NODE.JS
SERVER SIDE JAVASCRIPT

Introduction Node.js

Node.js was created by **Ryan Dahl** starting in 2009.

For more information visit:
http://www.nodejs.org

What about Node.js?

1. JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language
2. Servers are normally thread based but Node.JS is "Event" based. Node.JS serves each request in a Evented loop that can able to handle simultaneous requests.
3. Node.JS programs are executed by V8 Javascript engine the same used by Google chrome browser

HOW DOES IT DIFFER?

From Node.js white paper:
"Node is similar in design to and influenced by systems like Ruby's event machine or Python's twisted. Node takes the event model a bit further—it presents the event loop as a language construct instead of as a library."

WHAT CAN YOU DO WITH NODE ?

- It is a command line tool. You download a tarball, compile and install the source.
- It lets you Layered on top of the TCP library is a HTTP and HTTPS client/ server.
- Node provides a JavaScript API to access the network and file system.

WHAT CAN'T DO WITH NODE?

- Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.
- Node can't run on GUI, but run on terminal

INTERMEZZO: EVENT-BASE PROGRAMMING

Event-based programming

- Code is executed upon activation of events
  - Events = type entities
  - Events are generated by external actions (e.g. mouse click)
- Procedural (or functional) programming requires explicit calls following the control flow

Publish-subscribe

- Publishers public structured events to an event service
- Subscribers express interest in particular events through subscriptions
- Match: subscriptions against published events
  - Delivery of event notifications to subscribers

Applications

- Financial Information Systems
- Live feed (real time data)
- Ubiquitous computing (the internet of things)
- Monitoring applications

Advantages

- Heterogeneity
  - Interoperability is made easier
- Asynchronicity
  - Publishers DO NOT have to synchronize with subscribers when sending events

THREADS VS EVENT-DRIVEN

Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/ select or O_NONBLOCK) environments

Why node.js use event-based?

In a normal process cycle the webserver while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn't wait for the IO operation to complete while it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

Standard web programming


```
var result =
  db.query("select * from T");
// use result
```

Requires multiple execution stacks

Function Call-Back

```
db.query("select..", function (result) {
  // use result
});
```

allows the program to return to the event loop immediately.

SYSTEM REQUIREMENTS

- Node runs best on the POSIX-like operating systems. These are the various UNIX derivatives (Solaris, and so on) or work a likes (Linux, Mac OS X, and so on).
- While Windows is not POSIX compatible, Node can be built on it either using POSIX compatibility environments (in Node 0.4x and earlier).

Node.js VS Apache

Platform	Number of request per second
PHP (via Apache)	3187,27
Node.js	5569,30

A web server

```
var http = require('http');
http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

The code first loads the HTTP library. It then creates a new server, and sets a callback function that will be called whenever a browser connects to the server.
The server is then set to listen to incoming connections at 127.0.0.1:1337 using the listen command

issues

Problems:
The HTTP library is very low level.
It allows you to operate directly on the HTTP request (req) and response objects (res)
but it doesn't give you any higher-level abstrations that would help you build a web application.

Some drawbacks.
You only get to specify a single callback function.
You have to retrieve the URL from the req parameter and match it with your list of available URLs manually.
You need to set response headers manually.

WEB Frameworks


```
var express = require("express");
var app = express();
app.get("/", function(req, res) {
        res.send("Hello world!");
        res.end();
}).listen(1337);
console.log('Server running at http://127.0.0.1:1337/');
```

More info

- http://devlup.com/javascript/node-js-server-side-javascript/2198/
- http://nodejs.org
- http://net.tutsplus.com/tutorials/javascript-ajax/this-time-youll-learn-node-js/