Web Technologies

Lecture 12

Node.js

Javascript beyond browsers

- **Javascript** was first used in browsers for client side programming
- However, Javascript is a **complete language** which can be used in man **contexts** to achieve the same things as in other languages
- **Node.js** is just another context allowing Javascript to run in the backend outside a browser

Node.js

- Relies on Google's **V8 VM**
  - Same Javascript runtime environment as the one used by Google Chrome
  - 1st version released on Sep 2, 2008
- Comes with lots of **modules** which help developers to avoid writing code from scratch
- Node.js
  - **Runtime + library**
- *Create real-time websites with push capability*

Node.js popularity

Request handling

- Traditional web servers handle multiple requests:
  - Synchronous
  - Fork processes
  - Threads
- These block thread execution
  - Example: eventually the server runs out of available PHP processes
- Memory is used more and more
  - 2Mb per request on an 8GB RAM machine → 4,000 concurrent requests
  - In Node.js it can be as big as 1M concurrent connections

Node.js request handling

- Single process
- Event driven loop
  - Each request gets to the loop ring with a callback method
  - Non blocking & scalable

Communicating with the server

- Traditionally, in client-server architectures, clients send requests to servers which reply back
  - Client polls server
  - If server is busy response time can be long
  - Lot of network traffic
- Solution
  - Long polling
  - Server side push
  - Websockets

Long polling

- Client sends request to server
- If response is unavailable on server side server does not reply with an empty message
- Instead it keeps connection open and replies as soon as response becomes available
    - Emulates server push
- Client immediately sends back another request

Server sent events

- SSE
- Similar to long polling but client does not send back a request after receiving the update from the server
  - Instead it keeps on receiving updates for the same request

Websockets

- HTML5
- **Full duplex TCP connection** independent on the HTTP request/response
  - Server can send data to clients without any HTTP requests
- **Publish and subscribe** to sockets for any change in data
- Lightweight
  - No HTTP request headers
  - Much more websocket connections than HTTP connections

Node.js performance

Source: http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/

Hello world

- Node.js binary: https://nodejs.org/en/download/
- Run code by typing: *node hello-console.js*
- Source: http://howtonode.org/hello-node
- Filenames end with .js
  - Just like regular Javascript files
- Example:

```
// Call the console.log function
console.log("Hello World");
```

Simple web server

```
// Load the http module to create an http server
var http = require('http');

// Configure our HTTP server to respond with Hello World to all requests
var server = http.createServer( function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});

// Listen on port 8000, IP defaults to 127.0.0.1
server.listen(8000);

// Put a friendly message on the terminal
console.log("Server running at http://127.0.0.1:8000/");
```

Dispatcher

- You may need a *dispatcher* to handle different URLs:

```
var dispatcher = require('httpdispatcher');
…
var server = http.createServer( function (request, response) {
        dispatcher.dispatch(request, response);
});
…
dispatcher.setStatic('resources');
//A sample GET request
dispatcher.onGet("/page1", function(req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Page One');
});
//A sample POST request
dispatcher.onPost("/post1", function(req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Got Post Data');
});
```

When to (not) use Node.js

- Not for CPU intensive applications
  - Web application with relational DB
  - No heavy server-side computation/processing
    - Fibbonacci computation
- Good for fast scalable networked applications
  - Chat
  - API on top of an Object DB (e.g., MongoDB)
  - Queued inputs (e.g., message queues such as RabbitMQ, ZeroMQ)
  - Data streaming
  - Proxy servers
  - Real-time traffic/system monitoring

What’s next?

- Cloud computing