COMP 2406/2006: Fundamentals of Web Applications

Fall 2013 Final Exam Solutions

December 16, 2013

1. **(false)** HTTP cookies are only sent to a web server when explicitly requested.
2. **(true)** Arbitrary (even undefined) properties of objects can be accessed in JavaScript without generating a runtime error.
3. **(true)** `var` causes a symbol to be defined within the inner most function context.
4. **(true)** Read access to object properties are resolved by looking at the object's properties and then by looking at the chain of objects pointed to by the objects on the `prototype` chain.
5. **(false)** The `function` function invocation pattern allows you to set the value of `this`.
6. What HTTP command is normally used to request the contents of a page? **A: GET**
7. (18) How will program functionality change or what sort of error do you expect to see when the following lines are deleted from *small-adventure2*?

   i. app.js, lines 7
   A: The program won't be able to find Jade templates for `res.render()` calls, making the server crash on the first page request.

   ii. app.js, line 10
   A: Line 30 of index/routes.js will generate a server runtime error from `req.body.player` because you cannot get the property of undefined, which is what req.body's value is (because `request.body` is what is filled out by bodyParser()).

   iii. routes/index.js, line 18
   A: Rooms will stay undefined because `setupRoomsDB()` will never be called, meaning any reference to its properties will produce a server runtime reference error. So line 67 of routes/index.js will cause the server to crash.

   iv. routes/index.js, line 22
   A: If the player has already entered their name in the session and they visit /, instead of redirecting to /game the request will hang.

   v. routes/index.js, line 33
   A: The player will start in the "Unknown room" (the room that is returned when the current room cannot be found) and won't be able to leave it (unless there is a room named "undefined").

   vi. routes/index.js, line 48
   A: After the player enters their name, the request for /game will hang rather than give the game page.

   vii. routes/index.js, line 73
   A: The AJAX request for the room contents will never return, meaning that the default values for the room (e.g., "Room Name") will never be changed.

   viii. views/layout.jade, line 9
   A: The initial request for / will return a page with a blank body (empty contents).

   ix. views/room.jade, line 11
   A: No change. (There will be no initial "exits" to remove, but the exit buttons will be created because the #exitList node still exists.)

8. (2) Is the player's name incorporated into room descriptions in *small-adventure2*? If so, how? If not, how would you change the code so it is?
A: The player name is incorporated into room descriptions because of lines 70–72 in routes/index.js. The string replace() method does the substitution.

9. (2) In the `express` module, `bodyparser()` should be originally assigned as a method of what object? Why?
A: It should be assigned as a method of the exports object, as that is the object that is returned as a result of calling require().

10. (2) All HTTP requests return a status code saying how the request went. Successful requests normally have a 200 status; missing pages produce a 404 status code, meaning not found. By default calls to `res.redirect()` return a status code of 302. What do you think this status code means, or if you know, what does it mean?
A: It means "redirect": that a page has moved to a new URL and that new URL should be loaded.

11. (2) If client-side JavaScript is turned off, what are the contents of the page returned for a request to `/game` (assuming the user has already entered their name in this session)? Why?
A: The page will just show what was defined in the Jade template for the room page (a large heading saying "Room Name" followed by a paragraph saying "Room Description" followed by a paragraph saying "Go to:" with one Exits button in a list followed by a Quit submit button). The styling may also be messed up because bootstrap uses some JavaScript (but it is mostly CSS which will be unaffected). This happens because none of the code in adventure.js will run in the client, and this is the code that changes the room page to have the actual contents of a room.

12. (10) Five functions are defined in *adventure.js*. What do each of theses functions do, and when are each of them called? Explain briefly.
A: The functions are:
    - **line 1, anonymous: used to set up a private scope, is called right after it is defined (when the file is loaded in the page).**
    - **line 2, getRoom(): Defines the function that updates the page with the contents of the current room. Is called when the page finishes loading and whenever an exit button is clicked.**
    - **line 3, anonymous: defines a callback handles the result of the AJAX GET request for /getContents. Called for each request to /getContents.**
    - **line 9, anonymous: Called for each room exit to create the exit buttons and associated exit callbacks (by the array forEach). This implicit loop runs once for every call to getRoom().**
    - **line 14, anonymous: This is a callback that runs when an exit button is clicked. It changes the current room on the server then displays that new room.**

13. (2) Lines 17–22 in *small-adventure2/app.js* define routes for six HTTP requests (3 GETs and 3 POSTs). The server will also respond successfully to several other HTTP requests. What are some of these requests, and what part of the application's code is responsible for generating those responses?
A: It will respond to requests for files in the public subdirectory, e.g., for jQuery, bootstrap, and adventure.js. This functionality is enabled by line 14 of app.js, the call to express.static().

14. (4) When does the code on lines 17–22 of *small-adventure2/app.js* run? What is the effect of their execution?
A: They run when the server is first started and only then. They register callbacks to handle GET and POST requests for six URLs (/, /game, etc.). Those callbacks are invoked when those URLs are requested.

15. (12) In *small-adventure2/routes/index.js*, there are six calls to methods defined (directly or indirectly) by the mongoose module. What are these six, and what does each do?
A: The six functions are:

- **line 6, mongoose.connect(): Defines how to reach MongoDB and what collection to use.**
- **line 7, db.on() (which is really mongoose.connection.on): defines a callback that will log an error if a connection to MongoDB cannot be established.**
- **line 10, mongoose.Schema(): Defines the schema for documents in the collection.**
- **line 16, mongoose.model(): compiles the schema, returning a model object that can be used to retrieve and store documents.**
- **line 18, db.once() (really mongoose.connection.once): once the connection to MongoDB has been established, call setupRoomDB so that we get a valid Room model.**
- **line 67, Room.findOne(): Finds the document that has a name the same as the current room (req.session.currentRoom).**

Code listings begin on next page.

small-adventure2/app.js:

```
var express = require('express');
var routes = require('./routes');
var http = require('http');
var app = express();

app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.cookieParser('COMP2406 is over!'));
app.use(express.session());
app.use(app.router);
app.use(express.static(__dirname + '/public'));
app.use(express.errorHandler());

app.get('/', routes.index);
app.get('/game', routes.game);
app.get('/getContents', routes.getContents);
app.post('/start', routes.start);
app.post('/quit', routes.quit);
app.post('/doAction', routes.doAction);

http.createServer(app).listen(app.get('port'), function(){
    console.log('Express server listening on port ' + app.get('port'));
});
```

small-adventure2/routes/index.js:

```
var sanitize = require('validator').sanitize;
var mongoose = require('mongoose');
var db = mongoose.connection;
var Room;

mongoose.connect('mongodb://localhost/2406rooms');
db.on('error', console.error.bind(console, 'MongoDB connection error:'));

var setupRoomsDB = function() {
    var roomSchema = mongoose.Schema({
        name: {type: String, required: true},
        title: {type: String, required: true},
        description: {type: String, required: true},
        roomExits: {type: [String], required: true}
    });
    Room = mongoose.model("Room", roomSchema);
}
db.once('open', setupRoomsDB);

exports.index = function(req, res) {
    if (req.session.player) {
        res.redirect("/game");
    } else {
        res.render('index', { title: 'COMP 2406 Adventure Demo',
                              error: req.query.error });
    }
}

exports.start = function(req, res) {
    var player = req.body.player;

    req.session.player = sanitize(player).escape();
    req.session.currentRoom = "start";
    res.redirect("/game");
}

exports.quit = function(req, res) {
    req.session.destroy(function(err){
        if(err){
            console.log("Error: %s", err);
        }
    });
    res.redirect("/");
}

exports.game = function(req, res) {
    if (req.session.player) {
        res.render("room.jade", {title: "Adventure Demo"});
    } else {
        res.redirect("/");
    }
}

exports.doAction = function(req, res) {
    var action = req.body.action;
    var room = req.body.room;

    if (action === "move") {
        req.session.currentRoom = room;
        res.send("Success");
    } else {
        res.send("Error: InvalidAction");
    }
}

exports.getContents = function(req, res) {
    Room.findOne({ name: req.session.currentRoom },
                 function(err, theRoom) {
                     if (!err && theRoom) {
                         theRoom.description =
                             theRoom.description.replace("<?player>",
                                                         req.session.player);
                         res.send(theRoom);
                     } else {
                         res.send({name: "unknown",
                                   title: "Unknown room",
                                   description: "error finding room",
                                   roomExits: []});
                     }
                 });
}
```

small-adventure2/views/layout.jade:

```
doctype 5
html
  head
    title= title
    script(src='/libs/jquery/jquery.min.js')
    link(rel='stylesheet', href='/libs/bootstrap/css/bootstrap.min.css')
    block header
  body
    block content
```

small-adventure2/views/index.jade:

```
extends layout

block content
  h1= title
  p Please choose your player name
  div
    form(action="/start", method="post")
        div.control-group.input-append
            input(type="text", name="player")
            label.add-on(for="player") Player Name
        button(type="submit") Start
```

small-adventure2/views/room.jade:

```
extends layout

block header
  script(src='/javascripts/adventure.js')

block content
  h1(id="title") Room Name
  p(id="description") Room Description
  p Go to:
  div(id="exitList").btn-group
    button(type="button").btn.btn-default.exits Exits
  p
  form(action="/quit", method="post")
     button(type="submit") Quit
```

small-adventure2/public/javascripts/adventure.js:

```
(function() {
    var getRoom = function() {
        $.getJSON("/getContents", function(room) {
            var exits;
            $('#title').replaceWith('<h1 id="title">'+room.title+'</h1>');
            $('#description').replaceWith('<p id="description">' +
                                          room.description + '</p>');
            $('.exits').remove();
            room.roomExits.forEach(function(theExit) {
                $('#exitList').append(
                    '<button type="button" id="' + theExit +
                        '" class="btn btn-default exits">'
                        + theExit + '</button>');
                $('#'+theExit).on('click', function() {
                    $.post("/doAction", {action: "move",
                                         room: theExit});
                    getRoom();
                });
            });
        });
    }
    $(window).load(getRoom());
})();
```