Javascript
web development... and more

Jacobo Aragunde Pérez

jaragunde@igalia.com

blogs.igalia.com/jaragunde

igalia

Introduction

what's that?

- a scripting language
- born as a part of a web browser
- increasing popularity in the web world
  - from form validation to full control of pages
- currently used as a general-purpose language too

features

- imperative, structured, functional features
- dynamic typing
  - types are associated with values, not variables
- object based through prototypes
- interpreted

syntax

- similar to C or Java
  - functions
  - variables
  - objects
  - loops

```
function showMessage(message) {
    document.write(message);
}

var message = "hello world";
showMessage(message);

var messages = new Array(2);
messages[0] = "hello";
messages[1] = "world";

for(var i=0; i<messages.length; i++)
{
    showMessage(messages[i]);
}
```

some history

- First developed by Netscape for their Navigator web browser (1995)
- Quickly Microsoft added compatibility in Internet Explorer 3 (JScript, 1996)
- Standarized by Ecma International (ECMA-262, ECMAScript, 1997)
- There are out-of-standard features in JavaScript and JScript
- Relation with Java: pure marketing ;)

Hands-on

running JavaScript

- the easy way: including inside a HTML page

```
<head>
  <script type="application/javascript">
    //JavaScript code
    //...
  </script>
</head>
<body>
  <!-- HTML body -->
  <script type="application/javascript">
    //another JavaScript block
    //...
  </script>
  <noscript>
    <p>You have JavaScript disabled...</p>
  </noscript>
</body>
```

example: basic-example-1

running JavaScript

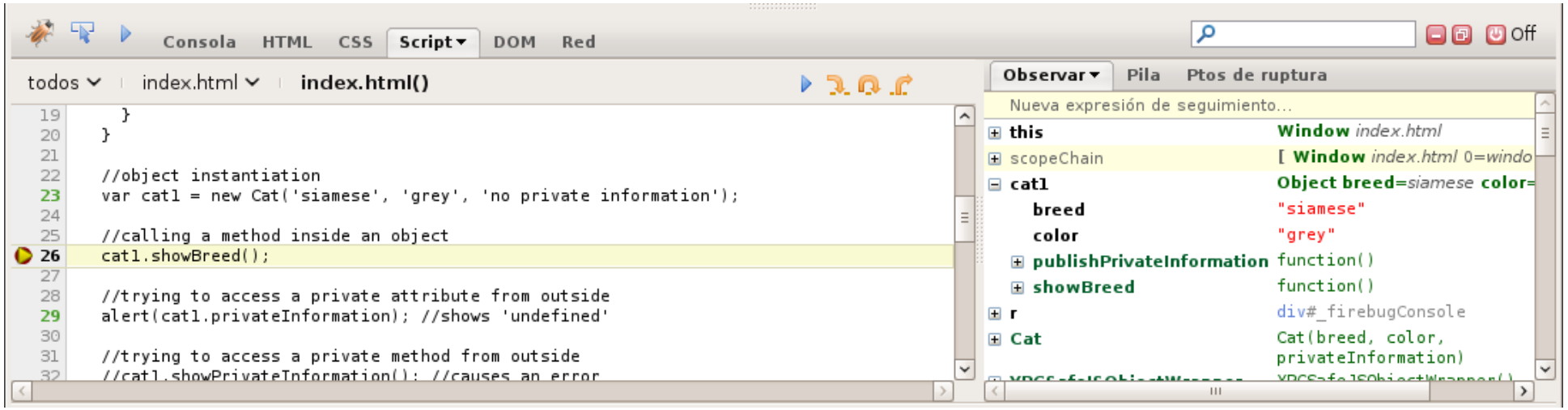
- including external code files (better practice)

```
<head>
  <script type="application/javascript" src="code.js"/>
</head>
<body>
  <!-- HTML body -->
  <noscript>
    <p>You have JavaScript disabled...</p>
  </noscript>
</body>
```

example: basic-example-2

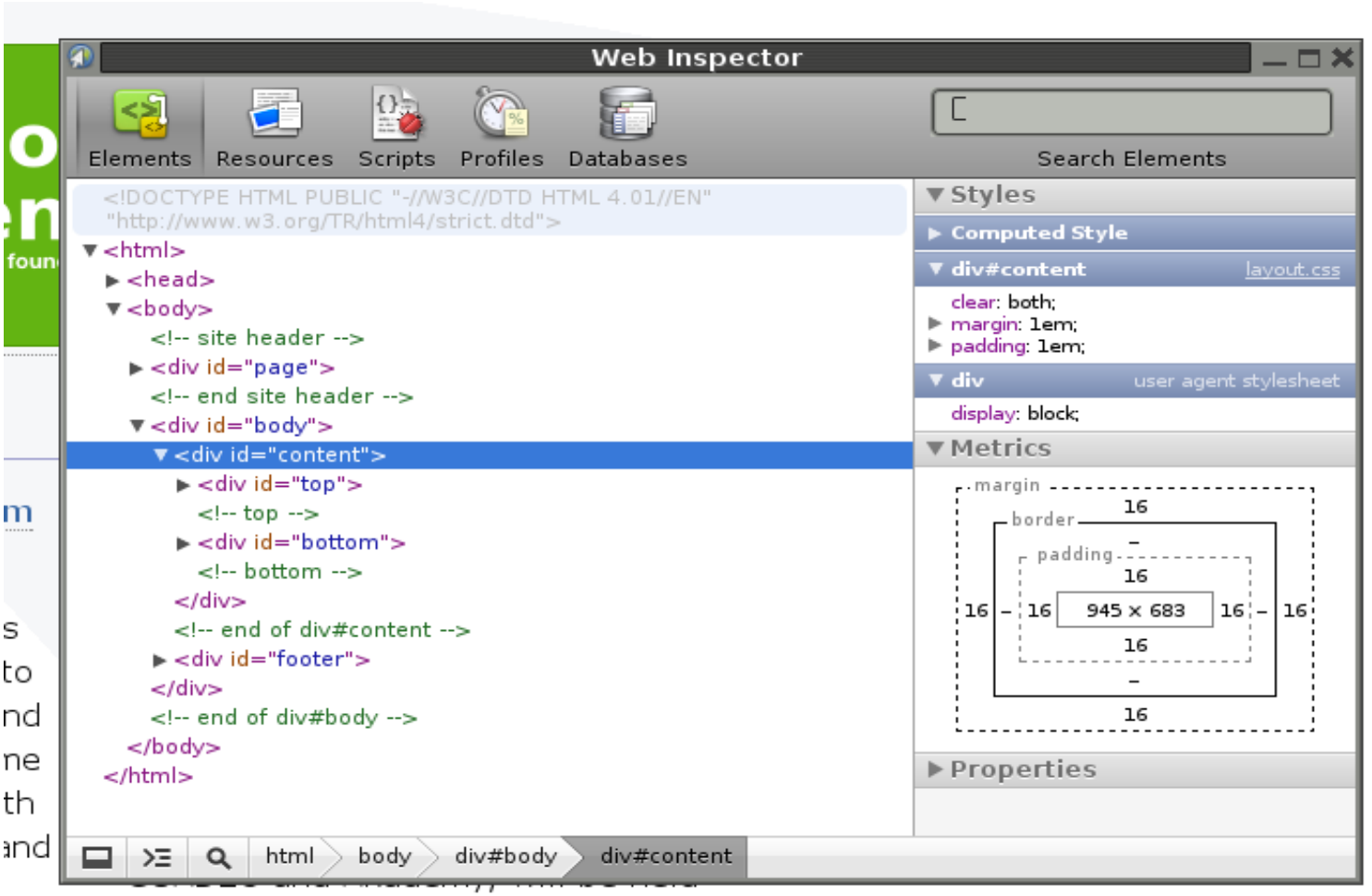
debugging JavaScript

- Firefox, Chrome: Firebug extension
  - http://getfirebug.com/
  - http://getfirebug.com/releases/lite/chrome/



debugging JavaScript

- Safari, Chrome, Epiphany: built-in web inspector



manipulating DOM

- DOM (Document Object Model), objects representing the HTML shown on screen
  - `window`, represents the browser window
  - `document`, represents the HTML document
  - `history`, represent the browsing history
  - `form`, `link`, `image`, etc. for the corresponding HTML tags.

`window` object

- manipulation of browser windows
  - `open()`
  - `close()`
  - `alert()`
  - `confirm()`
  - `prompt()`

example: dom-window

`document` object

- useful to navigate the DOM
  - `getElementById()`
  - `getElementsByName()`
  - `getElementsByTagName()`
- can alter the DOM
  - `write()`
  - `writeln()`

example: dom-document

events

- used as entry points to the JavaScript code
- some examples
  - `onload`, `onunload`
    - `window.onload` is usually the “main” entry point
  - `onfocus`, `onblur`, `onchange`
  - `onsubmit`
  - `onmouseover`, `onmouseout`

events

- how to use them
  - `<form onsubmit="onSubmitCode();">`
  - ```
    myForm.onsubmit = function () {
        onSubmitCode();
    };
    ```

example: validation
    ```

general members of DOM objects

- properties
  - `className`, `nodeValue`, `length`, `width`...
- methods
  - `appendChild()`, `removeChild()`, `blur()`, `focus()`...
- events
  - `onblur`, `onclick`, `onfocus`...
- reference:
  - http://www.w3schools.com/jsref/dom_obj_all.asp

good practices

- indent with spaces (4 spaces recommended)
- avoid lines wider than 80 characters
- include the code in separate `.js` files
- declare variables before using them
- start constructor function names with capital
- start other variable or function names with lower case
- always end statements with ;

mythbusters

- the `language` property
  - deprecated
- the `<!-- //-->` trick
  - discouraged, only useful with **really old** browsers
- values for the `type` property
  - `text/javascript` is obsolete (RFC4329)
  - replaced with `application/javascript`
  - anyway, it can be avoided when using `src=""`
    - the server sends the MIME type

Object-oriented? Really?

classes vs prototypes

- classes define the objects structure and behavior
  - objects are instantiated from classes
- prototype-based languages clone objects that serve as prototypes to reuse their structure and behavior
  - objects are instantiated from other objects

how does it work?

- functions double as object constructors along with their typical role
- difference between functions/constructors: `new` keyword
  - prefixing a function call with `new` creates a new object and calls that function with its local `this` keyword bound to that object for that invocation

how does it work?

```
function Cat(breed, color) {
    this.breed = breed;
    this.color = color;
    this.showBreed = function() {
      alert(this.breed);
    }
}

//object instantiation
var cat1 = new Cat('siamese', 'grey');

//calling a method inside an object
cat1.showBreed();

//alternate syntax to define a constructor
var Dog = function (breed, color) {
    this.breed = breed;
    this.color = color;
}
```

example: prototyping

private members

- defined using `var` keyword inside the constructor

```
function Cat(breed, color, privateInformation) {
    //...

    var privateInformation = privateInformation;

    var showPrivateInformation = function() {
        alert(privateInformation);
    }

    this.publishPrivateInformation = function() {
        showPrivateInformation();
    }
}
```

example: prototyping

private members

```
//object instantiation
var cat1 = new Cat('siamese', 'grey', 'no private information');

//trying to access a private attribute from outside
alert(cat1.privateInformation); //shows 'undefined'

//trying to access a private method from outside
//cat1.showPrivateInformation(); //causes an error

//privileged method accessing private data
cat1.publishPrivateInformation();
```

extending a prototype

- Using the `prototype` member
- New public members (both methods and attributes) can be added
- it's done in run time

extending a prototype

```
function Cat(breed, color) {
  this.breed = breed;
  this.color = color;
  this.showBreed = function() {
    alert(this.breed);
  }
}

//extending the prototype in run-time with a new method
Cat.prototype.showColor = function (){
  alert(this.color);
}

//calling the new method normally
cat1.showColor();
```

extending a prototype

```
//extending the prototype in run-time with a new attribute
Cat.prototype.age = null;

//setting a value for that new attribute
cat1.age = 2;

//adding a new attribute with a default value
Cat.prototype.legs = 4;
alert(cat1.legs); //shows '4'

//of course, the value can be modified later
cat1.legs = 3;
alert(cat1.legs); //shows '3'
```

visibility of members

- private
  - only accessible inside the constructor
- privileged (methods)
  - accessible from outside
  - can access private members
- public
  - accessible from outside
  - can't access private members

visibility of members

```
function Constructor() {
    var privateAttribute = "private attribute";
    this.publicAttribute = "public attribute";

    var privateFunction = function () {
        alert(privateAttribute); //can access a private member
    }

    this.privilegedFunction = function () {
        privateFunction(); //can access a private member
    }
}

Constructor.prototype.publicFunction = function () {
    //these lines would produce an error because
    //we can't access private members from here
    //alert(privateAttribute);
    //privateFunction();
    alert(this.publicAttribute);  //can access public members
    this.privilegedFunction(); //can access privileged members
}
```

example: visibility

classical inheritance

- it's done extending the prototype, too

```
//definition of the parent "class"
function Animal(animalClass) {
    this.animalClass = animalClass;

    this.getAnimalClass = function () {
        return this.animalClass;
    }
}

//definition of the child
function Cat(breed, color) {
    this.breed = breed;
    this.color = color;
}

//inheritance
Cat.prototype = new Animal("mammal");
```

example: classical-inheritance

classical inheritance

```
//create an instance of a Cat
var cat1 = new Cat("siamese", "grey");

//run an inherited method
alert(cat1.getAnimalClass()); //"mammal"

//overwrite an inherited property
cat1.animalClass = "unknown";
alert(cat1.getAnimalClass()); //"unknown"

//last test: instanceof operator
if (cat1 instanceof Animal) {
    alert("cat1 is an instance of Animal");
}
if (cat1 instanceof Cat) {
    alert("cat1 is an instance of Cat");
}
```

example: classical-inheritance

JSON

what's that?

- JSON stands for JavaScript Object Notation
- lightweight data-interchange format
- subset of JavaScript language
- after having become popular, was standarized in RFC 4627

what's that?

```
var object = {
    firstMember: "I am a string",
    secondMember: false,
    thirdMember: [
        {
            arrayElementFirstMember: 10,
            arrayElementSecondMember: 20
        },
        {
            arrayElementFirstMember: 30,
            arrayElementSecondMember: 40
        }
    ],
    forthMember: 50
};

alert(object.secondMember); //false
alert(object.thirdMember[0].arrayElementFirstMember); //10
alert(object.thirdMember[1].arrayElementFirstMember); //30
```

example: json

what's that?

- The notation can be used to define any object
- Watch out! This example is **not** standard JSON
  - JSON is a **subset** of JavaScript

```
var myDog = {
    breed: "bulldog",
    showBreed: function () {
        window.alert(this.breed);
    }
};

myDog.showBreed(); //bulldog

myDog.breed = "pitbull";

myDog.showBreed(); //pitbull
```

Helper libraries

why?

- hide differences among browsers to the programmer
- simplify navigation through the DOM
- ease the use of asynchronous requests
- add new features
- and, in general, simplify the most common tasks

jQuery

- http://jquery.com/
- focused on simplifying JavaScript code
- handy features for
  - document traversing
  - event handling
  - manipulating CSS and animating
  - Ajax interactions
- provides a **basic** set of widgets

jQuery

- how to use it
  - `<script src="jquery.js"></script>`
- how it works
  - heavy overload of the operation `$()`
    - returns 'glorified' DOM elements

example: jquery-basic-example

document traversing

- accessing elements through id
  - `$('#myDivId')`
- accessing elements through class
  - `$('.myCssClass')`
- accessing elements through tag
  - `$('input')`

document traversing

- selectors can be combined
  - `$('input.myCssClass')`
- XPath expressions are allowed
  - `$("div#buttons > input")`
- `filter()` and `not()` help selecting only certain elements from a group of similar ones
  - `$("li").not(":has(ul)")`

example: jquery-dom-and-events

event handling

- syntax
  - `$('#myButton').click(onClickFunction);`
  - `$(document).ready(onReadyFunction);`
    - `ready` event replaces traditional `window.onload`
- can be applied to a set of objects transparently
  - `$('.button').click(onClickFunction);`
- every `onxxx` event (`onclick`, `onchange`, etc.) has a jQuery equivalent, plus some more

example: jquery-dom-and-events

manipulating CSS

- add classes to a (set of) DOM element(s)
  - `$("#myElement").addClass("myClass");`
- retrieve information about CSS
  - `$("#myElement").css( "color"); //returns color`
- set CSS properties directly
  - `$("#myElement").css("color","red");`

example: jquery-css

animating

- show and hide
  - `$("#hideMe").hide('slow');`
- custom animation
  -
```
$("#animateMe").animate(
    { width:"50%", fontSize:"12px" },
     'slow');
```
- animations can be combined
  -
```
$('#button').show('slow').
      animate({ width:"50%" }, 'slow').
       hide('slow');
```

example: jquery-animating

Ajax interactions

- `$.ajax()`, or higher-level alternatives like `$.get()`, `$.post()` or `$.load()`

```
$.ajax({
    url: "jquery-ajax-test.html",
    success: function (data) {
        $("#result").append(data);
    }
});

$.get("jquery-ajax-test.html",
    function (data) {
        $("#result").append(data);
    }
);

$("#result").load("jquery-ajax-test.html");
```

example: jquery-ajax

ExtJS

- http://www.sencha.com/products/js/
- focused on providing heavier components with complex functionality
- class-styled API
- provides components for
  - windows
  - grids
  - charts
  - place-holders
  - and lots more!

ExtJS

- how to use it?

```
<!-- ** CSS ** -->
<link rel="stylesheet" type="text/css"
    href="ext/resources/css/ext-all.css" />

<!-- ** Javascript ** -->
<!-- ExtJS library: base/adapter -->
<script src="ext/adapter/ext/ext-base.js"></script>
<!-- ExtJS library: all widgets -->
<script src="ext/ext-all-debug.js"></script>
<script>
    // Path to the blank image
    Ext.BLANK_IMAGE_URL =
        'ext-3.2.1/resources/images/default/s.gif';
</script>
```

example: ext-basic-example

document traversing

- syntax
  - `Ext.get('myId'); //selection by id`
  - `Ext.select('.myClass'); //multiple selection using CSS selectors`
- similar to jQuery, but...
  - these functions return specific objects from ExtJS

example: ext-dom-and-events

document traversing

- however, building the page directly in the JS source is a common practice
  - HTML stores the data
  - JS 'shapes' them

example: http://www.sencha.com/deploy/dev/examples/window/hello.html

event handling

- syntax
  - `element.on('click',onClickFunction);`
  - `Ext.onReady(onReadyFunction);`
    `//special case`
- every ExtJS element / widget has its own set of events
  - check them in the API

example: ext-dom-and-events

managing ExtJS components

- constructor receives an object of config options

```
var win = new Ext.Window({
    layout:'fit',
    width:500,
    height:300,
    closeAction:'hide',
    plain: true,
});
```

- check the API for config options, methods and events of the component

example: window component

```
var myWindow = new Ext.Window({
    title:'hello world!',
    layout:'vbox',
    width:500,
    height:300,
    closeAction:'hide',
    plain: true,

    items: [
        ...,
        ...
    ],

    buttons: [{
        text: 'Close',
        handler: function(){
            myWindow.hide();
        }
    }]
});
```

example: ext-window

Ajax interactions

- `Ext.Ajax`, class

```
Ext.Ajax.request({
   url: 'foo.php',
   success: successFunction,
   failure: failureFunction,
   headers: {
       'my-header': 'foo'
   },
   params: { foo: 'bar' }
});
```

- integrated in the widgets

```
Ext.get('messageBox').load({
    url: 'bar.php',
});
```

example: ext-ajax

Ajax interactions

```
new Ext.form.ComboBox({
    store: new Ext.data.Store({
        autoLoad: true,
        autoSave: false,
        baseParams: {
            ...
        },
        proxy: new Ext.data.HttpProxy({
            url: 'getCustomerProjectsService.php',
            method: 'GET'
        }),
        reader: new Ext.data.XmlReader(...),
        listeners: {
            'load': function () {
                ...
            }
        },
    }),
    ...
});
```

References

references

- http://javascript.crockford.com/
  - articles about good practices on JavaScript
- http://www.w3schools.com
  - HTML, CSS and JavaScript reference
- http://www.json.org/
  - JSON specification
- http://en.wikipedia.org
  - of course! ;)

references

- http://jquery.com/
  - reference and tutorials about this framework
- http://www.sencha.com/products/js/
  - reference and tutorials about this framework

Thanks ~~for playing~~ for your attention!