
HTML 5
The New Standard For HTML
BUSINESS
mos.bayomi@gmail.com

HTML 5 GEOLOCATION

You Are Here (And So Is Everybody Else)

mos.bayomi@gmail.com

HTML 5 Geolocation

- Geolocation is the art of figuring out where you are in the world and (optionally) sharing that information with people you trust.
- There are many ways to figure out where you are:
  - Your IP address.
  - Your wireless network connection.
  - Which cell tower your phone is talking to.
  - Or dedicated GPS hardware that receives latitude and longitude information from satellites in the sky.

mos.bayomi@gmail.com

The Geolocation API

- Geolocation API lets you share your location with trusted websites.
- The ***latitude*** and ***longitude*** are available to JavaScript on the page.
- Which in turn can send that information back to the remote web server and do fancy location-aware things like finding local businesses or showing your location on a map.

mos.bayomi@gmail.com

The Geolocation API(cont'd)

- The geolocation API centers around a new property on the global `navigator` object:

```
navigator.geolocation
```

The simplest use of the geolocation API looks like this:

```
function get_location() {
navigator.geolocation.getCurrentPosition(show_map);
}
```

To detect support for the geolocation API you can use Modernizr:

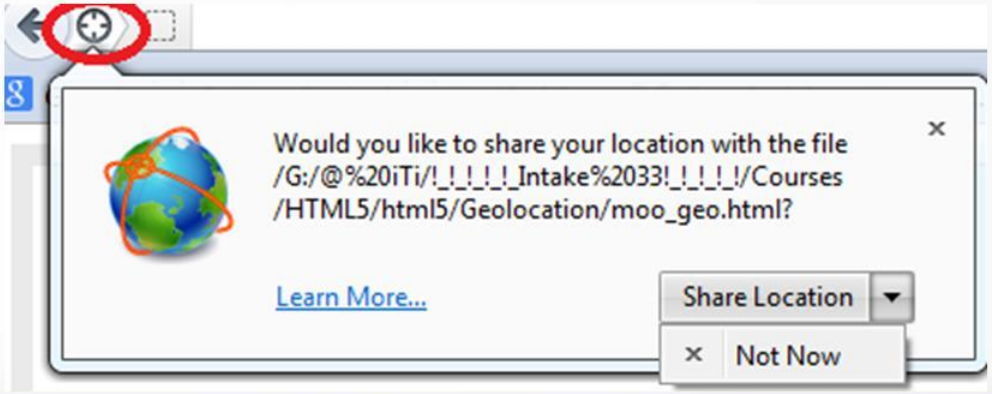
```
if (Modernizr.geolocation)
```


mos.bayomi@gmail.com

The Geolocation -- Security

- Your browser will never force you to reveal your current physical
- location to a remote server.

- The user experience differs from browser to browser.
-
- In Mozilla Firefox, calling the `getCurrentPosition()` function of the geolocation API will cause the browser to pop up an "infobar" at the top of the browser window.



`getCurrentPosition(callback function)`

- You just saw the *JavaScript* code that causes this *infobar* to appear.
- It's a single function call, that takes a callback function (which we called `show_map()`).
- The call to `getCurrentPosition()` will return immediately, but that doesn't mean that you have access to the user's location.
- The first time you are guaranteed to have location information is in the callback function.

mos.bayomi@gmail.com

getCurrentPosition(callback function)

- The callback function looks like this:

```
function show_map(position) {
var latitude = position.coords.latitude;
var longitude = position.coords.longitude;
// let's show a map or do something interesting!
}
```

- The callback function will be called with a *single* parameter, an object with two properties: `coords` and `timestamp`.

- The timestamp is just that, the date and time when the location was calculated.

mos.bayomi@gmail.com

Slide Notes

(Since this is all happening asynchronously, you can't really know when that will happen in advance.

It might take some time for the user to read the info bar and agree to share her location, devices with dedicated GPS hardware may take some more time to connect to a GPS satellite, and so on.)

The `position.coords` object

- The `position.coords` object have some properties:

Property	Type	Notes
`coords.latitude`	double	Decimal degrees
`coords.longitude`	double	Decimal degrees
`coords.altitude`	doubleor null	Meters above the reference ellipsoid
`coords.accuracy`	double	Meters
`coords.altitudeAccuracy`	doubleor null	Meters
`coords.heading`	doubleor null	Degrees clockwise from true north
`coords.speed`	doubleor null	Meters/second

Slide Notes

Only three of the properties are guaranteed to be there (coords.latitude, coords.longitude, and coords.accuracy).

The rest might come back as null, depending on the capabilities of your device and the backend positioning server with which it communicates.

The headingand speedproperties are calculated based on the user's previous position, if possible.

Handling Errors

- Your web application wants the user's location but the user doesn't want to give it to you.
- But what does that look like in code?

- It looks like the second argument to the `getCurrentPosition()` function—an error handling callback function:

```
navigator.geolocation.getCurrentPosition(
            show_map, handle_error).
```

- If anything goes wrong, your error callback function will be called with a `PositionError` object.
- It has two properties :- `code` and `message`

Handling Errors(cont'd)

- The `code` property will be one of the following:
- PERMISSION_DENIED(1): if the user clicks the "Don't Share" button or otherwise denies you access to his location.
- POSITION_UNAVAILABLE(2): if the network is down or the positioning satellites can't be contacted.
- TIMEOUT(3): if the network is up but it takes too long to calculate the user's position.
- UNKNOWN_ERROR(0) if anything else goes wrong.

Handling Errors(cont'd)

```
function handle_error(err) {
   if (err.code== 1) {
   // user said no!
   }
}
```


mos.bayomi@gmail.com

PositionOptions object

- The `getCurrentPosition()` function takes an optional third argument,a `PositionOptions` object.

- There are three properties you can set in a `PositionOptions` object:
- `enableHighAccuracy`: is exactly what it sounds like.

- `timeout`: specifies the number of milliseconds your web application is willing to wait for a position.

- `maximumAge`: allows the device to answer immediately with a cached posi-tion.

mos.bayomi@gmail.com

Slide Notes

Ex:

```
navigator.geolocation.getCurrentPosition(
success_callback, error_callback, {maximumAge: 75000});
```

What you're saying is that you don't necessarily need the user's currentlocation.

You would be satisfied with knowing where he was 75 seconds (75000 milliseconds) ago.

watchPosition()

- If you need to find the user's location continuously, You need to use the `watchPosition()` function.
- It has the same structure as `getCurrentPosition()`.
- The difference is that your callback function will be called every time the user's location changes.
- There is no need to actively poll the position.
- The device will determine the optimal polling interval, and it will call your callback function whenever it determines that the user's position has changed.

mos.bayomi@gmail.com

clearWatch()

- The `watchPosition()` function itself returns a number.
- You should probably store this number somewhere.
- If you ever want to stop watching the user's location changes, you can call the `clearWatch(t)` method and pass it this number, and the device will stop calling your callback function.
- It looks like `setInterval()` and `clearInterval()`

mos.bayomi@gmail.com

google maps API

- With `geolocation` object API you can get lat and long.
- To use this properties on the map you can use google maps API.
- First: link to the google maps API.

```
<script
src=" http://maps.google.com/maps/api/js?sensor=false">
</script>
```

- Second: create a google map object

```
var map = new
google.maps.Map(document.getElementById("map_div"), myOptions);

//the div to display on , and options for map
```


mos.bayomi@gmail.com

google maps API(cont'd)

```
var myOptions = {
 zoom: 8,   //the zoom level 0 to 21
 center: new google.maps.LatLng(latitude, longitude),
//the center of map displayed
 mapTypeId: google.maps.MapTypeId.ROADMAP
//Map Type
  }
```

- Map Type
  - ROADMAP displays the normal, default 2D tiles of Google Maps.
  - SATELLITE displays photographic tiles.
  - HYBRID displays a mix of photographic tiles and a tile layer for prominent features (roads, city names).
  - TERRAIN displays physical relief tiles for displaying elevation and water features (mountains, rivers, etc.).

HTML5 STORAGE

HTML5 Storage

- It's a way for web pages to store named ***key/value*** pairs locally, within the client web browser.

- Like the data stored in cookies, this data persists even after you navigate away from the website, close your browser tab, exit your browser, or what have you.

- You store data based on a named key, and then you can retrieve that data with the same key.

mos.bayomi@gmail.com

HTML5 Storage

- The named key is a string.
- The data can be any type supported by JavaScript, including strings, Booleans, integers, or floats.
- However, the data is actually stored as a string.
- There are two types of Storage:
  - `sessionStorage` and `localStorage`.
- Both of them are in the **`window`** object

mos.bayomi@gmail.com

Setting Values

- Setting a value can easily be done in a single statement:

```
window.sessionStorage.setItem('myFirstKey', 'myFirstValue');
```

- The function we are calling is setItem, which takes a key string and a value string.

- This particular call will set into the session storage the string `myFirstValue`, which can later be retrieved by the key `myFirstKey`.

mos.bayomi@gmail.com

Retrieving Values

- To retrieve the value, mak a call to the `getItem` function.

```
alert(window.sessionStorage.getItem('myFirstKey'));
```

- We can write:

```
window.sessionStorage.myFirstKey = 'myFirstValue';

alert(window.sessionStorage.myFirstKey);
```


mos.bayomi@gmail.com

Local Versus Session Storage

- Sometimes, an application needs values that persist beyond the life of a single tab or window or need to be shared across multiple views.
- In these cases, it is more appropriate to use a different HTML5 Web Storage implementation: `localStorage`.

- The primary behavioral differences are how long the values persist and how they are shared

mos.bayomi@gmail.com

Local Versus Session Storage(cont'd)

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.

mos.bayomi@gmail.com

Storage methods and attributes

- The **`length`** attribute specifies how many key-value pairs are currently stored in the storage object.
- Remember that storage objects are specific to their origin, so that implies that the items (and length) of the storage object only reflect the items stored for the current origin.

- The **`key(index)`** function allows retrieval of a given key.
- **`getItem(key)`** function is one way to retrieve the value based on a given key.
- the value `null` will be returned if the key does not exist in storage.

mos.bayomi@gmail.com

Storage methods and attributes

- `setItem(key, value)` function will put a value into storage under the specified key name, or replace an existing value if one already exists under that key name.
- Note that it is possible to receive an error when setting an item value.

- if the user has storage turned off for that site, or if the storage is already filled to its maximum amount, a `QUOTA_EXCEEDED_ERR` error will be thrown during the attempt.

mos.bayomi@gmail.com

Storage methods and attributes

- `removeItem(key)` function : If a value is currently in storage under the specified key, this call will remove it. If no item was stored under that key, no action is taken.

mos.bayomi@gmail.com

HTML5 WEB WORKERS

Threads in JS

mos.bayomi@gmail.com

Using the HTML5 Web Workers API

- HTML5 Web Workers provide background processing capabilities to web applications and typically run on separate threads
- So that JavaScript applications using HTML5 Web Workers can take advantage of multicore CPUs
- Separating long-running tasks into HTML5 Web Workers also avoids the slow-script warnings

mos.bayomi@gmail.com

Creating HTML5 Web Workers

- Web Workers are initialized with the URL of a JavaScript file, which contains the code the worker will execute.
- This code sets event listeners and communicates with the script that spawned it.
- The URL for the JavaScript file can be a relative or absolute URL with the same origin.

```
worker = new Worker("myWorker.js");
```


mos.bayomi@gmail.com

Communicating with HTML5 Web Workers

- Once the Web Worker is founded, you can use the `postMessage` API to send data to and from Web Workers.

- Depending on your browser/version, `postMessage()` can accept either a string or JSON object as its single argument, ***but not functions or objects***
- The latest versions of the modern browsers support passing a JSON object.,

```
worker.postMessage("a message from parent page");
```


mos.bayomi@gmail.com

Communicating with HTML5 Web Workers

- When `postMessage()` is called from the main page, our worker handles that message by defining an `onmessage` handler for the message event.
- The message payload (in this case 'a message from parent page ') is accessible in `event.data`

myWorkers.js

```
onmessage = function myFunction(event)
        {
         event.data;
        ……
        postMessage('a message to the parent page');
        }
```

a message from parent page

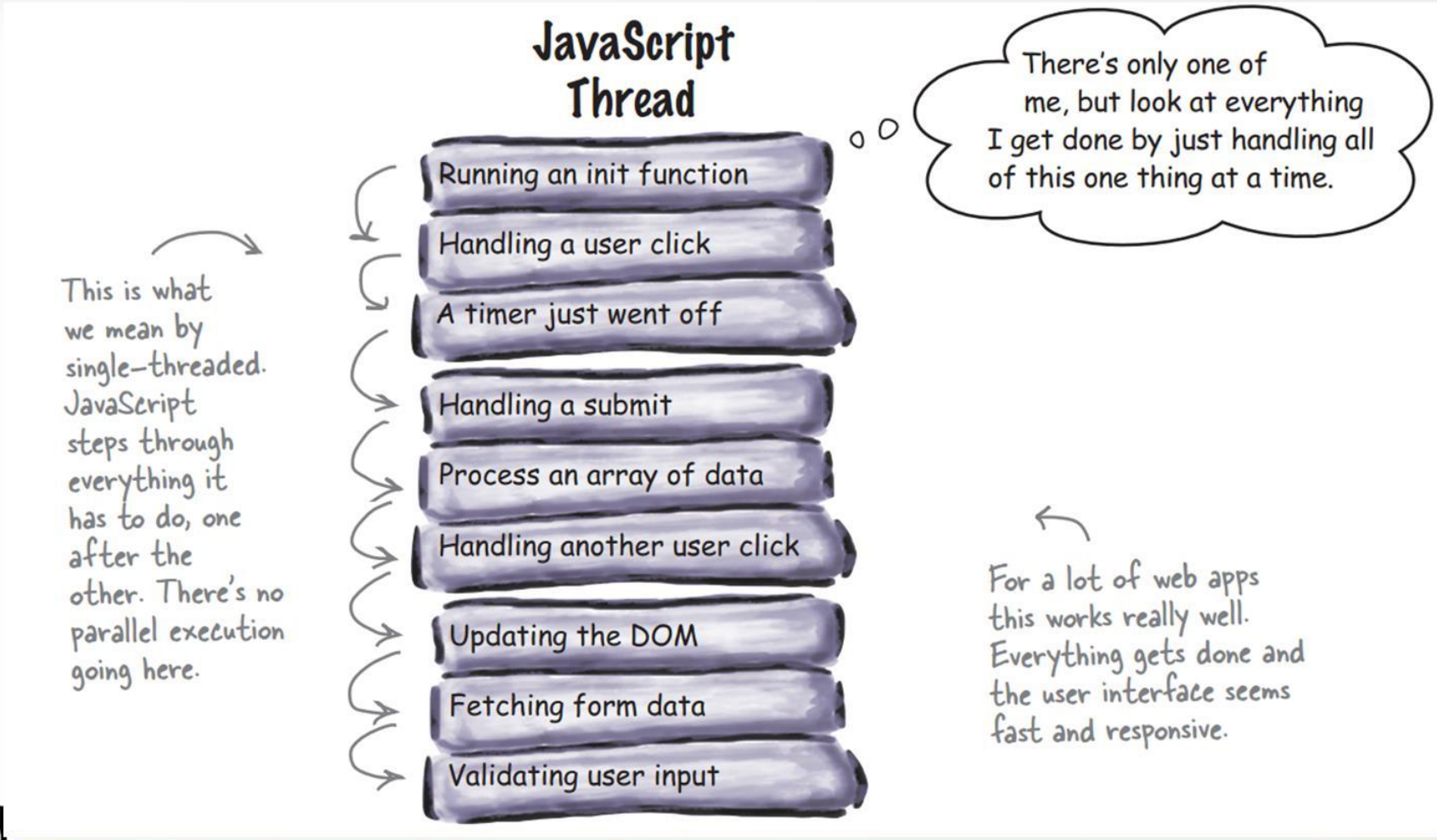
Handling Errors

- As with any JavaScript logic, you'll want to handle any errors that are thrown in your web workers.
- If an error occurs while a worker is executing, the an `ErrorEvent` is fired.
- The event contains three useful properties for figuring out what went wrong:
  - **filename** - the name of the worker script that caused the error
  - **lineno** - the line number where the error occurred
  - **message** - a meaningful description of the error.

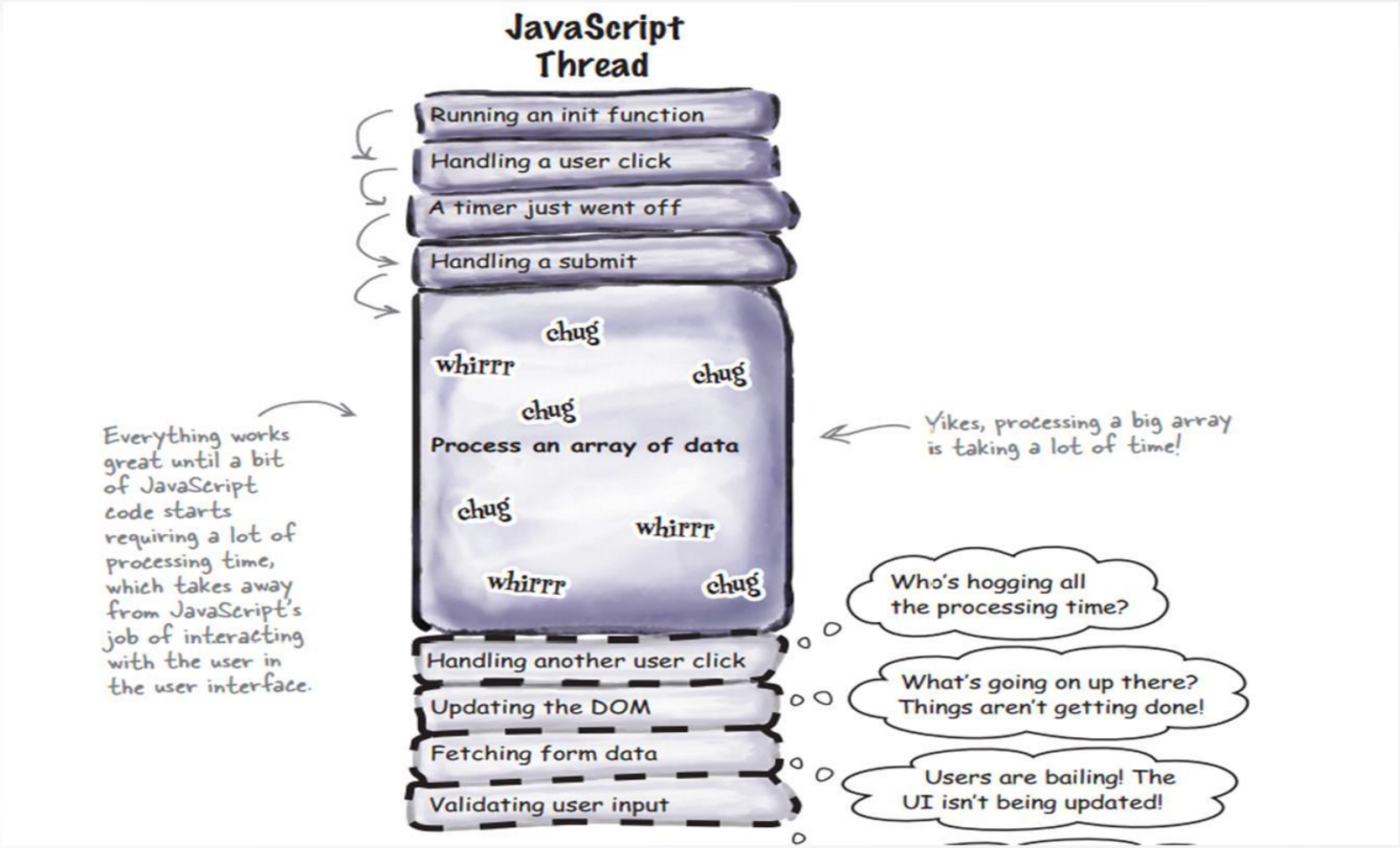
```
Worker.onerror = function(e)
{.....}
```


mos.bayomi@gmail.com

How JavaScript spends its time



When single-threaded goes BAD



Adding another thread of control to help