Mobile dev 02

19/10/2020

Paolo Cargnin

Struttura del corso

1° lezione	2° lezione	3° lezione	4° lezione
Progressive Web apps	Progressive Web apps		

Sommario lezione

- Metodi di cache
- Indexed Db
- Rendiamo una app in angular Proegressive

Strategia di cache

- Cachizzare a richiesta
- Dare una pagina di fallback per l’offline

Salvare nuove cache lato client (feed.js)

```
function onSaveButtonClicked(event) {
  console.log('clicked');
  if ('caches' in window) {
    caches.open('user-requested')
      .then(function(cache) {
        cache.add('https://httpbin.org/get');
        cache.add('/src/images/sf-boat.jpg');
      });
  }
}
```

(aggiorniamo il serviceWorker!)

Offline fallback page

1. Cachizziamo il file offline.html
2. Gestiamo il .catch di fetch:

```
.catch(function(err) {
  return caches.open(CACHE_STATIC_NAME)
    .then(function(cache) {
      return cache.match('/offline.html');
    });
});
```

Offline fallback page

1. Check del file:

```
if (event.request.url.includes('/help')){
  return cache.match('/offline.html');
}
```

Offline fallback page

1. Muuuch better:

```
if (event.request.headers.get('accept').includes('text/html')){
   return cache.match('/offline.html');
}
```

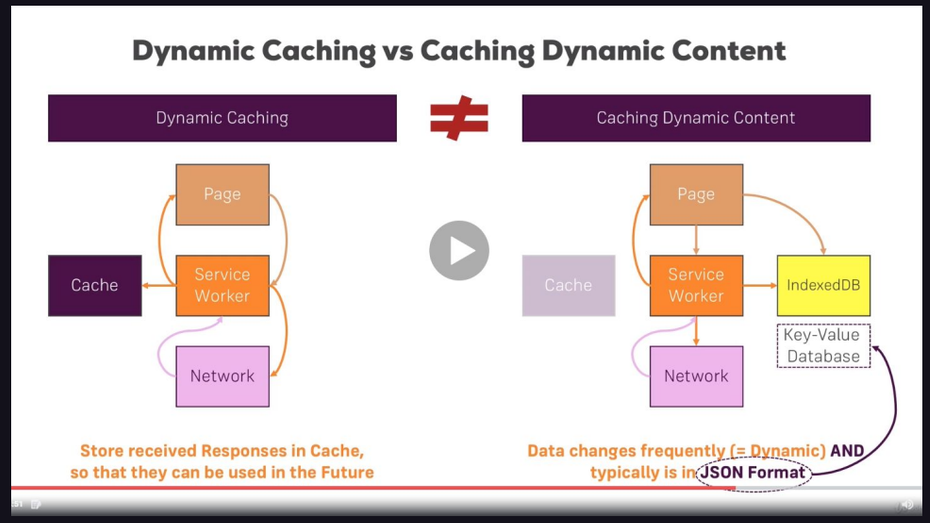
Tipologie di cache

1. Cache con network fallback
2. Cache only
3. Network only
4. Network with cache fallback
5. Cache then network
6. Trim Cache

Usare la cache sui dati

IndexedDB

Dynamic caching vs caching dynamic content



Cos’è indexedDB

- Un key-value database transazionabile nel browser
- Può gestire un grande numero di informazioni, file compresi
- Si accede in maniera asincrona

Can i use it?

https://caniuse.com/?search=indexedDB

Ci serve comunque, prima...
Un db lato server!

- inizializziamo firebase
- Creiamo il real-time database (accesso libero)

carichiamo una dipendenza nel nostro serviceWorker

importScripts(‘/src/js/idb.js’)

Aggiungiamolo anche alla static Cache

Salviamo i post in indexedDB!

```
var dbPromise = idb.open('posts-store', 1, function (db) {
  if (!db.objectStoreNames.contains('posts')) {
    db.createObjectStore('posts', {keyPath: 'id'});
  }
});


...
event.respondWith(fetch(event.request)
      .then(function (res) {
        var clonedRes = res.clone();
        clonedRes.json()
          .then(function(data) {
            for (var key in data) {
              dbPromise
                .then(function(db) {
                  var tx = db.transaction('posts', 'readwrite');
                  var store = tx.objectStore('posts');
                  store.put(data[key]);
                  return tx.complete;
                });
            }
          });
        return res;
      })
    );
...
```

Creiamo un set di metodi

Creiamo utility.js

Spostiamo la logica di indexDb in quel file:

```
var dbPromise = idb.open('posts-store', 1, function (db) {
  if (!db.objectStoreNames.contains('posts')) {
    db.createObjectStore('posts', {keyPath: 'id'});
  }
});

function writeData(st, data) {
  return dbPromise
    .then(function(db) {
      var tx = db.transaction(st, 'readwrite');
      var store = tx.objectStore(st);
      store.put(data);
      return tx.complete;
    });
}
```

```
importScripts('/src/js/utility.js');


.then(function (data) {
      for (var key in data) {
        writeData('posts', data[key])
      }
});
```

Creiamo un set di metodi

Aggiorniamo il Service Worker

```
importScripts('/src/js/utility.js');

...
.then(function (data) {
      for (var key in data) {
        writeData('posts', data[key])
      }
});
...
```

Leggiamo i dati!

aggiungiamo utility.js nel nostro index.html

aggiungiamo la funzione readAllData()

```
function readAllData(st) {
  return dbPromise
    .then(function(db) {
      var tx = db.transaction(st, 'readonly');
      var store = tx.objectStore(st);
      return store.getAll();
    });
}
```

Leggiamo i dati!

Aggiorniamo il file... feed.js!

```
if ('indexedDB' in window) {
  readAllData('posts')
    .then(function(data) {
      if (!networkDataReceived) {
        console.log('From cache', data);
        updateUI(data);
      }
    });
}
```

Alcuni problemi che potrebbero sorgere:

1. **Il dato viene modificato?**
No problema. Vediamo perché

2. **Il dato viene cancellato?**
In Offline-mode potremmo avere dei problemi...

Dobbiamo cancellare i dati non appena sappiamo che non esistono più

Aggiungiamo la funzione clearAllData()

```
function clearAllData(st) {
  return dbPromise
    .then(function(db) {
      var tx = db.transaction(st, 'readwrite');
      var store = tx.objectStore(st);
      store.clear();
      return tx.complete;
    });
}
```

Aggiorniamo il serviceWorker

```
event.respondWith(fetch(event.request)
  .then(function (res) {
    var clonedRes = res.clone();
    clearAllData('posts')
      .then(function () {
        return clonedRes.json();
      })
      .then(function (data) {
        for (var key in data) {
          writeData('posts', data[key])
        }
      });
    return res;
  })
);
```

Come cancellare i singoli elementi?

```
function deleteItemFromData(st, id) {
  dbPromise
    .then(function(db) {
      var tx = db.transaction(st, 'readwrite');
      var store = tx.objectStore(st);
      store.delete(id);
      return tx.complete;
    })
    .then(function() {
      console.log('Item deleted!');
    });
}
```

Angular progressive web apps!

https://angular.io/guide/service-worker-getting-started