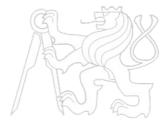
Middleware Architectures 2

Lecture 1: Asynchronous I/O

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • http://vitvar.com



Czech Technical University in Prague
Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/w20







Modified: Sun Mar 14 2021, 11:18:53 Humla v0.3

Overview

- Asynchronous I/O Overview
- Asynchronous I/O in JavaScript
- JavaScript Language Overview

Recall: Application Server

- Environment that runs an application logic
 - Client communicates with AS via an application protocol
 - Client Browser, application protocol HTTP
- Terminology
 - Application Server × Web Server × HTTP Server
 - → AS is a modular environment; provides technology to realize enterprise systems
 - → AS contains a Web server/HTTP server
 - We will deal with Web server only
- Two major models to realize communication
 - Blocking I/O (also called synchronous I/O)
 - Non-blocking I/O (also called asynchronous I/O)
- A technology we will work with
 - Node.js runs server-side Javascript

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitval

- 3 -

Programming Models

- Concurrency
 - Multiple tasks have the ability to run in an overlapping manner
 - Concurrency does not imply parallelism!
- Multiprocessing
 - CPU-bounded tasks
 - Allows to process multiple processes on different CPUs
- Multithreading
 - I/O bound tasks
 - Multiple threads execute tasks
 - A process may contain multiple threads
 - It uses preemtive multitasking
 - → OS decides how long a task should run (no tasks cooperation)
 - → context switching
 - Threads can access shared memory; you need to controll this

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Asynchronous I/O

- Asynchronous I/O
 - A style of concurrent programming; it is not a parellelism
 - Single-threaded, single process design
 - It uses cooperative multitasking
- Asynchronous processing of a task
 - Tasks are running in so called **event loop**
 - A task is able to "pause" when they wait for some result
 - \rightarrow A task let other tasks to run
 - Asynchronous code faciliates concurrent execution
 - → It gives the "look and feel" of concurrent execution

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 5 -

Overview

- Asynchronous I/O Overview
- Asynchronous I/O in JavaScript
- JavaScript Language Overview

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Web 2.0 Application Architecture Web Application client-side technologies for presentation and user App server technologies interactions HTML Server-side HTTP request DOM. JavaScript, SQL JavaScript Servlet (Java) HTTP response **Database** Application Web Browser Server HTTP application protocol over Web 2.0 Application HTTP Server-side req/resp DOM, JavaScript, SOL **JavaScript** Servlet (Java) **Database** AJAX **Application** Web Server Asynchronous calls to server Web Services (Web API) Dynamic creation and manipulation of HTML dynamic JavaScript code

JavaScript

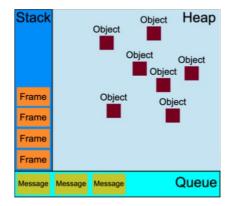
- Lightweight, interpreted, object-oriented language
- Client-side (browser) and server-side (node.js, AppsScript)
- Standard
 - Current stable release is ECMAScript 2020
- Major characteristics
 - Function is an Object

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- \rightarrow passing functions as arguments to other functions
- → returning functions as values from other functions
- \rightarrow assigning functions to variables
- \rightarrow storing functions in data structures.
- Anonymous functions
 - \rightarrow declared without any named identifier to refer to it
- Arrow functions
- Closures

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Javascript Runtime



Event Loop:

```
while (queue.waitForMessage()) {
   queue.processNextMessage()
}
```

- Stack
 - Contains frames, i.e. function parameters and local variables
- Heap
 - Objects are allocated in a heap, a region of memory.
- Queue
 - A list of messages to be processed
 - Message is data and callback to be processed

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 9 -

Stack

• When running a program...

```
function foo(b) {
  let a = 10
  return a + b + 11
}

function bar(x) {
  let y = 3
  return foo(x * y)
}

console.log(bar(7)) //returns 42
```

- 1. calling bar: a frame is created with bar's arguments and variables.
- 2. bar calls foo: a new frame with foo's args and vars is created.
- *3.* **foo** *returns: the top frame element is popped out of the stack.*
- 4. bar returns: the stack is empty.

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 10 –

Event Loop

Event loop

```
while (queue.waitForMessage()) {
   queue.processNextMessage()
}
```

- -Message = data + callback to be processed
- Messages are process completely one by one
 - → No "clashes" across messages' processing
 - → Processing should not block for a long time Workers
- Brwoser adds a new message when an event occurs and there is an event listener

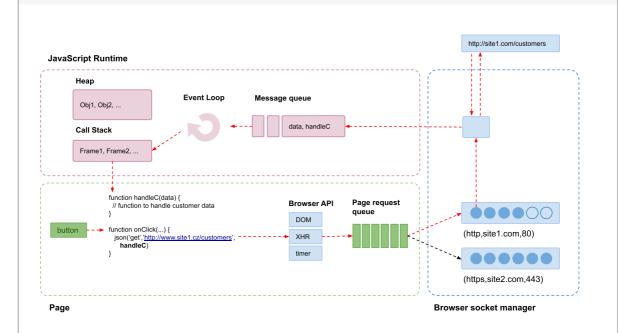
Run-to-completion

- Each message is processed fully before any other message is processed.
- A function runs entirely before any other code runs
 - → unlike in preemtive multitasking
- If a message takes much time to complete, all work can be blocked!

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 11 -

Handling Request



Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 12 -

Multiple Runtimes

- Runtime
 - Stack, Heap, Message Queue
 - iframe and a Web worker has its own runtimes
- Communication between runtimes
 - Runtimes communicate via postMessage
 - A runtime can receive a message if it listens to message events

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 13 -

Web Workers

- A code that runs in a worker thread
 - Every worker runs event loop; communicate via posting messages
 - Can do anything, but manipulate DOM
 - Can spawn a new workers
 - They are thread-safe
- Workers Types
 - Dedicated workers accessible by scripts that created them
 - Shared workers accessible by multiple scripts (iframes, windows, workers)
- Example

```
// main.js
var myWorker = new Worker('worker.js');

something.onchange = function() {
   myWorker.postMessage([value1,value2]);
}

// worker.js
onmessage = function(e) {
   var workerResult = 'Result: ' + (e.data[0] * e.data[1]);
   postMessage(workerResult);
}

// ... and terminate
myWorker.terminate()
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 14 -

Node.js

- Node.is
 - Web server technology, very efficient and fast!
 - Event-driven I/O framework, based on JavaScript V8 engine
 - → *Any I/O is non-blocking (it is asynchronous)*
 - One worker thread to process requests
 - → You do not need to deal with concurrency issues
 - More threads to realize I/O
 - Open sourced, @GitHub ₺, many libraries ₺
 - Future platform for Web 2.0 apps
- Every I/O as an event
 - reading and writing from/to files
 - reading and writing from/to sockets

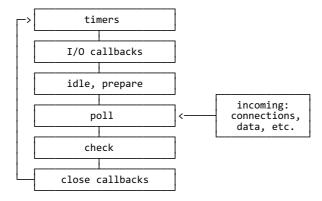
```
// pseudo code; ask for the last edited time of a file
stat( 'somefile', function( result ) {
   // use the result here
} );
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 15

Node.js Event Loop

• Allows Node.js to perform asynchronous I/O operations.



- Six phases, each phase has a FIFO queue of callbacks to execute.
 - → *timers executes callbacks sheduled by* setTimeout() *and* setInterval()
 - \rightarrow I/O callbacks executes all I/O callbacks except close callbacks.
 - \rightarrow *idle/prepare* used internally
 - → poll retrieve new I/O events
 - → check invokes setImmediate() callbacks
 - → close callbacks executes close callback, e.g. socket.on('close', ...).

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 16 -

HTTP Server in Node.js

HTTP Server implementation

- server running at 127.0.0.1, port 8080.

```
const http = require('http');

const hostname = '127.0.0.1';

const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World');
});

server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 17 -

Google Apps Script

- Google Apps Script
 - JavaScript cloud scripting language
 - easy ways to automate tasks across Google products and third party services
- You can
 - Automate repetitive processes and workflows
 - Link Google products with third party services
 - Create custom spreadsheet functions
 - Build rich graphical user interfaces and menus

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Overview

- Asynchronous I/O Overview
- Asynchronous I/O in JavaScript
- JavaScript Language Overview

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 19 -

Objects and Arrays

Objects and Arrays

Functions

```
// assign a function to a variable
var minus = function(a, b) {
    return a - b;
}

// call the function;
// now you can pass 'minus' as a parameter to another function
var r2 = minus(6, 4);
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Functions

- Function Callbacks
 - You can use them to handle asynchronous events occurrences

```
// function returns the result through a callback, not directly;
// this is not a non-blocking I/O, just demonstration of the callback
function add(a, b, callback) {
    callback(a + b);
}

// assign the callback to a variable
var print = function(result) {
    console.log(result);
};

// call the function with callback as a parameter
add(7, 8, print);
```

Functions as values in object

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 21 -

Closures

- Closures
 - A function value that references variables from outside its body

```
function adder() {
    var sum = 0;
    return function(x) {
        sum += x;
        return sum;
    }
}

var pos = adder();

console.log(pos(3)); // returns 3
console.log(pos(4)); // returns 7
console.log(pos(5)); // returns 12
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 22 –

Objects

• this problem

- A new function defines its own this value.

```
function Person() {
    // The Person() constructor defines `this` as an instance of itself.
    this.age = 0;

setInterval(function growUp() {
        // the growUp() function defines `this` as the global object,
        // which is different from the `this`
        // defined by the Person() constructor.
        this.age++;
    }, 1000);
}

var p = new Person();

- Solution

function Person() {
    var that = this;
    that.age = 0;

setInterval(function growUp() {
        // The callback refers to the `that` variable of which
        // the value is the expected object.
        that.age++;
    }, 1000);
}
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 23 -

Arrow Functions

- Arrow function expression
 - defined in ECMAScript 2015
 - shorter syntax than a function expression
 - non-binding of this

```
function Person(){
this.age = 0;

setInterval(() => {
    this.age++; // | this | now refers to the person object
}, 1000);
}

var p = new Person();
```

• Syntax, function body

```
// concise body syntax, implied "return"
var func = x => x * x;

// with block body, explicit "return" required
var func = (x, y) => { return x + y; };

// object literal needs to be wrapped in parentheses
var func = () => ({foo: 1});
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 24 –

Callback Hell

• Callback in callback

- Complex asnychronous code is hard to understand and manage

Solution

- Promise a proxy to a "future" value of the function
- Async/await language constructs to work with asynchronous code

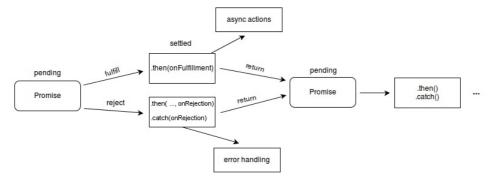
Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 25 -

Promise Object

Promise

- An object representing completion or failure of an asynchronous operation.
- A proxy for a value not necessarily known when the promise is created.



Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

Callback Hell Example

• A callback in a callback

```
const request = require('request');
request("http://w20.vitvar.com/toc.json", { json: true },
(err, res, body) => {
   if (err)
        console.log("error: " + err)
   else {
        console.log(body)
        request("http://mdw.vitvar.com/toc.json", { json: true },
        (err, res, body) => {
        if (err)
            console.log("error: " + err)
        else
            console.log(body)
        }
        else
        console.log(body)
        }
}
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

– 27 –

Promise Example

• A chain of Promise objects

```
const request = require('request');

function get_json(url) {
    return new Promise((resolve,reject)=>{
        request(url, { json: true }, (err, res, body) => {
        if (err)
            reject(err)
        else
            resolve(body)
        })

})

get_json('http://w20.vitvar.com/toc.json')
.then((data)=>{
        console.log(data)
        return get_json('http://mdw.vitvar.com/toc.json')
})
.then((data)=>{
        console.log(data)
})
.then((data)=>{
        console.log(data)
})
.catch((err)=>{
        console.log("error: " + err)
})
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

async/await

async

- the function always returns a Promise
- if there is no Promise, the returned value is wrapped into Promise

```
async function f() {
    return 1;
}

f().then((v) => alert(v));
```

await

- makes program to wait until the promise is resolved or rejected
- it returns the resolved value and throws an exception when the promise is rejected
- can only be usded inside async function

```
async function f() {
  var promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

var result = await promise; // wait untill the promise is resolved
  alert(result);
}

f();
```

Lecture 1: Asynchronous I/O, CTU Summer Semester 2020/2021, @TomasVitvar

- 29 -