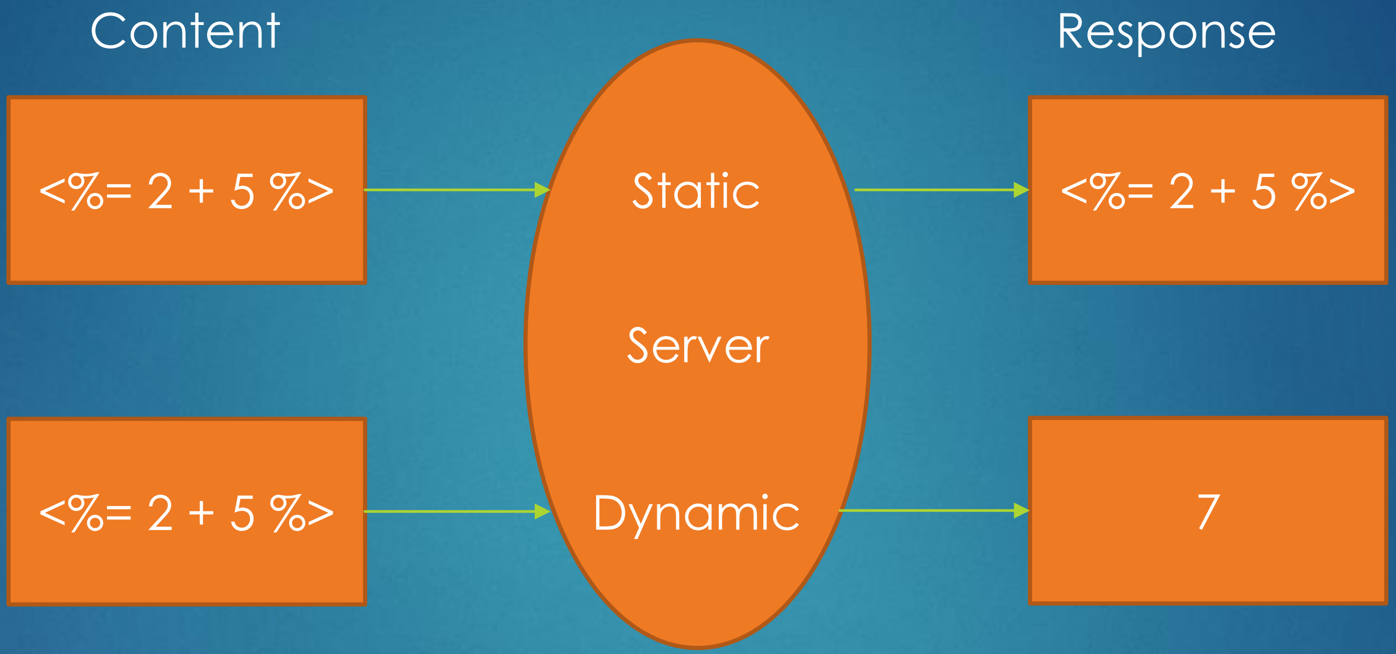
#05 Web Server (CGI, Node.js)

CLIENT/SERVER COMPUTING AND WEB TECHNOLOGIES

Web Servers

- Top 3 web servers (May 2014)
  - Apache: 38%
  - IIS: 33%
  - nginx: 15%
- Primary function is to store, process and deliver web pages to clients
- Support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages
  - Dynamic Content !!
- Communication protocol is Hypertext Transfer Protocol (HTTP)
- Can also be found embedded in devices such as printers, routers, webcams and serving only a local network

Static vs Dynamic



- Dynamic web content is built when it is requested, by the user directly, or programmatically while a user is on a page

Dynamic Content

- CGI provides an interface between the Web server and programs that generate the Web content
- FastCGI allows a single, long-running process to handle more than one user request while keeping close to the CGI programming model
- SCGI is similar to FastCGI but is designed to be easier to implement
- Platform Specific
  - Microsoft IIS: ISAPI (Internet Server API)
  - Java: **Servlet Container**
  - Ruby: Rack
    - wrapping HTTP requests and responses it unifies the API for web servers
  - Perl: WSGI (Web Server Gateway Interface)
    - a low-level interface between web servers and web applications
    - Plack is also available(influenced by Rack)

CGI

- Common Gateway Interface
  - provides an interface between the Web server and programs that generate the Web content
- CGI directory is a directory containing executable scripts (or binary files)
- Server runs specified script in a separated process.
  - Anything that the script sends to standard output is passed to the Web client
- Information from web server can be passed to a script via environment variables, e.g., **QUERY_STRING**
- CGI scripts can be written in any programming languages, e.g., Perl, Python

Node as a Script

http://larsjung.de/node-cgi/

- Node-CGI
  - npm install -g node-cgi

<< Apache2 configuration file >>

```
<Directory /var/www/html/cgi>
    Options      +ExecCGI +SymLinksIfOwnerMatch
    Action        node-script  /cgi-bin/node-cgi
    AddHandler    node-script  .nd
</Directory>
```

<< CGI Script (test.nd) in JavaScript >>

```
for(k in env){
   writeLine(k + "=" + env[k] + "<br/>");
}
```

`env` is an exported variable from `process.env`
See: http://nodejs.org/api/process.html#process_process_env

Sample Result

```
REDIRECT_HANDLER=node-script
REDIRECT_STATUS=200
HTTP_HOST=192.168.1.122
HTTP_CONNECTION=keep-alive
HTTP_ACCEPT=text/html,application/xhtml+
xml
HTTP_USER_AGENT=Mozilla/5.0
HTTP_ACCEPT_ENCODING=gzip, deflate, sdch
HTTP_ACCEPT_LANGUAGE=en-US
SERVER_SIGNATURE=Apache/2.4.10 (Ubuntu)
SERVER_SOFTWARE=Apache/2.4.10 (Ubuntu)
SERVER_NAME=192.168.1.122
SERVER_ADDR=192.168.1.122
SERVER_PORT=80
REMOTE_ADDR=192.168.1.6
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/node-cgi
REMOTE_PORT=51183
REDIRECT_QUERY_STRING=a=2
REDIRECT_URL=/cgi/test.nd
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=a=2
REQUEST_URI=/cgi/test.nd?a=2
SCRIPT_NAME=/cgi-bin/node-cgi
PATH_INFO=/cgi/test.nd
PATH_TRANSLATED=/var/www/html/cgi/test.nd
```

http://192.168.1.122/cgi/test.nd?a=2

Node as a Server

- http built-in module is available to create a web server

```
var http = require('http');
var server = http.createServer(function(req, res){
   res.writeHead(200, {'Content-type': 'text/plain'});
   res.end('Hello world\n');
});

server.listen(8000);
console.log('Server is ready!');
```

Express

- minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications
  - npm install express

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send('Hello world')
});

app.listen(8000);
```

res.send(body) - When the parameter is a String, the method sets the Content-Type to "text/html"

Express Routing

- Routing refers to the definition of end points (URIs) to an application and how it responds to client requests.
- A route is a combination of
  - a URI
  - a HTTP request method (GET, POST, and so on)
  - one or more handlers for the endpoint.
- It takes the following structure

```
app.METHOD(path, [callback...], callback)
```

  - app is an instance of express
  - METHOD is an HTTP request method
  - path is a path on the server
  - callback is the function executed when the route is matched.

Express Middleware

- An Express application is essentially a series of middleware calls.
- Middleware is a function with access to the request object (req), the response object (res), and the next middleware in line.
- Middleware can:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.
- If the current middleware does not end the request-response cycle, it must call next() to pass control to the next middleware

Middleware Example

```
// a middleware with no mount path; gets executed for every
request to the app
app.use(function (req, res, next) {
   console.log('Time:', Date.now());
   next();
});

// a middleware mounted on /user/:id; will be executed for any
type of HTTP request to /user/:id
app.use('/user/:id', function (req, res, next) {
   console.log('Request Type:', req.method);
   next();
});
```

/user/:id is an example of mount point.

Built-in/3rd party middleware

- Only 1 built-in middleware
  - express.static (built-in) is based on serve-static, and is responsible for serving the static assets of an Express application
    - app.use(express.static('public'));
- Useful 3rd party middleware (must be installed with *npm*)
  - cookie-parser: Parse Cookie header and populate req.cookies with an object keyed by the cookie names
  - express-session: Simple session middleware for Express
  - body-parser: Provide JSON body parser, Raw body parser, Text body parser and URL-encoded form body parser

HTTP Messages

```
// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))
```

head

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

body

```
licenseID=string&content=string&/paramsXML=string
```

- First line indicates whether the message is a *request* or a response.
- Followed by multiple headers such as User-Agent, Host
- `\r\n` is a delimiter separating head and body
- Body can be anything from simple text to images; see Content-Type

Example: adding

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

var urlencodedParser = bodyParser.urlencoded({ extended: false });
app.use(express.static(__dirname + '/public'));

app.post('/add', urlencodedParser, function(req, res){
   var result = parseInt(req.body.a) + parseInt(req.body.b);
   res.send('Result = ' + result);
});

app.listen(8000);
```

>> npm install express body-parser

```
<html>
<head>
<title>Adding Form</title>
</head>
<body>
   <form action="/add" method="post">
     A: <input type="number" name="a"/><br/>
     B: <input type="number" name="b"/><br/>
     <button type="submit">Add</button>
   </form>
</body>
</html>
```

>> http://localhost:8000/form.html

Web Session Tracking

- HTTP is a "stateless" protocol
  - each time a client retrieves a Web page, the client opens a separate connection to the Web server
  - the server automatically does not keep any record of previous client request.
- Session Tracking
  - URL Rewriting
    - put session id into URL, e.g., http://abc.com/action;sessionid=12345
    - works for the browsers when they don't support cookies
  - Hidden From Fields: similar to URL rewriting when using method GET
    - embedded session id in HTTP body if using method POST
  - Cookies
  - Sessions

Cookies (on Client)

- Cookies are store in client (Scalable but not safe)
- A webserver can assign a unique session ID as a cookie to each web client
  - Client (browser) sends assigned cookie for subsequent requests

```
app.use(cookieParser('keyboard cat'))

app.get('/ck_get', function(req, res) {
   res.send(req.cookies)
})

app.get('/ck_set', function(req, res) {
   res.cookie('a', 10)
   res.send('ok')
})
```

Sessions (on Server)

- Session ID is probably stored in
  - Cookie
  - HTTP URL or Body
  - HTTP Header (Session-Id)
- Session information can be all kept in server side (Safe but not quite sca

```
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 }}))
app.use(function(req, res, next) {
   var sess = req.session
   if (sess.views) {
      sess.views++
   } else {
      sess.views = 1
   }
})
```

Express Template Engine

- Before Express can render template files, the following application settings have to be set.
  - views, the directory where the template files are located.
  - view engine, the template engine to use.

```
app.set('views', './views')
app.set('view engine', 'ejs')

app.get('/fruit', function(req, res){
   res.render('fruit', {fruits: ['banana', 'apple']})
})
```

```
<ul>
<% fruits.forEach(function(fruit){ %>
   <li><%= fruit %></li>
<% }); %>
</ul>
```

References

- http://en.wikipedia.org/wiki/Web_server
- http://www.tutorialspoint.com/jsp/jsp_session_tracking.htm
- http://expressjs.com/guide/using-middleware.html
- http://expressjs.com/guide/routing.html
- http://expressjs.com/guide/using-template-engines.html
- http://www.tutorialspoint.com/http/