

1

#03 Socket Programming

CLIENT/SERVER COMPUTING AND WEB
TECHNOLOGIES

2

#Take tutorial at
<http://www.codecademy.com/en/tracks/javascript>

Intro to JavaScript

3

Language basics

- ▶ JavaScript and Java have the common C syntax, but unrelated.
- ▶ JavaScript is case sensitive
- ▶ Statements terminated by returns or semi-colons (;)
 - ▶ `x = x+1;` same as `x = x+1`
 - ▶ Semi-colons can be a good idea, to reduce errors
- ▶ "Blocks"
 - ▶ Group statements using `{ ... }`
 - ▶ Not a separate scope, unlike other languages
- ▶ Variables
 - ▶ Define a variable using the var statement
 - ▶ Define implicitly by its first use, which must be an assignment
 - ▶ Implicit definition has global scope, even if it occurs in nested scope?

4

JavaScript blocks

- ▶ Use `{ }` for grouping; not a separate scope


```
js> var x=3;
js> x
3
js> {var x=4; x}
4
js> x
4
```
- ▶ Not blocks in the sense of other languages
 - ▶ Only function calls and the *with* statement cause a change of scope

JavaScript primitive datatypes

5

- ▶ Boolean
 - ▶ Two values: *true* and *false*
- ▶ Number
 - ▶ 64-bit floating point, similar to Java double and Double
 - ▶ No integer type
 - ▶ Special values *NaN* (not a number) and *Infinity*
- ▶ String
 - ▶ Sequence of zero or more Unicode characters
 - ▶ No separate character type (just strings of length 1)
 - ▶ Literal strings using ' or " characters (must match)
- ▶ Special values
 - ▶ *null* and *undefined*
 - ▶ `typeof(null) = object;` `typeof(undefined)=undefined`

Objects

6

- ▶ An object is a collection of named properties
 - ▶ Simple view: hash table or associative array
 - ▶ Can define by set of name:value pairs
 - ▶ `objBob = {name: "Bob", grade: 'A', level: 3};`
 - ▶ New members can be added at any time
 - ▶ `objBob.fullName = 'Robert';`
 - ▶ Can have methods, can refer to *this*
- ▶ Arrays, functions regarded as objects
 - ▶ A property of an object may be a function (=method)
 - ▶ A function defines an object with method called "() "
 - `function max(x,y) { if (x>y) return x; else return y;};`
 - `max.description = "return the maximum of two arguments";`

Function Examples

7

- ▶ Anonymous functions make great callbacks


```
setTimeout(function() {
  console.log("done");
}, 10000)
```
- ▶ Curried function


```
function CurriedAdd(x){
  return function(y){ return x+y}
};
g = CurriedAdd(2);
g(3)
```
- ▶ Variable number of arguments


```
function sumAll() {
  var total=0;
  for (var i=0; i< sumAll.arguments.length; i++)
    total+=sumAll.arguments[i];
  return(total);
}
sumAll(3,5,3,5,3,2,6);
```

8

Intro to Node.js

Node.js

9

- ▶ Evented I/O for V8 JavaScript with a goal of an easy way to build scalable network programs.
- ▶ High-performance **network applications framework**, well optimized for high concurrent environments.
- ▶ It's a **command line** tool.
- ▶ Node.js uses an **event-driven, non-blocking I/O** model, which makes it lightweight.
- ▶ It makes use of **event-loops** via JavaScript's **callback** functionality to implement the non-blocking I/O.
- ▶ Programs for Node.js are written in JavaScript but not in the same JavaScript we are use to. There is no DOM implementation provided by Node.js, i.e. you **can not** do this:


```
var element = document.getElementById("elementId");
```
- ▶ Everything inside Node.js runs in a **single-thread**.

Getting Started & Hello World

10

- ▶ Install/build Node.js.
- ▶ Open your favorite editor and start typing JavaScript.
- ▶ When you are done, open cmd/terminal and type this:

```
'node YOUR_FILE.js'
```

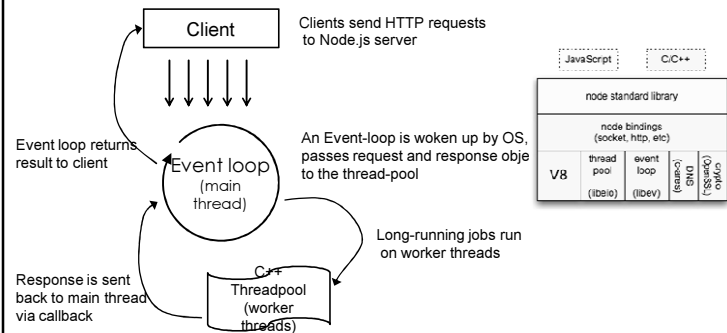
- ▶ Here is a simple example, which prints 'hello world'

```
var sys = require("sys");
setTimeout(function(){
    sys.puts("world");
}, 3000);
sys.puts("hello");
//it prints 'hello' first and waits for 3 seconds and then prints
'world'
```

Some Theory: Event-loops

11

- ▶ Event-loops are the core of event-driven programming, almost all the UI programs use event-loops to track the user event, for example: Clicks, Ajax Requests etc.



Some Theory: Non-Blocking I/O

12

- ▶ Traditional I/O


```
var result = db.query("select x from table_Y");
doSomethingWith(result); //wait for result!
doSomethingWithoutResult(); //execution is blocked!
```
- ▶ Non-traditional, Non-blocking I/O


```
db.query("select x from table_Y",function (result){
    doSomethingWith(result); //wait for result!
});
doSomethingWithoutResult(); //executes without any
delay!
```

Node.js Ecosystem

13

- ▶ Node.js heavily relies on **modules**, in previous examples **require** keyword loaded the http & net modules.
- ▶ Creating a module is easy, just put your JavaScript code in a separate js file and include it in your code by using keyword require, like:


```
var modulex = require('./module');
```
- ▶ Libraries in Node.js are called packages and they can be installed by typing


```
npm install package_name
```

 //package should be available in npm registry @ npmjs.org
- ▶ **NPM** (Node Package Manager) comes bundled with Node.js installation.

Socket Programming

WITH NODE.JS

14

What is a socket?

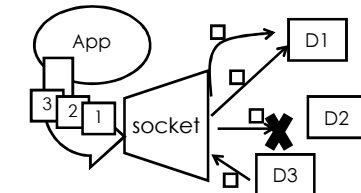
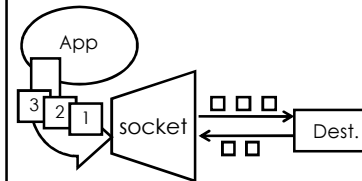
15

- ▶ An interface between application and network
 - ▶ The application creates a socket
 - ▶ The socket type dictates the style of communication
 - ▶ reliable vs. best effort
 - ▶ connection-oriented vs. connectionless
- ▶ Once configured the application can
 - ▶ pass data to the socket for network transmission
 - ▶ receive data from the socket (transmitted through the network by some other host)

Two essential types of sockets

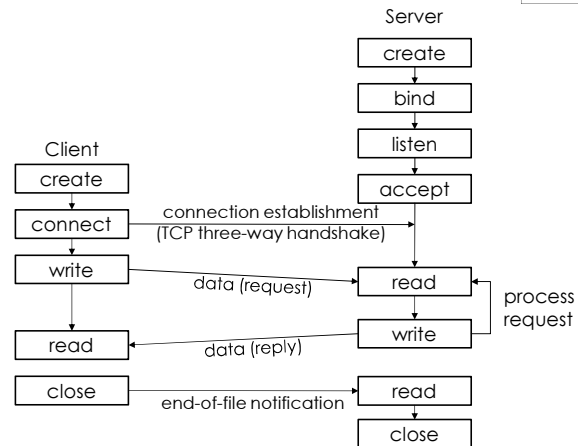
16

- | | |
|--|--|
| <ul style="list-style-type: none"> ▶ TCP Socket <ul style="list-style-type: none"> ▶ Stream-oriented ▶ reliable delivery ▶ in-order guaranteed ▶ connection-oriented ▶ bidirectional | <ul style="list-style-type: none"> ▶ UDP Socket <ul style="list-style-type: none"> ▶ Datagram-oriented ▶ unreliable delivery ▶ no order guarantees ▶ no notion of "connection" – app indicates destination for each packet ▶ can send or receive |
|--|--|



TCP Socket Operations

17



TCP Server

18

```

var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

// Create a server instance, and chain the listen function to it
// The function passed to net.createServer() becomes the event handler for the 'connection' event
// The sock object the callback function receives UNIQUE for each connection
net.createServer(function(sock) {

  // We have a connection - a socket object is assigned to the connection automatically
  console.log('CONNECTED: ' + sock.remoteAddress + ':' + sock.remotePort);

  // Add a 'data' event handler to this instance of socket
  sock.on('data', function(data) {

    console.log('DATA: ' + sock.remoteAddress + ':' + sock.remotePort);
    // Write the data back to the socket, the client will receive it as data from the server
    sock.write('You said "' + data + '"');

  });

  // Add a 'close' event handler to this instance of socket
  sock.on('close', function(data) {
    console.log('CLOSED: ' + sock.remoteAddress + ':' + sock.remotePort);
  });

}).listen(PORT, HOST);

console.log('Server listening on ' + HOST + ':' + PORT);
  
```

TCP Server (another version)

19

```

var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

var server = net.createServer();
server.listen(PORT, HOST);

server.on('connection', function(sock) {

  // We have a connection - a socket object is assigned to the connection automatically
  console.log('CONNECTED: ' + sock.remoteAddress + ':' + sock.remotePort);

  // Add a 'data' event handler to this instance of socket
  sock.on('data', function(data) {

    console.log('DATA: ' + sock.remoteAddress + ':' + sock.remotePort);
    // Write the data back to the socket, the client will receive it as data from the server
    sock.write('You said "' + data + '"');

  });

  // Add a 'close' event handler to this instance of socket
  sock.on('close', function(data) {
    console.log('CLOSED: ' + sock.remoteAddress + ':' + sock.remotePort);
  });

});
  
```

TCP Client

20

```

var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

var client = new net.Socket();
client.connect(PORT, HOST, function() {

  console.log('CONNECTED TO: ' + HOST + ':' + PORT);
  // Write a message to the socket as soon as the client is connected,
  // the server will receive it as message from the client
  client.write('I am Chuck Norris!');

});

// Add a 'data' event handler for the client socket
// data is what the server sent to this socket
client.on('data', function(data) {

  console.log('DATA: ' + data);
  // Close the client socket completely
  client.destroy();

});

// Add a 'close' event handler for the client socket
client.on('close', function() {
  console.log('Connection closed');
});
  
```

UDP Socket

21

```
var dgram = require("dgram");
var server = dgram.createSocket("udp4");

server.on("message", function (msg, rinfo) {
  console.log("server got: " + msg + " from " +
    rinfo.address + ":" + rinfo.port);
});

server.bind(41234); // server listening 0.0.0.0:41234
```

UDP Server

```
var dgram = require('dgram');
var message = new Buffer("Some bytes");
var client = dgram.createSocket("udp4");

client.send(message, 0, message.length, 41234, "localhost",
  function(err, bytes) {
    client.close();
  }
);
```

UDP Client

dgram.createSocket(type, [callback])

- type String. Either 'udp4' or 'udp6'
- callback Function. Attached as a listener to message events. Optional
- Returns: Socket object

References

22

- ▶ John Mitchell, "JavaScript"
- ▶ Vikash Singh, "Node.js: The Server-side JavaScript"
- ▶ Jeff Kunkle, "Node.js Explained", <http://kunkle.org/nodejs-explained-pres/>
- ▶ Hacksparrow.com, "TCP Socket Programming in Node.js", <http://www.hacksparrow.com/tcp-socket-programming-in-node-js.html>