عنوان: **WebStorage: قسمت دوم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۴/۰۹ ۱:۰
آدرس: www.dotnettips.info
گروه‌ها: Cookie, IndexedDB, Webstorage, HTML 5

در این مقاله قصد داریم نحوه‌ی کدنویسی webstorage را با کتابخانه‌هایی که در مقاله قبل معرفی کردیم بررسی کنیم.
ابتدا روش ذخیره سازی و بازیابی متداول آن را بررسی میکنیم که تنها توسط دو تابع صورت می‌گیرد. مطلب زیر برگرفته از w3Schools است:
دسترسی به شیء webstorage به صورت زیر امکان پذیر است:

```
window.localStorage
window.sessionStorage
```

ولی بهتر است قبل از ذخیره و بازیابی، از پشتیبانی مرورگر از webstorage اطمینان حاصل نمایید:

```
if(typeof(Storage) !== "undefined") {
    // Code for localStorage/sessionStorage.
} else {
    // Sorry! No Web Storage support..
}
```

برای ذخیره سازی و سپس خواندن به شکل زیر عمل می‌کنیم:

```
localStorage.setItem("lastname", "Smith");

//========================
localStorage.getItem("lastname");
```

خواندن می‌تواند حتی به شکل زیر هم صورت بگیرد:

```
var a=localStorage.lastname;
```

استفاده از store.js برای مرورگرهایی که از webstorage پشتیبانی نمی‌کنند به شکل زیر است:

```
//ذخیره مقدار
store.set('username', 'marcus')

//بازیابی مقدار
store.get('username')

//حذف مقدار
store.remove('username')

//حذف تمامی مقادیر ذخیره شده
store.clear()

//ذخیره ساختار
store.set('user', { name: 'marcus', likes: 'javascript' })

//بازیابی ساختار به شکل قبلی
var user = store.get('user')
alert(user.name + ' likes ' + user.likes)

//تغییر مستقیم مقدار قبلی
store.getAll().user.name == 'marcus'

//بازخوانی تمام مقادیر ذخیر شده توسط یک حلقه
store.forEach(function(key, val) {
    console.log(key, '==', val)
})
```

همچنین بهتر هست از یک فلگ برای بررسی فعال بودن storage استفاده نمایید. به این دلیل که گاهی کاربرها از پنجره‌های

private استفاده می‌کنند که ردگیری اطلاعات آن ممکن نیست و موجب خطا می‌شود.

```
<script src="store.min.js"></script>
<script>
    init()
    function init() {
        if (!store.enabled) {
            alert('Local storage is not supported by your browser. Please disable "Private Mode", or
upgrade to a modern browser.')
            return
        }
        var user = store.get('user')
        // ... and so on ...
    }
</script>
```

در صورتیکه بخشی از داده‌ها را توسط localstorage ذخیره نمایید و بخواهید از طریق storage به آن دسترسی داشته باشید، خروجی string خواهد بود؛ صرف نظر از اینکه شما عدد، شیء یا آرایه‌ای را ذخیره کرده‌اید.
در صورتیکه ساختار JSON را ذخیره کرده باشید، می‌توانید رشته برگردانده شده را با json.stringify و json.parse بازیابی و به روز رسانی کنید.
در حالت cross browser تهیه‌ی یک sessionStorage امکان پذیر نیست. ولی می‌توان به روش ذیل و تعیین یک زمان انقضاء آن را محدود کرد:

```
var storeWithExpiration = {
// دریافت کلید و مقدار و زمان انقضا به میلی ثانیه
    set: function(key, val, exp) {
//ایجاد زمان فعلی جهت ثبت تاریخ ایجاد
        store.set(key, { val:val, exp:exp, time:new Date().getTime() })
    },
    get: function(key) {
        var info = store.get(key)
//در صورتی که کلید داده شده مقداری نداشته باشد نال را بر میگردانیم
        if (!info) { return null }
//تاریخ فعلی را منهای تاریخ ثبت شده کرده و در صورتی که
//از مقدار میلی ثاینه بیشتر باشد یعنی منقضی شده و نال بر میگرداند
        if (new Date().getTime() - info.time > info.exp) { return null }
        return info.val
    }
}

// استفاده عملی از کد بالا
// استفاده از تایمر جهت نمایش واکشی داده‌ها قبل از نقضا و بعد از انقضا
storeWithExpiration.set('foo', 'bar', 1000)
setTimeout(function() { console.log(storeWithExpiration.get('foo')) }, 500) // -> "bar"
setTimeout(function() { console.log(storeWithExpiration.get('foo')) }, 1500) // -> null
```

مورد بعدی استفاده از سورس cross-storage است. اگر به یاد داشته باشید گفتیم یکی از احتمالاتی که برای ما ایجاد مشکل می‌کند، ساب دومین هاست که ممکن است دسترسی ما به یک webstorage را از ساب دومین دیگر از ما بگیرد.
این کتابخانه به دو جز تقسیم شده است یکی هاب Hub و دیگری Client .

ابتدا نیاز است که هاب را آماده سازی و با ارائه یک الگو از آدرس وب، مجوز عملیات را دریافت کنیم. در صورتیکه این مرحله به فراموشی سپرده شود، انجام هر نوع عمل روی دیتاها در نظر گرفته نخواهد شد.

```
CrossStorageHub.init([
  {origin: /\.example.com$/,           allow: ['get']},
  {origin: /:\/\/(www\.)?example.com$/, allow: ['get', 'set', 'del']}
]);
```

حرف $ در انتهای عبارت باعث مشود که دامنه‌ها با دقت بیشتری در Regex بررسی شوند و دامنه زیر را معتبر اعلام کند:

```
valid.example.com
```

ولی دامنه زیر را نامعتبر اعلام می‌کند:

```
invalid.example.com.malicious.com
```

همچنین می‌توانید تنظیماتی را جهت هدرهای CSP و CORS، نیز اعمال نمایید:

```
{
  'Access-Control-Allow-Origin':  '*',
  'Access-Control-Allow-Methods': 'GET,PUT,POST,DELETE',
  'Access-Control-Allow-Headers': 'X-Requested-With',
  'Content-Security-Policy':      "default-src 'unsafe-inline' *",
  'X-Content-Security-Policy':    "default-src 'unsafe-inline' *",
  'X-WebKit-CSP':                 "default-src 'unsafe-inline' *",
}
```

پس کار را بدین صورت آغاز می‌کنیم، یک فایل به نام hub.htm درست کنید و هاب را آماده سازید: **hub.htm**

```
<script type="text/javascript" src="~/Scripts/cross-storage/hub.js"></script>
<script>
 CrossStorageHub.init([
      {origin: /.*localhost:300\d$/, allow: ['get', 'set', 'del']}
    ]);
  </script>
```

کد بالا فقط درخواست‌های هاست لوکال را از پورتی که ابتدای آن با 300 آغاز می‌شود، پاسخ می‌دهد و مابقی درخواست‌ها را رد می‌کند. متدهای ایجاد، ویرایش و حذف برای این آدرس معتبر اعلام شده است.
در فایل دیگر که کلاینت شناخته می‌شود باید فایل hub معرفی شود تا تنظیمات هاب خوانده شود:

```
var storage = new CrossStorageClient('http://localhost:3000/example/hub.html');

var setKeys = function () {

return storage.set('key1', 'foo').then(function() {
return storage.set('key2', 'bar');
      });
};
```

در خط اول، فایل هاب معرفی شده و تنظیمات روی این صفحه اعمال می‌شود. سپس در خطوط بعدی داده‌ها ذخیره می‌شوند. از آنجا که با هر یکبار ذخیره، return صورت می‌گیرد و تنها اجازه‌ی ورود یک داده را داریم، برای حل این مشکل متد then پیاده سازی شده است. متغیر setKeys شامل یک آرایه از کلیدها خواهد بود.
نحوه‌ی ذخیره سازی بدین شکل هم طبق مستندات صحیح است:

```
storage.onConnect().then(function() {

  return storage.set('key', {foo: 'bar'});

}).then(function() {

  return storage.set('expiringKey', 'foobar', 10000);

});
```

در کد بالا ابتدا یک داده دائم ذخیره شده است و در کد بعد یک داده موقت که بعد از 10 ثانیه اعتبار خود را از دست می‌دهد.
برای خواندن داده‌های ذخیره شده به نحوه زیر عمل می‌کنیم:

```
storage.onConnect().then(function() {
  return storage.get('key1');
}).then(function(res) {
  return storage.get('key1', 'key2', 'key3');
}).then(function(res) {
  // ...
});
```

کد بالا نحوه‌ی خواندن مقادیر را به شکل‌های مختلفی نشان میدهد و مقدار بازگشتی آن‌ها یک آرایه از مقادیر است؛ مگر اینکه تنها یک مقدار برگشت داده شود. مقدار بازگشتی در تابع بعدی به عنوان یک آرگومان در دسترس است. در صورتی که خطایی رخ دهد، قابلیت هندل آن نیز وجود دارد:

```
storage.onConnect()
    .then(function() {
      return storage.get('key1', 'key2');
    })

.then(function(res) {
      console.log(res); // ['foo', 'bar']
    })['catch'](function(err) {
      console.log(err);
    });
```

برای باقی مسائل چون به دست آوردن لیست کلیدهای ذخیره شده، حذف کلیدهای مشخص شده، پاکسازی کامل داده‌ها و ... به مستندات رجوع کنید.

در اینجا جهت سازگاری با مرورگرهای قدیمی خط زیر را به صفحه اضافه کنید:

```
<script src="https://s3.amazonaws.com/es6-promises/promise-1.0.0.min.js"></script>
```

ذخیره‌ی اطلاعات به شکل یونیکد، فضایی دو برابر کدهای اسکی میبرد و با توجه به محدود بودن حجم webstorage به 5 مگابایت ممکن است با کمبود فضا مواجه شوید. در صورتیکه قصد فشرده سازی اطلاعات را دارید می‌توانید از کتابخانه lz-string استفاده کنید. ولی توجه به این نکته ضروری است که در صورت نیاز، عمل فشرده سازی را انجام دهید و همینطوری از آن استفاده نکنید.

IndexedDB API

آخرین موردی که بررسی می‌شود استفاده از IndexedDB API است که با استفاده از آن میتوان با webstorage همانند یک دیتابیس رفتار کرد و به سمت آن کوئری ارسال کرد.

```
var request = indexedDB.open("library");

request.onupgradeneeded = function() {
  // The database did not previously exist, so create object stores and indexes.
  var db = request.result;
  var store = db.createObjectStore("books", {keyPath: "isbn"});
  var titleIndex = store.createIndex("by_title", "title", {unique: true});
  var authorIndex = store.createIndex("by_author", "author");

  // Populate with initial data.
  store.put({title: "Quarry Memories", author: "Fred", isbn: 123456});
  store.put({title: "Water Buffaloes", author: "Fred", isbn: 234567});
  store.put({title: "Bedrock Nights", author: "Barney", isbn: 345678});
};

request.onsuccess = function() {
  db = request.result;
};
```

کد بالا ابتدا به دیتابیس library متصل می‌شود و اگر وجود نداشته باشد، آن را می‌سازد. رویداد onupgradeneeded برای اولین بار اجرا شده و در آن می‌توانید به ایجاد جداول و اضافه کردن داده‌های اولیه بپردازید؛ یا اینکه از آن جهت به ارتقاء ورژن دیتابیس استفاده کنید. خصوصیت result، دیتابیس باز شده یا ایجاد شده را باز می‌گرداند. در خط بعدی جدولی با کلید کد ISBN کتاب تعریف شده است. در ادامه هم دو ستون اندیس شده برای عنوان کتاب و نویسنده معرفی شده است که عنوان کتاب را یکتا و بدون تکرار در نظر گرفته است. سپس در جدولی که متغیر store به آن اشاره می‌کند، با استفاده از متد put، رکوردها داخل آن درج می‌شوند. در صورتیکه کار با موفقیت انجام شود رویداد onSuccess فراخوانی می‌گردد.

برای انجام عملیات خواندن و نوشتن باید از تراکنش‌ها استفاده کرد:

```
var tx = db.transaction("books", "readwrite");
var store = tx.objectStore("books");

store.put({title: "Quarry Memories", author: "Fred", isbn: 123456});
store.put({title: "Water Buffaloes", author: "Fred", isbn: 234567});
```

```
store.put({title: "Bedrock Nights", author: "Barney", isbn: 345678});

tx.oncomplete = function() {
  // All requests have succeeded and the transaction has committed.
};
```

در خط اول ابتدا یک خط تراکنشی بین ما و جدول books با مجوز خواندن و نوشتن باز می‌شود و در خط بعدی جدول books را در اختیار می‌گیریم و همانند کد قبلی به درج داده‌ها می‌پردازیم. در صورتیکه عملیات با موفقیت به اتمام برسد، متغیر تراکنش رویدادی به نام oncomplete فراخوانی می‌گردد. در صورتیکه قصد دارید تنها مجوز خواندن داشته باشید، عبارت readonly را به کار ببرید.

```
var tx = db.transaction("books", "readonly");
var store = tx.objectStore("books");
var index = store.index("by_author");

var request = index.openCursor(IDBKeyRange.only("Fred"));
request.onsuccess = function() {
  var cursor = request.result;
  if (cursor) {
    // Called for each matching record.
    report(cursor.value.isbn, cursor.value.title, cursor.value.author);
    cursor.continue();
  } else {
    // No more matching records.
    report(null);
  }
};
```

در دو خط اول مثل قبل، تراکنش را دریافت می‌کنیم و از آنجا که می‌خواهیم داده را بر اساس نام نویسنده واکشی کنیم، ستون اندیس شده نام نویسنده را دریافت کرده و با استفاده از متد opencursor درخواست خود را مبنی بر واکشی رکوردهایی که نام نویسنده **fred** است، ارسال میداریم. در صورتیکه عملیات با موفقیت انجام گردد و خطایی دریافت نکنیم رویداد onsuccess فراخوانی می‌گردد. در این رویداد با دو حالت برخورد خواهیم داشت؛ یا داده‌ها یافت می‌شوند و رکوردها برگشت داده می‌شوند یا هیچ رکوردی یافت نشده و مقدار نال برگشت خواهد خورد. با استفاده از cursor.continue می‌توان داده‌ها را به ترتیب واکشی کرده و مقادیر رکورد را با استفاده خصوصیت value به سمت تابع report ارسال کرد.

کدهای بالا همه در مستندات معرفی شده وجود دارند و ما پیشتر توضیح ابتدایی در مورد آن دادیم و برای کسب اطلاعات بیشتر می‌توانید به همان مستندات معرفی شده رجوع کنید. برای idexedDB هم می‌توانید از این منابع + + + استفاده کنید که خود w3c منبع فوق العاده‌تری است.