ECS 189H
WEB PROGRAMMING

5/22

Announcements

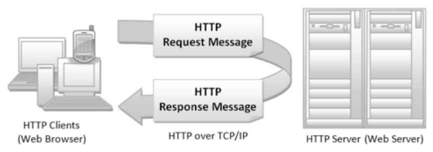
LOTS OF EXTENSIONS!

- Photobooth Part 1 due midnight THURSDAY 5/25
  - A little extra credit if you hand it in tonight
- Part 2 due Tues 5/30
- Midterm 2 Friday 6/2
- My lab hours 4-5:30, 71 Kemper
- No new material in this lecture

Asynchronous programming

- Request-response pattern we see in different forms:
  - Client makes request to server
  - Callback function run when response comes back
- Interaction is always initiated by client
- In each case – which is client? which is server?
- Four major elements:
  - specify request
  - set up callback
  - send off request
  - callback function run when response gets back

HTTP protocol



- Requires this request-response pattern for loading Web pages, AJAX interactions.
- But we find the same thing in JSONp, DB operations

JSONp callback

```
script.src = "https://query.yahooapis.com/v1/public/
   yql?q=select * from weather.forecast where woeid
   in (select woeid from geo.places(1) where
   text='"+newPlace
   +"')&format=json&callback=callbackFunction"
document.body.appendChild(script);

function callbackFunction(data) {
   var pgh = document.getElementById("forecast");
   pgh.textContent = JSON.stringify(data);  }
```

AJAX request

```
var oReq = new XMLHttpRequest();
var url = "http://138.68.25.50:?????/query?
                                    op=dumpDB";
oReq.open("GET", url);
function respCallback () {
   var dataArray = JSON.parse(this.responseText);
   addPhotosToDOM(dataArray); }
oReq.onload(respCallback);
oReq.send();
```

Photo upload

```
var oReq = new XMLHttpRequest();
var url = "http://138.68.25.50:????";
var selectedFile =
   document.getElementById('fileSelector').files[0];
var formData = new FormData();
formData.append("userfile", selectedFile);
oReq.open("POST", url, true);
oReq.onload = function()
   { console.log(oReq.responseText) };
oReq.send(formData);
```

DP operation

```
db.all('SELECT * FROM photoLabels',dataCallback);
function dataCallback(err, tableData) {
   if (err) {
       console.log(err);
       sendCode(400,reponse,"error reading DB" }
   else {
       sendCode(200,response,tableData); }
}
```

Why are DB ops asynchronous?

Why are DB ops asynchronous?

- Server should always be ready to respond as new HTTP requests come in
- A database request may take a while; disk access can be slow
- Server should not wait for a database operation to finish before getting started on new requests

Why…

- …are AJAX requests and API requests, made from the browser, asynchronous?

Why…

- …are AJAX requests and API requests from the browser asynchronous?
- Browser should respond to user button pushes, etc, immediately; should never get hung up waiting for requests running over the internet
- Especially when connections might be poor!

Bad example: DB request

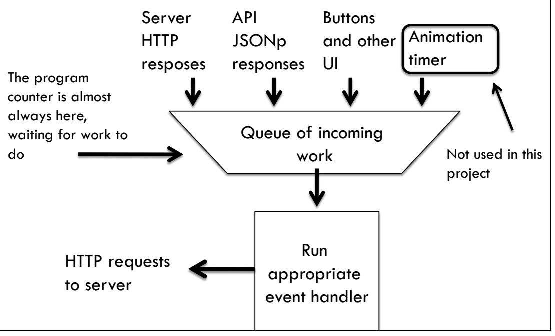
```
tableData =
   db.all('SELECT * FROM photoLabels',dataCallback);
console.log(tableData);
```

Prints out:

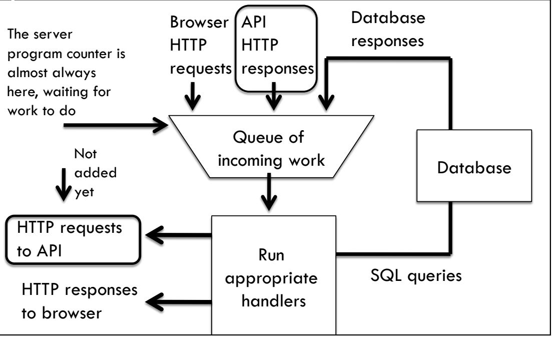
tableData contains: Database { open: false, filename: 'photos.db', mode: 65542 }

What's wrong?

Browser control flow



Server control flow

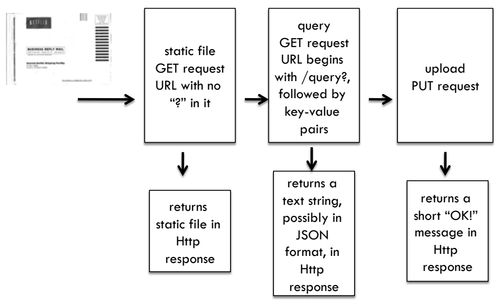


Request and response

- The server's job is to get HTTP requests and produce the appropriate HTTP response for each one.
- It is called on two objects, request and response. The response object is like a pre-addressed envelope, addressed to the machine that made the request.

(You are the server, Netflix is the browser, in this metaphor.)

Our server


static file GET request URL with no "?" in it
query GET request URL begins with /query?, followed by key-value pairs
upload PUT request
returns static file in Http response
returns a text string, possibly in JSON format, in Http response
returns a short "OK!" message in Http response

The response object

- The static server puts the requested static file into the response object, and sends it off
- The dynamic query server computes a response, often in JSON but also potentially in HTML, XML…, puts that into the response object, and sends it off
- Response object passed to function that will fill it in
- Often putting together the response requires doing an API or database request, so it won't be done immediately but in a callback

Closure

- Example from "Eloquent Javascript", Chapter 3.

```
function multiplier(factor) {
   return function inner (number) {
         return number * factor; };
   }
var twice = multiplier(2);
console.log(twice(5));
var thrice = multiplier(3);
console.log(thrice(5));
```

Closure

```
function multiplier(factor) {
   return function inner (number) {
         return number * factor; };
   }
```

- Function that returns a function
- factor is a local variable inside multiplier
- inner remembers value of factor when it was created

Closure

```
function multiplier(factor) {
   // return function inner (number) {
   return function (number) {
         return number * factor; };
   }
```

- Anonymous function version
- No reason inner function has to have a name; it will never be called except here

Using closure to pass response object

- From lecture Friday, how to answer query to add a label:

```
function answer(query, response) {
 … get current labels from DB via SQL request-response cycle,
   edit labels, send off UPDATE SQL command with callback…
    function updateCallback(err) {
       if (err) {  sendCode(400,response,"not found"); }
      else { sendCode(200,response,
         "added label "+newLabel+ " to "+imageFile); } }
  } // close answer(query, response)
```

Using closure to pass response object

```
function answer(query, response) {
 … get current labels from DB via SQL request-response cycle,
   edit labels, send off UPDATE SQL command with callback…
    function updateCallback(err) {
       if (err) {  sendCode(400,response,"not found"); }
      else { sendCode(200,response,
         "added label "+newLabel+ " to "+imageFile); } }
  } // close answer(query, response)
```

- updateCallback is defined inside answer, so it has access to all the variables of answer, even though it runs much later

Looping over image list

- Creating a separate onclick function for every image can be done neatly using closure and an anonymous funciton
- Once you query the database, you'll get an array containing the DB contents:

```
tableData = [{filename: "hula.jpg", labels: "Dance, Hula, Lei",
   favorite: 0},
            {filename: "eagle.jpg", labels: "Eagle, Bird, Beak",
   favorite: 0},
            {filename: "redwoods.jpg", labels: "Forest, Trees,
   Redwoods", favorite: 0}]
```

Looping over image list

- Loop over this list to insert a div containing an img for each picture
- We'd like to add an onclick function for each div (or for the hamburger button we put on each div). But we CANNOT do this (why?):

```
for (i=0; i<tableData.length; i++) {
…
   newDiv.onclick = showImageName("Photo "+i+", "+labels, i);
… }
```

Looping over image list

- We also cannot do this!

```
newDiv.onclick = function () {
   showImageName("Photo "+i+", "+labels, i);
}
```

- There is a separate onclick for each photo.
- But there is only one variable i, and when the onclick is called it will use whatever value i last contained.

Looping over image list

- But we can do this!

```
function createNewOnclick(index,labels) {
   return function() {
        showImageName("Photo "+index+", "+labels,
   index); } }
newDiv.onclick = createNewOnclick(i, labels);
```

- createNewOnclick returns a function.
- That anonymous function is in the closure of createNewOnclick, and remembers its local variables