JavaScript, Emulating Java, the native APIs

WS Slides #2

Scripting Languages

"A **scripting language** is a programming language that is used to manipulate, customise, and automate the facilities of an existing system ...
... the existing system is said to provide a host environment of objects and facilities, which completes the capabilities of the scripting language."

Existing system for us: The browser

JavaScript

JavaScript is a client (browser) side*) scripting language

- Allow authors to create highly interactive web pages (by manipulating the DOM and more)
- Implementation (dialect) of the ECMAScript 5.1 specification

*) JS has started to go server side, …

Executing JavaScript

Script file(s) downloaded from server (cached) at page request, typically

```
<!-- In HTML Page (if many scripts, order matters !!) -->
 <head>
   <!-- Will download script -->
   <script type="text/javascript" src="js/myjavascript.js" />
 </head>
 <body>
       ...
 </body>
```

- All script elements are executed by the browser in found order (as the document is loaded). Order matters!!!
- Normally only "event handling setup"-code executed during page load (after DOM construction)
- Possibilities for "code on demand" (security)
- Possible to disallow scripting

JavaScript vs Java

Looks close to Java but ...

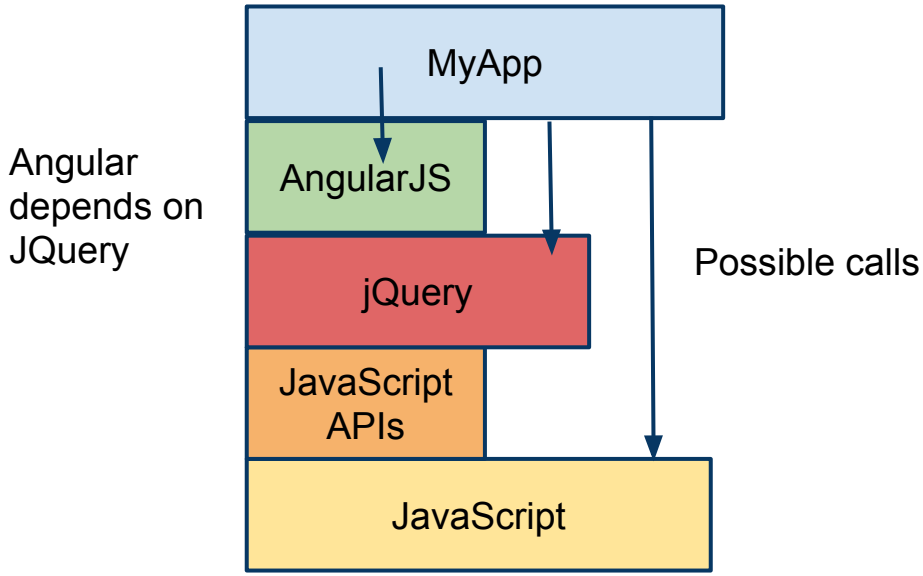
- Java and Javascript are similar like Car and Carpet are similar
- Still … syntax (C-languages) and basic statements if, for, … same or similar

Language somewhat related to functional languages

- Heavy use of (anonymous) functions

JavaScript Strategy in Course

We'll try to avoid any low level JS coding
- Specifically the native APIs
- We use a high level framework: AngularJS (by Google) and possible jQuery (by J. Resig) if needed.
- Still need some understanding (background information)



JS The Good Parts

JS The Awful Parts

Non exhaustive list ...

- Non-static typed, all type errors runtime (typical: Nothing happens..!) Spelling!
- Global variables, introducing (using) an undeclared variable makes it global (i.e. omitting the "var" keyword)
- All compilation units loaded into a common global namespace, name clashes
- No modularization mechanisms (packages..)
- Program semantics very strange: semi-colon insertions, variable reordering, equals, ...?!
- Many gotchas!! NetBeans will warn for some ...

There's a "strict" mode trying to reduce design flaws

JS Language Overview

- Object-based scripting language (no classes)
- **Object** is a dynamic collection of properties each with zero or more attributes
- **Properties** are containers that hold primitive values, objects or functions
- A primitive value is a member of one of the following built-in types: **Undefined, Null, Boolean, Number** (no int!!)**, and String**
- An object is a member of the remaining built-in type **Object**
- A **function** is a callable object
- A function that is associated with an object via a property is a **method** (like Java)

JS Built-in Objects

"ECMAScript defines a collection of built-in objects that round out the definition of ECMAScript entities. These built-in objects include the **global** object, the **Object** object, the **Function** object, the **Array** object, the **String** object, the **Boolean** object, the **Number** object, the **Math** object, the **Date** object, the **RegExp** object, the **JSON** object, and the Error objects **Error, EvalError**, **RangeError, ReferenceError, SyntaxError, TypeError** and **URIError**."
// ECMA Specification

- Note: Objects not classes, there are no classes in JS

JS Host Environment

"A web browser provides an ECMAScript host environment for client-side computation including, for instance, objects that represent windows, menus, pop-ups, dialog boxes, text areas, anchors, frames, history, cookies, and input/output [host objects]. Further, the host environment provides a means to attach scripting code to events such as change of focus, page and image loading, unloading, error and abort, selection, form submission, and mouse actions. Scripting code appears within the HTML and the displayed page is a combination of user interface elements and fixed and computed text and images. The scripting code is reactive to user interaction and there is no need for a main program."
// ECMA Specification

In browser the window object = the global object

Objects

Object are mutable maps like; Map<Property, Value> in Java

- Object structure possible change during lifetime
- No constructor
- Everything public
- Can contain sub-objects
- Objects manipulated via references like Java
- References have no types (untyped). Objects have
- Object literal (create object in line): { … } ← an objec
- Internal properties, pops up (not possible to manipulate with language, but need understanding)

Functions

Functions are (close to) objects with the additional capability of being callable

- Functions are first class members; Functions as parameters/result, references to, callbacks
- Anonymous functions common
- Inner functions (function inside function)
- Functions may have properties (data..!)
- Functions as information hiding. Objects inside function not accessible from outside. JS has function scope, not block scope (can't access inner parts of function)
- Has hidden property "arguments". Gives access to all supplied arguments and more...
- If return ... in function, function will return (possible with value).
- If no explicit return, returns "undefined"

Function cont.

Call conventions same as Java (by value)

- Some primitives are like objects (boxed)!
- Passing in a function is different (although it's an object)?!
- Different invocation patterns (!), more to come...

Methods

A function as a property of an object is a method

- Methods, will belong to object, not shared between objects

```
var o = { // Start object literal
          ...
          doIt: function() {  // Method (reference to anon. function)
              return … ;
          }

}; // End object literal

// Call
o.doIt();
```

Closures

"A closure is formed when one of those inner functions is made accessible outside of the function in which it was contained, so that it may be executed after the outer function has returned. At which point it still has access to the local variables, parameters and inner function declarations of its outer function. Those local variables, parameter and function declarations (initially) have the values that they had when the outer function returned and may be interacted with by the inner function."

More on Javascript Closures

Funny comment on web
"Closures are not hard to understand once the core concept is grokked. However, they are impossible to understand by reading any academic papers or academically oriented information about them!"

Prototype

All objects have an implicit reference to a "parent object"

- This is the **object prototype**
- Prototype object used to share properties between objects (remember; methods belong to individual object)
- Displayed in Chrome debugger as **__proto__**

The prototype chain

- All objects have a link to it's "parent", a chain of references. If asking for some property not found in actual object, the prototype chain is searched
- Final object in chain is Object object
- Possible to alter new children by assigning new values to the prototype for some parent (old children not affected)

Constructor Function

Function used to create and initialize new objects in conjunction with operator **new**

- Can't distinguish from normal function, except using leading uppercase for name (an idiom, not enforced)
- Imagine a freestanding constructor (outside object)

A constructor function has a not hidden reference to an automatically created unnamed prototype object

- This is the **prototype property**
- Accessed with NameOfContructorFunction.prototype
- The prototype objects has a reference back to constructor

Object Creation and Prototype

Using constructor function and new

"When a constructor creates an object, that object implicitly references the constructor's prototype property for the purpose of resolving property references."

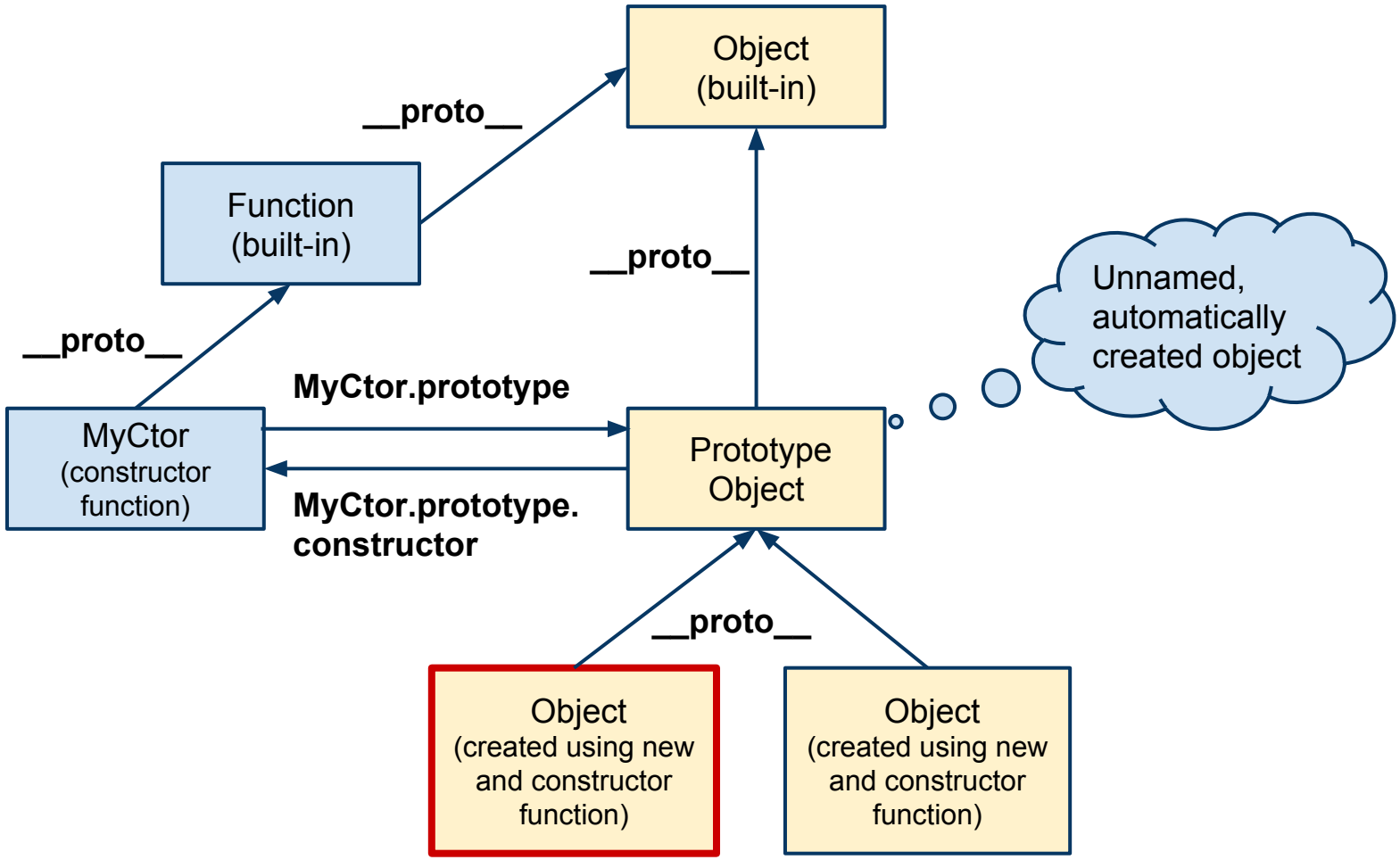
// ECMA Spec

An alternative to create objects (possible better) is Object.create(...) (not used by us)

Object Creation with Constructor

Executing

Object o = new MyCtor();



The “this” keyword

JavaScript “this” is different from Java

- In global scope **this** is the window object (the global object)
- The value of **this** in a function context is provided by the caller and determined by the current form of a call expression (how the function call is written syntactically!!!)
- this not statically bound (may vary)
- Complicated: We simplify by identifying fairly correct invocation patterns ...

Invocation patterns for “this”

Global function invocation: In function **this** is the global object

Method invocation: In method **this** is the object the method belongs to

Constructor function invocation (in conjunction with new) : In function **this** is the newly created object

Preserving “this”

Inner functions doesn’t share **this** with outer

To use **this** in inner function, have to store outer before executing inner

```
function Outer() {
    …
    var me = this;  // Store actual “this” for use in inner function
    function Inner (){
        // “this” changed, accessed “old this” value with me
    }
}
```

Sometimes handled automatically by some higher level libraries, jQuery, …

Miscellaneous

Strings: " " or ' '

Comments as in Java /* */ and //

Declaring variables using **var** keyword (optional but use!)

```
// Forget var ends up in Global scope (even in functions)
var a = { … }// a reference to object literal
```

Ending ';' optional but use!

Watch out for conversions and comparison, always use === for comparison

Exceptionhandling …

Emulating Java

We use the JS "pseudo-classical" style + module pattern to emulate classes

- Emulated class = constructor function (for object data) + constructor function prototype holding shared methods

- Put "class" in matching file like Java (class name = file name)
- Mostly we use Angular but also possible need this

Pseudo-Classical JavaScript

```
// Emulate a class (this is in file person.js)
var Person = function( name ){  // Constructor function
    // Set attributes here, "this" is the actual object!
    this.name = name;
};

Person.prototype = (function(){ // Shared by all objects
    // Public API here
    return { // Anonymous object
        setName : function( name ){
            this.name = name;     // "this" is the actual object
        },

        getName : function (){
            return this.name;
        }
    }
})();  // Call function immediately (prototype will reference object)
```

Pseudo-Classical JavaScript cont.

```
// MUST use new else disaster (this bound to global object)
var p = new Person( "Sara" );
```

Now...

… p is a reference to the returned anonymous object with methods setName and getName (previous slide)

```
// Call
var name = p.getName();
```

getName not found in p, so search prototype, method found! … call it, "this" supplied by caller (the p object), so "this" is really the actual object

The Native APIs

Very many (in various conditions)

Most important

- The DOM API, to dynamically change the DOM and CSS properties (change user view)
- The Event API, for event driven applications
- The XMLHttpRequest, for asynchronous HTTP calls

Note: We don't use any of. We use higher level APIs.

- This is just background information, more later …

Native DOM API

JavaScript to interact with the DOM tree
- Find nodes
- Add, edit, remove nodes
- …
- The document property of a window points to the DOM for the document loaded in that window (the tree root)

```
// Example native JS DOM API call. Avoid!
var e = document.getElementById( ...);

// Same as
var e = window.document.getElementById( ...);
```

When talking HTML we say element when DOM tree we say node

Event Driven Web Application

"DOM … events allow event-driven programming languages like JavaScript, JScript, ECMAScript, VBScript and Java to register various event handlers/listeners on the element nodes inside a DOM tree, e.g. HTML, XHTML, XUL and SVG documents. Historically, like DOM, the event models used by various web browsers had some significant differences. This caused compatibility problems. To combat this, the event model was standardized by the W3C in DOM Level 2." //Wikipedia

HTML Events

Huge collection

- Hierarchy of APIs for event handling: DOM Level 0-N.

- Event handling, connecting listeners to elements, handling events will be done from inside AngularJS, more to come…

```
// Typical event handling in AngularJS
// Register listener (callback function) for click event
$scope.listeners.doClick = function(evt) {
          // Event handling code here
       };
```

More to come...

Event Capturing and Bubbling

Some knowledge of HTML event capturing and bubbling can possible be useful

- Possible to use same listener for all children of an element

JavaScript Object Notation, JSON

Text-based open standard designed for human-readable data interchange

- Representations of JavaScript objects
- Language independent (but derived from JavaScript)
- Lightweight (compare XML)
- Normally automatic conversion from/to JSON / JavaScript object in browser at request and response
- If not converted, use the below in client

```
//Native
var obj = JSON.parse('{"name":"John"}');
// Have JSON (text) get an object (using JQuery API)
var obj = $.parseJSON('{"name":"John"}');
```

- Often used as parameters, i.e. send an inline object

Example JSON

```
{
    "firstName": "John",                        // Note " not '
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"    // integer 10021 also possible!
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

XMLHttpRequest

API for asynchronous HTTP calls
- Aka "XHR" (possible to inspect in debuggers)

XMLHttpRequest + DOM API make it possible to implement AJAX (Asynchronous JavaScript and XML)

"With Ajax, Web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page"
// Wikipedia

Possible: **Single page applications**. Update page by asynchronously retrieving JSON

Asynchronous Programming

JavaScript is single threaded, if long time call, GUI will freeze

- Solution: Asynchronous calls. Will return immediately
- When operation finished where to return data (program has continued)? Solution: Supply callback function
- If long call chains leads to very nested (messy) code
- Solution: Return object representing the future result (a promise). Use object "inline", gives flat code. Handled by Angular more to come…

Bookmarkability

Important for user to be able to bookmark pages
- Example: Paste URL into mail

If using AJAX the URL possible want change (single page application)
- Problems with bookmarks, forward/backward, favorites
- Possible solution in HTML5, use URL hash mark (#) and History API or hashchange event
- Angular will use hash mark

JavaScript Same Origin Policy

Security Issue

- A script can read only the properties of windows and documents that have the same origin (i.e., that were loaded from the same host, through the same port, and by the same protocol) as the script itself.
- NOTE: Local files uses file:// not http://(possible to config. Chrome to accept)

- Untrustworthy script in one window could use DOM methods to read the contents of documents in other browser windows, which might contain private information.
- Poses particular problems for large websites with many servers
  - JSONP, trick to circumvent restrictions
  - CORS, mechanism to allow resources from different domains

Developing with JavaScript

Must have a JS debugger (most errors runtime)
- Chrome (Tools > Developer tools)
- In Firefox, addon Firebug (Tools menu). Installed in school

Possible to step code, inspect values etc.
- Debugging dynamically downloaded JS in Chrome debugger needs tweak

```
//Last in file products.js
//@ products.js
```