Consulting
Development
Education

JavaScript

Jaroslav Porubän, Dominik Lakatoš

CDE Services

Further materials

- Chuck Easttom: *Advanced JavaScript*. 3rd Edition, Jones & Bartlett Publishers, 2007, 616 p. ISBN 1598220330.
- Stoyan Stefanov: *Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications and libraries*. Packt Publishing, 2008, 356 p. ISBN 1847194141.
- Frank Zammetti: *Practical JavaScript, DOM Scripting and Ajax Projects.* Apress, 2007, 576 p. ISBN 1590598164.
- JavaScript Tutorial, http://www.w3schools.com/JS/
- Standard ECMA-262 - ECMAScript Language Specification http://www.ecma-international.org/publications/standards/Ecma-262.htm
- Wikipedia
  - JavaScript, http://en.wikipedia.org/wiki/JavaScript
  - JavaScript syntax, http://en.wikipedia.org/wiki/JavaScript_syntax
  - JSON, http://en.wikipedia.org/wiki/JSON
  - AJAX, http://en.wikipedia.org/wiki/Ajax_(programming)

CDE Services

Content

- Language basics
  - variables
  - expressions, operators
  - statements
  - functions
- Object oriented programming in JavaScript
- JavaScript API
- DOM API
- Creation of applications in JavaSript, debugging
- Ajax, JSON

CDE Services

JavaScript language

- Script language
  - Prototype based – object oriented approach
  - Dynamic
  - Weakly-typed
  - Functions as variable values – first-class value
  - Syntax similar with C, Java (not OOP parts)
- JavaScript is mostly used in web browser – client-side JavaScript
- *"Java is to JavaScript what Car is to Carpet"*

CDE Services

History

- Created by Netscape as LiveScript
- Later renamed to JavaScript (Sun Microsystems) thanks to Java popularity
- March 1996 – Netscape Navigator 2.0
- August 1996 – Internet Explorer 3.0 JScript
- 1996 standard by ECMA - ECMAScript – 3 base dialects
  - JavaScript
  - ActionScript
  - JScript

CDE Services

DOM – Document Object Model

- Object representation of document
- Allows for manipulation with document and provides platform-independend interface for interacting with document in web browser
  - travers document, find out its structure
  - modify document
  - catch events (e.g. click)
  - Interaction with web browser – window, frames, forms, status bar, history, navigation

JavaScript HelloWorld – internal script

```
<html>
  <head>
    <title>simple page</title>
  </head>
  <body>
    <script type="text/javascript">
       document.write('Hello World!');
    </script>
    <noscript>
      <p>Your browser either does not support
    JavaScript, or you have JavaScript turned
    off.</p>
    </noscript>
  </body>
</html>
```

JavaScript HelloWorld – external script

```
<html>
  <head><title>simple page</title></head>
  <body>
    <script type="text/javascript"
       src="test.js">
    </script>
    <noscript>
      <p>Your browser either does not support
    JavaScript, or you have JavaScript turned
    off.</p>
    </noscript>
  </body>
</html>
```

test.js

```
document.write('Hello World Other!');
```

CDE Services

Consulting
Development
Education

Language basics

CDE Services

Comments

- Same as language Java (C, C++)
- One-line comment

```
// a short, one-line comment
```

- Multi-line comments

```
/* this is a long,
   multi-line comment,
   spanning multiple lines. */
```

- Embedding of comments is not supported – cannot put multi-line comment into another multi-line comment

CDE Services

Variables

- Variable is named cell, which serves for storing a specified value
- Variable does not have defined type – weakly-typed system
- Value carries own type information
- One variable can store different types (not at once)
- Keyword `var` serves for declaration of variable with local scope

```
var variable = 3;
variable = "Jano";
variable = 3.8;
```

- Redeclaration of variable does not cancel its previous value

```
var x = 5;
var x;
```

CDE Services

Identifiers

- JavaScript is **case sensitive** in names of identifiers
  - First character must be a letter or an underscore _
  - Next charactes can be a letter, number, underscore or number (0 to 9).
  - Identifier cannot be a reserved word (keyword)
- Variables mostly use lowercase letters, words are separated with “camel case” or underscores

```
number_of_digits
numberOfDigits
```

CDE Services

Data types

- Primitive data types
  - **undefined** value
  - **null**
  - **Number**
  - **String**
  - **Boolean**
- Objects
- Arrays (specific type of object)

CDE Services

Undefined type, Null type

- **Undefined** value
  - Declared, not assigned value

```
var a;
console.log(a); //undefined
```

- Type **null**
  - Contains only one value – `null`
  - Value `null` is not from any other type

```
console.log(undefined == null) // true
console.log(undefined === null) //false
```

CDE Services

Number type

- The only numerical type in Javascript for representation of integers and decimal numbers, IEEE-754 Double
- Literals for numbers

```
345
34.5
3.45e2
0377        octal notation
0xFF        hexadecimal notation
Infinity          positive infinity
-Infinity         negative infinity
NaN               Not a Number, e.g. after conversion
```

- Inaccuracy with big numbers

```
var a = 12345678901234567
console.log(a)
```

CDE Services

String type

- String is a sequence of characters
- String literals

```
var s1 = "Hello, world!";
var s2 = 'Greetings';
var s3 = '"Hello!" he said.';
var s4 = 'Hello"; wrong!!!
```

- String is not changeable (immutable)
- Escape sequences for strings
  - \n \t \\ \' \"

CDE Services

Boolean type

- Logical values
- Data type defines 2 values and appropriate literals
  - **`false`** and **`true`**
- It is not recommended to use numbers as logical values (as it is required in some other languages e.g. C language)

```
console.log(0 == false)   //true
console.log(1 == true)    //true
console.log(0 === false) //false
console.log(1 === true)   //false
```

CDE Services

Operators

- Used in expressions
  - aritmetic
  - assignment
  - comparative
  - logic
  - bitwise
  - string

CDE Services

Arithmetic operators

+	addition
–	subtraction
*	multiplication
/	division
%	remainder of division (modulo)
+	unary plus
–	unary minus
++	increment (prefix, postfix)
––	decrement (prefix, postfix)

CDE Services

Assignment operators

=	assignment
+=	addition and assignment
-=	subtraction and assignment
*=	multiplication and assignment
/=	division and assignment
%=	modulo and assignment

CDE Services

Comparison and logic operators

==	equals
!=	not-equals
>	larger than
>=	larger or equals than
<	lower than
<=	lower or equals than
===	equals and same type
!==	not equals or not same type
&&	logical conjunction (and)
||	logical disjunction (or)
!	logical negation

CDE Services

Bitwise and string operators

&	bitwise and
|	bitwise or
^	xor
<<	shift left (adds zeroes)
>>	shift right (adds sign bit)
>>>	shift right (adds zeroes)
~	bit inversion (not)
+	string concatenation
+=	string concatenation and assignment

CDE Services

Conditional operator

- Similar to *if...then...else*
- Result is a value

```
condition ? expression : expression
```

- Notation

```
result = (condition) ? expression :
   alternative;
```

- is equal with

```
if (condition)
   result = expression;
else
   result = alternative;
```

- Example

```
console.log(x > 6 ? "large" : "small");
```

CDE Services

Special operators

- Operator `typeof` allows to determine type of value (variable)

```
var a = 10;
console.log(typeof a); //"number"
if (typeof a === "number") {. . .}
```

- Operator `instanceof` allows to determine if value is of some specific Object
- Operator `void` evaluates any expression to `undefined` value, effectively discarding expression's return value

CDE Services

Comparison of types in JS - problems ahead

Always use === for comparison of any type

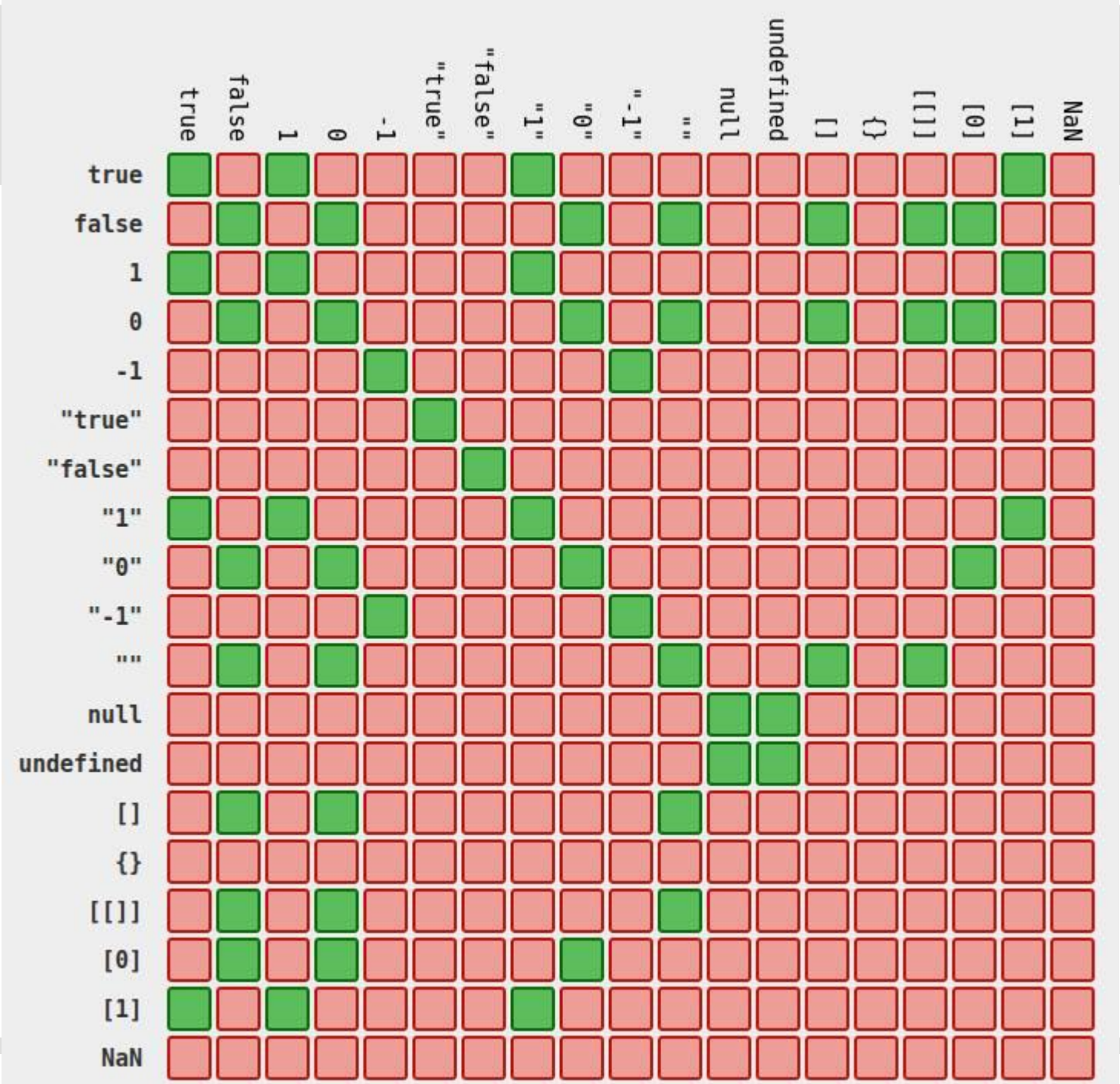
use of == is more dangerous - automatic type casting!!!

```
'' == '0'                  // false
0 == ''                 // true
0 == '0'                   // true
false == 'false'           // false
false == '0'               // true
false == undefined // false
false == null              // false
null == undefined          // true
' \t\r\n ' == 0 // true
```

Javascript type conversion explained

CDE Services

Equality table



CDE Services

Statements

- Usage of semicolon at the end of each statement is recommended but not required when each statement is on own line
- Block of statements (sequence of statements) can be created with curly brackets `{ a }`
- Example

```
{
   var x;
   x = 7 + x;
   document.write(x);
}
```

CDE Services

Statement `if`

- Conditional statement

```
if(expression)
   statement
```

- If `expression` is evaluated as `true` value then `statement` is executed
- Example

```
if (x <= 10)
   document.write("small");
```

CDE Services

Statement `if...else`

- Conditional statement with `else` branch

```
if(expression)
   statement1
else
   statement2
```

- If `expression` is evaluated to `true` value than `statement1` is executed otherwise `statement2` is executed
- Example

```
if (x <= 10)
   document.write("small");
else
   document.write("large");
```

CDE Services

Loop statement `while`

- Loop statement with condition specified at the beginning

```
while (expression)
    statement
```

- `statement` is executed as long as specified condition (`expression`) is evaluated to `true`
- Example

```
var i = 0;
while (i < 10) {
    document.write("*");
    i++;
}
```

CDE Services

Loop statement `do...while`

- Loop statement with condition specified at the end

```
do
    statement
while (expression)
```

- `statement` is executed at least once and as long as a specified condition (`expression`) is evaluated to `true` value
- Example

```
var i = 0;
do {
    document.write("*");
    i++;
} while (i < 10);
```

CDE Services

Loop statement `for`

- Loop statement

```
for (initial; condition; operation)
    statement
```

- Shorter and more advanced version of `while`. Before loop starts statement `initial` is executed and after each iteration statement `operation` is executed
- Example

```
for (var i = 0; i < 10; i++)
 document.write("*");
```

CDE Services

Loop statement `for...in`

- Loop statement

```
for (var prop-name in object-name)
    statement using object-name[prop-name];
```

- Statement for iterating over all items of array or properties of object
- Example

```
var x = [1, 2, 3, 5, 7, 9];
for (var i in x) {
    document.write(x[i]);
}
```

CDE Services

Statements
`break` and `continue`

- Statement `break` ends execution of any loop or `switch` statement. It "jumps out" of a loop.

```
var i = 0;
for(;;){
    if (i == 5)
        break;
    i++;
    document.write(i);
}
```

- Statement `continue` breaks one iteration and continues with the next iteration in the loop
- With `labels` it is possible to ends any block or loop in the code.

```
label: statement
```

CDE Services

Statement `switch`

- `switch` statement is used to perform different action based on different conditions.

```
var theDay = 3;
switch (theDay)
{
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this
   weekend!");
}
```

CDE Services

Statement `try...catch...finally`

- Errors will happen! ☺
- Catching errors in block

```
try {
  console.log(x); //error
} catch (err) {
  alert(err.description);
} finally {
  console.log("Always done");
}
```

- If there is an exception during execution of `try` block it is possible to react with statements inside `catch` block
- Block `finally` is optional part but is executed always

CDE Services

Statement `throw`

- Statement for propagation of exception (error)

  ```
  throw expression
  ```

- Value of `expression` can be String, Number, Boolean or Object
- Example

```
try {
  throw "error";
} catch (e) {
  console.log(e);
}
```

- Exception is propagated to the exception handler block in the opposite direction of function calls

CDE Services

Statement `with`

- Statment `with` allows to set predefined object on which next statements will be executed
- NOT RECOMMENDED to use as it introduces confusion and some bugs
- Example

```
with(document) {
    var a = getElementById('a');
    var b = getElementById('b');
    var c = getElementById('c');
};
```

CDE Services

Dialog box

- Alert Box

```
alert("I am an alert box!");
```

- Confirm Box

```
var r = confirm("Press a button");
if (r == true)
    document.write("Pressed OK!");
else
    document.write("Pressed Cancel!");
```

- Prompt Box

```
var name = prompt("Enter your name",
   "Bond");
if (name != null && name != "")
  document.write("Hello " + name);
```

CDE Services

Function

- Function is a parametrized block of statements
- Function has a body (usually not empty block os statements) and list of arguments (can be empty)
- In case that function is not ended with `return` statement the returned value is `undefined`
- Example

```
function sum(x, y) {
  return x + y;
}

console.log(sum(2, 3));
```

CDE Services

Variables – local and global

```
x = 0; //A global variable
var y = 'Hello!'; //A global variable

function f(){
  var z = 'foxes'; //A local variable
  twenty = 20; //twenty is global
     //because declared without var
  return x; //x here is global
}

// The value of z is no longer available
f();  //Call function f
var f = twenty; //twenty is global
```

CDE Services

Closures

- Inner function can use „hidden“ data from outer (parent) function
- Usefull for data hiding

```
var digit_name = function() {
  var names = ["zero", "one", "two",
   "three", "four", "five", "six", "seven",
   "eight", "nine"];

  return function(n) {
     return names[n];
  };
}();

alert(digit_name(3)); // "three"
```

Automatic semicolon

- Javascript adds semicolon at the end of line when error

```
return
{
    ok: false
};
```

Bad style
bad results

```
return;
{
    ok:
false;
}
```

```
return {
    ok: false
};
```

Correct style,
correct result
☺

CDE Services

Test your code

- Javascript allows many bad practices in code
- Try to avoid them by using practices recommended by developers
  - Test your code: http://www.jslint.com/

CDE Services

Consulting
Development
Education

Object oriented programming in JavaScript

CDE Services

Objects

- Object oriented programming is based on prototypes (slots)
- Object – structure of **data and functions**
- Variables of object = field or property

  `objectReference.propertyName`

  or

  `objectReference["propertyName"]`
- Functions of object – methods

  `objectReference.methodName([arguments])`

  resp.

  `objectReference["methodName"]([arguments])`
- Example

```
document.write("hello!")
document["write"]("hello!")
```

Constructors and creation of objects

- Constructor is a special function for defining default values of created object
- Keyword `this` – reference to actual object – object on which the method was called

```
function Student(name, age)
{
  this.name = name;
  this.age = age;
}

var student1 = new Student("Jano", 16);
console.log(student1.name
   + " " + student1.age);
```

CDE Services

Removing objects

- Performed automatically – garbage collector
- Can be deleted manually with `delete`

```
var student1 = new Student("Jano", 16);
console.log(student1.name + " " + student1.age);

student1.nick = "John";
console.log(student1.name + " " +
   student1.nick);

delete student1.nick;
console.log(student1.nick);

delete student1;
```

CDE Services

Function as a value

- Function is also a value and can be assigned to variable

```
function sum(x, y) {
  return x + y;
}

var f1 = sum;

var f2 = function (x, y) { return x + y; }

console.log("sum " + sum(2,3));
console.log("f1 " + f1(2,3));
console.log("f2 " + f2(2,3));
```

CDE Services

Definition of method

- Method is a function on object

```
function displayStudent()
{
    var result = "";
    result += this.name + " - " + this.age;
    result += "<BR>";
    document.write(result);
}

function Student(name, age)
{
    this.name = name;
    this.age = age;
    this.display = displayStudent;
}

var student1 = new Student("Jano", 16);
student1.display();
```

CDE Services

Inheritance

- Inheritance is realized using the prototypes
- Each object has property `prototype`

```
function Base() {
  this.anOverride = function() {};

  this.aBaseFunction = function() {};
}

function Derive() {
  this.anOverride = function() {};
}

var base = new Base();
Derive.prototype = base;
var der = new Derive();
der.aBaseFunction();
```

Objects – literals

- Objects are represented as associative array (map) – named slots

```
var student1 = {
   "name" : "James",
   age : 33,
   nick : "Bond"
};

console.log(student1.name + " " +
   student1.age);

console.log(student1["name"] + " " +
   student1["age"]);
```

CDE Services

Consulting
Development
Education

JavaScript API

CDE Services

Global object

- Created before first script execution
- Reserved values
  `NaN, Infinity, undefined`
- Evaluation of string as script (NOT SAFE)
  `eval (x)`
- Transformation of string to number
  `parseInt(string, radix), parseFloat(string)`
- Testing number for specific value: NaN, Infinity
  `isNaN(number), isFinite(number)`
- URI manipulation
  `decodeURI(encodedURI), encodeURI(uri)`
- Constructors for predefined object types
  `Object, Function, Array, String, Boolean, Number, Date, RegExp`
- Object for mathematic operations
  `Math`

CDE Services

Arrays

- Array is structured data type
- The individual items are accessed using the index (0 … array size - 1)

```
var myArray = [];
myArray.push("hello world");
alert(myArray[0]);
alert(myArray.length);
myArray[100] = 88;
alert(myArray[100]);
alert(myArray.length);
```

CDE Services

`Array` object prototype

- Length of the array
  `length`
- Conversion to string
  `toString()`
- Taking out the last element of the array
  `pop()`
- Inserting element to the end of the array
  `push(`*`item`*`)`
- Joining arrays to one array (concatenation)
  `concat([`*`item1`*`[, `*`item2`*`[, … ]]])`
- Joining elements of array to String, each element is separated by specified separator
  `join(`*`separator`*`)`
- Changing the order of elements in the array to oppossite
  `reverse()`
- Taking out the first element of the array
  `shift()`

CDE Services

`Array` object prototype

- Selecting part of the array - sub-array
  `slice(start, end)`
- Inserting elements to the front of the array
  `unshift([item1[, item2[, …]]])`
- Getting the first index of specified element in the array
  `indexOf(searchElement[, fromIndex])`
- Getting the last index of specified element in the array
  `lastIndexOf(searchElement[, fromIndex])`
- Testing each element of the array to fulfill the predicate
  `every(callbackfn[, thisArg])`
- Testing elements of the array if at least one fulfulls the predicate
  `some(callbackfn[, thisArg])`
- Executing action for each element of the array
  `forEach(callbackfn[, thisArg])`

CDE Services

`Array` object prototype

- Filter elements which fulfill to predicate
  `filter(callbackfn[, thisArg])`
- Mapping elements of the array to new array
  `map(callbackfn[, thisArg])`
- Reduces elements of array to single value
  `reduce(callbackfn[, initialValue])`
- Reduces elements starting from last element
  `reduceRight(callbackfn[, initialValue])`
- Callback functions should be in form
  `function(item, index, object)`
  or for `reduce`, `reduceRight`:
  `function(previousItem, actualItem, index, object)`

CDE Services

Object representing arguments

- It is possible to create function with variable number of arguments
- Array contains all arguments, which have been used for calling the function

```
function test() {
  console.log("len = " + arguments.length)

  for(var i = 0; i < arguments.length; i++) {
     console.log("item" + i + " = "
          + arguments[i]);
    }
}

test(1,2,3);
```

CDE Services

`String` object prototype

- Length of string

  `length`
- Getting the character from position

  `charAt(`*`pos`*`)`
- Joining (concatenation) of strings

  `concat([`*`string1`*`[, `*`string2`*` [, …]]])`
- Getting the first starting index of substring in string

  `indexOf(`*`searchString`*`, `*`position`*`)`
- Getting the last starting index of substring in string

  `lastIndexOf(`*`searchString`*`, `*`position`*`)`

CDE Services

`String` object prototype

- Getting substring from string

  `substring(`*`start, end`*`)`
- Transforming string to lowercase letters

  `toLowerCase()`
- Transforming string to uppercase letters

  `toUpperCase()`
- Spliting string to substrings according to defined separator; result is an array of strings

  `split(`*`separator[, limit]`*`)`

CDE Services

`Number` object prototype

- Greatest positive number
  `Number.MAX_VALUE`
- Lowest positive number
  `Number.MIN_VALUE`
- Not a Number
  `Number.NaN`
- Negative infinity
  `Number.NEGATIVE_INFINITY`
- Positive infinity
  `Number.POSITIVE_INFINITY`
- Transformation to string with optional radix specification
  `toString([radix])`
- Transformation to localized string
  `toLocaleString()`
- Transformation to string with fixed numbers of digits after floating point
  `toFixed(fractionDigits)`

`Math` object

- Mathematic functions and constants

```
Math.PI, Math.E
Math.abs(v)
Math.ceil(v), Math.round(v),
Math.floor(v)
Math.pow(v1, v2), Math.sqrt(v)
Math.sin(v), Math.cos(v),
Math.tan(v)
Math.random()
Math.max([value1[, value2[, … ]]])
Math.min([value1[, value2[, … ]]])
```

CDE Services

`Date` object prototype

- Working with date and time
- Constructor

```
new Date([year, [month [, date
[, hours [, minutes [, seconds
[, ms ]]]]]]])
```

- Conversion to string

```
toString(), toDateString(),
toTimeString(),
toLocaleString(),
toLocaleDateString(),
toLocaleTimeString()
```

CDE Services

RegExp objects

- Serves for testing string conformity to defined pattern (regular expression)
- Designed according to Perl regular expressions
- Regular expressions in JavaScript can be written in form of literals in form:

```
/pattern/modifiers
```

- Example

```
var r = /[0-9]+/g;
var s = "abc12def345gh";
//["12", "345"]
console.log(s.match(r));
```

CDE Services

Regular expressions – modifiers, properties

- Modifiers (flags)
  - `g` – global – next execution uses last position (without this flag it starts from beginning)
  - `m` – multiline – changes behavior for beginning and end of line to allow more lines to match between characters ("^" and "$")
  - `i` – ignoreCase – ignores letter size
- Vlastnosti
  - `source` – original regular expression
  - `lastIndex` – last checked index of matching operation

CDE Services

Regular expressions

[abc]	only character a or b or c
[^abc]	every character but a, b, c
[a-zA-Z]	a to z or A to Z
^	Line start
$	Line end
X?	X once or not at all
X*	X zero, once or multiple times
X+	X once or multiple times
X{n}	X exactly n-times
X{n,}	X at least n-times
X{n,m}	X at least n-times and maximum m-times
XY	X following with Y
X|Y	X or Y
(X)	capturing group
(?:X)	non-capturing group
\d \D \w \W\s \S	predefined character sets

CDE Services

JavaScript Timers

- `setTimeout`
  - Execute operation **once** with a delay
  - `setTimeout (expression, timeout);`
  - returns ID – can be used to cancel with `clearTimeout(ID);`
- `setInterval`
  - Execute operation in loop in specified interval
  - `setInterval (expression, interval);`
  - returns ID – can be used to cancel with `clearInterval(ID);`

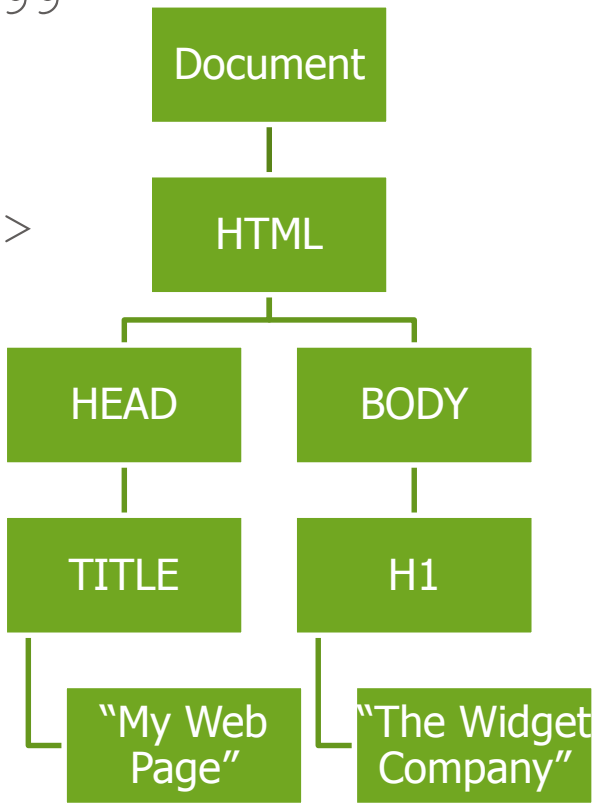
CDE Services

Consulting
Development
Education

Document Object Model

Example of DOM of HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    strict.dtd">
<html
    xmlns="http://www.w3.org/199
    9/xhtml" xml:lang="en"
    lang="en">
  <head>
    <title>My Web Page</title>
  </head>
  <body>
    <h1 class="intro">The
   Widget Company</h1>
  </body>
</html>
```



CDE Services

Events and processing of events

- Every element in documents can create set of events, which can used as trigger for executing Javascript code
- Using the attribute of element it is possible to define code, which will be executed in case of event – event handler
- Examples of events
  - mouse button pressed
  - mouse movement over the element
  - selecting an item from list
  - pressing a button on keyboard
  - sending a form

CDE Services

Types of events

- `onabort` – Interuption of image loading
- `onblur` – Loosing the focus on an element
- `onchange` – Change of value in an element
- `onclick` – Mouse button click
- `ondblclick` – Mouse button double click
- `onerror` – Error during loading of page or image
- `onfocus` – Getting the focus on an element
- `onkeydown` – Keyboard key went down
- `onkeypress` – Keyboard key was pressed (and released)
- `onkeyup` – Keyboard key was released
- `onload` – Successful loading of page or image

CDE Services

Typu udalostí

- `onmousedown` – Mouse button pressed
- `onmousemove` – Mouse moved
- `onmouseout` – Mouse cursor left an element
- `onmouseover` – Mouse cursor went over an element
- `onmouseup` – Mouse button released
- `onreset` – Reset button pressed (on form)
- `onresize` – Window size changed
- `onselect` – Text selected
- `onsubmit` – Form submitted
- `onunload` – Leaving the web page

CDE Services

Handling the event – example

```
<html>
<head>
<script type="text/javascript">
function upperCase() {
 var x = document.getElementById("fname").value
 document.getElementById("fname").value = x.toUpperCase()
}
</script>
</head>

<body>
    Enter your name:
    <input type="text" id="fname" onblur="upperCase()">
</body>
</html>
```

CDE Services

Events `onload` and `onunload`

- Are created when user opens(leave) page
- `onload`
  - handling browser detection
  - initialization settings
  - reading/setting of cookies
- `onunload`
  - setting of cookies

CDE Services

Event `onsubmit`

- Handling of this event is mostly to validate data for submittion
- Return value can be
  - true – if form was validated successfully and can be submitted to server
  - false – if form didn't pass validation and won't be submitted to server

```
<form method="post"
  action="cde.html"
  onsubmit="return checkForm()">
```

CDE Services

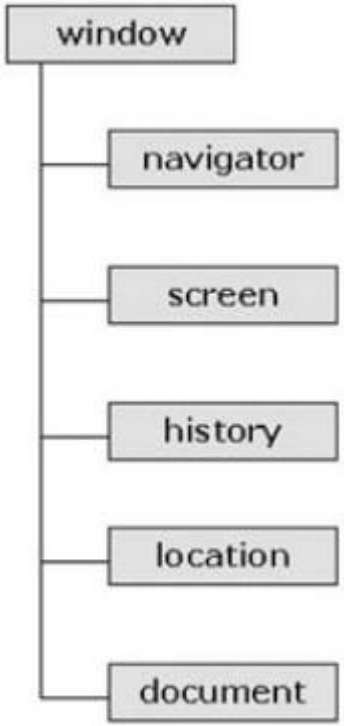
HTML DOM

- HTML DOM defines standard set of objects for representation and manipulation of HTML document
- Every part of HTML is accessible through DOM also with content and attributes
  - Everything can be changed
- HTML DOM is interface not dependent on any language
  - We are using Javascript, but it is possible to use DOM even with other languages as VBScript, Java etc.

CDE Services

DOM object types

- Anchor
- Document
- Event
- Form
- Frame, Frameset, IFrame
- Image
- Location
- Navigator
- Option a Select
- Screen
- Table, TableHeader, TableRow, TableData
- Window object



Getting HTML object with identifier

- Every HTML object with unique identifier can be accessed using following method

  `document.getElementById()`
- Returns one object (element)

```
function getElement() {
    var x = document.getElementById("myHeader");
    alert("I am a " + x.tagName + " element");
}
```

```
<h1 id="myHeader" onclick="getElement()">Click
    to see what element I am!</h1>
```

CDE Services

Getting HTML object by name inside form

- Objects with specified name (usually inside form) can be accessed with following method `document.getElementsByName()`
- Returns multiple objects (elements)

```
function getElements() {
    var x = document.getElementsByName("myInput");
    var message = "Count = " + x.length + "\n";
    for(var i = 0; i < x.length; i++) {
        message += "Value = " + x[i].value + "\n";
    }
    alert(message);
}

<input name="myInput" type="text" size="20"><br />
<input name="myInput" type="text" size="20"><br />
<input name="myInput" type="text" size="20"><br />
<br />
<input type="button" onclick="getElements()"
    value="Test it">
```

CDE Services

Getting and modifying the content of element

- DOM object property `innerHTML` can be used to fully manipulate HTML code inside of this element

```
function changeContent() {
    var element =
        document.getElementById("testDiv");
    alert(element.innerHTML);
    element.innerHTML =
        "<h1>This is new content</h1>";
}

<div id="testDiv">
    This is default value.
</div>

<input type="button" onclick="changeContent()"
    value="Test it">
```

CDE Services

Forms

- Forms are accessible through its name, same applies to subelements – possible to use dotted notation
- Property `value` contains actual value of `input` element

```
function validate() {
    var fname = document.testForm.fname.value;
    var age = document.testForm.age.value;
    var email = document.testForm.email.value;
    //Do validation
}

<form name="testForm" onsubmit="return validate()">
    Name (max 10 chararcters):
    <input type="text" id="fname" size="20"><br />
    Age (from 1 to 100):
    <input type="text" id="age" size="20"><br />
    Email:
    <input type="text" id="email" size="20"><br />
    <br />
    <input type="submit" value="Submit">
</form>
```

CDE Services

DOM

- Everything is DOM node
  - Document
  - Elements
  - Attributes
  - Text
  - Comments (in HTML)
- Watch out for whitespaces
  - also text node

CDE Services

DOM – properties of nodes

- Every DOM node contains properties
  - `nodeType`
    - It is number representing type of node (element, attribute, text...)
  - `nodeName`
    - Name of node, e.g. name of `h1` element is `"H1"`
  - `nodeValue`
    - Value of node, for elements it is `null`, for textual nodes it is textual content, for attributes it is value assigned to attribute

CDE Services

DOM – node types (constants)

Value	Constant
1	**Node.ELEMENT_NODE**
2	**Node.ATTRIBUTE_NODE**
3	**Node.TEXT_NODE**
4	Node.CDATA_SECTION_NODE
5	Node.ENTITY_REFERENCE_NODE
6	Node.ENTITY_NODE
7	Node.PROCESSING_INSTRUCTION_NODE
8	**Node.COMMENT_NODE**
9	**Node.DOCUMENT_NODE**
10	Node.DOCUMENT_TYPE_NODE
11	Node.DOCUMENT_FRAGMENT_NODE
12	Node.NOTATION_NODE

CDE Services

DOM – relations between nodes

- Every node has property for accessing related nodes
  - `childNodes`
  - `firstChild`
  - `lastChild`
  - `nextSibling`
  - `previousSibling`
  - `parentNode`
- Attibutes can be accessed by
  - property `attributes`
  - `getAttributeNode(name);` //node
  - `getAttribute(name);` //value

CDE Services

Changing DOM - adding

- Adding element to DOM
  - `elementVar = document.createElement( tagName );`
  - `element.appendChild( elementVar );`
    - Adds element to end of parent element
- Example:

```
var welcome = document.getElementById( "welcome" );
var horizRule = document.createElement( "hr" );
var paragraph = welcome.firstChild;
welcome.insertBefore( horizRule, paragraph );
```

CDE Services

Changing DOM

- Removing an element
  - `removedElement = element.removeChild( elementToRemove );`
- Replacing an element
  - Remove + add on the same place
  - `element.replaceChild(newElem, oldElem);`
- Example

```
var welcome = document.getElementById( "welcome" );
var horizRule = welcome.lastChild;
var newPara = document.createElement( "p" );
newPara.appendChild( document.createTextNode( "Feel free to have a browse." ) );
welcome.replaceChild( newPara, horizRule );
```

CDE Services

Changing DOM

- Element can be only on one place in DOM
- Moving element
  - Getting element from its place
  - Inserting element to new place
  - Element is automatically moved to the new place – removed from the previous place

```
<ul>
  <li id="widget1"><a
    href="superwidget.html">SuperWidget</a></li>
  <li id="widget2"><a
    href="megawidget.html">MegaWidget</a></li>
  <li id="widget3"><a
    href="wonderwidget.html">WonderWidget</a></li>
</ul>

var superWidget = document.getElementById( "widget1" );
var ul = superWidget.parentNode;
ul.appendChild( superWidget );
```

CDE Services

Changing attributes in DOM

- `Att = getAttributeNode( name );`
  - `Att.value`
- `document.createAttribute( attributeName );`
- `element.setAttributeNode( attributeNode );`
- `element.removeAttributeNode( attributeNode );`
- Shorter and cleaner way
  - `element.setAttribute( name, value );`
  - `element.removeAttribute( name );`

CDE Services

Consulting
Development
Education

JSON

CDE Services

JSON

- **JSON** - JavaScript Object Notation
- Format for exchange of structured data
  - Textual – easy to read
  - Easy processing
  - Language independent
  - Subset of ECMAScript – object literals
- http://www.json.org/
- http://www.json.org/json2.js
- JSON is subset of YAML

CDE Services

Data types

- Simple data types: string, number, boolean
- Structured data types
  - arrays
  - objects
- Does not allow to serialize cyclic structures

```
{
   "name" : "James Bond",
   "age" : 33,
   "agent" : true,
   "hobbies" : ["cars", "women"]
};
```

CDE Services

Serialization – object to JSON text

- Transformation of Javascript values (objects) to JSON
  - It skips functions

```
var str =
  JSON.stringify(object);
```

CDE Services

Deserialization – JSON text to object

- Transformation of JSON text to object/array

```
try {
   object = JSON.parse(str);
} catch(e) {
   //catch parsing exception
}
```

- Transformation with `eval` function is dangerous !

CDE Services

Consulting
Development
Education

HTML5

Dominik Lakatoš

CDE Services

HTML revisited

- HTML5 is combination of
  - HTML
  - CSS
  - Javascript
- New features
  - Canvas – for drawing
  - Local storage – small browser database for each page
  - Video – play and control video
  - Manipulate history
- Great further reading: http://diveintohtml5.info/
  - Games: http://bombermine.com, http://play-ttd.com/

CDE Services

Consulting
Development
Education

Dominik Lakatoš

CDE Services

Features of jQuery

- **Javascript library**
- **Cross browser support**
- DOM manipulation made easy
- Reuse CSS selector knowledge
- Change CSS / classes simply
- Great AJAX support
- Easy to learn and adapt
- Extensible – plenty of plugins
- NOT for static pages

CDE Services

jQuery

- http://jquery.com/

CDE Services

Include jQuery

Online

```
<html>
 <head>
   <script
     src="http://code.jquery.com/jquery-
1.10.2.min.js">
   </script>
 </head>
 <body>
   …
 </body>
</html>
```

Downloaded

```
<script src="jquery-1.10.2.min.js"></script>
```

CDE Services

jQuery selectors

- Similar to CSS3 selectors + added functionality
- Use `$("selector")` function
  - `$` is function name (shortcut for `jQuery`)
- Returns set of corresponding elements
  - jQuery functions works natively on set of elements
- Example (jQuery selector API) :

`$("h2")` – get elements by tag name

`$("#tagId")` – get elements by ID

`$(".className")` – get elements by class name

`$("h1.title")` – any combination

CDE Services

Example – jQuery works on sets

- Change of innerHTML for all <p> elements
- Classical Javascript

```
  var elementy =
document.getElementsByTagName("p");
  for (var i=0; i < elementy.length; i++) {
    elementy[i].innerHTML = "zmeneny text";
   }
```

- jQuery

```
$("p").html("zmeneny text");
```

jQuery after load

- Running JS after page has loaded
  - Cannot change DOM before finished loading
  - Have to wait to draw full web page

```
$(document).ready(function()
{
    console.log("ready!");
});
```

alebo

```
$(function() {…});
```

CDE Services

Attributes

- Get and set usually use the same function with different parameters
- Example
  - Get attribute:

```
var a =
$("h2").attr("name");
```

  - Set attribute:

```
$("h2").attr("name","value")
;
```

CDE Services

Class specific attributes

- Class specific attributes

```
$("selector").hasClass("className");
$("selector").removeClass("className");
$("selector").toggleClass("className");
```

- Access HTML content

```
//get data
 $("selector").html();
 //set html
$("selector").html("html
```

CDE Services

Events and animations

- Event binding and unbinding

```
$("selector").on("click",
    function(){
    //do something
 });
```

or (some events have shortcuts names)

```
$("selector").click(function…);
```

- Simple animations

```
$("selector").show(speed);
$("selector").hide(speed);
```

CDE Services

For each

Every next one element will have bigger border

```
var border = 1;
$("selector").each(function(
){
    $(this)

.css("border",border+"px");
    border += 1;
});
```

CDE Services

Too big API to show all ☺

- jQuery API is quite big
- It can change a little bit with new versions
- Always try to be up to date
- Take a look for what you need
- http://api.jquery.com/

CDE Services

jQuery UI

- Standalone project with UI components
- Use for rapid prototyping of nice simple UI
- http://jqueryui.com/

CDE Services

Consulting
Development
Education

AJAX

CDE Services

Basic (old) web applications

- **"Press, wait, display"**
  - Every communication with server means reloading of full page
  - **Synchronous communication – request + response**
  - User is forced to wait for response and reload of page
- **Page oriented**
  - Navigation between pages is mostly done by server
- **Loss of context on reload**
  - Page scroll, filled input fields
- **No immediate feedback** for user activity
  - Need to wait for page refresh
- **Low number of control elements**

CDE Services

Rich Internet Applications

- RIA – Rich Internet Application
- Web applications with properties and functionality of desktop applications
- RIA technologies
  - Java Applet
  - Adobe Flex
  - Java WebStart
  - DHTML
  - Silverlight (Windows only)
  - JavaFX (Java Platform)
  - **AJAX**

CDE Services

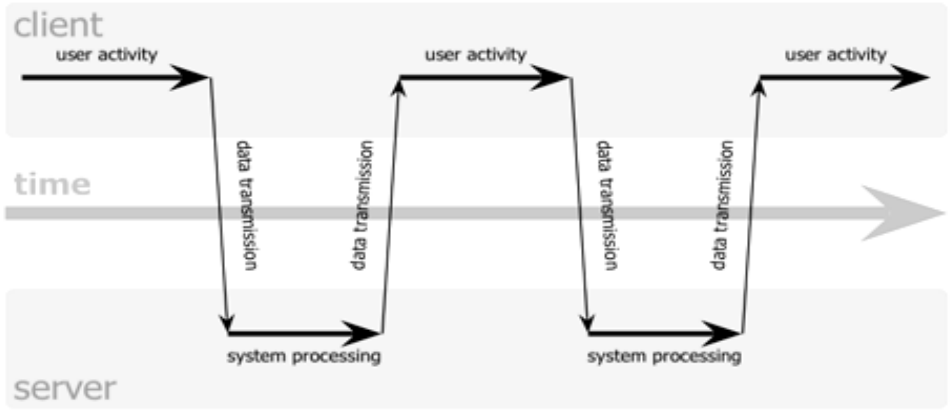
AJAX

- AJAX – Asynchronous JavaScript and XML
- Examples of usage
  - Validation of input on server during filling of form
  - Autocomplete of forms
  - Advanced controls
  - **Dynamic change of part of page** without any page refresh and redrawing of all page content

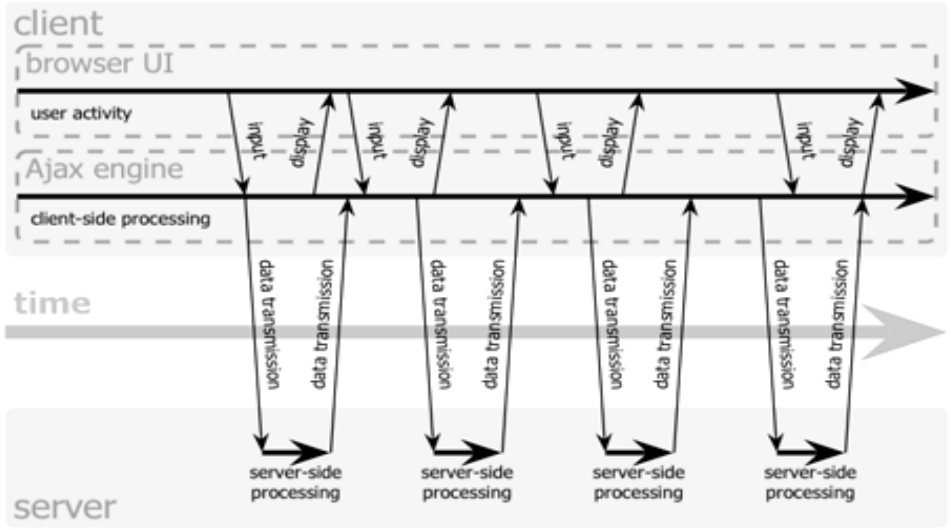
CDE Services

Classic vs. AJAX communication model

classic web application model (synchronous)

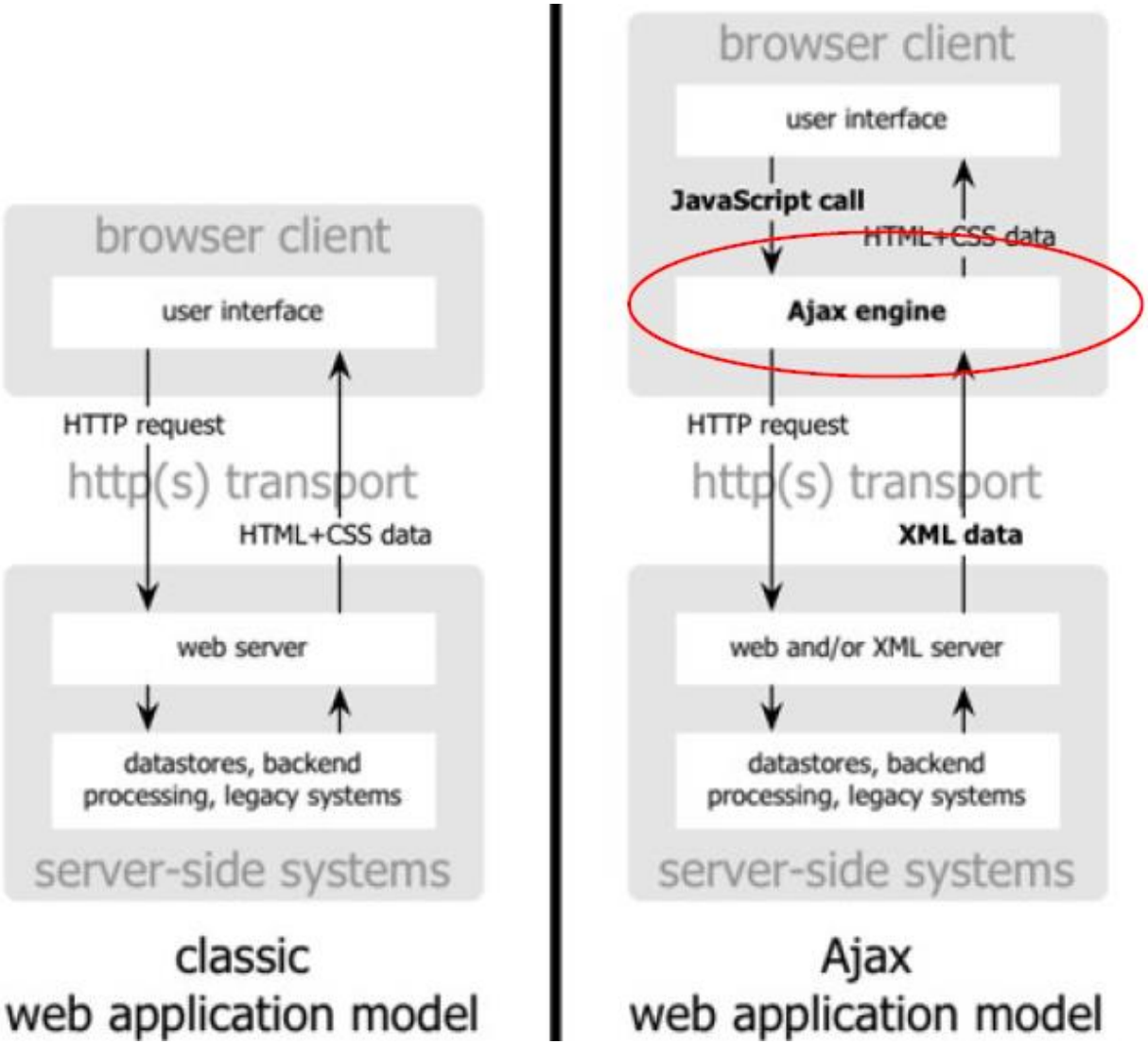


Ajax web application model (asynchronous)



CDE Services

Classic vs. AJAX communication model



CDE Services

Technologies used in AJAX

- HTML (XHTML)
- CSS
- XML
- Document Object Model (DOM)
- JavaScript
- JavaScript Object Notation (JSON)
- HTTP
- **XMLHttpRequest** (Javascript object type)

CDE Services

XMLHttpRequest

- JavaScript objekt
- Supported by actual web browsers
- Serves for communication with server though standard HTTP protocol
  - GET/POST methods
- Communication with server can be executed in background – asynchronously
- Does not interrupt user actions
- MIME type of response from server
  - text/xml
  - text/plain
  - text/json
  - text/javascript

CDE Services

XMLHttpRequest methods

- `open("HTTP metóda", "URL", asyn)`
  - specification of HTTP method, target URL, mode of transfer – asynchronously (true/false)
- `send(content)`
  - Sends request to server
- `abort()`
  - Cancel request
- `getAllResponseHeaders()`
  - Getting reponse headers
- `getResponseHeader("label")`
  - Getting specific response header
- `setRequestHeader("label", "value")`
  - Setting request header value

CDE Services

XMLHttpRequest properties

- `onreadystatechange`
  - Javascript function, which will be executed on each change of state
- `readyState`
  - contains actual state of request
    - 0 = uninitialized
    - 1 = loading
    - 2 = loaded
    - 3 = interactive (some data have been returned)
    - 4 = complete
- `status`
  - HTTP status code returned by server (200 = OK)
- `responseText`
  - String containing response from server
- `responseXML`
  - XML document representing response data from server (only if data was sent in XML type)
- `statusText`
  - Textual description of server status code

CDE Services

Example – request creation

```
if (window.XMLHttpRequest) {
  request = new XMLHttpRequest();
} else if (window.ActiveXObject) {
  request = new
      ActiveXObject("Microsoft.XMLHTTP");
}

var url = "./person";
request.onreadystatechange = processRequest;
request.open("GET", url, true);
request.send(null);
```

CDE Services

Example – response processing

```
function processRequest() {
   if (request.readyState == 4) {
      try{
         if (request.status == 200) {
           var message = request.responseText;
           //Handle message
         }
      } catch (e) {
         //Error
      }
   }
}
```

CDE Services

jQuery AJAX

```
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( data ) {
     $( "#weather-temp" ).html( "<strong>" +
   data +
                                "</strong>
   degrees" );
  }
});
```

CDE Services

Consulting
Development
Education

Give me MoOoRE brains

☺

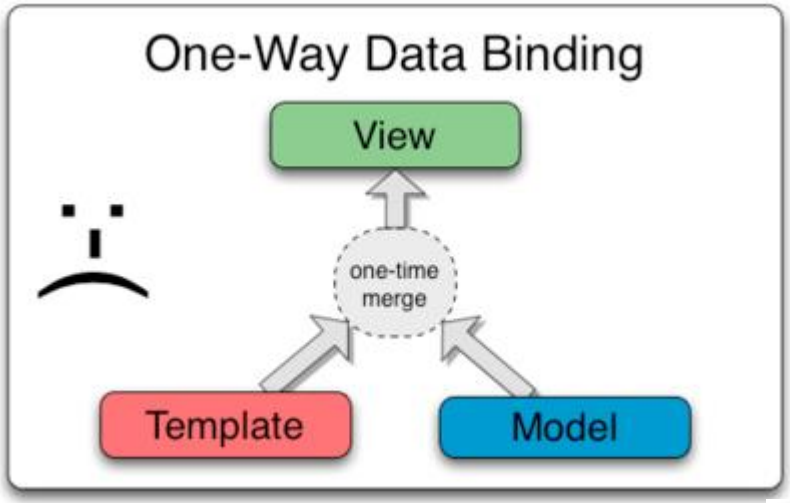
CDE Services

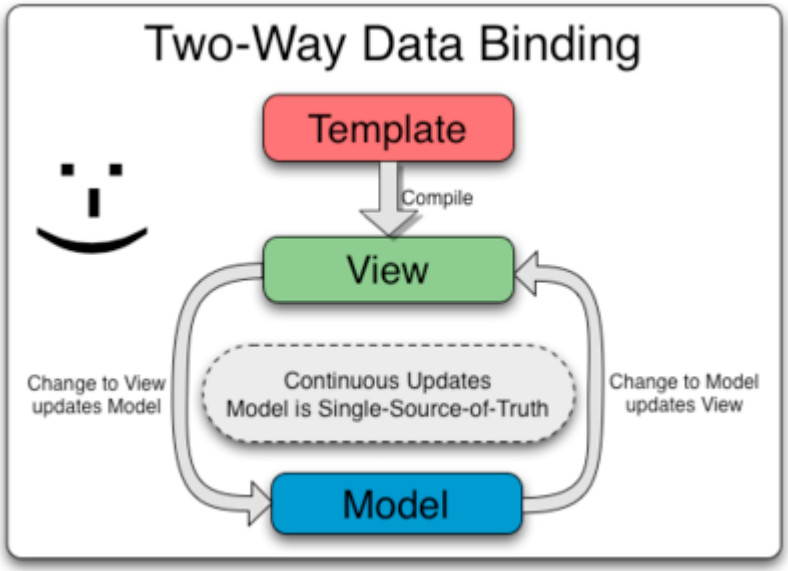
ANGULARJS

by Google

- Frontend web framework
- Templating system with working bidirectional binding
- Uses jQuery subset - jQLite
- Don’t reinvent the wheel, use existing directives of Angular for DOM manipulation
- Take a look: http://angularjs.org/

AngularJS Data Binding





- http://docs.angularjs.org/guide/

CDE Services

You can also try

BACKBONE.JS