Some tools for building web-based groupware

Michel Beaudouin-Lafon
mbl@lri.fr

Javascript

- More powerful than you think
- Functional: functions are first-class objects
  - **function** map(a, f) {
    var sum = 0;
    for (var i = 0; i < a.length; i++)
    sum += f(a[i]);
    return sum;
    }
  - map([1, 2, 3], **function**(x) { **return** x*x }); // 14

Javascript

- Functional: functions are first-class objects
  - **function** currify(f, x) {
    return function(y) {
    return f(x, y);
    };
    }
  - **function** add(a, b) { **return** a+b; }
  - **var** add4 = currify(add, 4);
  - add4(2); // 6

Javascript

- Object-oriented: based on the notion of **prototype** instead of the notion of class
  - **var** p = {
    x: 10,
    y: 25,
    moveBy: **function**(dx, dy) {
    this.x += dx; this.y += dy
    }
    }
  - p.moveBy(5, 10); // p = { x: 15, y: 35, moveBy: …}

Javascript

- Each object inherits from its **prototype**
  - **var** protoPoint = {
    x: 0, y: 0,
    moveBy: **function**(dx, dy) {
    this.x += dx; this.y += dy;
    }
    }
  - **var** p = {}; p.prototype = protoPoint;
    // note : the above does NOT work – see next slide
    // (a constructor is the only way to define a prototype)
  - p.x; // 0 (from the prototype)
    p.x = 10; // x is now a property of p
    p.moveBy(5, 10); // p = {x: 15, y: 10}

Javascript

- We can re-create a class-based system
  - **function** Point() { // this is a constructor function
    this.x = 0; this.y = 0
    // also assigns this.prototype
    }
  - **var** p = **new** Point(); // calls Point with a new object
  - // this is equivalent to a "method" of "class" Point
    Point.prototype.moveBy = **function**(dx, dy) {
    this.x += dx; this.y += dy;
    }
  - p.moveBy(5, 10); // p = {x:5, y:10};

Typescript

- Javascript with types
  - function fib(n: number): number { … }
- Classes
  - class A extends Object {
    constructor() { … }
    doSomething(n:number) { … }
    }
- Translates to plain Javascript
- Requires description files for external libraries
  - DefinitelyTypes web site

Node.js

- Use javascript to program web app servers
- Module system
  - Modules are installed with npm,
    the Node Package Manager: npm install <package>
  - Packages can be local or global (npm –g)
  - A package is imported with require:
    var fs = require('fs');
    fs.fileExists(…)
  - Node has built-in modules to access the OS:
    file system, processes, HTTP protocol, …

Node.js

- Node implements a reactive programming model based on events and event handlers
- Asynchronous event handling
  - **var** Emitter = require('events').EventEmitter;
    var chatRoom = new Emitter(), participants = [];
  - chatRoom.on('hello', **function**(name) {
    console.log(name, 'says Hi!');
    });
  - chatRoom.on('hello', **function**(name) {
    participants.push(name);
    });
  - chatRoom.emit('hello', 'Alice');

Node.js

- Using continuations (or callbacks) to program in an asynchronous world
  - Many node modules use callbacks, which are called when the task is actually performed:
  - **var** fs = require('fs');
    var fd = fs.open('myFile', 'w');
    if (fd) fs.write(fd, 'Hello'); // wrong: file not opened!
  - fs.open('myFile', 'w', **function** (err, fd) {
    if (! err) fs.write(fd, 'Hello', **function**(err) { …});
    });

Promises

- Facilitate asynchronous programming
- Available as library (Q, bluebird, promised-io, …)
- An asynchronous call returns a Promise object instead of taking a callback
- The then function of the promise is used to specify what to do when the call succeeds:
  - var fs = require('promised-io/fs');
    fs.readdir(path).then(function(files) {
    console.log(files.join('\n'))
    });

Promises

- Instead of being nested like callbacks, Promises are chained
- fs.open(path, 'w')
  .then(function(file) {
  return file.write('hello world');
  }).then(function(file) {
  return file.close();
  }).catch(function(err) {
  console.log('could not open', path);
  })

Node.js – Express HTTP server

- Basis of many web applications
  - **% npm install express**
  - **var** express = require('express'),
    app = express();
    app.get('/', **function**(request, response) {
    response.send('Hello World');
    });
    app.listen(8080);
  - // serve static files
    app.use(express.static(__dirname + '/public'));

Node.js – socket.io

- Communication between web page and server
- Server :
  - **% npm install socket.io**
  - **var** io = require('socket.io').listen(http);
    io.on('connection', **function** (client) {
    client.on('msg', **function** (data) {
    // send to all other clients
    client.broadcast.emit('msg', data);
    });
    });

Node.js – socket.io

- Client (web page)
  - <script src="/socket.io/socket.io.js"></script>
    <script>
    var client = io ();
    client.on('connect', **function** () {
    client.emit('msg', 'Hello');
    });
    client.on('msg', **function**(data) {
    $('#chat').append(data);
    });
    </script>

Node.js - sharejs

- Shared text strings and objects synchronized through operational transformation (experimental)
  - **var** client = require('share').client;
    client.open('myDoc', 'text', url, **function**(err, doc) {
    doc.insert('Hello world', 0);
    doc.on('change', **function**(op) {
    console.log(doc.snapshot);
    });
    });

dnode

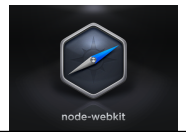
- Remote function call for node.js and browser
- Server:
  - **var** dnode = require('dnode');
    var server = dnode({
      listFiles**: function** (s, cb) {
        fs.readdir(path, function(err, files) {
          cb(files);
        });
      }
    });
  - server.listen(5004);

dnode

- Client:
  - **var** dnode = require('dnode');
    var d = dnode.connect(5004);
  - d.on('remote', **function** (remote) {
      remote.listFiles('.', **function** (files) {
        console.log(files.join('\n');
        d.end();
      });
    });
  - $ node server.js &
    $ node client.js
    (list of files in current directory of server)

Node-webkit

- Combines a node.js server and a web browser
- Access to desktop: files, menus, …
- Desktop-based web applications
- Avoids web protocol issues (same-origin, …)
- Global namespace shared between node server and every web window

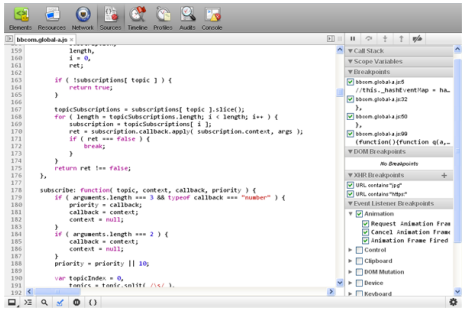


Node-webkit

- A web page that lists the files in the current directory
  - <script>
    var fs = require('fs'); // node module in HTML file!
    fs.readdir('.', **function**(err, files) {
      files.map(**function**(filename) {
        $('#list').append('
'+filename+'
');
      });
    });
    </script>
  - <ul id='list'>

Debugging

- In the browser: developer tools

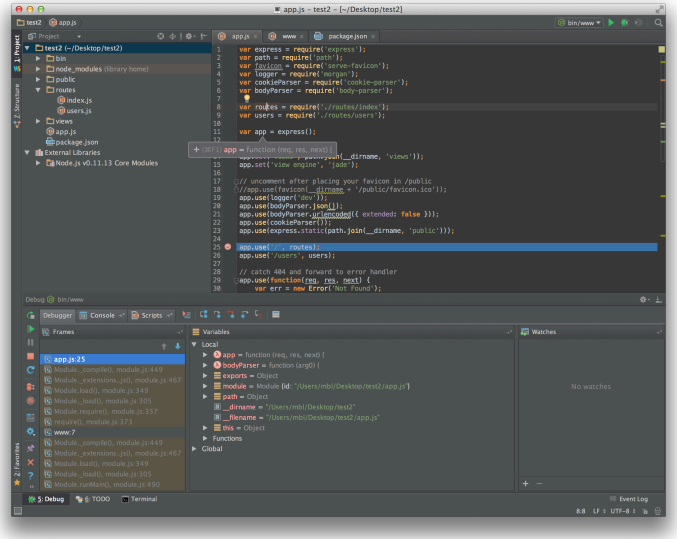


Debugging

- Node.js
  - Built-in debugger:
    % node debug myscript.js
    (add calls to debugger in the script)
  - Remote debugging with web inspector
    - Install the node-inspector package (once)
      % npm install –g node-inspector
    - Run the script in debug mode
      % node --debug-brk myscript.js
      % node-inspector &
    - Browse to http://localhost:8080/debug?port=5858

WebStorm

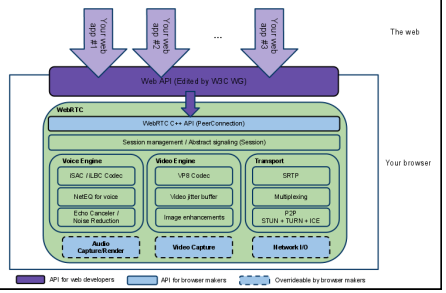
- Integrated Development Environment (IDE)
- Project manager
- Editor
  - Multiple cursors
  - Refactoring
  - Linting
- Debugger
- Supports Typescript



The (near) future: WebRTC

- Set of APIs for real-time communication
- Supports audio, video, data
- Web APIs as well as native APIs

- But:
- Complex
- Still unstable
- Chrome & Firefox only



Project ideas

- Shared drawing tool (doodling)
- Shared browsing
- Shared annotation of any web page
- Multi-room chat
- …