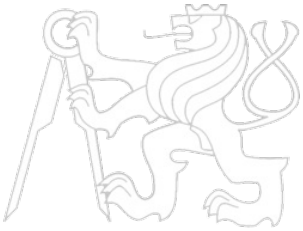
Web 2.0

Lecture 1: Introduction to JavaScript

doc. Ing. Tomáš Vitvar, Ph.D.

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

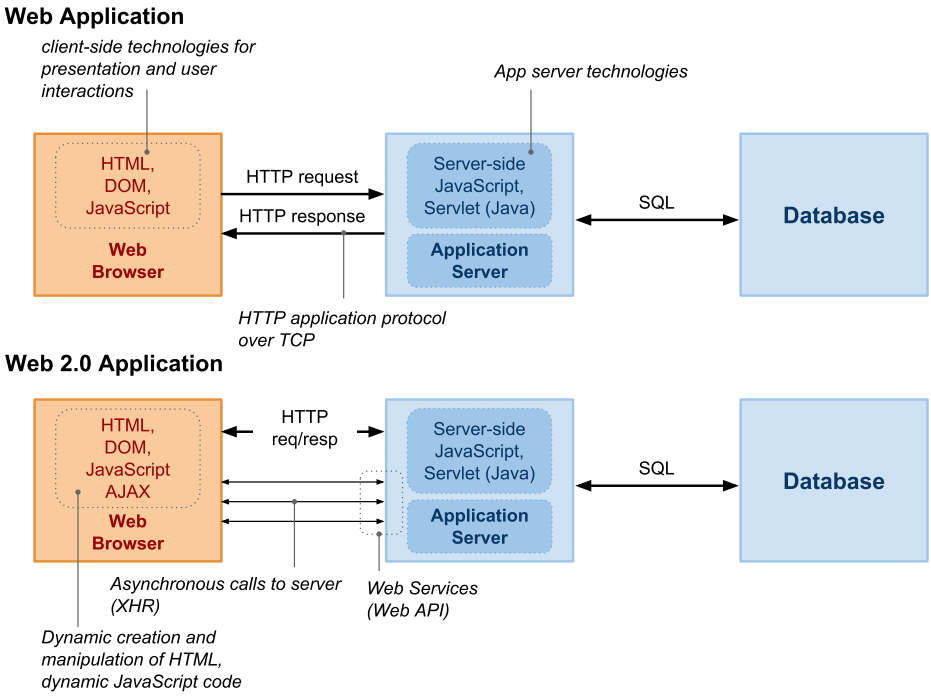


Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/w20

Modified: Tue Feb 20 2018, 10:59:56
Humla v0.3

Web 2.0 Application Architecture



JavaScript

- Lightweight, interpreted, object-oriented language
- Standard
  - *Current stable release is ECMAScript 2017 (standard ECMA-262)*
- Major characteristics
  - *First-class functions*
    - → *functions as first-class citizens*
    - → *language supports: passing functions as arguments to other functions, returning functions as values from other functions, assigning functions to variables or storing them in data structures.*
  - *Anonymous functions*
    - → *declared without any named identifier to refer to it*
  - *Closures*

Overview

- JavaScript Basics
- Server-side JavaScript

Objects and Arrays

- Objects and Arrays

```
// objects - key/value pairs
var obj = { name: "Tomas", "main-city" : "Innsbruck", value : 3 };
obj.name = "Peter"; // assign the name property another value
obj["main-city"] = "Prague"; // another way to access object's values; it's not an array

// arrays
var a = ["Tomas", "Peter", "Alice"];
for (var i = 0; i < a.length; i++)
    // do something with a[i]

// combinations of arrays and objects
var obj_a = [
    { name: "Tomas", city: "Innsbruck" },
    { name : "Peter", city : "Prague" },
    { name : "Alice", cities : ["Prague", "Brno"] } ];

for (var j = 0; j < obj_a.length; j++)
    // do something with obj_a[j].name, ...
```

- 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);
```


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

```
var obj = {
    data : [2, 3, "Tomas", "Alice", 4 ],

    getIndexdOf : function(val) {
        for (var i = 0; i < this.data.length; i++)
            if (this.data[i] == val)
                return i;
        return -1;
    }
}

obj.getIndexOf(3); // will return 1
```

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
```


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);
}
```

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" needed
var func = (x, y) => { return x + y; };

// remember to wrap the object literal in parentheses
var func = () => ({foo: 1});
```


Promise

- Promise
  - *An object representing completion or failure of an asynchronous operation.*
- Example `Promise` object

```
function myAsyncFunction(url) {
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.open("GET", url);
    xhr.onload = () => resolve(xhr.responseText);
    xhr.onerror = () => reject(xhr.statusText);
    xhr.send();
  });
}

myAsyncFunction("http://www.cvut.cz").then(successCallback, failureCallback);
```

Overview

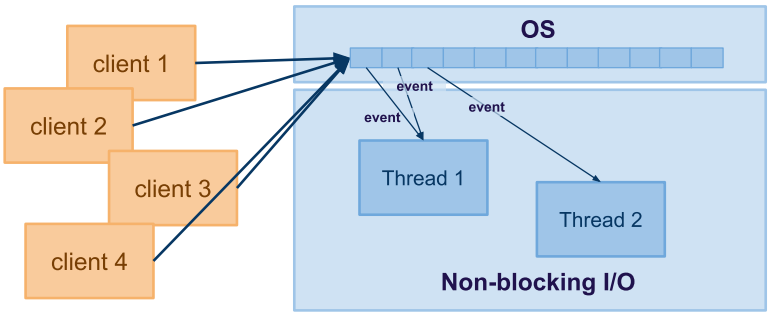
- JavaScript Basics
- Server-side JavaScript

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 look at
  - *Node.js – runs server-side Javascript*

Non-Blocking I/O Model

- Connections maintained by the OS, not the Web app
  - *The Web app registers events, OS triggers events when occur*



- Characteristics
  - *Event examples: new connection, read, write, closed*
  - *The app may create working threads, but controls the number!*
    → much less number of working threads as opposed to blocking I/O

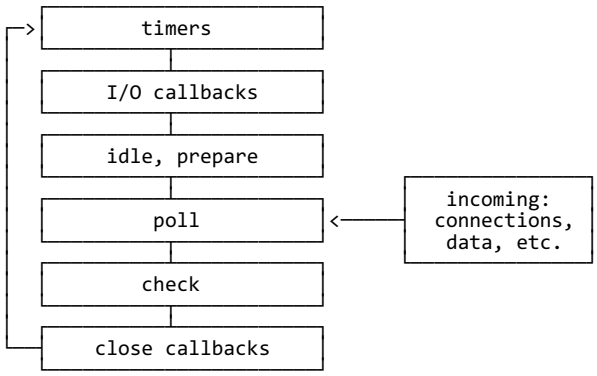
Node.js

- Node.js
  - *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*

```
1  // pseudo code; ask for the last edited time of a file
2  stat( 'somefile', function( result ) {
3    // use the result here
4  } );
```

Node.js Event Loop

- Allows Node.js to perform non-blocking I/O operations.



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

HTTP Server in Node.js

- HTTP Server implementation
  - *server running at* `138.232.189.127`*, port* `8080`*.*

```
1  // http library
2  var http = require("http");
3
4  http.createServer(function(req, res) {
5      // check the value of host header
6      if (req.headers.host == "company.cz") {
7          res.writeHead(201, "Content-Type: text/plain");
8          res.end("This is the response...");
9      } else ;
10         // handle enterprise.com app logic...
11 }).listen('0.0.0.0', 8080);
```

  - *Test it using Telnet*

```
1  telnet 138.232.189.127 8080
2  # ...lines omitted due to brevity
3  GET /orders HTTP/1.1
4  Host: company.cz
5
6  HTTP/1.1 201 OK
7  Content-Type: plain/text
8
9  This is the response...
```

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*

```
1  // create spreadsheet menu
2  function onOpen() {
3    var ss = SpreadsheetApp.getActiveSpreadsheet();
4    var menuEntries = [ {name: "Say Hi", functionName: "sayHi"},
5                        {name: "Say Hello", functionName: "sayHello"} ];
6    ss.addMenu("Tutorial", menuEntries);
7  }
8
9  function sayHi() {
10   Browser.msgBox("Hi");
11 }
12
13 function sayHello() {
14   Browser.msgBox("Hello");
15 }
```


Rhino

- Rhino
  - *open-source implementation of JavaScript written entirely in Java*
  - *managed by the Mozilla Foundation*
    - → *also provides another implementation of JavaScript engine written in C named SpiderMonkey*
  - *typically embedded into Java applications to provide scripting to end users*
  - *core language only and doesn't contain objects or methods for manipulating HTML documents*
  - *enabling development of webapps with JavaScript in containers like Jetty, Tomcat, and Google AppEngine*

```
1  $ cat echo.js
2  for (i in arguments) {
3    print(arguments[i])
4  }
5  $ java org.mozilla.javascript.tools.shell.Main echo.js foo bar
6  foo
7  bar
8  $
```