time you modify myRequestHandler.js, the above code will notice and replace the local requestHandler with the new code. Any existing requests will continue to use the old code, while any new incoming requests will use the new code. All without shutting down the server, bouncing any requests, prematurely killing any requests, or even relying on an intelligent load balancer.

Discussion courtesy of: Chetan

solution at: http://github.com/shimondoodkin/node-hot-reload

notice that you have to take care by yourself of the references used.

that means if you did : var x=require('foo'); y=x;z=x.bar; and hot reloaded it.

it means you have to replace the references stored in x, y and z. in the hot reaload callback function.

some people confuse hot reload with auto restart my nodejs-autorestart module also has upstart integration to enable auto start on boot. if you have a small app auto restart is fine, but when you have a large app hot reload is more suitable. simply because hot reload is faster.

Also I like my node-inflow module.

Discussion courtesy of: Shimon Doodkin

node-supervisor is awesome

usage to restart on save:

```
npm install supervisor -g
supervisor app.js
```

by isaacs - http://github.com/isaacs/node-supervisor

Discussion courtesy of: Anup Bishnoi

I am working on making a rather tiny node "thing" that is able to load/unload modules at-will (so, i.e. you could be able to restart part of your application without bringing the whole app down). I am incorporating a (very stupid) dependency management, so that if you want to stop a module, all the modules that depends on that will be stopped too.

So far so good, but then I stumbled into the issue of how to reload a module.

Apparently, one could just remove the module from the "require" cache and have the job done. Since I'm not keen to change directly the node source code, I came up with a *very* hacky-hack that is: search in the stack trace the last call to the "require" function, grab a reference to it's "cache" field and..well, delete the reference to the node:

```
var args = arguments
while(!args['1'] || !args['1'].cache) {
args = args.callee.caller.arguments
}
var cache = args['1'].cache
util.log('remove cache ' + moduleFullpathAndExt)
delete( cache[ moduleFullpathAndExt ] )
```

Even easier, actually:

```
var deleteCache = function(moduleFullpathAndExt) {
delete( require.cache[ moduleFullpathAndExt ] )
}
```

Apparently, this works just fine. I have absolutely no idea of what that arguments["1"] means, but it's doing its job. I believe that the node guys will implement a reload facility someday, so I guess that for now this solution is acceptable too. (btw. my "thing" will be here: https://github.com/cheng81/wirez , go there in a couple of weeks and you should see what I'm talking about)

Discussion courtesy of: cheng81

i found a simple way:

```
delete require.cache['/home/shimin/test2.js']
```

Discussion courtesy of: Inshua

nodemon came up first in a google search, and it seems to do the trick:

```
npm install nodemon -g
cd whatever_dir_holds_my_app
nodemon app.js
```

Discussion courtesy of: JnBrymn

If somebody still comes to this question and wants to solve it using only the standard modules I made a simple example:

```
var process = require('process');
var cp = require('child_process');
var fs = require('fs');

var server = cp.fork('server.js');
console.log('Server started');

fs.watchFile('server.js', function (event, filename) {
server.kill();
console.log('Server stopped');
server = cp.fork('server.js');
console.log('Server started');
```

```
});

process.on('SIGINT', function () {
server.kill();
fs.unwatchFile('server.js');
process.exit();
});
```

This example is only for one file (server.js), but can be adapted to multiple files using an array of files, a for loop to get all file names, or by watching a directory:

```
fs.watch('./', function (event, filename) { // sub directory changes are not seen
console.log(`restart server`);
server.kill();
server = cp.fork('server.js');
})
```

This code was made for Node.js 0.8 API, it is not adapted for some specific needs but will work in some simple apps.

UPDATE: This functional is implemented in my module simpleR, GitHub repo

Discussion courtesy of: micnic

Use this:

```
function reload_config(file) {
if (!(this instanceof reload_config))
return new reload_config(file);
var self = this;

self.path = path.resolve(file);

fs.watchFile(file, function(curr, prev) {
delete require.cache[self.path];
_.extend(self, require(file));
});

_.extend(self, require(file));
}
```

All you have to do now is:

```
var config = reload_config("./config");
```

And config will automatically get reloaded :)

Discussion courtesy of: offlinehacker

loaddir is my solution for quick loading of a directory, recursively.

can return

```
{ 'path/to/file': 'fileContents...' } or { path: { to: { file: 'fileContents'}
} }
```

It has callback which will be called when the file is changed.

It handles situations where files are large enough that watch gets called before they're done writing.

I've been using it in projects for a year or so, and just recently added promises to it.

Help me battle test it!

https://github.com/danschumann/loaddir

Discussion courtesy of: Funkodebat

You can use auto-reload to reload the module without shutdown the server.

install

```
npm install auto-reload
```

example

data.json

```
{ "name" : "Alan" }
```

test.js

```
var fs = require('fs');
var reload = require('auto-reload');
var data = reload('./data', 3000); // reload every 3 secs

// print data every sec
setInterval(function() {
console.log(data);
}, 1000);

// update data.json every 3 secs
setInterval(function() {
var data = '{ "name":"' + Math.random() + '" }';
fs.writeFile('./data.json', data);
}, 3000);
```

Result:

```
{ name: 'Alan' }
{ name: 'Alan' }
{ name: 'Alan' }
{ name: 'Alan' }
{ name: 'Alan' }
{ name: '0.8272748321760446' }
{ name: '0.8272748321760446' }
{ name: '0.8272748321760446' }
{ name: '0.07935990858823061' }
{ name: '0.07935990858823061' }
{ name: '0.07935990858823061' }
{ name: '0.20851597073487937' }
{ name: '0.20851597073487937' }
{ name: '0.20851597073487937' }
```

Discussion courtesy of: Lellansin

another simple solution is to **use fs.readFile instead of using require** you can save a text file contaning a json object, and create a interval on the server to reload this object.

pros:

- no need to use external libs
- relevant for production (reloading config file on change)
- easy to implement

cons:

- you can't reload a module - just a json containing key-value data

Discussion courtesy of: Imri

For people using Vagrant and PHPStorm, file watcher is a faster approach

- disable immediate sync of the files so you run the command only on save then create a scope for the *.js files and working directories and add this command

  vagrant ssh -c "/var/www/gadelkareem.com/forever.sh restart"

where forever.sh is like

```
#!/bin/bash

cd /var/www/gadelkareem.com/ && forever $1 -l /var/www/gadelkareem.com/.tmp/log/forever.log -a app.js
```

Discussion courtesy of: gadelkareem

I recently came to this question because the usual suspects were not working with linked packages. If you're like me and are taking advantage of npm link during development to effectively work on a project that is made up of many packages, it's important that changes that occur in dependencies trigger a reload as well.

After having tried node-mon and pm2, even following their instructions for additionally watching the node_modules folder, they still did not pick up changes. Although there are some custom solutions in the answers here, for something like this, a separate package is cleaner. I came across node-dev today and it works perfectly without any options or configuration.

From the Readme:

In contrast to tools like supervisor or nodemon it doesn't scan the filesystem for files to be watched. Instead it hooks into Node's require() function to watch only the files that have been actually required.

Discussion courtesy of: Aaron Storck

yet another solution for this problem is using forever

Another useful capability of Forever is that it can optionally restart your application when any source files have changed. This frees you from having to manually restart each time you add a feature or fix a bug. To start Forever in this mode, use the -w flag:

```
forever -w start server.js
```

Discussion courtesy of: Teoman shipahi

Not necessary to use *nodemon* or other tools like that. Just use capabilities of

your IDE.

Probably best one is *IntelliJ WebStorm* with hot reload feature (automatic server and browser reload) for *node.js*.

Discussion courtesy of: lukyer

node-dev works great. npm install node-dev

It even gives a desktop notification when the server is reloaded and will give success or errors on the message.

start your app on command line with:

node-dev app.js

Discussion courtesy of: l3o

Best server-side javascript servers

Problem

I've been wondering to try out server-side javascript for a while. And I'm finding a good amount of servers, like: Node.js Rhino SpiderMonkey among others.

Could anyone with experience on server-side javascript, tell me which are the best engines? and why? I like the Node.js because it's based on Google's V8 engine. And seems easy to use. But some feedback on what you would choose would be great.

Edit:

Some benchmarks for Node.

I'm thinking on going with this one but feedback is still welcome.

Thanks

Problem courtesy of: fmsf

Solution

I think each solution has its own advantages/disadvantages

here a list of SSJS solutions:

- **Aptana Jaxer**: sadly abandoned
- **Sitepoint Persevere**: based on rhino - include JSDB, supports JSON Query - by Kris Zyp, the author of JSON Schema
- **RingoJS**: based on rhino - ex Helma NG successor of Helma which existed from long time ago - multi-thread - nice community - great actor on CommonJS
- **Narwhal**: can work on either spidermonkey, V8, or webkit JavaScriptCore - another great actor on CommonJS - defined the JSGI API in the Jack Server
- **Joyent NodeJS**: based on V8 (fast) - all is running in a single thread - all the code must be written using callbacks - lot of modules available via npm (Node Package Manager)
- **4D Wakanda**: based on Webkit JavaScriptCore aka SFX or Nitro (which has been faster than V8 and could be faster again) - include an NoSQL Object oriented JavaScript datastore with a native REST API - multi-threaded - provides a studio with a debugger, a Model Designer, and a GUI Designer - provides a Framework with Widgets directly bindable to the datastore and between each others
- **APE project**: based on spidermonkey - a Push engine to synchronize in live data between many visitor browsers
- **10gen MongoDB**: A NoSQL Document store allowing code in either Erlang & JavaScript (using spidermonkey)
- **Apache CouchDB**: Another NoSQL Document store also allowing code in either Erlang & JavaScript (using spidermonkey)

See a Server-Side JavaScript presentation with some history, benchmarks, & descriptions

Solution courtesy of: Alexandre Morgaut

Discussion

The word *server* is badly used. Those are different JavaScript implementations.

Personally the only thing I'm doubtful about these implementations is about the low amount of system libraries. Most of them come with standard stuff (sockets, filesystem etc..) but lack support for Threads and concurrency. That's why I usually use *Rhino*, unless I care for speed.

Discussion courtesy of: Luca Matteis

It might help to describe what problem you are trying to solve. There are a lot of "server-side" JS implementations that fill various needs. An there also several "server" platforms built around these implementations. I wrote one implementation on top of Rhino and Java Servlets, called Myna. Myna is for web applications, and I detailed some of its advantages in another question.

Are you trying to deploy web applications? Are you trying to script existing code? Does your code need to start quickly from a commandline, or is it going to be implemented as a daemon?

There are currently no clear winners in this space that do all of these things well, but there are good choices for specific needs.

Discussion courtesy of: Mark Porter

How would MVC-like code work in Node.js?

Problem

I'm starting to get my head around node.js, and I'm trying to figure out how I would do normal MVC stuff. For example, here's a Django view that pulls two sets of records from the database, and sends them to be rendered in a template.

```
def view(request):
things1 = ThingsOne.objects.all()
things2 = ThingsTwo.objects.all()
render_to_response('template.html, {'things1': things1, 'things2': things2})
```

What might a similar node.js function look like?

Problem courtesy of: Chip Tol

Solution

http://boldr.net/mvc-stack-node-js-ejsgi-scylla-mustache is a great little article with a full github example of a MVC pattern using dirfferent Node modules. It also lists alternate modules currently available. It answered this question for me better than http://howtonode.org/ which has some good tuts but I couldn't find anything on MVC there.

Solution courtesy of: Denis Hoctor

Discussion

The easiest way to do this is with expressjs, which is an MVC framework for Node. Node is just what it says, evented I/O for the web.

The example on the http://expressjs.com should help with the basics but to answer your question directly.

```
var express = require('express');

var app = express.createServer();

app.get('/whatever', function(req, res) {

Things1.objects.getAll(function(things1) {
Things2.objects.getAll(function(things2) {
var options = { locals: { things1: things1, things2: things2 }};
res.render('thingstemplate.ejs', options); // or thingstemplate.jade or whatever
});
});
});

app.listen('80', ''); // port and optional hostname to bind
```

Discussion courtesy of: Timothy Meade

TowerJS is a popular MVC framework based on

- MongoDB (database)
- Redis (background jobs)
- CoffeeScript
- Stylus
- Jasmine (tests)
- jQuery

Site http://towerjs.org/

Source https://github.com/viatropos/tower

Discussion courtesy of: 250R

RailwayJS is a MVC framework, written in JavaScript based on ExpressJS and runs over nodeJS platform. It is inspired by Ruby on Rails framework. You can read about MVC architecture of RailwayJS here: http://jsmantras.com/blog/RailwayJS-Routing

Discussion courtesy of: JS Mantras

This content originated from StackOverFlow and has been re-organized into the above recipe.

Is there a solution that lets Node.js act as an HTTP reverse proxy?

Problem

Our company has a project that right now uses nginx as a reverse proxy for serving static content and supporting comet connections. We use long polling connections to get rid of constant refresh requests and let users get updates immediately.

Now, I know there is a lot of code already written for Node.js, but is there a solution that lets Node.js act as a reverse proxy for serving static content as nginx does? Or maybe there is a framework that allows to quickly develop such a layer using Node.js?

Problem courtesy of: Igor Zinov'yev

Solution

node-http-proxy sounds like what you want

```
var sys = require('sys'),
http = require('http'),
httpProxy = require('http-proxy').httpProxy;

http.createServer(function (req, res){
var proxy = new httpProxy;
proxy.init(req, res);
proxy.proxyRequest('localhost', '9000', req, res);
}).listen(8000);

http.createServer(function (req, res){
res.writeHead(200, {'Content-Type': 'text/plain'});
res.write('request successfully proxied!' + '\n' + JSON.stringify(req.headers,
true, 2));
res.end();
}).listen(9000);</code></pre>
```

Solution courtesy of: JimBastard

Discussion

dogproxy might be able to help you, if not as a full solution then possibly as the building blocks for one.

However, you might wish to reconsider keeping nginx for serving static content -- it is specifically designed and tuned for this particular task. You would be adding a lot of overhead in using node.js to serve static content - much like using PHP to serve static files.

Discussion courtesy of: Phillip B Oldham

Nodejs in-memory storage

Problem

How do i store data to be used for all the clients in my server? (like the messages of a chat)

Problem courtesy of: user259427

Solution

The server that node.js allows you to build, is an application server, which means that state is preserved, between request, on the server side. The following snippet demonstrates this:

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

var number = 0;

http.createServer(function (req, res) {
console.log(req.method, req.url);

res.writeHead(200, {'Content-Type': 'text/html'});
res.write('<h1>Number is: ' + number + '</h1>');
res.end();

number++;

}).listen(8000);

sys.puts('Server running at http://127.0.0.1:8000/');
```

Solution courtesy of: Ionuț G. Stan

Discussion

If you want more features have a look at redis-node-client

Discussion courtesy of: Bharani

Or use a native node storage mechanism (written in node.js)

http://github.com/felixge/node-dirty

Discussion courtesy of: christkv

node-cache package is currently the best for key value store and it allows synchronous as well as async storage/retrieval/deletion of keys.

npm link

Discussion courtesy of: rnjai

I wrote Bx for this purpose; it gives you a simple in-memory cache with:

- Key-value storage
- Optional expiration on any stored data
- Support for schemas using JSON-schema

Although I'm plugging my own repository here, I can assure you it works well and it's been used in production at my own company, Onshape for over a year without issues. At the end of the day it's a pretty simple tool; not much to mess up here.

However, if you're storing data that's meant to be permanent you're going to want a database such as MongoDB (w/ Mongoose), MySQL, etc. rather than a cache like Bx or Redis.

Discussion courtesy of: tylucaskelley

Writing files in Node.js

Problem

I've been trying to find a way to write to a file when using Node.js, but with no success. How can I do that?

Problem courtesy of: Gjorgji

Solution

There are a lot of details in the filesystem API. The most common way (as far as I know) is:

```
var fs = require('fs');
fs.writeFile("/tmp/test", "Hey there!", function(err) {
if(err) {
return console.log(err);
}

console.log("The file was saved!");
});
```

Solution courtesy of: Brian McKenna

Discussion

You can of course make it a little more advanced. Non-blocking, writing bits and pieces, not writing the whole file at once:

```
var fs = require('fs');
var stream = fs.createWriteStream("my_file.txt");
stream.once('open', function(fd) {
stream.write("My first row\n");
stream.write("My second row\n");
stream.end();
});
```

Discussion courtesy of: Fredrik Andersson

Currently there are three ways to write a file:

1. fs.write(fd, buffer, offset, length, position, callback)

   You need to wait for the callback to ensure that the buffer is written to disk. It's not buffered.

2. fs.writeFile(filename, data, [encoding], callback)

   All data must be stored at the same time; you cannot perform sequential writes.

3. fs.createWriteStream(path, [options])

   Creates a WriteStream, which is convenient because you don't need to wait for a callback. But again, it's not buffered.

A WriteStream, as the name says, is a stream. A stream by definition is “a buffer” containing data which moves in one direction (source ► destination). But a writable stream is not necessarily “buffered”. A stream is “buffered” when you write n times, and at time n+1, the stream sends the buffer to the kernel (because it's full and needs to be flushed).

In other words: “A buffer” is the object. Whether or not it “is buffered” is a property of that object.

If you look at the code, the WriteStream inherits from a writable Stream object. If you pay attention, you’ll see how they flush the content; they don't have any buffering system.

If you write a string, it’s converted to a buffer, and then sent to the native layer and written to disk. When writing strings, they're not filling up any buffer. So, if you do:

```
write("a")
write("b")
write("c")
```

You're doing:

```
fs.write(new Buffer("a"))
fs.write(new Buffer("b"))
fs.write(new Buffer("c"))
```

That's *three* calls to the I/O layer. Although you're using "buffers", the data is not buffered. A buffered stream would do: fs.write(new Buffer ("abc")), one call to the I/O layer.

As of now, in Node.js v0.12 (stable version announced 02/06/2015) now supports two functions: cork() and uncork(). It seems that these functions will finally allow you to buffer/flush the write calls.

For example, in Java there are some classes that provide buffered streams (BufferedOutputStream, BufferedWriter...). If you write three bytes, these bytes will be stored in the buffer (memory) instead of doing an I/O call just for three bytes. When the buffer is full the content is flushed and saved to disk. This improves performance.

I'm not discovering anything, just remembering how a disk access should be done.

Discussion courtesy of: Gabriel Llamas

```
var path = 'public/uploads/file.txt',
buffer = new Buffer("some content\n");

fs.open(path, 'w', function(err, fd) {
if (err) {
throw 'error opening file: ' + err;
}

fs.write(fd, buffer, 0, buffer.length, null, function(err) {
if (err) throw 'error writing file: ' + err;
fs.close(fd, function() {
console.log('file written');
})
});
});
```

Discussion courtesy of: Mister P

Here we use w+ for read/write both actions and if the file path is not found the it would be created automatically.

```
fs.open(path, 'w+', function(err, data) {
if (err) {
console.log("ERROR !! " + err);
} else {
fs.write(data, 'content', 0, 'content length', null, function(err) {
if (err)
console.log("ERROR !! " + err);
fs.close(data, function() {
console.log('written success');
})
});
}
```

```
});
```

Content means what you have to write to the file and its length, 'content.length'.

Discussion courtesy of: Gunjan Patel

I liked *Index of ./articles/file-system*.

It worked for me.

See also *How do I write files in node.js?*.

```
fs = require('fs');
fs.writeFile('helloworld.txt', 'Hello World!', function (err) {
if (err)
return console.log(err);
console.log('Hello World > helloworld.txt');
});
```

Contents of helloworld.txt:

```
Hello World!
```

Discussion courtesy of: Sérgio

You can use library easy-file-manager

install first from npm npm install easy-file-manager

Sample to upload and remove files

```
var filemanager = require('easy-file-manager')
var path = "/public"
var filename = "test.jpg"
var data; // buffered image

filemanager.upload(path,filename,data,function(err){
if (err) console.log(err);
});

filemanager.remove(path,"aa,filename,function(isSuccess){
if (err) console.log(err);
});
```

Discussion courtesy of: Christoper

```
var fs = require('fs');
fs.writeFile(path + "\\message.txt", "Hello", function(err){
if (err) throw err;
console.log("success");
});
```

For example : read file and write to another file :

```
var fs = require('fs');
```

```
var path = process.cwd();
fs.readFile(path+"\\from.txt",function(err,data)
{
if(err)
console.log(err)
else
{
fs.writeFile(path+"\\to.text",function(erro){
if(erro)
console.log("error : "+erro);
else
console.log("success");
});
}
});
```

Discussion courtesy of: Masoud Siahkali

This content originated from StackOverFlow and has been re-organized into the above recipe.

NodeJS and HTTP Client - Are cookies supported?

Problem

NodeJS is a fantastic tool and blazing fast.

I'm wondering if HTTPClient supports **cookies** and if can be used in order to simulate very **basic browser behaviour**!

Help would be very much appreciated! =)

EDIT:

Found this: node-httpclient (seems useful!) ***not working!***

Problem courtesy of: RadiantHex

Solution

Just get cookies from Set-Cookie param in response headers and send them back with future requests. Should not be hard.

Solution courtesy of: Kuroki Kaze

Discussion

Short answer: no. And it's not so great.

I implemented this as part of npm so that I could download tarballs from github. Here's the code that does that: https://github.com/isaacs/npm/blob/master/lib/utils/fetch.js#L96-100

```
var cookie = get(response.headers, "Set-Cookie")
if (cookie) {
cookie = (cookie + "").split(";").shift()
set(opts.headers, "Cookie", cookie)
}
```

The file's got a lot of npm-specific stuff (log, set, etc.) but it should show you the general idea. Basically, I'm collecting the cookies so that I can send them back on the next request when I get redirected.

I've talked with Mikeal Rogers about adding this kind of functionality to his "request" util, complete with supporting a filesystem-backed cookiejar, but it's really pretty tricky. You have to keep track of which domains to send the cookies to, and so on.

This will likely never be included in node directly, for that reason. But watch for developments in userspace.

EDIT: This is now supported by default in Request.

Discussion courtesy of: isaacs

If you are looking to do cookies client side you can use https://github.com/mikeal/request

M.

Discussion courtesy of: Martin Cleaver

The code below demonstrates using cookie from server side, here's a demo API server that parse cookies from a http client and check the cookie hash:

```
var express = require("express"),
app = express(),
hbs = require('hbs'),
mongoose = require('mongoose'),
port = parseInt(process.env.PORT, 10) || 4568;

app.configure(function () {
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(express.static(__dirname + '/public_api'));
app.use(express.errorHandler({ dumpExceptions: true, showStack: true }));
});
app.get('/api', function (req, res) {
var cookies = {};
req.headers.cookie && req.headers.cookie.split(';').forEach(function( cookie )
```

```
{
var parts = cookie.split('=');
cookies[ parts[ 0 ].trim() ] = ( parts[ 1 ] || '' ).trim();
});
if (!cookies['testcookie']) {
console.log('First request');
res.cookie('testcookie','testvaluecookie',{ maxAge: 900000, httpOnly: true });
res.end('FirstRequest');
} else {
console.log(cookies['testcookie']);
res.end(cookies['testcookie']);
}
});

app.listen(port);
```

On the client side, just make a normal request to the server api above, i'm using request module, it by default transfers cookie for each request.

```
request(options, function(err, response, body) {
console.log(util.inspect(response.headers));
res.render("index.html", {layout: false,user: {
username: req.session.user + body
}});
});
```

Discussion courtesy of: revskill

Zombie.js is another choice if you want browser-like behaviour. It "maintains state across requests: history, cookies, HTML5 local and session stroage, etc.". More info on zombie's cookie api: http://zombie.labnotes.org/API

There is also PhantomJS and PhantomJS-based frameworks, like CasperJS.

Discussion courtesy of: dgo.a

A feature-complete solution for cookies

The self-made solutions proposed in the other answers here don't cover a lot of special cases, can easily break and lack a lot of standard features, such as persistence.

As mentioned by isaacs, the request module now has true cookie support. They provide examples with cookies on their Github page. The examples explain how to enable cookie support by adding a "tough-cookie" cookie jar to your request.

NOTE: A cookie jar contains and helps you manage your cookies.

To quote their Readme (as of April 2015):

Cookies are disabled by default (else, they would be used in subsequent requests). To enable cookies, set jar to true (either in defaults or options) and install tough-cookie.

The cookie management is provided through the tough-cookie module. It is a stable, rather feature-complete cookie management tool that implements the "HTTP State Management Mechanism" - RFC 6265. It even offers a variety of

options to persist (store) cookies, using a cookie store.

Discussion courtesy of: Domi

This content originated from StackOverFlow and has been re-organized into the above recipe.

How do you get a list of the names of all files present in a directory in Node.js?

Problem

I'm trying to get a list of the names of all the files present in a directory using Node.js. I want output that is an array of filenames. How can I do this?

Problem courtesy of: resopollution

Solution

You can use the fs.readdir or fs.readdirSync methods.

fs.readdir

```
const testFolder = './tests/';
const fs = require('fs');

fs.readdir(testFolder, (err, files) => {
files.forEach(file => {
console.log(file);
});
})
```

fs.readdirSync

```
const testFolder = './tests/';
const fs = require('fs');

fs.readdirSync(testFolder).forEach(file => {
console.log(file);
})
```

The difference between the two methods, is that the first one is asynchronous, so you have to provide a callback function that will be executed when the read process ends.

The second is synchronous, it will returns the file name array, but it will stop any further execution of your code until the read process ends.

Solution courtesy of: CMS

Discussion

The answer above does not perform a recursive search into the directory though. Here's what I did for a recursive search (using node-walk: npm install walk)

```
var walk = require('walk');
var files = [];

// Walker options
var walker = walk.walk('./test', { followLinks: false });

walker.on('file', function(root, stat, next) {
// Add this file to the list of files
files.push(root + '/' + stat.name);
next();
});

walker.on('end', function() {
console.log(files);
});
```

Discussion courtesy of: Ruben Tan

Get files in all subdirs

```
function getFiles (dir, files_){
files_ = files_ || [];
var files = fs.readdirSync(dir);
for (var i in files){
var name = dir + '/' + files[i];
if (fs.statSync(name).isDirectory()){
getFiles(name, files_);
} else {
files_.push(name);
}
}
return files_;
}

console.log(getFiles('path/to/dir'))
```

Discussion courtesy of: Tito100

Here's an asynchronous recursive version.

```
function ( path, callback){
// the callback gets ( err, files) where files is an array of file names
if( typeof callback !== 'function' ) return
var
result = []
, files = [ path.replace( /\/\s*$/, '' ) ]
function traverseFiles (){
if( files.length ) {
var name = files.shift()
fs.stat(name, function( err, stats){
```

```
if( err ){
if( err.errno == 34 ) traverseFiles()
// in case there's broken symbolic links or a bad path
// skip file instead of sending error
else callback(err)
}
else if ( stats.isDirectory() ) fs.readdir( name, function( err, files2 ){
if( err ) callback(err)
else {
files = files2
.map( function( file ){ return name + '/' + file } )
.concat( files )
traverseFiles()
}
})
else{
result.push(name)
traverseFiles()
}
})
}
else callback( null, result )
}
traverseFiles()
}
```

Discussion courtesy of: Oggy Transfluxitor Jones

```
function getFilesRecursiveSync(dir, fileList, optionalFilterFunction) {
if (!fileList) {
grunt.log.error("Variable 'fileList' is undefined or NULL.");
return;
}
var files = fs.readdirSync(dir);
for (var i in files) {
if (!files.hasOwnProperty(i)) continue;
var name = dir + '/' + files[i];
if (fs.statSync(name).isDirectory()) {
getFilesRecursiveSync(name, fileList, optionalFilterFunction);
} else {
if (optionalFilterFunction && optionalFilterFunction(name) !== true)
continue;
fileList.push(name);
}
}
}
```

Discussion courtesy of: Francois

Just a heads up: if you're planning to perform operations on each file in a directory, try vinyl-fs (which is used by gulp, the streaming build system).

Discussion courtesy of: XåpplI'-I0llwlg'I -

IMO the most convinient way to do such tasks is to use a glob tool. Here's a glob package for node.js. Install with

```
npm install glob
```

Then use wild card to match filenames (example taken from package's website)

```
var glob = require("glob")

// options is optional
glob("**/*.js", options, function (er, files) {
// files is an array of filenames.
// If the `nonull` option is set, and nothing
// was found, then files is ["**/*.js"]
// er is an error object or null.
})
```

Discussion courtesy of: KFL

Here's a simple solution using only the native fs and path modules:

```
// sync version
function walkSync(currentDirPath, callback) {
var fs = require('fs'),
path = require('path');
fs.readdirSync(currentDirPath).forEach(function (name) {
var filePath = path.join(currentDirPath, name);
var stat = fs.statSync(filePath);
if (stat.isFile()) {
callback(filePath, stat);
} else if (stat.isDirectory()) {
walkSync(filePath, callback);
}
});
}
```

or async version (uses fs.readdir instead):

```
// async version with basic error handling
function walk(currentDirPath, callback) {
var fs = require('fs'),
path = require('path');
fs.readdir(currentDirPath, function (err, files) {
if (err) {
throw new Error(err);
}
files.forEach(function (name) {
var filePath = path.join(currentDirPath, name);
var stat = fs.statSync(filePath);
if (stat.isFile()) {
callback(filePath, stat);
} else if (stat.isDirectory()) {
walk(filePath, callback);
}
});
});
}
```

Then you just call (for sync version):

```
walkSync('path/to/root/dir', function(filePath, stat) {
```

```
// do something with "filePath"...
});
```

or async version:

```
walk('path/to/root/dir', function(filePath, stat) {
// do something with "filePath"...
});
```

The difference is in how node blocks while performing the IO. Given that the API above is the same, you could just use the async version to ensure maximum performance.

However there is one advantage to using the synchronous version. It is easier to execute some code as soon as the walk is done, as in the next statement after the walk. With the async version, you would need some extra way of knowing when you are done. Perhaps creating a map of all paths first, then enumerating them. For simple build/util scripts (vs high performance web servers) you could use the sync version without causing any damage.

Discussion courtesy of: Ali

Dependencies.

```
var fs = require('fs');
var path = require('path');
```

Definition.

```
// String -> [String]
function fileList(dir) {
return fs.readdirSync(dir).reduce(function(list, file) {
var name = path.join(dir, file);
var isDir = fs.statSync(name).isDirectory();
return list.concat(isDir ? fileList(name) : [name]);
}, []);
}
```

Usage.

```
var DIR = '/usr/local/bin';

// 1. List all files in DIR
fileList(DIR);
// => ['/usr/local/bin/babel', '/usr/local/bin/bower', ...]

// 2. List all file names in DIR
fileList(DIR).map((file) => file.split(path.sep).slice(-1)[0]);
// => ['babel', 'bower', ...]
```

Please note that fileList is way too optimistic. For anything serious, add some error handling.

Discussion courtesy of: Hunan Rostomyan

Took the general approach of @Hunan-Rostomyan, made it a litle more concise and added excludeDirs argument. It'd be trivial to extend with includeDirs, just

follow same pattern:

```
import * as fs from 'fs';
import * as path from 'path';

function fileList(dir, excludeDirs?) {
return fs.readdirSync(dir).reduce(function (list, file) {
const name = path.join(dir, file);
if (fs.statSync(name).isDirectory()) {
if (excludeDirs && excludeDirs.length) {
excludeDirs = excludeDirs.map(d => path.normalize(d));
const idx = name.indexOf(path.sep);
const directory = name.slice(0, idx === -1 ? name.length : idx);
if (excludeDirs.indexOf(directory) !== -1)
return list;
}
return list.concat(fileList(name, excludeDirs));
}
return list.concat([name]);
}, []);
}
```

Example usage:

```
console.log(fileList('.', ['node_modules', 'typings', 'bower_components']));
```

Using Promises with ES7

Asynchronous use with mz/fs

The mz module provides promisified versions of the core node library. Using them is simple. First install the library...

```
npm install mz
```

Then...

```
const fs = require('mz/fs');
fs.readdir('./myDir').then(listing => console.log(listing))
.catch(err => console.error(err));
```

Alternatively you can write them in asynchronous functions in ES7:

```
async function myReaddir () {
try {
const file = await fs.readdir('./myDir/');
}
catch (err) { console.error( err ) }
};
```

Update for recursive listing

Some of the users have specified a desire to see a recursive listing (though not in the question)... Use fs-promise. It's a thin wrapper around mz.

```
npm install fs-promise;
```

then...

```
const fs = require('fs-promise');
fs.walk('./myDir').then(
listing => listing.forEach(file => console.log(file.path))
).catch(err => console.error(err));
```

Discussion courtesy of: Evan Carroll

You don't say you want to do it recursively so I assume you only need direct children of the directory.

Sample code:

```
const fs = require('fs');
const path = require('path');

fs.readdirSync('your-directory-path')
.filter((file) => fs.lstatSync(path.join(folder, file)).isFile());
```

Discussion courtesy of: Tyler Long

Load fs:

```
const fs = require('fs');
```

Read files *async*:

```
fs.readdir('./dir', function (err, files) {
// "files" is an Array with files names
});
```

Read files *sync*:

```
var files = fs.readdirSync('./dir');
```

Discussion courtesy of: Eduardo Cuomo

Uncatchable errors in node.js

Problem

So I'm trying to write a simple TCP socket server that broadcasts information to all connected clients. So when a user connects, they get added to the list of clients, and when the stream emits the close event, they get removed from the client list.

This works well, except that sometimes I'm sending a message just as a user disconnects.

I've tried wrapping stream.write() in a try/catch block, but no luck. It seems like the error is uncatchable.

Problem courtesy of: Peter Burns

Solution

The solution is to add a listener for the stream's 'error' event. This might seem counter-intuitive at first, but the justification for it is sound.

stream.write() sends data asynchronously. By the time that node has realized that writing to the socket has raised an error your code has moved on, past the call to stream.write, so there's no way for it to raise the error there.

Instead, what node does in this situation is emit an 'error' event from the stream, and EventEmitter is coded such that if there are no listeners for an 'error' event, the error is raised as a toplevel exception, and the process ends.

Solution courtesy of: Peter Burns

Discussion

Peter is quite right,

and there is also another way, you can also make a catch all error handler with

```
process.on('uncaughtException',function(error){
// process error
})
```

this will catch everything which is thrown...

it's usually better to do this peter's way, if possible, however if you where writing, say, a test framework, it may be a good idea to use process.on('uncaughtException',...

here is a gist which covers (i think) all the different aways of handling errors in nodejs http://gist.github.com/636290

Discussion courtesy of: dominic

I had the same problem with the time server example from here My clients get killed and the time server then tries to write to closed socket.

Setting an error handler does not work as the error event only fires on reception. The time server does no receiving, (see stream event documentation).

My solution is to set a handler on the stream close event.

```
stream.on('close', function() {
subscribers.remove(stream);
stream.end();
console.log('Subscriber CLOSE: ' + subscribers.length + " total.\n");
});
```

Discussion courtesy of: Heater

How can I edit on my server files without restarting nodejs when i want to see the changes?

Problem

I'm trying to setup my own nodejs server, but I'm having a problem. I can't figure out how to see changes to my application without restarting it. Is there a way to edit the application and see changes live with node.js?

Problem courtesy of: Robert Hurst

Solution

Check out Node-Supervisor. You can give it a collection of files to watch for changes, and it restarts your server if any of them change. It also restarts it if it crashes for some other reason.

"Hot-swapping" code is not enabled in NodeJS because it is so easy to accidentally end up with memory leaks or multiple copies of objects that aren't being garbage collected. Node is about making your programs accidentally fast, not accidentally leaky.

Solution courtesy of: isaacs

Discussion

Nodules is a module loader for Node that handles auto-reloading of modules *without* restarting the server (since that is what you were asking about):

http://github.com/kriszyp/nodules

Nodules does intelligent dependency tracking so the appropriate module factories are re-executed to preserve correct references when modules are reloaded without requiring a full restart.

Discussion courtesy of: Kris Zyp

Use this: https://github.com/remy/nodemon

Just run your app like this: nodemon yourApp.js

Discussion courtesy of: Looloooii

What's “Live Coding”?

In essence, it's a way to alter the program while it runs, without restarting it. The goal, however, is to end up with a program that works properly when we (re)start it. To be useful, it helps to have an editor that can be customized to send code to the server.

Take a look: http://lisperator.net/blog/livenode-live-code-your-nodejs-application/

Discussion courtesy of: John Smith Optional

There should be some emphasis on what's happening, instead of just shotgunning modules at the OP. Also, we don't know that the files he is editing are all JS modules or that they are all using the "require" call. Take the following scenarios with a grain of salt, they are only meant to describe what is happening so you know how to work with it.

1. Your code has already been loaded and the server is running with it

    - **SOLUTION** You need to have a way to tell the server what code has changed so that it can reload it. You could have an endpoint set up to receive a signal, a command on the command line or a request through tcp/http that will tell it what file changed and the endpoint will reload it.

      ```
      //using Express
      var fs = require('fs');
      app.get('reload/:file', function (req, res) {
      fs.readfile(req.params.file, function (err, buffer) {
      //do stuff...
      });
      ```

```
});
```

2. Your code may have "require" calls in it which loads **and caches** modules

   - **SOLUTION** since these modules are cached by require, following the previous solution, you would need a line in your endpoint to delete that reference

     ```
     var moduleName = req.params.file;
     delete require.cache[moduleName];
     require('./' + moduleName);
     ```

There's a lot of caveats to get into behind all of this, but hopefully you have a better idea of what's happening and why.

Discussion courtesy of: Sparkida

I think node-inspector is your best bet.

Similar to how you can Live Edit Client side JS code in Chrome Dev tools, this utilizes the Chrome (Blink) Dev Tools Interface to provide live code editing.

https://github.com/node-inspector/node-inspector/wiki/LiveEdit

Discussion courtesy of: Gaurav Ramanan

if you would like to reload a module without restarting the node process, you can do this by the help of the **watchFile** function in **fs** module and **cache clearing** feature of **require**:

Lets say you loaded a module with a simple require:

```
var my_module = require('./my_module');
```

In order to watch that file and reload when updated add the following to a convenient place in your code.

```
fs.watchFile(require.resolve('./my_module'), function () {
console.log("Module changed, reloading...");
delete require.cache[require.resolve('./my_module')]
my_module = require('./my_module');
});
```

If your module is required in multiple files this operation will not affect other assignments, so keeping module in a global variable and using it where it is needed from global rather than requiring several times is an option. So the code above will be like this:

```
global.my_module = require ('./my_module');
//..
fs.watchFile(require.resolve('./my_module'), function () {
console.log("Module changed, reloading...");
delete require.cache[require.resolve('./my_module')]
global.my_module = require('./my_module');
});
```

Discussion courtesy of: Tahsin Turkoz

You can also use the tool PM2. Which is a advanced production process tool for node js. http://pm2.keymetrics.io/

Discussion courtesy of: nidhin

nodejs and database communication - how?

Problem

I've heard much good about nodejs and writting client-server application with it. But I can't get, for example, when developing IM client-server application, how nodejs server script is supposed to talk to database server to actually store it's data? Or may be I miss something and nodejs server scripts are not supposed to do that? If so, please, push me to correct direction.

I've noticed DBSLayer http://code.nytimes.com/projects/dbslayer/wiki, but it looks like it's still in beta.

Problem courtesy of: Vladislav Rastrusny

Solution

You need to grab a module that handles the communication to the database you want. See here for a list of modules for node.js. Popular databases that work well with node.js are MongoDB, CouchDB and Redis.

Solution courtesy of: stagas

Discussion

As stagas says, you can use a module that handles communication if you want to use an external database.

If you want an internal (=embedded) database, you can use one written in javascript you can require like any other module such as NeDB or nStore. They are easier to use and useful if your webapp doesn't need to handle a lot of concurrent connections (e.g. a tool you make for yourself or a small team), or if you write a desktop app using Node Webkit

Discussion courtesy of: Louis Chatriot

Is there a Thrift or Cassandra client for Node.js/JavaScript

Problem

I would like to start using Cassandra with a node.js deployment, but I can't find a Thrift or Cassandra client for Node.js and/or JavaScript.

Is there one?

Is there a simple means of generating Thrift connections?

Update: The short answer to this question turns out to be no, there is no JS client for Thrift that is compatible with Cassandra.

Further Update: The next release of Cassandra (0.8 at time of writing) is going to have support for an Avro API. There is already node.js module for Avro support.

Problem courtesy of: Toby Hede

Solution

Someone made one now: https://github.com/wadey/node-thrift

Update: Rackspace released a node cassandra api: http://code.google.com/a/apache-extras.org/p/cassandra-node/

Update: They moved it to github: https://github.com/racker/node-cassandra-client

Update: There is a CQL driver now too: https://github.com/simplereach/helenus

Update: There is a CQL driver, that uses the Cassandra native protocol https://github.com/jorgebay/node-cassandra-cql

Update: DataStax released a CQL driver for Cassandra using the native protocol: https://github.com/datastax/nodejs-driver

Solution courtesy of: Zanson

Discussion

https://issues.apache.org/jira/browse/THRIFT-550

edit: take a look at https://github.com/wadey/node-thrift

Discussion courtesy of: Schildmeijer

Zanson already mentioned that Rackspace released the Cassandra API for Node.js, but it's worth noting that their Google Code page isn't their primary base of operations. The github page is where you can stay the most up to date:

https://github.com/racker/node-cassandra-client

Discussion courtesy of: netpoetica

The official Datastax driver is now node-cassandra-cql rebranded to nodejs-driver:

https://github.com/datastax/nodejs-driver

It uses CQL3.

Discussion courtesy of: srlm

Could someone give me an example of node.js application

Problem

I'm trying to understand the differences between some of the newer web programming frameworks that now exists, namely Node.js, Rails, and Sinatra.

Could someone give me an example of applications that would work best on each of the frameworks?

That is to say, what is an application that would be best suited for Node.js as opposed to Rails or Sinatra and what is an application that is best suited for Rails as opposed to Node.js and Sinatra etc.....

Problem courtesy of: TheDelChop

Solution

Sinatra and Rails are both web frameworks. They provide common web development abstractions, such as routing, templating, file serving, etc.

node.js is very different. At it's core, node.js is a combination of V8 and event libraries, along with an event-oriented standard library. node.js is better compared to EventMachine for Ruby.

For example, here's an event-based HTTP server, using EventMachine:

```
require 'eventmachine'
require 'evma_httpserver'

class MyHttpServer < EM::Connection
include EM::HttpServer

def post_init
super
no_environment_strings
end

def process_http_request
response = EM::DelegatedHttpResponse.new(self)
response.status = 200
response.content_type 'text/plain'
response.content = 'Hello world'
response.send_response
end
end

EM.run{
EM.start_server '0.0.0.0', 8080, MyHttpServer
}
```

And here's a node.js example:

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

http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello world');
}).listen(8000);
```

The benefit of this approach is that the server doesn't block on each request (they can be processed in parallel)!

node.js has its whole standard library built around the concept of events meaning that it's much better suited to any problem that is I/O bound. A good example would be a chat application.

Sinatra and Rails are both very refined, stable and popular web frameworks. node.js has a few web frameworks but none of them are at the quality of either of those at the moment.

Out of the choices, if I needed a more stable web application, I'd go for

either Sinatra or Rails. If I needed something more highly scalable and/or diverse, I'd go for node.js

Solution courtesy of: Brian McKenna

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

Node.js for lua?

Problem

I've been playing around with node.js (nodejs) for the past few day and it is fantastic. As far as I can tell, lua doesn't have a similar integration of libev and libio which let's one avoid almost any blocking calls and interact with the network and the filesystem in an asynchronous manner.

I'm slowly porting my java implementation to nodejs, but I'm shocked that luajit is much faster than v8 JavaScript AND uses far less memory!

I imagine writing my server in such an environment (very fast and responsive, very low memory usage, very expressive) will improve my project immensly.

Being new to lua, I'm just not sure if such a thing exists. I'll appreciate any pointers.

Thanks

Problem courtesy of: Shahbaz

Solution

See lualibevent and lua-ev and also Lua Gem #27

Solution courtesy of: Doug Currie

Discussion

You can get node.js style non-blocking IO with lua-handlers.

It even has an async. HTTP Client, which makes it really easy to start parallel HTTP requests. See the test_http_client.lua file as a example of the HTTP client interface.

Discussion courtesy of: Neopallium

Looks like the following is exactly what I was looking for: LuaNode https://github.com/ignacio/LuaNode

Discussion courtesy of: Shahbaz

if i understood the question right, take a look at http://openresty.com/

Discussion courtesy of: erbo

A recent corresponding project is **Luvit** "(Lua + libUV + jIT = pure awesomesauce)".

From the announcement:

this is basically luajit2 + libuv (the event loop library behind nodejs). It compiles as a single executable just like nodejs and can run .lua files. What makes it different from the stock luajit distribution is it has several built-in modules added and some slightly different semantics.

Notice that we're not running as a CGI script to apache or anything like that. The lua script *is* the http server. You get your callback called every time an http request is made to the server.

Discussion courtesy of: Clement J.

luvit aims to be to Lua exactly what Node.js is to Javascript. Definitely a promising project.

Discussion courtesy of: Jared Norman

You might want to take a look at Luvit or a gander at the Lua Github site. I think it takes the approach of implementing Node.js functionality *right inside Lua*. You write Lua code on the client side and on the server side. Here is a description of Luvit approach to doing Node.js functionality in Lua.

Discussion courtesy of: JohnnySoftware

You might also have a look at luv:

https://github.com/richardhundt/luv

from the lua mailing list:

How does luv relate to Luvit - LuaJIT + libuv (Node.js:s/JavaScript/Lua/)?

It doesn't really. Luvit borrows heavily from node.js's architecture (reactor callbacks, etc.), links statically against luajit, provides it's own module system and executable. Luv is just a Lua module which binds to libuv. The key difference is that Luv is more like an m-n threading engine combining coroutines and OS threads while using the libuv event loop under the hood.

So other than the fact that they both bind to libuv, they don't have much in common.

Discussion courtesy of: polypus74

You should also check out Lapis. It's a very lightweight and fast framework for OpenResty: http://leafo.net/lapis/

I've really been enjoying it and predict it will have a bright future!

As you would expect with anything built to leverage OpenResty, it's benchmarks are insanely good: https://www.techempower.com/benchmarks/#section=data-r12&hw=peak&test=query

The author of Lapis also wrote a CoffeeScript-like language for Lua called MoonScript which is quite nice: http://moonscript.org/

Discussion courtesy of: Alex Petty

Node.js or Erlang

Problem

I really like these tools when it comes to the concurrency level it can handle.

Erlang looks like much more stable solution but requires much more learning and a lot of diving into functional language paradigm. And it looks like Erlang makes it much better when it comes to multi cores CPUs(fix me if I'm wrong).

But which should I choose? Which one is better in the short/long term perspective?

My goal is to learn a tool which makes scaling my web projects under high load easier than traditional languages.

Problem courtesy of: user80805

Solution

I would give Erlang a try. Even though it will be a steeper learning curve, you will get more out of it since you will be learning a functional programming language. Also, since Erlang is specifically designed to create reliable, highly concurrent systems, you will learn plenty about creating highly scalable services at the same time.

Solution courtesy of: Justin Ethier

Discussion

I'm looking at the same two alternatives you are, gotts, for multiple projects.

So far, the best razor I've come up with to decide between them for a given project is whether I need to use Javascript. One existing system I'm looking to migrate is already written in Javascript, so its next version is likely to be done in node.js. Other projects will be done in some Erlang web framework because there is no existing code base to migrate.

Another consideration is that Erlang scales well beyond just multiple cores, it can scale to a whole datacenter. I don't see a built-in mechanism in node.js that lets me send another JS process a message without caring which machine it is on, but that's built right into Erlang at the lowest levels. If your problem isn't big enough to need multiple machines or if it doesn't require multiple cooperating processes, this advantage isn't likely to matter, so you should ignore it.

Erlang is indeed a deep pool to dive into. I would suggest writing a standalone functional program first before you start building web apps. An even easier first step, since you seem comfortable with Javascript, is to try programming JS in a more functional style. If you use jQuery or Prototype, you've already started down this path. Try bouncing between pure functional programming in Erlang or one of its kin (Haskell, F#, Scala...) and functional JS.

Once you're comfortable with functional programming, seek out one of the many Erlang web frameworks; you probably shouldn't be writing your app directly to something low-level like inets at this late stage. Look at something like Nitrogen, for instance.

Discussion courtesy of: Warren Young

I can't speak for Erlang, but a few things that haven't been mentioned about node:

- Node uses Google's V8 engine to actually compile javascript into machine code. So node is actually pretty fast. So that's on top of the speed benefits offered by event-driven programming and non-blocking io.
- Node has a pretty active community. Hop onto their IRC group on freenode and you'll see what I mean
- I've noticed the above comments push Erlang on the basis that it will be useful to learn a functional programming language. While I agree it's important to expand your skillset and get one of those under your belt, you shouldn't base a project on the fact that you want to learn a new programming style
- On the other hand, Javascript is already in a paradigm you feel comfortable writing in! Plus it's javascript, so when you write client side code it will look and feel consistent.
- node's community has already pumped out tons of modules! There are modules for redis, mongodb, couch, and what have you. Another good module to look into is Express (think Sinatra for node)

Check out the video on yahoo's blog by Ryan Dahl, the guy who actually wrote node. I think that will help give you a better idea where node is at, and where it's going.

Keep in mind that node still is in late development stages, and so has been undergoing quite a few changes—changes that have broke earlier code. However, supposedly it's at a point where you can expect the API not to change too much more. So if you're looking for something fun, I'd say node is a great choice.

Discussion courtesy of: Jarsen

I'm a long-time Erlang programmer, and this question prompted me to take a look at node.js. It looks pretty damn good.

It does appear that you need to spawn multiple processes to take advantage of multiple cores. I can't see anything about setting processor affinity though. You could use taskset on linux, but it probably should be parametrized and set in the program.

I also noticed that the platform support might be a little weaker. Specifically, it looks like you would need to run under Cygwin for Windows support.

Looks good though.

Edit

Node.js now has native support for Windows.

Discussion courtesy of: dsmith

While I'd personally go for Erlang, I'll admit that I'm a little biased against JavaScript. My advice is that you evaluate few points:

1. Are you reusing existing code in either of those languages (both in terms of source code, and programmer experience!)
2. Do you need/want on-the-fly updates without stopping the application (This is where Erlang wins by default - its runtime was designed for that case, and OTP contains all the tools necessary)
3. How big is the expected traffic, in terms of separate, concurrent operations, not bandwidth?
4. How "parallel" are the operations you do for each request?

Erlang has really fine-tuned concurrency & network-transparent parallel distributed system. Depending on what exactly is the project, the availability of a mature implementation of such system might outweigh any issues regarding learning a new language. There are also two other languages that work on Erlang VM which you can use, the Ruby/Python-like Reia and Lisp-Flavored Erlang.

Yet another option is to use both, especially with Erlang being used as kind of "hub". I'm unsure if Node.js has Foreign Function Interface system, but if it has, Erlang has C library for external processes to interface with the system just like any other Erlang process.

Discussion courtesy of: p l

It looks like Erlang performs better for deployment in a relatively low-end server (512MB 4-core 2.4GHz AMD VM). This is from SyncPad's experience of comparing Erlang vs Node.js implementations of their virtual whiteboard server application.

Discussion courtesy of: adib

whatsapp could never achieve the level of scalability and reliability without erlang https://www.youtube.com/watch?v=c12cYAUTXXs

Discussion courtesy of: Henry H.

There is one more language on the same VM that erlang is -> Elixir

It's a very interesting alternative to Erlang, check this one out.

Also it has a fast-growing web framework based on it-> Phoenix Framework

Discussion courtesy of: JustMichael

This content originated from StackOverFlow and has been re-organized into the above recipe.

How do I get the path to the current script with Node.js?

Problem

How would I get the path to the script in Node.js?

I know there's process.cwd, but that only refers to the directory where the script was called, not of the script itself. For instance, say I'm in /home/kyle/ and I run the following command:

```
node /home/kyle/some/dir/file.js
```

If I call process.cwd(), I get /home/kyle/, not /home/kyle/some/dir/. Is there a way to get that directory?

Problem courtesy of: Kyle Slattery

Solution

I found it after looking through the documentation again. What I was looking for were the `__filename` and `__dirname` module-level variables.

Solution courtesy of: Kyle Slattery

Discussion

So basically you can do this:

```
fs.readFile(path.resolve(__dirname, 'settings.json'), 'UTF-8', callback);
```

Use resolve() instead of concatenating with '/' or '\' else you will run into cross-platform issues.

Note: __dirname is the local path of the module or included script. If you are writing a plugin which needs to know the path of the main script it is:

```
require.main.filename
```

or, to just get the folder name:

```
require('path').dirname(require.main.filename)
```

Discussion courtesy of: Marc

```
var settings =
JSON.parse(
require('fs').readFileSync(
require('path').resolve(
__dirname,
'settings.json'),
'utf8'));
```

Discussion courtesy of: foobar

When it comes to the main script it's as simple as:

```
process.argv[1]
```

From the Node.js documentation:

process.argv

An array containing the command line arguments. The first element will be 'node', **the second element will be the name of the JavaScript file**. The next elements will be any additional command line arguments.

If you need to know the path of a module file then use __filename.

Discussion courtesy of: Lukasz Wiktor

Every Node.js program has some global variables in its environment, which represents some information about your process and one of it is __dirname.

Discussion courtesy of: Hazarapet Tunanyan

You can use process.env.PWD to get the current app folder path.

Discussion courtesy of: AbiSivam

This command returns the current directory:

```
var path = process.cwd();
```

For example, to use the path to read the file:

```
var fs = require('fs');
fs.readFile(process.cwd() + "\\text.txt", function(err, data)
{
if(err)
console.log(err)
else
console.log(data.toString());
});
```

Discussion courtesy of: Masoud Siahkali

I know this is pretty old, and the original question I was responding to is marked as duplicate and directed here, but I ran into an issue trying to get jasmine-reporters to work and didn't like the idea that I had to downgrade in order for it to work. I found out that jasmine-reporters wasn't resolving the savePath correctly and was actually putting the reports folder output in jasmine-reporters directory instead of the root directory of where I ran gulp. In order to make this work correctly I ended up using **process.env.INIT_CWD** to get the initial Current Working Directory which should be the directory where you ran gulp. Hope this helps someone.

```
var reporters = require('jasmine-reporters');
var junitReporter = new reporters.JUnitXmlReporter({
savePath: process.env.INIT_CWD + '/report/e2e/',
consolidateAll: true,
captureStdout: true
});
```

Discussion courtesy of: Dana Harris

If you want something more like $0 in a shell script, try this:

```
var path = require('path');

var command = getCurrentScriptPath();

console.log(`Usage: ${command} <foo> <bar>`);

function getCurrentScriptPath () {
// Relative path from current working directory to the location of this script
var pathToScript = path.relative(process.cwd(), __filename);

// Check if current working dir is the same as the script
if (process.cwd() === __dirname) {
// E.g. "./foobar.js"
return '.' + path.sep + pathToScript;
} else {
// E.g. "foo/bar/baz.js"
return pathToScript;
}
}
```

Discussion courtesy of: dmayo3

JavaScript Standard Library for V8

Problem

In my application, I allow users to write plugins using JavaScript. I embed V8 for that purpose. The problem is that developers can't use things like HTTP, Sockets, Streams, Timers, Threading, Cryptography, Unit tests, et cetra.

I searched Stack Overflow and I found node.js. The problem with it is that you can actually **create** HTTP servers, and start processes and more things that I do not want to allow. In addition, node.js has its own environment (./node script.js) and you can't embed it. And it doesn't support Windows - I need it to be fully cross platform. If those problems can be solved, it will be awesome :) But I'm open to other frameworks too.

Any ideas?

Thank you!

Problem courtesy of: Alon Gubkin

Solution

In the end, I built my own library.

Solution courtesy of: Alon Gubkin

Discussion

There is CommonJS, which defines a "standard" and a few implementations available of that standard - one of which is node.js.

But from what I can see, it's still fairly immature and there aren't many "complete" implementations.

Discussion courtesy of: Dean Harding

node.js:: what does hostname do in `listen` function?

Problem

I own two domains, abc.com and xyz.com (not the real ones I own, but they work as an example). They both point to the same ip address. The following is my server js file:

```
var sys=require('sys'),
http=require('http'),
settings=require('./settings');



var srv = http.createServer(function(req, res) {
var body="<b>Hello World!</b>"
res.writeHead(200, {
'content-length': body.length,
'content-type': 'text/html',
'stream': 'keep-alive',
'accept': '*/*'
}
);
res.end(body);
});

srv.listen(8000, 'abc.com' ); // (settings.port, settings.hostname);
```

I then visit http://abc.com:8000/ and http://xyz.com:8000/ and they both display the webpage. I thought that I would only be able to see the page on abc.com since that's what I set as the hostname.

However, when I put '127.0.0.1' as the hostname, then I can only view the page via wget on the server itself.

So what *does* the hostname parameter do?

Problem courtesy of: Alexander Bird

Solution

The following segment of code inside net.js that defines the listen function is pertinent:

```
// the first argument is the port, the second an IP
var port = arguments[0];
dns.lookup(arguments[1], function (err, ip, addressType) {
if (err) {
self.emit('error', err);
} else {
self.type = addressType == 4 ? 'tcp4' : 'tcp6';
self.fd = socket(self.type);
bind(self.fd, port, ip);
self._doListen();
}
});
```

So basically providing a url as the hostname parameter does not allow shared hosting. All node.js does is do the work for you of resolving a hostname to an ip address -- and since in my case both domains point to the same ip, both will work.

For me to do shared hosting, I must find another way.

Solution courtesy of: Alexander Bird

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

What is the benefit of using NginX for Node.js?

Problem

From what I understand Node.js doesnt need NginX to work as a http server (or a websockets server or any server for that matter), but I keep reading about how to use NginX instead of Node.js internal server and cant find of a good reason to go that way

Problem courtesy of: Purefan

Solution

Here http://developer.yahoo.com/yui/theater/video.php?v=dahl-node Node.js author says that Node.js is still in development and so there may be security issues that NginX simply hides.
On the other hand, in case of a heavy traffic NginX will be able to split the job between many Node.js running servers.

Solution courtesy of: mbg

Discussion

But be prepared: nginx don't support http 1.1 while talking to backend so features like keep-alive or websockets won't work if you put node behind the nginx.

UPD: see nginx 1.2.0 - socket.io - HTTP/1.1 - Proxy websocket connections for more up-to-date info.

Discussion courtesy of: Mikhail Korobov

In addition to the previous answers, there’s another practical reason to use nginx in front of Node.js, and that’s simply because you might want to run more than one Node app on your server.

If a Node app is listening on port 80, you are limited to that one app. If nginx is listening on port 80 it can proxy the requests to multiple Node apps running on other ports.

It’s also convenient to delegate TLS/SSL/HTTPS to Nginx. Doing TLS directly in Node is possible, but it’s extra work and error-prone. With Nginx (or another proxy) in front of your app, you don’t have to worry about it and there are tools to help you securely configure it.

Discussion courtesy of: Nate

How to check the number of open connections in node.js?

Problem

I have a machine running node.js (v0.1.32) with a tcp server (tcp.createServer) and a http server (http.createServer). The http server is hit by long polling requests (lasting 50 sec each) from a comet based application on port 80. And there are tcp socket connections on port 8080 from an iphone application for the same purpose.

It was found that the server was not able to handle more connections (especially the tcp connections while the http connections appeared fine!!??) for a while and was normal only after a restart.

For load testing the connections I have created a tcp server and spawned 2000 requests and figured that the connections starting to fail after the max file descriptor limit on machine is reached (default 1024). Which is a really very small number.

So, a novice question here: How do I scale my application to handle more number of connections on node.js and how I handle this issue.

Is there a way to find out how many active connections are there at the moment?

Thanks Sharief

Problem courtesy of: Sharief Shaik

Solution

Hey Joey! I was looking for a unix solution that would help me figure out how many open connections at a given moment anytime on my machine. The reason was my server was not able to handle requests after a certain number of connections. And figured that my machine can handle only 1024 open connections at a time i.e., the ulimit file descriptor value which defaults to 1024. I have modified this value by setting ulimit -n that suits my requirement.

So to check the open connections I used lsof that gives me the list of open files and figured how many connections are open via each port I was using.

Solution courtesy of: Sharief Shaik

Discussion

I don't know if there's a built-in way to get the number of active connections with Node, but it's pretty easy to rig something up.

For my comet-style Node app I keep an object that I add connections to as a property. Every X seconds I iterate over that object and see if there are any connections that should be closed (in your case, anything past your 50 second limit).

When you close a connection, just delete that property from your connections object. Then you can see how many connections are open at any time with Object.size(connections)

Discussion courtesy of: joeynelson

Preferred DOM library for parsing html pages with node.js?

Problem

I'm looking for libraries that parse html pages (with some leniency, not strict xml parsers). Can anyone recommend any?

Edit: Came across jsdom

Problem courtesy of: meder omuraliev

Solution

I recently heard about jsdom via Video: Elijah Insua – jsdom: a CommonJS Implementation of the DOM from the Yahoo Developer Network.

Solution courtesy of: Kevin Hakanson

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

$.getJSON from jQuery (in Rails app) not reading JSON from cross-domain Node.js/Express app / jsonp?

Problem

From googling/forums I think there could be two issues, neither of which I know how to fix:

1) I need to do something with the jsonp callback in the node.js request (which was generated automatically by jquery b/c of the callback=? param) - either add it to the header (where? and how?) or add it to the json response (again, where? and how?)

2) There could be some issue with the node.js request connection staying open, and therefore never signaling the callback?

jquery code:

```
url = http://nodeapp.com/users?screen_name=s&callback=?
$.getJSON(this_url, function(data){
var users = data
$.each(users, function(i, item){
$("#users").append(item...);
})
```

node.js/express.js (coffee script) code:

```
get '/user', ->
users.all... =>
@header("Content-Type", 'application/javascript')
@respond 200, JSON.stringify(users)
```

Problem courtesy of: Michael Waxman

Solution

I am not familiar with RoR but when you request http://nodeapp.com/users?screen_name=s&callback=foo, the response should look like this:

```
foo({ first_name: 'John', last_name: 'Smith' });
```

and not:

```
{ first_name: 'John', last_name: 'Smith' }
```

As far as the callback=? parameter is concerned you don't need to modify it, jQuery will pass a random value so that the anonymous callback could be invoked.

Solution courtesy of: Darin Dimitrov

Discussion

There is currently no discussion for this recipe.

This content originated from StackOverFlow and has been re-organized into the above recipe.

Server side javascript on Google app engine

Problem

Is there any way to run a Javascript engine, like Spidermonkey, on Google App Engine? Spidermonkey is a C module, so obviously that wont work (GAE doesn't allow those types of modules)... is there something else available?

Problem courtesy of: Nick Franceschina

Solution

Here is an article about running Rhino on AppEngine/Java. That should get you a long way towards a real, functioning JavaScript application on AppEngine.

Solution courtesy of: Adam Crossland

Discussion

If you are looking for a JavaScript framework (as opposed to calling Java methods from JavaScript) you could try RingoJS (formerly Helma NG). It's a Rhino-based JavaScript framework that can run in AppEngine.

There's also AppengineJS, which can run on RingoJS (or Narwhal, which I haven't personally used). It's a port of the Python SDK (with mostly predictable changes to fit JavaScript conventions better). It's not complete, but it's close enough to work in most cases. It's nicer to use than using the Java API directly.

Discussion courtesy of: Matthew Crumley

I've also built ApeJS if you want to try it out. It's much more minimal than the competition.

Discussion courtesy of: Luca Matteis

Google is now supporting custom language on Google App Engine. So we can do Node.js

https://www.youtube.com/watch?v=Q8jZHc0NS6A

https://developers.google.com/cloud/managed-vms

Discussion courtesy of: fernandopasik

This content originated from StackOverFlow and has been re-organized into the above recipe.

How can I share code between Node.js and the browser?

Problem

I am creating a small application with a JavaScript client (run in the browser) and a Node.js server, communicating using WebSocket.

I would like to share code between the client and the server. I have only just started with Node.js and my knowledge of modern JavaScript is a little rusty, to say the least. So I am still getting my head around the CommonJS require() function. If I am creating my packages by using the 'export' object, then I cannot see how I could use the same JavaScript files in the browser.

I want to create a set of methods and classes that are used on both ends to facilitate encoding and decoding messages, and other mirrored tasks. However, the Node.js/CommonJS packaging systems seems to preclude me from creating JavaScript files that can be used on both sides.

I also tried using JS.Class to get a tighter OO model, but I gave up because I couldn't figure out how to get the provided JavaScript files to work with require(). Is there something am I missing here?

Problem courtesy of: Simon Cave

Solution

If you want to write a module that can be used both client side and server side, I have a short blog post on a quick and easy method: *Writing for Node.js and the browser*, essentially the following (where this is the same as window):

```
(function(exports){

// Your code goes here

exports.test = function(){
return 'hello world'
};

})(typeof exports === 'undefined'? this['mymodule']={}: exports);
```

Alternatively there are some projects aiming to implement the Node.js API on the client side, such as Marak's gemini.

You might also be interested in DNode, which lets you expose a JavaScript function so that it can be called from another machine using a simple JSON-based network protocol.

Solution courtesy of: Caolan

Discussion

Don't forget that the string representation of a JavaScript function represents the source code for that function. You could simply write your functions and constructors in an encapsulated way so they can be toString()'d and sent to the client.

Another way to do it is use a build system, put the common code in separate files, and then include them in both the server and client scripts. I'm using that approach for a simple client/server game via WebSockets where the server and client both run essentially the same game loop and the client synchronises up with the server every tick to make sure nobody's cheating.

My build system for the game is a simple Bash script that runs the files through the C preprocessor and then through sed to clean up some junk cpp leaves behind, so I can use all the normal preprocessor stuff like #include, #define, #ifdef, etc.

Discussion courtesy of: Dagg Nabbit

The server can simply send JavaScript source files to the client (browser) but the trick is that the client will have to provide a mini "exports" environment before it can exec the code and store it as a module.

A simple way to make such an environment is to use a closure. For example, say your server provides source files via HTTP like http://example.com/js/foo.js. The browser can load the required files via an XMLHttpRequest and load the code like so:

```
ajaxRequest({
method: 'GET',
url: 'http://example.com/js/foo.js',
onSuccess: function(xhr) {
var pre = '(function(){var exports={};'
, post = ';return exports;})()';
window.fooModule = eval(pre + xhr.responseText + post);
}
});
```

The key is that client can wrap the foreign code into an anonymous function to be run immediately (a closure) which creates the "exports" object and returns it so you can assign it where you'd like, rather than polluting the global namespace. In this example, it is assigned to the window attribute fooModule which will contain the code exported by the file foo.js.

Discussion courtesy of: maerics

I would recommend looking into the RequireJS adapter for Node.js. The problem is that the CommonJS module pattern Node.js uses by default isn't asynchronous, which blocks loading in the web browser. RequireJS uses the AMD pattern, which is both asynchronous and compatible with both server and client, as long as you use the r.js adapter.

Discussion courtesy of: Husky

now.js is also worth a look. It allows you to call server-side from the client-side, and client-side functions from the server-side

Discussion courtesy of: balupton

Epeli has a nice solution here http://epeli.github.com/piler/ that even works without the library, just put this in a file called share.js

```
(function(exports){

exports.test = function(){
return 'This is a function from shared module';
};

}(typeof exports === 'undefined' ? this.share = {} : exports));
```

On the server side just use:

```
var share = require('./share.js');

share.test();
```

And on the client side just load the js file and then use

```
share.test();
```

Discussion courtesy of: broesch

None of the previous solutions bring the CommonJS module system to the browser.

As mentioned in the other answers, there are asset manager/packager solutions like Browserify or Piler and there are RPC solutions like dnode or nowjs.

But I couldn't find an implementation of CommonJS for the browser (including a require() function and exports / module.exports objects, etc.). So I wrote my own, only to discover afterwards that someone else had written it better than I had: https://github.com/weepy/brequire. It's called Brequire (short for Browser require).

Judging by popularity, asset managers fit the needs of most developers. However, if you need a browser implementation of CommonJS, Brequire will probably fit the bill.

2015 Update: I no longer use Brequire (it hasn't been updated in a few years). If I'm just writing a small, open-source module and I want anyone to be able to easily use, then I'll follow a pattern similar to Caolan's answer (above) -- I wrote a blog post about it a couple years ago.

However, if I'm writing modules for private use or for a community that is standardized on CommonJS (like the Ampersand community) then I'll just write them in CommonJS format and use Browserify.

Discussion courtesy of: Peter Rust

Maybe this is not entirely in line with the question, but I thought I'd share this.

I wanted to make a couple of simple string utility functions, declared on String.prototype, available to both node and the browser. I simply keep these functions in a file called utilities.js (in a subfolder) and can easily reference it both from a script-tag in my browser code, and by using require (omitting the .js extension) in my Node.js script:

my_node_script.js

```
var utilities = require('./static/js/utilities')
```

my_browser_code.html

```
<script src="/static/js/utilities.js"></script>
```

I hope this is useful information to someone other than me.

Discussion courtesy of: Markus Amalthea Magnuson

If you want to write your browser in Node.js-like style you can try dualify.

There is no browser code compilation, so you can write your application without limitations.

Discussion courtesy of: farincz

Write your code as RequireJS modules and your tests as Jasmine tests.

This way code can be loaded everywhere with RequireJS and the tests be run in the browser with jasmine-html and with jasmine-node in Node.js without the need to modify the code or the tests.

Here is a working example for this.

Discussion courtesy of: Blacksonic

Checkout the jQuery source code that makes this work in the Node.js module pattern, AMD module pattern, and global in the browser:

```
(function(window){
var jQuery = 'blah';

if (typeof module === "object" && module && typeof module.exports === "object")
{

// Expose jQuery as module.exports in loaders that implement the Node
// module pattern (including browserify). Do not create the global, since
// the user will be storing it themselves locally, and globals are frowned
// upon in the Node module world.
module.exports = jQuery;
}
else {
// Otherwise expose jQuery to the global object as usual
window.jQuery = window.$ = jQuery;

// Register as a named AMD module, since jQuery can be concatenated with other
// files that may use define, but not via a proper concatenation script that
```

```
// understands anonymous AMD modules. A named AMD is safest and most robust
// way to register. Lowercase jquery is used because AMD module names are
// derived from file names, and jQuery is normally delivered in a lowercase
// file name. Do this after creating the global so that if an AMD module wants
// to call noConflict to hide this version of jQuery, it will work.
if (typeof define === "function" && define.amd) {
define("jquery", [], function () { return jQuery; });
}
}
})(this)
```

Discussion courtesy of: wlingke

I wrote this, it is simple to use if you want to set all variables to the global scope:

```
(function(vars, global) {
for (var i in vars) global[i] = vars[i];
})({
abc: function() {
...
},
xyz: function() {
...
}
}, typeof exports === "undefined" ? this : exports);
```

Discussion courtesy of: heeeelo

Parse PHP Session in Javascript

Problem

I have recently gone into extending my site using node.js and have come to realisation I need a session handler for my PHP sessions. Now everything was cool and dandy and node.js reads the php sessions and can propogate it's own session with the php ones. I am using database sessions so the session data gets saved into a field in the database.

I have however found a slight problem. I am attempting to read the session data into node.js and it's really quite a strange string. I have been able to get the strucutre of each session variable down to:

'field_name'|'type':'length':'value';

Now on certain strings the value field can be missing on other strings the length can be missing (when a variable is Null). The type can also be more than b, s, i; it can also be N (NULL).

I had originally thought up of a huge translator for JS but this just somehow seems a very wrong way to do it.

Has anyone here tried to extract php session variables in JS before and is there any kind of script that could help? Maybe there is a formatting thing I can use on PHP side to make my life a lot easier in node.js?

Edit: the Schema looks like:

```
{ _id: { id: 'L:\u00c1\u009d\u008e\u00ad\u000e}<\u0002\u0000\u0000' }
, session_id: 'a2clfnjhopv1srs5k5elgbfjv5'
, user_id: 0
,                                                                    session_data:
'logged|b:0;uid|i:0;server_key|N;AUTH_TIER2|b:0;email|s:0:"";cheese|s:6:"cheese"
, active: 1
, expires: 1278920567
}
```

This is the mongo db record for a user session. The field needing to be translated is session_data. There is some kind of formatting error when pasting it in since stackoverflow wont format that as code when I try and make it for some reason.

I tried to JSONfy the field before but it lost it's types and didn't read Null entries etc so I stopped that

Thanks,

Problem courtesy of: Sammaye

Solution

I'm relatively sure PHP uses the serialize and unserialize functions to handle session data.

There is a JavaScript implementation of unserialize in PHP.JS, you can find it here: http://phpjs.org/functions/unserialize

Solution courtesy of: Jani Hartikainen

Discussion

Just to reply with the way I've found: force PHP to use JSON instead :

Use the class below, with session_set_save_handler to store data as JSON and let PHP still using his own system. Link this class using this function at the beginning of each script you use session : http://php.net/manual/ru/function.session-set-save-handler.php

PS : here the class use memcache to store JSON resulting object, on Node.JS, use a memcache object, and retrieve like this :

- get the PHPSESSID from cookies
- use that characters string to bring a key from memcached with "sessions/id" where id is the key and session just the string (see class below).

Now you should have only JSON in memcache, so it's becoming a simple game with Node.JS to work with.

PS : of course you can work with/improve this, you can also use this not only with memcache (a db for example)

```
<?php
/**
* This class is used to store session data with memcache, it store in json the session to be used more easily in Node.JS
*/
class memcacheSessionHandler{
private static $lifetime = 0;
private static $memcache = null;

public function __construct(){
self::$memcache = new Memcache();
self::$memcache->addServer('localhost', 11211);
}

public function __destruct(){
session_write_close();
self::$memcache->close();
self::$memcache = null;
}

public static function open(){
self::$lifetime = ini_get('session.gc_maxlifetime');
return true;
}

public static function read($id){
$tmp = $_SESSION;
$_SESSION = json_decode(self::$memcache->get("sessions/{$id}"), true);
$new_data = session_encode();
$_SESSION = $tmp;
return $new_data;
}
```

```
public static function write($id, $data){
$tmp = $_SESSION;
session_decode($data);
$new_data = $_SESSION;
$_SESSION = $tmp;
return    self::$memcache->set("sessions/{$id}",    json_encode($new_data),    0,
self::$lifetime);
}

public static function destroy($id){
return self::$memcache->delete("sessions/{$id}");
}

public static function gc(){
return true;
}

public static function close(){
return true;
}
}
?>
```

Discussion courtesy of: Deisss

Here is a session_decode function based on the unserialize function of phpjs: https://github.com/vianneyb/phpjs/blob/master/functions/var/session_decode.js

works for me!

Discussion courtesy of: vianney

I had the same problem. I needed to integrate php with node.js. I used js-php-unserialize to do this. The use is pretty straightforward.

```
var PHPUnserialize = require('php-unserialize');
console.log(PHPUnserialize.unserializeSession(yourData));
```

For example if you

```
var yourData = 'A|s:1:"B";userID|s:24:"53d620475e746b37648b4567";';
```

you will get this result:

```
{
A: 'B',
userID: '53d620475e746b37648b4567'
}
```

Discussion courtesy of: Salvador Dali

What's Next?

If you enjoyed these 50 recipes on Programming Node.js and are looking for more. I would suggest visiting our website where there are literally thousands more recipes to read at: Node.js Recipes.

Table of Contents