G. Fejezet

JavaScript

A JavaScript, mint a neve is jelzi, a Java nyelv szkript változata. Erőssége abban rejlik, hogy míg szinte a Java nyelv összes lehetőségével rendelkezik, szkript tulajdonsága miatt forrás szinten lehet beágyazni HTML oldalba, így egy interaktív HTML oldal készítésekor nem válik külön magának az oldalnak és az interaktivitást megvalósító vezérlésnek a megírása. Nem csoda hát, hogy több szkriptszerű nyelv is született már (például Jscript, VBScript, mind az MS-től), de ez idáig lehetőségei miatt a JavaScript látszik a legelterjedtebbnek.

A JavaScriptet a Netscape 2.0 böngésző megjelenésével vezették be, ezért még mindig erősen Netscape-specifikus, de manapság már megkezdődött a szabványosítása. Maga a nyelv a következő rétegekből tevődik össze:

- JavaScript alapnyelv (Core) : ez definiálja a nyelv szintaxisát és alapelemeit, például a vezérlőszerkezeteket, típuskonstrukciókat, alaptípusokat stb.. . .
- Kliensoldali JavaScript : az alapnyelvet böngészők vezérlését lehetővé tevő objektumokkal, illetve egy HTML oldalt leíró dokumentum objektummodellel (Document Object Model (DOM)) egészíti ki. Különböző verzióinak használatához a következő verziójú böngészők szükségesek:
  - 1.0 : Navigator 2.0 - vagy ezzel ekvivalens,
  - 1.1 : Navigator 3.0 - vagy ezzel ekvivalens,
  - 1.2 : Navigator 4.0 - vagy ezzel ekvivalens,
  - 1.3 : Navigator 4.06 - vagy ezzel ekvivalens,

  Az 1.3-as változat teljesen kompatibilis az ECMA (European Computer Manufacturers Association - Európai Számítógépgyártók Szövetsége) 262-es szabványával (ECMAScript), amit az ISO is elismer (ISO-16262).
- Szerveroldali JavaScript : az alapnyelvet szervereken végezhető funkciók (például adatbázis-elérés, fájlműveletek stb.. . .) vezérlését lehetővé tevő objektumokkal egészíti ki.

Terjedelmi okok miatt mi csak az 1.3-as változat kliensoldali lehetőségeivel fogunk foglalkozni.

A G.1.. táblázat a JavaScript és a Java áttekintő összehasonlítását mutatja, minden benne szereplő fogalomról a későbbiekben még bővebben lesz szó.

> *Megjegyzés*: Ezen fejezetnek nem célja a JavaScript, illetve a HTML részletes ismertetése. Teljes leírásért lásd a [SM-Scr 99] címet.
> A továbbiakban a szintaxisleírásokban szögletes zárójelbe tett elemek használata opcionális.

G.1. JavaScript beágyazása

A JavaScriptet három helyen is lehet használni egy HTML dokumentumon belül:

- A <SCRIPT> </SCRIPT> HTML kulcsszavak közt utasítások végrehajtására és funkciók definiálására.
- HTML elemeknél eseménykezelők megírására.

	JavaScript	Java (applet)
Beágyazás:	A forrás a HTML oldalba van beágyazva.	Egy applet elkülönül a HTML oldaltól, csak a vezérlése történik onnan.
Végrehajtás:	A kliensgép böngészőprogramja interpretálja a HTML oldalba beágyazott forrásszöveget.	A szervergépen előzőleg lefordított és onnan letöltött bináris bájtkódot a kliens gép interpretálja.
Programok felépítése:	Utasítások és eseménykezelők halmaza.	Osztályok definíciója.
Nyelv típusa:	Objektum-(prototípus-) alapú.	Objektumorientált (osztályalapú).
Típusosság:	Laza: típusokat nem kell deklarálni.	Erős.
Típusellenőrzés:	Dinamikus: objektumreferencia típusellenőrzés csak futási időben történik.	Statikus: objektumreferencia típusellenőrzés már fordításkor megtörténik.

G.1. ábra: JavaScript-Java összehasonlítás

- HTML elemek paramétereinek megadásakor. Ahol egy paraméterértéket kell megadni, ott állhat JavaScript kifejezés is a következő formában:
  paraméternév=&{*JavaScript kifejezés*};
  Például a <HR WIDTH="&{szazalek()};" ALIGN="LEFT"> a HTML oldal szélessége annyi százalékánál húz vonalat a bal oldalra, amennyit a szazalek() (a fájlban korábban már definiált) JavaScript függvény visszaadott.

Megjegyzendő, hogy a JavaScript a HTML-lel szemben különbséget tesz kis- és nagybetűk között.

A <SCRIPT> HTML kulcsszó

A <SCRIPT> </SCRIPT> HTML kulcsszavak közé tett JavaScript a HTML oldal letöltése után, de még a megjelenítés előtt kerül kiértékelésre. A kiértékelés alatt a definiált függvények eltárolódnak, az utasítások pedig végrehajtódnak. A szintaxis a következő:

```
<SCRIPT
       [LANGUAGE=szkriptnyelv]
       [SRC=fájl]
>
JavaScript utasítások
[<NOSCRIPT>
HTML utasítások
</NOSCRIPT>]
</SCRIPT>
```

- language = a szkript nyelvét adja meg. Jelenleg a következő értékeknek van jelentésük:
  - "JavaScript" - a JavaScript alapverzióját jelöli. Ilyen szkripteket már a Netscape 2.0 is végre tud hajtani.
  - "JavaScript1.1" - a JavaScript 1.1-es verzióját jelöli. Ilyen szkripteket csak a legalább 3.0-ás Netscape tud végrehajtani.

  - "JavaScript1.2" - a JavaScript 1.2-es verzióját jelöli. Ilyen szkripteket csak a legalább 4.0-ás Netscape Communicator tud végrehajtani.

- src = a szkriptet tartalmazó fájl URL címe. A fájl egyszerű ASCII formátumú, csak a szkript szövegét tartalmazhatja, rendszerint .js kiterjesztéssel. Ekkor a HTML szervert úgy kell felkonfigurálni, hogy a .js fájlokat application/x-javascript MIME típusúként[1] ismerje fel (ez persze csak távoli webszerveren található szkriptfájl megadása esetén követelmény). Ezen attribútum megadása esetén a <SCRIPT> </SCRIPT> kulcsszavak között minden JavaScript sor figyelmen kívül marad.

A <NOSCRIPT> </NOSCRIPT> kulcsszavak közötti HTML szöveg csak akkor fog látszani, ha a böngésző nem képes JavaScript értelmezésére vagy az nincs engedélyezve. Célszerű a <SCRIPT> </SCRIPT> kulcsszavak között álló szkript teljes szövegét HTML megjegyzéselemek közé tenni, nehogy a régebbi böngészőprogramok, melyek még nem ismerik a <SCRIPT> kulcsszót, megjelenítsék magát a szkript szövegét.

```
<SCRIPT>
<!-- Itt egy HTML megjegyzésbe rejtett JavaScript szkript kezdődik
...
// és itt a HTML megjegyzésbe rejtett JavaScript szkript vége -->
<NOSCRIPT>
<BLINK>Ez a böngésző nem képes JavaScript megjelenítésére!</BLINK>
Töltsön le egy jobbat
<A HREF="http://home.netscape.com/comprod/mirror/index.html">
innen
<IMG SRC="NSNow.gif">
</A>!
</NOSCRIPT>
</SCRIPT>
```

G.2. A JavaScript nyelv leírása

Ebben a pontban a JavaScript alapnyelvet fogjuk ismertetni. Ennek ismerete szükséges mind a kliensoldali, mind a szerveroldali JavaScript alkalmazásakor.

Típusok

A JavaScript gyengén típusos nyelv. Ez pontosan azt jelenti, hogy egy azonosító használatakor annak típusát nem kell megadni, sőt később ez a típus meg is változhat. Az ismert alaptípusok a következők:

- Szám: nincs különbség az egész és a lebegőpontos számok között. Így egy számot a következő formákban lehet megadni:

  - Lebegőpontosként: tizedesponttal vagy kitevős alakban.
  - Decimális egészként: az első számjegy nem 0.
  - Oktális egészként: az első számjegy mindig 0.
  - Hexadecimális egészként: az első két karakter 0x.

  Megjegyezzük, hogy az egész számok ábrázolása 32, a lebegőpontosoké pedig 64 biten történik.

[1] A MIME típus a webszerverről letöltött dokumentum tartalmának a feldolgozását segítő típusinformáció, melyet a webszerver a dokumentummal együtt ad vissza.

- Logikai (boolean): csak true vagy false értéke lehet.
- Szöveg: a Java String típusával teljesen megegyezik, azaz itt is objektumként történik a megvalósítása. A következő escape szekvenciák használata lehetséges:
  - `\b` : A visszatörlés (backspace) karaktert jelöli.
  - `\f` : A lapdobás karaktert jelöli.
  - `\n` : A soremelés karaktert jelöli.
  - `\r` : A kocsivissza (carriage return) karaktert jelöli.
  - `\t` : A tabulátor karaktert jelöli.
  - `\'` : Az aposztróf karaktert jelöli.
  - `\"` : Az idézőjel karaktert jelöli.
  - `\\` : A \ karaktert jelöli.
  - `\`*nnn* : Oktálisan megadott ASCII kódú karaktert jelöl.
  - `\x`*nn* : Hexadecimálisan megadott ASCII kódú karaktert jelöl.
  - `\u`*nnnn* : Hexadecimálisan megadott UNICODE karaktert jelöl. Megjegyzendő, hogy a JavaScriptben csak sztringeken és megjegyzéseken belül lehet Unicode karaktereket használni.

  A szimpla idézőjelek is szöveget határolnak, mivel nincs karakter típus.
- Tömb: tömböt egy Array objektum reprezentál. Csak egydimenziós tömb létezik, az elemeket pedig szögletes zárójelek közt lehet megadni. Tömbelemekre hivatkozni szögletes zárójelbe tett indexekkel lehet, az indexelés 0-tól indul.
- null: az üres referenciát jelzi.
- undefined: ismeretlen, eddig nem definiált értéket jelez. Változók értékének összehasonlítására használva megtudhatjuk, hogy adott változó inicializálva lett-e.
- Objektum.

Típuskonverziók

- Mivel csak egyféle számtípus van, ezért például egész számokkal végzett osztás esetén (ha az eredmény megkívánja) a művelet értéke már lehet lebegőpontos szám is.
- A szöveggé történő konverzió a "+" operátor használata esetén automatikus.
- Szövegből számot a parseInt(String) és parseFloat(String) függvények állítanak elő, melyek visszatérési értéke az első karaktertől kezdve a lehető leghosszabban értelmezhető szám, illetve NaN (ami egy érvénytelen számot jelölő konstans), ha már az első karakter sem számjegy. Ezen konverzió minden matematikai operátor (kivéve "+") használata esetén automatikus.
- Logikai kifejezésekben az üres szöveg, illetve a 0 a false (hamis), míg nem üres szöveg/objektum, illetve nem 0 szám a true (igaz) értéknek felel meg. Logikai kifejezés értéke mindig az utoljára kiértékelt részkifejezés típusának fog megfelelni, azaz például:

```
false || 4 == 4
"blabla" && 5 == 5
"blabla" || 5 == "blabla"
"" || true == true
```

- A null érték logikai környezetben a false értéknek, míg numerikus környezetben 0-nak felel meg.
- Az undefined használható logikai értékként is, ilyenkor a false értéknek felel meg.
- Minden objektum típusa ekvivalens egymással.

Azonosítók

A Java Unicode előfordító hiánya miatt a JavaScript azonosítók neveinek a következő szintaxist kell követniük:

- A név első karaktere csak betű vagy az aláhúzás karakter lehet.
- A többi karakter vagy betű, vagy számjegy vagy az aláhúzás karakter lehet.

Egy betű az angol ABC kis- és nagybetűi közül bármelyik lehet. A kis- és nagybetűk egymástól különbözőnek számítanak.

Változók

Változót nem kell deklarálni, definíciójakor pedig nem kell a típusát explicit módon megadni. Minden értékadáskor a változó típusa is megváltozhat. Adott blokkra lokális változó első definíciójakor ki kell tenni a változó neve elé a var kulcsszót. Ennek hiányában az első definíció – az összes blokkon kívül pedig még var kulcsszó használatakor is – globális változót hoz létre. Inicializálatlan változó alapértelmezés szerint az undefined értéket tartalmazza. Inicializálatlan globális változóra történő hivatkozás hibát vált ki, míg inicializálatlan lokális változó esetén a kiértékelés undefined, illetve numerikus környezetben NaN-t eredményez. Globális változókat más HTML oldalakba ágyazott JavaScriptből is el lehet érni.

Operátorok

Az operátorok használata és prioritása nagyrészt megegyezik a Java operátorokéval. A különbségek a következők:

- Mivel az operátorok kiértékelésére mindig az automatikus típuskonverzió után kerül sor, ez zavaró lehet a == és != használata esetén (például 3=="3" teljesül). Ezért bevezették a === és !== operátorokat, melyek kiértékelésekor elmarad az automatikus típuskonverzió (ekkor már 3==='3' nem teljesül, azaz 3!=="3").
- Használható a C-ben megszokott vessző operátor, amely egyszerűen kiértékeli mind az előtte, mind az utána következő kifejezést, és eredményül az utóbbi értékét adja vissza.
- A delete operátorral adott globális változót, objektumot vagy annak mezőjét, illetve egy tömbelemet lehet törölni. Visszaadott logikai értéke jelzi, hogy a törlés sikeresen végrehajtódott-e. Például beépített objektumok mezőit nem lehet törölni.
- A typeof operátor az argumentumáról (annak kiértékelése nélkül) szöveges formában visszaadja annak típusát. Ennek segítségével például le lehet kérdezni, hogy egy adott mező/változó definiálva van-e. Ellenkező esetben ugyanis a typeof eredménye undefined lesz.
- A void operátor az argumentumának megadott kifejezést kiértékeli, de nem ad vissza semmi értéket. Ez például akkor lehet hasznos, ha valamilyen környezetben visszaadott érték automatikusan feldolgozásra/megjelenítésre kerülne, akkor a visszatérési értéket ezen operátor használatával lehet letiltani.

Operátorok prioritása

A következő lista az operátorokat azok csökkenő prioritási sorrendjében sorolja fel, azaz minél előrébb áll egy operátor a listában, annál nagyobb a prioritása.

- Zárójelezés

- Objektum mezőjére hivatkozás: `.` `[]`
- Objektum létrehozása: `new`
- Függvény meghívása: `()`
- Unáris operátorok: `!` (logikai tagadás) `~` (bitenkénti invertálás) `-` `+` (előjel) `++` `--` (eggyel növelés/csökkentés) `typeof` `void` `delete`
- Multiplikatív operátorok: `*` `/` `%` (maradékképzés)
- Additív operátorok: `+` `-`
- Bitenkénti eltolás: `<<` `>>` `>>>`
- Relációs operátorok: `<` `<=` `>` `>=`
- Egyenlőségvizsgálat: `==` `===` `!=` `!==`
- Bitenkénti ÉS: `&`
- Bitenkénti kizáró-VAGY: `^`
- Bitenkénti VAGY: `|`
- Logikai ÉS: `&&`
- Logikai VAGY: `||`
- Feltételes kifejezés: `?:`
- Értékadás: `=` `+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `>>>=` `&=` `^=` `|=`
- Vessző operátor: `,`

Utasítások

JavaScriptben használható minden szabványos Java utasítástípus is. Sőt a nyelv szkript / objektumalapú mivolta miatt a JavaScript utasításainak halmaza és azok megengedett szintaxisa bővebb a Javabeli megfelelőiknél.

- Feltételes utasítások
  - `if` (*feltétel*) *utasítás1*[`else` *utasítás2*] : megjegyzendő, hogy a feltétel kiértékelésekor (ha szükséges) automatikus típuskonverzió hajtódik végre, ami meglepő eredményekhez vezethet, például `if (new Boolean(false))` esetén az igaz ág fog végrehajtódni.
  - `switch` (*kifejezés*) {
    `case` *érték* : *utasítások* [`break`]
    ...
    [`default` : *utasítások*]
    } : A Javához hasonlóan itt is az az ág fog végrehajtódni, amelyet a *kifejezés* aktuális értékének megfelelő *érték* előz meg, illetve ha ilyen nincs, akkor a `default` ág. Szintén használható a `break` utasítás az aktuális ág elhagyására (különben a végrehajtás ráfuthat egy másik értékhez tartozó ágra is). Különbség a Javával szemben, hogy a kifejezés bármilyen (nemcsak egész) típusú lehet.
- Ciklusok
  - `for` ([*inicializálás*]; [*ciklusfeltétel*]; [*ciklusléptetés*]) *utasításblokk* : általános ciklusszerkezet. Először az inicializáló utasítások kerülnek végrehajtásra, majd ezután kerül sor az iterálásra. Minden iteráció előtt kiértékelődik a ciklusfeltétel, csak annak teljesülése esetén hajtódik végre a ciklusmagot képező utasításblokk. Egy egyszerű példa: `for (var i=0; i<10; i++) {ciklusmag}`
  - `while` (*ciklusfeltétel*) *utasításblokk* : elöltesztelő ciklus. A ciklusmag mindaddig végrehajtódik, míg a ciklusfeltétel igaz marad. Ennek ellenőrzése a ciklusmag végrehajtása előtt történik, tehát maga a ciklus egyszer sem hajtódik végre, ha a feltétel már kezdetben hamis.

  - do *utasításblokk* while (*ciklusfeltétel*) : hátultesztelő ciklus. A ciklusmag mindaddig végrehajtódik, míg a ciklusfeltétel igaz marad, viszont ennek ellenőrzése a ciklusmag végrehajtása után történik, tehát maga a ciklus legalább egyszer mindenképp végrehajtódik.

- Címkék

  - Minden JavaScript utasítás megjelölhető egy eléje tett címkével. A címkét a nevével és egy utána tett kettősponttal jelöljük.
  - break [*címke*] : felfüggeszti az aktuális [illetve az adott címkével megjelölt] ciklus/utasításblokk végrehajtását és a vezérlést az azt közvetlenül követő utasítás kapja meg.
  - continue [*címke*] : az aktuális [illetve az adott címkével megjelölt] ciklus újabb iterációját kezdi meg. Ez while-ciklus esetén a ciklusfeltétel újraértékelését, míg for-ciklus esetén előbb a ciklusléptetés végrehajtását jelenti.

- Objektum-manipulációs utasítások

  - for (*változó* in *objektum*) *utasításblokk* : az adott objektum minden mezőjének nevén egyszer végigmegy a kijelölt ciklusváltozó és végrehajtódik a ciklusmag. Például a következő függvény visszaadja a paraméterként kapott objektum összes mezőjének nevét és értékét:

```
function props(obj, name) {
         var result="";
         for (var i in obj)
             result+=name+"."+i+"="+obj[i]+"\n";
         return result;
}
```

  - with (*objektum*) *utasításblokk* : segítségével az utasításblokk végrehajtása alatt alapértelmezett objektumot lehet kijelölni, így ezen objektum mezőire történő hivatkozáskor nem kell annak nevét újra kitenni.

Utasításblokkokat létrehozni a Javához hasonlóan kapcsoszárójelekkel lehet. Megjegyzendő, hogy egy utasítást nemcsak a pontosvesszővel lehet lezárni, hanem ugyanilyen szerepet tölt be a sorvége is.

Megjegyzések

A Javához hasonlóan a /* */ jelpáros közé lehet megjegyzést írni. A // karakterpárral hozhatunk létre egysoros megjegyzéseket, melyek az adott sor végéig tartanak.

Függvények

A Java nyelvvel ellentétben a JavaScript — az objektumalapú szemlélete miatt — lehetővé teszi a strukturált programozásban megszokott eljárások és függvények használatát (pontosabban ezen eljárások és függvények egy névtelen, globális objektumhoz tartoznak). A C szemléletének megfelelően itt is csak függvények léteznek. A definíciót követő minden függvényhivatkozás a függvény végrehajtását jelenti. Valójában egy függvény egy Function objektumnak felel meg.

Függvény definiálása

Egy függvény definiálása csak <SCRIPT> </SCRIPT> HTML kulcsszavak közt történhet. A szintaxis a következő:

```
function név([paraméter1, ...]) {
        utasítások
        [return érték]
}
```

Érdemes az összes, több helyen is használt függvényt a <HEAD> </HEAD> HTML kulcsszavak közt definiálni, mivel a dokumentum ezen része lesz először feldolgozva, így a későbbiekben az itt definiált függvényekre történő bármilyen hivatkozáskor a függvény már ismert lesz. Nem definiált függvényre történő hivatkozás futás közben hibát vált ki.

Függvény paraméterei

Egy függvény formális paraméterlistája csak változónevekből áll, mivel a gyenge típusosság miatt ezek típusát nem kell megadni. A paraméterátadás mindig érték szerint történik, de objektum típusú paramétereknél csak az objektum referenciája kerül érték szerint átadásra, azaz egy paraméterként kapott objektumon a függvényen belül végrehajtott változtatások a függvényen kívül is láthatóvá válnak. A formális paraméterlistában szereplő paraméterek száma sem jelent semmilyen megkötést a függvény hívásakor az aktuális paraméterek számára, ugyanis a paramétereket dinamikusan is le lehet kezelni a [*függvénynév*.]arguments tömb felhasználásával. A paraméterek aktuális számát tehát az arguments.length adja meg. A következő példában ezt kihasználva tetszőleges számú szöveges paramétert fűzünk össze az első paraméterként megadott elválasztójelet felhasználva:

```
function cat(separator) {
        result=""
        for (var i=1; i<arguments.length; i++) {
               result += arguments[i]
               if (i<arguments.length-1) result += separator
        }
        return result
}
```

Függvény hívása

Függvény hívása annak nevének leírásával történik, zárójelben az aktuális paraméterlistát felsorolva. A zárójelpárt még paraméterek hiányában is ki kell tenni. Meghívásakor a függvénynek már definiáltnak kell lennie.

Függvény visszatérési értéke

Egy függvény a return utasítással tud értéket visszaadni, de ha csak eljárásként akarjuk használni, akkor a return használata nem kötelező, sőt a visszaadott érték felhasználása sem kötelező a hívó részéről.

Függvények egymásba ágyazása

A JavaScriptben lehetőségünk van függvények egymásba ágyazására. Ilyenkor a belső függvényeket csakis a beágyazó függvény használhatja, azon kívül azok nem láthatóak. Ugyanakkor a belső függvény látja a beágyazó függvény minden lokális változóját.

Objektummodell

A JavaScript objektumalapú nyelv. Egy objektum tulajdonképpen egy asszociatív tömbnek felel meg, ahol a tömb indexelése nemcsak sorszám alapján, hanem tetszőleges szöveg-kulcsszóval is történhet. Az így azonosított tömbelemek változókat és más objektumokat tartalmazhatnak, ezek az objektum mezői. Egy objektumhoz függvényt is lehet rendelni, ezek lesznek a metódusok.

Az objektumok felépítését nem osztályok írják le, hanem maga a „konstruktor"(függvény) a felelős az objektum felépítéséért. Épp ezért öröklődésről sem beszélhetünk. Egy közös őshöz hasonló funkciót lát el az Object beépített objektum, melynek mezőivel és metódusaival minden JavaScript objektum rendelkezik.

Egyes metódusok nem kötődnek objektumokhoz, azokat objektumok nélkül is meg lehet hívni az úgynevezett prototípuson keresztül. Ezen metódusokat nevezhetjük *statikus metódusoknak*.

Objektum mezői

Egy objektum mezőire a következő két módon lehet hivatkozni (itt mindig ki kell tenni a szögletes zárójeleket):

- objektum[mezőnév] vagy objektum.mezőnév : a megadott nevű mezőre hivatkozik.
- objektum[szám] : az objektum adott számmal azonosított mezőjére hivatkozik.

Minden létrehozott objektum dinamikusan bővíthető újabb mezőkkel. Új mező hozzáadását egyszerűen a mezőre történő értékadással lehet elérni. Ilyenkor csak az az egy objektum bővül a megadott mezővel, amely a mezőnek való értékadáskor aktuális volt.

Objektum metódusai

A metódusok nem mások, mint függvényt tartalmazó mezők. Így elérésük és használatuk megegyezik az objektum bármely más mezőjének használatával. Egy metóduson belül az aktuális objektumot a this kulcsszóval lehet elérni.

Konstruktorfüggvény

Egy új objektum létrehozása történhet a new operátorral, amelyet (az osztályok hiányában) egy függvényre, az ún. konstruktorfüggvényre kell alkalmazni. A new operátor létrehoz egy üres objektumot, majd ezen új objektummal meghívja a megadott konstruktorfüggvényt, amelyen belül ezen új objektumot a this kulcsszóval már el lehet érni. Így egy konstruktorfüggvényt azonos felépítésű objektumok inicializálására lehet felhasználni. Ezért is mondhatjuk azt, hogy a nyelv prototípus alapú, hisz egy konstruktorfüggvény pont egy objektum-*prototípust* ír le.

Prototípus

Azonos konstruktorfüggvénnyel létrehozott objektumok egyszerre bővíthetők további mezőkkel a prototípuson keresztül. Ez pontosan azt jelenti, hogy azonos konstruktorfüggvénnyel létrehozott objektumok azonos prototípus-objektumot kapnak. Az összes JavaScript objektum rendelkezik egy prototípus objektummal, melyet annak *prototype* mezőjén keresztül lehet elérni. Az objektum rendelkezik prototípusának minden mezőjével, azaz ennek segítségével lehet az örökléshez hasonló viselkedést megvalósítani.

A G.2.. táblázat a JavaScript prototípus-alapú szemléletmódját és az objektumalapú szemléletmódot veti össze.

JavaScript	OO
Prototípus alapú.	Osztály alapú.
Csak objektumok léteznek.	Osztály és objektum különálló fogalmak.
Objektum definíciója és létrehozása konstruktorfüggvénnyel.	Definíció az osztály definiálásakor, objektum létrehozása pedig konstruktor végrehajtásakor történik.
Objektum hierarchiát prototípus-reláció határoz meg.	Objektum hierarchiát az osztályok öröklődési hierarchiája határozza meg.
Objektumhierarchia csak egyszeres alá/fölé rendelő lehet (mivel csak egy prototípus adható meg).	Többszörös öröklődés is lehetséges.
Mezők öröklődése a prototípusláncon keresztül történik.	Tagok öröklődése az osztályhierarchián keresztül történik.
Az objektum létrehozásakor a konstruktor az alapértelmezett mezőket hozza létre. A mezők halmaza ezután dinamikusan változtatható.	Az osztálydefiníció pontosan meghatározza minden objektum tagjainak halmazát, ez dinamikusan nem változtatható meg.

G.2. ábra: JavaScript-OO összehasonlítás

Objektum inicializátor

A konstruktorfüggvények mellett új objektum létrehozására használható még az objektum inicializátor, melynek szintaxisa a következő: [*obj.változó* =]{*mezőnév*:*érték*,...} Ez tulajdonképp nem más, mint a létrehozandó objektum mezőinek, illetve azok értékeinek felsorolása. Megjegyzendő, hogy egy inicializátor minden egyes előfordulásakor újra kiértékelődik és egy új objektumot állít elő.

Objektum törlése

A többé már nem használt JavaScript objektumok (a Javához hasonlóan) automatikusan törlődnek, így a legegyszerűbb, ha egy felesleges objektumra vonatkozó minden referenciát null-ra állítunk. De lehetőségünk van egy objektum azonnali törlésére is a delete operátorral, mely igazat ad vissza, ha sikerült a törlés. Sikertelen akkor lehet egy törlési kísérlet, ha megpróbálunk például beépített objektumot törölni. Ez az operátor objektumok mezőire is alkalmazható.

Objektumkezelési példa

```
//objektum inicializátor
trabi = {neve:"trabi", kora:25}

//objektum törlése
delete trabi

//konstruktorfüggvény
function kocsi(neve, kora) {
    this.neve = neve
    this.kora = kora
}
trabi = new kocsi("trabi", 15)
skodi = new kocsi("skodi", 14)
```

```
ladi = new kocsi("ladi", 13)

//objektum bővítése mezővel
ladi.tipus = 1500

//objektum prototípus bővítése metódussal
kocsi.prototype.avul = function avul(mennyit) {
    this.kora = this.kora+mennyit
}
//metódus meghívása
ladi.avul(1)

//objektum mezőjének törlése
delete ladi.neve

//öröklés megvalósítása
function busz(neve, kora, hely) {
    kocsi.call(this, neve, kora)
    this.hely = hely
}
busz.prototype = new kocsi

buszi = new busz("ikari", 20, 100)
```

Ezek után a már definiált props függvényt meghívva a ladi objektumra a következő eredményt kapjuk:

```
ladi.kora=14
ladi.tipus=1500
ladi.avul=
function avul(mennyit) {
    this.kora = this.kora + mennyit;
}
```

Mint látható, az öröklődés megvalósításakor trükkhöz folyamodtunk: felhasználtuk az „ős” kocsi prototípus konstruktorfüggvényét. Ez még nem lenne elég, mivel így csak a meghívott konstruktorfüggvényben beállított mezőket vennénk át (de így akár több konstruktorfüggvényt is meghívhatunk, mintegy szimulálva a többszörös öröklődést). A többi mező átvételéhez be kell még állítanunk a konstruktorfüggvény helyes prototípus-objektumát is. Ennek hatására a létrehozott buszi objektum a következő mezőkkel rendelkezik:

```
buszi.neve=ikari
buszi.kora=20
buszi.hely=100
buszi.avul=
function avul(mennyit) {
    this.kora = this.kora + mennyit;
}
```

Beépített mezők

Ezen mezők tulajdonképp egy névtelen, felsőszintű (top-level) objektum mezőinek tekinthetők.

- Infinity : végtelent reprezentáló numerikus érték.
- NaN : nem-számot (Not a Number) reprezentáló numerikus érték. Semmilyen numerikus értékkel nem egyenlő (még magával sem!).
- undefined : nem inicializált változók kezdőértéke.

Beépített függvények

Ezen függvények tulajdonképpen egy névtelen, felsőszintű (top-level) objektum metódusainak tekinthetők.

- escape : a paramétereként megadott szöveget kódolja ISO-Latin-1 karakterkészletre úgy, hogy azt WWW címeknél paraméterként meg lehessen adni. Az így kódolt szöveget az unescape-pel lehet visszakódolni.
- eval : aktuális objektum figyelembevétele nélkül kiértékeli a paramétereként megadott szöveget. Ha az JavaScript utasítás volt, akkor végrehajtja az utasítást. Ha a paraméter nem szöveg volt, akkor nem történik kiértékelés, a visszatérési érték pedig maga a paraméter lesz.
- isFinite : megmondja a paramétereként megadott számról, hogy az véges-e. Csak NaN, illetve +/- végtelen esetén ad vissza hamisat.
- isNaN : megmondja a paramétereként megadott számról, hogy az *NaN*-e.
- Number : a paramétereként megadott objektumot számmá alakítja.
- parseFloat : átalakítja az első paramétereként megadott szöveget lebegőpontos számmá. Az átalakítás a szöveg leghosszabb számként értelmezhető prefixét veszi figyelembe. Ha már az első karakter sem értelmezhető számként, a visszaadott érték NaN lesz.
- parseInt : átalakítja az első paramétereként megadott szöveget egész számmá. Ha második paraméterként egy számot is megadunk, akkor ezzel az átalakításkor használandó számrendszer alapját határozhatjuk meg. Ha ezt nem adjuk meg, és az átalakítandó szöveg "0x" prefixszel kezdődik, az átalakítás hexadecimális, "0" prefix esetén oktális, egyéb esetben pedig decimális számrendszerben történik. Az átalakítás a szöveg leghosszabb számként értelmezhető prefixét veszi figyelembe. Ha már az első karakter sem értelmezhető számként, a visszaadott érték NaN lesz.
- String : a paramétereként megadott objektumot szöveggé alakítja.
- taint : a paramétereként megadott mező/változó/függvény/objektum, illetve paraméter elhagyása esetén a teljes szkript szerverre történő küldését csakis a felhasználó explicit engedélyével teszi lehetővé.
- unescape : a paramétereként megadott kódolt URL szövegparamétert visszakódolja szöveggé.
- untaint : a paramétereként megadott mező/változó/függvény/objektum, illetve paraméter elhagyása esetén a teljes szkript szerverre történő küldését újra lehetővé teszi a felhasználó explicit engedélye nélkül.

Beépített prototípusok

Beépített prototípusok alatt azon JavaScript prototípusokat értjük, melyek a nyelv magjának részét képezik, tehát nem keverendők össze a HTML oldalt leíró kliensoldali objektumokkal. Azokról később még részletesen szólunk.

Object : az „ősprototípus”

Mint már említettük, ezen Object objektum mezőivel és metódusaival minden JavaScript objektum rendelkezik.

Objektumot kétféleképp lehet létrehozni:

1. new Object() : konstruktorfüggvény.
2. {} : inicializátor.

Egy objektum alapértelmezett mezői a következők:

- constructor : az objektumot létrehozó konstruktorfüggvény.
- prototype : az objektum prototípusa.

Objektum alapértelmezett metódusai a következők:

- toSource() : szövegként visszaadja az objektum inicializátorát. Ez a forma csak az objektum saját mezőit és metódusait fogja tartalmazni, a prototípuson keresztül megadottakat nem.
- toString() : szövegként visszaadja az objektum tartalmát. Ez a metódus hívódik meg minden olyan esetben, amikor az objektumot szöveges formára kell konvertálni.
- valueOf() : primitív értékként adja vissza az objektum tartalmát.
- watch(*mező*, *callback*) : figyeli az adott mezőt, és értékének minden változásakor meghívja a megadott callback függvényt, amely három paramétert kap: a mező nevét, annak régi és új értékét, visszatérési értékként pedig a jóváhagyott új értéket adja meg. A callback mindaddig meghívódik (még az adott mező törlése, majd újrafelvétele esetén is), míg annak regisztrációját meg nem szüntetjük az unwatch meghívásával.
- unwatch(*mező*) : megszünteti az adott mező változásának figyelését. Utóbbi két metódust felhasználva figyelhető és megvétózható egy objektum megváltozása.

Példa a watch, unwatch használatára:

```
objektum = {intmezo:1}
objektum.watch("intmezo",
   function (mezo, regiert, ujert) {
      // dialógusban megjelenítés
      alert("objektum." + mezo + " megváltozott "
         + regiert + "-ről " + ujert + "-ra")
      // változást elfogadjuk
      return ujert
   })
objektum.intmezo = 2
objektum.intmezo = 3
delete objektum.intmezo
objektum.intmezo = 4
objektum.unwatch('intmezo')
objektum.intmezo = 5
```

Ennek futtatásakor háromszor fog megjelenni egy dialógusablak a következő szövegekkel:

```
objektum.intmezo megváltozott 1-ről 2-ra
objektum.intmezo megváltozott 2-ről 3-ra
objektum.intmezo megváltozott 3-ról 4-ra
```

Array : tömbök

A JavaScript nem rendelkezik beépített tömb alaptípussal, helyette egy Array objektum használandó.

Tömböt létrehozni háromféleképp lehet:

1. new Array(*tömbelem*, ...) : a felsorolt tömbelemekkel létrehoz egy már inicializált tömböt.
2. [*tömbelem*, ...] : az előzőnek megfelelő tömb inicializátor.
3. new Array(*tömbhossz*) : a megadott tömbhosszal létrehoz egy üres tömböt.

Tömb létrehozása után annak elemeire szögletes zárójelbe tett indexszel lehet hivatkozni. Az indexelés 0-tól indul. Még nem létező indexre történő értékadás automatikusan a megfelelő méretűre növeli a tömböt. Többdimenziós tömböt tömbök tömbjeként lehet megvalósítani.

Egy tömb alapértelmezett mezői a következők:

- length : a tömb elemeinek száma.

Tömb alapértelmezett metódusai a következők:

- concat(*tömb*, ...) : összefűzi az aktuális tömb másolatát (sekély másolat[2]) és a paraméterekként megadott tömböket. A metódus ezen új összefűzött tömböt adja vissza, magát az aktuális tömböt nem változtatja meg.
- join([*szeparátor*]) : olyan szöveget ad vissza eredményként, amely a tömb elemeit tartalmazza egymás mögé fűzve, köztük a megadott szeparátorral. Ennek elhagyása esetén vessző lesz a szeparátor.
- pop() : kiveszi a tömb utolsó elemét (csökkentve eggyel a tömb méretét), és ezt adja vissza visszatérési értékként.
- push(*tömbelem*, ...) : bővíti a tömböt a megadott elemekkel annak végétől kezdve.
- reverse() : felcseréli a tömb elemeinek sorrendjét, azaz az első elem lesz az utolsó stb...
- shift() : kiveszi és visszaadja a tömb első elemét (a tömb mérete eggyel csökken).
- slice(*kezdőindex*[, *végindex*]) : visszaadja a tömb kért szeletének (sekély) másolatát. Az eredeti tömb nem változik. Ha a végindex nincs megadva, akkor a kezdőindextől a tömb végéig tartó résztömb másolatát adja vissza. Ha a végindex negatív, akkor az a tömb végétől értendő. A végindexszel kijelölt elem már nem kerül másolásra.
- splice(*kezdőindex, darabszám*[, *tömbelem*, ...]) : az adott kezdőindextől kezdve kitörli a tömb megadott darabszámú elemét, majd beszúrja oda az esetleg megadott új tömbelemeket. Visszatérési értékként a kitörölt tömbelemeket adja vissza.
- sort([*rendezőfüggvény*]) : rendezi a tömb elemeit. Ha nincs megadva rendezőfüggvény, a rendezés a tömbelemeket szöveggé konvertálva lexikografikusan történik. A rendezőfüggvény két paramétert kap, visszatérési értéke pedig 0, ha a két elem egyenlő; negatív, ha az első paraméter kisebb, mint a második; pozitív, ha az első paraméter nagyobb, mint a második.
- unshift(*tömbelem*, ...) : a megadott tömbelemeket beszúrja a tömb elejére, és visszaadja a tömb új méretét.

Boolean : logikai érték csomagoló-objektum

Ez az objektum megfelelője a primitív logikai JavaScript típusnak. Konstruktorában paraméterként megadhatjuk az objektum kezdeti logikai értékét.

[2] Sekély másolat csak a referenciák másolását jelenti, nem pedig a hivatkozott objektumokét.

Date : dátumok

A JavaScript nem rendelkezik beépített dátum alaptípussal, helyette egy Date objektum használandó. A Javához hasonlóan az 1970. január elseje 0 óra 0 perc óta eltelt ezredmásodpercek számával azonosít egy adott dátumot/időpontot.

Dátumot létrehozni négyféleképp lehet:

1. new Date() : az aktuális dátummal hoz létre egy dátumobjektumot.
2. new Date(*ezredmásodpercek*) : 1970. január elseje 0 óra 0 perctől eltelt ezredmásodperceket megadva hoz létre új dátumobjektumot.
3. new Date(*időpontsztring*) : a parse metódus alapján hoz létre új dátumobjektumot.
4. new Date(*éééé, hh*[*, nn*[*, óó*[*, pp*[*, mm*[*ms*]]]]]) : a megadott paraméterek alapján hoz létre új dátumobjektumot.

Dátum nem rendelkezik saját mezőkkel. Az alapértelmezett metódusok a következők:

- get/set[UTC]Date() : visszaadja/beállítja a dátum napját a hónapon belül (ennek számozása 1-től indul) [UTC-ben].
- get[UTC]Day() : visszaadja a dátum napját a héten belül (ennek számozása 0-tól indul, ahol 0=vasárnap) [UTC-ben].
- get/set[UTC]FullYear() : visszaadja/beállítja a dátum teljes évszámát [UTC-ben].
- get/set[UTC]Hours() : visszaadja/beállítja az időpontból az órát (0-23) [UTC-ben].
- get/set[UTC]Milliseconds() : visszaadja/beállítja az időpontból az ezredmásodpercek számát (0-999) [UTC-ben].
- get/set[UTC]Minutes() : visszaadja/beállítja az időpontból a percek számát (0-59) [UTC-ben].
- get/set[UTC]Month() : visszaadja/beállítja a dátum hónapját (0-11) [UTC-ben].
- get/set[UTC]Seconds() : visszaadja/beállítja az időpontból a másodpercek számát (0-59) [UTC-ben].
- get/setTime() : a dátumot mint 1970. január elseje 0 óra 0 perc óta eltelt ezredmásodpercek száma adja vissza/állítja be.
- getTimezoneOffset() : időzóna időeltérése a GMT-hez képest percekben megadva.
- parse(*időpontsztring*) : ez a metódus objektumpéldány nélkül is használható (azaz statikus). Visszaadja a szövegként megadott időpontot 1970. január elseje 0 óra 0 perc óta eltelt ezredmásodpercek formájában. Sikertelen konverzió esetén NaN-t ad vissza. Sikeres konverzió érdekében a hónapot szövegesen kell megadni, de ezen kívül az év, hó és nap megadási sorrendje tetszőleges.
- toLocaleString() : az időpontnak megfelelő lokalizált szöveget adja vissza.
- toUTCString() : az időpontnak megfelelő UTC szabványú szöveget adja vissza.
- UTC(*éééé, hh, nn*[*, óó, pp, mm, ms*]) : statikus metódus, a megadott időpontnak megfelelő, 1970. január elseje 0 óra 0 perctől eltelt ezredmásodpercek számát adja vissza.

Function : függvények

Minden JavaScript függvényt egy Function objektum reprezentál.

Függvényt létrehozni kétféleképp lehet:

1. *név* = new Function([*argumentum, . . .*]*függvénytörzs*) : ez a forma tulajdonképp egy függvény típusú változót hoz létre. A változó függvényként használható, azaz a hivatkozott függvény a változónév után tett zárójelek (és az esetleges felsorolt paraméterek) hatására végrehajtódik. Ennek a formának a hátránya, hogy a függvénytörzset megadó kifejezés a függvény végrehajtásakor mindig újra kiértékelődik.

2. function *név*([*argumentum, ...*]) {*függvénytörzs*} : ez egy előfordított függvényt definiál. A név nem egy változó, hanem valóban a függvény neve.

Függvény objektum alapértelmezett mezői a következők:

- arguments : a függvény paramétereit megadó tömb.
- arity : a függvény definíciójakor megadott formális paraméterek száma.
- length : a függvény meghívásakor megadott formális paraméterek száma.

Függvény objektum alapértelmezett metódusai a következők:

- apply(*aktuális objektum*[, *paramétertömb*]) : meghívja az aktuális függvényt, melynek törzsén belül az aktuális objektum (this) az első paraméterként megadott objektum lesz, az aktuális paraméterértékeket pedig az esetleges második paraméterként megadott tömb tartalmazza. Ennek segítségével akár konstruktorfüggvényeket is egymás után lehet fűzni, mint ahogy azt a következő példa is mutatja:

```
function cikk(neve, ara){
   this.neve = neve
   this.ara = ara
}
function raktarcikk(neve, ara, darab){
   this.darab = darab;
   cikk.apply(this, arguments);
}
rc = new raktarcikk("cikki", 10, 1000)
// rc = {neve:"cikki", ara:10, darab:1000}
```

- call(*aktuális objektum*[, *paraméter, ...*]) : meghívja az aktuális függvényt, melynek törzsén belül az aktuális objektum (this) az első paraméterként megadott objektum lesz, az aktuális paraméterértékeket pedig a többi paraméter adja meg. A apply-tól csak a paraméterértékek megadási módjában különbözik.

Math : matematikai függvények és konstansok

Ezen prototípus az alapvető matematikai függvényeket és konstansokat tartalmazza statikus formában, azaz azok eléréséhez nem kell objektumot használni. Épp ezért ilyen objektumot nekünk soha sem kell példányosítani.

A Math objektum alapértelmezett mezői a következők:

- E : a természetes logaritmus alapszáma.
- LN10 : 10 természetes alapú logaritmusa.
- LN2 : 2 természetes alapú logaritmusa.
- LOG10E : E tízes alapú logaritmusa.
- LOG2E : E kettes alapú logaritmusa.
- PI : π.
- SQRT1_2 : 1/2 négyzetgyöke.
- SQRT2 : 2 négyzetgyöke.

A Math objektum alapértelmezett metódusai a következők:

- abs : az argumentumként megadott szám abszolút értékét adja vissza.
- acos : az argumentumként megadott szám arkuszkoszinuszát adja vissza radiánban. Érvénytelen (< -1 vagy $1 <$) paraméter esetén NaN-t ad vissza.

- asin : az argumentumként megadott szám arkuszszinuszát adja vissza radiánban. Érvénytelen (< -1 vagy $1 <$) paraméter esetén NaN-t ad vissza.
- atan : az argumentumként megadott szám arkusztangensét adja vissza radiánban.
- atan2(y, x) : az argumentumként megadott koordinátájú irányvektor szögének arkusztangensét adja vissza radiánban.
- ceil : az argumentumként megadott számnál nem kisebb legkisebb egész számot adja vissza.
- cos : az argumentumként radiánban megadott szög koszinuszát adja vissza.
- exp : az E argumentumként megadott számú hatványát adja vissza.
- floor : az argumentumként megadott számnál nem nagyobb legnagyobb egész számot adja vissza.
- log : az argumentumként megadott szám természetes alapú logaritmusát adja vissza. Érvénytelen (< 0) paraméter esetén NaN-t ad vissza.
- max : a két argumentum maximumát adja vissza.
- min : a két argumentum minimumát adja vissza.
- pow : az első argumentum második argumentumként megadott számú hatványát adja vissza.
- random : egy $0 - 1$ közé eső álvéletlen számot ad vissza.
- round : az argumentumként megadott számhoz legközelebb eső egész számot adja vissza.
- sin : az argumentumként radiánban megadott szög szinuszát adja vissza.
- sqrt : az argumentumként megadott szám négyzetgyökét adja vissza. Érvénytelen (< 0) paraméter esetén NaN-t ad vissza.
- tan : az argumentumként radiánban megadott szög tangensét adja vissza.

Number : számok

Ezen prototípus tulajdonképp numerikus értékek objektumba csomagolását végzi. Konstruktorfüggvénye ennek megfelelően egyetlen numerikus paramétert vár.

Egy Number objektum alapértelmezett mezői a következők:

- MAX_VALUE : ez a statikus mező a legnagyobb még ábrázolható számot adja meg. Ennél nagyobb értékeket a JavaScript végtelennek tekint.
- MIN_VALUE : ez a statikus mező a legkisebb még ábrázolható pozitív számot adja meg. Ennél kisebb értékeket a JavaScript nullának tekint.
- NaN : megfelel a felsőszintű NaN mezőnek.
- NEGATIVE_INFINITY : a negatív végtelent ábrázoló numerikus érték.
- POSITIVE_INFINITY : a pozitív végtelent ábrázoló numerikus érték.

RegExp : Reguláris kifejezések

Egy reguláris kifejezés segítségével szövegkeresési mintákat fogalmazhatunk meg. Ezen minták megadásakor a reguláris kifejezést egy szöveg segítségével adhatjuk meg, melyben a keresendő karaktereken kívül az alábbi speciális jeleket, illetve jelöléseket használhatjuk:

- \ : Ez az úgynevezett escape karakter, melynek segítségével speciális karakterek is megadhatók keresési mintaként, illetve ez vezet be minden különleges jelsorozat keresését előíró jelet (lásd később).
- \o*nnn* : Oktálisan megadott ASCII kódú karaktert reprezentál.
- \x*nn* : Hexadecimálisan megadott ASCII kódú karaktert reprezentál.

- `^` : Sor elejét reprezentálja.
- `$` : Sor végét reprezentálja.
- `*` : A megelőző karakter tetszőleges számú (akár nulla) előfordulását írja elő.
- `+` : A megelőző karakter tetszőleges számú (de legalább egyszeri) előfordulását írja elő.
- `?` : A megelőző karakter nulla vagy egyszeri előfordulását írja elő.
- `.` : Tetszőleges karakter egyszeri előfordulását írja elő, kivéve ha az a sor elején áll.
- Kapcsos zárójelek : A megelőző karakter többszöri ismétlődését írja elő. A zárójelben legfeljebb két egész szám állhat vesszővel elválasztva, így lehet az adott karakter előfordulásának számára egy minimumot és maximumot megadni. Ha maximumot nem akarunk adni, akkor a második szám elhagyható, de a vesszőt ekkor is ki kell tenni. Ha csak egy szám áll a zárójelben, akkor a kért karakter pontosan csak annyiszor fordulhat elő a szövegben.
- Kerek zárójelek : A zárójelezett mintának megfelelő szöveget megjegyzi, lehetővé téve ezáltal annak későbbi felhasználását.
- `\`*n* : Az *n*. (bal)zárójellel megjelölt minta újbóli előfordulását írja elő. Ha a balzárójelek száma kisebb, mint *n*, akkor azt oktális karakterként értelmezi.
- Szögletes zárójelek : A zárójelben megadott karakterhalmaz egy elemének előfordulását írja elő. A karakterhalmazt a karakterek egymás után írásával vagy kötőjelet használva tartományokat kijelölve lehet megadni. Ha a zárójeleken belül az első karakter egy `^`, akkor az invertálja a karakterhalmazt, azaz a halmazban megadott karakterek nem szerepelhetnek az adott helyen.
- `|` : Két minta előfordulásának vagylagosságát írja elő.
- `[\b]` : A visszatörlés (backspace) karakter előfordulását írja elő.
- `\b` : Szóhatárt, azaz egy szó után következő üres (whitespace) karaktert ír elő.
- `\B` : Az előző tagadása.
- `\c`*X* : Adott kontrollkarakter (Ctrl-*X*) előfordulását írja elő.
- `\d` : Számjegy előfordulását írja elő. Ugyanaz, mint `[0-9]`.
- `\D` : Az előző tagadása. Ugyanaz, mint `[^0-9]`.
- `\f` : Lapdobás előfordulását írja elő.
- `\n` : Soremelés előfordulását írja elő.
- `\r` : Kocsivissza (carriage return) előfordulását írja elő.
- `\s` : Egy darab üres karakter előfordulását írja elő. Ugyanaz, mint `[ \f\n\r\t\v]`.
- `\S` : Az előző tagadása. Ugyanaz, mint `[^\f\n\r\t\v]`.
- `\t` : Tabulátor előfordulását írja elő.
- `\v` : Vertikális tabulátor előfordulását írja elő.
- `\w` : Ugyanaz, mint `[A-Za-z0-9_]`.
- `\W` : Az előző tagadása. Ugyanaz, mint `[^A-Za-z0-9_]`.

Reguláris kifejezést létrehozni kétféleképp lehet:

1. new RegExp("*minta*"[, "*kapcsolók*"]) : konstruktorfüggvény. A *kapcsolók* az alábbiak lehetnek:

   - g : minden előfordulás keresése.
   - i : betűnagyság figyelmen kívül hagyása.
   - gi : az előző kettő egyszerre.

   Ennél a formánál a minta megadásakor ügyeljünk arra, hogy speciális karaktereket (pl. \) csakis az escape-karakterrel(\) lehet megadni, mivelhogy itt a mintát, mint sztringet kell specifikálni.

2. */minta/*[*kapcsolók* : ez az objektum-inicializátornak megfelelő forma. Az így megadott reguláris kifejezés előfordítódik, míg a konstruktorfüggvénnyel megadott minden egyes hivatkozásakor újra kiértékelődik.

Egy reguláris kifejezés objektum alapértelmezett mezői a következők:

- $1 - $9 : a legutóbbi reguláris kifejezés első kilenc megjegyzett mintáját tartalmazó statikus mezők.
- $_ : ugyanaz, mint input.
- $* : ugyanaz, mint multiline.
- $& : ugyanaz, mint lastMatch.
- $+ : ugyanaz, mint lastParen.
- $‘ : ugyanaz, mint leftContext.
- $’ : ugyanaz, mint rightContext.
- global : logikai értékként visszaadja, hogy globális-e a keresés.
- ignoreCase : logikai értékként visszaadja, hogy kis-nagybetűk azonosan kezelendőek-e.
- input : statikus mező, mely a legutóbbi kereséskor felhasznált kiindulási szöveget tartalmazza. Ha az exec vagy test metódust paraméter nélkül hívjuk meg, akkor ezen mező értéke adja a kiindulási szöveget. A következő esetekben kerül automatikusan beállításra az input mező értéke:
  - TEXT szövegmező minden eseménykezelőjében az input mező a szövegmező tartalmát veszi fel értékként.
  - TEXTAREA szövegmező minden eseménykezelőjében az input mező a szövegmező tartalmát veszi fel értékként. Mivel ez a fajta szövegmező többsoros szöveget is tartalmazhat, ezért a multiline mező értéke is igazra állítódik.
  - SELECT legördülős menü minden eseménykezelőjében az input mező a kiválasztott menüpont szövegét veszi fel értékként.
  - Link objektum minden eseménykezelőjében az input mező a hyperlink szövegét veszi fel értékként.

  Az eseménykezelő lefutása után törlődik az input mező értéke.
- lastIndex : egész értékként megadja, hogy minden előfordulás keresésekor a következő keresés a kezdeti szöveg mely pozíciójától folytatódik. Ha lastIndex nagyobb, mint a kezdeti szöveg hossza, a test és exec metódusok nem fognak semmit sem találni és visszaállítják ezen mezőt nullára, egyébként pedig minden keresés után a megtalált részszöveg utáni pozícióra fog mutatni.
- lastMatch : statikus mező, a mintának legutoljára megfelelő részszöveget tartalmazza.
- lastParent : statikus mező, a minta utolsó zárójelezett részének legutoljára megfelelő részszöveget tartalmazza.
- leftContext : statikus mező, a kezdeti szöveg mintának legutoljára megfelelő részszövegét megelőző prefixe.
- multiline : statikus logikai mező, jelzi, hogy a keresés a szöveg minden során végigmenjen-e vagy fejeződjön be a sorvége elérésekor. HTML elemek eseménykezelőinek lefutása után hamisra állítódik ez a mező.
- rightContext : statikus mező, a kezdeti szöveg mintának legutoljára megfelelő részszövegét követő szuffixe.

Reguláris kifejezés alapértelmezett metódusai a következők:

- compile(*minta*[, *kapcsolók*]) : (újra)fordítja a konstruktorfüggvénnyel létrehozott reguláris kifejezést. Ezek után a kifejezés nem fog újra kiértékelődni minden egyes használatakor.
- exec([*kezdeti szöveg*]) : elvégzi a reguláris kifejezés keresését a megadott szövegen [illetve az input mező tartalmán]. Az eredményként visszaadott tömb a következőket tartalmazza:
  - index : a minta aktuális előfordulási indexe a kiindulási szövegen belül 0-tól kezdve.
  - input : a keresés kiindulási szövege.
  - 0 : ugyanaz, mint RegExp.lastMatch.
  - 1, ... *n* : a reguláris kifejezés zárójelezéssel megjegyzett részmintái.
- test([*kezdeti szöveg*]) : logikai értékként visszaadja, hogy megfelel-e a reguláris kifejezésnek a megadott szöveg [illetve az input mező tartalma].

String : szöveg

Ezen prototípus tulajdonképp szövegek objektumba csomagolását végzi. Konstruktorfüggvénye ennek megfelelően egyetlen szöveges paramétert vár. Idézőjelekkel megadott szövegliterálok használata esetén a JavaScript automatikusan létrehoz egy becsomagoló objektumot, így szövegliterálokra is alkalmazható minden String metódus.

Egy String objektum alapértelmezett mezői a következők:

- length : az adott szöveg hossza.

Egy sztring alapértelmezett metódusai a következők:

- anchor(*címke*) : a megadott címkével elérhető HTML link végpontot rendeli az aktuális szöveghez. Tehát a `"Tartalomjegyzék".anchor("TOC")` metódus a `<A NAME="TOC">Tartalomjegyzék</A>` HTML sort generálja.
- big : az aktuális szöveget a legnagyobb betűmérettel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".big()` metódus a `<BIG>Tartalomjegyzék</BIG>` HTML sort generálja.
- blink : az aktuális szöveget villogva megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".blink()` metódus a `<BLINK>Tartalomjegyzék</BLINK>` HTML sort generálja.
- bold : az aktuális szöveget vastag betűkkel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".bold()` metódus a `<B>Tartalomjegyzék</B>` HTML sort generálja.
- charAt : visszaadja az aktuális szöveg paraméterként megadott pozícióján álló karakterét. Az indexelés 0-tól kezdődik. Nem megengedett index esetén üres szöveget ad vissza.
- charCodeAt : egészként visszaadja az aktuális szöveg paraméterként megadott pozícióján álló karakterének UNICODE kódját. Az indexelés 0-tól kezdődik. Nem megengedett index esetén NaN-t ad vissza.
- concat(*szöveg*[, ...]) : visszaadja az aktuális és a megadott szöveg(ek) összefűzésével előálló új szöveget.
- fixed : az aktuális szöveget fix szélességű betűkkel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".fixed()` metódus a `<TT>Tartalomjegyzék</TT>` HTML sort generálja.
- fontcolor(*szín*) : az aktuális szöveget a megadott színnel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".fontcolor("gold")` metódus a `<FONT COLOR="gold">Tartalomjegyzék</FONT>` HTML sort generálja.

- fontsize(*betűméret*) : az aktuális szöveget a megadott betűmérettel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".fontsize(5)` metódus a `<FONTSIZE=5>Tartalomjegyzék</FONTSIZE>` HTML sort generálja. A paraméter az előre definiált hét darab fontméretet adhatja meg 1-től 7-ig, vagy előjeles számként kérheti az aktuális betűmérethez képesti kicsinyítést vagy nagyítást.
- fromCharCode(*betűkód*[, ...]) : statikus metódus, amely visszaad egy, a felsorolt UNICODE betűkód(ok)ból álló új szöveget.
- indexOf(*részszöveg*[, *kezdőindex*]) : a részszöveg első előfordulásának [a megadott kezdőindextől kezdve] indexét adja vissza, vagy −1-et ad, ha a keresés eredménytelen.
- italics : az aktuális szöveget dőlt betűkkel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".italics()` metódus a `<I>Tartalomjegyzék</I>` HTML sort generálja.
- lastIndexOf(*részszöveg*[, *végindex*]) : a részszöveg utolsó előfordulásának [a megadott végindextől kezdve] indexét adja vissza, vagy −1-et ad, ha a keresés eredménytelen.
- link(*URL*) : az aktuális szöveghez, mint linkhez hozzárendeli a megadott URL-t. Tehát a `"Tartalomjegyzék".link("toc.html")` metódus a `<A HREF="toc.html">Tartalomjegyzék</A>` HTML sort generálja.
- match(*reguláris kifejezés*) : Végrehajtja a paraméterként megadott reguláris kifejezés exec metódusát az aktuális szövegre. Ezen exec metódus eredménytömbje lesz a visszatérési érték.
- replace(*reguláris kifejezés, szöveg[értékű függvény]*[]) : visszaad egy új sztringet, amely az aktuális szövegből a reguláris kifejezés alapján történő helyettesítéssel képződik. String típusú második paraméter esetén egyszerűen ezen megadott szöveg áll a reguláris kifejezést kielégítő minden egyes rész-sztring helyén. Ezen szöveg-paraméterben fel lehet használni a RegExp $1 - $9, lastMatch, lastParent, leftContext és rightContext mezőit. Ha a második paraméter függvény típusú, akkor ezen függvénynek kell visszaadnia a reguláris kifejezést aktuálisan kielégítő rész-sztringet helyettesítő szöveget. Ezen függvény első paramétere a reguláris kifejezést aktuálisan kielégítő rész-sztringet tartalmazza, míg az esetleges további paraméterek a reguláris kifejezés zárójelezett részkifejezéseinek eleget tevő szövegrészeket adják meg.
  A következő példa megcseréli egy adott szöveg szavainak sorrendjét az alábbi permutáció szerint: 2n 2n+1 -> 2n+1 2n:

```
re = /(\w+)\s(\w+)/g
str = "John Smith 3dik 4edik"
newstr=str.replace(re, "$2 $1")

newstr2=str.replace(re, function(match, elso, masod) {
    return masod + " " + elso
})
```

- search(*reguláris kifejezés*) : Megkeresi a paraméterként megadott reguláris kifejezés első előfordulásának indexét az aktuális szövegen belül. Ha nincs találat, a visszaadott érték −1.
- slice(*kezdőindex*[, *végindex*]) : a szöveg részszövegét adja vissza új sztringként a kezdőindextől kezdve [a végindexet már nem tartalmazva. Ha a végindex negatív, akkor az a szöveg végétől számolandó.].
- small : az aktuális szöveget a legkisebb betűmérettel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".small()` metódus a `<SMALL>Tartalomjegyzék</SMALL>` HTML sort generálja.

- split(*elválasztójel*[, *max. darabszám*]) : visszaad egy szövegtömböt [legfeljebb a megadott maximális hosszal], melynek elemeit az aktuális szöveg feldarabolásával kapjuk. Az elválasztójel a daraboláskor használt szóhatárt adja meg sztringként vagy akár reguláris kifejezésként.
- strike : az aktuális szöveget áthúzott betűkkel megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".strike()` metódus a `<STRIKE>Tartalomjegyzék</STRIKE>` HTML sort generálja.
- sub : az aktuális szöveget alsó indexként megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".sub()` metódus a `<SUB>Tartalomjegyzék</SUB>` HTML sort generálja.
- substr(*kezdőindex*[, *hossz*]) : visszaadja az aktuális szöveg [kért hosszú] részszövegét a megadott kezdőindextől kezdve. Ha a kezdőindex negatív, akkor az a szöveg végétől értendő. Érvénytelen paraméterek esetén üres sztringet ad vissza.
- substring(*kezdőindex, végindex*) : visszaadja az aktuális szöveg részszövegét.
- sup : az aktuális szöveget felső indexként megjelenítő HTML sort generál. Tehát a `"Tartalomjegyzék".sup()` metódus a `<SUP>Tartalomjegyzék</SUP>` HTML sort generálja.
- toLowerCase : visszaadja új sztringként az aktuális szöveg kisbetűs változatát.
- toUpperCase : visszaadja új sztringként az aktuális szöveg nagybetűs változatát.

G.3. Kliensoldali JavaScript

Mint már korábban említettük, a kliensoldali JavaScript az alap JavaScript nyelvet HTML oldalak felépítésének leírását lehetővé tevő objektumokkal egészíti ki. Magát a JavaScript programot forrásszöveg szintjén egy HTML oldal tartalmazza. A beágyazott JavaScript kód végrehajtása előtt a böngészőprogram létrehoz egy objektumhierarchiát, amely a befoglaló HTML oldal elemein kívül a böngésző elérhető szolgáltatásait is modellezi. Az adott HTML oldal elemeit leíró objektumhierarchiát *Dokumentum Objektum Modellnek*(DOM) nevezzük, melynek jelenleg folyik a W3C (World Wide Web Consortium) által történő szabványosítása.

A böngészőprogram egy HTML oldal betöltésekor kiértékeli és eltárolja a JavaScript függvényeket, felépíti a Dokumentum Objektum Modellt az oldal tartalma alapján és a <SCRIPT> </SCRIPT> HTML kulcsszavak közti JavaScript utasításokat végrehajtva elvégzi az oldal megjelenítését. Mivel egy HTML oldalon belüli JavaScript blokk akkor kerül végrehajtásra, amikor a böngésző elérte azt a HTML oldal soronkénti feldolgozása közben, az oldal adott helyén elhelyezett JavaScript blokk akár dinamikusan is befolyásolhatja a megjelenítendő oldal felépítését. Ennek következménye még, hogy minden ilyen szkriptblokk egyszer és egymás után hajtódik végre, mégpedig a HTML oldal megjelenítésekor.

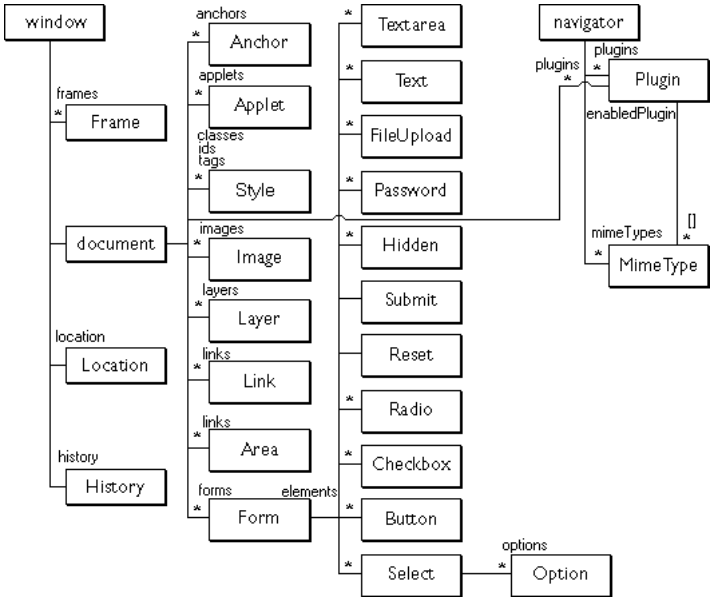
DOM - Dokumentum Objektum Modell

Mint már korábban említettük, a *DOM* a JavaScriptet tartalmazó HTML oldal felépítését leíró objektumok összessége. Mivel ezen DOM objektumokat mindig a böngésző hozza létre automatikusan, ezért nevezhetjük a HTML oldal szerkezetét reprezentáló objektumokat kliens-oldali JavaScript objektumoknak.

A böngészőprogram egy HTML oldal betöltésekor építi fel dinamikusan a Dokumentum Objektum Modellt az oldal tartalma alapján. Mivel az oldal feldolgozása és megjelenítése soronként, az oldal elejétől kezdődik, ezért esetleges JavaScript kódban csak a már teljesen definiált HTML elemeket reprezentáló DOM objektumokra lehet hivatkozni.

Ugyanezen soronkénti feldolgozás miatt, ha a HTML oldal adott része már megjelenítésre került, annak kinézetét megváltoztatni JavaScriptből csakis az oldal újratöltésével együtt lehetséges.[3] A JavaScript ezen tulajdonságát, miszerint a DOM objektumok a HTML oldal szerkezetét reprezentálják és emiatt onnan veszik tulajdonságaikat megadó értékeiket, valamint hogy a DOM objektumok JavaScriptből történő módosítása maga után vonhatja az aktuális HTML oldal megjelenítésének változását, a JavaScript *reflection*(önvizsgálati) tulajdonságának nevezzük.

A HTML oldalt reprezentáló DOM objektumok egy hierarchiát alkotnak, melyet a G.3.. ábra szemléltet.



G.3. ábra: A DOM hierarchia

Ezen hierarchia tartalmazási reláció alapján épül fel, azaz azt ábrázolja, hogy például a window objektum rendelkezik egy document objektummal és esetleg több Frame objektummal is. A közvetlenül tartalmazott objektumok az őt tartalmazó objektumban, mint mező vagy objektumtömb jelennek meg. Mező esetén a tartalmazott objektumra annak nevével lehet hivatkozni, objektumtömb esetén pedig tömbindexként mind az objektum neve, mind annak a definiálási sorrendet követő 0-val kezdődő sorszáma használható. Ez alól egyedüli kivétel a document objektum, melynek neve elé nem kell kitenni a window prefixet, minden más DOM objektum elérésekor annak teljes elérési útvonalát meg kell adni.

Tekintsük át röviden, hogy a DOM milyen objektumokat tartalmaz minden esetben:

- window : ez a legfelsőbb szintű objektum, a teljes böngészőablakra vonatkozó tulajdonságokat reprezentálja.
- document : a megjelenített HTML oldal jellemzőit reprezentálja.
- location : az aktuális URL tulajdonságait reprezentálja.
- history : a felhasználó által korábban meglátogatott URL-ek tulajdonságait reprezentálja.

[3] Ez alól kivételt jelentenek a HTML űrlap elemek.

- navigator : az aktuális böngésző tulajdonságait reprezentálja.

Másfajta objektumok csak akkor jönnek létre, ha azokat a megjelenített HTML oldal tartalmazza. A továbbiakban majd részletesen ismertetjük minden DOM objektum tulajdonságát és lehetséges eseményeit.

Miután a HTML oldal megjelenítésre került, a böngészőprogram a felépített DOM objektumainak eseménykezelőit fogja meghívni egy-egy esemény bekövetkeztekor, azaz az oldal megjelenítése után JavaScript kód csakis eseményvezérelten hajtódik végre. Esemény alatt az oldalt megtekintő és a böngészőt vezérlő felhasználó lehetséges cselekedeteit értjük, például adott HTML link kiválasztása, gomb megnyomása vagy szöveg megadása egy HTML űrlapon stb....

Eseménykezelők

Eseménykezelő JavaScript kód az adott HTML elem megfelelő eseményének bekövetkezésekor kerül végrehajtásra. Az általános szintaxis a következő:
<*HTML_objektum ... eseménykezelő*="*JavaScript kód*">
Az eseménykezelő kódon belül a this hivatkozás mindig az aktuális HTML objektumot reprezentáló DOM objektumnak felel meg. Ha az eseménykezelő kód több utasításból áll, akkor azokat pontosvesszővel vagy új sorral kell elválasztani.

Minden eseménykezelő egy logikai értéket ad vissza, ami azt jelzi, hogy az adott esemény feldolgozása folytatódhat-e. Például ha egy hyperlink rákattintási eseménykezelőjében hamis értéket adunk vissza, a böngésző nem fogja megjeleníteni a kiválasztott linket.

Megjegyzendő, hogy eseménykezelő kód csak akkor kerül végrehajtásra, ha az adott esemény valóban be is következett, így egy eseményt szimuláló metódus (pl. click()) meghívásakor nem fog a hozzátartozó eseménykezelő kód (pl. onClick) is végrehajtódni, viszont minden eseménykezelőt meghívhatunk explicit módon, az adott HTML objektum metódusaként.

Megjegyzendő még, hogy mivel az eseménykezelő JavaScript kódot dupla idézőjelek közt kell megadni, a kódon belül csak szimpla idézőjelek használandók szövegliterálok jelölésére. Ez a megkötés kikerülhető, ha az eseménykezelő kód csak egy JavaScript függvény meghívásából áll, melyet már korábban egy <SCRIPT> </SCRIPT> blokkban definiáltunk.

Eseménykezelők meghívási sorrendje

Alapértelmezett esetben mindig csak annak a DOM objektumnak az eseménykezelője hívódik meg, amely objektummal az esemény történt. Például ha egy HTML űrlapon belül egy gombot megnyomunk, akkor annak a gombnak fog meghívódni a rákattintási eseménykezelője még akkor is, ha esetleg a gombot tartalmazó űrlap, vagy maga a dokumentum is definiál rákattintási eseménykezelőt. Sőt, ha a gomb maga nem kezeli ezen eseményt, de egy esetleges befoglaló objektum igen, akkor sem fogja az megkapni alapértelmezés szerint a gomb által le nem kezelt eseményt.

Az események ezen feldolgozási rendje megváltoztatható, azaz mielőtt egy esemény elérné azt az objektumot, ahol tulajdonképp az bekövetkezett, annak feldolgozása egy magasabb szintű DOM objektumhoz átirányítható. Ez az átirányítás csak a böngészőablakot reprezentáló window, a HTML dokumentumot reprezentáló document és egy HTML réteget reprezentáló layer DOM objektumoknál lehetséges, azok következő metódusaink felhasználásával:

- captureEvents : a megadott eseménymaszknak megfelelő eseményeket ezentúl az aktuális konténer fogja megkapni, nem pedig az a DOM objektum, ahol az valójában keletkezett.
- releaseEvents : visszaállítja a megadott eseménymaszknak megfelelő események alapértelmezett feldolgozási rendjét.
- routeEvent : a paraméterként megadott eseményt további feldolgozás céljából továbbadja a JavaScript. Ez azt jelenti, hogy ha még más objektum is elkapja az adott eseményt a captureEvents metódussal, akkor annak adja tovább azt. Ha már nincs ilyen objektum és az a DOM objektum, ahol maga az esemény keletkezett, definiál eseménykezelőt ehhez az eseményhez, akkor az fog végrehajtódni. A visszatérési érték a végrehajtott eseménykezelő visszatérési értéke lesz. Megjegyzendő, hogy ha az esemény feldolgozása alatt egy új HTML oldal került megjelenítésre, akkor nem fog visszatérni ezen metódus végrehajtása.
- handleEvent : ezen metódussal minden olyan DOM objektum rendelkezik, amely események kezelésére képes. Segítségével közvetlenül végeztetjük el egy esemény feldolgozását, megkerülve ezzel a beállított esemény-feldolgozási rendet.

Eseménymaszkként a kérdéses eseményt azonosító konstans, illetve ilyen konstansok bitenként kombinációja (|) használható. A JavaScriptben használható események nevét, a hozzájuk tartozó eseménymaszkokat, illetve ezen eseményeket feldolgozó eseménykezelők nevét a G.4.. táblázat tartalmazza.

Eseményleíró objektum : Event

Minden eseménykezelő paraméterként megkap egy eseményt leíró objektumot. Ezen objektumot tehát mint az eseménykezelő függvénynek átadott paramétert érhetjük el. Mivel egy HTML objektum attribútumaként közvetlenül, saját függvény definiálása nélkül is megadhatjuk az eseménykezelő kódot, ezért ilyenkor nincs lehetőségünk formális paramétereket definiálni, de ekkor is elérhetjük az eseményobjektumot arguments[0]-ként, vagy egyszerűen az event referencián keresztül. Megjegyzendő, hogy ezen event referencia csakis az eseménykezelőben megadott JavaScript kódon belül használható.

Eseményt leíró objektumot mindig a böngésző program hoz létre esemény bekövetkeztekor, így soha nem mi magunk hozunk létre ilyen objektumot.

Egy Event objektum alapértelmezett mezői a következők:

- data : dragdrop esemény esetén a rádobott objektumok URL-jeit megadó sztringtömböt tartalmazza.
- height : az aktuális ablak/frame magassága képpontokban megadva.
- layerX : resize eseménykor az objektum szélességét, különben pedig az egérkurzor vízszintes pozícióját tartalmazza az aktuális HTML rétegen belül képpontokban megadva.
- layerY : resize eseménykor az objektum magasságát, különben pedig az egérkurzor függőleges pozícióját tartalmazza az aktuális HTML rétegen belül képpontokban megadva.
- modifiers : az esemény bekövetkeztekor a módosítóbillentyűk állapotát megadó mező. Értéke a következő egész konstansok bitenkénti összekapcsolása lehet:
  - Event.ALT_MASK : az Alt billentyű megnyomását jelzi.
  - Event.CONTROL_MASK : a Ctrl billentyű megnyomását jelzi.
  - Event.SHIFT_MASK : a Shift billentyű megnyomását jelzi.
  - Event.META_MASK : a Meta billentyű megnyomását jelzi.
- pageX : az egérkurzor vízszintes pozíciója a HTML oldalon belül képpontokban megadva.

Esemény neve	Eseménymaszk	Eseménykezelő neve[4]	Használt Event mezők	Kiváltó DOM objektumok	Leírás
abort	Event.ABORT	onAbort	type, target	Image	adott HTML elem megjelenítését megszakították.
blur	Event.BLUR	onBlur	type, target	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window	adott HTML elem elvesztette a beviteli fókuszt.
change	Event.CHANGE	onChange	type, target	FileUpload, Select, Text, Textarea	adott HTML elem tartalma megváltozott és elvesztette a fókuszt.
click	Event.CLICK	onClick	type, target, which, modifiers, Link esetén még layerX, layerY, pageX, pageY, screenX, screenY	Button, document, Checkbox, Link, Radio, Reset, Submit	adott HTML elem megnyomása egérkattintással / billentyűzettel.
dblclick	Event.DBLCLICK	onDblClick	type, target, which, modifiers, layerX, layerY, pageX, pageY, screenX, screenY	document, Link	adott HTML elem megnyomása egérkattintással / billentyűzettel.
dragdrop	Event.DRAGDROP	onDragDrop	type, target, data, modifiers, screenX, screenY	window	adott HTML elemen objektum rádobása. Normál esetben ez a kapott objektum megjelenítését eredményezi, de hamis visszatérési értékkel ez megakadályozható.

G.4. ábra: JavaScript események 1.

Esemény neve	Eseménymaszk	Eseménykezelő neve[5]	Használt Event mezők	Kiváltó DOM objektumok	Leírás
error	Event.ERROR	onError	type, target	Image, window	adott HTML elem megjelenítésekor hiba lépett fel. null eseménykezelő megadása esetén a böngésző nem fog semmi JavaScript hibát kijelezni. A hibakezelő három paramétert kap: a hibaüzenetet, a hiba fellépésének URL-jét és azon dokumentumon belüli sorszámát. Ha a böngésző hibajelzését meg akarjuk akadályozni, akkor a hibakezelőnek igaz értéket kell visszaadnia.
focus	Event.FOCUS	onFocus	type, target	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window	adott HTML elem megkapta a beviteli fókuszt.
keydown / keyup	Event.KEYDOWN / Event.KEYUP	onKeyDown / onKeyUp	type, target, which, modifiers, layerX, layerY, pageX, pageY, screenX, screenY	document, Image, Link, Textarea	adott HTML elemen billentyű lenyomása / felengedése. Ez az esemény mindig a keypressed esemény előtt / után következik be. keydown esetén ha hamis értéket adunk vissza, azzal letilthatjuk a keypressed esemény kiváltását.
keypress	Event.KEYPRESS	onKeyPress	type, target, which, modifiers, layerX, layerY, pageX, pageY, screenX, screenY	document, Image, Link, Textarea	adott HTML elemen billentyű megnyomása. Ez az esemény csak akkor váltódik ki, ha azt egy esetleges keydown eseménykezelő nem tiltja. Ekkor viszont hosszabb billentyűlenyomáskor ezen esemény többször is kiváltódik.

G.5. ábra: JavaScript események 2.

Esemény neve	Esemény-maszk	Eseménykezelő neve[6]	Használt Event mezők	Kiváltó DOM objektumok	Leírás
mousedrag	Event. MOUSEDRAG	onMouseDrag	type, target, data, modifiers, screenX, screenY	window	adott HTML elem fölött objektum áthúzása.
mousemove	Event. MOUSEMOVE	onMouseMove	type, target, layerX, layerY, pageX, pageY, screenX, screenY	Alapértelmezés szerint sehol sem lép fel.	adott HTML elemen az egér mozgatása. Mivel ez rengeteg esemény keletkezését jelentené, ezért csak az az objektum kapja meg az ilyen eseményeket, amely ezt külön kéri a captureEvents metódus használatával.
mouseout	Event. MOUSEOUT	onMouseOut	type, target, layerX, layerY, pageX, pageY, screenX, screenY	Layer, Link	az adott HTML elemet elhagyta az egérmutató. Alapértelmezés szerint ekkor a defaultStatus szöveg jelenik meg a böngésző státuszsorában. Ha ezt el akarjuk kerülni, akkor ezen eseménykezelőben igaz értéket kell visszaadni.
mouseover	Event MOUSEOVER	onMouseOver	type, target, layerX, layerY, pageX, pageY, screenX, screenY	Layer, Link	adott HTML elemre lépett az egérmutató. Alapértelmezés szerint Link esetén ekkor a böngésző státuszsorában megjelenik a hivatkozott URL. Ha ezt el akarjuk kerülni, akkor ezen eseménykezelőben igaz értéket kell visszaadni.
mousedown	Event. MOUSEDOWN	onMouseDown	type, target, which, modifiers, layerX, layerY, pageX, pageY, screenX, screenY	Button, document, Link	adott HTML elemen egérgomb lenyomása. Ha itt hamis értéket adunk vissza, akkor a HTML elem nem kerül megnyomott állapotba.

G.6. ábra: JavaScript események 3.

Esemény neve	Eseménymaszk	Eseménykezelő neve[7]	Használt Event mezők	Kiváltó DOM objektumok	Leírás
mouseup	Event.MOUSEUP	onMouseUp	type, target, which, modifiers, layerX, layerY, pageX, pageY, screenX, screenY	Button, document, Link	adott HTML elemen egérgomb felengedése. Ha itt hamis értéket adunk vissza, a lenyomott HTML elem nem érzékeli a megnyomást.
load	Event.LOAD	onLoad	type, target, window esetén pedig még width, height	Image, Layer, window	adott HTML elem betöltése befejeződött.
move	Event.MOVE	onMove	type, target, screenX, screenY	window	ablak mozgatása.
reset	Event.RESET	onReset	type, target	Form	HTML űrlap kezdeti értékeinek visszaállítása.
resize	Event.RESIZE	onResize	type, target, width, height	window	ablak méretének megváltozása.
select	Event.SELECT	onSelect	type, target	Text, Textarea	adott HTML elem szövegtartalmának kiválasztása.
submit	Event.SUBMIT	onSubmit	type, target	Form	HTML űrlap jóváhagyása. Ha itt hamis értéket adunk vissza, akkor nem kerül elküldésre az űrlap tartalma.
unload	Event.UNLOAD	onUnload	type, target	window	adott HTML oldal elhagyása.

G.7. ábra: JavaScript események 4.

- pageY : az egérkurzor függőleges pozíciója a HTML oldalon belül képpontokban megadva.
- screenX : az egérkurzor vízszintes pozíciója a képernyőn képpontokban megadva.
- screenY : az egérkurzor függőleges pozíciója a képernyőn képpontokban megadva.
- target : azon DOM objektum, ahol az esemény eredetileg keletkezett.
- type : az esemény neve.
- which : az eseményt kiváltó gomb ASCII kódja, vagy az eseményt kiváltó egérgomb kódja:
  - 1 : bal egérgomb
  - 2 : középső egérgomb
  - 3 : jobb egérgomb
- width : az aktuális ablak/frame szélessége képpontokban megadva.
- x : ugyanaz, mint layerX.
- y : ugyanaz, mint layerY.

navigator : a böngészőt reprezentáló objektum

A JavaScript mindig automatikusan létrehoz egy navigator objektumot, amely az aktuálisan használt böngészőprogramot reprezentálja. Segítségével a böngésző tulajdonságait lehet lekérdezni.

A navigator eseménykezelői

Nincsen semmilyen eseménykezelője sem.

A navigator alapértelmezett mezői

Ezen mezők mindegyike csak olvasható.

- appCodeName : a böngészőprogram kódneve.
- appName : a böngészőprogram neve.
- appVersion : a böngészőprogram verziója. A verziószám után zárójelben rendszerint a böngészőt futtató platform neve, valamint egyéb információk találhatók.
- language : a böngésző aktuálisan kiválasztott nyelvét jelölő kétbetűs kód, esetleges aláhúzásjellel és országkóddal kiegészítve.
- mimeTypes : a böngésző által támogatott MIME típusokat reprezentáló MimeType objektumok tömbje. A tömb elemeire a MIME típus nevével is hivatkozni lehet.
- platform : azon operációs rendszer neve, amelyre a böngészőprogramot fordították.
- plugins : a böngésző által támogatott segédprogramokat, úgynevezett *plug-in*-eket reprezentáló Plugin objektumok tömbje. A tömb elemeire a plug-in nevével is hivatkozni lehet. A tömb rendelkezik még egy további metódussal is (refresh), melynek segítségével aktualizálható a telepített plug-inek listája. Ezen metódus egyetlen logikai paramétert vár, amely megadja, hogy az aktualizálás után újratöltődjenek-e azon HTML lapok, melyek beágyazott objektumot tartalmaznak (<EMBED>). A böngésző futása közben telepített plug-inek csak azután használhatóak, miután újraindítottuk a böngészőt, vagy meghívjuk ezen refresh metódust.
- userAgent : HTTP fejlécben a webszerverhez küldött kliens-azonosító szöveg.

A navigator alapértelmezett metódusai

- javaEnabled : logikai értékként visszaadja, hogy a böngészőben engedélyezve van-e a Java használata.
- preference(*név*[, *érték*]) : lekérdezi/beállítja a megadott nevű böngészőopciót. Értéket megadva lehet az opciót beállítani, ilyenkor a visszatérési érték megegyezik az újonnan beállított értékkel. A lehetséges böngészőopciók a következők:
  - general.always_load_images : logikai érték, a képek automatikus letöltését szabályozza.
  - security.enable_java : logikai érték, a Java használatát szabályozza.
  - javascript.enabled : logikai érték, a JavaScript használatát szabályozza.
  - browser.enable_style_sheets : logikai érték, HTML stílusok használatát szabályozza.
  - autoupdate.enabled : logikai érték, az automatikus programfrissítés használatát szabályozza (SmartUpdate).
  - network.cookie.cookieBehavior : 0 - minden cookie-t elfogad. 1 - csak azon cookie-t fogadja el, amelyik ugyanahhoz a szerverhez kerül vissza, ahonnan az származik. 2 - nem engedélyezi a cookie-k fogadását.
  - network.cookie.warnAboutCookies : logikai érték, cookie fogadásakor történő figyelmeztetés megjelenését szabályozza.
- savePreferences : elmenti a felhasználó aktuális böngészőopció beállításait. Ez a böngészőprogram bezárásakor automatikusan megtörténik.

MimeType : a böngésző támogatott típusait reprezentáló objektum

Egy MimeType objektum egy MIME (Multipart Internet Mail Extension) típust reprezentál. Ilyen objektumokat a böngésző automatikusan hoz létre, az összes általa támogatott MIME típus reprezentálására.

A MimeType eseménykezelői

Nincsen semmilyen eseménykezelője.

A MimeType alapértelmezett mezői

- description : az aktuális MIME típus szöveges leírása.
- enabledPlugin : a MIME típus megjelenítését végző plug-in. Adott MIME típus megjelenítésére több plug-in is képes lehet, de csak egyet lehet a megjelenítéshez bekonfigurálni. Ezen mező alapján könnyen eldönthető, hogy telepítve van-e egy kérdéses plug-in:

```
function pluginInstalled(mimetype) {
         mimetype = navigator.mimeTypes[mimetype]
         if (mimetype) return mimetype.enabledPlugin
         return false
}
```

- suffixes : adott MIME típusú fájlokat jelölő fájlkiterjesztések vesszővel ellátott listája.
- type : a típus MIME típusát megadó szöveg.

A MimeType alapértelmezett metódusai

Nincs más metódusa az Object-nél tárgyaltakon kívül.

Plugin : a böngésző által támogatott típusokat megjelenítő segédprogramokat reprezentáló objektum

Egy Plugin objektum a böngésző által támogatott MIME típusok megjelenítését végző segédprogramot, programmodult reprezentál. Ilyen objektumokat a böngésző az összes általa támogatott MIME típushoz automatikusan létrehoz.

Ez az objektum tulajdonképpen egy tömb, melynek elemei az adott segédprogram által támogatott MIME típusokat reprezentáló MimeType objektumok. Ezen tömbelemek indexelésére a MIME típus neve is használható. Egy segédprogram ugyanis több MIME típust is támogathat, illetve adott MIME típus megjelenítésére több plug-in is képes lehet. A megjelenítéshez használt aktuális plug-int a MimeType enabledPlugin mezője jelöli ki.

A Plugin eseménykezelői

Nincsen semmilyen eseménykezelője.

A Plugin alapértelmezett mezői

- description : az aktuális plug-in szöveges leírása.
- filename : a plug-in programfájl elérési útvonala.
- name : a plug-in neve.

A Plugin alapértelmezett metódusai

Nincs más metódusa az Object-nél tárgyaltakon kívül.

document : az aktuális HTML oldalt reprezentáló objektum

HTML szinten a <BODY> kulcsszónak feleltethető meg. Az aktuális ablakban megjelenített dokumentumot reprezentáló document objektumot a window.document referencián keresztül lehet elérni.

A document eseménykezelői

Bár az onBlur, onFocus, onLoad és onUnload eseménykezelőket a <BODY> kulcsszónál lehet megadni, azok mégis a window eseménykezelői.

- onClick
- onDblClick
- onKeyDown
- onKeyPress
- onKeyUp
- onMouseDown
- onMouseUp

A document alapértelmezett mezői

- alinkColor : az aktivált hivatkozások (az egérgomb lenyomása utáni, de még az elengedése előtti állapot) színét megadó szövegmező. Megfelel a <BODY> ALINK attribútumának.
- anchors : a dokumentumban definiált hivatkozási pontokat (<A NAME=>) reprezentáló Anchor objektumok tömbje. A tömb elemeire a hivatkozás nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- applets : a dokumentumba beágyazott appleteket (<APPLET>) reprezentáló Applet objektumok tömbje. A tömb elemeire az applet nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- bgColor : a dokumentum háttérszínét megadó szövegmező. Megfelel a <BODY> BGCOLOR attribútumának.
- classes.*osztálynév*.*HTML elem* : HTML stílusokat reprezentáló Style objektumok elérését lehetővé tevő mező. Ezen keresztül lehet elérni a megadott nevű HTML stílusosztályba tartozó, adott HTML elem megjelenítését szabályozó Style objektumot. A HTML elem helyén all-t megadva az összes HTML elem megjelenítését szabályozó Style objektumot lehet elérni. <STYLE> HTML elemek közt a document prefix elhagyható. A következő két példa egymással ekvivalens:

```
<SCRIPT>
   document.classes.MindenPiros.all.color="red"
</SCRIPT>
<STYLE TYPE="text/javascript">
   classes.MindenPiros.all.color="red"
</STYLE>
<P CLASS="MindenPiros">
Ez biza piros.
</P>
```

- cookie : szöveges mező, amely az aktuális HTML oldalhoz tartozó, kliensoldalon tárolt kiegészítő információkat, az úgynevezett cookie-kat reprezentálja. Használatának módjára később még visszatérünk.
- domain : a dokumentumot küldő szerver neve. Csak az azonos domainű szkriptek láthatják egymást. Ezen mező értéke csakis az aktuális értékéből annak eleje törlésével képezhető.
- embeds : a dokumentumba beágyazott objektumok (<EMBED>) tömbje. A tömb elemeire az objektum nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- fgColor : a dokumentum szövegszínét megadó szövegmező. Megfelel a <BODY> TEXT attribútumának.
- forms : a dokumentumban található beviteli űrlapokat (<FORM>) reprezentáló Form objektumok tömbje. A tömb elemeire az űrlap nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- height : a dokumentum magassága képernyőpontokban mérve.
- ids.*azonosító* : HTML stílusokat reprezentáló Style objektumok elérését lehetővé tevő mező. Ezen keresztül lehet elérni a megadott azonosítójú HTML elem megjelenítését szabályozó Style objektumot. <STYLE> HTML elemek közt a document prefix elhagyható. A következő két példa egymással ekvivalens:

```
<SCRIPT>
   document.ids.Piros.color="red"
```

```
</SCRIPT>
<STYLE TYPE="text/javascript">
   classes.ids.Piros.color="red"
</STYLE>
<P ID="Piros">
Ez biza piros.
</P>
```

- images : a dokumentumba beágyazott képeket (<IMG>) reprezentáló Image objektumok tömbje. A tömb elemeire a kép nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- lastModified : a dokumentum utolsó módosításának dátuma szövegként. Ezen dátumot a szerver a HTTP fejlécben adja meg. Mivel a fejléc ezen mezője nem kötelező, előfordulhat, hogy annak értéke üres. Ilyenkor a kapott dátum a 0 értéknek megfelelő dátum, 1970. január elseje lesz.
- layers : a dokumentum HTML rétegeit (<LAYER>, <ILAYER>) reprezentáló Layer objektumok tömbje. A tömb elemeire a réteg nevével, vagy azok mélységbeli 0-val kezdődő sorszámával lehet hivatkozni, ahol a 0 a legmélyebben található réteg.
- linkColor : hivatkozások színét megadó szövegmező. Megfelel a <BODY> LINK attribútumának.
- links : a dokumentum hyperhivatkozásait (<A HREF=>, <AREA HREF=>) reprezentáló Link és Area objektumok tömbje. A tömb elemeire a hivatkozás nevével, vagy azok dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- plugins : a dokumentumba beágyazott objektumok (<EMBED>) megjelenítését végző plug-ineket reprezentáló Plugin objektumok tömbje. A tömb elemeire a plug-in dokumentumbeli, 0-val kezdődő sorszámával lehet hivatkozni.
- referrer : Az adott dokumentumot behívó dokumentum URL-je. Csak akkor van értéke, ha az aktuális dokumentum egy hivatkozáson keresztül került betöltésre, és a szerver elküldi ezt az információt a HTTP fejlécben.
- tags.*HTML elem* : HTML stílusokat reprezentáló Style objektumok elérését lehetővé tevő mező. Ezen keresztül lehet elérni a megadott típusú HTML elem megjelenítését szabályozó Style objektumot. <STYLE> HTML elemek közt a document prefix elhagyható. A következő két példa egymással ekvivalens:

```
<SCRIPT>
   document.tags.H1.color="red"
</SCRIPT>
<STYLE TYPE="text/javascript">
   classes.tags.H1.color="red"
</STYLE>
<H1>
Ez biza piros.
</H1>
```

- title : a dokumentum címét tartalmazó szövegmező. Tartalma megfelel a <TITLE> HTML elemek közt álló szövegnek.
- URL : az aktuális dokumentum teljes URL-jét tartalmazó szövegmező.
- vlinkColor : a már meglátogatott hivatkozások színét megadó szövegmező. Megfelel a <BODY> VLINK attribútumának.
- width : a dokumentum szélessége képpontokban mérve.

A document alapértelmezett metódusai

- captureEvents : a paraméterként megadott eseménymaszknak megfelelő események feldolgozásának átirányítása.
- close : lezárja a dokumentum tartalmát beolvasó adatfolyamot és végrehajtja az addig beolvasott értékek alapján a dokumentum megjelenítését.
- contextual(*kontextus*[, ...],*állítandó stílus*) : adott kontextusban álló HTML stílust reprezentáló Style objektumot kérdezi le. A kontextus és az állítandó stílus paraméterek Style objektumok. A visszaadott érték a kért kontextus esetén használandó stílust reprezentáló Style objektum. <STYLE> HTML elemek közt a document prefix elhagyható. A következő két példa egymással ekvivalens:

```
<STYLE TYPE="text/javascript">
   contextual(document.tags.H1, document.tags.EM).color="blue";
</STYLE>
<SCRIPT>
document.contextual(document.tags.H1,
                    document.tags.EM).color="red";
</SCRIPT>
<H1>Ez biza
<EM>piros</EM>
</H1>
```

- getSelection : az aktuálisan kijelölt szöveget adja vissza.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- open([*MIME típus*[, "replace"]]) : (megadott típusú) adatfolyamot nyit (az aktuális dokumentum-bejegyzést felülírva). Az alapértelmezett MIME típus a text/html.
- releaseEvents : befejezi a megadott eseménymaszknak megfelelő események átirányítását.
- routeEvent : folytatja a paraméterben megadott esemény feldolgozását.
- write : a megadott paramétereket szövegesen kiírja az aktuális adatfolyamra.
- writeln : a megadott paramétereket és egy sorvége jelet szövegesen kiír az aktuális adatfolyamra.

1. példa: Dinamikusan generált HTML

A JavaScript legfontosabb felhasználási területe a dinamikus HTML generálás. Ez azt jelenti, hogy mivel a <SCRIPT> kulcsszavak közé foglalt JavaScript kód akkor hajtódik végre, amikor a böngésző az oldal megjelenítésekor eléri azt, és JavaScriptből lehetőségünk van az aktuálisan megjelenítés alatt álló dokumentumot adatfolyamként elérni, a szkript ezen adatfolyamra dinamikusan küldhet ki adatokat. Ennek eredménye, hogy a HTML oldal megtekintésekor nem csak az kerül megjelenítésre, amit az oldal forrásfájlja statikusan tartalmaz, hanem a szkriptek dinamikusan generált eredményei is láthatóvá vállnak. Mindez csak a dokumentum betöltése közben működik. Egy HTML oldal megtekintésekor ugyanis a böngésző megnyit egy text/html MIME típusú adatfolyamot, amely a tartalmát a megadott URL-ről veszi. Ezen adatfolyam feldolgozása jelenti az oldal megjelenítését. Ha JavaScriptből hozzáfűzünk valamit ezen adatfolyamhoz a write/writeln metódusokkal, az ugyanúgy megjelenítésre kerül, mintha azt maga a megjelenítendő HTML oldal tartalmazná. A HTML adatfolyam végének elérésekor automatikusan lezárul az adatfolyam, a dokumentum megjelenítése befejeződik. Épp ezért eseménykezelőkön belül már nincs nyitott adatfolyam, ott nem élhetünk a dinamikus generálás lehetőségével. Viszont bármikor megnyithatunk egy új adatfolyamot a open metódus segítségével. Itt

a paraméterenként megadható MIME típus dönti el, hogy a rákövetkező write/writeln-ek kimenete hogyan kerül feldolgozásra:

- text/html : ez az alapértelmezett típus. Az adatfolyamot mint HTML szöveget a böngésző fogja értelmezni és megjeleníteni.
- text/plain : Az adatfolyamot mint ASCII szöveget a böngésző fogja megjeleníteni.
- image/gif : Az adatfolyamot mint bináris gif képet a böngésző fogja értelmezni és megjeleníteni.
- image/jpeg : Az adatfolyamot mint bináris jpeg képet a böngésző fogja értelmezni és megjeleníteni.
- image/x-bitmap : Az adatfolyamot mint bináris képet a böngésző fogja értelmezni és megjeleníteni.
- Minden más típust az azt megjeleníteni képes plug-in fog feldolgozni, azaz a write / writeln kimenetét ezen plug-in fogja megkapni.

A megnyitott adatfolyamra a write/writeln metódusokkal lehet adatot küldeni. Ha ezen metódusok meghívásakor még nincs nyitott adatfolyam, akkor automatikusan megnyitásra kerül egy az alapértelmezett text/html típussal. A megnyitott adatfolyamot mindig zárjuk le a close metódussal, mert csak ekkor garantált a teljes adatfolyam feldolgozása. HTML adatfolyam újranyitása az aktuális oldal megjelenítésének törlését vonja maga után.

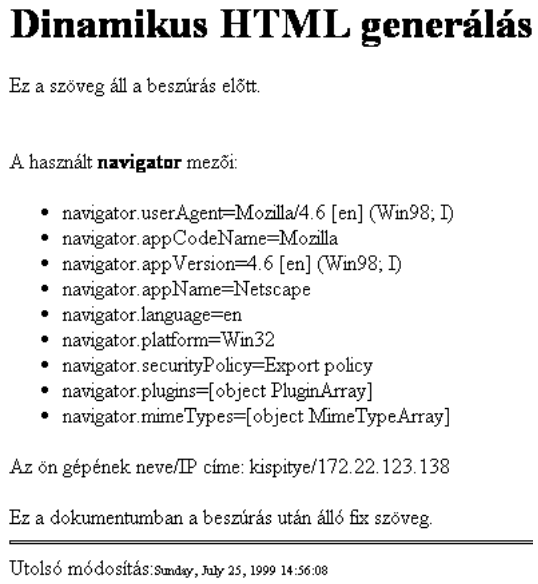
A következő példa dinamikusan kilistázza a felhasznált böngészőprogramot reprezentáló navigator objektum mezőit.

A teljes HTML fájl

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
function props(obj, name, ishtml) {
         var result=""
         if (ishtml) result="<UL>\n"
         for (var i in obj) {
             if (ishtml) result+="<LI>"
             result+=name+"."+i+"="+obj[i]+"\n"
         }
         if (ishtml) result+="</UL>\n"
         return result
}
function getLastModified() {
         lastmoddate = Date.parse(document.lastModified)    // dátummá alakítás
         if(lastmoddate == 0) return "???"
         else return new Date(lastmoddate).toLocaleString()
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<H1>Dinamikus HTML generálás</H1>
Ez a szöveg áll a beszúrás előtt.<P>
<SCRIPT>
<!--
    document.writeln("<BLINK>Ezen szöveg lett beszúrva</BLINK><BR>")
    document.writeln("A használt <b>navigator</b> mezői:")
```

```
    document.writeln(props(navigator, "navigator", true))
    window.onerror=null                                    //nincs hibakijelzés
    document.writeln("Az ön gépének neve/IP címe: "+
                     java.net.InetAddress.getLocalHost())
//-->
</SCRIPT>
<NOSCRIPT>
<BLINK>
Ez a böngésző nem ismeri (vagy nincs engedélyezve benne) a JavaScript-et!
Esetleg ez Netscape 2.x, ami már tud JavaScript-ezni, csak a NOSCRIPT
attribútumot nem ismeri!
</BLINK>
</NOSCRIPT>
<P>Ez a dokumentumban a beszúrás után álló fix szöveg.
<SCRIPT>
document.write("<HR>Utolsó módosítás:<FONT SIZE=-2>"+getLastModified())
</SCRIPT>
</BODY>
</HTML>
```

Megjegyzés: A továbbiakban a teljes HTML fájlokat már nem közöljük, csakis a számunkra fontos JavaScript részletekre térünk ki.



G.8. ábra: Dinamikusan generált HTML képe

A szkript eredménye

A dokumentum megjelenítésekor a beszúrás előtti és utáni szöveg között megjelenik a beszúrt szöveg, azaz kilistázódnak az oldalt megtekintő felhasználó által használt böngészőprogram jellemzői és azon IP-cím is, ahol a böngészőprogram fut (ha ezt a szkript

meg tudja állapítani). A dokumentumot egy vízszintes vonal alá kiírt 'Utolsó módosítás: *dátum*' szöveg zárja. Megjegyzendő, hogy a dinamikusan beszúrt HTML szöveg a dokumentum nyomtatásakor nem látszik.

window : felsőszintű DOM objektum

A window a böngészőablak egészét leíró alapértelmezett felsőszintű DOM objektum, mivel metódusainak és mezőinek elérésekor nem kell kitenni az objektum nevét. A böngészőprogram mindig automatikusan létrehoz egy ilyen objektumot.

A window eseménykezelői

- onBlur
- onDragDrop
- onError
- onFocus
- onLoad
- onMove
- onResize
- onUnload

A window alapértelmezett mezői

- closed : logikai mező, jelzi, hogy a reprezentált ablak még nyitva van-e. Azért szükséges, mert az ablak becsukásakor az azt reprezentáló objektum nem szűnik meg, annak csak ezen mezője lesz igazra állítva.
- crypto : a böngésző titkosítási szolgáltatásainak elérését biztosító objektum, amely a következő metódusokkal rendelkezik:
  - random : visszaad egy véletlen sztringet, melynek bájtban mért hosszát a metódus paramétereként lehet megadni.
  - signText(*szöveg, mód* [*, jogosultságigazoló, ...*]) : a megadott szöveghez digitális aláírást csatol a kiválasztott *jogosultságigazolás* alapján, ezen szöveg lesz a metódus visszatérési értéke. Ha hiba lépett fel a művelet közben, a visszatérési érték az "error:" prefixszel kezdődik. A *mód* értéke "ask" vagy "auto" lehet. Az első esetben a felhasználónak kell kiválasztania interaktívan egy jogosultságigazolást, míg a második esetben a böngésző automatikusan választja ki azt a további megadott paraméterek közül.
- defaultStatus : alapértelmezett státuszszöveg; akkor jelenik meg, ha nincs semmi más üzenet megjelenítve a státuszsorban.
- document : a HTML oldalt reprezentáló document objektum.
- frames : a HTML oldalt felépítő Frame objektumok tömbje.
- history : az aktuális ablakban korábban megtekintett URL-eket reprezentáló History objektum.
- innerHeight : az ablak belsejének magassága képpontokban mérve.
- innerWidth : az ablak belsejének szélessége képpontokban mérve.
- length : ugyanaz, mint frames.length.
- location : az aktuális URL-t reprezentáló Location objektum.
- locationbar : a böngésző helykiválasztó sorát reprezentálja, ahol a könyvjelzők és az aktuális URL is kijelzésre kerül. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, amely a böngésző említett területének láthatóságát adja meg.

- menubar : a böngésző menüsorát reprezentálja. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, amely a böngésző említett területének láthatóságát adja meg.
- name : az ablak neve. Ezen név alapján történik egy ablak azonosítása.
- offscreenBuffering : logikai mező, megadja, hogy az ablak megjelenítése az úgynevezett offscreen technológiával történik-e.
- opener : az ablakot megnyitó ablak objektumra referencia. Ha egy ablakot nem az open metódussal nyitottunk meg, akkor opener=null.
- outerHeight : az ablak magassága képpontokban mérve.
- outerWidth : az ablak szélessége képpontokban mérve.
- pageXOffset : az ablakban megjelenített dokumentum ezen vízszintes pozíciója esik az ablak bal felső sarkába.
- pageYOffset : az ablakban megjelenített dokumentum ezen függőleges pozíciója esik az ablak bal felső sarkába.
- parent : ezen ablakot, mint FRAME-et tartalmazó ablakobjektumra referencia. Normál ablakoknál (azaz nem FRAME esetén) parent=self.
- personalbar : a böngésző személyes könyvjelzőket tartalmazó sorát reprezentálja. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, amely a böngésző említett területének láthatóságát adja meg.
- screenX : az ablak bal felső sarkának vízszintes koordinátája.
- screenY : az ablak bal felső sarkának függőleges koordinátája.
- scrollbars : a böngésző görgetősávjait reprezentálja. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, amely a böngésző említett területének láthatóságát adja meg.
- self : az aktuális ablakobjektumra referencia.
- status : a böngésző státuszsorában megjelenő szöveget adja meg.
- statusbar : a böngésző státuszsorát reprezentálja. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, ami a böngésző említett területének láthatóságát adja meg.
- toolbar : a böngésző vezérlő-gombsorát reprezentálja. Ezen objektum csak egyetlen logikai mezővel (visible) rendelkezik, ami a böngésző említett területének láthatóságát adja meg.
- top : a legfelsőbb szintű HTML oldalt tartalmazó ablakobjektumra referencia. Normál ablakoknál (azaz nem FRAME esetén) top=self.
- window : ugyanaz, mint self.

A window alapértelmezett metódusai

- alert : a paraméterként megadott szöveget egy figyelmeztető böngészőablakban megjeleníti. Az ablak csak egy OK gombot tartalmaz.
- atob : a paraméterként megadott base-64 kódolással kódolt szöveget visszakódolja, ez lesz a visszatérési érték.
- back : visszalép az előző dokumentumra. Ugyanaz, mintha a felhasználó megnyomta volna a visszalépés gombot.
- blur : elveszi a fókuszt.
- btoa : a paraméterként megadott sztringet elkódolja base-64 kódolással, ez lesz a visszatérési érték.
- captureEvents : a paraméterként megadott eseménymaszknak megfelelő események feldolgozásának átirányítása.

- clearInterval : megszünteti a paraméterként megadott azonosítójú ismétlődő függvényhívást.
- clearTimeout : megszünteti a paraméterként megadott azonosítójú időzített függvényhívást.
- close : becsukja az ablakot. Paraméterként egy ablakreferenciát lehet megadni, ennek hiányában az aktuális ablak lesz becsukva.
- confirm : a paraméterként megadott szöveget egy dialógus böngészőablakban megjeleníti. Az ablak egy OK és egy Cancel gombot tartalmaz. Ha az OK gombbal csukjuk be a dialógust, akkor igaz lesz a visszatérési érték, különben pedig hamis lesz.
- disableExternalCapture : megszünteti az enableExternalCapture hatását.
- enableExternalCapture : engedélyezi az eseményátirányítást abban az esetben is, ha egy ablak több FRAME-et tartalmaz különböző szerverekről.
- find([*szöveg*[*, kis=nagybetű, visszafelé*]]) : a megadott szöveg keresése [kis/nagybetűket azonosnak véve, ha a második, logikai paraméter igaz, illetve visszafelé, ha a harmadik, logikai paraméter igaz]. Ha nem adunk meg paramétert, akkor a kereső dialógus fog előjönni. A visszatérési érték igaz, ha sikeres a keresés.
- focus : megkapja a fókuszt.
- forward : továbblép a következő dokumentumra. Ugyanaz, mintha a felhasználó megnyomta volna a tovább gombot.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- home : A kiindulási dokumentum megnyitása. Ugyanaz, mintha a felhasználó megnyomta volna a „haza” gombot.
- moveBy : az ablakot relatívan elmozgatja a két megadott paraméter(+x, +y) értéke alapján.
- moveTo : az ablakot elmozgatja a két megadott paraméter(x, y) által kijelölt helyre.
- open(*URL, név*[, opciók]) : új böngészőablak nyitása a megadott névvel. Visszatérési értékként megkapjuk az új ablakobjektumot. A megadott URL lehet üres sztring is, ami egy teljesen üres ablak megnyitását eredményezi. Ablak neve csak alfanumerikus karakterekből és az aláhúzásjelből állhat. Ha a megadott nevű ablak már létezik, akkor nem fog új ablak kinyílni, hanem a megadott nevűben fog a kért dokumentum megjelenni. A megadható opciókat egy sztring tartalmazza, az opciók pedig vesszővel elválasztva vannak felsorolva a következő alakban: *opció=érték*. Az opció sztringben nem használható a szóköz karakter. Logikai opció esetén elég csak az opció nevét megadni, vagy értékként használjunk yes/no-t, illetve 1/0-t. Ha nem adunk meg opciókat, akkor minden logikai típusú opció igazra állítódik. De ha adunk meg értékeket, akkor minden logikai opció (kivéve titlebar és hotkeys) alapértelmezett értéke a kikapcsolt állapot lesz. A kért ablak megnyitása után a nyitáskor beállított opciók már nem változtathatóak meg. A megadható opciók a következők:
  - alwaysLowered : logikai opció, ami azt jelzi, hogy az ablak mindig minden más ablak alatt lesz látható.
  - alwaysRaised : logikai opció, ami azt jelzi, hogy az ablak mindig minden más ablak felett lesz látható.
  - dependent : logikai opció, ami azt jelzi, hogy az ablak az aktuális ablak gyerekeként jöjjön-e létre. Ilyenkor a szülő ablak becsukása automatikusan maga után vonja a gyerekablak becsukását.
  - directories : logikai opció, a böngésző könyvjelző sorának láthatóságát szabályozza.
  - hotkeys : logikai opció, a böngésző billentyűkombinációinak engedélyezését szabályozza menü nélküli ablak esetén.

  - innerHeight : az ablak belsejének magassága képpontokban mérve.
  - innerWidth : az ablak belsejének szélessége képpontokban mérve.
  - location : logikai opció, a böngésző aktuális URL kijelzőjének láthatóságát szabályozza.
  - menubar : logikai opció, a böngésző menüjének láthatóságát szabályozza.
  - outerHeight : az ablak magassága képpontokban mérve.
  - outerWidth : az ablak szélessége képpontokban mérve.
  - personalbar : logikai opció, a felhasználó könyvjelzőinek láthatóságát szabályozza.
  - resizable : logikai opció, az ablak méretezhetőségét adja meg.
  - screenX : az ablak bal felső sarkának képernyő-koordinátája.
  - screenY : az ablak bal felső sarkának képernyő-koordinátája.
  - scrollbars : logikai opció, a böngésző görgetősávjainak láthatóságát szabályozza.
  - status : logikai opció, a böngésző státuszsorának láthatóságát szabályozza.
  - titlebar : logikai opció, az ablak fejlécének láthatóságát szabályozza.
  - toolbar : logikai opció, a böngésző vezérlőgomb sorának láthatóságát szabályozza.
  - z-lock : logikai opció, ami azt jelzi, hogy az ablak aktiválásakor az nem fog más ablakok fölé kerülni.

- print : kinyomtatja az ablak aktuális tartalmát.
- prompt(*szöveg*[, *alapérték*]) : a paraméterben megadott szöveget egy beviteli böngészőmezőben jeleníti meg [a bevitt értéket kezdetben az alapértékre állítja]. Visszatérési értéke a beviteli ablakban megadott szöveg; null lesz, ha megszakítjuk a bevitelt.
- releaseEvents : befejezi a megadott eseménymaszknak megfelelő események átirányítását.
- resizeBy : átméretezi az ablakot a megadott két relatív paraméter (+x, +y) alapján.
- resizeTo : átméretezi az ablakot a megadott méretre (x, y).
- routeEvent : folytatja a paraméterben megadott esemény feldolgozását.
- scrollBy : görgeti az ablak tartalmát a megadott két relatív paraméter (+x, +y) alapján.
- scrollTo : elgörgeti az ablak tartalmát úgy, hogy a megadott pont (x, y) kerül a bal felső sarokba.
- setHotKeys : a paraméterként megadott logikai értékkel lehet szabályozni, hogy használhatók-e a böngésző billentyűkombinációi, ha nem látszik a menüsor.
- setInterval(*"kifejezés"* vagy *függvény, idő*[, *argumentum, ...*]) : a megadott kifejezést vagy függvényt [a megadott paramétereket átadva] ismételten végrehajtja az ezredmásodpercben megadott időközönként. A metódus azonnal visszatér, visszatérési értékként pedig egy azonosítót kapunk vissza, ezzel lehet később az ütemezést leállítani.
- setResizable : a paraméterként megadott logikai értékkel lehet az ablak átméretezhetőségét szabályozni.
- setTimeout(*"kifejezés"* vagy *függvény, idő*[, *argumentum, ...*]) : a megadott kifejezést vagy függvényt [a megadott paramétereket átadva] egyszer végrehajtja az ezredmásodpercben megadott idő elteltével. A metódus azonnal visszatér, visszatérési értékként pedig egy azonosítót kapunk vissza, ezzel lehet később az ütemezést leállítani.
- setZOptions : megadja az ablak mélységbeli viselkedését. Paraméter elhagyása esetén az alapértelmezett viselkedés lesz visszaállítva. A paraméterként megadott szöveg a következő lehet:

  - alwaysLowered : az ablak mindig minden más ablak alatt lesz látható.
  - alwaysRaised : az ablak mindig minden más ablak felett lesz látható.
  - z-lock : az ablak aktiválásakor az nem fog más ablakok fölé kerülni.

- stop : megállítja az aktuális letöltést.

2. példa: Görgetett státuszsor

Ez a teljes példa megmutatja, hogyan lehet szöveg görgetett megjelenítését elvégezni a státuszsorban. Ugyanakkor azt is láthatjuk, hogy hogyan kell a böngésző beépített dialógusait használni és hogy lehet új böngészőablakot megnyitni.

A JSWindowTest.html fájl

```
<HTML>
<HEAD>
<SCRIPT>
document.write("<TITLE>")                // ablak nevének kijelzése a fejlécben
if (window.name == "" || window.name == "windowtest") {             // fő ablak
    window.name = "windowtest"
    document.write("JavaScript window teszt")
} else
    document.write(window.name)
document.write("</TITLE>")

var timerID = null
var timerRunning = false
var scrollingText = "Alapértelmezett görgetett szöveg"

function stopScroll() {                                  // görgetés megállítása
    if(timerRunning) clearTimeout(timerID)
    timerRunning = false
}

function startScroll() {                                   // görgetés indítása
    if (!timerRunning) timerID = setInterval("scrollText()",200)
    timerRunning = true
}

function scroll(text){                              // görgetendő szöveg megadása
    stopScroll()                                             // nehogy fusson már
    if (text==null) {
        text = "Nincs megadva szöveg!"
        alert(text)
    }
    scrollingText = " "+text+" "                // legalább kettő hosszú legyen
    startScroll()
    scrollText()
}

function scrollText() {                                  // görgetés egyhelyben
    scrollingText = scrollingText.substr(1)+scrollingText.charAt(0)
    status = scrollingText
}

ablakszam = 0          // ablaknévhez ID, hogy mindig új ablak legyen megnyitva
```

```
function inditas(event) {
    var result = routeEvent(event)
    if (confirm("Akar új ablakot nyitni?")) {
        window.open(window.location, window.name+"_"+(++ablakszam),
            "status,innerWidth=400,innerHeight=300,resizable")
    } else window.onload()                  // görgetendő szöveg újra megadható
    return result
}
</SCRIPT>
</HEAD>
<BODY onload="
scroll(prompt('Adja meg a görgetendő szöveget:', scrollingText))
">
<HR>
<H1>Window teszt</H1>
Klikkeljen az egérrel az ablakra!
<HR>
<SCRIPT>
    window.captureEvents(Event.MOUSEUP)
    window.onMouseUp = inditas
</SCRIPT>
A státuszsorban hogy görög a szöveg!
</BODY>
</HTML>
```

G.9. ábra: Státuszsor görgetése, böngésző dialógusok kinézete

A szkript eredménye

A dokumentum megjelenítésekor meg kell adni a státuszsorban görgetendő szöveget. Ha a szöveg megadását félbeszakítjuk, akkor egy dialógusablak figyelmeztetni fog. A megadott szöveg a böngésző státuszsorában fog görgetetten megjelenni. Ha rákattintunk az egérrel a böngészőablakra, akkor előbb válaszolnunk kell azon kérdésre, hogy új ablakot akarunk-e nyitni, majd újra megadhatjuk a görgetendő szöveget, ami igenlő válasz esetén az új ablakban, míg ellenkező esetben az aktuális ablak státuszsorában fog görgetetten megjelenni.

Frame : osztott böngészőablakok

A FRAMESET és FRAME HTML kulcsszavak lehetővé teszik, hogy egy böngészőablakban több HTML oldal is megjelenítésre kerülhessen. Minden FRAME egy külön HTML oldalt tartalmaz, ezek megjelenítése a böngészőablak vízszintesen és/vagy függőlegesen megosztott területein, az úgynevezett *frame*-ekben történik. Egy frame tartalmazhat további frame-eket, egy valódi tartalmazási hierarchiát kialakítva így.

Minden framet egy Frame objektum reprezentál, amely technikailag teljesen megegyezik a window objektummal. Egy HTML oldal frame-jei a window objektum frames tömbjén keresztül érhető el. Ezen tömb indexelése történhet a frame nevével vagy annak definíció szerinti 0-val kezdődő sorszámával. Egy frame-et és egy normál HTML oldalt reprezentáló window objektumok a következőkben különböznek egymástól:

- Normál window esetén parent=top=self, míg frame-ek esetén ezen referenciák a tartalmazási hierarchiát reprezentálják.
- Frame esetén a defaultStatus és status mezőknek adott értékek csak akkor jelennek meg a böngésző státuszsorában, ha a kurzor a megfelelő frame-ben tartózkodik.
- Frame-eket nem lehet bezárni a close metódussal.
- HTML szintjén nem lehet a frame-nek onBlur és onFocus eseménykezelőt megadni, csak JavaScripten keresztül.
- Frame-ekre lehet azok nevével hivatkozni, míg ablaknál csak az open metódusnál van szerepe a névnek.

3. példa: Online vásárlás

Az Internet kereskedelmi alkalmazásának legjobb példái az olyan hálózati alkalmazások, ahol a hálózaton keresztül válogathatunk az árukból, képzeletbeli bevásárlókocsinkba összegyűjtjük, amit meg akarunk rendelni, majd a rendelést szintén a hálózaton keresztül adjuk le. Ezen példa az ilyen típusú, *online vásárlás*nak nevezett alkalmazások egy lehetséges (egyszerű) változatát szemlélteti.

Az alkalmazást frame-ek segítségével valósítjuk meg. A JavaScript lehetőségeit pedig a bevásárlókocsink kezeléséhez használjuk ki.

A főoldal HTML forrása

```
<HTML>
<HEAD>
<TITLE>JSFrameTest</TITLE>
<SCRIPT>
megvettlista = []                                   // megvásárolt elemek tömbje

function vesz(neve, ara) {                              // adott áru megvétele
    while (true) {
```

```
        db = prompt("Hány darab "+neve+"-t akar venni?",1)
        if (db == null) return                                  // megszakítva
        dbszam = parseInt(db)
        if (isNaN(dbszam) || dbszam<1) alert("Érvénytelen szám: "+db)
        else break                              // érvényes darabszám lett megadva
    }
    var index = 0
    for (; index<megvettlista.length; index++) {     // már vettünk ilyen árut?
        adatok = megvettlista[index].split(":")
        if (adatok[0]==neve) {                              // igen, már van ilyen
            if (confirm("Már van "+adatok[2]+" db. "+neve+
                        " a bevásárlókocsiban.\n"+
                        "Akar még "+dbszam+" db.-ot venni hozzá?")) {
                dbszam += parseInt(adatok[2])
                break
            } else return
        }
    }
    megvettlista[index] = neve+":"+ara+":"+dbszam     // megvett áru eltárolása
    alert(dbszam+" db. "+neve+" bekerült a bevásárlókocsiba.")
}
function nemveszmeg(mit) {                    // adott tételt mégsem vesszük meg
    for (var i=0; i<megvettlista.length; i++)
        if (megvettlista[i] == mit) {                     // töröljük a listából
            megvettlista.splice(i, 1)
            break
        }
}
function kilistaz(hova) {  // bevásárlókocsi tartalmának listázása táblázatként
    hova.writeln("<TABLE BORDER=\"1\" CELLPADDING=\"3\" "+
                 "CELLSPACING=\"0\" WIDTH=\"100%\">")
    hova.writeln("<TR><TD>Név<TD>Ár<TD>Darabszám<TD>Összeg")

    osszosszeg = 0                                       // összegzést is végzünk
    osszdarabszam = 0
    megvettlista.sort()
    for (var i=0; i<megvettlista.length; i++) {
        adatok = megvettlista[i].split(":")
        osszeg = adatok[2] * adatok[1]
        osszosszeg += osszeg
        osszdarabszam += parseInt(adatok[2])
        hova.writeln("<TR><TD>"+adatok[0]+"<TD>"+adatok[1]+"<TD>"+adatok[2]+
                     "<TD>"+osszeg+"<TD>")
        hova.writeln("<FORM><INPUT TYPE=\"button\" VALUE=\"töröl\" onclick=\""+
                     "parent.nemveszmeg('"+megvettlista[i]+"');"+
                     "window.history.go(0)\"></FORM>")   // lista újrarajzolása
    }
    hova.writeln("<TR><TD COLSPAN=2>Összesen<TD>"+
                 osszdarabszam+"<TD>"+osszosszeg)
    hova.writeln("</TABLE>")
}
</SCRIPT>
</HEAD>
<FRAMESET rows="100,*">
<FRAME NAME="header" SRC="JSFrameTestHeader.html">
<FRAMESET cols="20%,80%">
<FRAME NAME="menu" SRC="JSFrameTestMenu.html">
```

```
<FRAME NAME="work" SRC="JSFrameTestHeader.html">
</FRAMESET>
</FRAMESET>
<NOFRAMES>
<BLINK>
Ez a böngésző nem ismeri a FRAME-eket!
</BLINK>
</NOFRAMES>
</HTML>
```

G.10. ábra: Online üzlet alkalmazás

A szkript eredménye

A teljes böngészőablakot három részre osztottuk fel: felül mindig a cég logója látható, ahol éppen vásárolgatunk. Bal oldalt egy menürendszer található, itt lehet az áruk kategóriáit kiválasztani. A maradék terület pedig a munkaterület, itt lehet például a nekünk tetsző árut kiválasztani, vagy a már kiválasztott áruk listáját megtekinteni és a rendelést feladni.

Ha kiválasztunk egy árut, meg kell adni, hogy hány darabot szeretnénk belőle vásárolni. Ha már korábban is kijelöltük ugyanezt az árut, a program figyelmeztetni fog. Miután befejeztük a vásárlást, a bevásárlókocsi hivatkozást kiválasztva kilistázódik minden áru, amit meg szeretnénk venni. Itt lehetőségünk van még a lista módosítására, vagy a megrendelés elküldésére.

A JavaScriptet, mint már említettük, a bevásárlókocsink kezeléséhez használtuk. A megvásárolt áruk listáját a frame hierarchia tetején definiált megvettlista tömbbel reprezentáljuk. A tömb szöveges formában tartalmazza az árukat. Minden egyes tömbelem tartalmazza az áru nevét, árát és a megrendelt darabszámot, mindezt kettőspontokkal elválasztva. Áru vétele a főframe vesz metódusának meghívásával történik. Ezen metódus első paramétere az áru neve, második paramétere pedig annak ára. Érdemes például egy gomb eseménykezelőjéből meghívni:

```
onClick="parent.vesz('kicsikocsi', 1)"
```

Itt kihasználjuk a frame-ek azon lehetőségét, hogy azok a tartalmazási hierarchián keresztül elérik egymást. A metódus a megvett árut felveszi a listába, vagy módosítja a megvett darabszámot.

G.11. ábra: Bevásárlókocsi tartalma

A bevásárlókocsi tartalmának kilistázása dinamikus HTML generálással történik. A bevásárlókocsi hivatkozás céljaként megadott HTML oldal magát a listát a következő formában tartalmazza:

```
<SCRIPT>
parent.kilistaz(document)
</SCRIPT>
```

A táblázat utolsó oszlopába egy nyomógomb kerül, melynek segítségével az adott tétel törölhető a listáról. A törlés az adott tétel kitörlését jelenti a listát reprezentáló tömbből, majd újra megjelenítjük a megváltozott lista tartalmát.

Location : az aktuális URL-t reprezentáló objektum

Minden window objektumhoz tartozik egy, az aktuálisan megjelenített URL-t reprezentáló Location objektum, amelyet a window.location referenciával lehet elérni. Ezen objektum az aktuális URL különböző összetevőit reprezentálja. Bármely összetevő megváltoztatása az új URL-en található dokumentum betöltését vonja maga után. Ha egy ilyen objektumnak szöveges értéket adunk értékül, az ugyanolyan hatású, mintha a href mezőt állítottuk volna be.

A Location eseménykezelői

Nincsen semmilyen eseménykezelője.

A Location alapértelmezett mezői

- hash : HTTP URL-ek esetén az oldalon belüli aktuális címke (<A NAME=>).
- host : Az URL-lel megcímzett számítógép neve.
- hostname : Az URL-lel megcímzett számítógép neve és annak portja.
- href : A teljes URL szöveges reprezentációja.
- pathname : Az URL-lel megcímzett objektum kiszolgálón belüli elérési útvonala.
- port : Az URL portja.
- protocol : Az URL protokollja kettősponttal a végén.
- search : HTTP URL-ek esetén az URL-hez továbbítandó paraméterek az őket bevezető kérdőjellel együtt.

A Location alapértelmezett metódusai

- reload : újra betölti az aktuális URL tartalmát. Egy logikai értéket lehet paraméterként megadni, ami jelzi, hogy mindenképp a szerverről újra le kell hozni a dokumentumot (igaz érték esetén), vagy jöhet az akár a cache-ből is.
- replace : a szöveges paraméterként megadott URL-t betölti úgy, hogy nem képződik új bejegyzés az URL-listában, azaz a betöltés előtt aktuális dokumentumhoz nem lehet majd visszatérni a böngésző visszalépés gombjával.

History : az eddig meglátogatott URL-ek reprezentáló objektuma

Minden window objektumhoz tartozik egy, az adott ablakban idáig megjelenített URL-eket reprezentáló History objektum, melyet a window.history referenciával lehet elérni. Ezen objektum az eddig megtekintett URL-eket a megtekintésük sorrendjének megfelelően egy tömbben tárolja. A tömb elemei az URL-ek szöveges reprezentációi azok betöltési sorrendjének megfelelően, azaz a legelőször megtekintett URL indexe 0 lesz. Ha egy ilyen objektumot szöveg típusra konvertálunk, akkor az eddig meglátogatott URL-eket kilistázó HTML forrásszöveget kapunk.

A History eseménykezelői

Nincsen semmilyen eseménykezelője.

A History alapértelmezett mezői

- current : az aktuális URL szövegesen.
- length : az URL történetlista hossza.
- next : a következő URL szövegesen.
- previous : az előző URL szövegesen.

A History alapértelmezett metódusai

- back : betölti a megelőző URL-t.
- forward : betölti a következő URL-t.
- go : betölti a megadott URL-t. Szöveges paraméter esetén a listában szereplő, ahhoz leginkább hasonlító URL kerül betöltésre. Ha a paraméter egész típusú, akkor az aktuális URL pozíciójához relatívan megadott pozíciójú URL kerül betöltésre. 0 paraméter esetén az aktuális dokumentum lesz újra betöltve.

Link : hivatkozást reprezentáló objektum

Egy hivatkozást a <A HREF=*URL*> HTML kulcsszó jelöl ki. Adott hivatkozásra klikkelve a megnevezett URL kerül megjelenítésre.

A Link eseménykezelői

- onClick
- onDblClick
- onKeyDown
- onKeyPress

- onKeyUp
- onMouseDown
- onMouseOut
- onMouseOver
- onMouseUp

A Link alapértelmezett mezői

Egy Link objektum rendelkezik a Location objektum minden mezőjével. Ezeken kívül még a következő mezőkkel rendelkezik:

- target : azon ablak neve, ahol a megadott URL-t meg fogja jeleníteni. Megfelel a TARGET attribútumnak.
- text : a hivatkozás látható szövege.
- x : a hivatkozás pozíciója a dokumentumon belül.
- y : a hivatkozás pozíciója a dokumentumon belül.

A Link alapértelmezett metódusai

- handleEvent : a paraméterként kapott esemény feldolgozása.

Area : hivatkozási területet reprezentáló objektum

Egy hivatkozási területet az <AREA HREF=*URL*> HTML kulcsszó jelöl ki. Adott hivatkozási területre klikkelve a megnevezett URL kerül megjelenítésre. Egy Area objektum teljesen megegyezik a Link objektummal, eltérés csak a használható eseménykezelőkben van.

Az Area eseménykezelői

- onDblClick
- onMouseOut
- onMouseOver

A következő példa egy képet egy alsó és egy felső hivatkozási területre oszt fel:

```
<MAP NAME="felezoMap">
   <AREA NAME="teteje" COORDS="0,0,50,25" HREF="javascript:void(0)"
      onMouseOver="window.status='Most a tetején van!';return true"
      onMouseOut="window.status='Most lement a tetejéről';return true">
   <AREA NAME="alja" COORDS="0,25,50,50" HREF="javascript:void(0)"
      onMouseOver="window.status='Most az alján van!';return true"
      onMouseOut="window.status='Most lement az aljáról!';return true">
</MAP>
<IMG SRC="images\hatter.jpg" ALIGN="top" HEIGHT="50" WIDTH="50"
     USEMAP="#felezoMap">
```

Anchor : hivatkozási pontot reprezentáló objektum

Egy hivatkozási pontot a <A NAME=*név*> HTML kulcsszó jelöl ki.

Az Anchor eseménykezelői

Nincsen semmilyen eseménykezelője.

Az Anchor alapértelmezett mezői

- name : a hivatkozási pont neve. Megfelel a NAME attribútumnak.
- text : a hivatkozási pont látható szövege.
- x : a hivatkozási pont pozíciója a dokumentumon belül.
- y : a hivatkozási pont pozíciója a dokumentumon belül.

Az Anchor alapértelmezett metódusai

Nincs más metódusa az Object-nél tárgyaltakon kívül.

Style : HTML elemek stílusát reprezentáló objektum

A Style objektumokkal dinamikus stílusformázás valósítható meg JavaScriptből. Ezen objektumok elérése a document classes, ids és tags mezőin keresztül, illetve annak contextual metódusával történik.

A Style eseménykezelői

Nincsen semmilyen eseménykezelője.

A Style alapértelmezett mezői

- align : a HTML elem igazítását adja meg annak szülőjén belül. Értéke left, right, none valamelyike lehet. Megfelel a HTML stílus float értékének.
- backgroundColor : a HTML elem háttérszíne. Megfelel a background-color HTML stílus értéknek.
- backgroundImage : a HTML elem háttérképének URL-je. Megfelel a HTML stílus background-image értékének.
- borderBottomWidth : a HTML elem kerete aljának szélessége. Értéke szöveges, mérőszámból és mértékegységből áll. Megfelel a HTML stílus border-bottom-width értékének.
- borderColor : a HTML elem keretének színe. Értéke lehet none is, ha nincs szín megadva. Megfelel a HTML stílus border-color értékének.
- borderLeftWidth : a HTML elem kerete bal szélének szélessége. Értéke szöveges, mérőszámból és mértékegységből áll. Megfelel a HTML stílus border-left-width értékének.
- borderRightWidth : a HTML elem kerete jobb szélének szélessége. Értéke szöveges, mérőszámból és mértékegységből áll. Megfelel a HTML stílus border-right-width értékének.
- borderStyle : a HTML elem keretének típusát megadó szövegmező. Értéke a következők valamelyike lehet: none, solid, double, inset, outset, groove, ridge. Megfelel a HTML stílus border-style értékének.
- borderTopWidth : a HTML elem kerete tetejének szélessége. Értéke szöveges, mérőszámból és mértékegységből áll. Megfelel a HTML stílus border-top-width értékének.
- clear : a HTML elemen belüli elemek igazítását adja meg. Értéke a következők valamelyike lehet: left, right, both, none. Megfelel a HTML stílus clear értékének.
- color : a HTML elem szövegének színe. Megfelel a HTML stílus color értékének.

- display : a HTML elem sorfolytonos megjelenítésének módja. Értéke a következők valamelyike lehet: none, block, inline, list-item. Megfelel a HTML stílus display értékének.
- fontFamily : a HTML elem megjelenítéséhez felhasznált font. Értéke szöveges, a felhasználandó fontok vesszővel elválasztott nevét tartalmazza. Ha a lista elején levő font nem elérhető, akkor a következő kerül felhasználásra, ... A következő értékek minden platformon elérhetők: serif, sans-serif, cursive, monospace, fantasy. Megfelel a HTML stílus font-family értékének.
- fontSize : a HTML elem megjelenítéséhez felhasznált font mérete. Értéke szöveges, a következők valamelyike lehet: xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, vagy állhat mérőszámból és mértékegységből, illetve százalékból és százalékjelből. Megfelel a HTML stílus font-size értékének. Alapértelmezett értéke a medium.
- fontStyle : a HTML elem megjelenítéséhez felhasznált font típusa. Értéke szöveges, a következők valamelyike lehet: normal, italic. Megfelel a HTML stílus font-style értékének.
- fontWeight : a HTML elem megjelenítéséhez felhasznált font vastagsága. Értéke szöveges, a következők valamelyike lehet: normal, bold, small, bolder, lighter, vagy egy szám 100 és 900 közt, ahol minél nagyobb a szám, annál vastagabb a font. Megfelel a HTML stílus font-weight értékének.
- lineHeight : HTML elemekből álló sor magassága. Megadható abszolút, vagy az aktuális mérethez viszonyítva százalékosan vagy multiplikatív alakban. Ha értéke normal, a böngésző maga állapítja meg a sor magasságát. Megfelel a HTML stílus line-height értékének.
- listStyleType : listaelemek előtt megjelenő jel. Értéke szöveges, a következők valamelyike lehet: disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none. Megfelel a HTML stílus list-style-type értékének.
- marginBottom : a HTML elem alatt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Ha értéke auto, a böngésző maga állapítja meg annak nagyságát. Megfelel a HTML stílus margin-bottom értékének.
- marginLeft : a HTML elem alatt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Ha értéke auto, a böngésző maga állapítja meg annak nagyságát. Megfelel a HTML stílus margin-bottom értékének.
- marginRight : a HTML elem mellett jobbra kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Ha értéke auto, a böngésző maga állapítja meg annak nagyságát. Megfelel a HTML stílus margin-right értékének.
- marginTop : a HTML elem felett kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Ha értéke auto, a böngésző maga állapítja meg annak nagyságát. Megfelel a HTML stílus margin-top értékének.
- paddingBottom : a HTML elem tartalma és alja közt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Megfelel a HTML stílus padding-bottom értékének.
- paddingLeft : a HTML elem tartalma és bal széle közt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Megfelel a HTML stílus padding-left értékének.
- paddingRight : a HTML elem tartalma és jobb széle közt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Megfelel a HTML stílus padding-right értékének.

- paddingTop : a HTML elem tartalma és teteje közt kihagyandó minimális terület mérete. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Megfelel a HTML stílus padding-top értékének.
- textAlign : a HTML elem szövegének igazítását megadó szöveges mező. Értéke a következők valamelyike lehet: left, right, center, justify. Megfelel a HTML stílus text-align értékének.
- textDecoration : a HTML elem szövegének speciális megjelenítését vezérli. Értéke a következők valamelyike lehet: none, underline, line-through, blink. Megfelel a HTML stílus text-decorations értékének.
- textIndent : bekezdés szélessége. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Megfelel a HTML stílus text-indent értékének.
- textTransform : a HTML elem szövegének betűmérete. Szöveges értéke a következők valamelyike lehet: none, capitalize, uppercase, lowercase. Megfelel a HTML stílus text-transform értékének.
- whiteSpace : megadja, hogy az üres karakterek megjelenjenek-e. Értéke a következők valamelyike lehet: normal, pre. Megfelel a HTML stílus white-space értékének.
- width : a HTML elem szélessége. Megadható abszolút vagy az aktuális mérethez viszonyítva százalékosan. Ha értéke auto, a böngésző maga állapítja meg annak nagyságát. Megfelel a HTML stílus width értékének.

A Style alapértelmezett metódusai

- borderWidths : a négy paraméter rendre a stílus borderTopWidth, borderRightWidth, borderBottomWidth és borderLeftWidth mezőit állítja be.
- margins : a négy paraméter rendre a stílus marginTop, marginRight, marginBottom és marginLeft mezőit állítja be.
- paddings : a négy paraméter rendre a stílus paddingTop, paddingRight, paddingBottom és paddingLeft mezőit állítja be.

Layer : HTML rétegeket reprezentáló objektum

A HTML oldal rétegeit (<DIV>, <LAYER>, <ILAYER>) a document layers tömbjén keresztül lehet elérni. A tömbelemek indexelésére a réteg ID-je is használható.

A Layer eseménykezelői

- onMouseOver
- onMouseOut
- onLoad
- onFocus
- onBlur

A Layer alapértelmezett mezői

- above : az adott réteg felett látszódó réteg.
- background : a háttérképet reprezentáló Image.
- bgColor : a réteg háttérszíne.
- below : adott réteg alatti látszódó réteg.
- clip : a látható felületet megadó vágónégyszög objektum, amely a következő mezőkkel rendelkezik: bottom, height, left, right, top, width.
- document : a rétegen megjelenő HTML dokumentumot reprezentáló objektum.

- left : a réteg pozíciója annak szülőjén belül.
- name : a réteg neve. Megfelel az ID attribútummal megadott értéknek.
- pageX : a réteg pozíciója az oldalon belül.
- pageY : a réteg pozíciója az oldalon belül.
- parentLayer : a szülő réteg.
- siblingAbove : a testvérrétegek közül közvetlenül ezen réteg felett látszódó réteg.
- siblingBelow : a testvérrétegek közül közvetlenül ezen réteg alatt látszódó réteg.
- src : a rétegen látszó HTLM oldal URL-je. Megfelel az SRC attribútumnak.
- top : a réteg pozíciója annak szülőjén belül.
- visibility : a réteg láthatósága. Értéke a következők valamelyike lehet: show, hide, inherit.
- window : a réteget tartalmazó ablakot reprezentáló window vagy Frame objektum.
- x : ugyanaz, mint left.
- y : ugyanaz, mint top.
- zIndex : a réteg elhelyezkedésének mélységét jelölő pozitív szám. Alacsonyabb érték esetén a réteg egyre mélyebben van.

A Layer alapértelmezett metódusai

- captureEvents : a paraméterként megadott eseménymaszknak megfelelő események feldolgozásának átirányítása.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- load : megjeleníti az első, szöveges paraméterben megadott fájl tartalmát a rétegen. A megjelenítés szélességét a második, numerikus paraméterrel lehet állítani.
- moveAbove : az aktuális réteget a paraméterként megadott fölé helyezi. Ezután a két rétegnek azonos lesz a szülőrétege.
- moveBelow : az aktuális réteget a paraméterként megadott alá helyezi. Ezután a két rétegnek azonos lesz a szülőrétege.
- moveBy : a réteget relatívan elmozgatja a két megadott paraméter(+x, +y) értéke alapján.
- moveTo : a réteget elmozgatja a két megadott paraméter(x, y) által kijelölt helyre a szülő rétegen belül.
- moveToAbsolute : a réteget elmozgatja a két megadott paraméter(x, y) által kijelölt helyre az oldalon belül.
- releaseEvents : befejezi a megadott eseménymaszknak megfelelő események átirányítását.
- resizeBy : átméretezi az réteget a megadott két relatív paraméter (+x, +y) alapján.
- resizeTo : átméretezi az réteget a megadott méretre (x, y).
- routeEvent : folytatja a paraméterben megadott esemény feldolgozását.

4. példa: Menü megvalósítása HTML rétegekkel

Mivel a HTML rétegek szabadon pozícionálhatók egy adott rétegen/dokumentumon belül, segítségükkel könnyen megvalósítható a menüktől elvárt funkcionalitás.

A következő példa egy legfeljebb kétszintű menürendszer HTML rétegekkel történő megvalósítását mutatja meg. A menü alapállapotában csak a fő menüpontok látszanak egymás alatt, ezek közül valamelyikre kattintva a kiválasztott főmenüpont alatt megjelennek annak almenüpontjai. Adott fő- és almenüpontok kiválasztottságát, illetve az aktuális menüpontot, amire az egérkurzor éppen mutat, külön grafikus jelek jelölik meg.

Minden menüpontot egy kép reprezentál egy külön HTML rétegbe ágyazva. Ezt a képet pedig egy HTML hivatkozásba rakjuk bele, mivel szeretnénk ez egéreseményeket is feldolgozni. Mivel a menüpont megnyomásának hatására alapértelmezés szerint nem kell semmit sem csinálni, ezért a hivatkozási célnak egy üres JavaScript utasítást (javascript:void(0)) adtunk meg. Az egérkurzor belépésekor megjelöljük az aktuálisan mutatott fő-, illetve almenüpontot, ha azok még nincsenek kiválasztva. Az aktuális, illetve kiválasztott fő- és almenük jelölésére szintén külön rétegeket használunk, ezek láthatóságának és pozíciójának változtatásával érjük el a különböző menüpontok jelölését.

A menüt egy menüpontokat reprezentáló MenuPont objektumokból álló tömb reprezentálja. Egy menüpontot pedig a hozzátartozó kép, az aktiválásakor megjelenítendő státuszsor szövege és a menüpont alatt megjelenő almenüpontok jellemeznek. A példaprogram csak a menü megjelenését szimulálja, tehát adott menüpont kiválasztása nem vált ki semmilyen eseményt sem, de ennek beépítése az adott struktúrába már triviális.

A HTML rétegeket leíró stílusok minden réteget a bal oldalhoz igazítanak, a fő menüpontokat alapértelmezés szerint láthatóvá teszik, míg az almenüpontokat elrejtik. Ezen stílusokat megadó sorokat dinamikusan generáljuk. A HTML rétegeket megadó sorokat szintén dinamikusan generáljuk.

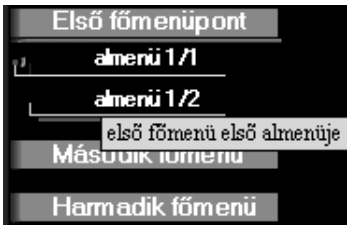
A HTML forrás

```
<HTML>
<HEAD>
<TITLE>Layer teszt</TITLE>
<SCRIPT>
function MenuPont(kep, felirat, almenuk) {              // Konstruktorfüggvény
         this.kep = kep                                  // megjelenítendő kép
         this.felirat = felirat          // stásuszsorban megjelenítendő szöveg
         this.almenuk = almenuk                                 // almenüpontok
}

menu = [                              // az aktuális menüt reprezentáló objektum
       new MenuPont("images/m1.gif", "első főmenü", [
           new MenuPont("images/m1s1.gif", "első főmenü első almenüje", []),
           new MenuPont("images/m1s2.gif", "első főmenü második almenüje", [])
       ] ),
       new MenuPont("images/m2.gif", "második főmenü", [
       ] ),
       new MenuPont("images/m3.gif", "harmadik főmenü", [
           new MenuPont("images/m3s1.gif", "harmadik főmenü első almenüje", [])
       ] )
       ]
menu.fomenu = -1                            // aktuálisan kiválasztott főmenüpont
menu.almenu = -1                            // aktuálisan kiválasztott almenüpont
menu.fomutatoid = "FOMUTATO"          // aktuális főmenüpontot jelölő layer neve
menu.fomutatokep = "images/point.gif"// aktuális főmenüpontot jelölő layer képe
menu.almutatoid = "ALMUTATO"          // aktuális almenüpontot jelölő layer neve
menu.almutatokep = "images/subpoint.gif"  // aktuális főmenüt jelölő layer képe
menu.fovalasztasid = "FOVALASZTAS"    // kiválasztott főmenüt jelölő layer neve
menu.fovalasztaskep = "images/select.gif"    // kiválasztott főmenüt jelölő kép
menu.alvalasztasid = "ALVALASZTAS"    // kiválasztott almenüt jelölő layer neve
menu.alvalasztaskep = "images/subselect.gif" // kiválasztott almenüt jelölő kép
menu.top = 60                                        // menü tetejének pozíciója
menu.fomenukoz = 10                     // főmenüpontok közt kihagyandó távolság
menu.almenukoz = 2                      // almenüpontok közt kihagyandó távolság
```

```
function layerId(fomenu, almenu) {//megadja a menüpontot tartalmazó layer nevét
    if (arguments.length == 1 || almenu < 0) return "M"+fomenu
    else return "M"+fomenu+"S"+almenu
}
function showLayer(id) {                        // adott layer-t láthatóvá teszi
         document.layers[id].visibility = "show"
}
function hideLayer(id) {                                // adott layer-t elrejti
         document.layers[id].visibility = "hide"
}
function showMenu(fomenu, almenu) {// menü megjelenítése, menüpont kiválasztása
         hideLayer(menu.fomutatoid)                   // minden mutató elrejtése
         hideLayer(menu.almutatoid)
         hideLayer(menu.fovalasztasid)
         hideLayer(menu.alvalasztasid)

         if (menu.fomenu>=0)// ha volt már almenü megjelenítve, annak elrejtése
            for (var i = 0; i<menu[menu.fomenu].almenuk.length; i++)
                hideLayer(layerId(menu.fomenu, i))

         var ypos = menu.top
         for (var i=0; i < menu.length; i++) {    // főmenüpontok megjelenítése
             document.layers[layerId(i)].top = ypos
             ypos += document.layers[layerId(i)].clip.height
             if (i==fomenu) {                     // almenüpontok megjelenítése
                for (var j=0; j < menu[i].almenuk.length; j++) {
                    ypos += menu.almenukoz
                    document.layers[layerId(i, j)].top = ypos
                    ypos += document.layers[layerId(i, j)].clip.height
                    showLayer(layerId(i, j))
                }
             }
             ypos += menu.fomenukoz
         }

         menu.fomenu = fomenu                   // megadott menüpont az aktuális
         menu.almenu = almenu
         if (menu.fomenu<0) return
                                        // kiválasztott főmenüpont megjelölése
         ypos = document.layers[layerId(fomenu)].top +
                document.layers[layerId(fomenu)].clip.height
         document.layers[menu.fovalasztasid].top = ypos
         showLayer(menu.fovalasztasid)
         if (menu.almenu<0) return
                                        // kiválasztott almenüpont megjelölése
         ypos = document.layers[layerId(fomenu, almenu)].top +
                document.layers[layerId(fomenu, almenu)].clip.height
         document.layers[menu.alvalasztasid].top = ypos
         showLayer(menu.alvalasztasid)
         return
}
function kivalaszt(fomenu, almenu) {             // adott menüpont kiválasztása
         if (menu.fomenu == fomenu && almenu == menu.almenu) return false
         showMenu(fomenu, almenu)
         return false             // eseményfeldolgozás továbbfutásának tiltása
}
function elhagy() {// aktuális menüpont mutatók elrejtése menüpont elhagyásakor
```

```
         hideLayer(menu.fomutatoid)
         hideLayer(menu.almutatoid)
}
function ralep(fomenu, almenu) {                  // menüpont aktuálissá tétele
         elhagy()

         if (almenu < 0)      // menü feliratának megjelenítése a státuszsorban
            window.status=menu[fomenu].felirat
         else
            window.status=menu[fomenu].almenuk[almenu].felirat

         if (menu.fomenu==fomenu && (almenu<0 || almenu==menu.almenu))
            return true

         var ypos
         if (almenu<0) {                          // főmenüpont aktuálissá tétele
            ypos = document.layers[layerId(fomenu)].top
            document.layers[menu.fomutatoid].top = ypos
            showLayer(menu.fomutatoid)
         } else {                                 // almenüpont aktuálissá tétele
            ypos = document.layers[layerId(fomenu, almenu)].top
            document.layers[menu.almutatoid].top = ypos
            showLayer(menu.almutatoid)
         }

         return true           // böngésző státuszsora változásának jóváhagyása
}

document.writeln("<STYLE TYPE=\"text/css\">")
for (var i=0; i<menu.length; i++) {            // layerek stílusának generálása
    document.writeln("#"+layerId(i)+
                     " {position:absolute; visibility:show; left:0px;}")
    for (var j=0; j<menu[i].almenuk.length; j++)
        document.writeln("#"+layerId(i, j)+
                         " {position:absolute; visibility:hidden; left:0px;}")
}
document.writeln("#"+menu.fomutatoid+
                     " {position:absolute; visibility:hidden; left:0px;}")
document.writeln("#"+menu.fovalasztasid+
                     " {position:absolute; visibility:hidden; left:0px;}")
document.writeln("#"+menu.almutatoid+
                     " {position:absolute; visibility:hidden; left:0px;}")
document.writeln("#"+menu.alvalasztasid+
                     " {position:absolute; visibility:hidden; left:0px;}")
document.writeln("</STYLE>")
</SCRIPT>
</HEAD>
<BODY leftmargin=0 topmargin=0 onLoad="showMenu(-1, -1)" BGCOLOR="black">
<SCRIPT>
for (var i=0; i<menu.length; i++) {
    document.writeln("<DIV ID=\""+layerId(i)+"\">"+
                          "<A HREF=\"javascript:void(0)\" "+
                             "onMouseDown=\"return kivalaszt("+i+",-1)\" "+
                             "onMouseOver=\"return ralep("+i+",-1)\" "+
                             "onMouseOut=\"elhagy()\">"+
                             "<IMG ALT=\""+menu[i].felirat+"\" "+
                                  "SRC=\""+menu[i].kep+"\" "+
```

```
                                    "BORDER=0"+
                                  ">"+
                                "</A>"+
                            "</DIV>")
    for (var j=0; j<menu[i].almenuk.length; j++)
        document.writeln("<DIV ID=\""+layerId(i,j)+"\">"+
                          "<A HREF=\"javascript:void(0)\" "+
                             "onMouseDown=\"return kivalaszt("+i+","+j+")\" "+
                             "onMouseOver=\"return ralep("+i+","+j+")\" "+
                             "onMouseOut=\"elhagy()\">"+
                             "<IMG ALT=\""+menu[i].almenuk[j].felirat+"\" "+
                                  "SRC=\""+menu[i].almenuk[j].kep+"\" "+
                                  "BORDER=0"+
                             ">"+
                          "</A>"+
                        "</DIV>")
}
document.writeln("<DIV ID=\""+menu.fomutatoid+"\">"+
                      "<IMG ALT=\"=>\" "+
                           "SRC=\""+menu.fomutatokep+"\" "+
                           "BORDER=0"+
                      ">"+
                  "</DIV>")
document.writeln("<DIV ID=\""+menu.fovalasztasid+"\">"+
                      "<IMG ALT=\"==\" "+
                           "SRC=\""+menu.fovalasztaskep+"\" "+
                           "BORDER=0"+
                      ">"+
                  "</DIV>")
document.writeln("<DIV ID=\""+menu.almutatoid+"\">"+
                      "<IMG ALT=\">\" "+
                           "SRC=\""+menu.almutatokep+"\" "+
                           "BORDER=0"+
                      ">"+
                  "</DIV>")
document.writeln("<DIV ID=\""+menu.alvalasztasid+"\">"+
                      "<IMG ALT=\"-\" "+
                           "SRC=\""+menu.alvalasztaskep+"\" "+
                           "BORDER=0"+
                      ">"+
                  "</DIV>")
</SCRIPT>
</BODY>
</HTML>
```



G.12. ábra: Menü HTML rétegekből

Image : képeket reprezentáló objektum

A HTML oldalba beágyazott képeket (<IMG>) a document images tömbjén keresztül lehet elérni. Ugyanakkor programból is hozhatunk létre képet reprezentáló objektumot az Image konstruktorfüggvénnyel, melynek két lehetséges paramétere az újonnan létrehozott kép mérete (szélesség, magasság). A kép adatainak letöltése annak elérési útjának megadásakor, nem pedig az első megjelenítésekor (ahogy a Javában) kezdődik el, és szintén párhuzamosan, egy háttérben futó programszálból történik.

Az Image eseménykezelői

- onAbort
- onError
- onKeyDown
- onKeyPress
- onKeyUp
- onLoad

Az Image nem rendelkezik onClick, onMouseOut és onMouseOver eseménykezelőkkel. Erre megoldás lehet, ha a képet egy Area vagy Link objektumba ágyazzuk.

Az Image alapértelmezett mezői

- border : kép keretének vastagsága képpontokban megadva. Megfelel a BORDER attribútumnak.
- complete : logikai érték, megadja, hogy a kép betöltése befejeződött-e.
- height : a kép magassága. Megfelel a HEIGHT attribútumnak.
- hspace : vízszintesen a kép és a szöveg közt kihagyandó terület. Megfelel a HSPACE attribútumnak.
- lowsrc : a kép kisfelbontású változatának URL-je. Ezt megváltoztatva a megadott kép fog megjelenni. Megfelel a LOWSRC attribútumnak.
- name : a kép neve. Megfelel a NAME attribútumnak.
- src : a kép URL-je. Ezt megváltoztatva a megadott kép fog megjelenni az eddigi méretnek megfelelően átméreteződve. Megfelel az SRC attribútumnak.
- vspace : függőlegesen a kép és szöveg közt kihagyandó terület. Megfelel a VSPACE attribútumnak.
- width : a kép szélessége. Megfelel a WIDTH attribútumnak.

Az Image alapértelmezett metódusai

- handleEvent : a paraméterként kapott esemény feldolgozása.

Form : beviteli űrlapok

A HTML oldalon található beviteli űrlapokat (<FORM>) a document forms tömbjén vagy az űrlap nevével megegyező mezőn keresztül lehet elérni.

A Form eseménykezelői

- onReset
- onSubmit

A Form alapértelmezett mezői

Egy Form elemeit annak elements tömbjén vagy az elem nevével megegyező mezőn keresztül lehet elérni. Ügyeljünk arra, hogy ha több elemnek is azonos a neve, akkor azon névvel megegyező nevű mező tömb típusú lesz.

- action : a cél URL, ahova a rendszer elküldi az űrlap tartalmát. Megfelel az ACTION attribútumnak.
- elements : az űrlapon található beviteli elemeket reprezentáló objektumok tömbje. A tömb elemeire a beviteli elem nevével is hivatkozni lehet.
- encoding : az űrlap tartalmának elküldésekor használt MIME kódolás neve. Megfelel az ENCTYPE attribútumnak.
- method : az űrlap tartalmának elküldésmódja. Megfelel a METHOD attribútumnak. Értéke get vagy post lehet.
- name : az űrlap neve. Megfelel a NAME attribútumnak.
- target : azon ablak neve, ahol az űrlap elküldése után kapott válasz meg fog jelenni. Megfelel a TARGET attribútumnak.

A Form alapértelmezett metódusai

- handleEvent : a paraméterként kapott esemény feldolgozása.
- reset : visszaállítja az űrlap elemeinek alapértelmezett értékét.
- submit : elküldi az űrlap tartalmát megfelelően kódolva a beállított URL-re. az űrlap elemeinek nem üres értéke az URL kérdőjel utáni részébe kerül & jelekkel elválasztva.

Button : nyomógombok

Beviteli űrlapon egy nyomógombot (<INPUT TYPE = "button">) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a gomb nevére hivatkozva lehet elérni ezt az objektumot.

A Button eseménykezelői

- onBlur
- onClick
- onFocus
- onMouseDown
- onMouseUp

A Button alapértelmezett mezői

- form : a gombot tartalmazó űrlap.
- name : a gomb neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans button.
- value : a gomb felirata. Megfelel a VALUE attribútumnak.

A Button alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- click : adott elem megnyomását szimulálja, de nem hívja meg annak onClick eseménykezelőjét.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

Checkbox : kipipálható mezők

Beviteli űrlapon egy kipipálható mezőt (<INPUT TYPE = "checkbox">) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezen objektumot. Megjegyzendő, hogy ez az objektum csak magát a kiválasztható dobozt reprezentálja, amelyhez nem tartozik felirat.

A Checkbox eseménykezelői

- onBlur
- onClick
- onFocus

A Checkbox alapértelmezett mezői

- checked : a mező aktuális állapotát jelző logikai érték.
- defaultChecked : a mező alapértelmezett állapotát jelző logikai érték. Megfelel a CHECKED attribútumnak.
- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans checkbox.
- value : a mező kiválasztását jelző érték. Megfelel a VALUE attribútumnak. Ha nincs megadva, az "on" érték jelzi a mező kiválasztottságát.

A Checkbox alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- click : adott elem megnyomását szimulálja, de nem hívja meg annak onClick eseménykezelőjét.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

FileUpload : fájlparaméter megadása

Beviteli űrlapon egy fájl kiválasztását lehetővé tevő mezőt (<INPUT TYPE = "file">) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezen objektumot.

A FileUpload eseménykezelői

- onBlur
- onChange
- onFocus

A FileUpload alapértelmezett mezői

- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans file.
- value : a mező tartalma, azaz a megadott fájlnév.

A FileUpload alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- select : adott elem tartalmának kijelölése.

Hidden : nem látható szövegmezők

A beviteli űrlapon ugyan nem látható, de az űrlap tartalmának elküldésekor szintén elküldésre kerülő szöveges mezőt (<INPUT TYPE = "hidden">) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezen objektumot.

A Hidden eseménykezelői

Nincsen semmilyen eseménykezelője.

A Hidden alapértelmezett mezői

- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans hidden.
- value : a mező tartalma.

A Hidden alapértelmezett metódusai

Nincs más metódusa az Object-nél tárgyaltakon kívül.

Option : választólista elemek

Beviteli űrlapon listából választható elemeket (<OPTION>) reprezentáló objektum. A lista options tömbjén keresztül lehet elérni ezt az objektumot. Létrehozható még az Option konstruktorfüggvénnyel is, melynek formája a következő: new Option([*szöveg*[, *VALUE*[, *CHECKED*[*ki van-e választva*]]]]).

Az Option eseménykezelői

Nincsen semmilyen eseménykezelője.

Az Option alapértelmezett mezői

- defaultSelected : logikai mező, megadja, hogy a listaelem alapértelmezés szerint ki van-e választva. Megfelel a SELECTED attribútumnak.
- index : az listaelem 0-val kezdődő sorszáma a listán belül.
- length : a listaelemet tartalmazó lista elemeinek száma.
- selected : logikai mező, megadja, hogy a listaelem ki van-e választva.
- text : a listaelem szövege.
- value : az a szöveg, amit az űrlap elküldésekor küld el, ha a listaelem ki van választva. Ha nincs megadva, akkor a text értéket küldi el. Megfelel a VALUE attribútumnak.

Az Option alapértelmezett metódusai

Nincs más metódusa az Object-nél tárgyaltakon kívül.

Password : Titkos adatok megadása

Beviteli űrlapon titkos adatok megadását lehetővé tevő mezőt reprezentáló objektum (<INPUT TYPE = "password">). Amikor adatokat gépelünk be egy ilyen mezőbe, akkor nem az fog megjelenni, amit begépeltünk, hanem minden egyes karakter helyén egy csillag fog állni. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezt az objektumot.

A Password eseménykezelői

- onBlur
- onFocus

A Password alapértelmezett mezői

- defaultValue : a mező alapértelmezett tartalma.
- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans password.
- value : a mező tartalma, azaz a beírt adat.

A Password alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- select : adott elem tartalmának kijelölése.

Radio : Rádiógombok

Beviteli űrlapon rádiógombokat (<INPUT TYPE = "radio">) reprezentáló objektum. Azonos nevű rádiógombok közül legfeljebb mindig csak egy lehet kiválasztva. az űrlap elements tömbjén keresztül vagy a mező nevére, mint tömbre hivatkozva lehet elérni ezen objektumot. Megjegyzendő, hogy ez az objektum csak magát a gombot reprezentálja, amelyhez nem tartozik felirat.

A Radio eseménykezelői

- onBlur
- onClick
- onFocus

A Radio alapértelmezett mezői

- checked : a mező aktuális állapotát jelző logikai érték.
- defaultChecked : a mező alapértelmezett állapotát jelző logikai érték. Megfelel a CHECKED attribútumnak.
- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans radio.
- value : a mező kiválasztását jelző érték. Megfelel a VALUE attribútumnak. Ha nincs megadva, az "on" érték jelzi a mező kiválasztottságát.

A Radio alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- click : adott elem megnyomását szimulálja, de nem hívja meg annak onClick eseménykezelőjét.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

Reset : visszaállítási nyomógomb

Beviteli űrlapon a visszaállítási nyomógombot (<INPUT TYPE = "reset">) reprezentáló objektum. Ezen gombot megnyomva az űrlap minden beviteli elemének értéke visszaáll az alapértelmezett értékre. Az űrlap elements tömbjén keresztül vagy a gomb nevére hivatkozva lehet elérni ezen objektumot.

A Reset eseménykezelői

- onBlur
- onClick
- onFocus

A Reset alapértelmezett mezői

- form : a gombot tartalmazó űrlap.
- name : a gomb neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans reset.
- value : a gomb felirata. Megfelel a VALUE attribútumnak.

A Reset alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- click : adott elem megnyomását szimulálja, de nem hívja meg annak onClick eseménykezelőjét.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

Select : listák

Beviteli űrlapon listákat (<SELECT>) reprezentáló objektum. A lista elemeit Option objektumok reprezentálják. Ha a listából csak egy elemet lehet kiválasztani, akkor az kombóboxként, különben pedig listaként kerül megjelenítésre. Az űrlap elements tömbjén keresztül vagy a gomb nevére hivatkozva lehet elérni ezen objektumot.

A Select eseménykezelői

- onBlur
- onChange
- onFocus

A Select alapértelmezett mezői

- form : a listát tartalmazó űrlap.
- length : a lista elemeinek száma.
- name : a lista neve. Megfelel a NAME attribútumnak.
- options : a lista elemeit reprezentáló Option objektumok tömbje.
- selectedIndex : az első kiválasztott listaelem 0-val kezdődő indexe. Ha nincs kiválasztva egy elem sem, akkor értéke -1.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans select-one vagy a több elem kiválasztását is engedélyező MULTIPLE attribútum megadása esetén select-multiple.

A Select alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

Submit : elküldési nyomógomb

Beviteli űrlapon az elküldési nyomógombot (<INPUT TYPE = "submit">) reprezentáló objektum. Ezen gomb megnyomásával az űrlap tartalmát elküldjük a megadott URL-re, majd a válaszként kapott HTML oldalt megjeleníti a böngésző. Az űrlap elements tömbjén keresztül vagy a gomb nevére hivatkozva lehet elérni ezen objektumot.

A Submit eseménykezelői

- onBlur
- onClick
- onFocus

A Submit alapértelmezett mezői

- form : a gombot tartalmazó űrlap.
- name : a gomb neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans submit.
- value : a gomb felirata. Megfelel a VALUE attribútumnak.

A Submit alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- click : adott elem megnyomását szimulálja, de nem hívja meg annak onClick eseménykezelőjét.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.

Text : szövegmezők

Beviteli űrlapon szöveg megadását lehetővé tevő mezőt (<INPUT TYPE = "text">) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezen objektumot.

A Text eseménykezelői

- onBlur
- onChange
- onFocus
- onSelect

A Text alapértelmezett mezői

- defaultValue : a mező alapértelmezett tartalma. Megfelel a VALUE attribútumnak.
- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Megfelel a TYPE attribútumnak, azaz értéke konstans text.
- value : a mező tartalma.

A Text alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- select : adott elem tartalmának kijelölése.

Textarea : többsoros szövegmezők

Beviteli űrlapon többsornyi szöveg megadását lehetővé tevő mezőt (<TEXTAREA>) reprezentáló objektum. Az űrlap elements tömbjén keresztül vagy a mező nevére hivatkozva lehet elérni ezen objektumot.

A Textarea eseménykezelői

- onBlur
- onChange
- onFocus
- onKeyDown
- onKeyUp
- onKeyPress
- onSelect

A Textarea alapértelmezett mezői

- defaultValue : a mező alapértelmezett tartalma.
- form : a mezőt tartalmazó űrlap.
- name : a mező neve. Megfelel a NAME attribútumnak.
- type : Értéke konstans textarea.
- value : a mező tartalma.

A Textarea alapértelmezett metódusai

- blur : elveszi a fókuszt az adott elemről.
- focus : adott elem megkapja a fókuszt.
- handleEvent : a paraméterként kapott esemény feldolgozása.
- select : adott elem tartalmának kijelölése.

5. példa: Űrlapok használata

A JavaScript egyik legfontosabb felhasználási területe a felhasználói űrlapokon bevitt adatok ellenőrzése. Ekkor ugyanis az adatok helyességének ellenőrzése már a kliensoldalon elvégezhető, nem kell még ezzel is terhelni a szerveroldalt. Tehát a szerverprogramokat egyszerűbbé teszi, csökkenti a szerver válaszidejét és persze kevesebb hálózati forgalmat is jelent, ha a lehető legtöbb adatellenőrzést kliensoldalon elvégezzük a JavaScript segítségével.

Ezen ellenőrzést két helyen is elvégezhetjük: az adott beviteli elem valamely eseménykezelőjében és az űrlap onSubmit eseménykezelőjében, amely közvetlenül az űrlap adatainak elküldése előtt hívódik meg. Ha itt hamis logikai értéket adunk vissza, akkor azzal megakadályozzuk az elküldést. Érdemes itt inkább a globálisabb feltételeket ellenőrizni, míg az egy adott beviteli elemre lokális megkötéseket vizsgálhatjuk annak valamely eseménykezelőjében. Ilyenkor csak arra vigyázzunk, hogy egy elem ellenőrzése nehogy kiváltsa egy másik elem ellenőrzését, mert könnyen végtelen ciklusba kerülhetünk[8].

A következő példa egy egyszerű megrendelési űrlapot mutat be. Az űrlap adatainak elküldésekor kerül sor minden ellenőrzésre. Ellenőrizzük, hogy minden szükséges mező ki van-e töltve, illetve egyes mezők tartalma megfelel-e megadott kritériumoknak (például darabszám csak pozitív egész lehet stb.). Csak akkor kerülnek a formadatok elküldésre, ha minden adat megfelel az ellenőrzési kritériumoknak.

A HTML forrás

```
<HEAD>
<SCRIPT>
function isEmpty(elem) {                              // üres-e az adott mező
                      // combobox akkor még üres, ha a legelső elem az aktuális
   if (elem.type.indexOf("select") == 0) return elem.selectedIndex <= 0
   return /^\s*$/.test(elem.value)   // szövegmező üres, ha nincs benne karakter
}
function isRequired(elem) {               // kötelező kitölteni az adott mezőt?
   if (typeof(elem.required) == "undefined") return false
   return elem.required
}
function checkRequired(elem) {                        // lehet-e üresen a mező?
   if (!isRequired(elem) || !isEmpty(elem)) return true
```

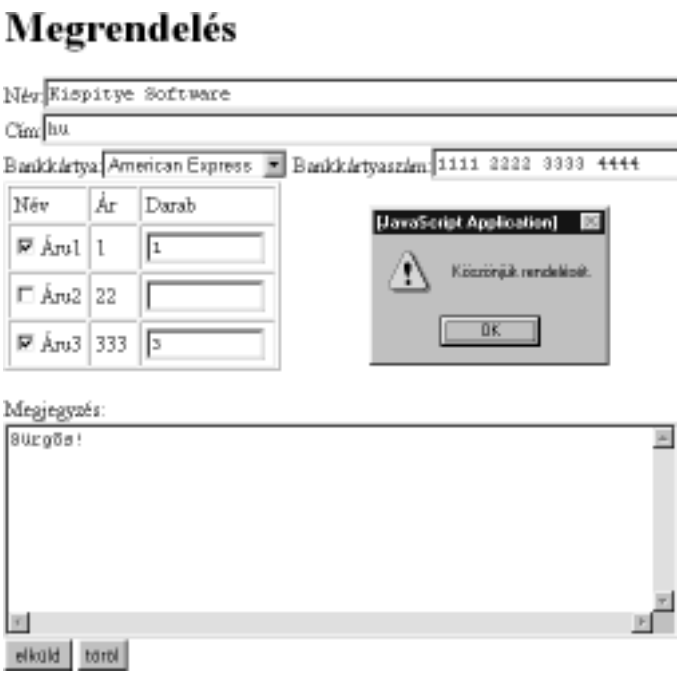
[8] Például ha két mező onBlur eseménykezelőjén visszaadjuk a fókuszt a komponensnek, akkor azok állandóan elveszik egymástól a fókuszt, lehet a böngészőt újraindítani.

```
   alert("A mező nem lehet üres!")
   elem.focus()
   return false
}
function checkAllRequired(form) {           // form minden elemét megvizsgálja,
   for (var i=0; i<form.elements.length; i++)              // hogy lehet-e üres
       if (!checkRequired(form.elements[i])) return false
   return true
}
function checkPosNum(elem) {           // adott elem pozitív számot tartalmaz-e
   if (!checkRequired(elem)) return false
   if (!isRequired(elem) && isEmpty(elem)) return true
   var val = parseInt(elem.value)
   if (val > 0 && val == elem.value) return true
   alert("A mező nem pozitív számot tartalmaz!")
   elem.select()
   elem.focus()
   return false
}
function checkRegExp(elem) {// adott elem megfelel-e egy reguláris kifejezésnek
   if (!checkRequired(elem)) return false
   if (!isRequired(elem) && isEmpty(elem)) return true
   if (elem.regexp.test(elem.value)) return true
   alert("A mező nem megfelelő értéket tartalmaz!")
   elem.select()
   elem.focus()
   return false
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Megrendelés</H1>
<FORM NAME="order" onsubmit="                       // elküldés előtt ellenőrzés
this.nr01db.required = this.nr01.checked              // darabszámok kellenek-e
this.nr02db.required = this.nr02.checked
this.nr03db.required = this.nr03.checked
if (!checkAllRequired(this)) return false    // minden szükséges mező kitöltve?
if (!checkRegExp(this.cardnum)) return false     // bankkártyaszám megfelelő-e?
if (!checkPosNum(this.nr01db)) return false// rendelt darabszámok megfelelők-e?
if (!checkPosNum(this.nr02db)) return false
if (!checkPosNum(this.nr03db)) return false              // van-e kijelölt áru?
if (!this.nr01.checked && !this.nr02.checked && !this.nr03.checked) {
   alert('Jelöljön ki legalább 1 árut!')
   return false
}
alert('Köszönjük rendelését.')
return false                                       //itt igaznak kellene állni...
">
Név:<INPUT TYPE="text" NAME="fullname" SIZE=60><BR>
Cím:<INPUT TYPE="text" NAME="address" SIZE=60><BR>
Bankkártya:<SELECT NAME="cardtype">
<OPTION>Kérem válasszon!
<OPTION>American Express
<OPTION>Visa
<OPTION>...
</SELECT>
Bankkártyaszám:<INPUT TYPE="text" NAME="cardnum" SIZE=25><BR>
```

```
<TABLE BORDER="1" CELLPADDING="3">
<TR><TD>Név<TD>Ár<TD>Darab
<TR><TD><INPUT TYPE="checkbox" NAME="nr01" VALUE="nr01" ONBLUR="
if (!this.checked) this.form.nr01db.value = ''
">Áru1<TD>1
<TD><INPUT TYPE="text" NAME="nr01db" SIZE=10 ONCHANGE="
this.form.nr01.checked = !isEmpty(this)
">
<TR><TD><INPUT TYPE="checkbox" NAME="nr02" VALUE="nr02" ONBLUR="
if (!this.checked) this.form.nr02db.value = ''
">Áru2<TD>22
<TD><INPUT TYPE="text" NAME="nr02db" SIZE=10 ONCHANGE="
this.form.nr02.checked = !isEmpty(this)
">
<TR><TD><INPUT TYPE="checkbox" NAME="nr03" VALUE="nr03" ONBLUR="
if (!this.checked) this.form.nr03db.value = ''
">Áru3<TD>333
<TD><INPUT TYPE="text" NAME="nr03db" SIZE=10 ONCHANGE="
this.form.nr03.checked = !isEmpty(this)
">
</TABLE>
<BR>Megjegyzés:<BR>
<TEXTAREA NAME="remark" COLS=60 ROWS=8></TEXTAREA>
<BR><INPUT TYPE="submit" VALUE="elküld">
<INPUT TYPE="reset" VALUE="töröl">
</FORM>
<SCRIPT>                              // mezők statikus jellemzőinek beállítása
document.order.fullname.required = true               // mindig szükséges mezők
document.order.address.required = true
document.order.cardtype.required = true
document.order.cardnum.required = true
document.order.cardnum.regexp = /^(\d+ )*\d+$/      // bankkártyaszám formátuma
document.order.fullname.focus()
</SCRIPT>
</BODY>
```

6. példa: cookie-k használata

Egy cookie segítségével információt lehet tárolni a böngészőprogramot futtató kliensszámítógépen, lehetővé téve ezáltal a kliensoldalon adott HTML lap megtekintéséhez szükséges adatok (pl. jelszó, felhasználói azonosító) megőrzését. Ezen információk a kliensgépen rendszerint egy cookies[.txt] nevű szövegfájlban tárolódnak. Egy cookie tehát egy szeletnyi információt tud eltárolni a böngészőprogramot futtató kliens gépen a következő formában:
név=*érték*[;expires=*lejárás dátuma*]
A név a cookie neve, amellyel majd a szkriptből hivatkozni lehet rá, az érték pedig az eltárolni kívánt információ. A név és az érték nem tartalmazhat szóközt, pontosvesszőt és semmilyen idézőjelet sem. Ha mégis használni szeretnénk ilyen karaktereket, akkor azokat kódolni kell az escape függvénnyel, ennek az eredményét az unescape függvény tudja visszaalakítani. A lejárat dátuma mindig GMT formátumban kell, hogy legyen. Erre használhatjuk a toGMTString függvényt, amely visszaadja adott dátum GMT formátumú reprezentációját. Ha nem adjuk meg a lejárat dátumát, akkor a cookie csak addig érvényes, amíg a böngészőprogram fut.



G.13. ábra: Megrendelési űrlap képe

A cookie-k használata a document.cookie objektumon keresztül történik. Az eltárolást és visszaolvasást legegyszerűbben saját funkciókkal valósíthatjuk meg, mint ahogy ezt a példaprogram is mutatja (setCookie, getCookie függvények).

A HTML forrás

```
<H1>cookie-k használata</H1>
<SCRIPT>
function setCookie(name, value, expire) {                    //cookie beállítása
         document.cookie=name+"="+escape(value)+
                         ((expire==null)?"":(";expires="+expire.toGMTString()))
}
function getCookie(Name) {                                   //cookie lekérdezése
         var search=Name+"="
         if (document.cookie.length>0) {                     //ha vannak cookie-k
            offset=document.cookie.indexOf(search)
            if (offset!=-1) {                               //ha a cookie létezik
               offset+=search.length          //index a cookie értékének elejére
               end=document.cookie.indexOf(";", offset)              //és végére
               if (end==-1) end=document.cookie.length
               return unescape(document.cookie.substring(offset, end))
            }
          }
}

var name=getCookie("JavaScriptTestCookie")
    if (name!=null) document.write("<H2>Hello, "+name+"!</H2>")
```

```
    else document.write("<H2>Üdvözlöm önt!</H2>")
</SCRIPT>
Ha regisztráltatja a nevét,
egy hétig ezen a néven fogja önt köszönteni ez a HTML oldal.<BR>
<FORM onSubmit="return false">
Az ön neve: <INPUT TYPE="text" NAME="username" SIZE=40>
<INPUT TYPE="submit" value="regisztráltat" onClick="
var ma=new Date();
var egyhetmulva=new Date();
    egyhetmulva.setTime(ma.getTime()+60*60*24*7)
    setCookie('JavaScriptTestCookie', this.form.username.value, egyhetmulva)
    history.go(0)                                          //oldal újrarajzolása
">
</FORM>
```

7. példa: interaktív JavaScript végrehajtás

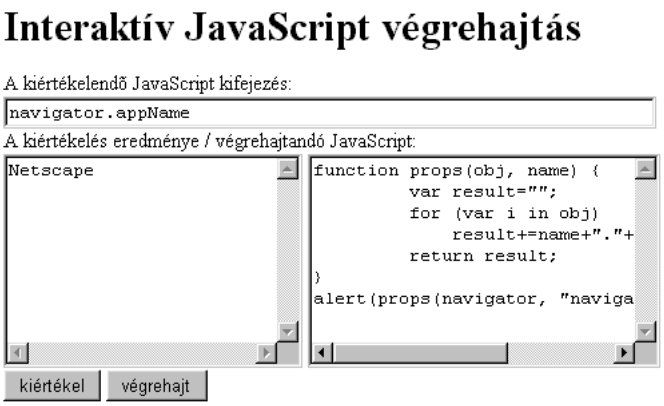
A JavaScript szkript tulajdonságának köszönhető, hogy futás közben, forrásszinten dinamikusan megadott kód is lefuttatható. Ennek két módját is bemutatja a következő példa.

A soreditorba beírt JavaScript kifejezés végrehajtását a Return billentyű, vagy a kiértékel gomb megnyomásával lehet kezdeményezni. A kiértékelés a kiértékel gomb onClick eseménykezelőjében, az eval függvény felhasználásával történik. Az űrlap tartalmának elküldését annak az onSubmit eseménykezelőjében visszaadott hamis érték akadályozza meg. A kiértékelés eredménye az első szövegmezőben válik láthatóvá. A második szövegmezőbe beírt JavaScript utasításokat pedig nem kiértékeli, hanem közvetlenül mint egy új függvényt hajtja végre a szkript a végrehajt gomb onClick eseménykezelőjében, a document objektumot, mint aktuális környezetet felhasználva. Ennek ugyanaz a hatása, mintha az itt megadott szkriptszöveg egy <SCRIPT> blokkban állna a dokumentum forrásszövegén belül.

A HTML forrás

```
<H1>Interaktív JavaScript végrehajtás</H1>
<FORM NAME="dinamikrun" onsubmit="
return false
">
A kiértékelendő JavaScript kifejezés:<BR>
<INPUT TYPE="text" NAME="expr" SIZE=60 VALUE="navigator.appName" onchange="
this.form.submitbutton.onclick()
">
<BR>A kiértékelés eredménye / végrehajtandó JavaScript:<BR>
<TEXTAREA NAME="result" COLS=25 ROWS=8></TEXTAREA>
<TEXTAREA NAME="parancs" COLS=30 ROWS=8>
function props(obj, name) {
         var result="";
         for (var i in obj)
             result+=name+"."+i+"="+obj[i]+"\n";
         return result;
}
alert(props(navigator, "navigator"))
</TEXTAREA>
<BR><INPUT TYPE="submit" NAME="submitbutton" VALUE="kiértékel" onclick="
this.form.result.value=eval(this.form.expr.value)
">
```

```
<INPUT TYPE="button" VALUE="végrehajt" onclick="
new Function(document.dinamikrun.parancs.value).call(document)
">
</FORM>
```



G.14. ábra: Interaktív végrehajtást szemléltető HTML képe

Böngésző felhasználói beállításai

A Navigatorban minden felhasználó elmentheti a saját beállításait. Ezen beállításokat Netscape/users alatt a felhasználóhoz tartozó könyvtárban található prefs.js fájl tartalmazza. Ezen fájl pedig egyszerűen JavaScriptet tartalmaz, a böngésző indításakor fut le és a user_pref funkciót meghívva állítja vissza az adott felhasználó elmentett beállításait. Így állítódik vissza többek közt a felhasználónként más és más cache és proxybeállítás, meglátogatott URL-ek listája stb....

JavaScript konzol

Az esetleges JavaScript hibaüzenetek alapértelmezés szerint egy külön ablakban, az úgynevezett *JavaScript konzolon* jelennek meg. Ezen ablak úgy jeleníthető meg, ha megtekintjük a javascript: URL-t. A konzolon még JavaScript kifejezések dinamikus végrehajtására is van lehetőségünk.

Hibakezelés

A JavaScriptnek szkript mivolta miatt igen hibatűrőnek kell lennie. Mivel a Java kivételkezelése erősen objektumorientált módszer, a JavaScriptben ez nincs megvalósítva. A nyelv hasonló módszert biztosít viszont az onError eseménykezelők segítségével.

Egy hiba fellépésekor az aktuális <SCRIPT> blokk vagy eseménykezelő megszakad, majd a window.onError eseménykezelő kapja meg a vezérlést. Az alapértelmezés szerinti hibakezelő kiírja a hiba szövegét, az aktuális fájl nevét és a hibás sor sorszámát a JavaScript konzolra. 10 hiba egymás utáni bekövetkezése esetén a "Túl sok JavaScript hiba" hibaüzenetet kapjuk, az ezután fellépő további hibák már nem kerülnek kijelzésre. Saját onError eseménykezelőt megadva ezen alapértelmezett funkció kikapcsolható, sőt null értékre állítva azt elmarad mindenféle hibajelzés.

Alapértelmezés szerint ha hiba lép fel és nincs még megnyitva a JavaScript konzol, akkor a böngésző státuszsorában meg fog jelenni egy erre figyelmeztető szöveg, de a konzol nem fog megjelenni automatikusan. Ha el szeretnénk érni, hogy a böngésző hiba esetén automatikusan megnyissa a JavaScript konzolt, akkor a felhasználói beállításokat tartalmazó fájlba írjuk be a következő sort:
`user_pref("javascript.console.open_on_error", true)`

Hibakeresés

Természetesen a hibakeresés legkényelmesebb módja valamilyen debugger program használata. Apróbb hibák esetén talán célravezetőbb lehet az Object watch és unwatch metódusainak használata, melyekkel egy objektum mezőjének változását lehet figyelemmel követni. Szintén egyszerű, de nagyon hasznos tud lenni, ha a szkript kritikus helyeire beépítünk egy-két alert hívást, és annak segítségével kiíratjuk a kérdéses változók tartalmát.

A hibakeresést megnehezítheti, hogy alapértelmezés szerint hiba fellépésekor nem kapunk semmilyen figyelmeztetést, csak a JavaScript konzolon jelenik meg a hibaüzenet. Ez viszont könnyen elkerülheti figyelmünket, és esetleg észre sem vesszük, hogy hiba történt. Ha azt akarjuk, hogy a böngésző minden hiba fellépésekor arra minket külön dialógusablakkal figyelmeztessen, akkor a felhasználói beállításokat tartalmazó fájlba írjuk be a következő sort: `user_pref("javascript.classic.error_alerts", true)`

Egyéb hasznos URL-ek

A már említett javascript: protokoll segítségével JavaScript kifejezéseket adhatunk meg URL-ként. A protokoll neve után megadott JavaScript kifejezés végrehajtódik, és annak eredménye szolgálja a hivatkozott URL tartalmát. Ha olyan URL-re van szükségünk, amelyik nem csinál semmit, használjuk a javascript:void(0) formát.

JavaScript segítségével dinamikusan generált HTML oldalak forrásszövegét megtekintve nem magát a JavaScript kódot, hanem az általa generált HTML forrást fogjuk látni. Ha magára a JavaScript kódra vagyunk kíváncsiak, ami az oldalt dinamikusan hozta létre, használjuk a view-source: protokollt: tegyük ki ezt a kérdéses URL elé, és máris látni fogunk minden JavaScript forrást.

Az aktuális HTML oldal felépítéséről, frame és űrlap hierarchiájáról a Page Info menüpont nyújt némi információt.

Az about: protokoll segítségével hasznos dolgokat tudhatunk meg a használt navigátorról. Önmagában megadva a böngésző névjegye hívható elő. about:cache alakban a böngésző által használt lemezcache-ről kapunk információt, about:plugins formájával pedig az installált plug-in segédprogramokról kapunk listát. Ha egy üres oldalra mutató URL-re van szükségünk, használhatjuk az about:blank formát.

JavaScript színkonstansok

Azon JavaScript objektumoknál, ahol valamilyen szín adható meg, RGB értékeket kell megadni a következő formában: #RRGGBB, ahol RR a piros, GG a zöld, BB pedig a kék színösszetevő hexadecimális értéke. Ezen direkt forma helyett előredefiniált színkonstansok is használhatók, melyeket a G.15.. táblázat tartalmaz. Ezen színkonstansok HTML színattribútumokban is használhatók.

aliceblue = F0F8FF	antiquewhite = FAEBD7
aqua = 00FFFF	aquamarine = 7FFFD4
azure = F0FFFF	beige = F5F5DC
bisque = FFE4C4	black = 000000
blanchedalmond = FFEBCD	blue = 0000FF
blueviolet = 8A2BE2	brown = A52A2A
burlywood = DEB887	cadetblue = 5F9EA0
chartreuse = 7FFF00	chocolate = D2691E
coral = FF7F50	cornflowerblue = 6495ED
cornsilk = FFF8DC	crimson = DC143C
cyan = 00FFFF	darkblue = 00008B
darkcyan = 008B8B	darkgoldenrod = B8860B
darkgray = A9A9A9	darkgreen = 006400
darkkhaki = BDB76B	darkmagenta = 8B008B
darkolivegreen = 556B2F	darkorange = FF8C00
darkorchid = 9932CC	darkred = 8B0000
darksalmon = E9967A	darkseagreen = 8FBC8F
darkslateblue = 483D8B	darkslategray = 2F4F4F
darkturquoise = 00CED1	darkviolet = 9400D3
deeppink = FF1493	deepskyblue = 00BFFF
dimgray = 696969	dodgerblue = 1E90FF
firebrick = B22222	floralwhite = FFFAF0
forestgreen = 228B22	fuchsia = FF00FF
gainsboro = DCDCDC	ghostwhite = F8F8FF
gold = FFD700	goldenrod = DAA520
gray = 808080	green = 008000
greenyellow = ADFF2F	honeydew = F0FFF0
hotpink = FF69B4	indianred = CD5C5C
indigo = 4B0082	ivory = FFFFF0
khaki = F0E68C	lavender = E6E6FA
lavenderblush = FFF0F5	lawngreen = 7CFC00
lemonchiffon = FFFACD	lightblue = ADD8E6
lightcoral = F08080	lightcyan = E0FFFF
lightgoldenrodyellow = FAFAD2	lightgreen = 90EE90
lightgrey = D3D3D3	lightpink = FFB6C1

G.15. ábra: JavaScript színkonstansok 1.

lightsalmon = FFA07A	lightseagreen = 20B2AA
lightskyblue = 87CEFA	lightslategray = 778899
lightsteelblue = B0C4DE	lightyellow = FFFFE0
lime = 00FF00	limegreen = 32CD32
linen = FAF0E6	magenta = FF00FF
maroon = 800000	mediumaquamarine = 66CDAA
mediumblue = 0000CD	mediumorchid = BA55D3
mediumpurple = 9370DB	mediumseagreen = 3CB371
mediumslateblue = 7B68EE	mediumspringgreen = 00FA9A
mediumturquoise = 48D1CC	mediumvioletred = C71585
midnightblue = 191970	mintcream = F5FFFA
mistyrose = FFE4E1	moccasin = FFE4B5
navajowhite = FFDEAD	navy = 000080
oldlace = FDF5E6	olive = 808000
olivedrab = 6B8E23	orange = FFA500
orangered = FF4500	orchid = DA70D6
palegoldenrod = EEE8AA	palegreen = 98FB98
paleturquoise = AFEEEE	palevioletred = DB7093
papayawhip = FFEFD5	peachpuff = FFDAB9
peru = CD853F	pink = FFC0CB
plum = DDA0DD	powderblue = B0E0E6
purple = 800080	red = FF0000
rosybrown = BC8F8F	royalblue = 4169E1
saddlebrown = 8B4513	salmon = FA8072
sandybrown = F4A460	seagreen = 2E8B57
seashell = FFF5EE	sienna = A0522D
silver = C0C0C0	skyblue = 87CEEB
slateblue = 6A5ACD	slategray = 708090
snow = FFFAFA	springgreen = 00FF7F
steelblue = 4682B4	tan = D2B48C
teal = 008080	thistle = D8BFD8
tomato = FF6347	turquoise = 40E0D0
violet = EE82EE	wheat = F5DEB3
white = FFFFFF	whitesmoke = F5F5F5
yellow = FFFF00	yellowgreen = 9ACD32

G.16. ábra: JavaScript színkonstansok 2.

G.4. LiveConnect: JavaScript - Java kommunikáció

Ha egy böngészőben mind a Java, mind a JavaScript engedélyezve van, akkor a két nyelv objektumai ki tudják használni a másik nyelv adta lehetőségeket is. Ezt a kommunikációs lehetőséget nevezik LiveConnect-nek.

Java elérése JavaScriptből

Javát futtatni képes böngésző használata esetén a JavaScript képes teljes mértékben kihasználni a Java osztályok szolgáltatásait. Ekkor a Java elérése JavaScriptből a következő három csoportba sorolható:

- Java objektumok, tömbök, metódusok, osztályok és csomagok használata. Ezek elérése a JavaPackage, JavaClass, JavaArray és JavaObject csomagoló objektumokon keresztül történik. A becsomagoló objektumok használata automatikusan történik, tehát forráskód szinten közvetlenül lehet használni bármilyen Java kifejezést, mindkét irányú típus-átalakításért a böngészőbe épített virtuális gép a felelős.
- Java appletek vezérlése.
- Java plug-in programok vezérlése.

A JavaPackage objektum

Egy JavaPackage objektum egy Java csomagot reprezentál. Az objektum mezőit az alcsomagokat reprezentáló JavaPackage objektumok, illetve az adott csomag osztályait reprezentáló JavaClass objektumok alkotják. A mezők neve megegyezik a reprezentált alcsomag, illetve osztály nevével.

A Packages objektum

Ezen objektumon keresztül érhető el a Java csomagokat reprezentáló JavaPackage hierarchia. Az objektum mezőit az alapvető Java csomagokat reprezentáló JavaPackage objektumok, illetve a név nélküli csomag osztályait reprezentáló JavaClass objektumok alkotják. A mezők neve megegyezik a reprezentált Java csomag, illetve osztály nevével.

A java objektum

Ez tulajdonképp egy JavaPackage objektum, amely a java Java csomagot reprezentálja. Megegyezik a Packages.java-val.

A netscape objektum

Ez tulajdonképp egy JavaPackage objektum, amely a netscape Java csomagot reprezentálja. Megegyezik a Packages.netscape-vel.

A sun objektum

Ez tulajdonképp egy JavaPackage objektum, amely a sun Java csomagot reprezentálja. Megegyezik a Packages.sun-nal.

A JavaClass objektum

Egy JavaClass objektum egy Java osztályt reprezentál. Az objektum mezőit és metódusait a reprezentált Java osztály statikus mezői és statikus metódusai alkotják. A mezők és metódusok neve megegyezik a reprezentált statikus Java mező, illetve metódus nevével. Megjegyzendő, hogy egy JavaClass objektum Java környezetben nem lesz automatikusan java.lang.Class objektummá átalakítva.

A JavaArray objektum

Egy JavaArray objektum egy Java tömböt reprezentál. Egyetlen mezője a length, amely a tömb hosszát adja meg. Egyetlen metódusa a toString, amely a tömb szöveges reprezentációját adja vissza.

A JavaObject objektum

Egy JavaObject objektum egy Java objektumot reprezentál. Az objektum mezőit és metódusait a reprezentált Java objektum publikus mezői és publikus metódusai alkotják. A mezők és metódusok neve megegyezik a reprezentált publikus Java mező, illetve metódus nevével.

Az Applet objektum

Egy Applet objektum egy a HTML oldalba beágyazott (<APPLET>) Java appletet reprezentál. Az objektum mezőit és metódusait a reprezentált applet publikus mezői és publikus metódusai alkotják. A mezők és metódusok neve megegyezik a reprezentált publikus Java mező, illetve metódus nevével. A HTML oldal appletjeit a document applets mezőjén keresztül lehet elérni.

A Plugin objektum

Egy Plugin objektum egy a HTML oldalba beágyazott objektumot (<EMBED>) megjelenítő plug-in segédprogramot reprezentál. Az objektum mezőit és metódusait a reprezentált objektum statikus mezői és metódusai alkotják. A mezők és metódusok neve megegyezik a reprezentált statikus mező, illetve metódus nevével. A HTML oldalba épített objektumokat a document embeds mezőjén keresztül lehet elérni.

JavaScript elérése Javából

JavaScript Javából történő elérése a DOM objektumainak Javából való elérését jelenti. Ezen elérés miként je a következő három csoportba sorolható:

- A DOM hierarchia és annak objektumai mezőinek, illetve függvényeinek elérése a netscape.javascript.JSObject becsomagoló osztály segítségével.
- JavaScript kivételek kezelése a netscape.javascript.JSException becsomagoló osztály segítségével.
- A böngésző plug-in programjainak vezérlése a netscape.plugin.Plugin becsomagoló osztály segítségével. Ugyanezen osztály segítségével írhatunk saját Java plug-in programokat.

Fordítás

netscape.javascript JavaScript használata esetén importálni kell a netscape.javascript csomagot, amit a fordítónak el kell tudnia érni, például a CLASSPATH környezeti változón keresztül. Ezen csomagot a navigátor esetén például a java_40.jar fájl tartalmazza.

A JavaScript elérésének biztosítása applet számára

Biztonsági okokból egy applet alapértelmezés szerint nem érheti el a DOM JavaScript objektumokat. Ennek engedélyezését a beágyazó HTML oldalon külön jelölni kell, az <APPLET> kulcsszóban a MAYSCRIPT paraméter megadásával. Nem engedélyezett JavaScript hozzáférés esetén mindig kivétel fog fellépni.

JavaScript objektumok és függvények Javában

Egy HTML oldal bármely JavaScript objektumát az engedélyezett applet egy JSObject típusú objektumnak látja. Érdemes minden ilyen JSObject objektum hozzáférést kivételkezelőn belül elvégezni, mivel így az esetlegesen fellépő JavaScript hibát jelző JSException kivétel egyből kezelhető.

A DOM hierarchia csúcsán lévő window JavaScript objektumot elérni a JSObject getWindow(Applet) metódusával lehet. Bármely JavaScriptbeli JSObject objektum mezőjének lekérdezése pedig az Object getMember(String) metódussal történik. Egy JSObject JavaScript objektum valamely metódusát a call(String, Object[]) vagy az eval(String) metódusokkal lehet végrehajtatni, ahol a sztring paraméter a végrehajtandó JavaScript kifejezést adja meg.

JavaScript-Java típuskonverziók

Mivel a Java és a JavaScript típusok nem ugyanazok, ezért a két nyelv egymás közti paraméterátadásakor típuskonverzióra van szükség. Ezen konverzió automatikusan hajtódik végre a következő szabályok szerint:

JavaScript számok átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- double : a számérték konvertálás nélkül kerül átadásra.
- java.lang.Double, java.lang.Object : a számértéket becsomagoló java.lang.Double objektum automatikusan jön létre.
- float : a számérték kisebb pontosságra konvertálás után kerül átadásra. Túl nagy vagy alacsony értékek pozitív, illetve negatív végtelenként kerülnek átadásra.
- byte, char, int, long, short : a számérték konvertálásakor a kerekítés a negatív végtelen felé történik. Túl nagy vagy alacsony érték futási hibát vált ki. NaN pedig 0-ként kerül átadásra.
- java.lang.String : a számérték szöveggé konvertálódik.
- boolean : 0 és NaN értékek hamis, egyéb értékek pedig igaz logikai értékként kerülnek átadásra.

JavaScript logikai értékek átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- boolean : a logikai érték konvertálás nélkül kerül átadásra.

- java.lang.Boolean, java.lang.Object : a logikai értéket becsomagoló java.lang.Boolean objektum automatikusan jön létre.
- java.lang.String : a logikai érték szöveggé konvertálódik. Igaz érték esetén „true", hamis értéke esetén pedig „false" az eredmény.
- byte, char, double, float, int, long, short : a logikai érték konvertálásakor az igaz érték 1-nek, míg a hamis érték 0-nak felel meg.

JavaScript sztringek átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- java.lang.String, java.lang.Object : a sztringet reprezentáló java.lang.String objektum automatikusan jön létre.
- byte, double, float, int, long, short : a sztring tartalma számmá konvertálódik.
- char : Mivel ilyen típus nincs a JavaScriptben, használjunk egy java.lang.Character objektumot reprezentáló JavaObject-et. Alapértelmezés szerint a sztring számmá konvertálódik, majd az így kijelölt kódú karakter lesz a konverzió eredménye.
- boolean : az üres sztring hamis, minden egyéb sztring igaz logikai értékként kerül átadásra.

JavaScript undefined átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- java.lang.String, java.lang.Object : automatikusan létrejön egy „undefined" szöveget tartalmazó java.lang.String objektum.
- boolean : hamis logikai érték kerül átadásra.
- double, float : NaN számérték kerül átadásra.
- byte, char, int, long, short : 0 számérték kerül átadásra.

JavaScript null átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- java.lang.Object : null referencia kerül átadásra.
- boolean : hamis logikai érték kerül átadásra.
- byte, char, double, float, int, long, short : 0 számérték kerül átadásra.

JavaScript JavaObject és JavaArray becsomagoló objektumok átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- Interfész vagy osztály, amely értékül kaphatja a becsomagolt objektumot : a becsomagolt Java objektum kerül átadásra.
- java.lang.String : a becsomagolt Java objektum toString metódusának eredménye kerül átadásra.
- byte, char, double, float, int, long, short : Ha a becsomagolt Java objektum rendelkezik a doubleValue metódussal, akkor ennek eredménye kerül átadásra. Ellenkező esetben hiba lép fel.
- boolean : ha a becsomagolt Java objektum null, akkor hamis, egyébként igaz logikai érték kerül átadásra.

JavaScript JavaClass becsomagoló objektumok átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- java.lang.Class : a becsomagolt Java osztály kerül átadásra.
- java.lang.Object, JSObject : az objektumot becsomagoló JSObject Java objektum kerül átadásra.
- java.lang.String : a becsomagolt Java osztály toString metódusának eredménye kerül átadásra.
- boolean : ha a becsomagolt Java osztály null, akkor hamis, egyébként igaz logikai érték kerül átadásra.

Egyéb JavaScript objektumok átadása Java metódusok paramétereként

A következő lista a Java paramétertípusok alapján mutatja be a konverziós szabályokat.

- java.lang.Object, JSObject : az objektumot becsomagoló JSObject Java objektum kerül átadásra.
- java.lang.String : az objektum toString metódusának eredménye kerül átadásra.
- byte, char, double, float, int, long, short : Az objektum a JavaScript konverziós szabályai alapján számmá konvertálódik, ennek eredménye kerül átadásra.
- boolean : ha az objektum null, akkor hamis, egyébként igaz logikai érték kerül átadásra.

Java-JavaScript típuskonverziók

- Java byte, char, short, int, long, float, double JavaScript számmá alakul.
- Java boolean JavaScript boolean-ná konvertálódik.
- Java JSObject objektum a becsomagolt JavaScript objektummá konvertálódik.
- Java tömbök a megfelelő JavaScript tömbbé (objektum+length mező) konvertálódnak.
- Minden más Java objektum a megfelelő beágyazó JavaObject, JavaClass JavaScript reprezentációvá alakul.

Megjegyzendő, hogy a Java Number, illetve String objektumok automatikusan csak egy becsomagoló JavaObject JavaScript objektummá alakulnak át. A további konverziót magunknak kell elvégeznünk.

8. példa: Applet vezérlése és HTML űrlapok kilistázása

A következő példa megmutatja, hogyan lehet Java appleteket JavaScripttel vezérelni, illetve hogyan lehet Java appletekből JavaScript objektumokat és metódusokat elérni. A példa egy appletből és az azt beágyazó HTML lapból áll. Maga az applet nem is rendelkezik eseménykezeléssel, hisz a vezérlése kívülről, JavaScript segítségével történik.

A soreditorban megadott szöveg (az ezt meghatározó text mező publikus) fog a ScriptTest appletben villogni. Az applet villogását a Villog checkbox-szal, villogása sebességét a lassan, közepesen, gyorsan rádiógombokkal lehet vezérelni, míg a színeket az Előtérszín, Háttérszín RGB értékekkel lehet megadni. A Dátum gomb megnyomásakor megjelenik a Java java.util.Date osztályának segítségével lekérdezett aktuális dátum. A Listáz gomb megnyomásakor pedig kilistázódnak a HTML oldalon definiált űrlapok és azok elemei a Java konzolra, külön figyelmeztető ablakokban pedig megjelennek ezen elemek mezőlistái is. A mezők listázását a showProps JavaScript függvény végzi, melynek meghívása viszont a Java appletből történik.

A JScriptTest applet forráskódja

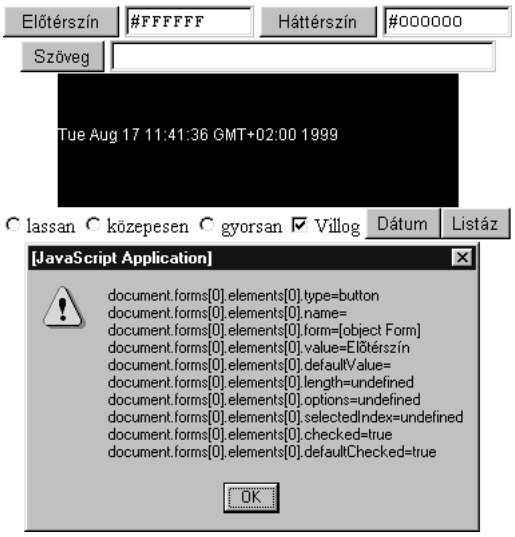
```
import netscape.javascript.*;                        //JavaScript eléréséhez kell
import java.applet.Applet;
import java.awt.*;

public class JScriptTest extends Applet implements Runnable {
  private Thread kicker=null;                                  //saját Thread
  public String text=null;
  public int ido;
  public void szinez(String szin, boolean eloter) {             //színbeállitás
    int val=0;
    try {
        if (szin.startsWith("0x")) {                        //hexában van megadva
           val=Integer.parseInt(szin.substring(2), 16);
        } else if (szin.startsWith("#")) {                  //hexában van megadva
              val=Integer.parseInt(szin.substring(1), 16);
        } else if (szin.startsWith("0") && szin.length()>1) {          //oktális
              val=Integer.parseInt(szin.substring(1), 8);
        } else val=Integer.parseInt(szin, 10);         //decimálisan van megadva
        if (eloter) setForeground(new Color(val));
        else setBackground(new Color(val));
        repaint();
    } catch (NumberFormatException e) {}
  }
  public void paint(Graphics g) {                                //szöveg kiírása
    if (text!=null) g.drawString(text, 0, 50);
  }
  public void start() {                                          //villogás indul
    if (kicker==null) {
       kicker=new Thread(this);
       kicker.start();
    }
  }
  public void stop() {                                          //villogás megáll
    if (kicker!=null) {
       kicker=null;
    }
  }
  public void run() {                                             //villogtatás
    while (Thread.currentThread()==kicker) {
          Color color=getForeground();
          setForeground(getBackground());
          setBackground(color);
          try {kicker.sleep(ido<500 ? 500 : ido);}
          catch (Exception e) {}
    }
  }
  public void listaz() {
    Applet applet;
    JSObject form;
    JSObject item;
    JSObject items;
    int itemcount;

    try {
         JSObject win=JSObject.getWindow(this);
```

```
        System.out.println("window="+win);                    //window objektum
        JSObject doc=(JSObject)win.getMember("document");   //document objektum
        System.out.println("document="+doc);
        JSObject forms=(JSObject)doc.getMember("forms");          //forms tömb
        System.out.println("forms="+forms);
        int imax=((Double)forms.getMember("length")).intValue();//form-ok száma
        System.out.println("forms.lenght="+imax);
        for (int i=0; i<imax; i++) {                          //form-okon végig
            form=(JSObject)forms.getSlot(i);
            System.out.println("forms["+i+"]: "+form);          //form objektum
            items=(JSObject)form.getMember("elements");         //elements tömb
            itemcount=((Double)items.getMember("length")).intValue();
            for (int j=0; j<itemcount; j++) {
                item=(JSObject)items.getSlot(j);               //elemeken végig
                System.out.println("    elements["+j+"]: "+item);
                Object[] params = {
                                item,
                                "document.forms["+i+"].elements["+j+"]"
                        };
                win.call("showProps", params);           //böngészőablakba infó
            }
        }
    } catch (Exception e) {System.out.println(e+"");}
  }
}
```

A HTML forrás

```
<HTML>
<HEAD>
<SCRIPT>
function showProps(obj, name) {
         var result=""
         for (var i in obj) {
             result+=name+"."+i+"="+obj[i]+"\n"
         }
         alert(result)
}
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<H1>LiveConnect: JavaScript - Java kommunikáció</H1>
<FORM NAME=ScriptTestForm>
<INPUT TYPE=button VALUE="Előtérszín"
   ONCLICK="document.ScriptTestApplet.szinez(ScriptTestForm.eloter.value,true)">
<INPUT TYPE=text NAME="eloter" SIZE=11 VALUE="#FFFFFF">
<INPUT TYPE=button VALUE="Háttérszín"
  ONCLICK="document.ScriptTestApplet.szinez(ScriptTestForm.hatter.value,false)">
<INPUT TYPE=text NAME="hatter" SIZE=11 VALUE="#000000"><BR>
<INPUT TYPE=button VALUE="Szöveg"
       ONCLICK="document.ScriptTestApplet.text=ScriptTestForm.text.value;
                document.ScriptTestApplet.repaint()">
<INPUT TYPE=text NAME="text" SIZE=35><BR>

<APPLET CODE=JScriptTest.class NAME=ScriptTestApplet
        WIDTH=300 HEIGHT=100 MAYSCRIPT>
```

```
</APPLET><BR>
<INPUT TYPE=radio NAME="sebes" VALUE="lassan" ONCLICK="
    document.ScriptTestApplet.ido=2000
">lassan
<INPUT TYPE=radio NAME="sebes" VALUE="közepesen" ONCLICK="
    document.ScriptTestApplet.ido=1000
">közepesen
<INPUT TYPE=radio NAME="sebes" VALUE="gyorsan" ONCLICK="
    document.ScriptTestApplet.ido=500
">gyorsan
<INPUT TYPE=checkbox CHECKED NAME=villog ONCLICK="
    if (this.checked) document.ScriptTestApplet.start();
    else document.applets['ScriptTestApplet'].stop();
">Villog
<INPUT TYPE=button VALUE="Dátum" ONCLICK="
    document.ScriptTestApplet.text=new java.util.Date();
    document.ScriptTestApplet.repaint()
"><INPUT TYPE=button VALUE="Listáz" ONCLICK="
    document.ScriptTestApplet.listaz()
">
</FORM>
</BODY>
</HTML>
```



G.17. ábra: A JScriptTest applet és a vezérlő HTML képe

G.5. Biztonság

A JavaScript alapértelmezés szerint az úgynevezett „*azonos származási hely*” politikának megfelelően engedélyezi a DOM objektumokhoz a hozzáférést. Ez azt jelenti, hogy csakis az azonos helyről származó dokumentumokban található JavaScript kód láthatja egy másik, de ugyanazon helyről származó dokumentumbeli JavaScript objektumokat. A

származási hely alatt pedig a dokumentum URL-jének prefixét értjük egészen a keresési útvonalig, amibe tehát beletartozik a protokoll, a szerver neve és a port száma. Ezen származási hely megváltoztatható a document domain mezőjén keresztül, de csakis az aktuális érték prefixe adható meg, ezzel mintegy bővítve azt a kört, ahonnan az aktuális szkript objektumai elérhetőek.

A származási hely azonosságának ellenőrzése a következő mezők elérésekor hajtódik végre:

- document : íráskor minden mezője esetén, míg olvasáskor csak a következő mezők elérésekor: anchors, applets, cookie, domain, embeds, forms, lastModified, length, links, referrer, title, URL, minden megnevezett űrlap és minden elérhető Java osztály.
- Form : elements
- Image : lowsrc, src
- Layer : src
- Location : minden mezője, kivéve x és y
- window : find

Ha egy dokumentum URL-je nem file: protokollú, akkor a <SCRIPT> SRC attribútumával megadott forrásszöveg URL nem lehet file: protokollú. Ennek ellenőrzését a felhasználói beállításokat tartalmazó fájlba beírt következő sorral lehet kikapcsolni:
`user_pref("javascript.allow.file_src_from_non_file", true);`

Egy HTML oldal rétegei esetén akkor kerül sor a származási hely azonosságának ellenőrzésére, ha valamely réteg megpróbálja elérni egy másik réteg vagy a szülő dokumentum DOM-ját.

Appletek esetén ha az applet JavaScriptet használ (MAYSCRIPT), akkor annak származási helyeként az appletet tartalmazó dokumentum URL-jét veszi figyelembe a rendszer.

Digitálisan aláírt szkript

A fentebb tárgyalt „*azonos származási hely*" politika által meghatározott láthatóságot felül lehet bírálni, ha a kérdéses szkript digitális aláírással van ellátva. Egy digitálisan aláírt szkriptnek akkor is meglehet minden joga, ha annak származási helye nem azonos az aktuális dokumentuméval. A digitális aláírást egy külön jar fájl tartalmazza, vagy ha a HTML oldal szervere SSL-en keresztül küldi a HTML oldalt, akkor az abban található összes szkript automatikusan digitálisan aláírttá válik, ahol az aláíráshoz használt kulcs az SSL-szerver publikus kulcsa lesz. Egy szkript digitális aláírását tehát az jelzi, ha megadja az ARCHIVE és ID attribútumokat. Az ARCHIVE attribútum a digitális aláírást tartalmazó jar fájlt adja meg. Elég csak egyszer megadni, az utána következő ID hivatkozások mindig a legutoljára specifikált ARCHIVE-ot használják. Az ID attribútum a jar fájlon belüli egyedi azonosítót adja meg.

A felhasználó azt is beállíthatja, hogy egy adott helyről származó minden szkript automatikusan digitálisan aláírtként viselkedjen. Ezt a felhasználói beállításokat tartalmazó fájlba beírt következő sorral lehet elérni:
`user_pref("signed.applets.codebase_principal_support", true);`

Mivel egy oldal több szkriptet is tartalmaz, azok pedig különböző digitális aláírásokkal is rendelkezhetnek, ezért az egész oldalt jellemző digitális aláírók halmaza az oldalon található összes szkript digitális aláírói halmazának a metszete lesz. Ez azt jelenti, ha ugyanazon oldalon van egy digitálisan aláírt és egy aláírás nélküli szkript is, akkor mindkét szkript úgy lesz végrehajtva, mintha egyik sem lenne digitálisan aláírva.

Bővített jogosultságok

Ha az alapértelmezett azonos származási hely politika nem teszi lehetővé a szkript számára valamely DOM objektum/mező elérését, akkor a szkriptnek bővített jogosultsághoz kell folyamodnia, ha mégis használni akarja a letiltott mezőket. Egy digitálisan aláírt szkript sem kapja meg automatikusan a bővített jogosultságokat, annak is először kérvényeznie kell azt. A kérés eldöntésekor játszik szerepet, hogy van-e digitális aláírása a szkriptnek. Ha igen, és annak alapján engedélyezhető az elérés, akkor a szkript minden további nélkül megkapja a kért jogosultságot. Ellenkező esetben csak a felhasználó jóváhagyása után lehetséges a kért jogosultság megszerezése.

Jogosultság kérése a következő Java sorral történik:
`netscape.security.PrivilegeManager.enablePrivilege(` *jogosultságnév*`);`
Ennek hatására az ezt futtató szkript megkapja a kért jogosultságot vagy a kérés elutasításra kerül. Mindez a szkript digitális aláírása alapján történik, illetve amellett még (és annak hiányában) a felhasználó döntheti el a jogosultság megadását a megjelenő dialógusdoboz segítségével. A megjelenő dialógusdoboz a kért jogosultság veszélyességét, rövid és hosszú leírását tartalmazza, és lehetőséget ad a kérés engedélyezésére, illetve megtagadására a megfelelő nyomógombok segítségével. A megkapott jogok csakis azon függvény törzsén belül érvényesek, ahonnan a jogosultság kérése is történt. Ha a jogosultság ezen élettartamát is csökkenteni akarjuk, használjuk a PrivilegeManager megfelelő metódusait (lásd referencia). A függvény befejeződése után a szkript nem fog tovább rendelkezni a megszerzett jogokkal. A jogosultságra a böngésző csak egyszer kérdez rá, és mindaddig emlékszik a válaszunkra, amíg maga a böngésző fut. Ha a megjelenő dialógusban azt is megadjuk, hogy permanensen emlékezzen a válaszunkra, akkor az adott URL-re vonatkozó jogosultság eldöntésekor soha többe nem fog dialógusablak előjönni, még akkor sem, ha újraindítjuk a böngészőt.

A jogosultság neve egy egyszerű szöveges konstans, vagy az úgynevezett *paraméteres jogosultságok* esetén a nevet megadó szövegkonstansból és egy elválasztó üres karakterrel közvetlenül utána írt paraméterértékből tevődik össze.

A jogosultságokat három rizikócsoportba lehet sorolni:

- Magas: ha megadjuk a jogosultságot, akkor a szkript olyan joghoz juthat, amely erősen veszélyeztetheti a felhasználó biztonságát.
- Közepes: ha megadjuk a jogosultságot, akkor a szkript olyan joghoz juthat, amely veszélyeztetheti a felhasználó biztonságát.
- Alacsony: ha megadjuk a jogosultságot, akkor a szkript olyan joghoz juthat, amely kismértékben veszélyeztetheti a felhasználó titkos adatait.

A jogosultságokat összetettségük szerint is csoportosítani lehet. Összetett jogosultság alatt olyan jogosultságot értünk, amelynek megadása, illetve megtagadása egyszerre több másik egyszerűbb jog megadását, illetve megtagadását vonja maga után. Ennek ellenére az összetett jogosultság mindig több, mint összetevőinek az összessége. Ezen összetett jogosultságokat *makró*, míg az azokat felépítő jogokat *primitív* jogosultságoknak nevezzük.

A G.18.. táblázat összefoglalja a kérhető jogosultságokat. Az első oszlopban a jogosultság nevét tüntettük fel, összetett jogosultság esetén pedig még az azt felépítő jogokat is felsoroltuk.

9. példa: history elérése

A jogosultságok tesztelését a history mezőn fogjuk végezni, mivel ezen mező eléréséhez UniversalBrowserRead jogosultság szükséges, és ráadásul rendelkezik még azzal a kellemes

Jogosultság neve	Leírás	Rizikó-csoport	Mikor szükséges
AdministratorRegistryAccess StandardRegistryAccess	A telepített szoftverek információit tartalmazó registry adatbázis teljes elérését engedélyezi.	Magas	Rendszerint csakis rendszeradminisztrációs célokra használandó.
StandardRegistryAccess PrivateRegistryAccess	A telepített szoftverek információit tartalmazó registry adatbázis közös részének elérését engedélyezi.	Közepes	Rendszerint csakis együttműködő szoftverek esetén használandó.
PrivateRegistryAccess	Csak az aktuális szoftver információit tartalmazó registry adatbázisrész elérését engedélyezi.	Alacsony	Rendszerint csakis szoftverek telepítése esetén használandó.
Account Setup UniversalBrowserRead UniversalBrowserWrite UniversalPreferencesRead UniversalPreferencesWrite UniversalTopLevelWindow	A böngésző telepítését és konfigurálását engedélyezi.	Magas	Rendszerint csakis a böngésző programból történő konfigurálása esetén használandó.
CanvasAccess UniversalBrowserWrite	A böngésző ablak grafikus területének teljes elérését engedélyezi.	Magas	Rendszerint saját grafikai megjelenítés esetén használandó.
DatabaseAccess UniversalFileAccess	A fájlrendszer teljes elérését engedélyezi.	Magas	Rendszerint adatbázis-kezelők esetén használandó.
PresentationAccess UniversalFileAccess	A fájlrendszer teljes elérését engedélyezi.	Magas	Rendszerint bemutatóprogramok esetén használandó.
SpreadsheetAccess UniversalFileAccess	A fájlrendszer teljes elérését engedélyezi.	Magas	Rendszerint táblázatkezelő programok esetén használandó.
WordProcessorAccess UniversalFileAccess	A fájlrendszer teljes elérését engedélyezi.	Magas	Rendszerint szövegszerkesztő programok esetén használandó.
Debugger UniversalExecAccess UniversalPropertyRead UniversalPropertyWrite UniversalFileRead UniversalListen UniversalAccept UniversalConnect	Program nyomkövetését engedélyezi.	Magas	Rendszerint nyomkövető programok esetén használandó.

G.18. ábra: JavaScript jogosultságok 1

Jogosultság neve	Leírás	Rizikó-csoport	Mikor szükséges
GamesAccess PrivateRegistryAccess	A regisztrációs adatbázis korlátozott elérését engedélyezi.	Alacsony	Rendszerint játékprogramok esetén használandó például az elért pontszámok eltárolásához.
IIOPRuntime UniversalListen UniversalAccept UniversalConnect	IIOP megvalósítását engedélyezi.	Magas	Rendszerint osztott objektumokkal dolgozó programok esetén használandó.
Netcaster SiteArchiveTarget UniversalBrowserAccess UniversalConnect UniversalConnectWithRedirect UniversalFileAccess UniversalPreferencesRead UniversalPreferencesWrite UniversalThreadAccess UniversalThreadGroupAccess	Hálózati csatornaprogramok megvalósítását engedélyezi.	Magas	Rendszerint hálózati csatornákkal – lehetőleg kapcsolatmentes állapotban – dolgozó programok esetén használandó.
TerminalEmulator UniversalLinkAccess UniversalPropertyRead UniversalConnect UniversalListen UniversalAccept	Terminál emulátorok megvalósítását engedélyezi.	Magas	Rendszerint terminál emulátorok és más kommunikációs programok esetén használandó.
Capabilities UniversalThreadAccess UniversalThreadGroupAccess UniversalLinkAccess UniversalPropertyRead UniversalPropertyWrite UniversalConnect UniversalListen UniversalAccept UniversalTopLevelWindow UniversalPackageAccess UniversalPackageDefinition	Plug-in programok megvalósítását engedélyezi.	Magas	Rendszerint plug-in programok esetén használandó.
UniversalFileAccess UniversalLinkAccess UniversalPropertyRead UniversalFileRead UniversalFileWrite UniversalFileDelete	Fájlok teljes elérését engedélyezi.	Magas	Rendszerint fájlokkal dolgozó programok esetén használandó.

G.19. ábra: JavaScript jogosultságok 2

Jogosultság neve	Leírás	Rizikó-csoport	Mikor szükséges
SiteArchiveTarget	Web-archívum fájl elérését engedélyezi.	Magas	Csakis hálózati csatornákat web-archívumokba foglaló programok esetén használandó. Rendszerint nem szabad ezt a primitív jogot megadni, mivel ezt általában más makrókkal szokták kérvényezni.
UniversalAccept	Hálózati kapcsolat felvételét engedélyezi.	Magas	Más hálózati gépekről kezdeményezett kapcsolatok felépítéséhez szükséges. java.net.ServerSocket accept()
UniversalAwtEventQueueAccess	A fő eseménysor elérését engedélyezi.	Magas	A böngészőben történő összes esemény figyeléséhez szükséges.
UniversalBrowserAccess UniversalBrowserRead UniversalBrowserWrite	A böngésző minden adatának olvasását és írását engedélyezi. Az aktuális szkript minden dokumentum esetében teljesíteni fogja az azonos származási hely kritériumot.	Magas	Ha egyszerre szükséges UniversalBrowserRead és UniversalBrowserWrite.
UniversalBrowserRead	A böngésző minden adatának olvasását engedélyezi. Az aktuális szkript minden dokumentum esetében teljesíteni fogja az azonos származási hely kritériumot.	Közepes	about: URL használata esetén, kivéve about:blank DragDrop esemény data mezőjének olvasásakor. History bármely mezőjének olvasásakor.

G.20. ábra: JavaScript jogosultságok 3

Jogosultság neve	Leírás	Rizikócsoport	Mikor szükséges
UniversalBrowserWrite	A böngésző minden adatának írását engedélyezi. Az aktuális szkript minden dokumentum esetében teljesíteni fogja az azonos származási hely kritériumot.	Magas	event objektum bármely mezőjének írásakor. window objektum menubar.visible, toolbar.visible, locationbar.visible, personalbar.visible, scrollbars.visible és statusbar.visible mezőinek írásakor. window objektum enableExternalCapture metódusának használatakor. window objektum close metódusának használatakor. window objektum moveBy, illetve moveTo metódusának használatakor, ha az ablak lekerül a képernyőről. window objektum open metódusának használatakor, ha a létrehozott ablak lekerül a képernyőről, vagy az kisebb 100∗100-nál vagy nagyobb a képernyő méreténél, vagy ha nincs neki fejléce, vagy ha megadjuk az alwaysRaised, alwaysLowered vagy z-lock attribútumokat. window objektum resizeBy, illetve resizeTo metódusának használatakor, ha az ablak kisebb lenne 100 ∗ 100-nál vagy nagyobb a képernyő méreténél. window objektum innerWidth vagy innerHeight mezőinek állításakor, ha az ablak kisebb lenne 100 ∗ 100-nál vagy nagyobb a képernyő méreténél.

G.21. ábra: JavaScript jogosultságok 4

Jogosultság neve	Leírás	Rizikócsoport	Mikor szükséges
UniversalConnect	Bármely hálózati cím elérését engedélyezi.	Magas	Más hálózati gépekkel való kapcsolat felvételéhez szükséges. java.net.DatagramSocket getLocalAddress(), receive(DatagramPacket), send(DatagramPacket) java.net.MulticastSocket send(DatagramPacket, byte) java.net.Socket Socket(String, int), Socket(InetAddress, int), Socket(InetAddress, int, boolean), Socket(String, int, InetAddress, int), Socket(InetAddress, int, InetAddress, int), Socket(String, int, boolean) netscape.net.URLConnection connect() sun.awt.macos.MToolkit getImageFromHash(Toolkit, URL) sun.awt.motif.MToolkit getImageFromHash(Toolkit, URL) sun.awt.win32.MToolkit getImageFromHash(Toolkit, URL) sun.awt.windows.MToolkit getImageFromHash(Toolkit, URL) sun.net.www.http.HttpClient openServer(String, int)
UniversalConnectWithRedirect	Bármely hálózati cím elérését engedélyezi.	Magas	Más hálózati gépekkel való kapcsolat felvételéhez szükséges. A megcímzett gép a kérést átirányíthatja.
UniversalDialogModality	Modális dialógus megjelenítését engedélyezi.	Magas	Modális dialógusdoboz megjelenése alatt a böngészőablak le lesz tiltva, ezért ilyen ablakok helytelen használata magát a böngészőprogramot is megzavarhatja.

G.22. ábra: JavaScript jogosultságok 5

Jogosultság neve	Leírás	Rizikócsoport	Mikor szükséges
UniversalExecAccess	Processz futtatását engedélyezi.	Magas	java.lang.Runtime exec(String, String[]), exec(String[], String[]) java.lang.System setErr(PrintStream), setIn(InputStream), setOut(PrintStream)
UniversalExitAccess	A böngésző elhagyását engedélyezi.	Magas	A böngésző virtuális gépét leállítja, majd bezárja a böngészőprogramot. java.lang.Runtime exit(int)
UniversalFdRead	Adatok hálózaton keresztül történő olvasását engedélyezi.	Magas	Hálózati adatok fájlleíró segítségével történő olvasásához szükséges.
UniversalFdWrite	Adatok hálózatra történő írását engedélyezi.	Magas	Hálózati adatok fájlleíró segítségével történő írásához szükséges.
UniversalFileDelete	Lokális fájlok törlését engedélyezi.	Magas	Bármely lokálisan elérhető fájl törléséhez szükséges. java.io.File delete()
UniversalFileRead	A szkript olvashat minden lokális fájlt.	Magas	FileUpload használatakor. java.io.File canRead(), exists(), getCanonicalPath(), isDirectory(), isFile(), isLink(), lastAccessed(), lastStatusChange(), length(), list() java.io.FileInputStream FileInputStream(String), FileInputStream(FileDescriptor) java.io.RandomAccessFile RandomAccessFile(String, String), RandomAccessFile(File, String) sun.awt.macos.MToolkit getImageFromHash(Toolkit, URL) sun.awt.motif.MToolkit getImageFromHash(Toolkit, URL) sun.awt.win32.MToolkit getImageFromHash(Toolkit, URL) sun.awt.windows.MToolkit getImageFromHash(Toolkit, URL)

G.23. ábra: JavaScript jogosultságok 6

Jogosultság neve	Leírás	Rizikócsoport	Mikor szükséges
FileRead	A szkript egy adott lokális fájlt olvashat.	Magas	Paraméteres jogosultság, a paraméter a kérdéses fájl neve.
UniversalFileWrite	A szkript írhat minden lokális fájlt.	Magas	Bármely lokálisan elérhető fájl módosításához szükséges. java.io.File canWrite(), mkdir(), renameTo(File) java.io.FileOutputStream FileOutputStream(String), FileOutputStream(String, boolean), FileOutputStream(FileDescriptor) java.io.RandomAccessFile RandomAccessFile(String, String), RandomAccessFile(File, String)
FileWrite	A szkript egy adott lokális fájlt módosíthat.	Magas	Paraméteres jogosultság, a paraméter a kérdéses fájl neve.
UniversalLinkAccess	Dinamikus programkönyvtárak használatát engedélyezi.	Magas	Bármely lokálisan elérhető bináris programkönyvtár felhasználásához szükséges. java.lang.Runtime load(String), loadLibrary(String) java.lang.System load(String), loadLibrary(String)
UniversalListen	Hálózati kapcsolati kérelmek figyelését engedélyezi.	Magas	Bármely hálózati gépről jövő kapcsolat felépítéséhez szükséges. java.net.DatagramSocket DatagramSocket(), DatagramSocket(int, InetAddress) java.net.MulticastSocket MulticastSocket(), MulticastSocket(int) java.net.ServerSocket ServerSocket(int, int, InetAddress)
UniversalMulticast	Multicast használatát engedélyezi.	Magas	Egyszerre több géppel történő multicast hálózati kommunikációhoz szükséges. java.net.DatagramSocket send(DatagramPacket) java.net.MulticastSocket joinGroup(InetAddress), leaveGroup(InetAddress), send(DatagramPacket, byte)

G.24. ábra: JavaScript jogosultságok 7

Jogosultság neve	Leírás	Rizikó-csoport	Mikor szükséges
UniversalPreferencesRead	A szkript olvashat minden felhasználói beállítást.	Közepes	Felhasználói beállítás lekérdezésekor a navigator preference metódusával.
UniversalPreferencesWrite	A szkript módosíthat minden felhasználói beállítást.	Magas	Felhasználói beállítás módosításakor a navigator preference metódusával.
UniversalPrintJobAccess	Nyomtatás engedélyezése.	Alacsony	A böngészőből történő nyomtatáshoz szükséges.
UniversalPropertyRead	Rendszerjellemzők olvasásának engedélyezése.	Közepes	Például az aktuális felhasználói név, illetve aktuális könyvtár nevének kiolvasáshoz szükséges. java.lang.System getProperty(String), getProperty(String, String)
UniversalPropertyWrite	Rendszerjellemzők írásának engedélyezése.	Magas	Például az aktuális biztonsági beállítások módosításához szükséges. java.lang.System getProperties(), setProperties(Properties)
UniversalSendMail	A szkript küldhet levelet a felhasználó nevében.	Közepes	űrlap tartalmának elküldésekor, ha a cél URL mailto: vagy news: protokollú.
UniversalSetFactory	Hálózati kapcsolatokhoz szükséges protokollkezelők definiálását engedélyezi.	Magas	java.net.ServerSocket setSocketFactory(SocketImplFactory) java.net.Socket setSocketFactory(SocketImplFactory)
UniversalThreadAccess	Idegen programszálak elérésének engedélyezése.	Magas	java.lang.Thread checkAccess(), interrupt(), resume(), setPriority(int), setName(String), setDaemon(boolean), stop(Throwable), suspend()

G.25. ábra: JavaScript jogosultságok 8

Jogosultság neve	Leírás	Rizikó-csoport	Mikor szükséges
UniversalThreadGroupAccess	Idegen programszál-csoportok elérésének engedélyezése.	Magas	java.lang.ThreadGroup checkAccess(int), destroy(), resume(), setDaemon(boolean), setMaxPriority(int), stop(), suspend(), ThreadGroup(ThreadGroup, String)
UniversalSystemClipboardAccess	A rendszer vágólapjának elérését engedélyezése.	Magas	A rendszer vágólapjának írásához, illetve olvasásához szükséges.
UniversalTopLevelWindow	Ablakok nyitásának engedélyezése.	Magas	A megnyitott ablakok nem lesznek figyelmeztető üzenettel ellátva. java.awt.Window Window(Frame)
LimitedInstall	Java programok telepítésének engedélyezése.	Magas	A bájtkód fájlok telepítése a Java Download könyvtárba történik.
FullInstall	Programok telepítésének / frissítésének engedélyezése.	Magas	A telepítés csakis egy jóváhagyó dialógusdoboz megjelenése után indul el.
SilentInstall	Programok jóváhagyás nélkül történő telepítésének/frissítésének engedélyezése.	Magas	A telepítés ezen jog megadása után további felhasználói beavatkozás nélkül is lefuthat.

G.26. ábra: JavaScript jogosultságok 9

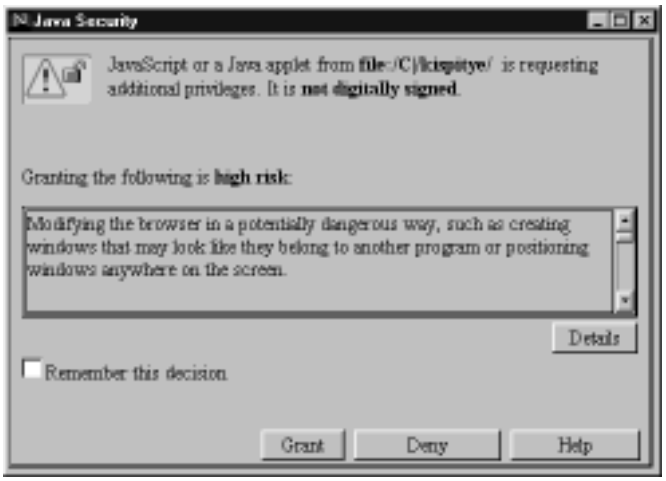
tulajdonsággal is, hogy szöveggé konvertálva HTML forrásszövegként adja vissza az eddig meglátogatott URL-ek listáját.

A soreditorba beírt JavaScript jogosultsággal felvértezve megpróbálja a program kilistázni a history tartalmát. Ellenőrzésképp a jogosultság megszerzése előtt, illetve annak lejárta után is elvégezzük a listázást. Ezen három lista egy új, teljesen üres ablakban jelenik meg.

Hiba esetén saját magunk jelezzük ki a hiba okát egy onError eseménykezelőt definiálva, és becsukjuk a már esetleg megjelenített listaablakot, mert nincs biztosítva, hogy teljesen végére ért a kilistázás.

A HTML forrás

```
<HEAD>
<SCRIPT>
var win                                                  // a létrehozott ablak
function getHistory(jog) {                               // history lekérdezése
  if (jog!=null) netscape.security.PrivilegeManager.enablePrivilege(jog)
  return window.history+""
}
function showHistory(jog) {                            // history megjelenítése
  win = window.open("", "win", "resizable,scrollbars")
  win.document.writeln("<HEAD><TITLE>SecurityTest "+jog+" joggal")
  win.document.writeln("</TITLE><HEAD><BODY>")
  win.document.writeln("<H1>Előtte jog nélkül</H1>")
  win.document.writeln(getHistory(null))
  win.document.writeln("<H1>Most "+jog+" joggal</H1>")
  win.document.writeln(getHistory(jog))
  win.document.writeln("<H1>Utána jog nélkül</H1>")
  win.document.writeln(getHistory(null))
  win.document.writeln("</BODY>")
  win.document.close()
}
function showError(hiba) {                                   // hiba kijelzése
  alert("HIBA!\n"+hiba)
  win.close()                                           // és az ablak becsukása
  return true
}
</SCRIPT>
</HEAD>
<BODY>
<H1>SecutityTest</H1>
<FORM>
Jog:<INPUT TYPE="text" NAME="jog" VALUE="UniversalBrowserAccess">
<INPUT TYPE="button" VALUE="history" ONCLICK="
  showHistory(this.form.jog.value)
">
</FORM>
<SCRIPT>
window.onerror = showError                                 // saját hibakezelés
</SCRIPT>
</BODY>
```



G.27. ábra: Jogosultságkérő dialógus képe

G.28. ábra: Sikeres listázás képe

G.6. package netscape.javascript

public final class JSObject

JavaScript objektumokat becsomagoló osztály.

public Object **call**(String, Object[])
: JavaScript függvény meghívása. Az első paraméter a függvény neve, a második paraméter pedig a függvény argumentumait tartalmazó tömb.

public Object **eval**(String)
: JavaScript forrásszöveg kiértékelése és végrehajtása.

public Object **getMember**(String)
: A Javascript objektum adott nevű mezőjének lekérdezése.

public Object **getSlot**(int)
: A JavaScript objektum (tömb) adott indexű mezőjének lekérdezése.

public static JSObject **getWindow**(java.applet.Applet)
: Az appletet tartalmazó window JavaScript objektum lekérdezése.

public void **removeMember**(String)
: A JavaScript objektum adott nevű mezőjének eltávolítása.

public void **setMember**(String, Object)
: A JavaScript objektum adott nevű mezője értékének beállítása.

public void **setSlot**(int, Object)
: A JavaScript objektum (tömb) adott indexű mezője értékének beállítása.

public class JSException extends Exception

JavaScript kivételeket becsomagoló osztály.

public **JSException**()
: JSException létrehozása.

public **JSException**(String)
: JSException létrehozása adott hibaszöveggel.

public **JSException**(String, String, int, String, int)
: JSException létrehozása adott hibaszöveggel, fájlnévvel, sorszámmal, a hiba forrásával és a hibás token indexével.

G.7. package netscape.plugin

public class **Plugin**

Plug-in programokat reprezentáló osztály.

public void **destroy**()
: A plug-in megsemmisítésekor hívódik meg.

public int **getPeer**()
: A JavaScript objektum natív azonosítóját adja vissza.

public JSObject **getWindow**()
: A plug-int tartalmazó window JavaScript objektum lekérdezése.

public void **init**()
: A plug-in inicializálásakor hívódik meg.

public boolean **isActive**()
: Megadja, hogy a plug-in aktív-e.

G.8. package netscape.security

public class **AppletSecurity** extends java.lang.SecurityManager

Appletek biztonsági megkötéseit megvalósító osztály.

public class **AppletSecurityException** extends java.lang.SecurityException

Appletek biztonsági megkötéseinek megsértésekor kiváltódó kivétel.

public class **ForbiddenTargetException** extends java.lang.RuntimeException

Jogosultság nem engedélyezésekor kiváltódó kivétel.

public class **ParameterizedTarget** extends UserTarget

public **ParameterizedTarget**(String név, Principal, String rizikócsoport, String rizikószín, String leírás, String leíró-url)
Konstruktor.

public final class **Principal**

Jogosultságkérőt reprezentáló osztály. A következő konstansokat definiálja:
public final static int

- CERT : Jogosultságkérő típus, rendszerint egy publikus kulcs.
- CERT_FINGERPRINT : Jogosultságkérő típus, rendszerint egy publikus kulcsból képzett ellenőrzőösszeg.
- CODEBASE_EXACT : Jogosultságkérő típus, rendszerint egy adott URL.
- CODEBASE_REGEXP : Jogosultságkérő típus, rendszerint egy adott mintának megfelelő URL.

public **Principal**(URL)
Adott URL-hez CODEBASE_EXACT típusú jogosultságkérés létrehozása.
public **Principal**(int, String)
Adott típusú jogosultságkérés létrehozása.
public **Principal**(int, byte[])
Adott típusú jogosultságkérés létrehozása.
public String **getNickname**()
Jogosultságkérés megnevezésének lekérdezése.
public String **getVendor**()
Jogosultságkérő nevének lekérdezése.
public boolean **isCert**()
Igazat ad vissza, ha a jogosultságkérés típusa CERT.
public boolean **isCertFingerprint**()
Igazat ad vissza, ha a jogosultságkérés típusa CERT_FINGERPRINT.
public boolean **isCodebase**()
Igazat ad vissza, ha CODEBASE_EXACT vagy CODEBASE_REGEXP a jogosultságkérés típusa.
public boolean **isCodebaseExact**()
Igazat ad vissza, ha a jogosultságkérés típusa CODEBASE_EXACT.
public boolean **isCodebaseRegexp**()
Igazat ad vissza, ha a jogosultságkérés típusa CODEBASE_REGEXP.
public String **toVerboseHtml**()
Visszaadja a jogosultságkérés részletes HTML leírását.
public String **toVerboseString**()
Visszaadja a jogosultságkérés részletes leírását.

public final class **Privilege**

Jogosultság birtoklását reprezentáló osztály. A következő konstansokat definiálja:
public final static int

- ALLOWED : Jogosultság engedélyezett.
- BLANK : Üres jogosultság: se nem engedélyezett, se nem tiltott.
- FOREVER : Jogosultság a kérés érvényességének lejártáig érvényes.
- FORBIDDEN : Jogosultság a kérés érvényességének lejártáig elutasítva.
- N_DURATIONS : Jogosultság érvényességének hossza.
- N_PERMISSIONS : Jogosultság érvényessége.
- SCOPE : Jogosultság csak a kérő metóduson belül érvényes.
- SESSION : Jogosultság a böngésző futásának ideje alatt érvényes.

public static int **add**(int, int)
: Visszaadja a paraméterként kapott két jog eredőjét.

public static Privilege **add**(Privilege, Privilege)
: Visszaadja a paraméterként kapott két jog eredőjét.

public static Privilege **findPrivilege**(int engedély, int hossz)
: Visszaadja a paramétereknek megfelelő jogosultság engedélyt.

public int **getDuration**()
: Visszaadja az engedély időtartamát.

public int **getPermission**()
: Visszaadja az engedély típusát.

public boolean **isAllowed**()
: Igazat ad vissza, ha a jogosultság engedélyezett.

public boolean **isBlank**()
: Igazat ad vissza, ha a jogosultság BLANK.

public boolean **isForbidden**()
: Igazat ad vissza, ha a jogosultság tiltott.

public boolean **sameDuration**(Privilege)
: Igazat ad vissza, ha a megadott jogosultság időtartama ugyanolyan, mint az aktuálisé.

public boolean **sameDuration**(int)
: Igazat ad vissza, ha a megadott jogosultság időtartama ugyanolyan, mint az aktuálisé.

public boolean **samePermission**(Privilege)
: Igazat ad vissza, ha a megadott jogosultság engedélyezése ugyanolyan, mint az aktuálisé.

public boolean **samePermission**(int)
: Igazat ad vissza, ha a megadott jogosultság engedélyezése ugyanolyan, mint az aktuálisé.

public final class **PrivilegeManager**

Jogosultságokat kezelő osztály. A következő konstansokat definiálja:
public final static int

- PROPER_SUBSET : Megfelelő részhalmaz.
- EQUAL : Azonos halmazok.
- NO_SUBSET : Nem részhalmaz.
- SIGNED_APPLET_DBNAME : .
- TEMP_FILENAME : .

public boolean **checkMatchPrincipal**(Class[, int])
: Visszaadja, hogy a megadott osztály rendelkezik-e minden olyan joggal, amellyel az adott mélységű szülőhierarchia is rendelkezik. A második paramétert elhagyva csak a közvetlen hívóval történik az ellenőrzés.

public boolean **checkMatchPrincipal**(Principal, int)
: Visszaadja, hogy a megadott osztály rendelkezik-e minden olyan joggal, amellyel az adott mélységű szülőhierarchia is rendelkezik.

public boolean **checkMatchPrincipalAlways**()
: A továbbiakban betöltésre kerülő osztályoknak a hívás pillanatában aktuális minden joggal rendelkezniük kell.

public void **checkPrivilegeEnabled**(Target[, Object]) throws ForbiddenTargetException
: Ellenőrzi a kért jogosultság meglétét [a kiegészítő adatok figyelembevételével]. Ha a jogosultság nem engedélyezett, kivételt vált ki.

public void **checkPrivilegeGranted**(String) throws ForbiddenTargetException
: Ellenőrzi a kért jogosultság meglétét. Ha a jogosultság nem engedélyezett, kivételt vált ki.

public void **checkPrivilegeGranted**(Target[[, Principal], Object]) throws ForbiddenTargetException
: Ellenőrzi a kért jogosultság meglétét [a megadott jogosultságkérő esetében] [a kiegészítő adatok figyelembevételével]. Ha a jogosultság nem engedélyezett, kivételt vált ki.

public int **comparePrincipalArray**(Principal[], Principal[])
: Jogosultságkérők halmazának összehasonlítása. Az eredmény PROPER_SUBSET, EQUAL vagy NO_SUBSET lehet.

public static void **disablePrivilege**(String|Target)
: Adott jogosultság tiltása.

public static void **enablePrivilege**(String) throws ForbiddenTargetException
: Jogosultság kérése. Ha a jogosultság nem engedélyezett, kivételt vált ki.

public static void **enablePrivilege**(Target[[, Principal], Object]) throws ForbiddenTargetException
: Jogosultság kérése [a megadott jogosultságkérő esetében] [a kiegészítő adatok figyelembevételével]. Ha a jogosultság nem engedélyezett, kivételt vált ki.

public Principal[] **getClassPrincipals**(Class)
: Adott osztály jogosultságkérőinek listája.

public Principal[] **getClassPrincipalsFromStack**(int)
: Adott osztály jogosultságkérőinek listája a hívási verem adott mélységében.

public static Principal[] **getMyPrincipals**()
: A hívó osztály jogosultságkérőinek listája.

public static PrivilegeManager **getPrivilegeManager**()
: Az aktuális jogosultságintéző lekérdezése.

public PrivilegeTable **getPrivilegeTableFromStack**()
: A hívó jogosultságtáblájának lekérdezése.

public static Principal **getSystemPrincipal**()
: A Java rendszerosztályokhoz tartozó jogosultságkérőt kérdezi le.

public boolean **hasPrincipal**(Class, Principal)
: Megadja, hogy az osztályt az adott jogosultságkérő aláírta-e.

public boolean **isCalledByPrincipal**(Principal[, int])
: Megadja, hogy az adott jogosultságkérő aláírta-e a [megadott veremmélységből] hívó osztályt.

public void **revertPrivilege**(String|Target)
: Visszaállítja az adott jogosultság engedélyezése vagy elutasítása előtti állapotát.

public class PrivilegeTable

Jogosultságok táblázata. Tulajdonképp Hashtable-nek megfelelően működik, ahol a jogosultságokhoz azok engedélyezettségét/tiltását rendeli. Még be nem állított jogosultságok lekérdezésekor BLANK, FOREVER típust ad vissza.

public class Target

Jogosultságot reprezentáló osztály.

public **Target**(String név, Principal, String rizikócsoport, String rizikószín, String leírás, String leíró-url[, Target[] makrójogosultság-összetevők])
: Konstruktor.

public Privilege **checkPrivilegeEnabled**(Principal[][, Object])
: Megadja a kérvényezőknek adható jogosultságengedélyt [a kiegészítő adatok figyelembevételével].

public Privilege **checkPrivilegeEnabled**(Principal, Object)
: Megadja a kérvényezőnek adható jogosultságengedélyt a kiegészítő adatok figyelembevételével.

public Privilege **enablePrivilege**(Principal, Object)
: Megadja a kérvényezőnek adható jogosultságengedélyt a kiegészítő adatok figyelembevételével.

public static Target **findTarget**(String[, Principal])
: [Adott jogosultságkérőhöz tartozó] jogosultság elérése. Nem létező jogosultság esetén null az eredmény.

public static Target **findTarget**(Target)
: Adott jogosultság elérése. Nem létező jogosultság esetén null az eredmény.

public String **getDescription**()
: A jogosultság leírását adja vissza.

public String **getDetailedInfo**(Object)
: A jogosultság leírását adja vissza a kiegészítő adatok figyelembevételével.

public String **getHelpUrl**()
: A jogosultság HTML leírásának címét adja vissza.

public String **getRisk**()
: A jogosultság rizikócsoportját adja vissza.

public String **getRiskColor**()
: A jogosultság rizikója színét adja vissza #rrggbb formában.

public static Target **getTargetFromDescription**(String)
: Visszaadja a jogosultságot annak leírása alapján.

public final Target **registerTarget**(String)
: Jogosultság regisztrálása. Ha már van ilyen néven regisztrált jogosultság, az lesz a visszatérési érték.

public class UserDialogHelper

Dialógusablakok szövegkonstansait reprezentáló osztály.

public final static String **targetRiskColorHigh**()
: Megadja a magas rizikócsoportot jelölő szöveg színét.

public final static String **targetRiskColorLow**()
: Megadja az alacsony rizikócsoportot jelölő szöveg színét.

public final static String **targetRiskColorMedium**()
: Megadja a közepes rizikócsoportot jelölő szöveg színét.

public final static String **targetRiskHigh**()
: Megadja a magas rizikócsoportot jelölő szöveget.

public final static String **targetRiskLow**()
: Megadja az alacsony rizikócsoportot jelölő szöveget.

public final static String **targetRiskLow**()
Megadja a közepes rizikócsoportot jelölő szöveget.

public class **UserTarget** extends Target

A jogosultság megadásakor a felhasználó is interaktívan beavatkozhat egy megjelenő dialógusdoboz segítségével.

public **UserTarget**(String név, Principal, String rizikócsoport, String rizikószín, String leírás, String leíró-url)
Konstruktor.