Web 2.0

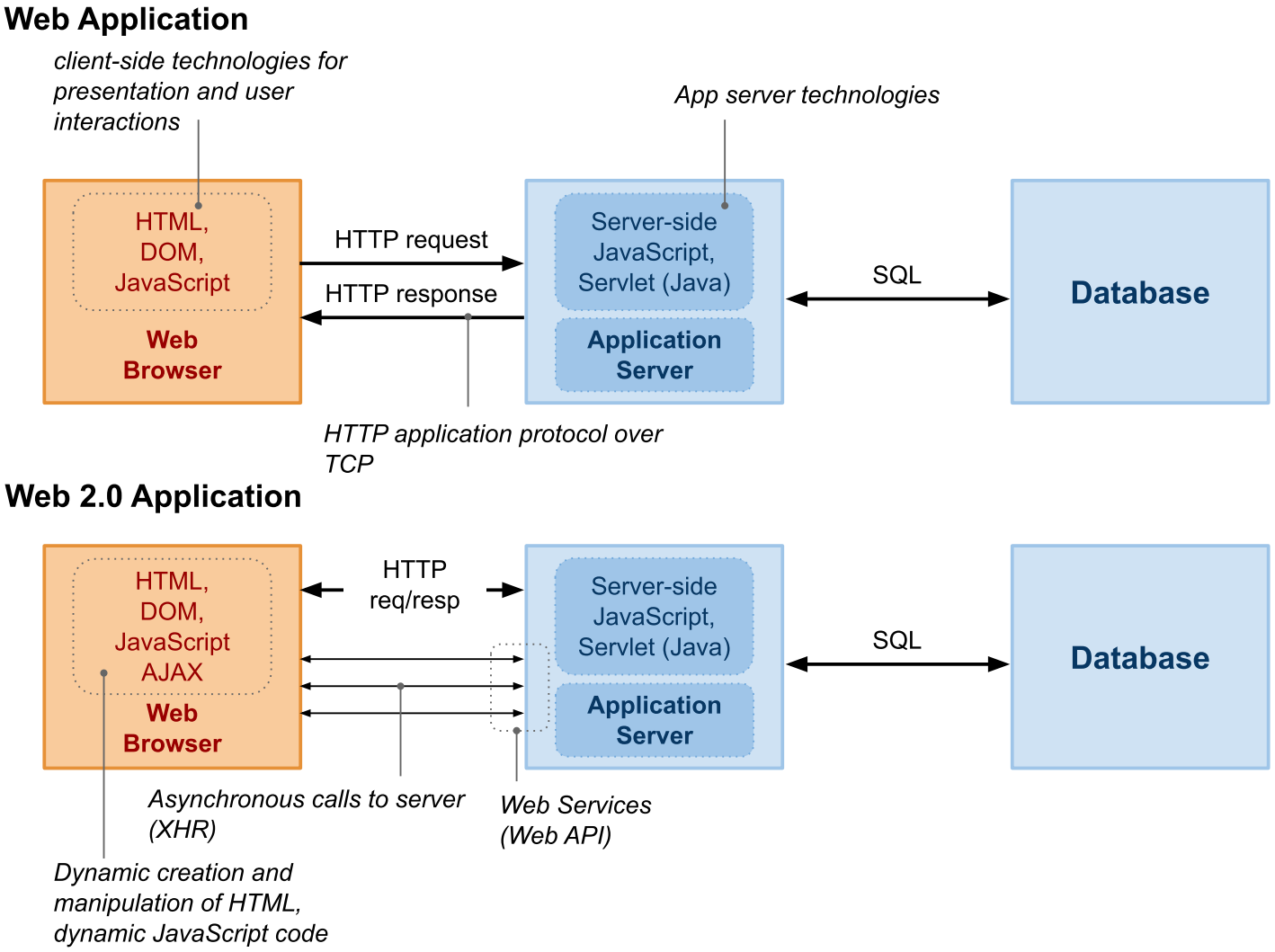
Lecture 1: Introduction to JavaScript

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: Sun Mar 10 2019, 22:00:05
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*

```
1  // function returns the result through a callback, not directly;
2  // this is not a non-blocking I/O, just demonstration of the callback
3  function add(a, b, callback) {
4      callback(a + b);
5  }
6
7  // assign the callback to a variable
8  var print = function(result) {
9      console.log(result);
10 };
11
12 // call the function with callback as a parameter
13 add(7, 8, print);
```

- Functions as values in object

```
1  var obj = {
2      data : [2, 3, "Tomas", "Alice", 4 ],
3
4      getIndexdOf : function(val) {
5          for (var i = 0; i < this.data.length; i++)
6              if (this.data[i] == val)
7                  return i;
8          return -1;
9      }
10 }
11
12 obj.getIndexOf(3); // will return 1
```

Closures

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

```
1  function adder() {
2          var sum = 0;
3          return function(x) {
4                  sum += x;
5                  return sum;
6          }
7  }
8
9  var pos = adder();
10
11 console.log(pos(3)); // returns 3
12 console.log(pos(4)); // returns 7
13 console.log(pos(5)); // returns 12
```

Objects

- `this` problem
  - *A new function defines its own* `this` *value.*

```
1  function Person() {
2    // The Person() constructor defines `this` as an instance of itself.
3    this.age = 0;
4
5    setInterval(function growUp() {
6      // the growUp() function defines `this` as the global object,
7      // which is different from the `this`
8      // defined by the Person() constructor.
9      this.age++;
10   }, 1000);
11 }
12
13 var p = new Person();
```

  - *Solution*

```
1  function Person() {
2    var that = this;
3    that.age = 0;
4
5    setInterval(function growUp() {
6      // The callback refers to the `that` variable of which
7      // the value is the expected object.
8      that.age++;
9    }, 1000);
10 }
```

Arrow Functions

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

```
1  function Person(){
2    this.age = 0;
3
4    setInterval(() => {
5      this.age++; // |this| now refers to the person object
6    }, 1000);
7  }
8
9  var p = new Person();
```

- Syntax, function body

```
1  // concise body syntax, implied "return"
2  var func = x => x * x;
3
4  // with block body, explicit "return" required
5  var func = (x, y) => { return x + y; };
6
7  // object literal needs to be wrapped in parentheses
8  var func = () => ({foo: 1});
```

Callback Hell

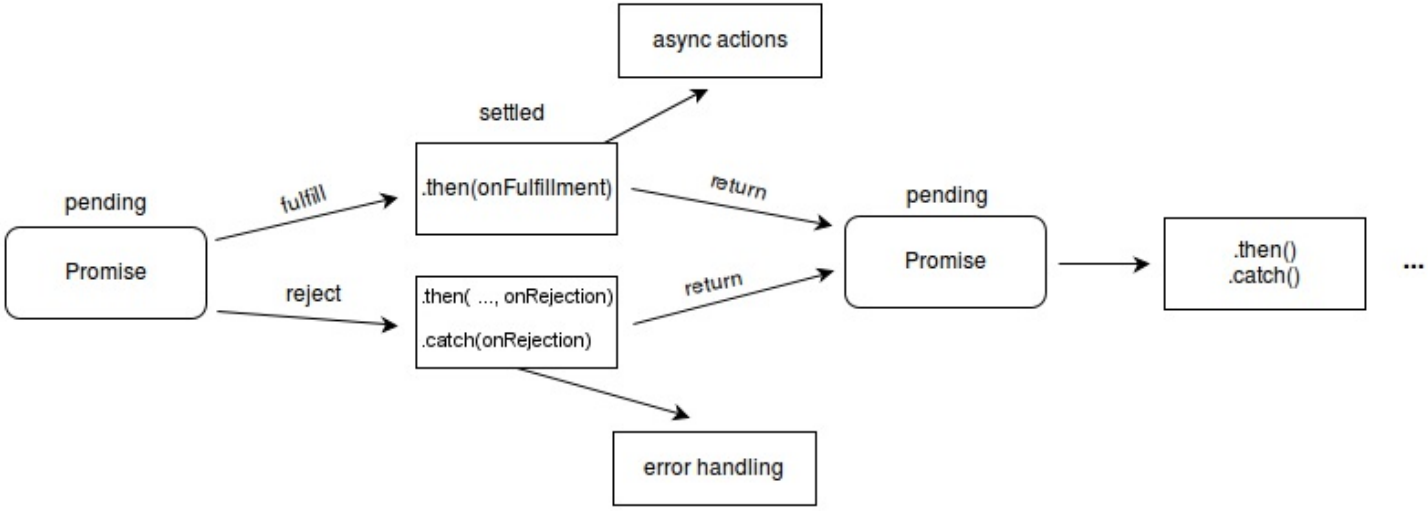
- Callback in callback

```
1  loadScript('/my/script1.js', function(script) {
2
3    loadScript('/my/script2.js', function(script) {
4
5      loadScript('/my/script3.js', function(script) {
6        // ...continue after all script 1,2 and 3 are loaded
7      });
8
9    })
10
11 });
```

  - *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*

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.*



Promise Example

- Example

```
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("some_cors_enabled_url")
.then(
  (responseText) => {
    console.log("success!");
  }
)
.catch(
  (statusText) => {
    console.log("failure!");
  }
);
```

async/await

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

```
1    async function f() {
2      return 1;
3    }
4
5  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*

```
1  async function f() {
2    var promise = new Promise((resolve, reject) => {
3      setTimeout(() => resolve("done!"), 1000)
4    });
5
6    var result = await promise; // wait untill the promise is resolved
7
8    alert(result);
9  }
10
11 f();
```

Overview

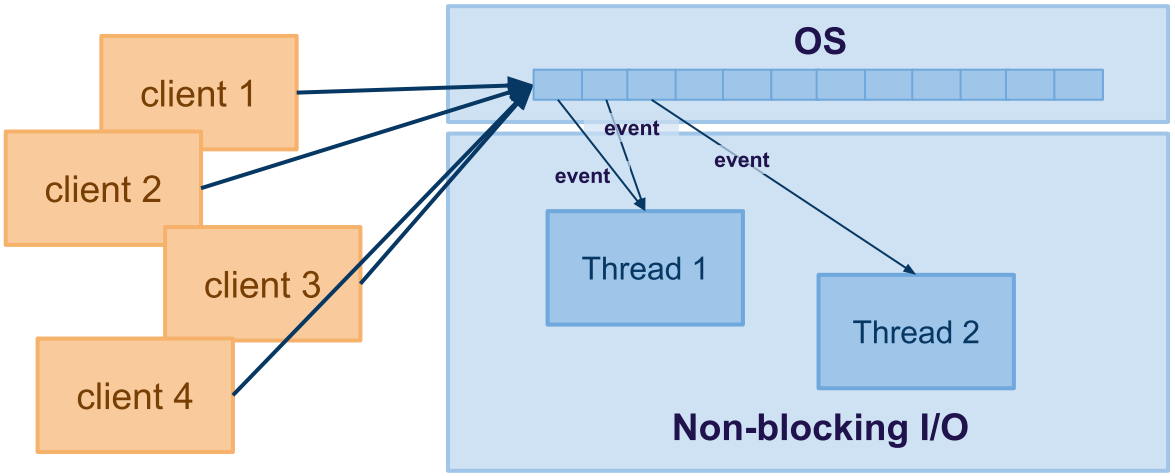
- JavaScript Basics
- Server-side JavaScript

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

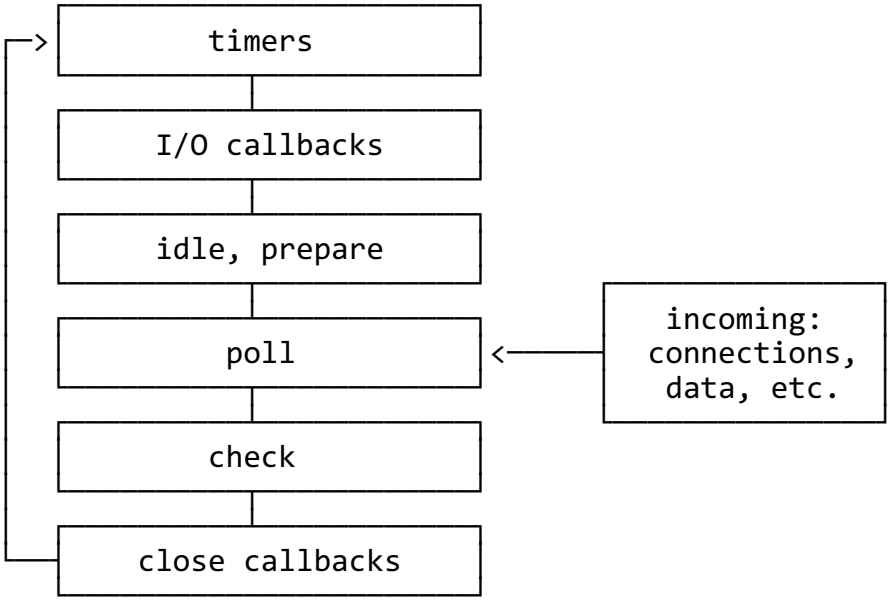
Node.js

- 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 }
```