Announcements

• HW4 regrade requests today 2-4pm in Upson 5132.
### Why JavaScript?

<table>
<thead>
<tr>
<th>Server-side Languages</th>
<th>Client-side Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP, Ruby, Python, Perl, C, Java, etc.</td>
<td>JavaScript, VBScript</td>
</tr>
</tbody>
</table>

- AJAX made JavaScript in high demand
- Most web developers know JavaScript
- Evangelists, like Douglas Crockford, helped push the “good parts” of the language
# JavaScript Engines

<table>
<thead>
<tr>
<th>SpiderMonkey</th>
<th>Mozilla, open-source, written in C</th>
<th>Interpreter, used in Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhino</td>
<td>Mozilla, open-source, written in Java</td>
<td>Compiles JavaScript to Java</td>
</tr>
<tr>
<td>V8</td>
<td>Google, open-source</td>
<td>Compiles JavaScript to native code</td>
</tr>
</tbody>
</table>
What is Node?

- Platform for writing server-side applications
- Libraries on top of V8
- Created by Ryan Dahl in 2009, maintained by Joyent
- Built in HTTP server library (i.e., run a web server without Apache)
How to Install Node

Download binaries:

http://nodejs.org/download/

Download source:

$ git clone https://github.com/joyent/node.git
$ cd node
$ ./configure
$ make
$ sudo make install
# Create a javascript file
$ cat hello.js
call.log("Hello")

# Invoke node with name of your file:
$ node hello.js
Hello

# Read Eval Print Loop (REPL):
$ node
> 1 + 3
4
> .help
Server-Side JavaScript Object Model

- On the client-side, the root global object was a Window
- On the server-side, the root global object is a process

```javascript
> process
{ title: 'node',
  version: 'v0.10.20',
  moduleLoadList:
    [ 'Binding evals',
      ...
    ],
  pid: 882,
}
> process.pid
882
```
Event-Loop

- JavaScript already organized around events (e.g., onchange, onclick)
- Node philosophy: I/O should be non-blocking
- HTTP requests, I/O, databases queries run separately, emit event when finished
- Callbacks are used to process the events (often cascading)
Sleepy Hello World

- Node version doesn’t “sleep”
- After 2 seconds, a timeout event triggers a **callback**
Asynchronous File Copy

```
var fs = require('fs')
src = process.argv[2]
fs.readFile(src, 'utf8', function (err, data) {
  if (err)
    throw err;
  fs.writeFile(dst, data, 'utf8', function (err) {
    if (err)
      throw err;
  });
});
```
Reading From stdin

```javascript
process.stdin.resume();
process.stdin.setEncoding('utf8');

process.stdin.on('data', function(chunk) {
  process.stdout.write('data: ' + chunk);
});

process.stdin.on('end', function() {
  process.stdout.write('end');
});
```
Asynchronous

!= Concurrent

• Code runs in response to events, not order of code in program
• Everything runs in a “single thread”
• Only one thing is happening at a time
• When an function exits, process the next event sequentially
Remembering State

```javascript
function printLater(message, timeout) {
    // Use closure to remember state!
    function handle() {
        console.log(message);
    }
    setTimeout(handle, timeout);
}
printLater("Hello Robert", 100);
printLater("Hello Jim", 50);
```

- Non-blocking code is usually difficult to write because you need to maintain state
- JavaScript *closures* do that for you
HTTP Server

server.js:

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

Run your server:

```
$ node server.js
```

In your browser:

![Image of browser showing Hello World](image_url)
Client Side

In an HTML form:

```html
<form action="http://127.0.0.1:1337">
```

jQuery post:

```javascript
$.post( "http://127.0.0.1:1337", ... );
```

jQuery get JSON:

```javascript
$.getJSON("http://127.0.0.1:1337",...);
```
HTTP Server

Passed to the server’s event listener for the request event

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

console.log('Server running at http://127.0.0.1:1337/');
```

Modify the response object.
Single method to write the body and close the connection.
TCP Echo

```javascript
var net = require('net')
var server = net.createServer()
server.on('connection', function(client) {
  client.write('connected')
  client.on('data', function(data) {
    client.write(data)
  })
})
server.listen(9000)
```

```
$ telnet 127.0.0.1 9000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '\]'.
connected
hi
```
Chat with Multiple Clients

```javascript
var net = require('net')
var server = net.createServer()
var clients = []
server.on('connection', function(client) {
    clients.push(client)
    client.on('data', function(data) {
        for (var i = 0; i < clients.length; i+=1) {
            if (client !== clients[i]) {
                clients[i].write(client.name + " says " + data)
            }
        }
    })
})
```
Last Slide

• Next lecture: Express, Jade