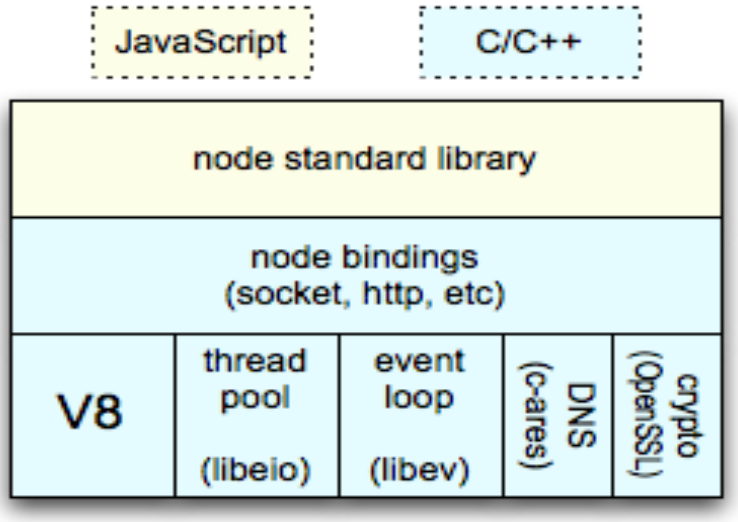
Introduction

http://nodejs.org/about/

Introduction to Node

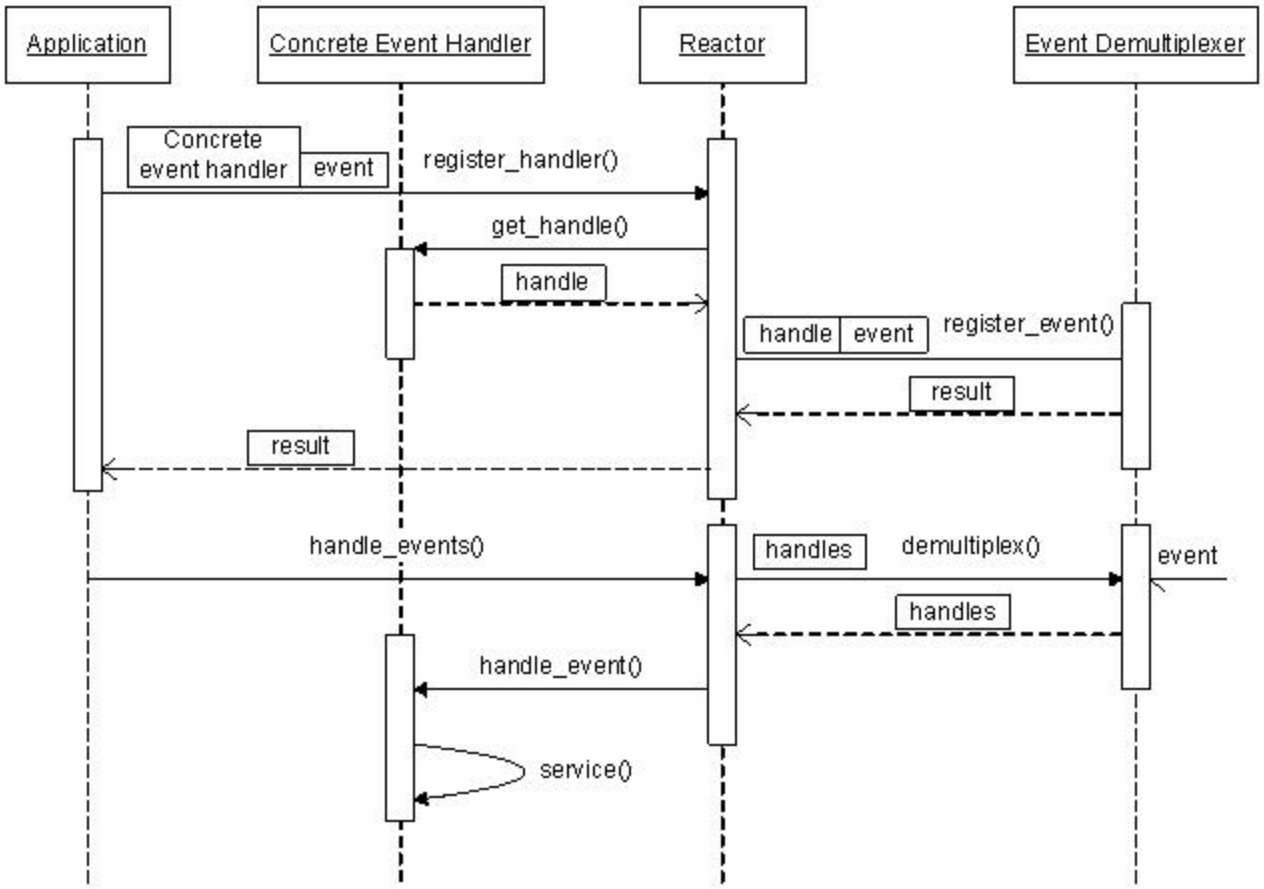
What is Node ?

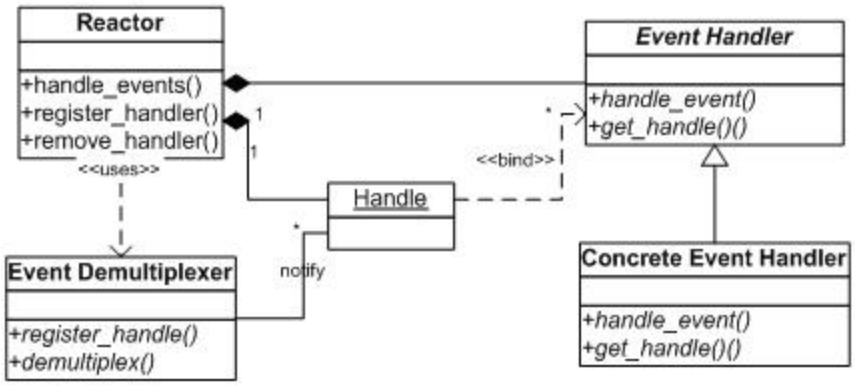


Node is an open source toolkit for developing server side applications based on the V8 JavaScript engine. Like Node, V8 is written in C++ and is mostly known for being used in Google Chrome.

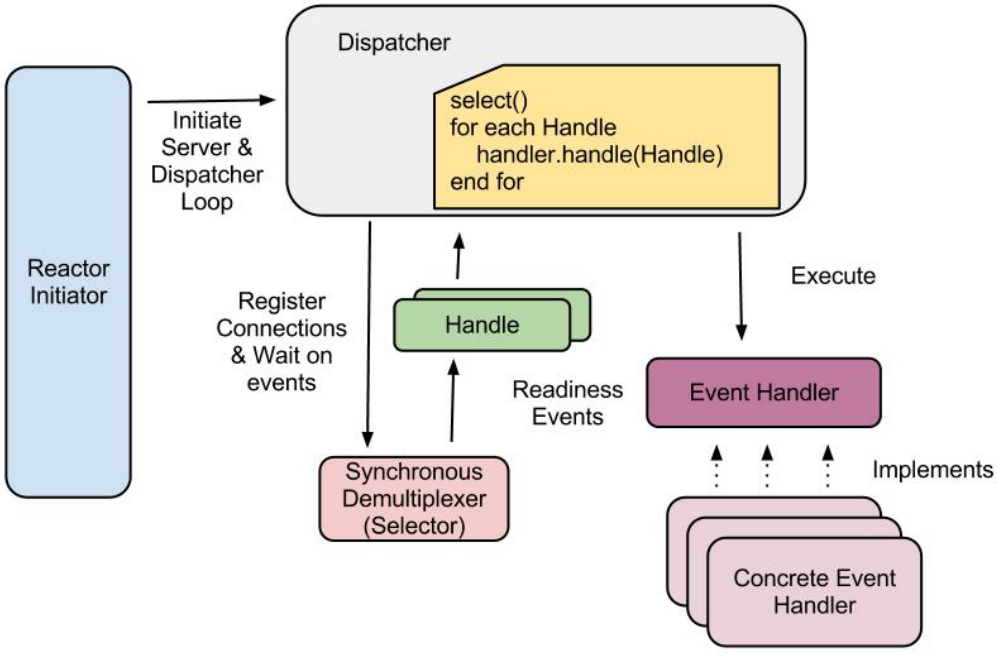
Node is part of the Server Side JavaScript environnement and extend JavaScript API to offer usual server side functionalities.

It's spirit is similar to Twisted for Python and EventMachine for Ruby http://timetobleed.com/eventmachine-scalable-non-blocking-io-in-ruby/.





Reactor Pattern http://chamibuddhika.wordpress.com/2012/08/



Node's Goal ?

It's goal is to offer an easy and safe way to build high performance and scalable network applications in JavaScript.
Those goals are achieved thanks it's architecture:

- Single Threaded :

Node use a single thread to run instead of other server like Apache HTTP who spawn a thread per request, this approach result in avoiding CPU context switching and massive execution stacks in memory. This is also the method used by nginx and other servers developed to counter the C10K problem.

- Event Loop :

Written in C++ using libuv library, the event loop use epoll for scalable event notification mechanism.

- Non blocking I/O :

Node avoid CPU time loss usually made by waiting for an input or an output response (database, file system, web service, ...) thanks to the full-featured asynchronous I/O provided by libuv library.

These characteristics allow Node to handle a large amount of traffic by handling as quickly as

possible a request to free the thread for the next one.
Node has a built-in support for most important protocols like TCP, DNS, and HTTP (the one that we will focus on). The design goal of a Node application is that any function performing an I/O must use a callback. That's why there is no blocking methods provided in Node's API.

JavaScript

Being an Event Driven Language, Javascript is the most suited to develop on the Node's "Event Loop" architecture. Node's applications really use javascript's strengths like anonymous functions and closures.

```
/ /Loading required modules
var http = require('http');

var SERVER_PORT = 8124;

// Creating HTTP Server
var server = http.createServer(function(request, response){
   // Called each time a request is made.
   response.writeHead(200, {'Content-Type': 'text/plain'});
   response.end('Hello World\n');
});

// Starting the server
server.listen(SERVER_PORT);

console.log('Server running on port : ' + SERVER_PORT);
```

This is quiet simple, we just create a http server by requiring the HTTP module and calling the createServer method. The callback method will be executed each time a request comes in.

To start the server call :
$ node server.js
Server running on port : 8124
We can now access to the server by using http://localhost:8124 and see the "Hello World".

Adding Socket.IO

http://socket.io/#how-to-use

The next step is to use Socket.IO to handle long-terms connections, replace the server.js content by the following one:

```
// Loading required modules
var http = require('http'),
    io = require('/path/to/socket.io');

var SERVER_PORT = 8124;

// Creating HTTP Server
var server = http.createServer();

// Starting the server
server.listen(SERVER_PORT);

// Attaching Socket.IO to the HTTP Server
var socket = io.listen(server);

console.log('Server running on port : ' + SERVER_PORT);
```

Two more lines, that's the only things we need to add to attach the Socket.IO module to the http server.
You might have noticed that the callback function of the http.createServer() has been removed, this is due to our use case where the server does not serve data to the client when they make a request but directly push data when an event is raised.
We can put a callback to the io.listen() method, it will be fired each time a request comes from the Socket.IO client side library.

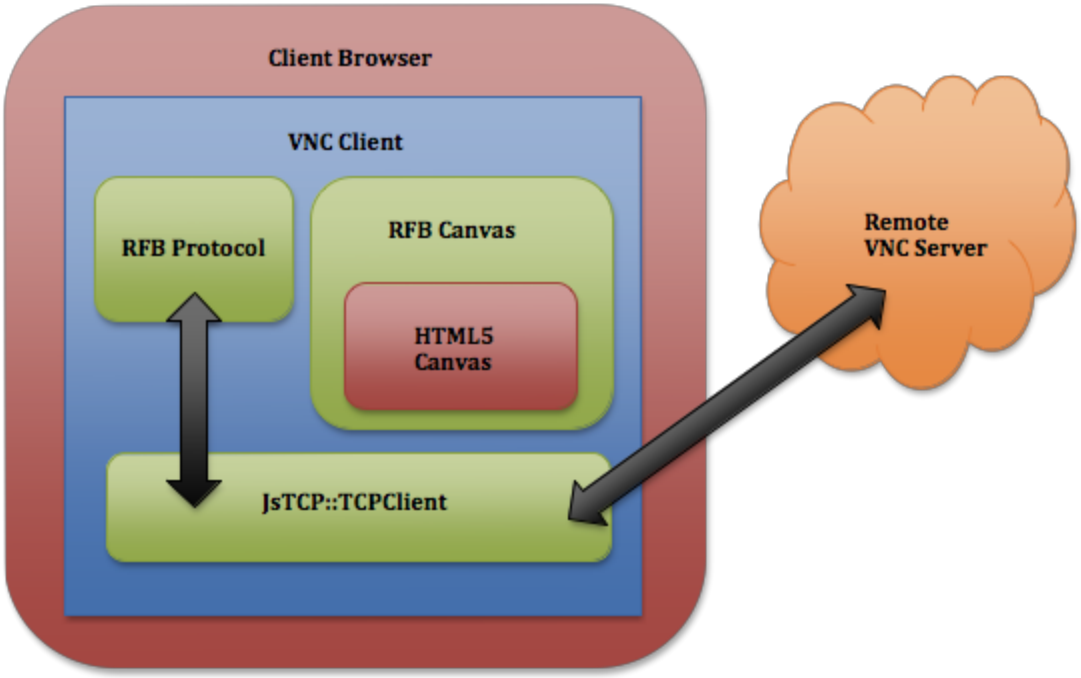
```
var socket = io.listen(server,function(client){
    // new client connected !
});
```

Socket IO example : VNC client in 24 hours
http://engineering.linkedin.com/javascript/vncjs-how-build-javascript-vnc-client-24-hour-hackday

Architecture

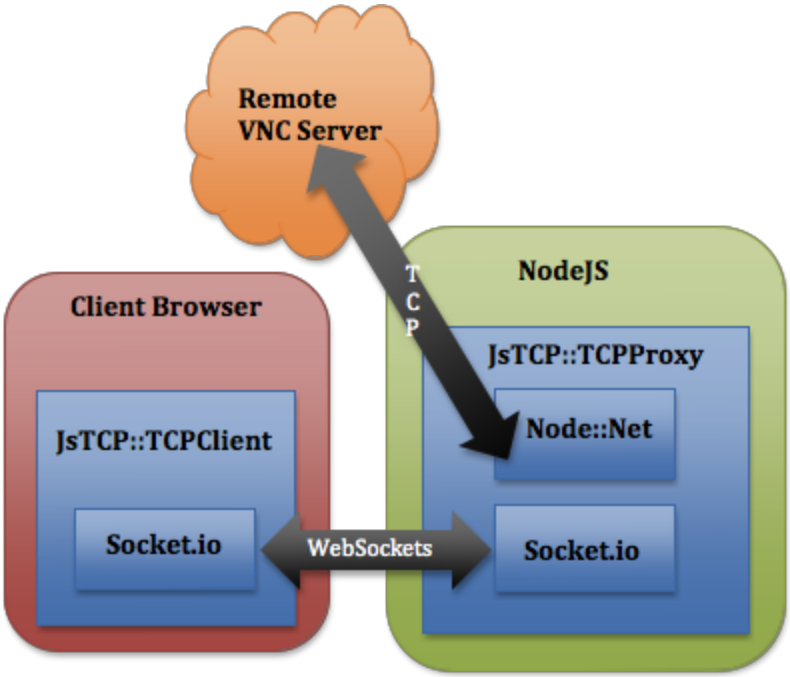
In order to get a VNC client working the browser, we needed the following pieces:

1. A way to connect to a remote server: the only way to do this within a browser is to use a proxy. We built the proxy using Node.js and established a persistent connection to it using socket.io.
2. Implement the RFB protocol: with Node.js abstracting away the TCP connection, the next step was to use the RFB protocol to communicate with the remote server.
3. Render the image in the browser: once the server was sending us data, we used the HTML5 canvas element to render it in the browser. This worked well, as we can conveniently transfer the 32bit pixel data in row major order directly to the canvas.



tcp.js: A TCP proxy written on top of Node.js and Socket.IO

The first step was establishing connectivity between the browser, the Node.js proxy, and a remote VNC host. Communication between the browser and Node.js is handled easily using socket.io. However, communication between Node.js and the VNC host is more complicated: it requires TCP.



We recognized the inevitable complexity of our hack and chose to spend a little extra time up front to cleanly abstract all of our connectivity concerns. Because we're using the same language in both the browser and the server (JavaScript), the result was a very clean abstraction that we called tcp.js. It's dead-simple to use:

```
var host = "127.0.0.1";
var port = 5900;

var sock = new TCPClient(host,port);

sock.on("connected", function() {
  log("connected to " + host + ":" + port);
  sock.send("Hello from a browser!");
});

sock.on("closed", function() {
  log("The connection has closed :(");
});

sock.on("data", function(msg){
  log("data arrived: " + msg);
});

sock.connect();
```

V8 Design
https://developers.google.com/v8/design

Hidden Classes
http://v8-io12.appspot.com/#30
V8 internally creates hidden classes for objects at runtime
Objects with the same hidden class can use the same optimized generated code

Express - Web framework for node.js
https://npmjs.org/package/express

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
```

```
  res.send('Hello World');
});

app.listen(3000);
```

MVC example
https://github.com/visionmedia/express/tree/master/examples/mvc/controllers/user

Firebird Node

https://github.com/hgourvest/node-firebird

Example Application
http://mariuz.android-dev.ro/atom-reader/feedread.html

Bibliography
http://blog.zenika.com/index.php?post/2011/04/10/NodeJS
http://s3.amazonaws.com/four.livejournal/20091117/jsconf.pdf

Why node js
Pure JavaScript driver is faster than node C++ wrapper for node.js

http://www.firebirdnews.org/?p=7271