# HTML5

https://www.tutorialspoint.com/html5/html5_overview.htm

A HTML5 a HTML szabvány következő nagy változata, amely felváltja a HTML 4.01, XHTML 1.0 és XHTML 1.1 verziókat. A HTML5 a tartalom világhálón történő strukturálásának és bemutatásának szabványa.

A HTML5 a World Wide Web Consortium (W3C) és a Web Hypertext Application Technology Working Group (WHATWG) közötti együttműködés.

Az új szabvány olyan funkciókat tartalmaz, mint a videolejátszás és a drag-and-drop, amelyek korábban olyan külső böngésző plug-inektől függtek, mint az Adobe Flash, a Microsoft Silverlight és a Google Gears.

Böngésző támogatás

Az Apple Safari, a Google Chrome, a Mozilla Firefox és az Opera legújabb verziói sok HTML5 funkciót támogatnak, és az Internet Explorer 9.0 is támogat bizonyos HTML5 funkciókat.

Az iPhone, iPad és Android telefonokra előre telepített mobil webböngészők kiválóan támogatják a HTML5-t.

Új funkciók

A HTML5 számos új elemet és attribútumot mutat be, amelyek segíthetnek a modern weboldalak felépítésében. Itt található a HTML5-ben bevezetett néhány legkiemelkedőbb szolgáltatás.

- Új szemantikai elemek - Ezek olyanok, mint a <header>, <footer>, and <section>.
- Forms 2.0 - Fejlesztések a HTML webes űrlapokban, ahol új attribútumokat vezettek be az <input> címkéhez.
- Persistent Local Storage- Elérés harmadik féltől származó pluginok igénybevétele nélkül.
- WebSocket - Új generációs kétirányú kommunikációs technológia webalkalmazásokhoz.
- Kiszolgáló által küldött események - A HTML5 olyan eseményeket mutat be, amelyek a webkiszolgálóról a webböngészőbe áramlanak, és ezeket Server-Sent Events (SSE) néven hívják.
- Canvas - Ez egy kétdimenziós rajzfelületet támogat, amelyet JavaScript-sel programozhat.
- Hang és videó - Hangot vagy videót beágyazhat a weboldalaira harmadik féltől származó beépülő modulok igénybevétele nélkül.
- Földrajzi helymeghatározás - Most a látogatók dönthetnek úgy, hogy megosztják fizikai helyüket az Ön webalkalmazásával.
- Microdata - Ez lehetővé teszi a HTML5-n kívüli saját szókincsek létrehozását, és a weboldalak egyéni szemantikával történő kibővítését.
- Drag and drop- Húzza az elemeket az egyik helyről a másik helyre ugyanazon a weboldalon.

Visszafelé kompatibilitás

A HTML5-et a lehető legnagyobb mértékben úgy tervezték, hogy visszafelé kompatibilis legyen a meglévő böngészőkkel. Új funkciói a meglévő funkciókra épültek, és lehetővé teszik, hogy tartalékot nyújtson a régebbi böngészők számára.

HTML5 dokumentum

A jobb tagolás érdekében a következő címkéket vezették be –

- section - Ez a címke egy általános dokumentumot vagy alkalmazásszakaszt jelöl. A h1-h6-tal együtt használható a dokumentum szerkezetének megjelölésére.
- article - Ez a címke egy dokumentum független tartalmi részét képviseli, például blogbejegyzés vagy újságcikk.
- aside - Ez a címke olyan tartalomrészletet jelöl, amely csak kissé kapcsolódik az oldal többi részéhez.
- header - Ez a címke egy szakasz fejlécét jelöli.
- footer - Ez a címke egy szakasz láblécét jelöli, és tartalmazhat információkat a szerzőről, szerzői jogi információkat és így tovább.
- nav - Ez a címke a dokumentum navigációra szánt szakaszát jelenti.
- dialog - Ez a címke használható egy beszélgetés megjelölésére.
- figure - Ezzel a címkével felirat társítható néhány beágyazott tartalommal, például grafikával vagy videóval.

```
<!DOCTYPE html>

<html>
   <head>
      <meta charset = "utf-8">
      <title>...</title>
   </head>

   <body>
      <header>...</header>
      <nav>...</nav>

      <article>
         <section>
            ...
         </section>
      </article>
      <aside>...</aside>

      <footer>...</footer>
   </body>
</html>
```

HTML5 – SVG

Az SVG jelentése Scalable Vector Graphics, és ez egy olyan nyelv, amely leírja a 2D-grafikákat és grafikus alkalmazásokat XML-ben, majd az XML-t egy SVG-megjelenítő rendereli.

Az SVG leginkább vektor típusú diagramokhoz használható, például kördiagramokhoz, kétdimenziós grafikonokhoz X, Y koordinátarendszerben stb.

Például kör megrajzolása:

```
<!DOCTYPE html>

<html>
   <head>

      <style>
         #svgelem {
            position: relative;
            left: 50%;
            -webkit-transform: translateX(-20%);
            -ms-transform: translateX(-20%);
            transform: translateX(-20%);
         }
      </style>
      <title>SVG</title>
      <meta charset = "utf-8" />
   </head>

   <body>
      <h2 align = "center">HTML5 SVG Circle</h2>

      <svg id = "svgelem" height = "200" xmlns = "http://www.w3.org/2000/svg">
         <circle id = "redcircle" cx = "50" cy = "50" r = "50" fill = "red" />
      </svg>
   </body>
</html>
```

HTML5 – MathML

A HTML5 HTML szintaxisa lehetővé teszi a MathML elemek használatát a dokumentum belsejében <math> ... </math> címkékkel.

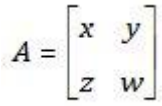
A legtöbb webböngésző képes megjeleníteni MathML címkéket. Ha böngészője nem támogatja a MathML-t, akkor azt javasoljuk, hogy használja a Firefox legújabb verzióját.

Mátrix bemutató példák

Tekintsük a következő példát, amelyet egy egyszerű 2x2-es mátrix képviseletére használnánk –

```
<!doctype html>

<html>
   <head>
      <meta charset = "UTF-8">
      <title>MathML Examples</title>
   </head>

   <body>
      <math xmlns = "http://www.w3.org/1998/Math/MathML">

         <mrow>
            <mi>A</mi>
            <mo>=</mo>
```

```
            <mfenced open = "[" close="]">

               <mtable>
                  <mtr>
                     <mtd><mi>x</mi></mtd>
                     <mtd><mi>y</mi></mtd>
                  </mtr>

                  <mtr>
                     <mtd><mi>z</mi></mtd>
                     <mtd><mi>w</mi></mtd>
                  </mtr>
               </mtable>

            </mfenced>
         </mrow>
      </math>

   </body>
</html>
```



HTML5 - Web Storage

A HTML5 két, a HTTP session cookie-hoz hasonló mechanizmust vezet be a strukturált adatok kliensoldali tárolására és a következő hátrányok leküzdésére.

- A cookie-k minden HTTP-kéréshez tartoznak, ezáltal ugyanazok az adatok továbbításával lelassítják a webalkalmazást.
- A sütik minden HTTP-kéréshez tartoznak, így titkosítatlan adatokat küldenek az interneten keresztül.
- A sütik körülbelül 4 KB adatra korlátozódnak. Nem elegendő a szükséges adatok tárolásához.
- A két tároló, a session storage and local storage, és különböző helyzetek kezelésére használják őket.
- Nagyjából minden böngésző legújabb verziói támogatják a HTML5 Storage-ot, beleértve az Internet Explorert is.

Session Storage

A Session Storage-t olyan esetekre tervezték, amikor a felhasználó egyetlen tranzakciót hajt végre, de több tranzakciót is végezhet különböző ablakokban egyszerre.

Például, ha a felhasználó repülőjegyeket vásárol két különböző ablakban, ugyanazon a webhelyen. Ha a webhely cookie-kat használt annak nyomon követésére, hogy a felhasználó melyik jegyet vásárolja, akkor amikor a felhasználó mindkét ablakban egyik oldalról a másikra kattintott, a jelenleg vásárolt jegy "szivárog" az egyik ablakból a másikba, ami a felhasználót esetleg vásárolj két jegyet ugyanarra a járatra, anélkül, hogy észrevennéd.

A HTML5 bemutatja a sessionStorage attribútumot, amelyet a helyek használnának az adatok hozzáadásához a Session Storage-hez, és hozzáférhető lesz ugyanarról a webhelyről az adott ablakban megnyílt oldalakhoz, azaz a munkamenethez (session), és amint bezárja az ablakot, a munkamenet (session) elveszne.

Az alábbiakban látható az a kód, amely beállítja a munkamenet (session) változót, és hozzáfér a változóhoz.

```
<!DOCTYPE HTML>

<html>
   <body>
      <script type = "text/javascript">

         if( sessionStorage.hits ) {
            sessionStorage.hits = Number(sessionStorage.hits) +1;
         } else {
            sessionStorage.hits = 1;
         }
         document.write("Total Hits :" + sessionStorage.hits );
      </script>

      <p>Refresh the page to increase number of hits.</p>
      <p>Close the window and open it again and check the result.</p>

   </body>
</html>
```

Local Storage

A Local Storage-t több ablakon átívelő, az aktuális munkameneten (Session) túl tartó tárolásra tervezték. Különösen a webalkalmazások megabájt felhasználói adatokat, például teljes felhasználó által készített dokumentumokat vagy egy felhasználó postaládáját kívánják tárolni az kliens oldalon teljesítmény okokból.

A sütik megint nem kezelik jól ezt az esetet, mert minden kéréssel továbbításra kerülnek.

```
<!DOCTYPE HTML>

<html>
   <body>
      <script type = "text/javascript">

         if( localStorage.hits ) {
            localStorage.hits = Number(localStorage.hits) +1;
         } else {
            localStorage.hits = 1;
         }
         document.write("Total Hits :" + localStorage.hits );
      </script>

      <p>Refresh the page to increase number of hits.</p>
      <p>Close the window and open it again and check the result.</p>

   </body>
</html>
```

Törölje a webtárhelyet (web storage)

Az érzékeny adatok helyi gépen történő tárolása veszélyes lehet, és biztonsági rést hagyhat.

A munkamenet-tárolási adatokat (Session Storage Data) a böngészők azonnal törlik, miután a munkamenet megszűnik.

A local storage beállítások törléséhez meg kell hívnia a localStorage.remove ('kulcs') parancsot; ahol a „kulcs” az eltávolítani kívánt érték kulcsa. Ha törölni szeretné az összes beállítást, akkor meg kell hívnia a localStorage.clear () metódust.

A következő a kód, amely törli a teljes local storage –t :

```
<!DOCTYPE HTML>

<html>
   <body>

      <script type = "text/javascript">
         localStorage.clear();

         // Reset number of hits.
         if( localStorage.hits ) {
            localStorage.hits = Number(localStorage.hits) +1;
         } else {
            localStorage.hits = 1;
         }
         document.write("Total Hits :" + localStorage.hits );

      </script>

      <p>Refreshing the page would not to increase hit counter.</p>
      <p>Close the window and open it again and check the result.</p>

   </body>
</html>
```

HTML5 – WebSockets

A WebSockets egy új generációs kétirányú kommunikációs technológia webalkalmazásokhoz, amely egyetlen foglalaton keresztül működik, és HTML 5-kompatibilis böngészőkben található JavaScript felületen keresztül látható.

Miután létrehozta a Web Socket kapcsolatot a webkiszolgálóval, küldhet adatokat böngészőből szerverre egy send () metódus meghívásával, és adatokat fogadhat szerverről böngészőre egy onmessage eseménykezelő segítségével.

Az alábbiakban az API hozza létre egy új WebSocket objektumot.

```
var Socket = new WebSocket(url, [protocal] );
```

Itt az első argumentum, url adja meg az URL-t, amelyhez csatlakozni kell. A második attribútum, a protokoll nem kötelező, és ha van, megad egy alprotokollot, amelyet a szervernek támogatnia kell a kapcsolat sikeres működéséhez.

WebSocket attribútumok

Az alábbiakban bemutatjuk a WebSocket objektum attribútumát. Feltételezve, hogy a fent említett Socket objektumot hoztuk létre –

| Sr.No. | Attribute & Description |
|---|---|
| 1 | **Socket.readyState**<br>A readonly readyState attribútum a kapcsolat állapotát képviseli. A következő értékek lehetnek:<br>• A 0 értéke azt jelzi, hogy a kapcsolat még nem jött létre.<br>• Az 1 érték azt jelzi, hogy a kapcsolat létrejött és a kommunikáció lehetséges.<br>• A 2 érték azt jelzi, hogy a kapcsolat véget ér.<br>• A 3 érték azt jelzi, hogy a kapcsolat lezárult, vagy nem sikerült megnyitni. |
| 2 | **Socket.bufferedAmount**<br>A readonly attribútum bufferedAmount az UTF-8 szöveg bájtjainak számát jelenti, amelyek a send () metódussal sorba kerültek. |

WebSocket események

Az alábbiakban bemutatjuk a WebSocket objektumhoz kapcsolódó eseményeket. Feltételezve, hogy a fent említett Socket objektumot hoztuk létre –

| Event | Event Handler | Description |
|---|---|---|
| open | Socket.onopen | Ez az esemény akkor jön létre, amikor a socket kapcsolat létrejön. |
| message | Socket.onmessage | Ez az esemény akkor fordul elő, amikor a kliens adatokat fogad a szervertől. |
| error | Socket.onerror | Ez az esemény akkor fordul elő, ha bármilyen hiba lép fel a kommunikációban. |
| close | Socket.onclose | Ez az esemény akkor következik be, amikor a kapcsolat megszakadt. |

WebSocket metódusok

Az alábbiakban bemutatjuk a WebSocket objektumhoz társított metódusokat. Feltételezve, hogy a fent említett Socket objektumot hoztuk létre –

| Sr.No. | Method & Description |
|---|---|
| 1 | **Socket.send()**<br>The send(data) method kapcsolat használatával továbbítja az adatokat. |
| 2 | **Socket.close()**<br>The close() method bármely meglévő kapcsolat megszakítására szolgálna. |

Kliensoldali HTML és JavaScript kód

Az oktatóanyag elkészítésekor csak kevés webböngésző támogatja a WebSocket () felületet. Kipróbálhatja a következő példát a Chrome, a Mozilla, az Opera és a Safari legújabb verziójával.

```
<!DOCTYPE HTML>

<html>
   <head>

      <script type = "text/javascript">
         function WebSocketTest() {

            if ("WebSocket" in window) {
               alert("WebSocket is supported by your Browser!");

               // Let us open a web socket
               var ws = new WebSocket("ws://localhost:9998/echo");

               ws.onopen = function() {

                  // Web Socket is connected, send data using send()
                  ws.send("Message to send");
                  alert("Message is sent...");
               };

               ws.onmessage = function (evt) {
                  var received_msg = evt.data;
                  alert("Message is received...");
               };

               ws.onclose = function() {

                  // websocket is closed.
                  alert("Connection is closed...");
               };
            } else {

               // The browser doesn't support WebSocket
               alert("WebSocket NOT supported by your Browser!");
            }
         }
      </script>

   </head>

   <body>
      <div id = "sse">
```

```
         <a href = "javascript:WebSocketTest()">Run WebSocket</a>
      </div>

   </body>
</html>
```

Telepítse a pywebsocket-et

Mielőtt tesztelné a fenti kliens programot, szüksége van egy szerverre, amely támogatja a WebSocket alkalmazást. Töltse le a mod_pywebsocket-x.x.x.tar.gz fájlt a pywebsocket-ből, amelynek célja egy Web Socket kiterjesztés biztosítása az Apache HTTP Server számára, és telepítse ezeket a lépéseket követve.

## HTML5 – Canvas

A HTML5 <canvas> elem segítségével egyszerűen és hatékonyan rajzolhat grafikákat a JavaScript használatával. Használható grafikonok rajzolására, fotókompozíciók készítésére vagy egyszerű (és nem is olyan egyszerű) animációk készítésére.

Itt van egy egyszerű <canvas> elem, amelynek csak két sajátos attribútuma van: szélesség és magasság, valamint az összes alapvető HTML5 attribútum, például id, name és osztály stb.

például:

```
<!DOCTYPE HTML>

<html>
   <head>

      <style>
         #mycanvas{border:1px solid red;}
      </style>
   </head>

   <body>
      <canvas id = "mycanvas" width = "100" height = "100"></canvas>
   </body>
</html>
```

```
var canvas  = document.getElementById("mycanvas");

if (canvas.getContext) {
   var ctx = canvas.getContext('2d');
   // drawing code here
} else {

   // canvas-unsupported code here
}
```

## HTML5 - Audio & Video

A HTML5 funkciói tartalmazzák a natív audio- és videotámogatást Flash nélkül.

A HTML5 <audio> és <video> címkék megkönnyítik a média hozzáadását egy webhelyhez. Be kell állítania az src attribútumot a médiaforrás azonosításához, és tartalmaznia kell egy vezérlő attribútumot, hogy a felhasználó lejátszhassa és szüneteltethesse a médiát.

Példa video:

```
<!DOCTYPE HTML>

<html>
   <body>

      <video  width = "300" height = "200" controls autoplay>
         <source src = "/html5/foo.ogg" type ="video/ogg" />
         <source src = "/html5/foo.mp4" type = "video/mp4" />
         Your browser does not support the <video> element.
      </video>

   </body>
</html>
```

Példa audio:

```
<!DOCTYPE HTML>

<html>
   <body>

      <audio controls autoplay>
         <source src = "/html5/audio.ogg" type = "audio/ogg" />
         <source src = "/html5/audio.wav" type = "audio/wav" />
         Your browser does not support the <audio> element.
      </audio>

   </body>
</html>
```

## HTML5 – Geolocation

A HTML5 Geolocation API segítségével megoszthatja tartózkodási helyét kedvenc webhelyeivel. A JavaScript képes rögzíteni a szélességi és hosszúsági fokokat, és elküldhető a háttérszerverre, és különféle helytudatos dolgokat végezhet, mint például helyi vállalkozások keresése vagy tartózkodási helyének megjelenítése a térképen.

Ma a legtöbb böngésző és mobil eszköz támogatja a Geolocation API-t. A földrajzi helymeghatározási API-k a globális navigációs objektum új tulajdonságával, azaz. Geolocation objektum, amely a következőképpen hozható létre:

```
var geolocation = navigator.geolocation;
```

Földrajzi helymeghatározási módszerek

A geolocation objektum a következő módszereket biztosítja

| Sr.No. | Method & Description |
|---|---|
| 1 | getCurrentPosition()<br>Ez a módszer beolvassa a felhasználó aktuális földrajzi helyzetét. |
| 2 | watchPosition()<br>Ez a módszer rendszeres frissítéseket kap az eszköz aktuális földrajzi helyzetéről. |
| 3 | clearWatch()<br>Ez a módszer törli a folyamatban lévő watchPosition hívást. |

Példa:

```
function getLocation() {
   var geolocation = navigator.geolocation;
   geolocation.getCurrentPosition(showLocation, errorHandler);
}
```

HTML5 – Microdata

A mikrodata egy szabványosított módszer arra, hogy további szemantikát biztosítson a weboldalain.

A Mikrodata segítségével meghatározhatja saját testreszabott elemeit, és elkezdheti az egyéni tulajdonságok beágyazását a weboldalain. Magas szinten a mikrodata név-érték párok csoportjából áll.

A csoportokat elemeknek nevezzük, és minden név-érték pár tulajdonság. Az elemeket és tulajdonságokat szabályos elemek jelentik.

Példa

Elem létrehozásához a itemscope attribútumot kell használni.

Tulajdonság hozzáadásához egy elemhez a itemprop attribútumot használják az elem egyik leszármazottjánál.

```
<html>
   <body>

      <div itemscope>
         <p>My name is <span itemprop = "name">Zara</span>.</p>
      </div>

      <div itemscope>
         <p>My name is <span itemprop = "name">Nuha</span>.</p>
      </div>

   </body>
</html>
```

Globális attribútumok

A Microdata öt globális attribútumot mutat be, amelyek minden elem számára elérhetők, és kontextust adnak a gépek számára az adatokról.

| Sr.No. | Attribute & Description |
|---|---|
| 1 | **itemscope**<br>Ez egy elem létrehozására szolgál. Az itemscope attribútum egy logikai attribútum, amely elmondja, hogy ezen az oldalon található Microdata, és innen indul. |
| 2 | **itemtype**<br>Ez az attribútum egy érvényes URL, amely meghatározza az elemet, és megadja a tulajdonságok kontextusát. |
| 3 | **itemid**<br>Ez az attribútum az elem globális azonosítója. |
| 4 | **itemprop**<br>Ez az attribútum meghatározza az elem tulajdonságait. |
| 5 | **itemref**<br>Ez az attribútum felsorolja a feltérképezéshez szükséges további elemeket, hogy megtalálja az elem név-érték párjait. |

Tulajdonságok adattípusok

A tulajdonságok általában olyan értékeket tartalmaznak, amelyek a fenti példában említett karakterláncok, de lehetnek URL-ek is. A következő példa egy tulajdonsággal rendelkezik, a "image", amelynek értéke URL –

```
<div itemscope>
   <img itemprop = "image" src = "tp-logo.gif" alt = "TutorialsPoint">
</div>
```

A tulajdonságoknak lehetnek dátum, időpont vagy dátum és idő értékek is. Ezt az időelem és a datetime attribútumával érhetjük el.

```
<html>
   <body>

      <div itemscope>
         My birthday is:
         <time itemprop = "birthday" datetime = "1971-05-08">
            Aug 5th 1971
         </time>
      </div>
```

```
   </body>
</html>
```

Mikrodata szókincs meghatározása

A mikroadatok szókincsének meghatározásához névtér URL-re van szükség, amely egy működő weboldalra mutat. Például a https://data-vocabulary.org/Person használható névtérként egy személyes mikrodata-szókincshez, a következő megnevezett tulajdonságokkal -

name - A személynév egyszerű karakterláncként

Photo - A személy képének URL-je.

URL - A személyhez tartozó webhely.

A személy tulajdonságainak körülbelül tulajdonságainak felhasználása a következő lehet:

```
<html>
   <body>

      <div itemscope>
         <section itemscope itemtype = "http://data-vocabulary.org/Person">
            <h1 itemprop = "name">Gopal K Varma</h1>

            <p>
               <img itemprop = "photo"
                  src = "http://www.tutorialspoint.com/green/images/logo.png">
            </p>

            <a itemprop = "url" href = "#">Site</a>
         </section>
      </div>

   </body>
</html>
```

HTML5 - Drag & drop

A Drag and Drop (DnD) hatékony felhasználói felület-koncepció, amely megkönnyíti az elemek másolását, átrendezését és törlését egérkattintások segítségével. Ez lehetővé teszi a felhasználó számára, hogy az egérgombot lenyomva tartsa egy elem felett, húzza egy másik helyre, és engedje el az egérgombot, hogy az elemet oda dobja.

A drag and drop funkcionalitás eléréséhez a hagyományos HTML4 segítségével a fejlesztőknek vagy komplex JavaScript programozást, vagy más JavaScript keretrendszert kell használniuk, például a jQuery stb.

Most a HTML 5 egy Drag and Drop (DnD) API-val állt elő, amely natív DnD támogatást hoz a böngészőhöz, így sokkal könnyebb kódolni.

A HTML 5 DnD-t az összes nagyobb böngésző támogatja, például a Chrome, a Firefox 3.5 és a Safari 4 stb.

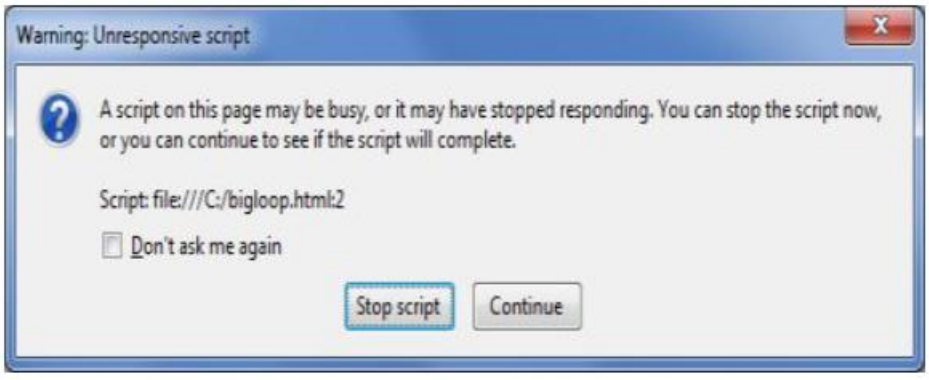
HTML5 - Web Workers

A JavaScript-et egyszálú környezetben történő futtatásra tervezték, vagyis több szkript nem futtatható egyszerre. Vizsgáljon meg egy olyan helyzetet, amikor kezelőfelület eseményeket kell kezelnie, nagy mennyiségű API adatot kell lekérdeznie és feldolgoznia, valamint kezelnie kell a DOM-ot.

A JavaScript felakasztja a böngészőt olyan helyzetekben, amikor a CPU kihasználtsága magas. Vegyünk egy egyszerű példát, ahol a JavaScript nagy cikluson megy keresztül -

```
<!DOCTYPE HTML>

<html>
   <head>
      <title>Big for loop</title>

      <script>
         function bigLoop() {

            for (var i = 0; i <= 10000; i += 1) {
               var j = i;
            }
            alert("Completed " + j + "iterations" );
         }

         function sayHello(){
            alert("Hello sir...." );
         }
      </script>

   </head>

   <body>
      <input type = "button" onclick = "bigLoop();" value = "Big Loop" />
      <input type = "button" onclick = "sayHello();" value = "Say Hello" />
   </body>
</html>
```

A Big Loop gombra kattintva a következő eredmény jelenik meg a Firefoxban –




Mi az a Web Worker?

A fentiekben ismertetett helyzet kezelhető a Web Workers segítségével, akik a számítási szempontból drága feladatokat elvégzik a felhasználói felület megszakítása nélkül, és általában külön szálakon futnak.

A Web Workers lehetővé teszik a hosszú ideig futó parancsfájlok futását, amelyeket nem szakítanak meg a kattintásokra vagy más felhasználói interakciókra válaszoló szkriptek, és lehetővé teszik a hosszú feladatok végrehajtását anélkül, hogy engednék tartani az oldalt.

A Web Workers háttérszkriptek, és viszonylag nagy súlyúak, és nem szánják őket nagy számban használni. Például nem lenne megfelelő egy Web Worker elindítása egy négy megapixeles kép minden pixeléhez.

Amikor egy szkript egy Web Worker belsejében fut, nem tud hozzáférni a weboldal ablakobjektumához (window.document), ami azt jelenti, hogy a Web Workers nem rendelkezik közvetlen hozzáféréssel a weboldalhoz és a DOM API-hoz. Bár a Web Workers nem tudják blokkolni a böngésző felhasználói felületét, mégis fogyasztanak CPU-ciklusokat.

Hogyan működnek a Web Workers?

A Web Workers egy JavaScript-fájl URL-jével inicializálják, amely tartalmazza a worker által végrehajtandó kódot. Ez a kód beállítja az event listener-eket, és kommunikál a főoldalról előidéző szkriptvel. A következő az egyszerű szintaxis –

```
var worker = new Worker('bigLoop.js');
```

Ha a megadott javascript fájl létezik, a böngésző új worker thread-et hoz létre, amelyet aszinkron módon tölt le. Ha a worker-hez vezető útvonal 404-es hibát ad vissza, akkor a worker leáll.

Ha alkalmazásának több támogató JavaScript-fájlja van, importálhatja azokat importScripts () módszerrel, amely a fájlneveket vesszővel elválasztott argumentumként veszi fel –

```
importScripts("helper.js", "anotherHelper.js");
```

A web worker megszületése után a web worker és a szülőoldala közötti kommunikáció a postMessage () módszerrel történik. Böngészőjétől / verziójától függően a postMessage () akár egy karakterláncot, akár egy JSON objektumot is elfogadhat egyetlen argumentumként.

A Web Worker által továbbított üzenethez a fő oldalon található onmessage esemény segítségével férhet hozzá. Most írjuk meg a bigLoop példánkat a Web Worker segítségével. Az alábbiakban látható a főoldal (hello.htm), amely egy web worker-t fog létrehozni a ciklus végrehajtásához és a változó végső értékének visszaadásához.

```
<!DOCTYPE HTML>

<html>
   <head>
      <title>Big for loop</title>

      <script>
         var worker = new Worker('bigLoop.js');
```

```
         worker.onmessage = function (event) {
            alert("Completed " + event.data + "iterations" );
         };

         function sayHello() {
            alert("Hello sir...." );
         }
      </script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello();" value = "Say Hello"/>
   </body>
</html>
```

A következő a bigLoop.js fájl tartalma. Ez a postMessage () API segítségével visszavezeti a kommunikációt a főoldalra –

```
for (var i = 0; i <= 1000000000; i += 1) {
   var j = i;
}
postMessage(j);
```

A web worker leállítása

A web workers nem állnak meg önmaguktól, de az őket elindító oldal megállíthatja őket a terminate () metódus meghívásával.

```
worker.terminate();
```

A terminated Web Worker már nem válaszol az üzenetekre, és nem végez további számításokat. Nem indíthatja újra a worker-t; ehelyett új worker-t hozhat létre ugyanazon URL segítségével.

A hibák kezelése

Az alábbiakban bemutatunk egy példát a Web Worker JavaScript fájl hibakezelési funkciójára, amely naplózza a hibákat a konzolra. Hibakezelési kód esetén a fenti példa a következővé válna –

```
<!DOCTYPE HTML>

<html>
   <head>
      <title>Big for loop</title>

      <script>
         var worker = new Worker('bigLoop.js');

         worker.onmessage = function (event) {
            alert("Completed " + event.data + "iterations" );
         };

         worker.onerror = function (event) {
            console.log(event.message, event);
         };
```

```
         function sayHello() {
            alert("Hello sir...." );
         }
      </script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello();" value = "Say Hello"/>
   </body>
</html>
```

A böngésző támogatásának ellenőrzése

Az alábbiakban bemutatjuk a böngészőben elérhető Web Worker szolgáltatás-támogatás észlelésének szintaxisát –

```
<!DOCTYPE HTML>

<html>
   <head>
      <title>Big for loop</title>
      <script src = "/js/modernizr-1.5.min.js"></script>

      <script>
      function myFunction() {

         if (Modernizr.webworkers) {
            alert("Congratulation!! you have web workers support." );
         } else {
            alert("Sorry!! you do not have web workers support." );
         }
      }
      </script>
   </head>

   <body>
      <button onclick = "myFunction()">Click me</button>
   </body>
</html>
```

HTML5 – IndexedDB

Az indexeddb egy új HTML5 koncepció, amely az adatokat a felhasználó böngészőjében tárolja. Az indexeddb nagyobb energiafogyasztást jelent, mint a local storage, és hasznos azoknak az alkalmazásoknak, amelyek nagy mennyiségű adat tárolását igénylik. Ezek az alkalmazások nagyobb hatékonyságot futhatnak és gyorsabban betöltődnek.

Miért érdemes használni az indexeddb fájlt?

A W3C bejelentette, hogy a Web SQL adatbázis egy elavult local storage specifikáció, ezért a webfejlesztőnek nem szabad tovább használni ezt a technológiát. Az indexeddb a webes SQL adatbázis alternatívája és hatékonyabb, mint a régebbi technológiák.

Jellemzők

- kulcspár értékeket tárol
- ez nem relációs adatbázis
- Az IndexedDB API többnyire aszinkron
- ez nem strukturált lekérdezési nyelv
- támogatta az adatok hozzáférését ugyanabból a tartományból

```
<!DOCTYPE html>

<html>
   <head>
      <meta http-equiv = "Content-Type" content = "text/html; charset = utf-8" />
      <script type = "text/javascript">

         //prefixes of implementation that we want to test
         window.indexedDB = window.indexedDB || window.mozIndexedDB ||
         window.webkitIndexedDB || window.msIndexedDB;

         //prefixes of window.IDB objects
         window.IDBTransaction = window.IDBTransaction ||
         window.webkitIDBTransaction || window.msIDBTransaction;
         window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange ||
         window.msIDBKeyRange

         if (!window.indexedDB) {
            window.alert("Your browser doesn't support a stable version of
IndexedDB.")
         }

         const employeeData = [
            { id: "00-01", name: "gopal", age: 35, email:
"gopal@tutorialspoint.com" },
            { id: "00-02", name: "prasad", age: 32, email:
"prasad@tutorialspoint.com" }
         ];
         var db;
         var request = window.indexedDB.open("newDatabase", 1);

         request.onerror = function(event) {
            console.log("error: ");
         };

         request.onsuccess = function(event) {
            db = request.result;
            console.log("success: "+ db);
         };

         request.onupgradeneeded = function(event) {
            var db = event.target.result;
            var objectStore = db.createObjectStore("employee", {keyPath: "id"});

            for (var i in employeeData) {
               objectStore.add(employeeData[i]);
            }
         }

         function read() {
            var transaction = db.transaction(["employee"]);
            var objectStore = transaction.objectStore("employee");
            var request = objectStore.get("00-03");

            request.onerror = function(event) {
               alert("Unable to retrieve daa from database!");
```

```
            };

            request.onsuccess = function(event) {
               // Do something with the request.result!
               if(request.result) {
                  alert("Name: " + request.result.name + ",
                     Age: " + request.result.age + ", Email: " +
request.result.email);
               } else {
                  alert("Kenny couldn't be found in your database!");
               }
            };
         }

         function readAll() {
            var objectStore = db.transaction("employee").objectStore("employee");

            objectStore.openCursor().onsuccess = function(event) {
               var cursor = event.target.result;

               if (cursor) {
                  alert("Name for id " + cursor.key + " is " + cursor.value.name +
",
                     Age: " + cursor.value.age + ", Email: " + cursor.value.email);
                  cursor.continue();
               } else {
                  alert("No more entries!");
               }
            };
         }

         function add() {
            var request = db.transaction(["employee"], "readwrite")
            .objectStore("employee")
            .add({ id: "00-03", name: "Kenny", age: 19, email: "kenny@planet.org"
});

            request.onsuccess = function(event) {
               alert("Kenny has been added to your database.");
            };

            request.onerror = function(event) {
               alert("Unable to add data\r\nKenny is aready exist in your database!
");
            }
         }

         function remove() {
            var request = db.transaction(["employee"], "readwrite")
            .objectStore("employee")
            .delete("00-03");

            request.onsuccess = function(event) {
               alert("Kenny's entry has been removed from your database.");
            };
         }
      </script>

   </head>
   <body>
      <button onclick = "read()">Read </button>
      <button onclick = "readAll()">Read all </button>
      <button onclick = "add()">Add data </button>
      <button onclick = "remove()">Delete data </button>
   </body>
</html>
```

HTML5 - Web messaging

A webes üzenetküldés az a képesség, hogy valós idejű üzeneteket küldjön a szerverről az ügyfél böngészőjébe. Felülírja a tartományok közötti kommunikációs problémát a különböző tartományokban, protokollokban vagy portokban

Üzenet esemény (Message Event)

Az üzenetesemények (Message Event) kiváltják a dokumentumok közötti üzenetküldést, a csatornaüzenetet (channel messaging), a szerver által küldött eseményeket és a webes socketeket. Ezt az Üzenet esemény(Message Event) kezelőfelülete írta le.

| Sr.No. | Attributes & Description |
|---|---|
| 1 | **data**<br>Karakterlánc-adatokat tartalmaz |
| 2 | **origin**<br>Tartománynevet és portot tartalmaz |
| 3 | **lastEventId**<br>Az aktuális üzenet esemény egyedi azonosítóját tartalmazza. |
| 4 | **source**<br>Tartalmaz egy hivatkozást az eredeti dokumentum ablakára |
| 5 | **ports**<br>Bármely üzenetport által küldött adatokat tartalmaz |

cross-document üzenet küldése

A cross-document üzenet elküldése előtt létre kell hoznunk egy új webböngészési környezetet, akár új iframe vagy új ablak létrehozásával. Az adatokat a postMessage () segítségével küldhetjük el, és két argumentuma van. Olyanok, mint -

- message - Az elküldendő üzenet
- targetOrigin - Eredet neve

Példák

Üzenet küldése iframe-ről gombra

```
var iframe = document.querySelector('iframe');
var button = document.querySelector('button');
```

```
var clickHandler = function() {
   iframe.contentWindow.postMessage('The message to send.',
      'https://www.tutorialspoint.com);
}
button.addEventListener('click',clickHandler,false);
```

üzenet fogadása a fogadó dokumentumban

```
var messageEventHandler = function(event){

   // check that the origin is one we want.
   if(event.origin == 'https://www.tutorialspoint.com') {
      alert(event.data);
   }
}
window.addEventListener('message', messageEventHandler,false);
```

Csatornaüzenetek

A böngészési összefüggések közötti kétirányú kommunikációt csatornás üzenetküldésnek nevezzük. Hasznos több eredetű kommunikációhoz.

A MessageChannel és a MessagePort objektumok

Az messageChannel létrehozása közben két portot hoz létre az adatok küldéséhez és továbbításához egy másik böngészési környezetbe.

- postMessage () - Tegye közzé az üzenet dobócsatornáját
- start () - Elküldi az adatokat
- close () - Bezárja a port-ot

Ebben a forgatókönyvben az adatokat az egyik iframe-ről a másik iframe-re küldjük. Itt a függvényben lévő adatokat hívjuk meg, és továbbítjuk az adatokat a DOM-nak.

```
var loadHandler = function() {
   var mc, portMessageHandler;
   mc = new MessageChannel();
   window.parent.postMessage('documentAHasLoaded','http://foo.example',[mc.port2]);

   portMessageHandler = function(portMsgEvent) {
      alert( portMsgEvent.data );
   }

   mc.port1.addEventListener('message', portMessageHandler, false);
   mc.port1.start();
}
window.addEventListener('DOMContentLoaded', loadHandler, false);
```

A kód felett az adatokat a 2. portról veszi át, most továbbítja az adatokat a második iframe-be

```
var loadHandler = function() {
   var iframes, messageHandler;
   iframes = window.frames;
```

```
   messageHandler = function(messageEvent) {

      if( messageEvent.ports.length > 0 ) {

         // transfer the port to iframe[1]

iframes[1].postMessage('portopen','http://foo.example',messageEvent.ports);
      }
   }
   window.addEventListener('message',messageHandler,false);
}
window.addEventListener('DOMContentLoaded',loadHandler,false);
```

Most a második dokumentum kezeli az adatokat a portMsgHandler függvény használatával.

```
var loadHandler() {

   // Define our message handler function
   var messageHandler = function(messageEvent) {

      // Our form submission handler

      var formHandler = function() {
         var msg = 'add <foo@example.com> to game circle.';
         messageEvent.ports[0].postMessage(msg);
      }
      document.forms[0].addEventListener('submit',formHandler,false);
   }
   window.addEventListener('message',messageHandler,false);
}
window.addEventListener('DOMContentLoaded',loadHandler,false);
```

HTML5 – CORS

A származási helyek közötti erőforrás-megosztás (Cross-origin resource sharing - CORS) egy olyan mechanizmus, amely lehetővé teszi a böngészőben egy másik tartomány korlátozott erőforrásait.

Tegyük fel, hogy ha a HTML5- video lejátszóra kattint a HTML5 demo szakaszokban. engedélyt fog kérni a kamerától. ha a felhasználó engedélyezi, akkor csak ő nyitja meg a kamerát, különben nem nyitja meg a kamerát webalkalmazásokhoz.

CORS kérés benyújtása

Itt a Chrome, a Firefox, az Opera és a Safari egyaránt az XMLHttprequest2 objektumot használja, az Internet Explorer pedig a hasonló XDomainRequest objektumot.

```
function createCORSRequest(method, url) {
   var xhr = new XMLHttpRequest();

   if ("withCredentials" in xhr) {

      // Check if the XMLHttpRequest object has a "withCredentials" property.
      // "withCredentials" only exists on XMLHTTPRequest2 objects.
      xhr.open(method, url, true);
   } else if (typeof XDomainRequest != "undefined") {

      // Otherwise, check if XDomainRequest.
      // XDomainRequest only exists in IE, and is IE's way of making CORS requests.
```

```
      xhr = new XDomainRequest();
      xhr.open(method, url);
   } else {

      // Otherwise, CORS is not supported by the browser.
      xhr = null;
   }
   return xhr;
}

var xhr = createCORSRequest('GET', url);

if (!xhr) {
   throw new Error('CORS not supported');
}
```

Eseménykezelők a CORS-ban

| Sr.No. | Event Handler & Description |
|---|---|
| 1 | **onloadstart**<br>Elindítja a kérést |
| 2 | **onprogress**<br>Betölti az adatokat és elküldi azokat |
| 3 | **onabort**<br>Törölje a kérést |
| 4 | **onerror**<br>a kérés nem sikerült |
| 5 | **onload**<br>kérelem betöltése sikeresen |
| 6 | **ontimeout**<br>időkorlát történt, mielőtt a kérelem teljesülhetett volna |
| 7 | **onloadend**<br>Amikor a kérés befejeződött, vagy sikeres, vagy sikertelen |

## Example of onload or onerror event

```
xhr.onload = function() {
   var responseText = xhr.responseText;
```

```
   // process the response.
   console.log(responseText);
};

xhr.onerror = function() {
   console.log('There was an error!');
};
```

## Example of CORS with handler

Below example will show the example of makeCorsRequest() and onload handler

```
// Create the XHR object.
function createCORSRequest(method, url) {
   var xhr = new XMLHttpRequest();

   if ("withCredentials" in xhr) {

      // XHR for Chrome/Firefox/Opera/Safari.
      xhr.open(method, url, true);
   } else if (typeof XDomainRequest != "undefined") {

      // XDomainRequest for IE.
      xhr = new XDomainRequest();
      xhr.open(method, url);
   } else {

      // CORS not supported.
      xhr = null;
   }
   return xhr;
}

// Helper method to parse the title tag from the response.
function getTitle(text) {
   return text.match('<title>(.*)?</title>')[1];
}

// Make the actual CORS request.
function makeCorsRequest() {

   // All HTML5 Rocks properties support CORS.
   var url = 'http://www.tutorialspoint.com';

   var xhr = createCORSRequest('GET', url);

   if (!xhr) {
      alert('CORS not supported');
      return;
   }

   // Response handlers.
   xhr.onload = function() {
      var text = xhr.responseText;
      var title = getTitle(text);
      alert('Response from CORS request to ' + url + ': ' + title);
   };

   xhr.onerror = function() {
      alert('Woops, there was an error making the request.');
   };
   xhr.send();
}
```

HTML5 - Web RTC

Web RTC, amelyet a World Wide Web Consortium (W3C) vezetett be. Ez támogatja a böngésző-böngésző alkalmazásokat a hanghívásokhoz, a videocsevegéshez és a P2P fájlmegosztáshoz.

Ha ki akarod próbálni a webes RTC elérhető a Chrome, az Opera és a Firefox számára. Jó kiindulópont az itt található egyszerű video chat alkalmazás. A Web RTC három API-t valósít meg az alábbiak szerint -

- MediaStream - hozzáférést kap a felhasználó kamerájához és mikrofonjához.
- RTCPeerConnection - hozzáférést kap audio vagy videohívási lehetőséghez.
- RTCDataChannel - hozzáférés a peer-to-peer kommunikációhoz.

MediaStream

A MediaStream a média szinkronizált folyamatait képviseli. Például kattintson a HTML5 Video player elemre a HTML5 demo szakaszban, vagy kattintson ide.

A fenti példa a stream.getAudioTracks () és a stream.VideoTracks () fájlokat tartalmazza. Ha nincs hangsáv, akkor egy üres tömböt ad vissza, és ellenőrzi a videofolyamot, ha a webkamera csatlakoztatva van, a stream.getVideoTracks () egy MediaStreamTrack tömböt ad vissza, amely a webkameráról közvetíti az adatfolyamot. Egyszerű példa a csevegőalkalmazások: a csevegőalkalmazás webkameráról, hátsó kameráról, mikrofonról kap streamet.

## Sample code of MediaStream

```
function gotStream(stream) {
   window.AudioContext = window.AudioContext || window.webkitAudioContext;
   var audioContext = new AudioContext();

   // Create an AudioNode from the stream
   var mediaStreamSource = audioContext.createMediaStreamSource(stream);

   // Connect it to destination to hear yourself
   // or any other node for processing!
   mediaStreamSource.connect(audioContext.destination);
}
navigator.getUserMedia({audio:true}, gotStream);
```

Képernyő rögzítése

Ez a Chrome böngészőben is lehetséges a mediaStreamSource segítségével, és ehhez HTTPS szükséges. Ez a funkció az operában még nem érhető el.

Session -vezérlés, hálózati és médiainformációk

A Web RTC megkövetelte a böngészők közötti peer-to-peer kommunikációt. Ehhez a mechanizmushoz szükség volt jelzésre, hálózati információkra, Session -vezérlésre és médiainformációkra. A webfejlesztők különböző mechanizmusokat választhatnak a böngészők

közötti kommunikációhoz, például SIP vagy XMPP, vagy bármilyen kétirányú kommunikációt. Az XHR mintapéldája itt található.

### Sample code of createSignalingChannel()

```
var signalingChannel = createSignalingChannel();
var pc;
var configuration = ...;

// run start(true) to initiate a call
function start(isCaller) {
   pc = new RTCPeerConnection(configuration);

   // send any ice candidates to the other peer
   pc.onicecandidate = function (evt) {
      signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
   };

   // once remote stream arrives, show it in the remote video element
   pc.onaddstream = function (evt) {
      remoteView.src = URL.createObjectURL(evt.stream);
   };

   // get the local stream, show it in the local video element and send it
   navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {
      selfView.src = URL.createObjectURL(stream);
      pc.addStream(stream);

      if (isCaller)
         pc.createOffer(gotDescription);

      else
         pc.createAnswer(pc.remoteDescription, gotDescription);

         function gotDescription(desc) {
            pc.setLocalDescription(desc);
            signalingChannel.send(JSON.stringify({ "sdp": desc }));
         }
      });
   }

   signalingChannel.onmessage = function (evt) {
      if (!pc)
         start(false);
         var signal = JSON.parse(evt.data);

      if (signal.sdp)
         pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));

      else
         pc.addIceCandidate(new RTCIceCandidate(signal.candidate));
};
```