HTTP CLIENT

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

`http.request` is used to make **outbound** http requests using any of the 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/11');
options.method = 'PATCH';
```

EXAMPLE

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('HEADERS: ' + JSON.stringify(res.headers));
  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.

EXAMPLE

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)
});
```

EXAMPLE

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.

EXAMPLE

Display request headers

05_server.js:

```
var server = require('http').createServer();

server.on('request', function (req, res) {
  console.log('REQUEST: ' + JSON.stringify(req.headers));
  res.end();
});

server.listen(8081, function () {
  console.log('Listening on %d.', this.address().port)
});
```

EXAMPLE

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);
}
```

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.

TUNING `HTTP.GLOBALAGENT.MAXSOCKETS`.

EXAMPLE

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)
});
```

EXAMPLE

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);
}
```

AGENT = FALSE

Setting a http.request agent to false disables socket pooling and disables keep-alive, setting `Connection: close`.

EXAMPLE

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);
}
```

SUMMARY

- `http.request` enables you to make outbound http requests.
- Understanding `http.globalAgent` and `http.globalAgent.maxSockets` behavior is important, especially in internal systems that primarily interface with the same hosts.