CS 5142
Scripting Languages

10/14/2013

Server-Side Javascript

Announcements

- HW4 regrade requests today 2-4pm in Upson 5132.

Why JavaScript?

Server-side Languages	Client-side Languages
PHP, Ruby, Python Perl, C, Java, etc.	JavaScript ~~VBScript~~

- 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

SpiderMonkey	Mozilla, open-source, written in C	Interpreter, used in Firefox
Rhino	Mozilla, open-source, written in Java	Compiles JavaScript to Java
V8	Google, open-source	Compiles JavaScript to native code

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
```

How to Write and Run Code

```
# Create a javascript file
$ cat hello.js
console.log("Hello")

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

# Read Eval Print Loop (REPL):
$ node
> 1 + 3
4
> .help
```

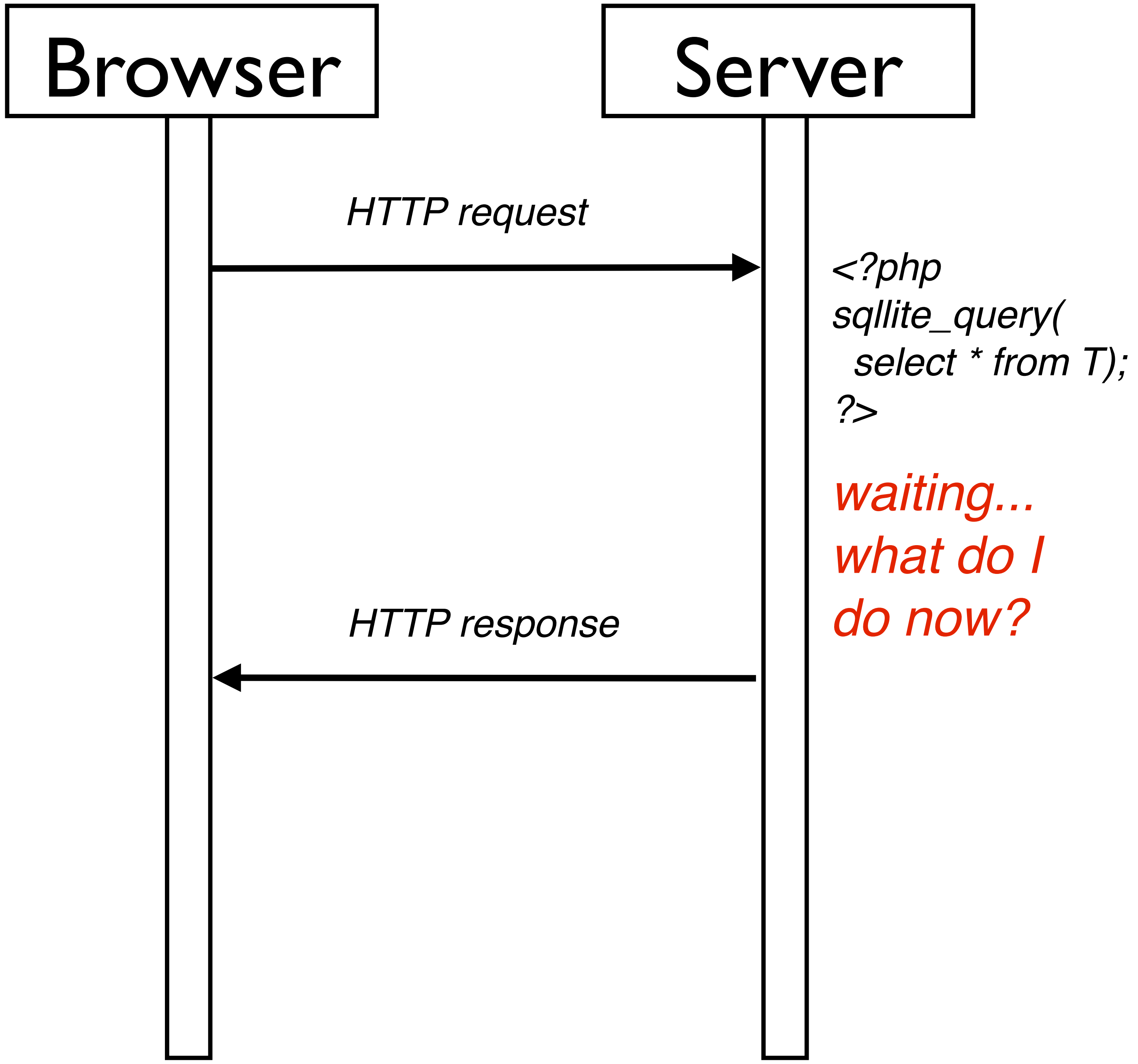
Server-Side JavaScript Object Model

```
> process
{ title: 'node',
  version: 'v0.10.20',
  moduleLoadList:
   [ 'Binding evals',
  ...
  pid: 882,

> process.pid
882
```

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

Event-Loop



```
<?php
sqllite_query(
  select * from T);
?>
```

- 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

```php
<?php
  echo ‘hello’;
  sleep(2);
  echo ‘world’;
?>
```

```javascript
setTimeout(
  function () {
    console.log("world");
  },2000);

console.log("hello")
```

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

Asynchronous File Copy

argv[0] = node
argv[1] = *scriptname*

Callback

Cascading callback

```
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

```
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');
});
```

paused by default

listen for data event

listen for end event

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

```
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

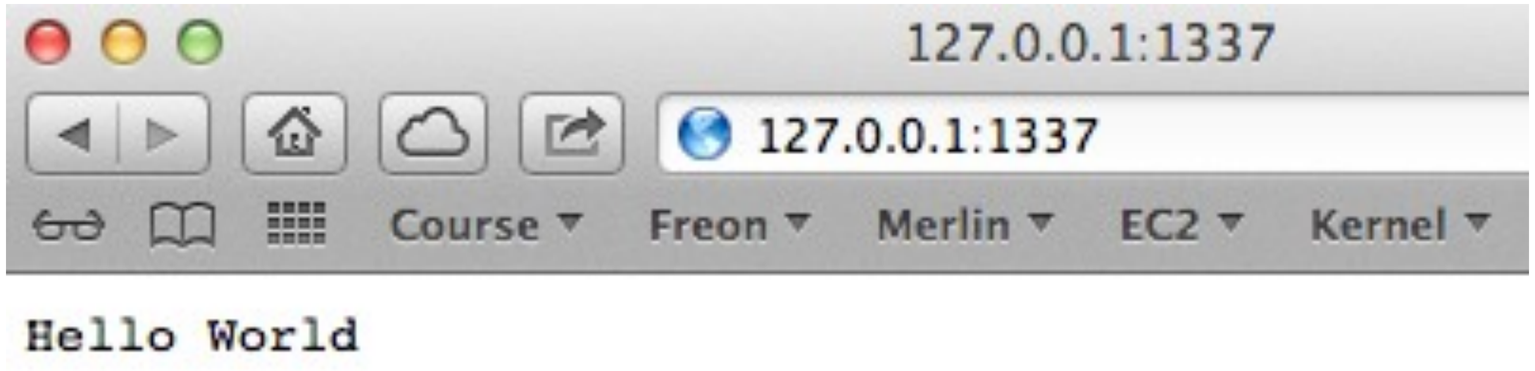
server.js:

```
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/');
```

Run your server:

```
$ node server.js
```

In your browser:



```
Hello World
```

Client Side

In an HTML form:

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

jQuery post:

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

jQuery get JSON:

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

HTTP Server

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

```
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/');
```

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

TCP Echo

```
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 '^]'.
connectedhi
hi
```

Chat with Multiple Clients

```
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