Cheatography

Node Cheatsheet Cheat Sheet

by camilo.herbert via cheatography.com/66323/cs/16551/

Basic Webserver example

```
var http = require('http');
http.c rea teS erv er( fun ction
(request, response) {
    res pon se.w ri teH ead -
(200, {'Cont ent -Type':
'text/ pla in'});
    res pon se.e nd ('Hello
World\n');
}).lis ten (8124);
consol e.l og( 'Server running
at http:/ /12 7.0.0.1 :8 124/');
```

Timers

To schedule execution of a one-time callback after delay milliseconds. Optionally you can also pass arguments to the callback.

setTim eou t(c all back, delay, [arg], [...]);

Stop a timer that was previously created with setTim eout().

clearT ime out(t);

To schedule the repeated execution of callback every delay millis econds. Optionally you can also pass arguments to the callback.

setInt erv al(cal lback, delay, [arg], [...]);

Stop a timer that was previously created with setInt erv al().

clearI nte rva l(t);

Timers (cont)

To schedule the " imm edi ate " execution of callback after I/O events callbacks and before setTimeout and setInt erval.

setImm edi ate (ca llback, [arg], [...]);

Stop a timer that was previously created with setImm edi ate().

clearI mme dia te(imm edi ate - Obj ect);

Allow you to create a timer that is active but if it is the only item left in the event loop, node won't keep the program running.

unref();

If you had previously unref()d a timer you can call ref() to explicitly request the timer hold the program open.

ref();

Events

Adds a listener to the end of the listeners array for the specified event.

emitte r.a ddL ist ene r(e vent, listener);

Same as emitte r.a ddL ist - ener().

emitte r.o n(e vent, listener);

Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.

Events (cont)

emitte r.o nce (event, listener);

Remove a listener from the listener array for the specified event.

emitte r.r emo veL ist ene r(e - vent, listener);

Removes all listeners, or those of the specified event.

emitte r.r emo veA llL ist ene - rs([ev ent]);

By default EventE mitters will print a warning if more than 10 listeners are added for a particular event.

emitte r.s etM axL ist ene - rs(n);

Returns an array of listeners for the specified event.

emitte r.l ist ene rs(event);

Execute each of the listeners in order with the supplied arguments. Returns true if event had listeners, false otherwise.

emitte r.e mit (event, [arg1], [arg2], [...]);

Return the number of listeners for a given event.

EventE mit ter.li ste ner Cou - nt(emi tter, event);

File system

Write buffer to the file specified by fd.

fs.wri te(fd, buffer, offset, length, position, callback);

Synchr onous version of fs.wri te(). Returns the number of bytes written.

fs.wri teS ync(fd, buffer, offset, length, position);

Read data from the file specified by fd.

fs.rea d(fd, buffer, offset, length, position, callback);

Synchr onous version of fs.read. Returns the number of bytesRead.

fs.rea dSy nc(fd, buffer, offset, length, position);

Asynch ron ously reads the entire contents of a file.

fs.rea dFi le(fil ename, [options], callback);

Synchr onous version of fs.rea - dFile. Returns the contents of the filename. If the encoding option is specified then this function returns a string. Otherwise it returns a buffer.

fs.rea dFi leS ync (fi lename, [optio ns]);

Asynch ron ously writes data to a file, replacing the file if it already exists. data can be a string or a buffer.

File system (cont)

fs.wri teF ile (fi lename, data, [options], callback);

The synchr onous version of fs.wri teFile.

fs.wri teF ile Syn c(f ile name, data, [optio ns]);

Asynch ron ously append data to a file, creating the file if it not yet exists. data can be a string or a buffer.

fs.app end Fil e(f ile name, data, [options], callback);

The synchr onous version of fs.app end File.

fs.app end Fil eSy nc(fil - ename, data, [optio ns]);

Watch for changes on filename, where filename is either a file or a directory. The returned object is a fs.FSW atcher. The listener callback gets two arguments (event, filename). event is either 'rename' or 'change', and filename is the name of the file which triggered the event.

fs.wat ch(fil ename, [options], [liste ner]);

Test whether or not the given path exists by checking with the file system. Then call the callback argument with either true or false. (should not be used)

fs.exi sts (path, callback);

Synchr onous version of fs.exists. (should not be used)

File system (cont)

fs.exi sts Syn c(p ath);

fs.Stats: objects returned from fs.stat(), fs.lstat() and fs.fstat() and their synchr - onous counte rparts are of this type.

stats.i sF ile();

stats.i sD ire ctory()

stats.i sB loc kDe vice()

stats.i sC har act erD evice()

stats.i sS ymb oli cLink()

(only valid with fs.lst at())

stats.i sF IFO()

stats.i sS ocket()

Returns a new ReadStream object.

fs.cre ate Rea dSt rea m(path, [optio ns]);

HTTP - Requests

Sends a chunk of the body.

reques t.w rit e(c hunk, [encod ing]);

Finishes sending the request. If any parts of the body are unsent, it will flush them to the stream.

reques t.e nd([data], [encod - ing]);

Aborts a request.

reques t.a bort();

Once a socket is assigned to this request and is connected socket.se tTi meout() will be called.

HTTP - Requests (cont)

reques t.s etT ime out (ti - meout, [callb ack]);
Once a socket is assigned to this request and is connected socket.se tNo Delay() will be called.
reques t.s etN oDe lay ([n oDe - lay]);
Once a socket is assigned to this request and is connected socket.se tKe epA live() will be called.
reques t.s etS ock etK eep Ali - ve([en able], [initi alD - elay]);
Emitted when a response is received to this request. This event is emitted only once.
reques t.o n(' res ponse', functi on(res ponse) { });
Emitted after a socket is assigned to this request.
reques t.o n(' soc ket', functi on(socket) { });
Emitted each time a server responds to a request with a CONNECT method. If this event isn't being listened for, clients receiving a CONNECT method will have their connec - tions closed.
reques t.o n(' con nect', functi on(res ponse, socket, head) { });
Emitted each time a server responds to a request with an upgrade. If this event isn't being listened for, clients receiving an upgrade header will have their connec tions closed.

HTTP - Requests (cont)

reques t.o n(' upg rade', functi on(res ponse, socket, head) { });
Emitted when the server sends a '100 Continue' HTTP response, usually because the request contained 'Expect: 100-co nti - nue'. This is an instru ction that the client should send the request body.
reques t.o n(' con tinue', function() { });

Global Objects

The filename of the code being executed. (absolute path)
__file name;
The name of the directory that the currently executing script resides in. (absolute path)
__dirname;
A reference to the current module. In particular module.ex ports is used for defining what a module exports and makes available through require().
module;
A reference to the module.ex - ports that is shorter to type.
exports;
The process object is a global object and can be accessed from anywhere. It is an instance of EventE mitter.

Global Objects (cont)

process;
The Buffer class is a global type for dealing with binary data directly.
Buffer;

Modules

Loads the module module.js in the same directory.
var module = requir e('./m odu - le.j s');
load anothe r_m odule as if require() was called from the module itself.
module.re qui re('./ ano the - r_m odu le.j s');
The identifier for the module. Typically this is the fully resolved filename.
module.id;
The fully resolved filename to the module.
module.fi lename;
Whether or not the module is done loading, or is in the process of loading.
module.lo aded;
The module that required this one.
module.pa rent;
The module objects required by this one.
module.ch ildren;
export s.area = function (r) {
return Math.PI * r;

Node Cheatsheet Cheat Sheet

by camilo.herbert via cheatography.com/66323/cs/16551/

Modules (cont)

```
};
```

If you want the root of your module's export to be a function (such as a constr uctor) or if you want to export a complete object in one assignment instead of building it one property at a time, assign it to module.ex ports instead of exports.

```
module.ex ports = functi on( -
width) {
    return {
        area: function() {
            return width *
width;
        }
    };
}
```

Stream - Writable

var writer = getWritableStreamSomehow();

This method writes some data to the underlying system, and calls the supplied callback once the data has been fully handled.

writab le.w ri te(chunk, [encod ing], [callb ack]);

If a writab le.w ri te(chunk) call returns false, then the drain event will indicate when it is approp riate to begin writing more data to the stream.

writer.on ce('dr ain', write);

Stream - Writable (cont)

Call this method when no more data will be written to the stream.

writab le.e nd ([c hunk], [encod ing], [callb ack]);

When the end() method has been called, and all data has been flushed to the underlying system, this event is emitted.

writer.on ('f inish', function() {});

This is emitted whenever the pipe() method is called on a readable stream, adding this writable to its set of destin - ations.

writer.on ('p ipe', functi - on(src) {});

This is emitted whenever the unpipe() method is called on a readable stream, removing this writable from its set of destin ations.

writer.on ('u npipe', functi - on(src) {});

Emitted if there was an error when writing or piping data.

writer.on ('e rror', functi - on(src) {});

Path

Normalize a string path, taking care of '..' and '.' parts.

path.n orm ali ze(p);

Join all arguments together and normalize the resulting path.

Path (cont)

path.j oin ([p ath1], [path2], [...]);

Resolves 'to' to an absolute path.

path.r eso lve ([from ...], to);

Solve the relative path from 'from' to 'to'.

path.r ela tiv e(from, to);

Return the directory name of a path. Similar to the Unix dirname command.

path.d irn ame(p);

Return the last portion of a path. Similar to the Unix basename command.

path.b ase name(p, [ext]);

Return the extension of the path, from the last '.' to end of string in the last portion of the path.

path.e xtn ame(p);

The platfo rm- spe cific file separator. '\\' or '/'.

path.sep;

The platfo rm- spe cific path delimiter, ';' or ':'.

path.d eli miter;

HTTP - Server Events

Emitted each time there is a request.

server.on ('r equ est', function (request, response) { });

When a new TCP stream is establ ished.

server.on ('c onn ect ion', function (socket) { });

Emitted when the server closes.

server.on ('c lose', function () { });

Emitted each time a request with an http Expect: 100-co ntinue is received.

server.on ('c hec kCo nti nue', function (request, response) { });

Emitted each time a client requests a http CONNECT method.

server.on ('c onn ect', function (request, socket, head) { });

Emitted each time a client requests a http upgrade.

server.on ('u pgr ade', function (request, socket, head) { });

If a client connection emits an 'error' event - it will forwarded here.

server.on ('c lie ntE rror', function (excep tion, socket) { });

HTTP - Responses

This sends a chunk of the response body. If this merthod is called and response.writeHead() has not been called, it will switch to implicit header mode and flush the implicit headers.

respon se.w ri te(chunk, [encod ing]);

Sends a HTTP/1.1 100 Continue message to the client, indicating that the request body should be sent.

respon se.w ri teC ont inue();

Sends a response header to the request.

respon se.w ri teH ead (st atu - sCode, [reaso nPh rase], [heade rs]);

Sets the Socket's timeout value to msecs. If a callback is provided, then it is added as a listener on the 'timeout' event on the response object.

respon se.s et Tim eou t(m secs, callback);

Sets a single header value for implicit headers. If this header already exists in the to-be-sent headers, its value will be replaced. Use an array of strings here if you need to send multiple headers with the same name.

respon se.s et Hea der (name, value);

HTTP - Responses (cont)

Reads out a header that's already been queued but not sent to the client. Note that the name is case insens itive.

respon se.g et Hea der (name);

Removes a header that's queued for implicit sending.

respon se.r em ove Hea der - (name);

This method adds HTTP trailing headers (a header but at the end of the message) to the response.

respon se.a dd Tra ile rs(hea - ders);

This method signals to the server that all of the response headers and body have been sent; that server should consider this message complete. The method, respon se.e nd(), MUST be called on each response.

respon se.e nd ([d ata], [encod ing]);

When using implicit headers (not calling respon se.w ri teH ead() explic itly), this property controls the status code that will be sent to the client when the headers get flushed.

respon se.s ta tus Code;

Boolean (read- only). True if headers were sent, false otherwise.

respon se.h ea der sSent;

HTTP - Responses (cont)

When true, the Date header will be automa tically generated and sent in the response if it is not already present in the headers. Defaults to true.

respon se.s en dDate;

Indicates that the underlying connection was terminated before respon se.e nd() was called or able to flush.

respon se.o n('cl ose', function () { });

Emitted when the response has been sent.

respon se.o n('fi nish', function() { });

Console

Prints to stdout with newline.

consol e.l og([data], [...]);

Same as consol e.log.

consol e.i nfo ([d ata], [...]);

Same as consol e.log but prints to stderr.

consol e.e rro r([data], [...]);

Same as consol e.e rror.

consol e.w arn ([d ata], [...]);

Uses util.i nspect on obj and prints resulting string to stdout.

consol e.d ir(obj);

Mark a time.

consol e.t ime (la bel);

Console (cont)

Finish timer, record output.

consol e.t ime End (la bel);

Print a stack trace to stderr of the current position.

consol e.t rac e(l abel);

Same as assert.ok() where if the expression evaluates as false throw an Assert ion Error with message.

consol e.a sse rt(exp res sion, [messa ge]);

Process

Emitted when the process is about to exit

proces s.o n(' exit', functi - on(code) {});

Emitted when an exception bubbles all the way back to the event loop. (should not be used)

proces s.o n(' unc aug htE xce - ption', functi on(err) {});

A writable stream to stdout.

proces s.s tdout;

A writable stream to stderr.

proces s.s tderr;

A readable stream for stdin.

proces s.s tdin;

An array containing the command line arguments.

proces s.argv;

An object containing the user enviro nment.

proces s.env;

Process (cont)

This is the absolute pathname of the executable that started the process.

proces s.e xec Path;

This is the set of node-s - pecific command line options from the executable that started the process.

proces s.e xec Argv;

Stream - Readable

var readable = getReadableStreamSomehow();

When a chunk of data can be read from the stream, it will emit a 'readable' event.

readab le.o n('re ada ble', function() {});

If you attach a data event listener, then it will switch the stream into flowing mode, and data will be passed to your handler as soon as it is available.

readab le.o n('data', functi - on(chunk) {});

This event fires when there will be no more data to read.

readab le.o n('end', function() {});

Emitted when the underlying resource (for example, the backing file descri ptor) has been closed. Not all streams will emit this.

readab le.o n('cl ose', function() {});

Emitted if there was an error receiving data.

Stream - Readable (cont)

readab le.o n('er ror', function() {});
The read() method pulls some data out of the internal buffer and returns it. If there is no data available, then it will return null.
This method should only be called in non-fl owing mode. In flowin g-mode, this method is called automa tically until the internal buffer is drained.
readab le.r ea d([size]);
Call this function to cause the stream to return strings of the specified encoding instead of Buffer objects.
readab le.s et Enc odi ng(enc - oding);
This method will cause the readable stream to resume emitting data events.
readab le.r es ume();
This method will cause a stream in flowin g-mode to stop emitting data events.
readab le.p au se();
This method pulls all the data out of a readable stream, and writes it to the supplied destin ation, automa tically managing the flow so that the destin ation is not overwh elmed by a fast readable stream.
readab le.p ip e(d est ina tion, [optio ns]);

Stream - Readable (cont)

This method will remove the hooks set up for a previous pipe() call. If the destin ation is not specified, then all pipes are removed.
readab le.u np ipe ([d est ina - tion]);
This is useful in certain cases where a stream is being consumed by a parser, which needs to " - un- con sum e" some data that it has optimi sti cally pulled out of the source, so that the stream can be passed on to some other party.
readab le.u ns hif t(c hunk);

HTTP

A collection of all the standard HTTP response status codes, and the short description of each.
http.S TAT US_ CODES;
This function allows one to transp arently issue requests.
http.r equ est (op tions, [callb ack]);
Set the method to GET and calls req.end() automa tic ally.
http.g et(opt ions, [callb - ack]);
Returns a new web server object. The reques tLi stener is a function which is automa tically added to the 'request' event.
server = http.c rea teS erv - er([re que stL ist ener]);

HTTP (cont)

Begin accepting connec tions on the specified port and hostname.
server.li ste n(port, [hostn - ame], [backlog], [callb ack]);
Start a UNIX socket server listening for connec tions on the given path.
server.li ste n(path, [callb - ack]);
The handle object can be set to either a server or socket (anything with an underlying _handle member), or a {fd: <n>} object.
server.li ste n(h andle, [callb ack]);
Stops the server from accepting new connec tions.
server.cl ose ([c all back]);
Sets the timeout value for sockets, and emits a 'timeout' event on the Server object, passing the socket as an argument, if a timeout occurs.
server.se tTi meo ut(msecs, callback);
Limits maximum incoming headers count, equal to 1000 by default. If set to 0 - no limit will be applied.
server.ma xHe ade rsC ount;
The number of millis econds of inactivity before a socket is presumed to have timed out.
server.ti meout;

HTTP - Messages

In case of server request, the HTTP version sent by the client. In the case of client response, the HTTP version of the connected-to server.

messag e.h ttp Ver sion;

The reques t/r esponse headers object.

messag e.h eaders;

The reques t/r esponse trailers object. Only populated after the 'end' event.

messag e.t rai lers;

The request method as a string. Read only. Example: 'GET', 'DELETE'.

messag e.m ethod;

Request URL string. This contains only the URL that is present in the actual HTTP request.

messag e.url;

The 3-digit HTTP response status code. E.G. 404.

messag e.s tat usCode;

The net.Socket object associated with the connec tion.

messag e.s ocket;

Calls messag e.c onn ect - ion.se tTi meo ut(msecs, callback).

messag e.s etT ime out (msecs, callback);