# NODE FIRM

Converbbil 2013 The Node Firm All Dights December



# In Node, HTTP is a first citizen. This includes the ability to make outbound requests:

#### 01\_client\_get.js:

```
var http = require('http');
http.get('http://www.google.com/humans.txt', function(res) {
    res.pipe(process.stdout);
};
```

HTTP.REQUEST

 $\label{eq:http.request} \mbox{http.request is used to make } \mbox{outbound} \mbox{ http requests using any of the } \mbox{HTTP request methods}.$ 

## HTTP.REQUEST OPTIONS

02\_get\_status.js:

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

var options = {
    hostname: 'www.google.com',
    port: 80,
    path: '/',
    method: 'GET',
    agent: false
};

var req = http.request(options, function(res) {
    console.log('STATUS: ' + res.statusCode);
});

req.end();
```

## HTTP.REQUEST OPTIONS

To easily build up a request options object from a string when additional options are necessary, simply pass the url string to url.parse.

```
var url = require('url');
var options = url.parse('http://sandbox.thenodefirm.com/archives/ll');
options.method = 'PATCH';
```

# HTTP POST request (location does not exist) 03\_client\_post.js:

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

var options = url.parse('http://www.google.com/upload');
options.method = 'POST';
console.log('OPTIONS: ' + JSON.stringify(options));

var req = http.request(options, function(res) {
    console.log('STATUS: ' + res.statusCode);
    console.log('SDATUS: ' + res.statusCode);
    res.setEncoding('utf8');
    res.on('data', function (chunk) {
        console.log('BODY: ' + chunk);
    });
});
on('error', function(err) {
        console.log('problem with request: ' + err.message);
});

// write data to request body
    req.write('data\n');
    req.end('data\n');
```

## HTTP.GET

- The only convenience method.
- Takes advantage of automatic parsing with url.parse of the first options parameter.
- Automatically calls req.end().

#### 01\_client\_get.js:

```
var http = require('http');
http:get('http://www.google.com/humans.txt', function(res) {
   res.pipe(process.stdout);
});
```

## TERMINATING REQUESTS

An unfulfilled or stuck request will leak resources.

To get around this use the setTimeout() method.

Don't be overly aggressive with timeout times.

30 seconds is a sensible value.

## Request ignore server

## 04\_server.js:

```
require('http').createServer(function(req, res) {
    // just do nothing, we should get a timeout event.
    //res.end('hello node')
}).listen(8081, function () {
    console.log('Listening on %d.', this.address().port)
});
```

## Timeout unfufilled request

## 04\_client\_terminate.js:

```
var http = require('http');
var timeout = process.argv[2] || 2000;

var req = http.get('http://localhost:8081', function(res) {
    res.on('data', function(d) {
        console.log('chunk with %d bytes', d.length);
    });
});

req.setTimeout(timeout, function () {
    console.log('request timed out after %dms', timeout);
});

req.on('error', function (err) {
    console.log('error: ', err);
});
```

## **CONNECTION: KEEP-ALIVE** HTTP Client uses Connection: keep-alive by default.

The connection to the server should be persisted until the next request.

#### Display request headers 05\_server.js:

```
var server = require('http').createServer();
 console.log('REQUEST: ' + JSON.stringify(req.headers));
 res.end();
server.listen(8081, function () {
```

## Make multiple client requests

#### 05\_client\_multi.js:

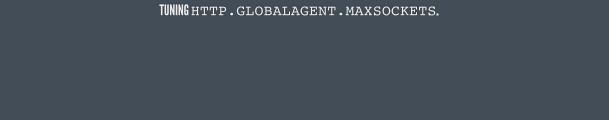
```
var http = require('http');
var count = Number(process.argv(2]) || 5;
for (var i=0; i < count; i++) {
  http.get('http://localhost:8081', function(res) {
    res.pipe(process.stdout);
  }).on('error', console.error);
}</pre>
```

# HTTP.GLOBALAGENT

By default Node limits the number of concurrent sockets an agent can have open per host to 5.

It's important to be aware of this constraint.

Internal requests especially are likely to find this suboptimal.



#### Display the number of connections on each request 06\_server\_connections.js:

```
var http = require('http');
var server = http.createServer();
server.on('request', function(req, res) {
    server.getConnections(function (err, connections) {
        console.log('connections %d', connections);
    });
    res.end('Node');
});
server.listen(8081, function () {
        console.log('Listening on %d.', this.address().port)
});
```

# Tuning maxSockets **06\_maxsockets.js:**

```
var http = require('http');
//http.globalAgent.maxSockets = 10;
var count = Number(process.argv[2]) || 10;
for (var i=0; i < count; i++) {
   get(i);
}
function get(i) {
   http.get('http://localhost:8081', function(res) {
      res.on('data', function (data) {
      console.log(i + ' ' + data);
   })
}).on('error', console.error);
}</pre>
```

## AGENT = FALSESetting a http.request agent to false disables socket pooling and disables keep-alive, setting Connection: close.

Use the previous server. Disable socket pooling.

## 07\_agent\_false.js:

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

var count = Number(process.argv[2]) || 10;
var options = url.parse('http://localhost:8081');
options.agent = false;

for (var i=0; i < count; i++) {
    get(i);
}

function get(i) {
    http.get(options, function(res) {
        res.on('data', function (data) {
            console.log(i + ' ' + data);
        })
    }).on('error', console.error);
}</pre>
```

## • http.request enables you to make outbound http requests.

Understanding http.globalAgent and

**SUMMARY** 

http.globalAgent.maxSockets behavior is important, especially in internal systems that primarily interface with the same hosts.