WEBES ALKALMAZÁSFEJLESZTÉS 1.

Horváth Győző

Egyetemi adjunktus

1117 Budapest,
Pázmány Péter sétány 1/C, 2.420
Tel: (1) 372-2500/1816

Tartalom

- A böngésző és a JavaScript
- Browser Object Model
- Document Object Model
  - JavaScript keretrendszerek
  - jQuery
  - Elemek kiválasztása
  - Elemek módosítása
  - Eseménykezelés
  - jQuery segédfüggvények

A böngésző és a JavaScript

BOM és DOM

Böngésző és JavaScript

- A kliens oldali webprogramozásban a JavaScript futtatókörnyezete a böngésző
- A böngészők és a JS motor közötti kapcsolattartásra két modell:
  - BOM (Browser Object Model): egész JavaScript környezet mögött álló óriási objektum
  - DOM (Document Object Model): a HTML struktúra JavaScript objektumokra való leképezése egyetlen objektumhierarchiába

Browser Object Model – BOM

BOM

- A DOM csak a HTML struktúráért felelős objektum-modell
- BOM: olyan réteg, ami a JS core és a böngésző között áll
- A BOM a kliens oldali webprogramozás alaprétege
  - Futtatókörnyezet
- A BOM a böngészőt leképező objektum-modell
  - A DOM ennek része (ahogy a HTML struktúra is része a böngészőablaknak)
- BOM nem szabványos, de egységes

BOM részei

- window
  - JavaScript core
  - location objektum
  - history objektum
  - document objektum
  - navigator objektum
  - screen objektum

window

- Böngésző minden ablakához vagy lapjához tartozik egy window nevű JavaScript objektum
- Ez a weboldal JavaScript futtatókörnyezetének alapja
- Ez a globális névtér
- Ennek része a HTML tartalom JavaScript objektuma is (DOM)
- Az ablak és a BOM élő kapcsolatban van: a BOM-beli változások hatással vannak a valódi ablakra

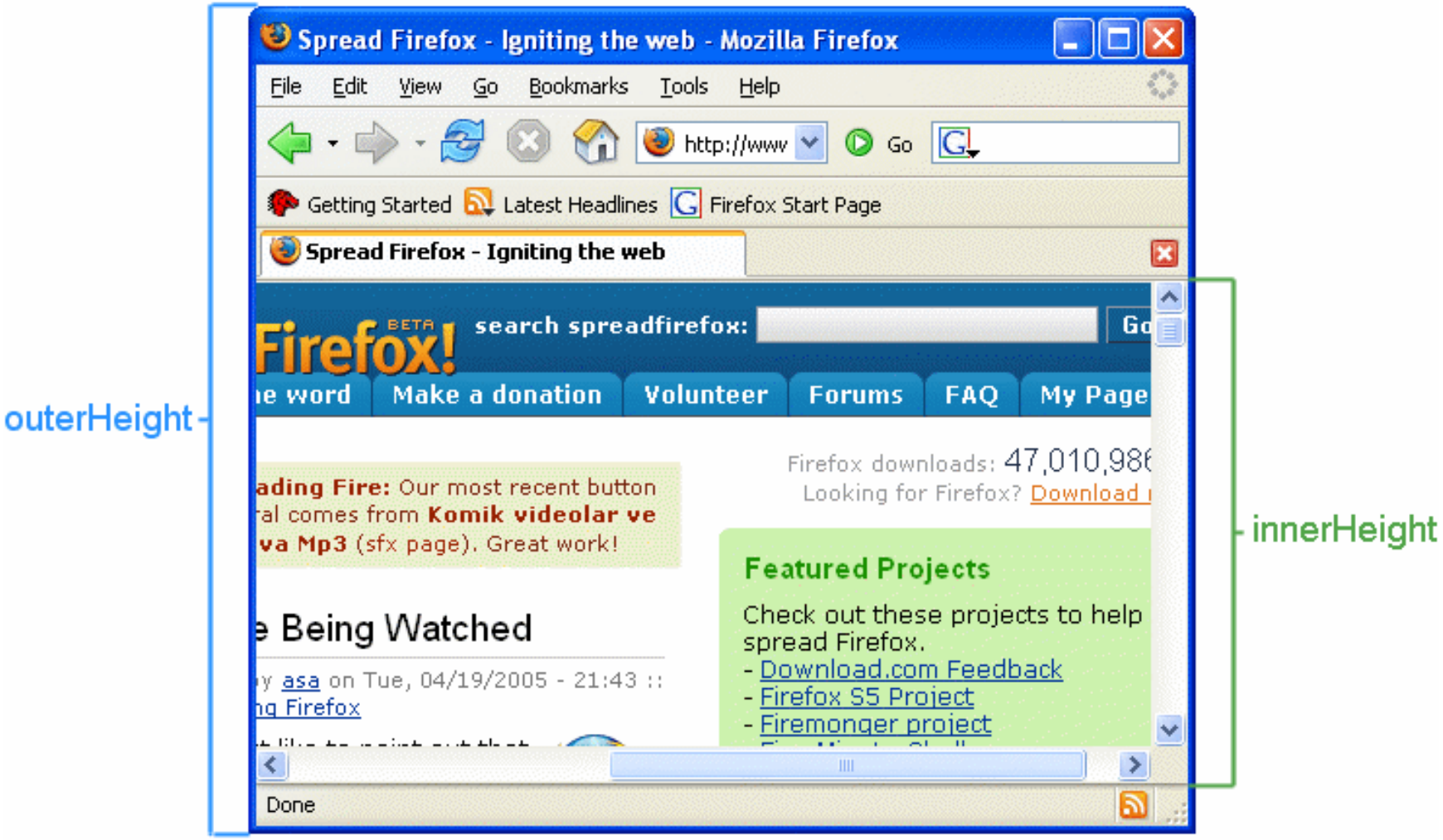
window metódusai és tulajdonságai

- resizeTo(w, h)
- resizeBy(dw, dh)
- moveTo(x, y)
- moveBy(dx, dy)
- innerWidth
- innerHeight
- outerWidth
- outerHeight

```
window.resizeTo(800, 600);     // Ablak: 800px széles, 600px magas
resizeTo(800, 600);            // Az eredmény ugyanaz - ugyanis sajnos
                               ez a metódus - sok társához hasonlóan - a
                               globális névteret szennyezi, tehát ha nem
                               használunk saját névteret könnyedén
                               felüldefiniálhatjuk

resizeTo = function() { /* ...sok-sok programkódom... */ };
window.resizeTo(800, 600);     // Most már az én metódusom futna le, nem
                               az átméretezés
```

window: innerHeight és outerHeight



window.location

- Aktuális weboldal URL-információja
- Tulajdonságok
  - href, protocol, hostname, port, pathname, search
  - Változás ezekben → assign metódussal újratöltés
- Metódusok
  - assign, replace, reload
- Átirányítások megvalósítása

```
location.href = 'http://www.crockford.com';
// átirányítás az oldalra, a jelenlegi oldal bekerül a history-ba, tehát a
böngésző vissza gombja az aktuális lapra irányít
location.href.replace('http://www.crockford.com');
// átirányítás az oldalra anélkül, hogy a jelenlegi oldal URL-jét elmentené a
history-ban. Hasznos, ha nem akarjuk, hogy egy oldalra visszanavigálhasson a
felhasználó
```

window.history

- Böngészési előzmények információja
- Modern böngészőkben korlátozott
- Oldalak közötti navigálásra hasznos
- Metódusok
  - back, forward, go

```
history.back();      // átirányítás az előző oldalra a böngészési
                        előzmények alapján
history.forward();   // átirányítás a következő oldalra a böngészési
                        előzmények alapján
history.go(-1);      // átirányítás az előző oldalra a böngészési
                        előzmények alapján
history.go(1);       // átirányítás a következő oldalra a böngészési
                        előzmények alapján
```

window.navigator

- Böngészőinformációk
  - típus, verzió, környezeti beállítás
- Kerüljük használatát
  - Hibás értékeket tartalmazhat bizonyos böngészőkben (kompatibilitási okok miatt)
  - Browser detection helyett ma már feature detection használata: adattagok, metódusok meglétének vizsgálata, hiányuk esetén emulálásuk
- Böngészőnk ujjlenyomata, azonosítás
  - operációs rendszer, plugin-ek, stb.

Időzítés JavaScriptben

- setTimeout, clearTimeout: egyszeri végrehajtás
- setInterval, clearInterval: ismétlődő végrehajtás
- 1. paraméter: nevesített vagy anoním függvénykifejezés
- 2. paraméter: ezredmásodperc
- Visszatérési érték: number típusú azonosító
- setInterval használatát kerüljük
  - időigényesebb feladatok esetén felhalmozódhat
  - szimuláljuk setTimeout-tal

setTimeout és clearTimeout


```
var kesleltetettKiiras = function() {
    console.log('Etel evett egy tehenet');
}

// Nevesített függvény kifejezéssel
var a = setTimeout(kesleltetettKiiras, 1000);
// 1mp múlva megjelenik a konzolon: 'Etel evett egy tehenet'

// Anonim függvénnyel ugyanez
var b = setTimeout(function() {
   console.log('Etel evett egy tehenet');
}, 1000);

// Az előbbi (időzített anonim) függvény futásának megelőzése
if (typeof b === "number") {
    clearTimeout(b);
}
```

setInterval és clearInterval


```
var c = setInterval(function() {
    console.log('de lehet nem ette meg!');
}, 1000);
// 1mp-es időközönként megjelenik a konzolban: 'de lehet nem ette
meg!'

// A setInterval leállítása
if (typeof c == "number") {
    clearInterval(c);
}
```

setInterval kiváltása setTimeout-tal


```
var d;
(function nevtelen() {
    console.log('de lehet nem ette meg!');
    d = setTimeout(nevtelen, 1000);
}());

clearTimeout(d);
```

Document Object Model – DOM

DOM

- A DOM tetszőleges faszerkezetű dokumentum (XML, XHTML, HTML) faszerkezetű objektumhierarchiába való leképezése.
- A dokumentumban az elemek fa struktúrában helyezkednek el.
- Minden elemnek megfelel egy objektum
- Ezek az objektumok az eredeti dokumentum-struktúrának megfelelően fa struktúrába vannak szervezve
  - szülő, gyerekek, testvérek
- A DOM ennek az objektumhierarchiának a lekérdezésére, bejárására, manipulálására ad megfelelő interfészt (tulajdonságok metódusok)

HTML DOM

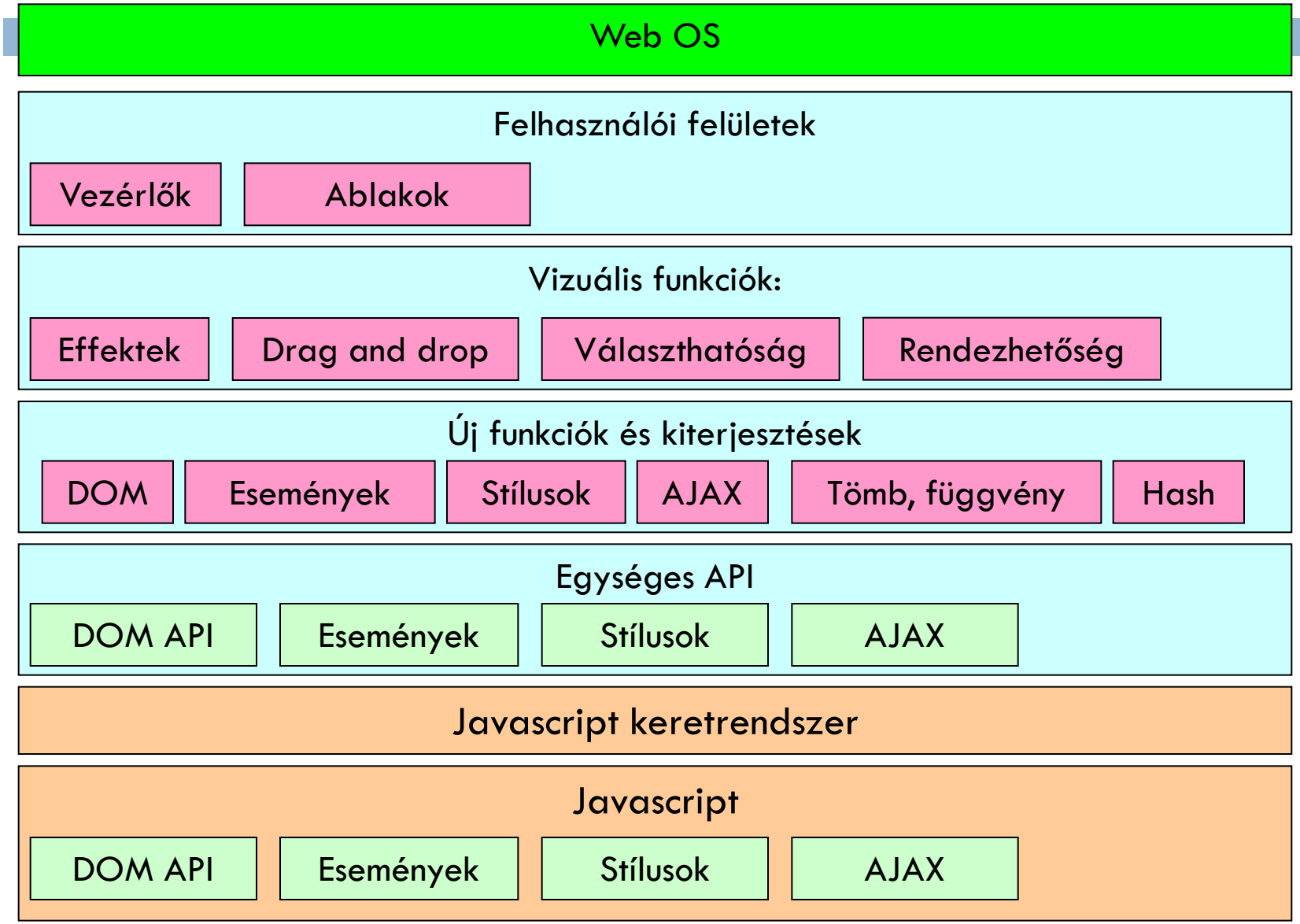
- A böngésző esetén a dokumentum a HTML dokumentum
- Ebben az elemek (tag-ek, szöveges részek) fa struktúrába szerveződnek
- Minden elemnek egy JavaScript objektum felel meg
- Ezek az objektumok a HTML struktúrának megfelelően fa struktúrába vannak szervezve
- A HTML DOM ehhez ad interfészt
- window.document a gyökere (DOM része a BOM-nak)

A DOM és a weboldal kapcsolata

- Nagyon szoros kapcsolat van a DOM és a weboldal között
- Élő kapcsolat
- Módosítás a weboldalon megváltoztatja az elemnek megfelelő JavaScript objektum tulajdonságait
- JavaScript objektum tulajdonságainak változtatása az eredeti HTML elem módosulását okozza
- Nagyon erőforrás igényes ennek felügyelete

JavaScript keretrendszerek

JavaScript keretrendszerek



JavaScript keretrendszerek

- A keretrendszerek egy absztrakt réteget alakítanak ki a DOM felett.
- A fejlesztés:
  - Kényelmesebb
  - Gyorsabb
  - Biztonságosabb
  - Böngészőfüggetlen
  - (Kevésbé hatékony)
- Bővítik a DOM szolgáltatásait új funkciókkal

JavaScript keretrendszerek

- Sokféle JavaScript keretrendszer van már
  - jQuery, YUI, Prototype, MooTools, Dojo, qooxdoo, ExtJS
- Funkcióikban, sebességben hasonlók
- Kódolási stílusuk eltérő (OOP vs funkcionális)
- Mi a jQuery-t használjuk
  - Egyszerű, intuitív, átlátható, jól dokumentált, gyorsan fejlődő, elterjedt

jQuery

jQuery

- JavaScript keretrendszer, függvénykönyvtár
- 2005-ben született, John Resig nevéhez fűződik
- Új szemléletmód
  - modern és kényelmes tervezési minták
  - meghagyta a JavaScript alapvető programozási stílusát
  - érthető tudott maradni
- Böngészőfüggetlen kód
- Rengeteg plugin
- Részei: jQuery Core (Sizzle), jQuery UI, Qunit, jQuery Mobile

jQuery filozófia

- Válasszunk ki elemeket a dokumentumból
- Rakjuk be egy tömbbe
- Ezeken hajtsunk végre műveleteket
- Segítsünk a programozónak
  - Elemeket kiválasztani és elérni
  - Automatikusan rakjuk a tömbbe
  - A ciklust szervezzük meg helyette!
- A lényeg marad: `$('p').hide();`
  - A kiválasztási szabály megadása: p
  - Művelet meghatározása: hide()

jQuery objektum

- A filozófia hátterében a keretrendszer legfontosabb alkotóeleme, a jQuery objektum áll
- Tömb: egy vagy több HTML elemnek megfelelő JavaScript objektumot tartalmaz
- Az elemeket szelektorokkal választjuk ki
- Objektum: számos metódus a tömbbeli elemek feldolgozásához
- Létrehozása a jQuery vagy a $ függvénnyel

jQuery objektum

- Tömbszerű objektum
- Valahogy így:

```
var o = {
    length: 0,
    splice: [].splice
}
console.log(o);
```

jQuery függvény, azaz $()

- jQuery(selector, [context]) – Kiválasztás, szelekció
  - jQuery(selector, [context])
  - jQuery(element)
  - jQuery(elementArray)
  - jQuery(jQuery object)
  - jQuery()
- jQuery(html, [ownerDocument]) – Létrehozás
  - jQuery(html, [ownerDocument])
  - jQuery(html, props)
- jQuery(callback) – Dokumentumbetöltődés
  - jQuery(callback)

jQuery telepítése, példaoldal


```
<!doctype html>
<html>
  <head>
    <!-- Az src attribútumba a jQuery forrás elérési útvonalát kell írni -->
    <script type="text/javascript" src="jquery.js"></script>
</head>
  <body>
    <h1>Teszt Elek CV</h1>

    <div>
        <img id="profilkep" class="adat" src="profil.jpg" alt="Profilkép" />
    </div>

    <div id="adatok">
        <ul>
            <li class="adat"><b>Leánykori név:</b>   Teszt Elek</li>
            <li class="adat"><b>Születési dátum:</b> 1950.05.05.</li>
            <li class="adat"><b>Végzettség:</b>      Géplakatos</li>
            <li class="adat"><b>Foglalkozás:</b>     Szoftvertesztelő</li>
        </ul>
    </div>
  </body>
</html>
```

Dokumentum betöltődése

- DOM-ot érintő JavaScript kódot a DOM felépülése után érdemes futtatni
- Ezt a DOM a megfelelő eseménnyel jelzi, erre kell feliratkozni

```
// Írjuk a programkódot a betöltődést figyelő eseménykezelőben:
$(document).ready(function() {
    console.log("Betöltődtem!");
});
// Létezik egy rövidített változata az előzőnek:
$(function() {
    console.log("Betöltődtem!");
});
// Természetesen nem szükséges anonim függvényt átadni az eseménykezelőnek,
függvény literált, például saját program-függvényünket is odaadhatjuk neki:
var programom = function() {
    console.log("Betöltődtem!");
}
$(programom);
```

jQuery interfészek

- DOM
  - Szelektorok (CSS 1, 2, 3)
  - Attribútumok
  - Struktúra bejárása
  - Módosítás
- CSS
- Események
- AJAX
- Effektek

Dokumentáció

- Továbbiakban az egyes interfészek filozófiájának a bemutatása következik
  - Lényeg
  - Hatékonysági megfontolások
- Ez az előadás nem referencia, így nem is teljes a bemutatás
- Dokumentáció: http://docs.jquery.com/Main_Page

Elemek kiválasztása – szelekció

Kiválasztás elve

- CSS szelektorral: $('css szelektor') → jQuery obj.
- CSS1-3 szelektorok böngészőfüggetlenül (pl. IE6!)
  - Alap szelektorok: elem, id, osztály
  - Attribútum szelektorok (pl. [name="value"])
  - Űrlap szelektrok (pl. :checkbox)
  - Tartalom szűrő szelektorok (pl. :contains())
  - pszeudo-szelektor (pl. :first, :animated, :hidden)
  - összetett szelektorok: többszörös, hierarchikus
- Kiválasztott elemek száma: length tulajdonság

Kiválasztás – példa


```
$("#profilkep");    // => [img#profilkep.adat profil.jpg], jQuery objektum a
                       kiválasztott profilkep azonosítójú HTML elemmel
$(".adat");         // => [img#profilkep.adat profil.jpg, li.adat, li.adat,
          li.adat, li.adat], kiválasztotta az összes .adat osztályú HTML elemet
$("li.adat");       // => [li.adat, li.adat, li.adat, li.adat]
$("div.adat");      // => [], üres jQuery objektum, mivel nincs ilyen adat
                          osztályú div az oldalon
$("#profilkep, #adatok");     // vesszővel elválasztva több CSS szelektor is
                                 megadható
$("img[src=profil.jpg]");     // kiválasztás attribútum alapján
$("ul li.adat b");            // kiválasztás CSS származtatással
$("ul li:first");             // kiválasztás pszeudo-szelektorral (az első li
                                 elemet adja vissza az összes olyan li közül, ami
                                 ul-ban van)
$("b:contains('Végzettség:')");   // kiválasztás pszeudo-szelektorral (a
                                 'Végzettség:' tartalmú <b> elemet adja vissza)

typeof $("#profilkep");           // => object

if ($("div.adat").length) { alert('Vagyok!'); }   // nem történik semmi (hamis)
if ($("li.adat").length)  { alert('Tényleg vagyok!'); } // megjelenik (igaz)
```

Láncolás (chaining)

- Minden jQuery metódus saját jQuery objektumával tér vissza
- → Egymás után fűzhetjük a metódusokat
- Túl hosszú láncolást kerüljük: nehéz módosítani és debugolni.

```
$("b").html('Adat:').css({ color: 'red', fontSize: 20 });
// a html metódussal megváltoztatjuk a kiválasztott elem(ek)
tartalmát, a css-el pedig módosítjuk a css attribútumait
```

Kiválasztott elemek cachelése

- Ha többször fel akarjuk használni a kiválasztott elemeket, akkor érdemes elmenteni őket
- Hungarian notation: $kepek
- A cachelt objektumok nem változnak az oldallal

```
var $adatok = $(".adat"),
    $divek = $("div");

// Minden adat osztályú HTML elem tartalmának módosítása:
for (var i=0, l=$adatok.length; i<l; i++) {
    $adatok.eq(i).html('Átírtalak!'); // Az eq a paraméterében megkapott
                                      indexű elemmel tér vissza
}
// Az előbbi egyszerűbben, mivel a jQuery objektum az utasításainkat -
amikor csak tudja - minden benne foglalt elemre végrehajtja:
$adatok.html('Átírtalak');
```

Elemek módosítása - manipuláció

Manipuláció

- Kiválasztott elem tulajdonságának megváltoztatása azonnal tükröződik a HTML dokumentumban
- Manipuláló metódusok getter és setter metódusok egyszerre
- setter metódusok láncolhatók
- getter metódusok értékkel térnek vissza
  - nem láncolhatók
  - több elem esetén az első elem értékével (van pár kivétel)

Stílusok manipulációja

- Egyedi style attribútumok módosítása
  - css metódus
  - css(propertyName)
  - css(propertyName, value)
    - .css(propertyName, function(index, value))
    - .css(map)
- Stílusosztály módosítása
  - addClass
  - removeClass
  - hasClass
  - toggleClass

Stílusok manipulációja


```
//Stílusattribútumok manipulációja
$("#adatok").css("width");              // => "1264px", getter

$("#adatok").css("width", "1000px")     // módosítás 1000px-re, setter
            .css("width");              // => "1000px"

$("#adatok").css({
    width: "1000px",
    fontSize: "20px"
}); // Több attribútum módosítása objektum literállal

//Stílusosztályok manipulációja
$("#adatok").addClass("ujCSSOsztaly");    // CSS osztály hozzáadása
$("#adatok").hasClass("ujCSSOsztaly");    // => true, CSS osztály
ellenőrzése
$("#adatok").removeClass("ujCSSOsztaly"); // CSS osztály elvétele
$("#adatok").toggleClass("ujCSSOsztaly"); // CSS osztály váltogatása
```

Animációk

- Az animáció CSS attribútumok gyors, egymás utáni módosítgatása
- animate metódus
  - .animate(properties, [duration], [easing], [complete])
  - .animate(properties, options)
- Effektusok
  - show, hide, toggle, fadeIn, fadeOut, fadeToggle, slideUp, slideDown, slideToggle
  - queue, dequeue, stop

Animációk

A) Egyszerű animáció

```
$("#adatok").animate({ marginTop: '100px' }, 400);    // 400ms alatt lefutó animáció

// Ugyanazon elemen futtatott két animate metódus egymás után fog lefutni, tehát a
második megvárja az elsőt
$("#adatok").animate({ marginTop: '100px' }, 400).animate({ marginTop: 0 }, 300);
```

B) Párhuzamos animációk

```
// Párhuzamos animációk, két külön elemen futtatott animáció párhuzamosan történik meg
$("#adatok").animate({ marginTop: '100px' }, 400)
            .animate({ marginTop: 0 }, 300);
$("li").animate({ fontSize: '30px', opacity: 0.5 }, 300)
       .animate({ fontSize: '12px', opacity: 1 }, 400);
```

C) Az animate paraméterei

```
// Relatív módosítás, azaz mindig 100-al növeli a felső margót. A harmadik paraméter a
csillapítás, mely alapesetben a "linear", de további pluginek nélkül még egy
választható, a "swing". A negyedik paraméter egy callback függvény, mely akkor kerül
meghívásra, amikor az animáció lefutott
$("#adatok").animate({ marginTop: '+=100' }, 400, "swing", function() {
    alert("Véget ért a móka mára.");
});
```

Animációk

- Mai böngészőkben a JavaScriptes animációknál sokkal hatékonyabb a CSS3 animáció
- Ha lehet, ezt használjuk
- CSS3
  - transform (nem animáció)
  - transition: átmenet két stílusállapot között
  - animation: keyframe alapú, ismétlődő

Animációk


```
.doboz {
  width: 100px;
  height: 100px;
  border: 2px solid orange;
  transition: width 1s ease-in-out 0s;
}
.doboz:hover {
  width: 500px;
}
```

```
<div id="doboz" class="doboz"></div>
```

Kiterjedés és pozíció

- A magasság, szélesség és elhelyezkedés kérdése relatív: honnan mérjük, padding, border kell-e
  - height, innerHeight, outerHeight (+width változatok)
  - position, offset
  - scrollTop, scrollLeft

```
$("#adatok").css("height");// => számított magasság
$("#adatok").height();     // => ugyanaz mint az előbbi, de number-el tér vissza
$("#adatok").innerHeight();    // => number, magasság+padding
$("#adatok").outerHeight();    // => number, magasság+padding+border (+margin,
                                    egy true paraméter esetén)

$("#adatok").position(); // => {top: 275, left: 10}, elempozíció, relatív a
                                szülőhöz
$("#adatok").offset();   // => {top: 280, left: 13}, elempozíció, relatív a
                                dokumentumhoz
```

HTML manipulációja

- Sokféle metódus létezik a HTML struktúra módosításához
  - elem létrehozása (jQuery() függvény)
  - elem lemásolása (clone)
  - elem hozzáadása, mozgatása (append, prepend, after, before, appendTo, prependTo, insertAfter, insertBefore, replace)
  - elem törlése (remove, detach)
  - tartalom kezelése (html, text)
  - attribútumok kezelése (attr)

HTML manipulációja

A) Új jQuery objektumok létrehozása

```
var $a = $("<a class='12a' href='index.html'><b>Béla, Bulcsú</b></div>");
// Hasonló elemdefiníció attribútum-objektummal:
var $a = $("<a />", {
                classNam: '12a',
                href: 'index.html',
                html: '<b>Béla, Bulcsú</b>'
            });
```

B) Attribútum és tartalom módosítása

```
$("<a />")
    .addClass('12a')
    .attr('href', 'index.html')
    .html('<b>Béla, Bulcsú</b>')
    .appendTo($("body"));

$("body").find('.12a').attr('href');  // => 'index.html'
$("body").find('.12a').html();       // => '<b>Béla, Bulcsú</b>', HTML tartalom
$("body").find('.12a').text();      // => 'Béla, Bulcsú', tartalom tag-ek nélkül
```

HTML manipulációja

C) jQuery objektumok beillesztése és törlése

```
$a.appendTo($("body"));          // A $a beillesztése a body végére
$a.prependTo($("body"));         // A $a beillesztése a body elejére
$a.insertAfter($("#adatok"));    // A $a beillesztése az adatok id-jű div után
$a.insertBefore($("#adatok")); // A $a beillesztése az adatok id-jű div elé
$a.remove();                     // A $a elem törlése

// Egy alternatív szintaxissal elérhető ugyanez, ekkor a két elem szerepe
megfordul. Technikailag ugyanaz mindkettő, a környezettől függ, hogy melyik
megoldás a kézenfekvőbb.
$("body").append($a);            // A body végére illeszti a $a-t
$("body").prepend($a);           // A body elejére illeszti a $a-t
$("#adatok").after($a);          // Az adatok id-jű div után illeszti a $a-t
$("#adatok").before($a);         // Az adatok id-jű div elé illeszti a $a-t
```

Bejárás

- A kiválasztott elem(ek)hez képesti elmozdulás
  - gyerek (children), leszármazott (find)
  - szülő (parent, parents, parentsUntil), ős (closest)
  - testvér (siblings, next, nextAll, nextUntil, prev, prevAll, prevUntil)
- jQuery objektum elemeinek módosítása
  - szűrése (is, has, filter, first, last)
  - bővítése (add)
- CSS szelektorok használata ehhez is
- Kerüljük a bonyolult struktúrájú bejárásokat!

Bejárás

A) Bejárás

```
$("#adatok").parent();    // szülő kiválasztása, most a body jQuery objektumával tér vissza
$("li").next();             // a következő elem kiválasztása, az elsőt kivéve az összes li
$("li").prev();             // az előző elem kiválasztása, utolsót kivéve az összes li
$("li").prev(".adat");      // az előző elem kiválasztása, de csak ha adat osztályú
$("b:first").closest('div'); // az elem felmenői közül az első, amelyikre igaz, hogy div
$("#adatok").find(".adat");  // az elem leszármazottai, melyekre igaz, hogy adat osztályúak
$("li").first();           // a kiválasztott elemek közül az első, jelen esetben az első li
$("li").last();             // a kiválasztott elemek közül az utolsó
$("li").eq(2);              // a kiválasztott elemek közül a harmadik!
$("#adatok").siblings("h1"); // az elem azon testvérei, melyek h1 típusúak
```

B) Szűrés

```
$("#adatok").add("li");      // az elem jQuery objektumához hozzáadjuk az összes li-t is
$("#adatok, li").not("li");  // a választásból kivesszük az összes li típusút
$(".adat").filter("img");    // a kiválasztott elemekből csak a képek maradnak
```

C) Azonosítás

```
$("li:first").is(".adat"); // => true, az elem megfelel e a feltétel CSS szelektornak
$("li:first").is("div");   // => false
$("div").has("b");           // azon elemek, melyeknek van olyan leszármazottjuk, melyre
                               igaz a feltétel; most csak az adatok azonosítóval
                               rendelkező div
```

Visszalépéses szelekció

- Láncolás során visszaléphetünk az előzőleg kiválasztott elemekhez a jQuery objektum módosítása után
- end metódus
- behúzással jelezni

```
$("#adatok")                                                    // hatókör: div#adatok
    .find("li")                                                 // hatókör: div#adatok
        .css({ padding: "5px" })                                // hatókör: div#adatok li
        .find("b")                                              // hatókör: div#adatok li
            .html('Csak a címeket hackoltam meg!!!‘)  // hatókör: div#adatok li b
            .end()                                              // hatókör: div#adatok li b
        .end()                                                  // hatókör: div#adatok li
    .animate({ paddingTop: '+=100' }, 200);                     // hatókör: div#adatok
```

Iterálás a szelekcióban

- Kiválasztott elemekkel speciális művelet végzése
- each metódus
  - anoním vagy nevesített függvénykifejezés
  - this az adott elem DOM objektuma → $(this)

```
var cimkek = [];
$("b").each(function() {
    cimkek.push( $(this).text().replace(':','') );
});

cimkek.join(', ');
// => "Leánykori név, Születési dátum, Végzettség, Foglalkozás"
```

Eseménykezelés

Eseménykezelés általában

- HTML elem számos eseménnyel rendelkezik
- Nagyon sok esemény következik be az oldal használata során
- Ezekhez csatolhatjuk a kódunkat (feliratkozunk)
- Azt a függvényt, amit az eseményhez kötünk eseménykezelőnek hívjuk
- DOM leglabilisabb része
  - Böngészőbeli nagy eltérések
  - Legalább négyféle eseménykezelési modell

Eseménykezelés lépései

- Hogyan tudok feliratkozni az eseményre?
- Hogyan érem el az eseményobjektumot?
- Milyen tulajdonságai vannak az eseményobjektumnak?
- Hogyan akadályozom meg az alapértelmezett műveletet?
- Hogyan állítom meg a bubbling-ot?

Eseménykezelés jQueryben

- Absztrakt réteget definiál, amely elfedi a különbségeket
- Metódusok
  - click, dblclick, mouseenter, mousemove, keydown, stb.
  - paraméterük egy függvénykifejezés
  - függvényen belül a this az eseményt kiváltó objektumra mutat

Egyszerű eseménykezelés

A) Eseménykezelő bekötése

```
$("#adatok").click(function() {
    console.log("Ha nem tudnád, klikkeltél!");
});
```

B) Saját elem elérése eseménykezelőben

```
$("#adatok").mouseenter(function() {
    var $this = $(this);   // általában ezzel a sorral kezdődik egy
                           eseménykezelő, érdemes lecache-elni a $(this)-t,
                           ne hozzuk létre újra és újra
    console.log("Rám vitted az egeret, Te gonosz!");
    console.log("Pedig én vagyok a híres: ", $this);
});
```

C) Egyszerű függvény használata eseménykezelőként

```
var esemenykezelo = function() {
    console.log("Helyes, helyes - levitted az egeret rólam!");
}
$("#adatok").mouseleave(esemenykezelo);
```

Általánosabb eseménykezelés

- bind metódus
  - ennek rövidített változatai az előző metódusok
  - egységesebb kód
  - 1. paraméterében akárhány eseményre feliratkozhatunk
  - megszüntetés az unbind metódussal (egységes kód)
  - névterek használata (click.névtér)
    - többszöri feliratkozás
    - csak a saját eseménykezelők leíratása
  - először leíratni, majd feliratni a névteres eseménykezelőt

bind használata

A) Függvény eseményhez kötése a bind metódussal

```
$("#adatok").bind("click", function() {
    console.log("Ha nem tudnád, klikkeltél!");
});
```

B) Eseménykezelők megszüntetése és egyszerre több definiálása

```
$("#adatok").unbind().bind("click mouseenter mouseleave", function() {
    console.log("Most mindenféle eseménykezelőt egyszerre definiáltál!");
});
```

C) Eseménykezelők létrehozásának jobb módszere, névtér használatával

```
$("#adatok").unbind("click.szovegel").bind("click.szovegel", function() {
    console.log("Szövegelek egyszer!");
});
$("#adatok").unbind("click.szovegel2").bind("click.szovegel2", function() {
    console.log("Szövegelek másodszor!");
});
```

D) Eseménykezelők megszüntetésének módjai

```
$("#adatok").unbind(".szovegel");       // A szovegel névterű eseménykezelők megszüntetése
$("#adatok").unbind("click.szovegel");  // A szovegel névterű click eseménykezelő
                                           megszüntetése
$("#adatok").unbind("click");           // Minden click eseménykezelő megszüntetése
$("#adatok").unbind();                  // Az elem összes eseménykezelőjének megszüntetése
```

Esemény delegálása

- Minden eseménykezelő egy külön objektum
- Sok eseménykezelő → jelentős memóriaszükséglet
  - pl. egy torpedójáték esetén
- Megoldás: esemény delegálása
  - egyetlen eseménykezelő írása a szülő elemen
  - átveszi az eseményt (bubbling)
  - event objektum tartalmazza az eseményt kiváltót
- delegate metódus
  - 1. paramétere egy szelektor lesz: akiknek az eseményét a szülőre delegáljuk
  - eseménykezelő pedig úgy viselkedik, mintha az eredeti elemen váltódott volna ki

delegate

A) Esemény delegálása a szülőről a gyerekeire

```
$("#torpedoTabla").delegate('a', 'click', function(e) {
    // A this arra az <a> elemre mutat, amire kattintottunk, az esemény
       teljes egészében át lett adva a linknek, holott - elvileg - a
       #torpedoTabla elem eseménykezelője
    console.log("Kattintás egy tábla következő négyzetén: ", $(this));
});
```

B) Saját névtérrel ellátott esemény delegálása és ennek megszüntetése

```
$("#adatok")
    .undelegate('li', 'click.delegalas‘)   // delegált eseménykezelő
                                              megszüntetése
    .delegate('li', 'click.delegalas', function() {
        console.log("Klikkelés a következő <li> elemen:", $(this));
    });
```

A delegálás előnyei

- Előnyei
  - hatékonyabb
  - a szülőn belül dinamikusan létrehozott újabb elemekre is meghívódik az eseménykezelő

Egységes eseménykezelés

- Az 1.7-es verziótól az előző eseménykezelési metódusok egyetlen metódusban lettek egységesítve
- on/off metódus
  - .on(events [, selector] [, data], handler(eventObject))
  - .off(events [, selector] [, handler(eventObject)])

on() és off()


```
//Közvetlen eseménykezelés
$('a').bind('click', myHandler);
$('a').on('click', myHandler);
$('form').on('submit', { val: 42 }, fn);
$('a').on('click.myPlugin', myHandler);
$('a').off('click.myPlugin');
//Delegált eseménykezelés
$('.comment').delegate('a.add', 'click', addNew);
$('.comment').on('click', 'a.add', addNew);
$('.comment').on('click.myDlg', 'a', addNew);
$('.comment').off('click.myDlg', 'a');
//Live kezelés
$('a').live('click', fn);
$(document).on('click', 'a', fn);
$('a').die('click');
$(document).off('click', 'a');
```

Az event objektum

- Eseményobjektum elérése az eseménykezelő függvény 1. paramétereként történik
- Tulajdonságok
  - which
  - pageX, pageY
  - target
  - type
- Metódusok
  - preventDefault(): alapértelmezett műveletek bekövetkezése akadályozható meg

Az event objektum

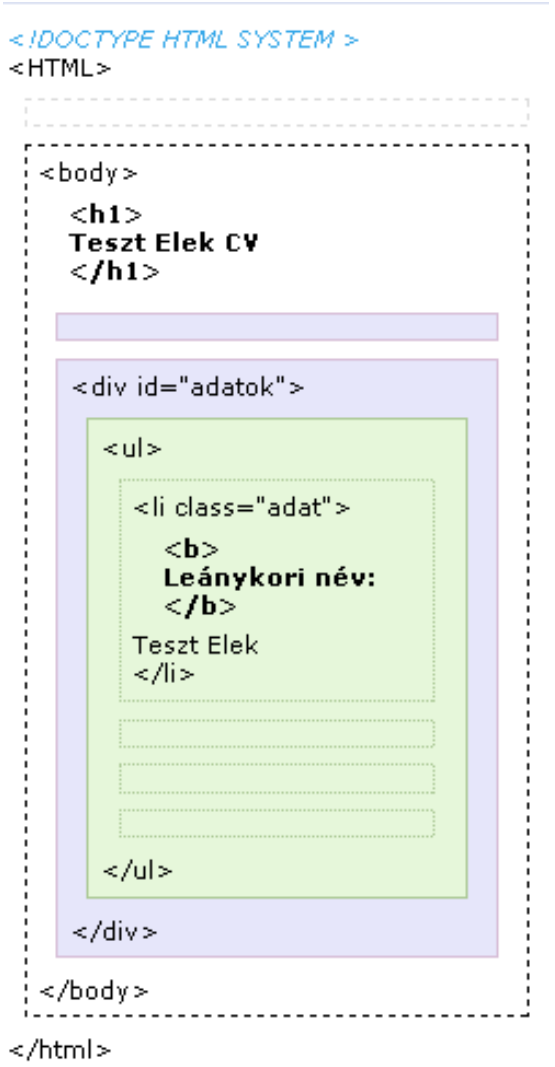
- Pl. linkre kattintáskor a hivatkozott oldalra ugrásának megakadályozása

```
$("a")
    .unbind(".pelda")
    .bind("click.pelda mouseover.pelda mouseleave.pelda", function(e) {
        var $this = $(this);
        console.log(e.pageX, '-', e.pageY);    //  197 - 334

        if (e.type == 'click') {
            console.log('KliK');
            e.preventDefault();    // böngészőesemény tiltása;
                                   klikkelés eseményre a link nem fog
                                   átdobni a href-ben megadott oldalra
        }
    });
```

Bubbling

- Az események a legbelső elemtől a legfelső szintig, a document-ig egymás után váltódnak ki.
- Alapesetben a document-ig minden esemény eljut
- Bubbling megszakítható az event objektum stopPropagation() metódusával.



Bubbling


```
$("li:first").bind('click', function(e) {
    console.log('1. li');
});
$("ul").bind('click', function(e) {
    console.log('2. ul');
});
$("#adatok").bind('click', function(e) {
    console.log('3. div#adatok');
    e.stopPropagation();                    // az event bubbling megakasztása
});
$(document).bind('click', function(e) {
    console.log('4. document');
});
/* Konzolban:
 *      1. li
 *      2. ul
 *      3. div#adatok
 *
 *  A document kattintás eseményéig nem jut el az eseménykezelő, mert a
div#adatok kattintás eseménykezelőjében a stopPropagation függvény megakasztja
 */
```

jQuery segédfüggvények

Adatmentés a DOM-ba

- A DOM elérése lassú
- jQuery saját megoldást fejlesztett ki
- data metódus
- Saját szelektor-objektumába menti az adatokat
- Legtöbbször jQuery szelektorok tárolására használjuk
  - cachelés
  - lokális vagy globális változó helyett
- HTML 5 data attribútumokat is támogatja

Adatmentés a DOM-ba

A) Tárolás a data segítségével

```
$("#adatok").data("adatokSzama", 10);    // setter
$("#adatok").data("adatokSzama");        // => 10, getter

// Tetszőlegesen bonyolult adatszerkezet tárolható:
$("#adatok").data("adatCimek", [ 'Leánykori név', 'Születési dátum'
]);
$("#adatok").data("adatCimek");
// => ["Leánykori név", "Születési dátum"]
```

B) Szelektor cachelése szülő elemben

```
var $adatok = $("#adatok");
$adatok.data("nev", $adatok.find(".adat:first"));

$adatok.data("nev").text();   // => "Leánykori név: Teszt Elek"
```

Egyéb jQuery segédfüggvények

- $.trim()
- $.type()
- $.extend()
  - objektumok bővítése, pl. alapértelmezett értékekkel

trim és type

A) Szövegek trimmelése

```
$.trim("    Ez egy sztring sok    space-el!  ");
// => "Ez egy sztring sok space-el!"
```

B) Típusazonosítás

```
typeof null;                    // => 'object'
$.type(null);                   // => 'null'
$.type(null) === "null";        // => true
$.type('123');                  // => 'string'
$.type(123);                    // => 'number'
$.type(NaN);                    // => 'number'
$.type(true);                   // => 'boolean'
$.type(function(){});           // => 'function'
$.type([]);                     // => 'array'
$.type({});                     // => 'object'
```

extend


```
var ablakotNyitokJol = function(opts) {
    var alapBeallitasok = {
        cim: 'Ablak jól',
        szelesseg: 500,
        magassag: 200,
        atlatszosag: 0.8,
        bezaraskor: function() {
            alert('Ablakot jól bezártam!');
        }
    }
    opts = $.extend(alapBeallitasok, opts);
    console.log(opts.cim, opts.magassag, opts.szelesseg);
    opts.bezaraskor();
}

ablakotNyitokJol({
    cim: "AblakZsiráf",
    magassag: 300,
    bezaraskor: function() { alert('Bezártam az ablakot!'); }
});
// Konzolon: AblakZsiráf 300 500
// Majd megjelenik a "Bezártam az ablakot!" szöveg a figyelmeztető ablakban
```

Optimalizációs megfontolások

jQuery optimalizációk, hatékony függvények closure segítségével

Hatékony jQuery szelektorok

- Lassú: rosszul megválasztott vagy felesleges szelektorok
- Gyors: egyszerű id, elem vagy osztály szelektorok
- Szabályokat jobbról balra értelmezi
  - $("#adatok .adat") vs $(".adat")
- Használjunk szűkítő metódusokat, filterezést
  - $("#adatok").find(".adat")
- Kerüljük az attribútumokra és a pszeudo-szelektorokra való szűrést

Hatékony jQuery szelektorok


```
$(".adat", "#adatok")
$("ul .adat");                                  // Nem hatékony
$("ul").find(".adat");                          // Hatékonyabb

$("#adatok li.adat");                           // Nem hatékony
$("#adatok").find("li").filter(".adat");        // Hatékonyabb

$(".adat:first");                               // Nem hatékony
$(".adat").first();                             // Hatékonyabb

$(".adat:nth-child(2)");                        // Nem hatékony
$(".adat").eq(2);                               // Hatékonyabb

$("input:checked");                             // Nem hatékony
$("input").filter(":checked");                  // Hatékonyabb
```

Elemek animálása

- A böngészőnek számos esetben újra kell rajzolni vagy újra el kell helyezni az elemeket. Ez egy lassú folyamat.
- Animálásnál esetenként sok száz módosítás lehet.
- Az animált elemek vagy szüleik legyenek abszolút pozíciójúak, így a böngészőnek nem kell az összes relatív szülő és testvér elem pozícióját újraszámolni.

Hatékony HTML manipuláció

- A DOM lassú, ezért csak a szükséges módosításokat végezzük el benne
- Új elemeket lehetőleg ne több lépésben adjuk hozzá, hanem egyben
- Több CSS módosítás helyett használjunk stílusosztályt
- Győződjünk meg, hogy van kiválasztva elem, mielőtt végrehajtanánk rajta metódusokat (felesleges függvényhívások)
- remove helyett detach (leválasztja és megőrzi a tulajdonságokat és az eseménykezelőket), és csak a jQuery objektumon dolgozunk, majd visszarakjuk

Hatékony HTML manipuláció

A) Gyorsabb struktúrabeillesztés a DOM-ba

```
var $adatok = $("#adatok"),
    $ujStruktura = $("<div />");
for (var i=0, $div; i<10; i++) { $ujStruktura.append("<div>"+i+"</div>"); }
$ujStruktura.appendTo($adatok); // szövegösszefűzéssel hatékonyabb megoldás írható
```

B) Osztálycserével sok DOM-módosítást spórolhatunk

```
$(".adat").css({ color: 'red', fontSize: '14px', lineHeight: '16px' });
// E helyett egy CSS osztályra való csere optimálisabb
// .kiemelt { color: red; font-size: 14px; lineHeight: 16px; }
$(".adat").addClass('kiemelt');
```

C) Nem létező elem vizsgálata a felesleges metódushívások elkerülésére

```
$(".nemLetezoElem").css({ width: '100px'}).html('Nem létezek?');
// A fenti helyett optimálisabb ha megvizsgáljuk a szelekció sikerességét
var $nemLetezo = $(".nemLetezoElem");
if ($nemLetezo.length) { $nemLetezo.css({ width: '100px'}).html('Nem létezek?'); }
```

D) A detach metódussal leválasztva már gyorsabban módosíthatjuk a HTML struktúra részeit

```
$("#adatok").css({ width: '100px', color: 'red', opacity: 0.5 }).html('Izébizé');
$("#adatok").detach()
            .css({ width: '100px', color: 'red', opacity: 0.5 }).html('Izébizé')
            .appendTo($("body"));
```

Hatékony függvények closure-rel

A) Egyszerű adattár

```
// Minden futtatáskor megkeresi a
.adat osztályú elemeket
var adatTar = function() {
    var $adatok = $('.adat');
    return $adatok;
};
adatTar();
```

B) Adattár closure-rel

```
// Csak egyszer keresi meg az
adatokat, mivel önkioldó, így az
adatTar függvény létrehozásakor
var adatTar = (function() {
    var $adatok = $('.adat');
    return function() {
        return $adatok;
    }
})();
adatTar();
```

C) Lazy loading

```
var adatTar = (function() {
    var $adatok;
// Biztosítja, hogy az adatok a
closure-ben maradnak, mivel a $adatok
a külső függvény lokális változója
    return function() {
        if (!$adatok) $adatok =
$('.adat');    // Egyszer keres, de
csak akkor, mikor először hívjuk meg a
függvényt - ekkor kimenti az adatokat
a closure-be
        return $adatok;
    }
})();

adatTar();
```

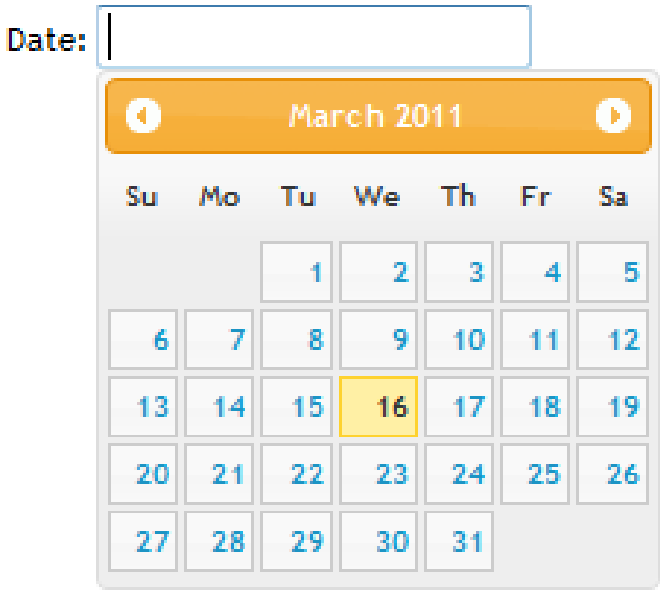
jQuery UI

jQuery UI

- Alacsony szintű interakciók és animációk
  - Draggable
  - Droppable
  - Resizable
  - Selectable
  - Sortable
- Effektek
- Magas szintű komponensek témákkal
  - Accordion
  - Autocomplete
  - Button
  - Datepicker
  - Dialog
  - Progressbar
  - Slider
  - Tabs

jQuery UI példa

- <input id="datepicker" type="text">
- $("#datepicker").datepicker();



Hivatkozások

Hivatkozások

- jQuery
  - http://jquery.com/
  - http://jqfundamentals.com/
- jQuery UI
  - jQuery User Interface
  - http://jqueryui.com/
  - http://jqueryui.com/demos/
  - Tessék átnézni a lehetőségeket!
  - Részletes a dokumentáció