HTML5 Offline Data

INF5750/9750 - Lecture 6 (Part II)

What is offline?

• The Web and Online are considered to be synonymous, then what is HTML offline?

• HTML content distributed over CDs/DVDs, “always offline”

• Apps can be online-offline and sync when online

• Certain information is kept offline and not synced. This is generally non-critical information like user searches etc.

• Two specific solutions to doing offline data
  - Application Cache (Logic offline)
  - Offline Storage (Data offline)

Application Cache

- The core to HTML5 Application Cache (AppCache) is cache manifest file, i.e. a text file that lists the resources the browser should cache for offline access

```
<html manifest="example.appcache"> ... </html>
OR
<html manifest="http://www.example.com/example.mf">
```

mimeType should be text/cache-manifest and follows same-origin policy

- Structure of a manifest file:

```
CACHE MANIFEST # 2010-06-18:v2
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Resources that require the user to be online.
NETWORK:
login.php
/myapi
http://api.twitter.com
```

Updating the cache

- *window.applicationCache* object is your programmatic access AppCache

```
var appCache = window.applicationCache;
switch (appCache.status) {
    case appCache.UNCACHED: // UNCACHED == 0
        return 'UNCACHED';
        break;
    case appCache.IDLE: // IDLE == 1
        return 'IDLE';
        break;
....
appCache.update();
if (appCache.status == window.applicationCache.UPDATEREADY) {
    appCache.swapCache(); // The fetch was successful, swap in the new cache.
}
```

- You can also use listeners to check the status of cache

```
// Fired after the first cache of the manifest.
appCache.addEventListener('cached', handleCacheEvent, false);

// An update was found. The browser is fetching resources.
appCache.addEventListener('downloading', handleCacheEvent, false);

// Fired for each resource listed in the manifest as it is being fetched.
appCache.addEventListener('progress', handleCacheEvent, false);

// Fired when the manifest resources have been newly redownloaded.
appCache.addEventListener('updateready', handleCacheEvent, false);
```

Offline storage options

- APIs for offline storage work in “*same origin policy*”:
  - Web Storage API (localStorage)
  - WebSQL
  - IndexedDB
  - File Storage

Web Storage

- Simplistic structure of key-value pairs like JSON

```
// Saving a full JSON object using Web storage
localStorage[“student"] = JSON.stringify(StudentObject)

// Retrieving the saved StudentObject
var student= JSON.parse(localStorage[“student"]).
```

Strengths of Web Storage	Weakness of Web Storage
1. Supported on all modern browsers, as well as on iOS and Android, for several years (IE since version 8). 2. Simple API signature. 3. Simple call flow, being a synchronous API. 4. Semantic events available to keep other tabs/windows in sync.	1. Poor performance for large/complex data, when using the synchronous API (which is the most widely supported mode). 2. Poor performance when searching large/complex data, due to lack of indexing. (Search operations have to manually iterate through all items.) 3. Poor performance when storing and retrieving large/complex data structures, because it's necessary to manually serialize and de-serialize to/from string values. The major browser implementations only support string values (even though the spec says otherwise). 4. Need to ensure data consistency and integrity, since data is effectively unstructured.

src: html5rocks.com

Web SQL (deprecated)

- In-browser implementation of SQL
- Most implementations in browsers are using SQLite

```
openDatabase('documents', '1.0', 'Local document storage', 5*1024*1024, function (db) {
    db.changeVersion('', '1.0', function (t) {
        t.executeSql('CREATE TABLE docids (id, name)');
    }, error);
});
```

- Overhead for converting to SQL statements from JavaScript
- Not object oriented in its application
- Few browsers supported and has been deprecated

Strengths of Web SQL Database	Weakness of Web SQL Database
1. Supported on major mobile browsers (Android Browser, Mobile Safari, Opera Mobile)	1. Deprecated. Will not be supported on IE or Firefox, and will probably be phased out from the other browsers at some stage.
2. Good performance generally, being an asynchronous API. Database interaction won't lock up the user interface. (Synchronous API is also available for WebWorkers.)	2. Steep learning curve, requiring knowledge of relational databases and SQL.
3. Good search performance, since data can be indexed according to search keys.	3. Suffers from object-relational impedence mismatch .
4. Robust, since it supports a transactional database model .	4. Diminishes agility, as database schema must be defined upfront, with all records in a table matching the same structure.
5. Easier to maintain integrity of data, due to rigid data structure.	

IndexedDB

- A balance between Web Storage and WebSQL
- Transactions available and search is indexed for speed
- IndexedDB databases store key-value pairs
- The IndexedDB API is mostly asynchronous
- IndexedDB uses DOM events to notify you when results are available
- IndexedDB is object-oriented
- IndexedDB does not use Structured Query Language (SQL)

Using IndexedDB

- The basic pattern that IndexedDB encourages is the following:
  - Open a database and start a transaction.
  - Create an object store.
  - Make a request for dB operation, like adding or retrieving data.
  - Wait for the operation to complete by listening to DOM event.
  - Do something with results (which is available on request object).

```
// Start by checking if browser supports the feature
if (!window.indexedDB) { window.alert("Your browser doesn't support IndexedDB"); }

// Let us open our database and check for error or success in opening the DB
var request = window.indexedDB.open("MyTestDatabase", 3);
request.onerror = function(event) {
    // Do something with request.errorCode! };
request.onsuccess = function(event) {
    // Do something with request.result! };

// Create an objectStore for this database
var objectStore = db.createObjectStore("name", { keyPath: "myKey" });
```

Storing and Retrieving data

```
// Data
var studentLocation = [
    { name: "Jane Fonda", age: 22, longitude: "23.352", latitude: “42.954"},
    { name : "Julian Assange", age: 25, longitude: "24.662", latitude: “42.991" }
 ];

const dbName = “loc";
var request = indexedDB.open(dbName, 2); //name = loc and version = 2

request.onupgradeneeded = function(event) {
    var db = event.target.result; // Create an objectStore to hold students.

    //Start with "name" as our key path because our model ensures it to be unique.
    var objectStore = db.createObjectStore(“students", { keyPath: “name" });
    // Create an index to search students by longitude and latitude.
    objectStore.createIndex("longitude", "longitude", { unique: false });
    objectStore.createIndex("latitude", " latitude ", { unique: false });

    // Store values in the newly created objectStore.
    for (var i in customerData) {
        objectStore.add(customerData[i]);
    }
};

//Adding a transaction and deleting data
var transaction = db.transaction([“loc"], "readwrite“).objectStore (“students") .delete(“Jane Fonda");
request.onsuccess = function(event) {  alert(‘Deleted successfully’); };

//Adding a transaction and retrieving data
db.transaction(“loc").objectStore(“students").get(“Julian Assange").onsuccess = function(event) { alert(“Longitude = " +
event.target.result.longitude); };
```

Using a cursor

- Using get() requires that you know which key to retrieve

```
// Get all students in store
var objectStore = db.transaction(“loc").objectStore(“students");

//Adding a transaction and retrieving data
objectStore.openCursor().onsuccess = function(event) {
    var cursor = event.target.result;
     if (cursor) {
         alert(“Student: " + cursor.key + " is " + cursor.value.name);
         cursor.continue();
     } else {
         alert("No more entries!");
     }
}
```

- Searching directly on an index

```
var index = objectStore.index(“latitude");
index.get("23.352").onsuccess = function(event) {
      alert(“At latitude 23.352, student is " + event.target.result.name);
};
```

- When the browser shuts down, any pending IndexedDB transactions are (silently) aborted -- they will not complete, and they will not trigger the error handler

Resources

- A summary of Offline Storage
  - http://www.html5rocks.com/en/features/storage
- IndexedDB at MDN
  - https://developer.mozilla.org/en-US/docs/IndexedDB/Using_IndexedDB
- Migrating from WebSQL to IndexedDB
  - http://www.html5rocks.com/en/tutorials/webdatabase/websql-indexeddb/