Internet, Intranet and Web — Lecture IV

Dynamic Web: client-side programming and content manipulation

Marco Solieri marco.solieri@lipn.univ-paris13.fr
Info et Réseaux en Apprentissage, Sup Galilée, Université Paris 13

November 19th, 2014

Outline

Contents

1 Client-side programming: ECMAScript

1.1 Introduction

(Java | J | ECMA)Script

- programming languge for browsers: client-side scripting
- multiparadigm: object-oriented, imperative, and functional

Bits of history

1995 Sun and Netscape released JavaScript

1996 Microsoft implemented a dialect, JScript

1997 Ecma standardization as ECMAScript

1999 XMLHTTP support added

2005 AJAX paradigm

2007 JavaScript libraries rich clients frameworks

JavaScript code delivery to the browser

- Web browser should load JS code from (X)HTML documents
- Two kinds of way of doing it.

1. Inclusion into `<script>` element:

   Direct internal inclusion, specifying it is not PC-DATA:

   ```
   <script type="text/javascript">
     <![CDATA[
       ...
     ]]>
   </script>
   ```

   Indirect external reference:

   ```
   <script src="/path/of/the/script.js">
   </script>
   ```

JavaScript code delivery to the browser

2. Association to user actions:

   Link reference of an anchor:

   ```
   <a href="javascript:void(a_function())">
     Link
   </a>
   ```

   Event associated to user events on some HTML element, (e.g. `onclick`, `onfocus`, `onchange`, `onmouseover`, `onmouseout`):

   ```
   <elm onevent="afunction()">
   ```

Browser support

No support at all:

- textual browsers (lynx, w3m),
- very old browsers (Navigator < 2, Explorer < 3),
- robots and crawlers,
- accessibility technologies for browsers (e.g.: screen-readers).

Limited support:

- old browsers, still very popular

How to check for support?

- Browser detection
  - difficult to implement correctly: lot of versions and platforms,
  - ineffective and error-prone.
- Feature detection
  - easy to realize: just ask if the function is available,
  - correct by definition.

1.2 Syntax and semantics

Basics

Code

- Case sensitive, whitespace insensitive
- Statement separation with `;`
- Comments:
  - inline, preceded by `//`
  - block, enclosed between `/*` and `*/`

Variables:

- optional declaration with `var` statement
- assignment operator `=`
- call by value (but objects are references)
- scoping style: classically lexical
  - declared variables are local: accessible in inner blocks
  - undeclared variables are global: accessible everywhere
  - code ordering is irrelevant (declarations are hoisted)

Type system

- Base types

  Scalar undefined, null, boolean, number (floating), string.

  Composite array, object, function.

- Type constructors: functions/objects and prototypes.
- Type introspector: `typeof` operator
- Type system characteristics:
  - implicit: types are unspecified
  - dynamic: automatic casting of values
  - weak: only values are typed, not variables
  - duck: type are inferred from behaviour

Flow control

Same syntax as C, Java, PHP ...

```
if (/* boolean expression */) {
  /* block */
} else {
  /* block */ }
```

```
switch (/* variable */) {
  case /* value */ : /* block */ ;
  // ... other cases
  default: /* block */ }
```

```
while (/* boolean expression */) {
  /* block */ }
```

```
for (/* init */ ; /* test*/ ; /*increment */)
  /* block */ }
```

Exceptions

Native exceptions mechanism

- throw an exception

```
throw(/* errorValue */)
```

- catch some exceptions

```
try {
  // block where exceptions might be thrown
} catch(/* errorValue */) {
  // block to do in the event of an exception
} finally {
  // block to do afterward, either way
}
```

- event handler `onError()` for very old browsers.

Function definition

- Definition by declaration, the classic way

```
function myFunct (arg1, /*...*/ argN) {
  // body
};
```

- Definition by expression

```
var myFunct = function (
  arg1, /*...*/ argN) {
    // body
};
```

- Definition by construction of an Object

```
var myFunct =  new Function (
  "arg1", /*...*/ "argN", "/*body*/");
```

Objects and structures

An object is a structure of named slots, where

- a primitive-type slot is a field, and
- a functional-type slot is a method.

Object definition as structure:

```
var objName = {
    fieldName: value,
    methodName: function(params){
        /*code*/
    },
    //...
};
```

Operations on objects:

```
// access to field
objName.fieldName = foo;
// invocation of method
objName.methodName(args);
// dynamic extension
objName.newSlot = bar;
```

Objects and constructors

Constructor definition:

```
function ObjName(fld) {
    this.fieldName = fld,
    this.methodName = function(args){
        /*code*/
    };
    //...
};
```

Object definitions by construction:

```
objFoo = new ObjName(foo);
```

Introspective operators:

1. `this` point to current object,

Objects and prototypes

Prototype

- object used as base for creation of other objects;
- defined in the `prototype` slot of a constructor;
- overridden by actual slots of objects;
- realizes a dynamic inheritance relationship between objects.

Example 1 (Dynamical prototyping relationship).

```
myFatherObj = { fatherSlot: /* original code */ };
ChildObj.prototype = myFatherObj;
function ChildObj(params) { /* overrides */ };
child = new ChildObj(); // now child has fatherSlot
ChildObj.prototype.fatherSlot = /* another code */;
// now child has another fatherSlot
```

Introspective operators:

2. `instanceof` test for typing along the prototype chain.

Built-in objects

Main operations available:

Array length, concatenation, sorting, iteration ...

Date parsing and printing

Math constants and many operations not in base syntax.

Function length, printing (output evaluable!).

Reg. Expr. search and replace occurrences in strings.

Error throwing exceptions.

1.3 Document Object Model

Browser objects

Model for representation and interaction of HTML, XHTML and XML documents.

- cross-platform
- language-independent
- similar to browser internals
- Navigator
  - Plugin
  - MimeType
- Window
  - Frame
  - document
    - Layer

    - Link
    - Images
    - Anchor
    - Applet
    - Plugin
    - Forms
  - Location
  - History

Window and Navigator

Window

- top-level object
- properties and methods of the main browser window
- position: `moveBy(x,y)`, `moveTo(x,y)` ...
- dimension: `resizeBy(x,y)`, `resizeTo(x,y)` ...
- window managements:`open("URLname","Windowname",["options"])`
- timer: `setTimeout(function(),msecs,["opt"])`

Navigator

- properties of the web client
- name, version, installed plugins, cookie support ...

Window: location and history

Location

- URL of the current document
- changes will make the browser redirect

History

- array of previously accessed URLs
- properties: `length`, `current`, `next`
- methods:`back()`, `forward()`, `go(int)`

Window: Document

Document

- representation of the document content
- properties and method to hierarchically access each element

Example 2 (Document objects).

- `window.document.title`: title of the document
- `window.document.forms[0].checkbox[1]`: the second checkbox of the first form
- `window.document.myobj`: the object called "myobj"

DOM example: passive use

Function performing a form validation, check for a field filling:

```
function Verify() {
    if (document.forms[0].elements[0].value
      == "") {
        alert("You must enter your nickname!");
        document.forms[0].elements[0].focus();
        return false;
    } else
        return true;
}
```

Function call on event of submission:

```
<form action="..."
  onSubmit="return Verify(this.form)">
<p>Nickname:
  <input type="text" name="nick" .../>
</p>
```

DOM example: active use

DOM manipulation:

- retrieving a reference to a particular element:

```
var c = document.getElementById('c12');
```

- changing attributes:

```
c.setAttribute('class','myTest');
c.removeAttribute('align') ;
```

- assembling and adding a brand-new paragraph:

```
var newPar = document.createElement('p');
var newText = document.createTextNode(
    'Magic appearance!');
newPar.appendChild(text);
c.appendChild(newP);
```

2 Four-layers web with XML

Four-layers web architecture with static HTML

- Full separation between application and presentation
- The web application in four layers:
  1. storage: files for binary data, DBMS for structured data
  2. application-logic: programs for content generation
  3. presentation: programs for content presentation

4. browser: HTML rendering

- Introduce an intermediate language:
  - XML definition for your own language
  - content marked up using your XML-based language
  - document manipulated for producing the (X)HTML

XML: eXtensible Markup Language

- The markup metalanguage
  - objective: exchange and interoperability of documents
  - long-term aim: substitution of HTML for the Web
- Advantages (in addition to being in fashion):

  Platform independent open standard, W3C recommendation since Februrary 1998

  Self-descriptive human-readable element names and DTD

  Convertibility easy conversion to web formats

  Flexible and rigorous syntax useful in lot of applications, no syntax exceptions

  Structured tree hierarchy: expressive and navigable

  Ready for i18n only i18n-aware encodings (e.g. UTF-8) no more ASCII

XML document "correctness"

Definition 3. An XML document is *well-formed* if:

- all elements are opened, closed and well-nested,
- there is a root element, containing all the others,
- all empty elements have a special emtpy symbol,
- all attributes are doubled-quoted,
- every entity is defined.

Definition 4 (Validity of an XML document)**.**

- is well formed,
- presents a definition,
- satisfies the definition.

2.1 Definition of XML language

DTD: Document Type definition

- The standard for the SGML family (HTML)
- Language for XML DTD close to the SGML
- Defines which
  - element types,
  - attribute lists,
  - entities, and
  - notations

  are allowed in the structure of the class of XML documents.
- Expressiviness: good for simple to medium complexity
- Use: decreasing

DTD, an example of definition

Example 5 (`people_list` DTD: 'example.dtd').

```
<!ELEMENT people_list (person)*>
<!ELEMENT person (
  name, birthdate?, gender?, socialsecurityno?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecurityno (#PCDATA)>
```

DTD, an example of defined document

Example 6 (`people_list` document).

```
<?xml version="1.0" encoding="UTF-8"
  standalone="no"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Theodor Holm Nelson</name>
    <birthdate>1937-6-17</birthdate>
    <gender>Male</gender>
  </person>
  <person>
    <name>Timothy John Berners-Lee</name>
    <birthdate>1955-6-8</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

XSD: XML Schema Definition

- W3C recommendation (April 2012): version 1.1
- Defines which
  - element types,

  - attribute lists,
  - entities, and
  - notations

  are allowed in the structure of the class of XML documents.

- Expressiveness: good for simple to high complexity In addition to DTD, it offers
  - types of data (at programming-language level too!),
  - groups of elements and attributes (m̃acros)
- Use: increasing

XSD, an example of definition

Example 7 (`SimpleAddress` XSD, an extract).

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema elementFormDefault="qualified"
            xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Address">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Recipient" type="xsd:string" />
        <xsd:element name="House" type="xsd:string" />
        <xsd:element name="Street" type="xsd:string" />
        <xsd:element name="Town" type="xsd:string" />
        <xsd:element name="County" type="xsd:string"
          minOccurs="0" />
        <xsd:element name="PostCode" type="xsd:decimal" />
        <xsd:element name="Country">
          <xsd:simpleType name="EUState">
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="ES" />
              <xsd:enumeration value="FR" />
```

XSD, an example of defined document

Example 8 (`SimpleAddress` document).

```
<?xml version="1.0" encoding="utf-8"?>
<Address
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    "SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

2.2 Manipulation of XML documents

XPATH

A syntax to extress location inside XML documents

- a value (boolean, string, number)
- a node set (element, attribute, comment, processing instructions)

Definition 9 (Location path)**.** A sequence of location steps, starting with and separated by '/'

Definition 10 (Location step)**. axis** the direction relative to the context

test the type and the name of the selected node

(predicates) boolean expression filter

XPATH 2.0 is W3C recommendation (January 2007)

XPATH's axis

- `self`
- `parent`
- `ancestor`
- `child`
- `descendant`
- `preceding-sibling`
- `preceding`
- `following-sibling`
- `following`

Example 11.

```
/child::doc/child::chapter/descendant::para
```

All the 'para' elements which are descendant of a 'chapter' element which is a 'doc' direct child of the root of the document.

XPATH's test and predicates

A test could select:

- nodes by name (eventually with namespace prefix)
- occurrences of a string: `text()`
- comments: `comment()`
- XML processing instructions: `processing-instruction()`
- any node: `node()`

A predicate filters the context, with

- position in the selected context sequence
- expression on structure, attributes ...
  - lot of library functions available: 112

XSL: eXtensible Stylesheet Language

Give "meaning" to XML documents with XML documents

- XSLT: Transformations
  - Document translation from a schema to another.
  - Declarative language
    - not how to transform,
    - but what I want.
- XSL-FO: Formatting Objects
  - A unified presentational language
  - XSL-FO document processed into PDF or PS formats

XSLT basics

Translation

- from a source XML document into a target XML document,
- declared in a stylesheet,
- operated by an XSLT processor.

Stylesheet essentially made of building declarations which:

1. select a pattern to search in the source document
   - selection language mostly based on XPATH
2. declare elements and text do be outputted in the target

XSLT declaration approaches

Pull
- iterative style
- good for simple data
- based on templates: define the skeleton of the target document and ask for inclusions from XML sources

Push
- recursive style
- good for full documents
- based on rewriting rules: for each input elements search for the most appropriate rule to apply

XSLT, an example

Example 12 (XML source doc).

```
<portfolio>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <sym>ZFFX</sym>
    <pr>92.250</pr>
  </stock>
  <!-- ... -->
</portfolio>
```

XSLT, an example

Example 13 (XSLT pull stylesheet). Producing an HTML4 document with previous data

```
<HTML
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/HTML4/">
  <BODY>
    <TABLE BORDER="2">
      <TR>
        <TD>Symbole</TD>
        <TD>Nom</TD>
        <TD>Prix</TD>
      </TR>
      <xsl:for-each select="portfolio/stock">
        <TR>
            <TD><xsl:value-of select="sym" /></TD>
            <TD><xsl:value-of select="name" /></TD>
            <TD><xsl:value-of select="pr" /></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </BODY>
</HTML>
```

2.3 XSLT processing

XSLT processing client-side: native

Native processing by browser

- XML processing instruction inside the document:

```
<?xml-stylesheet type="text/xml"
  href="style.xsl"?>
<mydoc> ... </mydoc>
```

- supported by recent browsers:
  - Chrome: XML, XSLT, and XPath from version 1.
  - IExplorer: XML, XSLT, and XPath from version 6 (not compatible),
  - Firefox: XML, XSLT, and XPath from version 3,
  - Safari: XML and XSLT from version 3,
  - Opera: XML, XSLT, and XPath from version 9.

XSLT processing client-side: JS

JS processing

- Needed when native browser support is missing,
- Useful for browser-targeted transformations.
- Procedure:
  1. load XML documents and XSLT sheets from the server,
  2. perform the transformation using JS libraries,
  3. insert the output (X)HTML in the original page.

Example: XSLT processing in Mozilla

```
// input XML
var xmlDoc = ...;

// Setup processor
var xslStylesheet = ...; // XSL
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(
    xslStylesheet);

// Transform and write DOM
var fragm =
    xsltProcessor.transformToFragment(
        xmlDoc,document);
document.body.appendChild(fragm);
```

Example: XSLT processing in IE

```
// Load XML
var xml = new ActiveXObject(
  "Microsoft.XMLDOM")
xml.async = false
xml.load("mydocument.xml")

// Load XSL
var xsl = new ActiveXObject(
    "Microsoft.XMLDOM")
xsl.async = false
xsl.load("mystylesheet.xsl")

// Transform
document.write(xml.transformNode(xsl))
```

XSLT processing client-side: PHP 4

XSLT library: Sablotron

- create a new XSLT processor

```
resource xslt_create (void)
```

- perform the transformation;

```
<?php mixed xslt_process(
    resource xh, string xml, string xsl
    [, string result
      [,array args [,array params]]]) ?>
```

- read the error number

```
<?php int xslt_errno (resource xh)?>
```

- read the error string

```
<?php mixed xslt_error (resource xh)?>
```

- destroy a processor

```
<?php void xslt_free (resource xh) ?>
```

XSLT processing client-side: PHP 5

XSLT library: `libxslt`

Example 14.

```
<?php
$xh = new xsltprocessor();
$xml=new DOMDocument();
$xml->load('travel.xml');
$xsl=new DOMDocument();
$xsl->loadXML(file_get_contents('travel.xsl'));
$xh->importStyleSheet($xsl);
$result=$xh->transformToXML($xml);
if ($result) { echo($result); }
else {
    echo("XSLT error: number ".xslt_errno()."\n");
    echo("             string ".xslt_error()."\n");
    exit; } ?>
```

3 Dynamic HTML

3.1 AJAX

Four-layers architecture 2.0

Browser loads:

- a simpler HTML page, with missing parts (template),
- JavaScript code (piece of program),
- some Ajax libraries (common implementation).

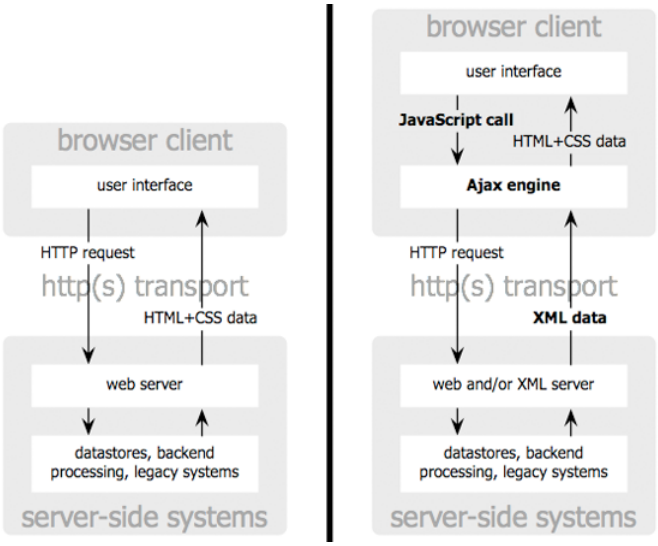
The client-side program can implement:

- presentation only: talks with the application-logic layer on server-side via XML,
- presentation and application-logic: almost everything on client-side, except for server requests and storage queries

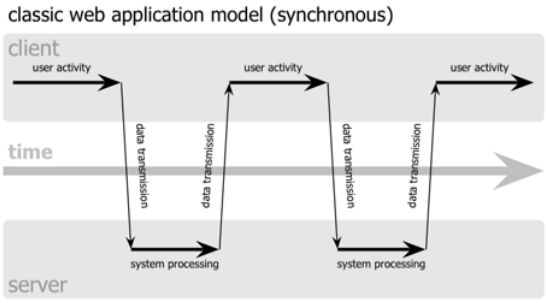
AJAX: Asynchronous Javascript And Xml

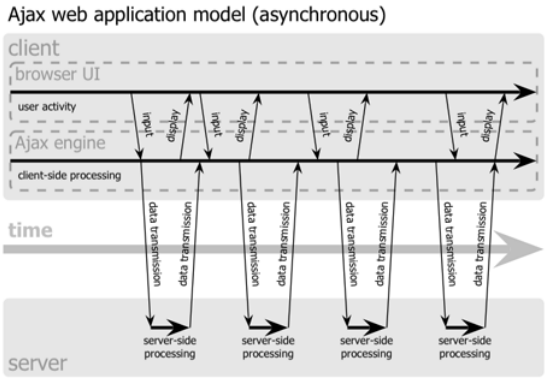
- Technique for implementation of web application
- Aims
  - interactivity: user experience improvement
  - speed: lower data exchange, lower server load
  - usability: lower refresh time
- Based on several existing technologies:
  - final documents: (X)HTML and CSS
  - document manipulation: JS modification on DOM
  - server communications: XMLHttpRequest (XHR)
  - intermediate formats: XML or JSON

Classic vs AJAX components



Classic vs AJAX communications





AJAX Cons

- Usability
  - no navigation in the browser
  - no bookmarking in the browser
  - no indexing by search engines
- Accessibility
  - no support for non-visual browser
  - need for alternative mechanisms

  Configuration
  - need JS enabling in the browser
  - need ActiveX objects enabling in IE browser

  Compatibility
  - need massive browser testing (many differences)
  - need alternatives for non-JS browsers

AJAX support and browser versions

- IE: since 5
- Gecko-based: Mozilla, Firefox and Netscape since 7.1
- KHTML/WebKit-based: Konqueror since 3.2, Safari since 1.2
- Opera since 8.0, Mobile version inlcuded

3.2 Structure of an AJAX application

AJAX application overview: XHR creation

Application in three phases:

1. Requests setup: creation and configuration
2. Requests execution: send requests, save replies
3. Page DOM manipulation.

Phase 1: XHR object creation

Example 15.

```
if (window.XMLHttpRequest) {
  // Gecko and KHTML/Webkit
  my_request = new XMLHttpRequest();
} else if (window.ActiveXObject) {
  // Internet Explorer
  try {
    my_request = new ActiveXObject(
      "Msxml2.XMLHTTP");
  } catch (e) {
    try {
      my_request = new ActiveXObject(
        "Microsoft.XMLHTTP");
    } catch (e) { /* ... */ } } }
```

Phase 1: XHR initialization

Specify the handler function (receiving the reply)

Example 16.

```
my_request.onreadystatechange =
  myHandlerFunction;
```

Open the connection, specifying:

- HTTP method
- destination URL
- asynchronous connection flag
- (username)
- (password)

Example 17.

```
my_request.open(
  'GET',
  'http://www.example.org/pa/t/h',
  true);
```

Phase 2: request submission

Send the request, specifying the body of the message, which typically is:

- `null` for GET methods (parameters are in the URL),
- query string for POST methods.

Example 18 (Sent a GET request).

```
my_request.send(null);
```

. . . but anything can be sent as body (e.g. complex request in XML): just set the MIME-type up.

Example 19 (Set the request handler).

```
my_request.setRequestHeader(
  'Content-Type', 'mime/type');
```

Phase 2: reply handling

The handler will check the request status:

0 uninitialized

1 loading

2 loded

3 interactive

4 complete

. . . and then reply with the appropriate status code (200, 404, 500. . .)

Example 20 (Handler function).

```
function nameOfTheFunction() {
  if (my_request.readyState == 4) {
    // reply received
    if (my_request.status == 200) {
      // Nickel!
    } else {
      // Ouch, an error? }
  }
} else { // reply not yet received }
```

Phase 2: reading the reply

Read the reply message as:

- plain text with `my_request.responseText`
- XMLDocument with `my_request.responseXML`

Example 21 (Simple document change).

```
// read the reply as XML tree
xmlDoc = my_request.responseXML;

// retrieve the desired fragment
fragment =
  xmlDoc.getElementsbyId("content").item(0);

// hot-modify on the current document
document.getElementById("area1").
  appendChild(fragment);
```

4 HTML5 new features

Document type declaration

- Starting idea for the new definition: the "abstract" language is defined by two syntaxes
  - XHTML
  - HTML

  ... but what really matters is the in-memory representation, the DOM HTML (yes, rather "concrete" as a definition).
- Document type statement:
  - Document type declaration now optional:

    ```
    <!DOCTYPE html>
    ```
  - MIME/content type stated in HTTP now mandatory:
    * `application/xhtml+xml` means it is XHTML5 document,
    * `text/html` means it is HTML5.

4.1 Document structure

Hierarchical containers

In HTML 4 and XHTML 1 the document structure is poor:

- layers for headings, not for content document (e.g. sections)
- generic container `<div>` with some `@class` attribute

New block elements, with:

- precise semantic,
- more easily processable.

Structure block elements:

- `<section>`
  - generic container (usually with attribute `@id`)
- `<article>`
  - self-contained and reusable portion of the document
  - used for blog post, news article, feed ...
- `<aside>`
  - portion which is related to, but not included in the text
  - used for side notes, sidebar, advertisement ...

Repeated blocks

Web pages often repeat pieces of content, common

- for all articles or sections of the site,
- for all pages of a paginated version of the content.

Repeated block elements:

- `<header>`
  - header of the current section or the whole page
  - can contain headings `<hN>`
  - used for table of contents, search form ...
- `<footer>`
  - conclusion of the current section or the whole page
  - not necessarily at bottom-page,
  - used for appendices, authors, copyright, colophon ...

Navigation block

Web pages often contain navigation links:

- internal, to other sections,
- external, to other page of the same site/

Navigation block element `<nav>`

- intended for containing lists of links,
- usually inside `<header>` of `<footer>`,
- more processable, therefore:
  - more customizable in presentation (e.g. based on user-agent),
  - more accessible by assistive technologies (e.g. screenreaders).

4.2 Canvas

Canvas

HTML5 introduces explicit drawing technologies

- Container block `<canvas>`
  - attributes `width` and `height` set the size
  - rectangular area where to draw
    * at page load, or
    * at some event;
  - coordinates (0,0) is the top-left corner.
- JavaScript API
  - method `getContext()` on the canvas,

– manipulation of objects in the context.

Example 22 (Canvas element).

```
<canvas id="squares" width="400" height="400">
    Not supported :-(
</canvas>
```

Drawing

Style properties, specifying color, gradient or pattern:

- `fillStyle`: for the internal area of shapes;
- `strokeStyle`: for the border of shapes or the lines.

Methods for drawing rectangles:

- `fillRect (x, y, width, height)`: draw a fill rectangle,
- `strokeRect (x, y, width, height)`: draw a rectangle,
- `clearRect (x, y, width, height)`: erase the rect. area.

Method for drawing lines, paths and polygons:

- `moveTo (x, y)`: move the pointer,
- `lineTo(x, y)`: trace a line starting from the pointer,
- `beginPath()`: begin a new path,
- `closePath()`: close the current path,
- `fill()`: fill the current path,
- `stroke()`: draw the current path lines.

Example

Example 23 (JavaScript drawing function).

```
function draw(){
  var canvas =
    document.getElementById('squares');
  if (canvas.getContext){
    var contx = canvas.getContext('2d');
    contx.fillStyle = "rgb(256,0,0)";
    contx.fillRect (0, 0, 200, 200);
    contx.strokeStyle = "rgb(0,100,20)";
    contx.strokeRect (40, 40, 220, 220); } }
```

Text

Text can be inserted in the canvas

- no structure, no flow,
- just as any other drawing object.

Properties for text drawing:

- `textAlign`: alignment,
- `font`: font family, size, and style.

Method for text drawing:

- `fillText(string, x, y)`: draw the string.

Images

Scalar images can be inserted in the canvas

- loaded from the page, or
- created by JavaScript.

Image drawing and resizing methods:

- `drawImage(img, x, y)`: draw the image with the top-left corner in the specified point fo the canvas;
- `drawImage(img, x, y, xsize, ysize)`: draw the resized image with the top-left corner in the specified point fo the canvas.

API and rich clients

Canvas JavaScript API:

- integrable with client side programming,
- user input event as triggers,
- powerful technologies for very rich applications.

Examples: multimedia players, graphical applications, games ...

- http://www.canvasdemos.com/
- http://zwibbler.com/

4.3 Audio and video

Audio and video

HTML5 introduces video and audio playing features:

- no more external plugins: Flash, Real Player, MPlayer, Quicktime
- loading file and stream media,
- DOM properties for managing playing,
- new elements: `<audio>` and `<video>`

Main attributes:

- `src`: URL of the media, file or stream,
- `width` and `height`: video window size,
- `controls`: flag for setting browser or customized player controls
- `autoplay`: flag for setting the automatic start on page load.

Sub-elements:

- `<source>`: for multiple media specification,
- `<track>`: subtitles or notes specification: source, language ....

Video codecs

The originally proposed codec was Ogg Theora:

- open format, with *libre* implementation,
- royalty free.

Video codec war:

Mozilla wants Ogg Theora

Apple wants H.264:

- support already present in its devices (iStuff),
- patents partially owned (others are Microsoft's).

Google wants VP8 (project WebM):

- proprietary format, acquired and then freed,
- recent support from Mozilla and Opera foundations.

Conclusions:

- no specification of codec for HTML5,
- full support need multiple of encoding.

API

Audio and video elements offers JavaScript methods to:

- control the playing `play()` and `pause()`;
- handle errors,
- ... then implement a fully-featured media player.

4.4 Form

Form

HTML5 introduces additional support to forms:

- simplifying and quickening user completion,
- checking user input.

New attributes of `<input>` element:

- `placeholder`: descriptive string for non-focused text field,
- `autofocus`: first field to be focused at form loading.

Input types

New types of `<input>` element

- `email`: syntactically correct email address
- `url`: syntactically correct URL;
- `number`: number, eventually bounded, with raising and lowering buttons and customizable step;
- `range`: bounded number, with a sliding selector and customizable step;
- `date`: date, with calendar selector;
- `color`: RGB code, with palette selector.

Support:

- very mixed, from browser to browser,
- need to checked, see http://wufoo.com/html5/

Automatic validation

Form validation

- usually performed with JavaScript functions,
- with HTML5 directly performed by the browser.

Validation features:

- automatic validation at `submit` event,
- `invalid` event in case of errors,
- disabled validation with `novalidate` attribute of:
  - `<form>` element, for the whole form, or
  - `<input>` element, for a specific field.
- compulsoriness of fields with `required` attribute of `<input>`.

4.5 Local storage

Local storage
HTML introduces support for permanent storage

- beyond little cookies,
- data accessible without server connection (offline browsing),
- unification of various existing technologies (e.g. Google Gears, IE userData).

Data organized as pairs $\langle key, value \rangle$ accessed with

- specific methods, such as `localStorage.getItem(k)` and `localStorage.setItem(k, v);`
- associative arrays, such as `localStorage[k] = v`.

Security issues
Types:

- all data stored as strings, eventually to be casted;
- no control on data types.

Code:

- can be stored, loaded and executed;
- higher risk of infections.

Privacy:

- malicious code can access some local storage data,
- higher risk of information disclosure.

4.6 And much more

Other new features
Geolocation

- aim: target information to the user position,
- dedicated JavaScript API, e.g. `getCurrentPosition()` method
- opt-in policy: no information sent without user's approval,

Microdata

- semantic information about document fragments,
- approach alternative to RDF and Semantic Web.

XML objects integration

- SVG for vectorial images
- MathML for complex mathematical content.

History API

- client-side application has few URL change, then no history
- support for JavaScript: bookmarking, backward navigation ...

Web workers

- low priority JavaScript tasks,
- high-load in background: no impact on user experience.

Conclusions

The end

Memorandum
Benefits of standards for formats, languages, protocols:

- maximize end-user support,
- minimize reuse of technologies,
- maximize future compatibility.

Benefits of layering (*divide et impera*):

- master complexity.
- increase extensibility,
- increase scalability.

Benefits of reuse

- minimize effort: do not rediscover the wheel!
- maximize understandability: use design patterns
- learn from recurrent mistakes: beware of anti-patterns!