HTML5 : An Introduction

Anjan

anjan.mitra@gmail.com

Agenda

- Data Storage
- Geolocation
- WebWorkers
- Messaging
- Web Sockets
- Offline Application

Quick Recap

- History, Vision And Future Of HTML5
- Getting Started With HTML5
- Structure Of A Web Page
- Forms
- Audio And Video
- Canvas

Html5 : Data Storage

Cookie

- Written by Server : used to track user information across web pages.
- Cookie values are transmitted back and forth every time a user visits a domain.
  - a small piece of text that allow data storage for a client
  - name/value pairs
  - used to remember user preferences on a web site.
    - document.cookie = "name=value; path=/; expires=date";

5

Problems With The Existing Cookie-based Model

- Cookie :
  - So, it is visible on the network & thus a security risk.
  - Consumes bandwidth.
- Downside:
  - Limited size : 4KB only per cookie .

6

Hacks Prior To HTML5

- To circumvent those limitations :
  - userData for Microsoft Internet Explorer
  - Flash cookies by Adobe
  - Gears by Google
  - dojox.storage

7

New Storage Options

- HTML5 provides a standardized and native API.
  - easily retrievable JavaScript objects that persist across page loads.
- Ways to store data on the client side:
  - Web Storage: Key/Value pair storage system
    - localStorage
    - sessionStorage
  - Web SQL Database: no more support from W3C.

8

Advantage

- Values survive either across page
  - Loads in a single window or tab
  - Across browser restarts.
- Stored data is not transmitted across the network.
- Values as high as a few megabytes.
- Thus, safe to use & has widespread support.

9

Checking for Browser support

```
function checkStorageSupport() {

  //sessionStorage
  if (window.sessionStorage) {
    alert('This browser supports sessionStorage');
  } else {
    alert('This browser does NOT support sessionStorage');
  }

  //localStorage
  if (window.localStorage) {
    alert('This browser supports localStorage');
  } else {
    alert('This browser does NOT support localStorage');
  }
}
```

10

Difference

sessionStorage	localStorage
Values persist only as long as the window or tab in which they were stored.	Values persist beyond window and browser lifetimes.
Values are only visible within the window or tab that created them.	Values are shared across every window or tab running at the same origin.

11

Storage Event

- An event mechanism
  - allows notifications of data updates to be communicated to interested listeners.
  - used to communicate between windows on the same origin.
- Fired on the window object for every window of the same origin as the storage operation.
  - window.addEventListener ("storage", displayStorageEvent, true);

12

Storage Interface

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
  deleter void removeItem(in DOMString key);
  void clear();
};
```

13

StorageEvent Interface

```
interface StorageEvent : Event {
  readonly attribute DOMString key;
  readonly attribute any oldValue;
  readonly attribute any newValue;
  readonly attribute DOMString url;
  readonly attribute Storage storageArea;
};
```

14

Html5 : Geolocation

15

Introduction To Geolocation

- Created by W3C not WHATWG
- Separate specification: Geolocation API Specification
- Users can share their location, find direction, etc.

16

Privacy Considerations

- Sharing user location ➔ a privacy concern!
- User must give explicit permission to the user agent
  - The user interface must include the URI of the document origin.
- Some privacy concerns that remain:
  - How long the location data is stored?
  - Is it shared with other sites/partners?
  - Can the location data be updated or deleted by the end user?

17

Many Ways To Get User Location

- GPS
- Location inferred from network - IP address, WiFi, etc.
- Cell IDs
- User input

18

IP Address-based Geolocation

Pros	Cons
Available everywhere	Not very accurate (wrong many times, but also accurate only to the city level)
Processed on the server side	Can be a costly operation

19

GPS-based Geolocation

Pros	Cons
Very accurate	It can take a long time getting a location fix, which can drain a user's device's batteries
	Does not work well indoors
	May require additional hardware

20

WiFi-based Geolocation

Pros	Cons
Accurate	Not good in rural areas with few wireless access points
Works indoors	

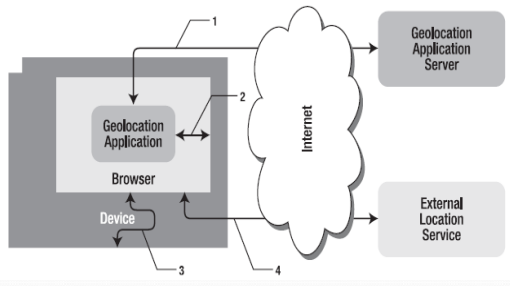
21

Cell-based Geolocation

Pros	Cons
Fairly accurate	Requires a device with access to a cell phone or cell modem
Works indoors	Not good in rural areas with fewer cell phone towers

22

Info Flow



23

Two Main Methods

- The Geolocation is a child of the Navigator object.
- Geolocation object:
  - getCurrentPosition()
  - watchPosition()
  - clearWatch(in long watchId)

```
if (navigator.geolocation) { //❶
  navigator.geolocation.getCurrentPosition(successCallback, errorCallback); //❷
} else {
  error('Sorry, your browser does not support geolocation');
}
```

24

The Position Object

- Used by the success callback of the getCurrentPosition() and watchPosition()
- The Position object has two properties:
  - A Coordinates object (coords)
  - A DOMTimeStamp object (timestamp)

```
interface Coordinates {
  readonly attribute double latitude; //❶
  readonly attribute double longitude; //❷
  readonly attribute double? altitude; //❸
  readonly attribute double accuracy; //❹
  readonly attribute double? altitudeAccuracy; //❺
  readonly attribute double? heading; //❻
  readonly attribute double? speed; //❼
};
```

25

Success Handler

```
function successCallback(pos) {
  alert("Latitude: " + pos.coords.latitude + " , Longitude: " + pos.coords.longitude);
}

if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
} else {
  error('Sorry, your browser does not support geolocation');
}
```

26

Handling Errors

- Error callback ➔ uses a PositionError object

```
interface PositionError {
  const unsigned short PERMISSION_DENIED = 1; //❶
  const unsigned short POSITION_UNAVAILABLE = 2; //❷
  const unsigned short TIMEOUT = 3; //❸
  readonly attribute unsigned short code; //❹
  readonly attribute DOMString message; //❺
};
```

27

Error callback

```
function errorCallback (err) {
  switch(err.code) {
    case 1:
      alert("Sorry, but this application does not have the permission to use geolocation
      break;
    case 2:
      alert("Sorry, but a problem happened while getting your location");
      break;
    case 3:
      alert("Sorry, this is taking too long...");
      break;
    default:
      alert("unknown");
  }
}

// Success Callback not shown here
// ...
// ...

if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
} else {
  error('Sorry, your browser does not support geolocation');
}
```

28

PositionOptions Object

- Its the last parameter of the getCurrentPosition and the watchPosition method.
- It allows the developers to specify options when getting user location.

```
interface PositionOptions {
  attribute boolean enableHighAccuracy; //❶
  attribute long timeout; //❷
  attribute long maximumAge; //❸
};
```

29

Html5 : Web Workers

Anjan

Current JavaScript Execution Model

- Runs in single-threaded environment.
  - Application could become unresponsive.
- Solution :
  - Break a big operation into smaller ones
  - Use timer e.g. setTimeout()

31

Introduction To Web Workers

- Run another script independent from the user interface script.
- Thus :
  - A worker is a thread - perform tasks in the background.
  - A worker is a JavaScript file that contains the "task" you want to perform in a separate thread.

32

Introduction To Web Workers

- Restrictions :
  - We cannot access everything we want in a worker - don't have direct access to:
    - the web page
    - the DOM API
- What we can access:
  - location object (read-only).
  - importScripts() method allows you access to script files in the same domain.
  - JavaScript objects such as Object, Array, Date, Math, String.
  - XMLHttpRequest object.
  - setTimeout() and setInterval() methods.

33

Usage Of Web Workers

- Are relatively heavy-weight.
- Are not intended to be used in large numbers.
  - Expected to be long-lived
  - Have a high start-up performance cost
  - Have a high per-instance memory cost.

34

Usage Of Web Workers

- Compute intensive e.g. calculate prime numbers.
- Operations on a client-side database.
- Parsing.
- Image or video processing.
- Etc...

35

Check for Browser Support

```
function loadDemo() {
  if (typeof(Worker) !== "undefined") {
    document.getElementById("support").innerHTML =
            "Excellent! Your browser supports HTML5 Web Workers";
  }
}
```

36

Communication APIs

- To spawn a new worker:
  var robot=new Worker('robot.js');
- To send a message between the main thread and the worker:
  robot.postMessage(msg_for_worker);
- To receive the message:
  self.onmessage=function(event){ //Do something };
  - The actual message is contained in event.data

37

Loading and Executing JS

- In the document : <script> … </script>
- Within the script:
  - importScripts("helper.js");
  - importScripts("helper.js", "anotherHelper.js");
- To stop :
  - worker.terminate(); // the page that created it, can stop it.

38

Using Timers

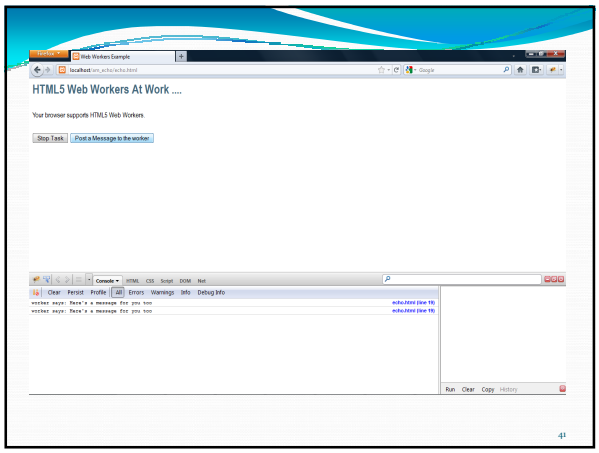
- Although HTML5 Web Workers cannot access the **window** object, they can make use of the full JavaScript timing API, typically found on the global window:
  - var t = setTimeout(postMessage, 2000, "delayed message");

39

Handling Errors

- Inside a worker, you can register to the error event :
  self.onerror=function(error) { //Do something };
- ErrorEvent interface :
  - filename: Name of the worker that caused the error
  - lineno: Line number where the error occurred
  - message: The error message

40



Code Snippets

```
document.getElementById("helloButton").onclick = function() {
    worker.postMessage("Here's a message for you");
}
```

```
worker.addEventListener("message", messageHandler, true);

function messageHandler(e) {
    // process message from worker
}
```

```
function errorHandler(e) {
    console.log(e.message, e);
}
```

To handle the errors, you must add an event listener to the main page:

```
worker.addEventListener("error", errorHandler, true);
```

Html5 : Messaging

Understanding the same origin policy

- Origin: SCHEME (PROTOCOL) + HOST + PORT
- Implemented into major browsers.
- It prevents a document or script loaded from one site of origin A from manipulating properties of or communicating with a document loaded from another site of origin B.
- It prevents attacks like hijacking an existing user session or phishing.

Understanding the same origin policy

URL	Same origin?	Reason
http://www.company.com/page2.html	YES	
http://www.company.com/dir/page.html	YES	
http://blog.company.com/page.html	NO	Different host
https://www.company.com/page2.html	NO	Different protocol
http://www.company.com:8080/page2.html	NO	Different port

45

Understanding the same origin policy

- Browsers check access against the same origin policy when for instance when we:
  - Manipulate browser windows
  - Request URL via the XmlHttpRequest (XHR)
  - Manipulate frames (including inline frames)
  - Manipulate documents (included using the object tag)
  - Manipulate cookies

46

Workaround to the same origin policy

- Using a proxy.
  - Makes the whole process slower.

47

Cross-document messaging

- Allows document from two different origin to communicate together in a secure way.
- Communication works with messages.
- The message event is used in :
  - Server-sent events.
  - Web sockets.
  - Cross-document messaging

48

Server Sent Event

- Provides push notifications from a server to a browser client in the form of DOM events.
- The Server-Sent Events EventSource API is now being standardized.
- It is commonly used to send message updates or continuous data streams to a browser client and designed to enhance native, cross-browser streaming through a JavaScript API called EventSource

EventSource Client API

- Unlike a WebSocket connection, SSE requires no additional HTTP handshakes.
- For all intents and purposes, an SSE channel is simply a long-lived, streaming HTTP connection.
  - This alone means that you can implement an SSE service on top of any streaming capable HTTP server.

EventSource

```
var source = new EventSource ('/my/sse_endpoint');
source.addEventListener('message', function(e) {console.log(e.data);
    });
source.addEventListener('open', function(e) { // Connection was opened.
    });
source.addEventListener('error', function(e)
     {if (e.eventPhase == EventSource.CLOSED) {// Connection was closed.}});
```

SSE vs. WebSockets

- Technically speaking, the WebSockets spec is a superset of SSE
- Then why do we need both ?
  - Simplicity of the EventSource API - both on the client, and especially on the server.
  - Chances are, many apps won't actually need the bi-directional capabilities of WebSockets,
- You can benefit a great deal from the simplified setup and deployment enabled by SSE.

Cross-document messaging

- MessageEvent has two important properties:
  - *data* that contains the actual message.
  - *Origin* of the message so we can check if we want to process it or not.

53

Checking for Browser Support

```
if (typeof window.postMessage === "undefined") {
   // postMessage not supported in this browser
}
```

54

Sending a message

```
<iframe id="iframe" src="http://output.example.com/output.html"></iframe>
<script>
  ...
  var consoleOutput=document.getElementById("iframe").contentWindow;
  consoleOutput.postMessage('message','http://output.example.com'); ❶
  ...
</script>
```

When sending a message (postMessage):

- Specify the receiver origin.
- The origin of the sender is automatically sent in the message.
  - It cannot be spoofed.

55

Receiving a message

```
<div id="consoleOutput"></div>
<script>
  ...
  function handleMessage(e) {
      if (e.origin == "http://input.example.com") { ❶
          var consoleOutput=document.getElementById("consoleOutput");
          consoleOutput.textContent+= e.origin + ": " + e.data + "\n"; ❷
      } else {
          // If the message is coming from an untrusted origin, just ignore it :)
      }
  }
  ...
  window.addEventListener("message", handleMessage, true);
</script>
```

After receiving a message (handleMessage):

- Check the origin of the sender and only process those from trusted origin.
- The message itself is contained in the data property of the message event.

56

addEventListener

```
window.addEventListener("message", messageHandler, true);
function messageHandler(e) {
    switch(e.origin) {
      case "friend.example.com":
      // process message
      processMessage(e.data);
      break;
    default:
      // message origin not recognized
      // ignoring message
  }
}
```

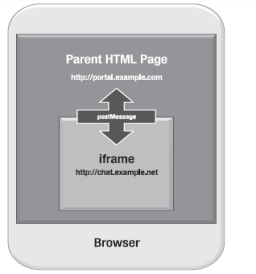
57

XMLHttpRequest Level 2

- It's the main JavaScript API to do AJAX.
- Has the ability to do cross-origin requests.

58

postMessage communication



59

Making a request

- The call is same as before.
- However, behind the scenes the request headers contain a *Origin* header.
  - Then server knows where does the request comes from.
- For other HTTP methods than GET, HEAD and POST (called simple methods), a preflight request is made to the server.

```
var client = new XMLHttpRequest();
client.open("GET", "http://fortune.com/tellFortune");
client.onreadystatechange = function() { /* do something */ }
client.send();
```

60

Making a request

- The origin header specifies where does the request comes from.

```
GET http://anotherdomain.com/script.php HTTP/1.1
Accept: */*
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Host: anotherdomain.com
Origin: http://main.html5learning.com:8000 ❶
Referer: http://main.html5learning.com:8000/
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_5; en-US) AppleWebKit/534.13
```

61

Making a request

- This response header specifies what origin is allowed to make a request

```
Access-Control-Allow-Origin:http://main.html5learning.com:8000 ❶
Access-Control-Allow-credentials:true ❷
Connection:Keep-Alive
Content-Encoding:gzip
Content-Length:83
Content-Type:text/html
Date:Wed, 02 Mar 2011 05:31:28 GMT
Keep-Alive:timeout=10, max=30
Server:Apache
Vary:Accept-Encoding
X-Powered-By:PHP/5.2.16
```

62

Progress Event

- Previous version of the *XMLHttpRequest* had only one event: *readystatechange*
  - This *readystatechange* is inconsistently implemented across browsers

63

Progress Event

- XMLHttpRequest level 2 has several new progress events & fine control :
  - loadstart
  - progress
  - abort
  - error
  - load
  - timeout
  - loadend

64

Html5 : Web Sockets

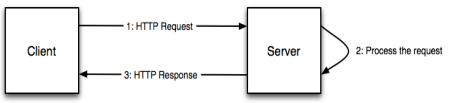
65

Overview

- Allows :
  - real-time
  - full duplex communication
  - Uses a single socket.
- Usage scenarios :
  - Chat : Facebook chat
  - Live sports result
  - Stock quote application

66

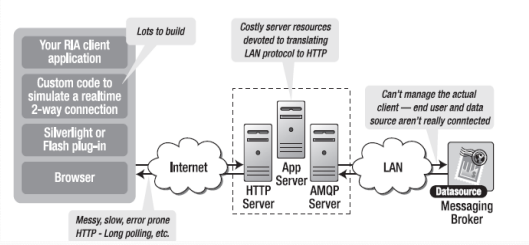
Current Scenario



- The HTTP protocol is half duplex.
- So, wrt "real-time" web applications many problems arise:
  - Data might get outdated by the time user sees it.
  - Very difficult to predict the change of data on the server side.
  - Continuously refreshing the page ⇒ very poor user experience!
- Solution : Ajax

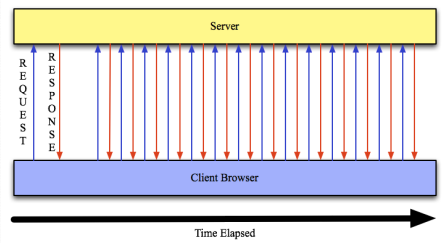
67

Complexity of real-time HTTP



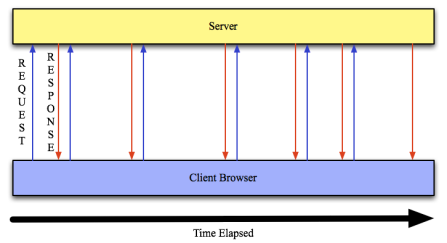
68

Short Polling (Pull)



69

Long Polling (async polling)



70

Current Network Overhead – Scenarios

- Use case A: 1,000 clients polling every second:
  - Network traffic is (871 × 1,000) = 871,000 bytes = 6,968,000 bits per second (6.6 Mbps)
- Use case B: 10,000 clients polling every second:
  - Network traffic is (871 × 10,000) = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)
- Use case C: 100,000 clients polling every 1 second:
  - Network traffic is (871 × 100,000) = 87,100,000 bytes = 696,800,000 bits per second (665 Mbps)

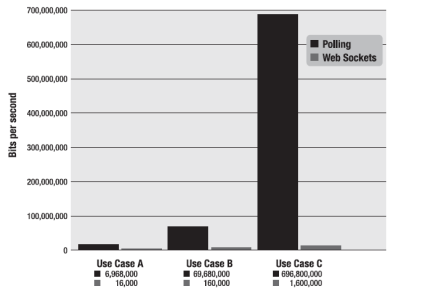
71

Network Overhead with WS – Scenarios

- Use case A: 1,000 clients receive 1 message per second:
  - Network traffic is (2 × 1,000) = 2,000 bytes = 16,000 bits per second (0.015 Mbps)
- Use case B: 10,000 clients receive 1 message per second:
  - Network traffic is (2 × 10,000) = 20,000 bytes = 160,000 bits per second (0.153 Mbps)
- Use case C: 100,000 clients receive 1 message per second:
  - Network traffic is (2 × 100,000) = 200,000 bytes = 1,600,000 bits per second (1.526 Mbps)

72

Comparison wrt the Overhead



73

Overview Of Web Sockets API And Protocol

- Full duplex
- Bi-directional communication channel using a single socket (TCP)
- The Web Sockets protocol exists with two schemes: ws: or wss: for secure web sockets
- There is a handshake to upgrade the connection from HTTP protocol to web socket protocol.

74

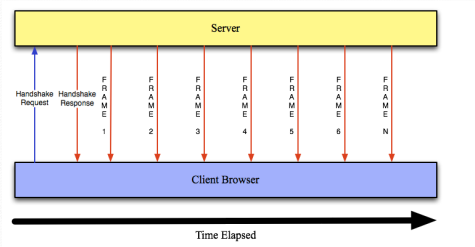
WebSocket Upgrade handshake

```
From client to server:
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4@1 46546xW%0l 1 5
Origin: http://example.com
```

```
From server to client:
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://example.com
WebSocket-Location: ws://example.com/demo
WebSocket-Protocol: sample
```

75

Web Sockets



76

Checking the Browser

```
function loadDemo() {
  if (window.WebSocket) {
    document.getElementById("support").innerHTML = "HTML5 WebSocket is supported in your
                                                    browser.";
  } else {
     document.getElementById("support").innerHTML = "HTML5 WebSocket is not supported in
                                                     your browser.";
  }
}
```

77

Interface Web Socket

```
interface WebSocket {
  readonly attribute DOMString URL;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
  attribute Function onopen;
  attribute Function onmessage;
  attribute Function onclose;
  boolean send(in DOMString data);
  void close();
};
```

78

Async Event Listeners

```
myWebSocket.onopen = function(evt) { alert("Connection open ..."); };
myWebSocket.onmessage = function(evt) { alert( "Received Message: " + evt.data); };
myWebSocket.onclose = function(evt) { alert("Connection closed."); };
```

79

Methods

- To create a Web Socket
  var socket=new WebSocket("ws://127.0.0.1:8080/");
- To send a message to the server
  socket.send ("This is the message sent by the client");

80

Methods

- To define event listeners for the Web Socket:

```
socket.onopen = function(){ alert("Connection to the socket is now open"); }

socket.onmessage = function(e){ alert("Message received: " + e.data); }

socket.onclose = function(e){ alert("Connection to the socket is closed"); }
```

81

Advantages Of Web Sockets

- HTTP request and response headers are sent only once during the handshake
  - A frame size is at least 2 byte: [oxoo byte][UTF-8 data message][oxFF byte]
- The server notifies the client of new messages, there is no need to hammer the server with unnecessary requests!
  - All of this obviously lead to reduce the network traffic and the network latency.

82

Offline Web Application

83

Typical Use cases

- Read and compose e-mail
- Edit documents
- Edit and display presentations
- Create to-do lists

84

Usefulness

- Can avoid the normal network requests needed to load an application.
  - Most of the application can load very quickly out of the local application cache, esp. for mobiles.
- If the cache manifest is up to date, the browser knows it does not need to check if the other resources are also up to date

85

Application Cache

- *application cache, which* is the place where browsers store the resources persistently, typically on disk.
  - about:cache page in Firefox

```
if(window.applicationCache) {
        // this browser supports offline applications
}
```

86

The manifest Attribute on the HTML Element

```
<!DOCTYPE  html>
<html manifest="application.manifest">
      .
      .
      .
</html>
```

87

Going Offline

- Some additions to the window.navigator object:
  - navigator.onLine -> bool

88

Checking Online Status

```
// When the page loads, set the status to online or offline
function loadDemo() {
  if (navigator.onLine) {
    log("Online");
  } else {
    log("Offline");
  }
}

// Now add event listeners to notify a change in online status
window.addEventListener("online", function(e) {
  log("Online");
}, true);

window.addEventListener("offline", function(e) {
  log("Offline");
}, true);
```

89

Manifest Files

#comments begin with the hash symbol
CACHE MANIFEST
files to cache
about.html
html5.css
index.html
boat.JPG

#do not cache
NETWORK
signup.html

FALLBACK
signup.html offline.html
/app/ajax/ default.html

90

window.applicationCache.status

Numerical Property	Cache Status
0	UNCACHED
1	IDLE
2	CHECKING
3	DOWNLOADING
4	UPDATEREADY
5	OBSOLETE

91

Common Events and Their Cache States

Event	Associated Cache State
onchecking	CHECKING
ondownloading	DOWNLOADING
onupdateready	UPDATEREADY
onobsolete	OBSOLETE
oncached	IDLE

92

Events Indicating Update Progress

- onerror
- onnoupdate
- onprogress

93

Developing Accessible Web App

94

WAI & ARIA

- WAI, stands for Web Accessibility Initiative
  - www.w3c.org/WAI
- ARIA: Accessible Rich Internet Application
- Accessible web applications provide means for people with disabilities to view, understand, and perform all the major tasks with the application.
  - Thus, applications that are equally usable for everyone.

95

WAI

- Keyboard Navigation
  - Users should be able to navigate through the web application and perform available tasks with keyboard e.g. tabbing.
- Use of Colour
  - Judicious use makes it easier and more comfortable to read the content for everyone.
  - Also, applications should not rely solely on colour to convey important meaning.

96

WAI

- CSS Sprites
  - Labels should be provided for the missing sprite images and they should perform the some action as the image.
- Role and States
  - A Screen reader used to just read out all content on the page.
  - Adding roles and states to the document so the screen reader can focus on only the important elements.
    - So, it's easier to understood by the listeners.

97

HTML5

- Accessibility overlaps a lot with usability,
  - *web-standards + usability = accessibility*.
- HTML5 contains several new elements that are considered semantic in that they more accurately describe the content they contain than a generic element such as a div.
- These new elements improve accessibility as standalone structure.
  - header, footer, section, article, aside, and nav.

98

ARIA roles

- The roles are banner, complementary, contentinfo, form, main, navigation, search.
- The header element
- If the header element is used at the main header for the entire page, the role banner can be assigned to it.
  - Only one header on a page can have the role banner.

  <header role="banner">. . .</header>
- Within the page there may be header elements used as article or section or aside headings. The ARIA role that can by used in that case is **heading**.

99

ARIA roles

<article> <header role="**heading**">
Article heading
</header>
Article body .....
</article>

- The heading role can also be used in tables.

100

ARIA roles

The footer element:

- The main page footer can use the role contentinfo.
  - But footer elements for subsections of a page may not use this role. It can only be used once on a page.

<footer role="**contentinfo**"> footer for entire page </footer>

- Depending on what kind of information is included in the footer for articles or other smaller page sections, the footer might be appropriately labeled with the complementary role.

<footer role="**complementary**">
informative footer for an article
</footer>

101

ARIA roles

The aside element:

- The aside element is meant for complementary material, not crucial to the page content, but supplementary.
  - Therefore the complementary role is perfect for it.

<aside role="**complementary**">
supplemental content
</aside>

102

ARIA roles

Main, Navigation, Form and Search roles:

- It is self-evident when to apply navigation, form, and search to page elements.
- But **main** is another role that can only *be used once* on a page to indicate the main content, which might be contained in a div or a section.

103

Examples

```
<header role="banner">
<nav role="navigation">
<aside role="complementary">
<section role="main">
<footer role="contentinfo">
```

104

Best Practices for a Better User Experience

105

Suggestions

- Use web storage in place of cookies.
  - USE CLIENT-SIDE storage INSTEAD OF SERVER ROUNDTRIPS.
- Use css transitions instead of javascript animation
  - Hardwareacceleration
- Use cache manifest for live sites, not just offline apps
- For cpu-heavy operations, web workers deliver
- Html5 form attributes and input types

106

Suggestions

- Use css3 effects instead of requesting heavy image sprites
- Websockets for faster delivery with less bandwidth than XHR
- Separate content from presentation
- Don't overuse div.
- Always declare a doctype
- Close Your Tags
- Use Lower Case Markup

107

Q & A

Thank You !!

108