Server-side Form Handling

IERG4210 Lecture 5

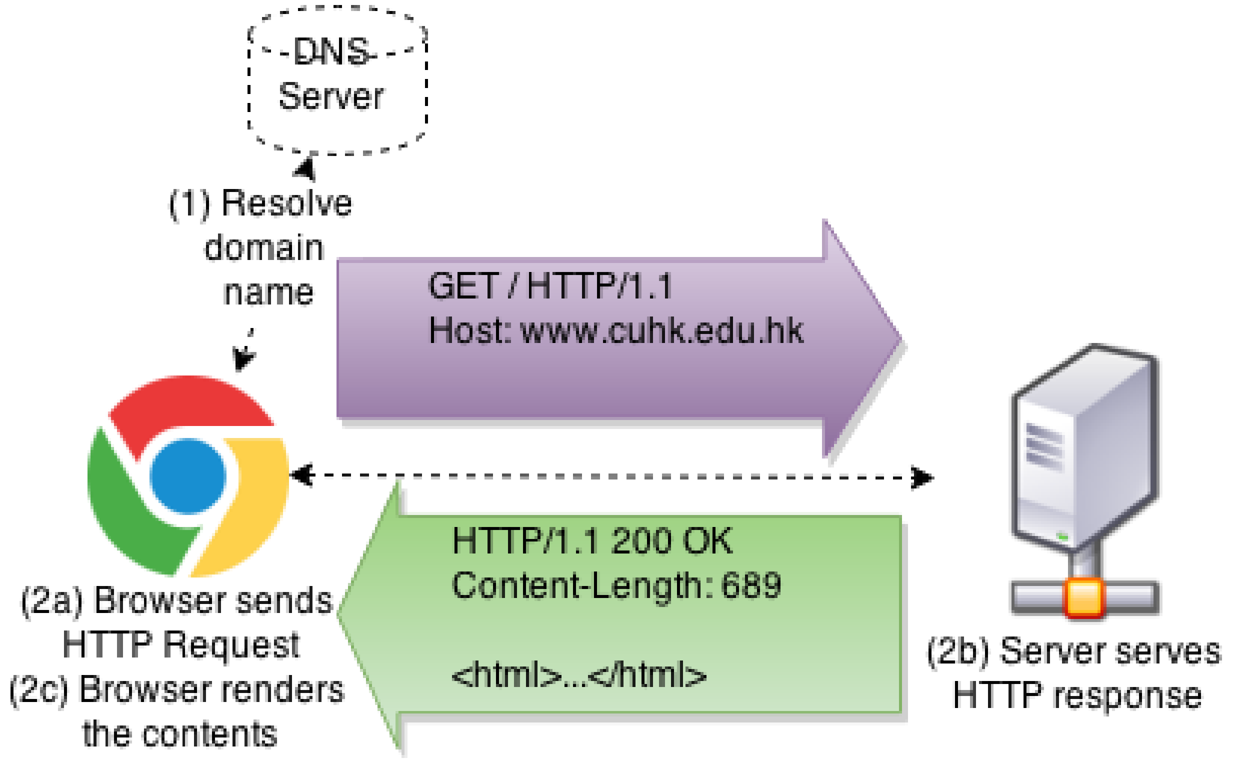
Dr. Adonis Fung
Information Engineering, CUHK
Paranoids, Yahoo!

IERG4210 Web Programming and Security, 2015 Spring.
Offered by Dept. of Information Engineering, The Chinese University of Hong Kong.

Agenda

- Web Server Comparisons
- Fast and Scalable Web Application
  - Application Framework using Express
  - Templated Rendering using ExpressHandlerbars
- Server-side Form Handling
  - Retrieval of Request Parameters
  - Input Validations
  - Output Formats (HTML/JSON/XML)
  - Output Sanitizations (to be detailed next week)

Web Server

- Recall: HTTP is a text-based application-layer protocol that defines how content is requested from a client application and served by a web server.
- 

Web Server Choices

- Market share: Apache > IIS > Nginx >> Node
-

	Apache	IIS	Nginx	Node
Concurrency Model	threaded / process-oriented approach (inefficient memory use and scheduling)		asynchronous event-driven approach (no blocking, more scalable)	
Common Programming Language	PHP	ASP.NET/PHP	None/PHP	JavaScript
Design Goals	full-featured generic purpose		less features/footprint specific purpose (e.g., cache/proxy)	specific purpose app framework bundled w/web server
OS	mostly *nix	M$ windows	mostly *nix	mostly *nix
Open-source	open-source (no upfront cost)	proprietary (requires licensing)	open-source (no upfront cost)	open-source (no upfront cost)

Fast and Scalable WebApp using Node.js

- Event-driven architecture and non-blocking I/O API from ground-up
  - Model best for slow/blocking (network) I/Os, now asynchronously handled
  - Benefits and Concepts covered in last lecture and reading
  - Resource efficient. Can easily scale up with Amazon Beanstalk
  - Unlike optionally async with a library (e.g., Twisted in Python)
- Code reuse due to single language across both client and server side
  - Developer-friendly to JS/AJAX folks
- Fast v8 JavaScript Engine (as in Chrome)
  - JIT Compilation: Compiled to binary, and runs like executable
  - Memory more efficient and Faster than vanilla PHP (FB made HipHop VM)

Node.js HTTP Server

- HTTP is a first class citizen
  - Built-in HTTP library, doing away with Apache/IIS/Nginx
- A Sample Hello World HTTP server:

SERVER.JS

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<h1>Hello World</h1>');
}).listen(3000, "localhost");
```

SHELL

```
$ node server.js
```

- Event-driven paradigm
  - Create a HTTP server binded to port 3000 of localhost
  - Execute the callback per request

NPM - Pacakge Manager for Node.js

- Package Manager: Simplify deployment by auto-resolving dependencies
  - A specification standard of (dev)dependencies
    devDependencies refers to those needed only during development but not runtime
    - NPM: `package.json` specifies the required packages
  - A repository to host packages published by developers
    - NPM: https://www.npmjs.com/
  - A CLI toolset to recursively install/manage required dependencies and versions
    - NPM: `npm install` looks for `./package.json`, and recursively installs all dependency packages
    - NPM: `npm install <packageName> --save` downloads packageName and marks it dependent in your package.json
    - NPM: `npm install <packageName> --save-dev` downloads packageName and marks it devDependent in your package.json
- Note: (1) `pip` is the package manager for python; (2) Amazon Beanstalk runs `npm install` on your package during remote deployment;

Web Application using ExpressHandlebars

- Delivering as an Interactive Workshop
  - 25 min. Get prepared with your laptop
  - Some walkthroughs and demonstrations
  - Teaching teams then workaround to help
- Quick Started:
  - Prerequisite:
    install Node.js and init a new project
  - Install dependencies:
    ```
    npm install express body-parser express-
    handlebars --save
    ```
  - Sample code using ExpressHandlebars

Modularize your files

- Modularizing Static Functions into a separate file

LIB/IERG4210-STATIC.JS

```
exports.hello = function () { console.lo
```

APP.JS

```
var ierg4210 = require('./lib/ierg4210-static');
ierg4210.hello();
```

- Modularizing an Object into a separate file

LIB/IERG4210-OBJECT.JS

```
module.exports = function Person(name) {t
Person.prototype.getName = function() {return this.name;};
```

APP.JS

```
var Person = require('./lib/ierg4210-object');
var peter = new Person('Peter', 'M');
peter.getName();
```

- Tips: Modularize your routes. Keep your server.js succinct.

Server-side Form Handling

- Recall how you send a request header/body: Lecture 4 slide #6
- Retrieving request parameters from path, querystring, and body

```js
// for parsing application/x-www-form-urlencoded
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({extended:true}));

app.use('/process/:action(\w*)', function(req, res){
  // refers to params named "action" (e.g., /process/abc)
  console.log('params: ' + req.params.action);
  // refers to query string named "q" (e.g., /process?q=abc)
  console.log('query: ' + req.query.q);
  // refers to POST parameter named "q"
  console.log('post: ' + req.body.q);
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<h1>Hello World!</h1>');
});
```

Ref: Express API

Input Validations

- Root Cause: User Inputs are always untrusted!
  - Unexpected user inputs could lead to unauthorized executions
  - Input validation problems ranked constantly high in 2007, 2010, 2013
    by OWASP Top 10 Application Security Risks
  - In 2013, those include: A1 Injection, A3 Cross-site Scripting, A4 Insecure Direct Object References
- Fundamental Defences: Restrict user inputs
  - Input Validations - pattern restrictions
    - blacklist malicious patterns: harder to sort out what's bad
    - whitelist acceptable patterns: easier to pin down the allowed space, hence whitelist whenever possible
  - Input Sanitizations/Encoding - escape sensitive chars. to stop executions
    - Type casting: `untrusted = parseInt(untrusted)` enforces an integer in JavaScript

Server-side Input Validations and Sanitizations

JS

```
app.use('/process', function(req, res){
  res.writeHead(200, {'Content-Type': 'text/html'});

  // input validation
  var InputRegExp = /^\w+$/;
  if (![req.body.firstname, req.body.lastname]
       .every(function(input){return InputRegExp.test(input);})) {
    res.end('Incorrect Inputs');
    return;
  }
  // input sanitization
  req.body.age = parseInt(req.body.age);


  // further processing only after proper validations and sanitizati
ons
  res.end('<h1>Hello, ' + req.body.firstname + req.body.lastname + '
 </h1>');
```

Server-side Input Validations and Sanitizations Using ExpressValidator

JS

```
var app = require('express')(),
  bodyParser = require('body-parser'),
  expressValidator = require('express-validator');

app.use(bodyParser.urlencoded({extended:true}));
app.use(expressValidator());

app.use('/process', function(req, res){
  res.writeHead(200, {'Content-Type': 'text/html'});

  // input validation
  req.checkBody('firstname', 'Invalid first name').notEmpty().isAlphanumeric();
  req.checkBody('lastname', 'Invalid last name').notEmpty().isAlphanumeric();
  // input sanitization
  req.sanitize('age').toInt();
  var errors = req.validationErrors();
  if (errors) {
    res.send('Errors: ' + JSON.stringify(errors), 400);
    return;
  }
  // further processing only after proper validations and sanitizations
  res.end('<h1>Hello, ' + req.body.firstname + req.body.lastname + ' </h1>');
});
```

Server-side v.s. Client-side Input Validations

- Reiterating once again, apply validations at:
  - server-side for security enforcement
  - client-side for user experience enhancement
  - Reason: client validation code is shipped to client, which can freely be manipulated and bypassed at browsers, while server logic is hidden from clients and will send only the resulted HTML. The computation integrity of validations can thus be protected.
- Security Best Practice:
  - Keep server- and client-side input validations as consistent as possible!!
  - Intrinsic advantage of Node.js: code/pattern reuse across client- and server-side

Output Formats

- HTML
  - Response size too bulky (bandwidth, latency)
  - Take server resources for data binding with templates
- JavaScript Object Notation (JSON)
  - Compact in response size. Fast JSON parser.
  - Facilitate shifting data binding and UI work to client-side
- XML
  - As bulky as HTML. Slower than JSON parser
  - Used in legacy web services supporting SOAP

- (Demo) In practice: Render minimal HTML at server-side (using templating framework like ExpressHandlebars). Do data binding with subtemplate at client-side (using Handlebars)

Some Logistics...

- Assignment Phase 2 Deadline: Feb 4, 2015, 5PM
- Self-studying PHP to enrich your knowledge and profile
    - Lecture Notes in 2012