SKRIPTA ZA VEŽBE IZ PREDMETA

INTERNET TEHNOLOGIJE

JavaScript, HTML DOM

Laboratorija za elektronsko poslovanje

Beograd 2010.

SADRŽAJ

1 UVOD

1.1 Tehnologije skriptovanja serverske odnosno klijentske strane

1.1.1 Skripovi serverske strane

Server - side skripting je Web server tehnologija koja omogućava da se korisnički zahtev ispunjava pomoću skripta koji se izvršava na serverskoj strani kako bi se generisale dinamičke *HTML* stranice. Obično se koristi da bi se interaktivne Web stranice povezale sa bazama podataka, radi identifikacije korisnika, ažuriranja sadržaja, stvaranja raznih diskusionih grupa i još zbog mnogo toga. Razlika između skriptovanja na serverskoj i korisničkoj strani je u tome što se kod korisničke strane skriptovi izvršavaju u veb čitaču korisnika, najčešće pomoću *JavaScript* - a.

Tokom prvih godina veba, sve se obavljalo preko *C* programa, *Perl* ili Shell skripti (skripti pisanih za komandni interpreter operativnog sistema) pomoću *Common Gateway Interface*-a. Te skripte su bile izvršavane od strane operativnog sistema, a rezultati su prosleđivani veb serveru. Danas se ovi, i drugi online scripting jezici, kao što su *PHP* i *ASP* najčešće direktno izvršavaju od strane veb servera ili preko modula (*mod_perl* i *mod_php*) koji se dodaju veb serveru. I jedan i drugi oblik se može iskoristiti da se napravi kompleksan veb sajt, ali se drugi način pokazao efikasnijim, zbog bržeg izvršavanja i manjeg korišćenja resursa. Najčešće korišćene server - side tehnologije skriptnih jezika su:

- ASP
- ASP.NET
- ColdFusion
- JSP (Java Server Pages)
- SSI (Server Side Includes)
- PERL
- Python i TCL
- PHP

1.1.2 Skriptovi klijentske strane

Client – side scripting tehnologije se obično odnose na grupu računarskih programa na vebu koji se izvršavaju na strani klijenta, tj. od strane veb čitača na klijentskim računarima, umesto na strani servera, od strane aplikacija veb servera. Ovaj vid računarskog programiranja je bitan deo DHTML (Dynamic HTML) koncepta, zato što omogućava da veb strane budu skriptovane. Na taj način, veb strane postaju dinamičke i menjaju svoj sadržaj u zavisnosti od akcija korisnika, uslova u okruženju (kao npr. doba dana) i ostalih varijabli.

Pored skripti serverske strane, postoje i skripte **klijentske strane** (**client – side)**. Web programeri pišu client – side skripte u jezicima poput JavaScript-a (Client – side JavaScript). Skripte klijentske strane su uglavnom „prikačene" u okviru HTML (HyperText Markup Language) dokumenta, ali takođe mogu biti ubačene u zaseban fajl, koji je referenciran u HTML dokumentu. Odmah po zahtevu za određenom stranom koja sadrži klijentski skript, neophodni fajlovi se šalju na klijentski računar od strane veb servera. Nakon toga, klijentski

čitač izvršava skriptu i prikazuje dokument (veb stranicu), sa svim vidljivim izlazima iz klijentske skripte. Skripte klijentske strane takođe mogu sadržati instrukcije koje veb čitač treba da sledi ako korisnik interaguje sa stranicom na određeni način, npr. ako klikne na odgovarajuće dugme na dokumentu. Takve instrukcije se mogu izvršavati bez komunikacije sa veb serverom.

Ako korisnici pogledaju izvorni kod veb strane koja sadrži klijentsku skriptu, mogu videti i kod te skripte. Mnogi veb programeri uče kako da pišu klijentske skripte istražujući izvorne kodove dokumenata drugih autora. S druge strane, kod serverskih skripti, korisnici ne mogu videti izvorni kod, jer se on izvršava na veb serveru i odmah se nakon toga prebacuje u HTML. Korisnici nisu čak ni svesni da je izvršena serverska skripta kada istražuju neki HTML dokument.

Skripte klijentske strane imaju veći pristup informacijama i funkcijama koje se nalaze na klijentskim računarima, dok skripte serverske strane imaju veći pristup istim na serveru. Skripte serverske strane zahtevaju instaliran intrpreter skriptnog jezika na serveru i proizvode isti output bez obzira na brauzer klijenta, operativni sistem koji klijent koristi i druge detalje klijentskog sistema. Skripte klijentske strane ne zahtevaju dodatno instaliran softver na serveru i tako postaju popularne autorima veb stranica koji imaju malo administrativnih ovlašćenja na serveru. Međutim, skripte klijentske strane zahtevaju da klijentski brauzer „razume" skriptni jezik u kome su pisane. Baš zbog toga je krajnje nepraktično pisanje skripti u jezicima koji nisu podržani od strane većine veb čitača. Nažalost, čak i skriptni jezici koji su podržani od strane širokog varijiteta veb čitača nisu uvek implementirani na isti način na svim čitačima i operativnim sistemima. Baš zbog toga, autori veb strana koje sadrže kod skriptnog jezika treba pažljivo da analiziraju „ponašanje" tog koda na različitim platformama pre nego što ga stave u upotrebu.

Skriptovi klijentske strane se upotrebljavalju:

- Da bi se dobili podaci koji su na ekranu klijenta ili u browseru klijenta
- Za online računarske igre
- Za personalizaciju prikaza (bez reloadovanje Web strane)
- Za predprocesiranje formi

Skriptovi serverske strane se upotrebljavaju:

- Za personalizaciju brauzera
- Za procesiranje (obradu) formi
- Za izgradnju i prikazivanje veb strana sa informacijama iz baza podataka.

Postoji i treća tehnologija skriptnih jezika koja koristi i serverske i klijentske tehnologije. To je *udaljeno skriptovanje* (remote scripting) koje koristi najbolje od oba, brzinu klijentskih skripti sa fleksibilnošću serverskog koda. Međutim, ni udaljeno skriptovanje nije bez problema zbog načina na koji radi.

Udaljeno skriptovanje, funkcioniše tako što dozvoljava skriptama klijentske strane pristup funkcijama koje se nalaze na strani servera, a koje server obrađuje. Ovaj pristup omogućuje kodu klijentske strane da koristi tehnologije skriptovanja koje im u uobičajenim okolnostima nisu dostupne, dok se generišu odgovori servera bez potrebe osvežavanja (refresh) veb strane. Nažalost, ograničenja brauzera koja se odnose na klijentske skriptne jezike postoje i kod udaljenog skriptovanja.

Ako se sve skripte pokreću direktno sa servera, on se dodatno opterećuje (obrada dodatnih podataka), a i potrebno je da prođe odgovarajući vremenski interval da server odgovori na zahteve posetilaca sajta.

Klijentske i udaljene skripte mogu da pomognu u predprocesiranju zahteva korisnika, ako to veb čitač podržava, ali kako podrška čitača nije uvek zagarantovana, nije preporučljivo osloniti se na procesiranje formi od strane klijenta.

2 Java Script

2.1 Osnovne karakteristike

Java Script pripada grupi jezika za skriptovanje, pre svega klijentske strane, mada se može izvršavati i na serveru (runat="server"). Java Script je najpopularniji jezik na Internetu, koji je dizajniran da poveća interaktivnost HTML strana. JavaScript jezik je nastao u kompaniji Netscape pod prvobitnim imenom LiveScript koje mu je dodelio direktor projekta Brendan Eich. Ovaj jezik je razvijan za dve namene.

Prva namena mu je bila da posluži kao skript jezik koji će administrator Web servera moći da koristi za upravljanje serverom i povezivanje njegovih strana sa drugim servisima, kao što su programi za pretraživanje koje korisnici koriste kako bi pronašli željenu informaciju.

Druga namena je bila da ovi skriptovi budu korišćeni na strani klijenta, u HTML dokumentima, kako bi na mnoge načine poboljšali Web strane. Na primer, autor bi pomoću LiveScript jezika mogao da kontroliše da li je korisnik pravilno uneo tražene podatke, pre nego što se ti podaci pošalju do servera i server pošalje odgovor. Time se štedi vreme i povećava efikasnost stranice, a sva izračunavanja i provere se vrše na klijentu tako da se zapošljava neiskorišćena snaga računara klijenta.

Početkom decembra 1995. godine, Netscape i Sun su zajednički objavili da će se skript jezik ubuduće zvati JavaScript. Iako je Netscape imao nekoliko dobrih marketinških razloga za usvajanje ovog imena, izmena je možda izazvala više konfuzije nego što je iko očekivao. Ispostavilo se da je prava poteškoća bila da se svetu objasni razlika između jezika Java i jezika JavaScript.

Nakon raznih poboljšanja i nestandardnih kopija, međunarodna organizacija za standarde u informacionim i komunikacionim sistemima (European Computer Manufactures Association - ECMA international) standardizovala je JavaScript 1997. godine pod nazivom ECMAscript. Pod ovom organizaciom je nastavio da se razvija do danas i imao je veliki broj značajnih poboljšanja. Trenutno, poslednja standardizovana verzija JavaScripta je 1.8.5.

JavaScript programi se koriste da bi detektovali i reagovali na događaje koji su inicirani od strane korisnika, kao npr. prelazak mišem preko linka i grafičkog objekta na strani. JavaScript kod može da poboljša kvalitet sajta navigacionim pomagalima, skrolovanim porukama, dialog prozorima, dinamičnim slikama, kolicima za kupovinu (shopping cart), itd. JavaScript omogućava korisnicima da kontrolišu izgled veb strane dok se dokument „tumači" od strane čitača. Uz pomoć JavaScripta, možemo da proverimo tačnost podataka koje je korisnik ukucao u formu, pre nego što se ti podaci prebace na server. JavaScript testira brauzer korisnika da vidi da li ima neophodne dodatke (plugins) za učitavanje tražene strane i ako ih nema, JavaScript usmerava korisnika na drugi sajt da ih nabavi. Njegovo jezgro sadrži osnovne elemente programskih jezika kao što su promenljive, tipovi podataka, kontrolne petlje, if/else izjave, switch izjave, funkcije i objekte. JavaScript se koristi za aritmetičke operacije, manipulacije datumom i vremenom, nizovima, stringovima i objektima. On takođe čita i upisuje vrednosti kuki fajlova i dinamički kreira HTML na osnovu vrednosti kukija.

Kada brauzer korisnika prihvati stranu koja sadrži JavaScript kod, kod se šalje JavaScript interpreteru, koji izvršava skriptu. Budući da svaki čitač ima svoj interpreter, često postoje

razlike u načinu na koji se skript izvršava. I dok se konkurentske kompanije stalno utrukuju poboljšavajući i modifikujući njihove čitače, novi problemi se javljaju. Ne što samo različite verzije čitača dovode do inkompatibilnosti, već se problemi javljaju i sa različitim verzijama istog brauzera. JavaScript engine (*JavaScript interpreter*) je interpreter koji interpretira JS source kod i izvršava skript. Od 2009.godine svi važniji browseri u svojim novim verzijama podržavaju JS engine, koji je standardizovan u okviru ECMAScript (ECMA-262) Edition 3.

Neke od mogućnosti koje Java Script pruža svojim korisnicima /programerima su:

- Kada Web strana treba da odgovori ili direktno reaguje na korisnikovu interakciju sa formama (polja za unos, dugmad, radio dugmad...) i hipertekstualnim vezama
- Kada je potrebno da se kontroliše kretanje kroz više okvira, priključke ili Java aplete u zavisnosti od korisnikovog izbora u HTML dokumentu
- Kada je potrebno da se na računaru klijenta izvrši validacija unetih podataka pre nego se ti podaci pošalju serveru
- Kada je potrebno da se stranica učini lepšom i organizovanijom pomoću padajućih menija, dinamičke promene izgleda i rasporeda elemenata na stranici (uz pomoć CSS-a i DOM-a)
- Reagovanje na određene događaje (klik mišem, učitavanje stranice…)
- Čitanje i pisanje HTML stranica
- Čitanje i menjanje sadržaja HTML strane

Razlika između JavaScript i Java

Bez obzira na nazive, Java i JavaScript su dva potpuno različita jezika po konceptu i dizajnu. Oni predstavljaju dve različite tehnike programiranja na Internetu. Java je programski jezik. JavaScript je (kako mu i samo ime kaže) script jezik. Razlika je u tome što se u Javi mogu kreirati pravi programi, koji mogu da se izvršavaju potpuno nezavisno od WWW čitača, ili Java apleti koji se mogu pozivati iz HTML dokumenta i koji se dovlače preko mreže i onda izvršavaju u okviru WWW čitača (browser-a).

Java Script	Java
Ne kompajlira se, već samo interpretira kod klijenta.	Kompajlira se na serveru pre izvršenja kod klijenta.
Objektno zasnovan. Koristi postojeće objekte, bez klasa ili nasleđa.	Objektno orjentisan. Apleti sadrže objekte sastavljene od klasa sa nasleđem.
Kod je integrisan u HTML.	Applet je izdvojen iz HTML-a.
Tip promenljive se ne deklariše.	Tip promenljive mora biti deklarisan.
Dinamička povezanost. Objektne reference proveravaju se u vreme izvršavanja.	Statička povezanost. Objektne reference moraju postojati u vreme kompajliranja.

Tabela 1 Poređenje JavaScript-a i Java-e

2.2 Gde smestiti Java Script kod i kako se izvršava

Komande JavaScript-a su u tekstualnom formatu, tako da se za pisanje skriptova može koristiti bilo koji tekst editor, npr. Notepad.

Da bi browser prepoznao JavaScript kôd, potrebno je da se taj kôd postavi unutar para oznaka <*script*> i </*script*>. U zavisnosti od Web čitača oznaka <*script*> ima širok dijapazon atributa koje se mogu podesiti kako bi se upravljalo skriptom. Jedan od atributa koji je zajednički za Navigator i Internet Explorer jeste atribut *language*. Ovo je osnovni atribut zato što svaka verzija čitača prihvata različite jezike za skriptovanje. Mogućnost koju prihvataju svi skriptabilni čitači jeste jezik JavaScript, kako je prikazano u sledećem redu:

```
<script language=˝JavaScript˝>
<!--znak za komentar u HTML-u
   Naredbe JavaScript-a
-->
</script>
```

Postoje stari čitači koji ne podržavaju JavaScript, oni će ignorisati nepoznate tagove <script> i </script> i njihov sadržaj tretirati kao sadržaj stranice koji treba prikazati. Da bi se izbeglo prikazivanje JavaScript kôda, na stranici naredbe skripta se smeštaju između simbola za HTML komentar (<!-- -->).

Kôd za povezivanje stranice sa eksternom bibiliotekom funkcija treba da se nađe u zaglavlju stranice (između tagova <*head*> i </*head*>), kako bi se biblioteka učitala cela pre nego što se izvrše drugi skriptovi na stranici.

Sledi primer JavaScripta.

```
<html>
<head>
<title>Jednostavan JavaScript</title>
</head>
<body>
Tekst prikazan na uobicajen nacin.
<br>
<script language=”JavaScript”>
<!-— skrivanje JavaScripta od citaca koji ga ne citaju
document.write(“Zdravo svima!”)
-->
</script>
</body>
</html>
```

Reč *document.write* je standardna Java Script komanda za ispisivanje izlaza na stranici pa će gornji kôd prikazati na stranici sledeći izlaz: „Zdravo svima!“.

Drugi način za uključivanje kôda napisanog JavaScript-om je povezivanje spoljašnje datoteke koja sadrži naredbe ili funkcije. Povezivanje u ovom slučaju se obavlja uz pomoć atributa *src* oznake <*script*> i to na sledeći način:

```
<script language=“JavaScript“ src=“mojskript.js“></script>
```

3 Osnove jezika - tipovi podataka, literali, promenljive, operatori

3.1 Promenljive

Definisanje promenljivih

Postoji nekoliko načina za definisanje promenljivih u JavaScript-u. Najčešće se promenljiva deklariše uz pomoć ključne reči *var* iza koje sledi ime promeljive. Dakle, ako hoćemo da definišemo promenljivu *godine*, naredba JavaScript-a glasi:

```
var godine;
```

Promenljivoj se može dodeliti vrednost pri deklarisanju promenljive (inicijalizacija) ili kasnije pomoću operatora dodele. Najčešće korišćeni operator za tu svrhu je znak jednakosti. U sledećem primeru, pri deklarisanju promenljive izvršena je i inicijalizacija promenljive (dodeljivanje početne vrednosti):

```
var godine = 22;
```

Ako se vrši deklaracija promenljive bez inicijalizacije, tada ta promenljiva dobija vrednost koju joj dodeljuje čitač. Najčešće je to vrednost praznog objekta. Kada vrednost promenljivoj želimo dodeliti kasnije u okviru naredbi skripta to se postiže navođenjem imena deklarisane promenljive i dodeljivanjem željene vrednosti pomoću operatora '='. Na primer:

```
godine = 22;
tekst = „Vrednost promenljive je ovaj tekst“;
```

Ključna reč *var* se koristi samo pri inicijalizaciji, odnosno deklarisanju promenljive, tj. samo jednom za vreme postojanja jedne promenljive. Za razliku od strogo tipiziranih jezika, u JavaScript-u, pri deklaraciji promenljive ne definiše se njen tip. U toku izvršavanja skripta jedna promenljiva, u stvari, može uzeti vrednosti različitih tipova.

Promenljivoj se kao vrednost može dodeliti i rezultat nekog izraza. Na primer:

```
var prviSabirak = 1;
var drugiSabirak = 2;
var zbir  = prviSabirak + drugiSabirak;
```

Java Script promenjiva može da sadrži bilo koji tip vrednosti. Što se tiče imena promenljivih, postoje određena pravila koja moraju da budu zadovoljena. U JavaScriptu imena promenljivih se sastoje od sekvenci slova (a-z, A-Z), cifara (0-9) i donje crte (_). JavaScript razlikuje velika i mala slova. Ključne reči (for, if, else, class, byte, int...) ne mogu se koristiti u nazivu identifikatora. U JavaScriptu ne moramo eksplicitno da deklarišemo promenljivu (mada je to dobra praksa).Promenljiva ne može da sadrži razmak. Najsigurnije je da ime sadrži samo jednu reč. Ako postoji potreba za imenom od dve reči, preporučuje se korišćenje jedne od dve konvencije. Prva je da se te dve reči razdvoje donjom crtom, dok u drugoj kombinacija reči počinje malim slovom, a početno slovo druge reči je veliko.

Prilikom deklarisanja promenljivih treba obratiti pažnju na sledeće :

- Pokušati da se deklarišu sve varijable koje će se koristi na početku programskog koda, čak iako još nema konkretnih vrednosti za njih.
- Koristiti ispravna imena varijabli. Ne koristiti rezervisane reči i reči koje su predugačke ili teške za pamćenje.
- Imena varijabli su osetljiva na velika i mala slova (case sensitive), *MojeIme* nije isto što i *mojeime.*
- Ne treba davati slična imena različitim varijablama.
- Iako nije uvek obavezno da se koristi *var* ključna reč prilikom deklarisanja promenljivih, dobro je koristiti je, sigurnije je.

3.1.1 Doseg promenljivih

U JavaScript-u se promenljive mogu definisati izvan ili unutar funkcije (o funkcijama će kasnije biti reči). Promenljive koje su definisane izvan funkcija zovu se *globalne* promenljive; one koje su definisane unutar funkcija zovu se *lokalne* promenljive. Globalne promenljive u JavaScript-u donekle se razlikuju od onih u većini drugih jezika. U JavaScript-u termin 'globalno' odnosi se na tekući dokument koji je učitan u prozor ili okvir čitača. Prema tome, kada se inicijalizuje jedna promenljiv, to znači da sve naredbe na toj stranici, uključujući i one unutar funkcija, imaju direktan pristup vrednosti te promenljive.

Važno je napomenuti da onoga trenutka kada se stranica ukloni iz čitača, sve globalne promenljive koje su definisane za tu stranicu brišu se iz memorije. Ako nam je potrebna vrednost koja će biti postojana iz stranice u stranicu, koriste se druge tehnike da uskladište tu vrednost (na primer, globalnu promenljivu u podešavanju okvira, ili kolačić).

Nasuprot globalnoj promenljivoj, lokalna promenljiva je definisana unutar funkcije. Doseg lokalne promenljive su samo naredbe u sklopu funkcije. Nijedna druga funkcija niti naredba izvan funkcije nemaju pristup lokalnoj promenljivoj.

Treba naglasiti još jednu bitnu stvar u vezi sa lokalnim i globalnim promenljivama. Naime, JavaScript dozvoljava da se na istoj stranici koriste globalne i lokalne promenljive sa istim imenom. Međutim, kada se to radi treba biti veoma obazriv i treba znati sledeće. Kada se unutar funkcije inicijalizuje vrednost lokalne promenljive istog imena kao i globalna promenljiva, mora se koristiti ključna reč *var* ispred imena lokalne promenljive. U suprotnom, interpreter će tu naredbu shvatiti kao da treba da dodeli vrednost globalnoj promenljivoj koja ima to ime i dobiće se neočekivani rezultat.

3.1.2 Imena promenljivih

Pri dodeljivanju imena promenljivama, važe ista pravila kao i kod većine drugih programskih jezika. Preporuka je da se dodeljuju opisna imena promenljivama kako bi se olakšalo čitanje i održavanje kôda. Pri dodeljivanju imena promenljivama u JavaScript-u važe sledeća ograničenja:

- Ne mogu se koristiti ključne reči JavaScript-a za dodeljivanje imena. To uključuje sve ključne reči koje se sada koriste i sve koje su u planu da budu uvedene.
- Ime promenljive ne sme da sadrži razmak
- JavaScript je *case sensitive*, što znači da promenljive *godine* i *Godine* nisu jedno te isto
- Ime promenljive ne sme da počne brojem
- Ime promenljive ne sme da sadrži znake interpunkcije, osim znaka za podvlačenje

3.2 Tipovi podataka

Svaka napisana naredba JavaScript-a na neki sebi svojstven način obrađuje podatke koji joj se proslede, na primer prikaz teksta na ekranu uz pomoć Java Scripta ili uključivanje i isključivanje radio dugmeta u obrascu. Svaki pojedinačni deo informacije u obrascu se naziva vrednost. U JavaScriptu postoji nekoliko tipova vrednosti. Formalni tipovi podataka su:

- String - niz znakova unutar navodnika
- Number - bilo koji broj koji nije pod navodnocima
- Boolean - logičko istinito ili neistinito
- Null - lišeno vrednosti
- Object - sva svojstva i metodi koji pripadaju objektu ili nizu
- Function - definicija funkcije

Stringovi

String je niz karaktera sačinjen od nula ili više karaktera zatvorenih u jednostrukim (‘) ili dvostrukim (“) navodnicima.

Primeri stringova:

```
• “prvi”
• ‘1234’
• “PRVA LINIJA \n druga linija”
```

Takođe je često neophodno koristiti navodnike unutar niza karaktera. To se može uraditi korišćenjem obrnute kose crte. Na primer:

```
var tekst=”<P>On je isao u skolu \”Naziv skole \”. ”
document.write(tekst)
```

U stringovima je dozvoljeno koristiti sledeće specijalne karaktere:

- \b = pomeraj za jedno mesto ulevo (backspace)
- \f = pomeraj jedan red dole (form feed)
- \n = na početak novog reda (new line character)
- \r = return (carriage return)
- \t = tabulator (tab).

JavaScript je slabo tipiziran jezik. Tipovi podataka će biti automatski konvertovani zavisno od mesta njihove upotrebe u programu. Tako na primer možemo definisati i inicijalizovati sledeću promenljivu:

```
var promenljiva=11
```

a kasnije je možemo predefinisati, dodeljujući joj niz karaktera, na primer

```
promenljiva=”Osnove JavaScript-a”
```

U kombinaciji broja i stringa, Java Script konvertuje broj u string. Na primer:

```
x = "Primer za kombinaciju " + 11
```

```
y = 11 + " jos jedan primer."
```

Za konverziju stringa u broj, koriste se funkcije:

- **EVAL** - ocenjuje string i ako je moguće pretvara ga u broj;
- **parseINT** - konvertuje string u integer, ako je moguće;
- **parseFLOAT** - konvertuje string u floating-point broj, ako je moguće.

Dobra je praksa definisati promenljivu sa var, kada je uvodimo. Specijalna ključna reč null ukazuje da je promenljivoj dodeljena null vrednost. To ne znači da je promenljiva nedefinisana. Promenljiva je nedefinisana kada joj nije dodeljena nikakva vrednost i tada je ne možemo dodeliti drugoj promenljivoj ili koristiti u izrazima, a da ne dobijemo run-time grešku (greška pri izvršavanju programa).

Celi brojevi

Celi brojevi u JavaScriptu mogu biti predstavljeni u tri osnove: u decimalnom (baza 10), u oktalnom (baza 8) i heksadecimalnom (baza 16) formatu.
Decimalni celi brojevi se predstavljaju kao niz cifara (0-9) bez vodeće nule.
Oktalni celi brojevi se predstavljaju kao niz cifara (0-7) predvođen sa nulom (“0”).

Heksadecimalni celi brojevi

Heksadecimalni celi brojevi se predstavljaju kao niz cifara (0-9) i slova (a-f i A-F) predvođen sa nulom koju sledi slovo x (“0x” ili “0X”).

Primer prestavljanja celog broja deset (10) u tri brojna sistema:

```
• decimalnom: 10.
• oktalnom: 012.
• heksadecimalnom: 0xA.
```

Brojevi u pokretnom zarezu

Brojevi u pokretnom zarezu imaju sledeće delove: decimalni ceo broj, decimalnu tačku (“.”), deo iza decimalnog zareza (decimalni ceo broj), eksponent (“e” ili “E”, praćen decimalnim celim brojem).

Primeri brojeva u pokretnom zarezu:

```
• 1.1234
• .1E23
• -1.1E12
• 2E-10
```

Boolean

Promenljive tipa Boolean može da ima samo dve vrednosti: **true** (tačno) i **false** (netačno).

3.3 Operatori

3.3.1 Aritmetički operatori

Operator	Opis	Primer	Rezultat
+	Sabiranje	X=2 y=2 x+y	4
-	Oduzimanje	X=5 y=2 x-y	3
*	Mnozenje	X=5 y=4 x*y	20
/	Deljenje	15/5 5/2	3 2.5
%	Ostatak deljenja	5%2 10%8 10%2	1 2 0
++	Povecanje za 1	X=5 x++	X=6
--	Smanjenje za 1	X=5 x--	X=4

Tabela 2 Aritmetički operatori JavaScript-a

3.3.2 Operatori dodeljivanja

Operator	Primer	Objašnjenje
=	X=y	X=y
+=	X+=y	X=x+y
- =	x-=y	X=x-y
*=	X*=y	X=x*y
/=	x/=y	X=x/y
%=	X%=y	X=x%y

Tabela 3 Operatori dodeljivanja JavaScript-a

3.3.3 Operatori poredjenja

Operator	Opis
==	Jednako
!=	Nije jednako
>	Vece
<	Manje
>=	Vece ili jednako
<=	Manje ili jednako

Tabela 4 Operatori poredjenja JavaScript-a

3.3.4 Logički operatori

Operator	Opis
&&	AND
||	OR
!	NOT

Tabela 5 Logički operatori JavaScript-a

3.3.5 String operator

String je obično tekst (pod navodnicima). Spajanje dve ili više promenljive vrši se pomoću operatora *"+"*:

```
txt1 = „Kako je ”;
txt2 = „lep dan”;
txt3 = txt1 + txt2;
```

Promenljiva txt3 sada ima vrednost *"Kako je lep dan".*

Za razmak između dve promenjive dovoljno je staviti razmak između njiih ili napraviti razmak posle jedne tj. pre druge promenljive.

```
txt1 = „Kako je ”;
txt2 = „lep dan”;
txt3 = txt1 + „ „ + txt2;
```

3.3.6 Uslovni operator

Java Script ima i jedan uslovni operator koji dodeljuje vrednost promenljivoj na osnovu nekog uslova.Sintaksa je sledeća:

```
imePromenljive=(uslov) ?vrednost1:vrednost2
```

Dakle,ako je uslov zadovoljen, promenljivoj se dodeljuje vrednost1 inače vrednost2.

4 Kontrolne strukture i petlje

Naredbe i struktura JavaScripta veoma podsećaju na onu koja se koristi u jezicima Java, C++ i C.

JavaScript program je izgrađen iz funkcija, naredbi, operatora i izraza. Osnovna jedinica je naredba ili izraz koji se završava sa tačkom-zarezom.

```
document.writeln("Pocetak!<BR>");
```

Prethodna komanda poziva writeln() metod, koji je deo document objekta. Tačka-zarez ukazuje na kraj komande. JavaScript komanda se može prostirati u više redova. Slično, može se više naredbi naći u jednom redu, dokle god se završavaju tačkom-zarezom.

Možemo grupisati naredbe u blokove naredbi, izdvojene velikim zagradama:

```
{
document.writeln("Da li ovo radi? ");
document.writeln("Radi!<BR>");
}
```

Blokovi naredbi se koriste u definiciji funkcija i kontrolnim strukturama.

Jedna od osnovnih mogućnosti svakog programskog jezika je da pošalje tekst na izlaz. U JavaScriptu izlaz može biti preusmeren na nekoliko mesta uključujući trenutni prozor dokumenta i pop-up dijalog. Osnovni izlaz je preusmeravanje teksta u prozor WWW klijenta, što se obavlja prosleđivanjem HTML koda. Rezultujući tekst će biti interpretiran kao HTML kod i prikazan u prozoru WWW klijenta. To ostvarujemo sa metodima write (šalje tekst u prozor WWW čitača bez pomeranja) i writeln (isto kao write(), s tim što se posle ispisa teksta kurzor pomera u sledeći red) objekta document:

```
document.write("Test");
document.writeln('Test');
```

Vrste naredbi koje donose odluke i ponavljaju se u petlji se nazivaju kontrolne strukture. Važan deo komandne strukture je uslov. Svaki uslov je jedan logički izraz koji dobija vrednost *true* ili *false*.

4.1 Konstrukcija if i if...else.Switch

4.1.1 if

Najjednostavnija odluke u programu jeste praćenje neke grane ili putanje programa ako je ispunjen određen uslov. Sintaksa za ovu konstrukciju je:

```
If(uslov){
        Kod koji se izvrsava ako je vrednost izraza true
}
```

Sledi primer kako napisati if naredbu:

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
var vreme = d.getHours()
if (vreme < 10)
{
document.write("<b>Dobro jutro!</b>")
}
</script>
<p>
Ovo je primer If naredbe.
</p>
<p>
Ukoliko je na vašem računaru manje od 10 sati,
dobićete poruku: “ Dobro jutro!“.
</p>
</body>
</html>
```

4.1.2 If...else

Ako su umesto jedne grane potrebne dve ili više koje obrada treba da prati koristi se *if...else* tj. *If...else if...else* konstrukcija.

Sintaksa za konstrukciju if...else je:

```
If(uslov){
        kod koji se izvrsava ako je vrednost izraza true
    }
 else {
        kod koji se izvrsava ako je vrednost izraza false
 }
```

Sledi primer za konstrukciju if…else:

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
var vreme = d.getHours()
if (vreme < 10) {
document.write("<b> Dobro jutro!</b>")
} else {
document.write("<b>Dobar dan!</b>")
}
</script>

<p>
Ovaj primer ilustruje If ... else naredbu
```

```
</p>
<p>
Ukoliko je na vašem računaru manje od 10 sati,
dobićete poruku: “Dobro jutro!“.
U suprotnom, dobićete poruku: “Dobar dan!“.
</p>
</body>
</html>
```

4.1.3 if..else if...else

Konstrukcija *if..else if…else* pogodna je kada je potrebno pratiti nekoliko izvršnih linija.

Sintaksa:

```
If(uslov1) {
Kod koji se izvrsava ako je vrednost izraza  uslov1 true
}
else if (uslov2) {
Kod koji se izvrsava ako je vrednost izraza  uslov2 true
}
else{
kod koji se izvrsava ako ni jedan od izraza uslov1 I uslov2 nema vrednost true
}
```

Primer za konstrukciju if…else if…else:

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
var vreme = d.getHours()
if (vreme < 10) {
document.write("<b> Dobro jutro!</b>")
} else if (vreme < 18){
document.write("<b>Dobar dan!</b>")
} else {
document.write("<b>Dobro veče!</b>")
}
</script>
<p>
Ovaj primer ilustruje If ... else If ... else naredbu
</p>
</body>
</html>
```

4.1.4 Switch

Pod nekim okolnostima , odluka tipa true ili false nije dovoljna za obradu podataka u skriptu. Svojstvo objekta ili vrednost promenljive mogu sadržati bilo koju od nekoliko vrednosti i

potreban je poseban put izračunavanja za svaku od njih. U JavaScriptu postoji kontrolna struktura koju koriste mnogi jezici. Na početku strukture se identifikuje o kom izrazu se radi i svakoj putanji izvršavanja dodeljuje se oznaka koja odgovara određenoj vrednosti. U pitanju je switch naredba.

```
switch(n) {
case 1:
  izvrši blok1 koda
  break
case 2:
  izvrši blok2 koda
  break
default:
kod koji se izvršava ako je n različito od vrednosti datih u slučajevima 1 i 2
}
```

Sledeći primer pokazuje kako napisati switch naredbu:

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
dan=d.getDay()
switch (dan)
{
case 5:
  document.write("<b>Konačno petak!</b>")
  break
case 6:
  document.write("<b>Subota URAAA!</b>")
  break
case 0:
  document.write("<b>Nedelja odmor!</b>")
  break
default:
  document.write("<b>Kad će vikend?!</b>")
}
</script>

<p>U ovom primeru se generiše različita poruka u zavisnosti od dana u nedelji. Primetite da je nedelja=0, ponedeljak=1, utorak=2, itd.</p>

</body>
</html>
```

Naredba *default* obezbeđuje nastavak po putanji izvršavanja kada vrednost izraza ne odgovara ni jednoj oznaci naredbe case.

Naredba *break* koja služi za izlazak iz petlje, ovde ima značajnu ulogu. Naime, ako nije navedeno break posle svake grupe naredbi u case granama, izvršiće se sve naredbe iz svake case grane bez obzira na to da li je nađena odgovarajuća oznaka.

4.2 For petlja. Konstrukcije while i do..while

Često kada se piše kod, želi se da se neki deo koda ponovi više puta zaredom. Umesto dodavanja nekoliko gotovo identičnih linija koda u script-u, koriste se petlje da bi se postigao identičan rezultat.

U Java Scriptu postoje dve vrste petlji:

- ***for***-kada želimo da se deo koda izvrši tačno određen broj puta
- ***while***-kada želimo da se određen deo koda izvršava sve dok je određen uslov zadovoljen.

4.2.1 for (; ;)

Formalna sintaksa za for petlju je:

```
for (var=početna_vrednost;var<=krajnja_vrednost;var=var+uvećanje)
{
        kod koji se izvršava
}
```

Primer ispisuje tri nivoa naslova:

```
for(i=1;i<=3;i++){
document.write(“<H”+i+”> Naslov na nivou “ i “</H”+i+”>”)
}
```

Sledeći primer označava petlju koja počinje od vrednosti i=0. Petlja će se ponavljati sve dok **i** ima vrednost manju ili jednaku sa 10. Svaki put kada se petlja izvrši vrednost **i** će se povećati za 1.

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
document.write("Broj je: " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

4.2.2 for .. in

Pomoću nje možemo proći kroz sve osobine (*properties*) nekog objekta. Koliko jedan objekat ima osobina, toliko puta će se izvršiti ova *for* petlja.

```
for (prom in obj){
        naredbe
}
```

4.2.3 while (var<=endvalue)

Formalna sintaksa za while petlju je:

```
while (uslov)
{
   kod koji se izvršava
}
```

Ova petlja izvodi akciju sve dok izraz *uslov* ne dobije vrednost false.

4.2.4 do..while

JavaScript nudi još jednu konstrukciju petlje zvanu ***do..while***. Formalna sintaksa za ovu konstrikciju je sledeća:

```
do{
    naredbe
}
while(uslov)
```

Razlika između **while** i **do..while** petlje je ta što se u **do..while** petlji naredbe izvršavaju bar jednom pre nego što se uslov ispita, dok u petlji **while** to nije slučaj.

4.3 Break i continue

Break naredba se koristi da bi se iskočilo iz petlje.
Continue naredba se koristi da bi se iskočilo iz tekuće petlje i nastavilo sa narednom vrednošću.

4.3.1 Break

Naredba ***break*** će prekinuti petlju i nastaviti izvršenje od prve linije koda koja sledi nakon petlje.

Primer:

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

```
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){break}
document.write("Broj je: " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

4.3.2 continue

Nareba ***continue*** će prekinuti tekuću petlju i nastaviti sa sledećom vrednošću.

Primer:

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3){continue}
document.write("Broj je: " + i)
document.write("<br />")
}
</script>
</body>
</html>
```

5 Funkcije

Funkcija je deo koda koja se izvršava kada se dogodi neki događaj ili kada je funkcija pozvana.

Primer za poziv funkcije:

```
<html>
<head>
<script type="text/javascript">
function mojaFunkcija ()
{
alert("Zdravo!!!")
}
</script>
</head>
<body>
<form>
<input type="button"
onclick=" mojaFunkcija ()"
value="Pozovi funkciju">
</form>
<p>Pritiskom dugmeta pozvaće se funkcija. Funkcija će izbaciti alert prozor sa porukom</p>
</body>
</html>
```

Pimer funkcije sa argumentima - kako proslediti promenljivu funkciji i koristiti promenljivu u funkciji:

```
<html>
<head>
<script type="text/javascript">
function mojaFunkcija (text)
{
alert(text)
}
</script>
</head>
<body>
<form>
<input type="button"
onclick=" mojaFunkcija ('Zdravo!')"
value="Pozovi funkciju">
</form>
<p>Pritiskom dugmeta poziva se funkcija sa argumentom. Funkcija će izbaciti alert prozor sa ispisanim ovim argumentom.
```

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

Primer funkcije sa argumentima koja vraća vrednost:

```
<html>
<head>
<script type="text/javascript">
function mojaFunkcija()
{
	return ("Zdravo, kako si?")
}
</script>
</head>
<body>
<script type="text/javascript">
document.write(mojaFunkcija ())
</script>
<p>Skript unutar body taga poziva funkciju.</p>
<p>Funkcija vraća tekst.</p>
</body>
</html>
```

Da bi se sprečio čitač od izvršenja skripta kada se stranica učitava, skript se stavlja u funkciju. Funkcija sadrži kod koji se izvršava kada se dogodi određani događaj ili kada je ista pozvana. Može se pozvati iz bilo kog dela stranice. Funkcija može biti definisana i u “head” i u “body” delu dokumenta. Da bi bili sigurni da je funkcija pročitana ili učitana od strane browser-a pre nego što je pozvana preporučuje se da se stavi u “head“ deo.

Primer:

```
<html>
<head>
<script type="text/javascript">
function ispisiPoruku()
{
	alert("Zdravo, kako si?")
}
</script>
</head>
<body>
<form>
<input type="button" value="Pritisni me!" onclick="ispisiPoruku()" >
</form>
</body>
</html>
```

U primeru iznad, ako linija *alert("Zdravo, kako si?")* nije sastavni deo funkcije, izvršiće se odmah nakon što se ta linija učita. Script se ne izvršava pre nego što korisnik pritisne dugme. Dodat je jedan *onClick* događaj dugmetu koji će prikazati poruku kada se dugme pritisne.

5.1 Sintaksa za kreiranje funkcije

Sintaksa za kreiranje funkcije je:

```
function imefunkcije(var1,var2,...,varX)
{
Neki kod
}
```

Var1,var2,..,varX su promenljive ili vrednosti prosleđene funkciji. Zagrade { } definiše početak i kraj funkcije.

Funkcije bez parametara moraju imati zagrade { } posle naziva imena funkcije. Ne treba zaboraviti značaj velikih slova u JavaScript-u. Reč funkcija mora biti napisana malim slovima, u suprotnom će se desiti greška. Takođe, kada se poziva funkcija njeno ime mora biti navedeno identično kao kada se ona definiše.

Naredba *return* se koristi kada je potrebno odrediti vrednost koju funkcija vraća, i sve funkcije koje vraćaju neku vrednost moraju imati *return* naredbu.

```
function prod(a,b)
{
x=a*b
return x
}
```

Prethodna funkcija vraća proizvod dva broja. Da bi se pozvala navedena funkcija moraju se proslediti i dva parametra:

```
product=prod(2,3)
```

Rezltujuća vrednost ove prod() funkcije je 6, i ona će biti smeštena u promenljivu nazvanu proizvod.

6 Iskačući (popup) prozori

„Programi vole da pričaju sa korisnikom, postaljaju pitanja, daju odgovore i odgovaraju na poziv“. Metod predstavlja način na koji se radi nešto, a JavaScript metodi su akcione reči (action words, doers) JavaScript jezika. Metodi su pokretači dešavanja.

JavaScript koristi dijalog prozore (dialog boxes) za interkaciju sa korisnikom. Dijalog prozori se kreiraju pomoću tri metoda:

- *alert box* (upozorenje)
- *confirm box(*dijalog za potvrdu*)*
- *prompt box(*odzivni okvir za dijalog)

6.1 Alert

Videli smo da se write() i writeln() metode koriste za ispis teksta na veb strani. Drugi način za slanje izlaza čitaču je **alert() metoda**. Alert() metoda pravi mali, nezavisan prozor – nazvan dijalog prozor, prozor za dijalog, koji sadrži mali trougao sa znakom uzvika. Personalizovana poruka korisnika je smeštena posle i ispod trougla, kao i OK dugme. Kada se dijalog prozor pojavi, svako izvršavanje koda je stopirano dok korisnik ne pritisne OK dugme. Izled dijalog prozora može varirati od čitača do čitača, ali u svakom čitaču funkcioniše na isti način. Alert se koristi kada hoćemo da budemo sigurni da će informacija doći do korisnika.

Kada se alert prozor otvori korisnik treba da klikne “ok” za nastavak.

Sintaksa:

```
alert(”nekitekst”)
```

Primer:

```
<html>
<head>
<script type="text/javascript">
function prikazi_alert()
{
alert("Ja sam alert!!")
}
</script>
</head>
<body>

<input type="button" onclick="prikazi_alert()" value="Prikaži alert prozor" />
</body>
</html>
```

6.2 Confirm

Confirm se koristi ako želimo da korisnik proveri ili prihvati nešto. Znak pitanja će se pojaviti u prozoru zajedno sa OK i Cancel dugmetom. Ako korisnik pritisne OK dugme, vraća se vrednost *true*, a ako pak pritisne Cancel, vraća se vrednost *false.* Ova metoda koristi samo jedan argument, pitanje koje je postavljeno korisniku.

Sintaksa:

```
confirm("nekitekst")
```

Primer:

```
<html>
<head>
<script type="text/javascript">
function prikazi_confirm()
  {
  var r=confirm("Pritisnite dugme")
  if (r==true) {
    document.write("Pritisnuli ste OK!")
    }  else  {
    document.write("Pritisnuli ste Cancel!")
    }
  }
</script>
</head>
<body>
<input type="button" onclick="prikazi_confirm()" value="Prikaži confirm dijalog" />
</body>
</html>
```

Primer:

```
<html>
<head>
<title>Korišćenje confirm dijaloga </title>
</head>
<body>
<script language = "JavaScript">
document.clear   // Briše sadržaj strane
if(confirm("Da li ste zaista dobro?" == true){
alert("Onda možemo da nastavimo");
} else {
alert("Pokušaćemo kada Vam bude bolje");
}
</script>
</body>
</html>
```

6.3 Prompt

Budući da JavaScript ne obezbeđuje jednostavan metod za prihvatanje korisničkih podataka, za ove potrebe, koriste se HTML forme i prompt dialog prozori. Prompt dialog prozor se pojavljuje sa jednostavnim poljem za tekst (textfield). Nakon što korisnik u prompt dialog unese tekst, vrednost se vraća. Prompt dialog box koristi dva argumenta: string, tj. tekst koji obično predstavlja pitanje postavljeno korisniku opominjući korisnika da odgovori na pitanje i još jedan string koji obično predstavlja default odgovor koji je napisan u polju za unos. Ako korisnik pritisne OK dugme, ceo tekst iz dijalog prozora se vraća. Ako pritisne cancel prozor će mu vratiti *null*.

Sintaksa:

```
prompt(“sometext“,“defaultvalue“)
```

Primer:

```
<html>
<head>
<script type="text/javascript">
function prikazi_prompt()
  {
  var ime=prompt("Molim vas unesite ime","Harry Potter")
  if (ime != null && ime != "")
    {
    document.write("Zdravo" + ime + "! Kako si danas?")
    }
  }
</script>
</head>
<body>
<input type="button" onclick="prikazi_prompt()" value="Prikaži prompt prozor" />
</body>
</html>
```

7 JavaScript obrađivači događaja (event handlers)

Događaji su signali koji se generišu kada se odigra određena akcija. JavaScript može da detektuje te signale i mogu da se pišu programi koji reaguju na te događaje. Događaji u veb čitaču su kada korisnik klikne na hipertekst link, kada se promeni podatak u ulaznom polju forme ili kada se završi učitavanje dokumenta.

7.1 Sistemski događaji

Sistemski događaji ne zahtevaju nikakvu akciju korisnika da bi se aktivirali. Na primer, to je signal da se HTML dokument učitao, da će trenutni HTML dokument biti izbačen, ili da je protekao određeni period vremena.

OnLoad

Ovaj događaj se aktivira pošto se traženi HTML dokument (i svi njegovi delovi) kompletno učita. Dodeljen je elementima <BODY> i <FRAMESET>. Kada se radi sa okvirima, često je neophodno da budemo sigurni da su neki okviri učitani, pre nego što druge obnovimo ili promenimo (na primer, u slučaju kada jedan okvir sadrži formu, čije elemente referencira drugi okvir). S obzirom na prirodu Interneta, ne postoje garancije kada će se i u kom roku učitati neki dokument (opterećenje linka može biti veliko, ili isti može biti u prekidu), ali ako koristimo OnLoad, možemo biti sigurni da su svi ostali okviri učitani.

OnUnload

Događaj OnUnload je koristan za “čišćenje” posle posete nekoj prezentaciji. Tokom posete nekoj prezentaciji, korisniku se može desiti da ima otvoreno nekoliko prozora kao rezultat rada programa u JavaScriptu. Njih treba zatvoriti, a to se najlakše obavlja obrađivanjem događaja OnUnload.

7.2 Događaji uzrokovani akcijama miša ili tastature

Događaji uzrokovani akcijama miša zahtevaju akciju korisnika (rad sa mišem) da bi se aktivirali.

OnMouseOver

OnMouseOver događaj ne zahteva da se klikne mišem na neki objekat da bi se aktivirao. Kada je pointer miša iznad nekog objekta, događaj je okinut. OnMouseOver se može upotrebiti za objašnjenje linkova ili dugmadi.

OnMouseOut

OnMouseOut događaj ne zahteva da se klikne mišem na neki objekat da bi se aktivirao. Kada pointer miša izađe iz područja nekog objekta, događaj je okinut. OnMouseOut se često koristi u sprezi sa događajem OnMouseOver.

OnClick

Najčešće korišćeni događaj uzrokovan akcijom miša je OnClick. Ovaj događaj se aktivira uvek kada korisnik klikne na objekat koji prihvata takav događaj. Objekti koji prihvataju OnClick događaj su linkovi, polja za potvrdu i dugmad (uključujući submit, reset i radio buttons).

OnFocus

Događaj Focus se pojavljuje kada objekat postane predmet u fokusu, ili uprošćeno rečeno "dođe u centar pažnje". To se događa kada korisnik klikne mišem na određeni objekat, ili tamo dođe pritiskajući taster tab. Ako korisnik može da unese podatke u objekat (ili da promeni trenutni izbor za slučaj lista izbora), tada objekat ima fokus.

Obrađivač događaja OnFocus se koristi samo sa objektima **text, textarea, password** i **select**.

OnBlur

Događaj Blur se pojavljuje kada objekt nije više u fokusu. To se može desiti sa prebacivanjem u drugi prozor ili aplikaciju, ili tako što kliknemom mišem na drugi objekt, ili što pomerimo focus pritiskom na taster tab.

Obrađivač događaja OnBlur se koristi samo sa objektima **text, textarea, password** i **select**, baš kao i OnFocus.

OnChange

Događaj OnChange se aktivira uvek kada objekat izgubi fokus i ako se njegova vrednost promenila.

Obrađivač događaja OnChange se koristi samo sa objektima **text, textarea, password** i **select**.

OnSelect

Događaj OnSelect se odnosi samo na **text, textare**a i **password** objekte i izvršava se kada korisnik označi deo teksta u jednom od nabrojanih tipova objekata.

OnSubmit

Događaj OnSubmit je povezan sa objektom forme. Najčešće je potrebno da se podaci uneti u polja forme, provere ili iskoriste pre prosleđivanja sadržaja forme HTTP serveru. Na taj način, moguće je smanjiti opterećenje samog HTTP servera, pošto se provera greški vrši na strani klijenta.

OnSubmit omogućava da se zaustavi prosleđivanje sadržaja forme, ako je to neophodno. Prihvata povratnu informaciju kao parametar. Ako JavaScript funkcija vrati vrednost false, sadržaj forme neće biti prosleđen HTTP serveru.

8 Java Script objekti

Java Script je objektno orjentisan programski jezik. Objektno orjentisan jezik dozvoljava definisanje sopstvenih objekata i pravljenje sopstvenih tipova promenljivih. Da bi se pomoglo skriptovima da kontrolišu objekte, ali i autorima da pronađu neki sistem u mnoštvu objekata na stranici, definisan je model objekata dokumenata (Document object model, DOM). Model je nešto kao plan organizacije objekata na stranici. Svaki objekat ima svojstva i metode.

Objekti sadrže podatke, osobine (svojstva) objekta (eng. properties), koji su JavaScript promenljive. Osobine su vrednosti vezane za objekat. Njima se može pristupiti preko sledećeg modela:

```
NazivObjekta.NazivSvojstva
```

I u nazivu objekta i nazivu osobine objekta razlikujemo mala i velika slova. Osobinu objekta definišemo tako što joj dodelimo vrednost.

```
<script type="text/javascript">
        var txt="Zdravo, kako si?"
        document.write(txt.length)
</script>
```

Rezultat ovog koda je 16.

Metode su akcije koje se izvršavaju nad objektima. U sledećem primeru se koristi UpperCase metod String objekta da bi pokazali velika slova u tekstu:

```
<script type="text/javascript">
        var str="Zdravo, kako si?"
        document.write(str.toUpperCase())
</script>
```

Rezultat ovog koda je ZDRAVO, KAKO SI?

8.1 Objekat String

Objekat String se korist da bi se manipulisalo sačuvanim delom teksta.Sledeći primer pokazuje kako koristiti osobinu dužine da bi se našla dužina stringa:

```
<html>
   <body>
     <script type="text/javascript">
         var txt="Zdravo! Kako si?"
         document.write(txt.length)
     </script>
   </body>
</html>
```

8.1.1 Metode objekta string

Postoje dve vrste metoda objekta *String*: jedne se odnose na manipulaciju stringova kao što je pronalaženje pozicije u stringu, zamena jednog stringa drugim stringom, i drugi koji se odnose na formatiranje kao što je u HTML-u.

Metoda	Ekvivalentno formatiranje u HTML-u
String.anchor(Name)	String
String.big()	<big>String</big>
String.blink()	<blink>String</blink>
String.bold()	String
String.fixed()	<tt>String</tt>
String.fontcolor(color)	String
String.fontsize(size)	String
String.italics()	String
String.link(URL)	String
String.small()	String
String.strike()	<strike>String</strike>
String.sub()	_{String}
String.sup()	^{String}

Tabela 6 Metode objekta String koje se odnose na formatiranje

Metoda	Opis
charAt(index)	Vraća karakter koji se nalazi na specificiranoj poziciji
charCodeAt(index)	Vraća *Unicode* karakter ekvivalentan karakteru koji se nalazi na specificiranoj poziciji
concat(string1, ..., stringn)	Izvršava konkatenaciju stringova koji se nalaze kao argumenti
fromCharCode(codes)	Kreira string od niza karaktera odvojenih tačka zarezom
indexOf(substr, startpos)	Pronalazi prvo pojavljivanje *substr*-a počev od pozicije *startpos* i vraća njegov indeks
lastIndexOf(substr, startpos)	Pronalazi poslednje pojavljivanje *substr*-a počev od pozicije *startpos* i vraća njegov indeks
replace(searchValue, replaceValue)	Zamenjuje *searchValue* sa vrednošću *replaceValue*
search(regexp)	Pretražuje regularne izraze i vraća indeks tamo gde je nađen
slice(startpos, endpos)	Vraća string koji sadrži deo stringa između početne i krajnje pozicije
split(delimiter)	Od stringa pravi niz reči, pri čemu su elementi niza delovi stringa odvojeni znakom *delimiter*
substr(startpos, endpos)	Vraća deo stringa počev od pozicije *startpos* do pozicije *endpos* pri čemu ona nije uključena
toLocaleLowerCase()	Vraća kopiju stringa gde su sva slova prebačena u mala
toLocaleUpperCase()	Vraća kopiju stringa gde su sva slova prebačena u velika
toLowerCase()	Konvertuje sve karaktere stringa u mala slova
toString()	Vraća isti string kao što je početni
toUpperCase()	Konvertuje sve karaktere stringa u velika slova
valueOf	Vraća vrednost stringa datog objekta

Tabela 7 Metode stringa koje se odnose na manipulaciju stringova

Primer korišćenja metoda objekta *string*.

```
<html>
<head><title>Manipulacija stringova</title></head>
</head>
<body>
<script language="JavaScript">
 function break_tag(){
    document.write("<br>");
 }
  var str1 = new String("The merry, merry month of June...");
  document.write("U stringu:<em> "+ str1 );
  document.write("</em> prvo 'm' je na poziciji " +
                    str1.indexOf("m"));
  break_tag();
  document.write("Poslednje 'm' je na poziciji " +
  str1.lastIndexOf("m"));
  break_tag();
  document.write("<em>str1.substr(4,5)</em> vraca<em> " +
                    str1.substr(4,5));
  break_tag();
  document.write(str1.toUpperCase());
  break_tag();
  document.write("Ako funkcija indexOf ne nadje string vraca:  " + str1.indexOf("x"));
</script>
</body>
</html>
```

Kao rezultat izvršavanja gornjeg programa dobićemo da je prvo *m* na poziciji *4*, a poslednje na poziciji *17*, *str1.substr(4,5)* vraća *merry,* zatim kada primenimo funkciju *toUpperCase*() na *str1*, dobićemo string *THE MERRY, MERRY MONTH OF JUNE...,* a pošto se *x* ne nalazi u stringu *str1* izlaz funkcije *str1.indexOf("x")* je *-1.* Još jedan primer rada sa stringovima:

```
<html><head><title>Rad sa stringovima</title></head>
<body>
Rad sa stringovima
<script language="JavaScript">
   var straddr = "PeraPeric@testserver.org";
   document.write("<br>Ime je: <em> " + straddr.substr(0,4) + "</em>.<br>");
   var namesarr = straddr.split("@" );
   document.write( "Korisnicko ime je<em> " +  namesarr[0]     + "</em>.<br>");
   document.write( "Ime servera je<em> " +  namesarr[1] + "</em>.<br>");
   document.write( "Prvi karakter u stringu je <em>" + straddr.charAt(0)+ "</em>.<br>");
   document.write(     "a     poslednji     karakter     u     stringu     je     <em>"     +
straddr.charAt(straddr.length - 1)
   + "</em>.<br>");
 </script>
 </body></html>
```

Kao rezultat izvršenja prethodnog primera dobićemo sledeći izlaz:

```
Rad sa stringovimaIme je: Pera.
Korisnicko ime je PeraPeric.
Ime servera je testserver.org.
Prvi karakter u stringu je P. a poslednji karakter u stringu je g.
```

Primer korišćenja *search* i *replace* metode:

```
<html><head><title>Rad sa metodama Search i Replace</title>
</head>
<body>
  Search i Replace metode<br>
  <script language="JavaScript">
     var straddr = "PeraPeric@testserver.org";
     document.write( "Prvobitni string je "+ straddr + "<br>");
     document.write( "Novi string je "+ straddr.replace("Pera","Nikola")+"<br>");
     var index=straddr.search("test");
     document.write("search() metoda je pronasla 'test' na poziciji "+ index +"<br>");
     var mysubstr=straddr.substr(index,4);
     document.write("Posle zamene \"test\" sa \"proba\" <br>");
     document.write(straddr.replace(mysubstr,"proba") + "<br>");
</script>
</body></html>
```

Rezultat izvršavanja prethodnog programa je:

```
Search i Replace metode
Prvobitni string je PeraPeric@testserver.org
Novi string je NikolaPeric@testserver.org
search() metoda je pronasla 'test' na poziciji 10
Posle zamene "test" sa "proba" PeraPeric@probaserver.org
```

8.2 Objekat Date

Date objekti se koriste za rad sa datumom i vremenom. Postoji veliki broj metoda za dobijanje informacija vezanih za datum i vreme. Datum je zasnovan na UNIX-ovom datumu počev od 1.januara 1970. i ne podržava datume pre njega. S obzirom da se JavaScript izvršava u čitaču *Date* objekat vraća vreme i datum lokalnog računara a ne servera. Ako vreme nije dobro podešeno na korisničkom računaru onda se ni vrednosti neće dobro prikazivati.

Primeri kreiranja Date objekta:

```
var Date = new Date();// Konstruktor koji vraća Date objekat
var Date = new Date("July 4, 2004, 6:25:22");
var Date = new Date("July 4, 2004");
var Date = new Date(2004, 7, 4, 6, 25, 22);
var Date = new Date(2004, 7, 4);
var Date = new Date(Milliseconds);
```

8.2.1 Svojstva Date objekta

Objekat Date ima samo jedno svojstvo *prototype*.

8.2.2 Metode Date objekta

Date objekat ima veliki broj metoda. U tabeli su prikazane neke od metoda.

Metoda	Opis
getDate	Vraća dan u mesecu
getDay	Vraća dan u nedelji
getFullYear	Vraća godinu ispisanu sa sve 4 cifre
getHours	Vraća sat u danu
getMonth	Vraća broj meseca (Januar – 0)
getTime	Vraća broj milisekundi počev od 1.januara 1970.
parse()	Konvertuje string u date objekat
setDate(value)	Postavlja datum u mesecu
setFullYear()	Postavlja godinu
setHours()	Postavlja sat u danu
setTime()	Postavlja vreme od 1.januara 1970. u milisekundama
toString()	Vraća string objekat koji predstavlja datum i vreme

Tabela 8 Neke od metoda objekta Date

```
<html>
 <head><title>Datum i vreme</title></head>
 <body>
   <script language="JavaScript">
     var now = new Date();    // Kreiranje instance Date objekta
     document.write("<font size='+1'>");
     document.write("<b>Lokalno vreme:</b> " + now + "<br>");
     var hours=now.getHours();
     var minutes=now.getMinutes();
     var seconds=now.getSeconds();
     var year=now.getFullYear();
     document.write("Godina je " + year +"<br>");
     document.write("<b>Vreme je:</b> " + hours + ":" + minutes + ":" + seconds);
     document.write("</font>");
   </script>
</body></html>
```

Primer izvršenja gornjeg koda je:

Lokalno vreme: Mon Feb 9 23:00:31 UTC+0100 2009
Godina je 2009
Vreme je: 23:0:31

Vrednost lokalnog vremena će zavisiti od trenutka izvršavanja programa i biće prikazano vreme na klijentu.

8.2.3 Prilagođavanje Date objekta korišćenjem svojstva prototype

Date objekat ima svojstvo *prototype* koje omogućuje proširenje objekta. Prilagođavanja podrazumeva dodavanje novih metoda i svojstava koji će biti nasleđeni kad se kreira objekat ovog tipa. S obzirom da metoda *getMonth* vraća broj meseca počev od 0, umesto 1, možemo da napravimo našu metodu koji će raditi drugačije.

```
<html><head><title>The Prototype Property</title>
  <script language = "javascript">
    function monthCorrect(){
      var now = this.getMonth();
      return(++now);
    }
      Date.prototype.getMesec=monthCorrect;
  </script>
 </head>
 <body>
   <script language="JavaScript">
     var today=new Date();
     document.write("Mesec po redu: " + today.getMesec() + ".<br>");
 </script>
</body></html>
```

Na ovaj način smo dodali novu metodu objektu *Date* koji možemo da koristimo na isti način kao što koristimo metode koje su ugrađene. Sada ćemo korišćenjem ove metode kao rezultat za januar dobiti vrednost 1, za februar vrednost 2 itd.

Sledeći primer ilustruje kako da se pokaže sat na web strani:

```
<html>
<head>
<script type="text/javascript">
function vreme()
{
var danas=new Date()
var h=danas.getHours()
var m=danas.getMinutes()
var s=danas.getSeconds()
//dodaje nulu ispred brojeva ako su manji od 10
m=proveriVreme(m)
s= proveriVreme (s)
document.getElementById('txt').innerHTML=h+":"+m+":"+s
t=setTimeout('vreme()',500)
}

function proveriVreme (i)
{
if (i<10)
  {i="0" + i}
  return i
}
</script>
</head>
<body onload="vreme()">
<div id="txt"></div>
</body>
</html>
```

8.3 Objekat Math

JavaScript obezbeđuje mnoštvo matematičkih mogućnosti. Sve ove mogućnosti sadržane su u matematičkom objektu math. Ovaj objekt se razlikuje od drugih po tome što se za njegovu upotrebu ne pravi njegova kopija. Skriptovi direktno pozivaju svojstva i metode objekta math i on je deo reference.

round() metod zaokružuje broj na najbliži integer.

Sintaksa

```
Math.round(x)
```

Primer

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

```
    <script type="text/javascript">
      document.write(Math.round(0.60) + "<br />")
      document.write(Math.round(0.50) + "<br />")
      document.write(Math.round(0.49) + "<br />")
      document.write(Math.round(-4.40) + "<br />")
     document.write(Math.round(-4.60))
    </script>
  </body>
</html>
```

random() metod vraća nasumice izabran broj između 0 i 1.

Sintaksa

```
Math.random()
```

Primer

```
<html>
   <body>
     <script type="text/javascript">
       document.write(Math.random())
     </script>
   </body>
</html>
```

max() metod vraća broj sa najvećom vrednošću od dva specifičbna broja.

Sintaksa

```
Math.max(x,y)
```

Primer

```
<html>
    <body>
      <script type="text/javascript">
        document.write(Math.max(5,7) + "<br />")
       document.write(Math.max(-3,5) + "<br />")
      document.write(Math.max(-3,-5) + "<br />")
      document.write(Math.max(7.25,7.30))
    </script>
  /body>
</html>
```

min() metod vraća broj sa najmanjom vrednosti izmađu dva specifična broja.

Sintaksa

```
Math.min(x,y)
```

Primer

```
<html>
   <body>
      <script type="text/javascript">
         document.write(Math.min(5,7) + "<br />")
         document.write(Math.min(-3,5) + "<br />")
         document.write(Math.min(-3,-5) + "<br />")
         document.write(Math.min(7.25,7.30))
      </script>
   </body>
</html>
```

8.4 Objekat Array

Array objekat se koristi za čuvanje seta vrednosti u jednom imenu promenljive. Niz je kao i u drugim programskim jezicima uređena kolekcija podataka, međutim za razliku od drugih programskih jezika, JavaScript dozvoljava da se u jednom nizu nalaze podaci različitog tipa (zbog slabe tipiziranosti JavaScript-a). Svakom elementu niza dodeljuje se vrednost indeksa preko koga mu se može direktno pristupiti. Indeksiranje u nizu počinje od nule.

Sledeća linija koda definiše objekat array koji je nazvan *myArray*:

```
var myArray = new Array;
```

Ključna reč *new* se koristi da bi se dinamički kreirao objekat tipa *Array*. On poziva konstruktor objekta: *Array()* pri čemu se kreira novi objekat tipa Array. Veličina niza može biti prosleđena kao argument konstruktora.

```
var myArray = new Array(100);
```

U gornjem primeru je kreiran objekat tipa *Array* koji ima 100 elemenata.

Postoje dva načina za dodavanje vrednosti nizu:

- moguće je dodati koliko god je potrebno vrednosti za definisanje promenljivih koje su potrebne

```
var mojakola=new Array();
mojakola[0]="Saab";
mojakola[1]="Volvo";
mojakola[2]="BMW";
```

- moguće je dodati ceo sadržaj za kontrolisanje niza

```
var mojakola=new Array("Saab",  "Volvo", "BMW")
```

8.4.1 Pristup nizu

Moguće je obratiti se posebnom elementu u nizu, obraćajući se imenu niza i broju indeksa. Indeks počinje od 0.

```
document.write(mojakola[0]);
```

Kao rezultat ovog koda pojaviće se: "*Saab".*

8.4.2 Asocijativni nizovi

Asocijativni niz je niz koji umesto numeričkih vrednosti koristi stringove za indeksiranje vrednosti. Postoji *asocijacija* između indeksa i vrednosti sačuvane na toj lokaciji. Indeks se obično zove i *key* a dodeljena vrednost *value*. Ovi *key/value* parovi su uobičajen način za čuvanje i pristup podacima. U sledećem nizu *drzave* postoji asocijacija između vrednosti indeksa (skraćenice za državu) i sačuvane vrednosti (celog imena države). Za iteraciju kroz elemente ovakvog niza možemo da koristimo *for* petlju.

```
<html><head><title>Asocijativni nizovi</title></head>
<body>
<h2>Niz indeksiran stringovima</h2>
<script language="JavaScript">
var drzave = new Array();
drzave["SR"] = "Srbija";
drzave["GR"] = "Grcka";
drzave["MK"] = "Makedonija";
for( var i in drzave ){
     document.write("Indeks je: "+ i + „\t”);
     document.write("Vrednost je: " + drzave[i]+ "<br>");
   }
</script>
</body>
</html>
```

Rezultat ovog ovog JavaScripta je spisak skraćenica i celih naziva država.

8.4.3 Svojstva Array objekta

Objekat Array ima samo tri svojstva. Najčešće korišćeno svojstvo je *length* koji određuje broj elemenata niza, tj. njegovu dužinu.

Svojstvo	Opis
constructor	Referencira konstruktor objekta *Array*
length	Vraća broj elemenata niza
prototype	Proširuje definiciju niza dodajući mu svojstva i metode

Tabela 9 Svojstva objekta Array

Sledi primer u kom je prikazana primena svojstava *Array* objekta.

```
<html>
<head>
<title>Svojstva niza</title>
<h2>Svojstva niza</h2>
<script language="JavaScript">
var knjiga = new Array(6); //Kreira se objekat Array koji ima 6 elemenata
knjiga[0] = "Senka vetra"; //Dodeljivanje vrednosti elementima niza
knjiga[1] = "Igra anđela";
knjiga[2] = "Zovem se crveno";
knjiga[3] = "Beograd ispod Beograda";
knjiga[4] = "Avanture nevaljale devojcice";
knjiga[5] = "Kradljivica knjiga";
</script>
</head>
<body>
<script language="JavaScript">
        document.write("Niz knjiga ima "  + knjiga.length + " elemenata");
</script>
</body>
</html>
```

Rezultat ovog JavaScripta će biti "*Niz knjiga ima 6 elemenata*".

8.4.4 Metode *Array* objekta

Metoda	Opis
concat()	Dodaje elemente jednog niza drugom nizu
join()	Spaja elemente niza odvojene separatorom i formira string
pop()	Briše i vraća poslednji element niza
push()	Dodaje elemente na kraj niza
reverse()	Obrće raspored elemenata u nizu
shift()	Briše i vraća prvi element niza
slice()	Kreira novi niz od elemenata postojećeg niza
sort()	Sortira niz po abedecnom ili numeričkom redosledu
splice()	Uklanja i/ili zamenjuje elemente niza
toLocaleString()	Vraća niz predstavljen stringom u lokalnom formatu
toString()	Vraća niz predstavljen stringom
unshift()	Dodaje elemente na početak niza

Tabela 10 Metode objekta Array

8.4.4.1 Concat() metoda

Metoda *concat()* dodaje elemente prosleđene kao argumente u postojeći niz i vraća novoformirani niz.

```
noviNiz = stariNiz.concat(new elementi);
```

Primer upotrebe:

```
<html>
<head><title>Upotreba concat() metode</title></head>
<body>
<script language="JavaScript">
     var imena1=new Array("Ana", "Vlada", "Dragan" );
     var imena2=new Array("Maja", "Sandra");
     document.write("Prvi niz: "+ imena1 + "<br>");
     document.write("Drugi niz: "+ imena2 + "<br>");
     document.write("Posle koriscenja Concat metode <br>");
     imena1 = imena1.concat(imena2);
     document.write(imena1);
</script>
</body>
</html>
```

Prvi niz: Ana,Vlada,Dragan
Drugi niz: Maja,Sandra
Posle koriscenja Concat metode
Ana,Vlada,Dragan,Maja,Sandra

Slika 1 Rezultat izvršavanja prethodnog programa

8.4.4.2. Pop() metoda

Pop() metode briše poslednji element niza i kao rezultat izvršavanja vraća taj skinuti element niza.

```
var skinuta_vrednost = mojNiz.pop();
```

Sledi primer upotrebe:

```
<html>
<head><title>Primer upotrebe pop() metode</title>
</head>
<body>
<script language="JavaScript">
  var imena=new Array("Ana", "Vlada", "Dragan", "Maja");
  document.write("Originalni niz: "+ imena +"");
  var poslElement=imena.pop();  // Skida poslednji element niza
  document.write("<br>Skinut element: "+ poslElement);
  document.write("<br>Novi niz "+ imena);
 </script>
</body>
</html>
```

Kao rezultat izvršavanja ovog programa dobićemo da originalni niz ima elemente: *Ana, Vlada, Dragan, Maja,* da je skinut element *Maja,* a novi niz ima elemente *Ana, Vlada, Dragan*.

8.4.4.3. Push() metoda

Metoda *push()* dodaje nove elemente na kraj niza, pri čemu se alocira potrebna memorija.

```
mojNiz.push(new elementi);
```

Primer upotrebe metode *push()*:

```
<html>
<head><title>Primena push metode</title></head>
<body>
<script language="JavaScript">
  var imena=new Array("Ana", "Vlada", "Dragan", "Maja");
  document.write("Prvobitni niz: "+ imena + "<br>");
  imena.push("Sandra","Tanja");
  document.write("Novi niz: "+ imena);
</script>
</body>
</html>
```

Kao rezultat izvršavanja gornjeg primera dobićemo ispisane elemente prvobitnog niza: *Ana, Vlada, Dragan,* Maja i dobićemo ispisane elemente novog niza: *Ana, Vlada, Dragan, Maja, Sandra, Tanja*.

8.4.4.4 Metode shift() i unshift()

Metoda *shift()* uklanja prvi element niza i kao rezultat vraća uklonjenu vrednost, dok *unshift()* metoda dodaje element na početak niza. Ove metode su kao *pop()* i *push()* sem što se odnose na početak niza a ne na njegov kraj.

8.4.4.5 Metoda slice()

Metoda *slice()* kopira elemente jednog niza u drugi i ima dva argumenta: prvi je broj koji predstavlja početni element od koga će krenuti kopiranje, a drugi element predstavlja poslednji element (on se ne uključuje). Pri tome treba imati u vidu da je prvi element niza na poziciji " 0". Pritom originalni niz ostaje nepromenjen.

```
var noviNiz = mojNiz.slice(indeks prvog elementa, indeks poslednjeg elementa);
```

Primer upotrebe metode *slice()*:

```
<html>
<head><title>slice() metoda</title></head>
<body>
<script language="JavaScript">
  var imena=new Array("Ana", "Vlada", "Dragan", "Maja",   "Sandra");
  document.write("Pocetni niz: "+ imena + "<br>");
  var noviNiz=imena.slice(2, 4);
  document.write("Novi niz dobijen koriscenjem slice metoda: ");
  document.write(noviNiz);
</script>
</body>
</html>
```

Kao rezultat izvršavanja gornjeg programa dobićemo ispisane elemente prvog niza: *Ana, Vlada, Dragan, Maja, Sandra*, kao i elemente novog niza dobijenog korišćenjem *slice* metode: *Dragan, Maja.*

8.4.4.6 Metoda splice()

Metoda *splice()* uklanja određeni broj elemenata od neke startne pozicije i dozvoljava zamenu izbačenih elemenata novim.

```
mojNiz.splice(indeks elementa, broj elemenata, [elementi zamene]);
```

Primer upotrebe metode *splice()*:

```
<html>
<head><title>Primer splice metode</title></head>
<body>
<script language="JavaScript">
        // splice(starting_pos, number_to_delete, new_values)
    var imena=new Array("Ana","Vlada", "Dragan", "Maja");
    document.write("Pocetni niz: "+ imena + "<br>");
    imena.splice(1, 2, "Sandra","Tanja","Pera");
    document.write("Novi niz: "+ imena);
</script>
</body>
</html>
```

Kao rezultat izvršavanja gornjeg programa dobićemo ispisane elemente početnog niza: *Ana, Vlada, Dragan, Maja,* kao i ispisane elemente novog niza: *Ana, Sandra, Tanja, Pera, Maja*.

8.5 Wrapper objekat

Osnovni tipovi podataka imaju pridružene objekte koji nose isto ime kao tip podatka koji predstavljaju. Tako da svi osnovni tipovi podataka kao što su *string*, *number*, *boolean* imaju objekte koji se isto zovu: *String*, *Number*, *Boolean*. Ovi objekti se zovu „*omotači*“ i pružaju svojstva i metode koji mogu biti definisani za objekte. Na primer, *String* objekat ima veliki broj metoda koji omogućuju promenu fonta, boje, stila stringa, *Number* objekat ima metode koje dozvoljavaju formatiranje broja. Pogodnost postojanja *wrapper* objekata je mogućnost primenjivanja i proširivanja svojstava objekta koji utiču na osnovne objekte.

8.6 Kreiranje objekata

Kreiranje objekta se odvija u dva koraka:

1. Definisanje tipa objekta preko pisanja funkcije,
2. Kreirajući instancu objekta sa new.

Da bi definisali tip objekta moramo napisati funkciju koja određuje njegovo ime, njegove osobine i metode.

Na primer, pretpostavimo da želimo da kreiramo tip objekta za studente. Neka se objekat zove student i neka ima osobine ime, srednje_ime, prezime i indeks. Da bi to uradili potrebno je da napišemo sledeću funkciju:

```
function student(ime, srednje_ime, prezime, indeks) {
        this.ime=ime;
        this.ime_roditelja = srednje_ime;
        this.prezime=prezime;
        this.Indeks=indeks;
}
```

Sada možemo da kreiramo objekat student1 pozivajući funkciju student preko new:

```
student1=new student(“Pera”, “Petar”,”Perić”, ”100/99”);
```

Objektu možemo da dodamo i metode. Recimo, funkcija prikaziProfil() koja ispisuje sve podatke o studentu:

```
function prikaziProfil() {
document.write("Ime: " + this.ime + "<BR>");
document.write("Srednje ime: " + this.ime_roditelja + "<BR>");
document.write("Prezime: " + this.prezime + "<BR>");
document.write("Indeks: " + this.Indeks + "<BR>");
}
```

Sada ćemo ponovo napisati funkciju student() preko koje definišemo objekat.

```
function student(ime, srednje_ime, prezime, indeks) {
this.ime=ime;
this.ime_roditelja = srednje_ime;
this.prezime=prezime;
this.Indeks=indeks;
this.prikaziProfil=prikaziProfil;
}
```

Kada definišemo objekat student1, funkciji prikaziProfil() se pristupa preko:

```
student1.prikaziProfil();
```

Objekat može da ima svojstvo koje je i samo objekat. Na primer, objekat tipa student, može u sebe uključivati svojstva profesor i predmet, koja su i sama objekti. I tako redom.

Objektu Student možemo dodeliti svojstva pod nazivima ime, ime_roditelja, prezime i Indeks na sledeći način:

```
Student.ime=”Pera”;
Student.ime_roditelja=”Petar”;
Student.prezime=”Perić”;
Student.Indeks=”100/99”;
```

Ovo nije jedini način pristupa svojstvima objekta. Nizovi su skup vrednosti organizovan po brojevima kome je dodeljeno jedno ime promenljive. Osobine objekta i nizovi su u JavaScriptu direktno povezani, oni zapravo predstavljaju različit način pristupa istoj strukturi podataka. Tako, na primer, osobinama objekta Student možemo pristupiti i na sledeći način, potpuno ekvivalentan prethodnom primeru:

```
Student[”ime”]=”Pera”;
Student[”ime_roditelja”]=”Petar”;
Student[”prezime”]=”Perić”;
Student[”Indeks”]=”100/99”;
```

Takođe je moguće prići svakoj od osobina niza i preko indeksa. Tako je potpuno ekvivalentno gornjim načinima ekvivalentan i pristup preko indeksa:

```
Student[0]=”Pera”;
Student[1]=”Petar”;
Student[2]=”Perić”;
Student[3]=”100/99”;
```

Metoda je funkcija pridružena objektu. Programer definiše metod na isti način kao što definiše funkciju. Potom tu funkciju pridružuje objektu na sledeći način:

```
NazivObjekta.NazivMetode = NazivFunkcije
```

gde je NazivObjekta postojeći objekat, NazivMetode je naziv koji dodeljujemo metodi, a NazivFunkcije je naziv funkcije koju povezujemo sa objektom. Potom možemo pozivati metodu u kontekstu objekta kao:

```
NazivObjekta.NazivMetode(parametri);
```

JavaScript ima specijalnu rezervisanu reč, **this**, koja se koristi za referenciranje na trenutni objekt. Na primer, ako imamo funkciju Proveri() koja proverava da li se vrednost osobine objekta nalazi u propisanim granicama:

```
function Poveri(obj, donja, gornja){
if((obj.value < donja)||(obj.value > gornja))
alert(“Pogrešna vrednost”)
}
```

Tada možemo pozvati funkciju Proveri() u svakom elementu forme pomoću obrađivača događaja OnChange, koristeći this da prosledimo element forme, kao u sledećem primeru:

```
<INPUT TYPE=”text” NAME=”godine” SIZE=3 OnChange=”Proveri(this,18,77)”>
```

Svaka veb stranica poseduje objekte:

- **window:** objekat najvišeg nivoa; sadrži svojstva primenjiva na celokupan prozor,
- **location:** sadrži svojstva tekuće URL lokacije
- **history:** sadrži svojstva prethodno posećenih URL
- **document:** sadrži svojstva sadržaja tekućeg dokumenta, kao što su naziv (title), boja pozadine (bgcolor), forme

Primer svojstava:

```
location.href = "http://www.fon.bg.ac.yu/index.html" //lokacija dokumenta
document.title = "Probni dokument" //naziv dokumenta (title)
document.fgColor = #000000   //boja slova
document.bgColor = #FFFFFF    //boja podloge
history.length = 5   //koliko poslednjih dokumenta da "pamti" u history
```

Brauzer može kreirati objekte na osnovu sadržaja stranice, npr.:

```
document.mojaforma   //forma
document.mojaforma.Check1    //check polje na formi
document.mojaforma.Button1    //taster na formi
```

Oni mogu imati svojstva kao što su:

```
document.mojaforma.action = "http://www.fon.bg.ac.yu/index.html"
document.mojaforma.method = get
document.mojaforma.length = 10
document.mojaforma.Button1.name = DUGME1
document.mojaforma.text1.value = "sadržaj tekst polja"
document.mojaforma.text1.name = TekstPolje1
document.mojaforma.Check1.defaultChecked = true
document.mojaforma.Check1.value = on
itd.
```

Mnogi objekti imaju metode koje emuliraju događaje.

Npr, **button** objekt ima **click** metodu koja emulira klik miša na tasteru.

Događaj (Event)	Nastaje kada korisnik:	**Obrađivač događaja (Event Handler)**
blur	izađe iz fokusa elementa forme	**onBlur**
click	klikne na elementu forme ili link	**onClick**
change	podesi/promeni vrednost "text", "textarea" ili izabranog elementa	**onChange**
focus	uđe u fokus nekog elementa forme	**onFocus**
load	učita stranicu u browser	**onLoad**
mouseover	pomera pokazivač miša preko linka ili "anchora"	**onMouseOver**
select	izabere "input" polje elementa forme	**onSelect**
submit	izvrši "submit" (slanje) forme	**onSubmit**
unload	"napusti" stranicu	**onUnload**

Tabela 11 Događaji objekta button

8.6.1 Prosleđivanje objekata funkcijama

Parametri jedne funkcije nisu ograničeni samo na nizove karaktera i brojeve. Mogu se prosleđivati čitavi objekti.

U sledećem primeru može se videti

- upotreba funkcije kojoj se kao parametar prenosi čitav objekat,
- koja koristi for.. in petlju da bi prebrojala sva svojstva objekta i
- koja vraća kao rezultat niz karaktera u kome su navedena sva svojstva i njihove vrednosti.
- Rezultat rada funkcije se vraća programu naredbom return čiji je parametar vrednost koja se vraća programu.

```
function vidi_svojstva(objekat, ime_objekta){
var i, rezultat = ""
for (i in objekat)
```

```
rezultat += ime_objekta + ”. ” + i + ” = ” + objekat[i] + ”\n”
return rezultat
}
```

Tako da ako funkciju pozovemo sa

```
vidi_svojstva(Avion, ”Avion”),
```

gde je Avion objekat koji ima sledeća svojstva: proizvodjac, model i brzina (i oni imaju vrednosti ”Boing”, ”747” i 950) kao rezultat ćemo dobiti:

```
Avion.proizvodjac = Boing
Avion.model = YUGO 55
Avion.brzina = 950
```

9 Pravljenje Kuki (Cookie) fajlova u JavaScript - u

HTTP (HyperText Transfer Protocol) je napravljen kao protokol bez stanja (stateless). “Stateless” znači da nakon završene transakcije između servera i čitača, veza između istih se gubi i ni server, ni brauzer nemaju podatke o izvršenoj transakciji. Jednostavno, sadržaj komunikacije je izbrisan.

Sa razvojem Interneta i e-poslovanja, javila se potreba da podaci o online kupcima budu sačuvani. Informacije kao npr. podaci o prethodnim narudžbinama kupaca, preferencije kupaca, Id kupaca, registracioni brojevi kupaca su morale biti negde uskladištene,a priroda HTTP protokola to nije omogućavala.

Netscape je još 1994. godine rešio taj problem tako što je uveo kuki fajlove. Kuki je lokalni fajl koji se koristi za skladištenje informacija na loklanom računaru. Kuki je trajan, u smislu, da se ne briše pri resetovanju ili gašenju računara. Ovaj fajl čuva podatke o sesiji između browsera klijenta i drugih računara na mreži. Ideja o kuki fajlovima je postala veoma popularna i danas kukije podržava većina čitača.

Termin “cookie” potiče od starih programerskih trikova za debagovanje i testiranje programskih tokova. Tada se kuki zvao „magični kuki“ (magic cookie). Još tada je kuki sadržao tekst koga su delila dva programska toka, kako bi mogli da uspostave međusobnu komunikaciju. Naziv „cookie“ je u savremeno informatičkom društvo prvi put uveo Netscape. To je tekstualni podatak koji se čuva u fajlu koji se sreće pod različitim nazivima, najčešće se zove ***cookie jar*** i nalazi se na hard disku klijenta. HTTP server šalje kuki brauzeru prilikom prvog konektovanja čitača na server, i od tada, čitač vraća kopiju kuki fajla serveru svaki put kada čitač i server uspostave konekciju. Informacije iz kuki fajlova se šalju između čitača i servera preko HTTP zaglavlja (HTTP headers).

Kuki fajlovi mogu da personaliziju veb stranu i da učine da bude “user-friendly”. Oni čuvaju bitne informacije o jeziku koji govori korisnik, o npr. muzičkim preferencijama koje korisnik gaji, o broju poseta korisnika sajtu, prate robu koju su korisnici “stavili” u virtuelnu potrošačku korpu. Kuki može da sačuva još mnogo toga. Međutim, kukiji mogu biti i uznemirujući i “dosadni” i često se postavlja pitanje sigurnosti jer se nepoznati fajlovi snimaju na hard disk posetilaca sajta. Ako korisnici ne vole i ne žele kukije na svom računaru, kukiji mogu da se isključe. Slika 2. prikazuje kako se vrši isključivanje (disabling) kukija u čitaču, konkretno, Internet Explorer (IE).

Slika 2 Isključivanje cookie fajlova u čitaču - Internet Explorer (IE)

Kukije brauzeru najčšće šalje CGI (Common Gateway Interface) program na serverskoj strani preko HTTP zahteva (HTTP requests) i odgovarajućih HTTP zaglavlja (HTTP headers). Pomoću JavaScript-a, mogu se setovati kukiji u lokalnom čitaču, i na taj način se eliminiše potrebu da CGI program rukuje kukijima, a posledica je smanjenje angažovanja servera. Podrazumevani životni vek vek cookie fajlova je dužina tekuće sesije.

Kukiji se sastoje od teksta u formi parova ključ/vrednost (key/value), kojima je nadimak „zrnca“ (crumbs). Do 20 ovakvih parova možete uskladištiti u jedan kuki string. Brauzer čuva samo jedan kuki po veb strani.

Kada se prave kukiji, zrnca se sastoje od parova ime/vrednost (name/value), koji se zovu atributi i svi parovi moraju biti razdvojeni tačkom zarez.

U okviru stringa, zarezi, tačke zarezi ili prazna mesta (blank spaces) nisu dozvoljeni. HTTP Set-Cookie zaglavlje ima sledeću sintaksu:

```
Set-Cookie: name=value; [expires=date];[path=putanja];[domain=ime_domena]; [secure];
```

Primer:

```
Set-Cookie: id="Bob";expires=Monday, 21-Oct-05 12:00:00
GMT;domain="bbb.com"; path="/"; secure;
```

9.1 Atributi Cookie fajlova

Kada setujete cookie fajl, važno je da razumete elemente od kojih se cookie sastoji. Cookie sadrži parove ime/vrednost, kao i skup opcionih atributa koji određuju datum do kada cookie traje (*expiration date*), domen (*domain*), putanju (*path*) i da li cookie mora da se pošalje

preko sigurnosnih komunikacionih kanala HTTPS (HyperText Transfer Protocol Secure). Svi navedeni atributi se dodeljuju kao stringovi.

9.1.1 Ime (name)

Cookie tekst se sastoji od imena cookie fajla i sačuvane vrednosti. To može biti ID sesije, korisničko ime, ili bilo šta drugo, što želite da sačuvate.

Sintaksa:

```
imeKukija=vrednost;
```

Primer:

```
id=456;
email=joe@abc.com;
name=Bob;
```

Nemojte mešati vrednost sa nazivom kukija. Ime kuki fajla je s leve strane znaka jednakosti, a tekst koji se čuva je s desne strane znaka. Vrednost koja se dodeljuje je tipa string. Ako se želi dodelljivanje više vrednosti jednom stringu, moraju se koristiti jedinstveni karakteri da bi se odvojile dodeljene vrednosti, kao npr. *Marko*Markovic*011.* U JavaScript-u se može koristiti ugrađena metoda ***escape()*** koja vraća URL kodiran string koji je prihvatljiv čitačima.

9.1.2 Datum isteka kukija (Expiration date)

Cookie fajl obično ističe kada se tekuća sesija čitača završi. Međutim, možete sami definisati datum isteka cookie fajla:

Sintaksa:

```
;expires=dan_u_nedelji, DD-MON-YY  HH:MM::SS GMT
```

Primer:

```
;expires= Friday, 15-Mar-04 12:00:00 GMT
```

9.1.3 Ime domena (Domain Name)

Ime domena, koje se obično ne koristi, pokazuje domen na koji se cookie odnosi. Ime domena dozvoljava da se cookie fajl deli više servera, a ne da je pristup cookie – ju ograničen na samo jedan server.

Sintaksa:

```
; domain=ime_domena
; domain=http://neki_domen.com
```

Primer

```
; domain=myelab.net
; domain=http://myelab.net
```

9.1.4 Putanja (Path)

Putanja pokazuje gde je cookie važeći za određeni server. Setovanjem cookie putanje, omogućava se svim stranama sa istog domena da ga dele (share).

Sintaksa:

```
; path=putanja
```

Primer

```
; path=/home
```

9.1.5 Zaštićen (Secure)

Ako je cookie zaštićen (secure), on mora da se šalje kroz sigure komunikacione kanale HTTPS (Secure HyperText Transfer Protocol).

Sintaksa:

```
; secure
```

9.2 Kreiranje Cookie fajla u JavaScriptu

9.2.1 Objekat Cookie

JavaScript čuva cookie kao objekat koji služi i za čitanje i za pisanje. Cookie – ji se prave tako što se dodeljuju atributi u svojstvu (property) cookie fajla.Ako postoje cookie fajlovi kada aktivirate browser, onda se oni odnose na tekući objekat, tj. dokument. Cookie svojstva sadrže string od parova ime/vrednost (name/value) koji predstavljaju imena svih cookie –ja i korespondirajućih vrednosti, kao npr. ID broj sesije ili ID korisnika. Svi drugi atributi setovani za cookie, kao npr. datum isteka, putanja, i zaštita nisu vidljivi.

9.2.2 Dodeljivanje Cookie atributa

Da biste napravili cookie, morate da dodelite parove ime=vrednost (name=value) za *document.cookie.property.* Budite oprezni sa navodnicima, budite sigurni da promenljive ne stavljate pod znake navoda, ali da tekst koji ide uz promenljive obavezno bude pod navodnicima.

Sintaksa:

```
ime=vrednost;[expires=date];[path=putanja];[domain=negde.com];[secure]
```

Primer:

```
document.cookie="id=" + form1.cookie.value ";expires=" + expiration_date+";path=/";
```

9.2.3 Escape() i unescape() ugrađene funkcije

Važno je znati da kada se dodeljuje string tipa *ime* = *vrednost* (name=value) kuki atributu, ne smeju se koristiti prazna mesta, tačka zarez ili zarez.

Funkcija escape() će kodirati string objekat kroz konvertovanje svih karaktera koji nisu alfanumerički u njihov heksadecimalni ekvivalent, sa znakom % ispred.

Primer:

```
%20 predstavlja space, a
%26 predstavlja ampersand.
```

Da bi se informacije slale između servera i čitača, brauzer kodira podatke. Ovaj postupak se zove URI kodiranje. Pošto Internet brauzeri rukuju kukijima, kuki stringovi mogu biti kodirani pomoću JavaScript ugrađene funkcije ***escape()***.

Funkcija ***unescape()*** konvertuje URI-kodiran string „nazad“ u svoj originalan format. Funkcije ***encodeURI()*** i ***decodeURI()*** su najnovije verzije escape() i unescape() funkcija.

Primer escape() i unescape() funkcija:

```
<html><head><title>The escape() Method</title>
   </head><center><h2>URL Encoding </h2>
   <script language="JavaScript">
     function seeEncoding(form){
          var myString = form.input.value;
         alert(escape(myString));
      }
     function seeDecoding(form){
          var myString = form.input.value;
         alert(unescape(myString));
      }
   </script>
   <body>
   <form name="form1">
       Type in a string of text:
      <p>
      <input type="text" name="input" size=40>
      <p>
      <input type="button"
          value="See encoding"
         onClick="seeEncoding(this.form);">
      <p>
```

```
      <input type="button"
         value="See decoding"
        onClick="seeDecoding(this.form);">
      <p>
   </form>
   </center>
   </body>
   </html>
```

Objašnjenje:

Funkcija nazvana seeEncoding() je definisana. Ova funkcija uzima referencu na formu kao svoj jedini parametar.

Ugrađena funcija escape() služi za URI kodiranje stringa koji je unet kao input od strane korisnika.

Funkcija nazvana seeDecoding() je definisana. Ona uzima referencu na formu kao svoj jedini parametar.

Ugrađena funkcija unescape() se koristi za konvertovanje URI kodiranog stringa “nazad” u svoj originalan ASCII format.

Kada korisnik klikne na prvo dugme (See Encoding), onClick događaj se aktivira i kodiran string će se pojaviti u alert dijalog prozoru.

Kada korisnik klikne na drugo dugme (See Decoding), onClick događaj se aktivira i u alert dijalog prozoru se prikazuje string u svom prvobitnom ASCII formatu.

9.2.4 Primer – pravljenje cookie fajlova

```
<html><head><title>Pravljenje kukija</title>
<script language="JavaScript">
  function napraviKuki(forma){
   var kada = new Date();
   kada.setMinutes(kada.getMinutes() + 1); // kuki istice za 1 minut
   imeGosta=forma.ime_gosta.value;
   document.cookie="ime="+escape(imeGosta)+";expires="+kada.toGMTString();
   alert(unescape(document.cookie));
   }

  function dobrodosli(forma){
   var pozicija=document.cookie.indexOf("ime=");
   if ( pozicija != -1){
// ako je nasao podstring (-1 je vrednost ako nije nasao podstring)
   var pocetak = pozicija + 4;
   var kraj=document.cookie.indexOf(";",pocetak);
//ako je ubacen neki od atributa kukija, odvojen je sa ; pa ce kraj naci
//poziciju od ;
   if(kraj == -1){ kraj=document.cookie.length;}
// -1 znaci da nije nadjen ; onda pretpostavlja da je duzina do kraja
//kukija
    gost= unescape(document.cookie.substring(pocetak,kraj));
    alert("Dobrodosli " + gost);
   } else { alert("Danas nema kukija (kolacica :) )");}
```

```
  }

</script>
</head>
<body onLoad="document.mojaForma.reset()" >
    <center>
    <form name="mojaForma">
        Kako se zovete?
        <br>
       <input type="text" name="ime_gosta" >
        <p>
      <input type="button" value="Make cookie"
            onClick="napraviKuki(this.form);">
            <p>
      <input type="button"
            value="Get Cookie" onClick="dobrodosli(this.form);">
        <p>
    </form>
</body>
</html>
```

Objašnjenje

Funkcija nazvana *napraviKuki()* je definisana. Ona uzima referencu na formu kao svoj jedini parametar. Ovo je funkcija koja pravi kuki.

Novi Date objekat je kreiran i dodeljen promenljivoj nazvanoj *kada*.

Date objekat formira datum, koji se odnosi na godinu dana od tekućeg datuma. To će biti datum isteka kukija.

Kuki je napravljen. Njegove ime je “ime” i vrednost je korisnikčko ime iz polja za unos *ime1*, sačuvano u promenljivu *ime2*. Datum isteka kukija je postavljen na godinu dana od tekućeg datuma i konvertovan je u GMT vreme, što je zahtevani format za “expire” atribut.

Funkcija nazvana *dobrodosli()* je kreirana. Ona uzima referencu na formu kao svoj jedini parametar. Svrha ove funkcije je da pozdravi korisnika na osnovu vrednosti kukija.

Početna indeksirana pozicija je postavljena na mestu gde string „ime=“ počinje u kuki stringu. U ovom primeru, to će biti pozicija 4.

Nakon odsecanja stringa (substring) vrednosni deo kukija će unescape() funkcija konvertovati URI-kodiran string nazad u svoj prvobitni ASCII format.

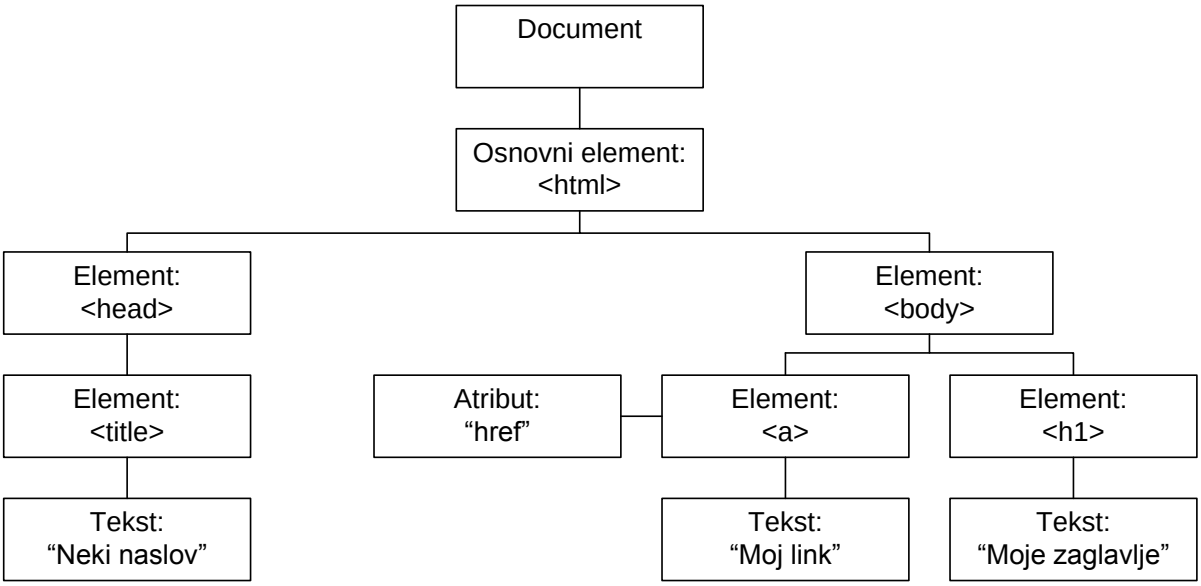
Korisnik je pozdravljen na osnovu vrednosti dobijenih iz kukija.

Korisnik ukucava svoje ime u polje za unos teksta .

Kada korisnik klikne na dugme *Napravi kuki*, onClick događaj se aktivira i kuki se pravi.

10 HTML DOM

HTML model objekata dokumenta (HTML DOM) definiše standardni način kako pristupiti stranici i obraditi HTML dokumente. Dom je HTML dokument u strukturi drveta koji se sastoji od elemenata, atributa i teksta.



Slika 3 Primer strukture HTML dokumenta

10.1 Šta je DOM?

JavaScript omogućava da se ceo HTML dokument restruktuira. Moguće je dodati, ukloniti, promeniti ili ponovo pokrenuti elemente na stranici.

Za promenu bilo čega na stranici, JavaScript može pristupiti svim elementima koji se nalaze u okviru HTML dokumenta. Ovaj pristup sa svim metodama i mogućnostima je dat kroz DOM.

Godine 1998. W3C je izdao prvi nivo DOM specifikacije. Ova specifikacija dozvoljava pristup i manipulaciju svakim posebnim elementom na HTML stranici. Svi brauzeri poštuju ovu preporuku, pa su zbog toga nestali i svi problemi koji prate DOM. JavaScript koristi DOM za čitanje i menjanje HTML, XHTML i XML dokumenata.

DOM je podeljen u različite nivoe:

- Core DOM - definiše standardni set objekata za svaki strukturni dokument
- XML - definiše standardni set objekata za XML dokumente
- HTML DOM - definiše standardni set objekata za HTML dokumente

10.2 HTML DOM ČVOROVI (NODOVI)

U skladu sa DOM-om, sve u HTML dokumentu je čvor.

Prema DOM-u:

- Ceo dokument je dokument čvor
- Svaki HTML tag je element čvor
- Tekstovi sadržani u HTML elementima su tekst čvorovi
- Svaki HTML atribut je atribut čvor
- Komentari su komentar čvorovi

Svi čvorovi u HTML dokumentu su u formi drveta. Svi čvorovi na drvetu čvorova su povezani jedan sa drugim. Sledeći primer to pokazuje:

```
<html>
 <head>
   <title>DOM Tutorijal</title>
 </head>
 <body>
  <h1>DOM - Prva lekcija</h1>
  <p>Zdravo, kako si?</p>
 </body>
</html>
```

Svaki čvor izuzev dokument čvora ima **roditeljski** čvor. Na primer roditeljski čvor od <head> i <body> čvorova je <html> čvor, a od "Zdravo, kako si?" tekst čvora je <p> čvor.

Većina čvorova ima svoje **potomak** čvorove. Na primer <head> čvor ima jedan potomak čvor <title> čvor. <Title> čvor takođe ima jedan potomak čvor "Dom Tutorijal" koji je tekst čvor.

Čvorovi su deca istih roditelja (**braća**) kada dele roditeljske čvorove. Na primer <h1> i <p> čvorovi deca istih roditelja jer je njihov roditeljski čvor <body> čvor.

Čvorovi takođe imaju potomke. **Potomci** su svi čvorovi koji su deca čvorova ili potomci dece, i tako dalje.

Čvorovi takođe imaju pretke. **Preci** su čvorovi koji su roditelji čvorova ili su roditelji roditelja, i tako dalje.

Kako pristupiti čvorovima?

Pomoću DOM-a je moguće pristupiti svakom čvoru u HTML dokumentu. Moguće je pronaći element kojim se može manipulisati na nekoliko načina:

- koristeći getElementById() i getElementyByTagName() metode
- koristeći perentNode, firstChild i lastChild svojstva element čvora

Pomoću ove dve metode moguće je pronaći svaki HTML element u dokumentu. Ove metode zanemaruju strukturu dokumenta. Ako želimo da nađemo sve <p> elemente u dokumentu, getElementById() metoda ih sve može naći, bez obzira na to kojeg su nivoa <p> elementi. getElementByTagName vraća tačan element, bilo gde da je sakriven u strukturi dokumenta.

getElementById() sintaksa

```
document.getElementById("nekiID");
```

GetElementByTagName() vraća sve elemente sa specifičnim imenom taga koji su potomci elemenata koji se koriste pri ovoj metodi.

getElementByTagName() sintaksa

```
document.getElementsByTagName("ime_taga");
```

Primer 1

Sledeći primer vraća listu čvorova svih <p> elemenata u dokumentu:

```
document.getElementsByTagName("p");
```

Primer 2

Sledeći primer vraća listu čvorova svih <p> elemenata koji su potomci elemenata sa id=”maindiv”:

```
document.getElementById('glavnidiv').getElementsByTagName("p");
```

Node list (lista čvorova)

Kada se radi sa čvorom list, obično se stavlja lista u varijablu na sledeći način:

```
var x=document.getElementsByTagName("p");
```

Sada promenljiva x sadrži listu svih <p> elemenata na stranici i moguće je pristupiti svim elementima preko njihovog indeksa. Indeks počinje od 0.

parentNode, firstChild i lastChild

Ova tri svojstva prate strukturu dokumenta i dozvoljavaju kratku razdaljinu kretanja u dokumentu:

```
<table>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>Alaska</td>
  </tr>
</table>
```

U prethodnom HTML kodu, prvo <td> je firstChild od <tr> elementa, a poslednje <td> je lastChild od <tr> elementa. Šta više <tr> je parentNode svakog <td> elementa.

Najjednostavnije korišćenje firstChild svojstva je pristup tekstu elementa:

```
var x=[prethodni paragraf];
var text=x.firstChild.nodeValue;
```

Osobina parentNode je često korišćena za promenu strukture dokumenta. Moguće je ukloniti node sa id=”glavnidiv” iz dokumenta. Prvo treba pronaći čvor sa zadatim id-em, zatim pomeriti se do njegovog roditeljskog čvora i na kraju izvršiti removeChild metod.

```
var x=document.getElementById("glavnidiv");
x.parentNode.removeChild(x);
```

Root Nodes (koreni čvorovi)

Postoje dva posebna svojstva dokumenta koje dozvoljavaju pristup tagovima:

- document.documentElement
- document.body

Prvo svojstvo vraća ime čvora i postoji u celom XML i HTML dokumentu. Drugo svojstvo je poseban dodatak za HTML stranice i daje direktan pristup do body taga.

10.3 Informacije o čvorovima

Svaki čvor ima svojstva koja sadrže informacije o čvorovima. Ta svojstva su sledeća:

- nodeName
- nodeValue
- nodeType

nodeName

nodeName svojstvo sadrži ime čvora.

- nodeName element čvora je naziv taga
- nodeName atribut čvora je naziv atributa
- nodeName tekst čvora je uvek #text
- nodeName document čvora je uvek #document

nodeValue

Kod tekst čvorova, ova osobina sadrži tekst.
Kod atribut čvorova, ova osobina sadrži vrednost atributa.
nodeValue osobna nije dostupna za dokument i element čvorove.

nodeType

Ova osobina vraća tip čvora.

Tip elementa	Tip čvora
Element	1
Atribut	2
Tekst	3
Komentar	8
Dokument	9

Tabela 12 Najvažniji tipovi čvorova i odgovarajuća oznaka

Sledeći primer pokazuje kako boja pozadine može biti promenjena u žutu kada korisnik klikne bilo gde na dokument:

```
<html>
<head>
<script type="text/javascript">
        function dajVrednost()
        {
        var x=document.getElementById("mojeZaglavlje")
        // uzima referencu na h1 tag (to je element čvor-nema value)
        // sa id-om mojeZaglavlje
        alert(x.innerHTML)
        // innerHTML
        }
</script>
</head>
<body>
        <h1 id="mojeZaglavlje" onclick="dajVrednost()">Ovo je zaglavlje</h1>
        <p>Kliknite na zaglavlje da dobijete vrednost</p>
</body>
</html>
```

10.4 HTML DOM OBJEKTI

Document	Opis
Anchor	Prikazuje ceo HTML dokument i koristi se za pristup svim elementima na stranici
Area	Prikazuje <a> element
Base	Prikazuje <area> element unutar image-map
Body	Prikazuje <base> element
Button	Prikazuje <body> element
Event	Prikazuje <button> element
Form	Prikazuje oblik događaja
Frame	Prikazuje <form> element
Frameset	Prikazuje <frame> element

Iframe	Prikazuje <frameset> element
Image	Prikazuje <iframe> element
Input button	Prikazuje <img> element
Input checkbox	Prikazuje dugme u HTML formi
Input file	Prikazuje checkbox u HTML formi
Input hidden	Prikazuje fileupload u HTML formi
Input password	Prikazuje osenčeno polje u HTML formi
Input radio	Prikazuje polje sa šifrom u HTML formi
Input reset	Prikazuje radio dugme u HTML formi
Input submit	Prikazuje dugme za resetovanje u HTML formi
Input text	Represents a submit button in an HTML form
Link	Prikazuje text-input polje u HTML formi
Meta	Prikazuje <link> element
Option	Prikazuje <meta> element
Select	Prikazuje <option> element
Style	Prikazuje izabranu listu u HTML formi
Table	Prikazuje individualnu style naredbu
TableData	Prikazuje <table> element
TableRow	Prikazuje <td> element
Textarea	Prikazuje <tr> element
Document	Prikazuje <textarea> element

Tabela 13 HTML Dom objekti

10.4.1 Objekat Window

Objekat prozor ili Window nalazi se na vrhu hijerarhije objekata JavaScript-a i predstavlja mesto za sadržaj HTML dokumenta u prozoru brauzera. Kako se sve akcije dokumenta odigravaju unutar prozora, prozor je na vrhu hijerarhije objekata JavaScript-a. Objekat Window se kreira automatski sa svakom instancom <body> ili <frameset> taga.

Osobine objekta Window

Osobina	Opis
closed	Vraća true ili false u zavisnosti od toga da li je prozor otvoren ili ne
defaultStatus	Postavlja ili vraća izostavljen tekst u Statusbaru
Document	Videti objekt document
History	Videti objekt history
Length	Postavlja ili vraća broj frejmova u prozoru
Location	Videti objekt location
Name	Postavlja ili vraća ime prozora
Opener	Vraća referencu na prozor koji je kreirao prozor
outerheight	Postavlja ili vraća spoljnju visinu prozora
pageXOffset	Postavlja ili vraća x-tu poziciju tekuće strane u odnosu na gornji levi ugao vidljive površine prozora.
pageYOffset	Postavlja ili vraća y poziciju tekuće strane u odnosu na gornji levi ugao vidljive površine prozora.
parent	Vraća prozor roditelj
personalbar	Reguliše da li će personalbar biti vidljiv ili ne
scrollbars	Reguliše da li će scrolbar-ovi biti vidljivi ili ne
Self	Vraća referencu trenutno aktivnog prozora
Status	Postavlja tekst u statusbar prozora
Statusbar	Reguliše da li će ili ne brauzerov statusbar biti vidljiv ili ne

toolbar	Reguliše da li će brauzerov toolbar biti vidljiv ili ne
Top	Vraća najdaljeg pretka konkretnog prozora

Tabela 14 Svojstva objekta Window

Metode objekta Window

Metoda	Opis
alert()	Pokazuje alert prozor sa porukom i dugmetom ok
blur()	Pomera fokus sa tekućeg prozora
clearInterval()	Ukida vreme isteka metodom
clearTimeout()	Ukida vreme isteka sa ovom
close()	Zatvara tekući prozor
confirm()	Pokazuje dijalog box sa porukom i dugmetom ok i dugmetom cancle (takozvani confirm box)
createPopup()	Kreira pop-up prozor
focus()	Postavlja fokus na tekući prozor
moveBy()	Pomera prozor relativno u odnosu na tekuću poziciju.
moveTo()	Pomera prozor na zadatu poziciju
open()	Otvara novi prozor browser
print()	Štampa sadržaj tekućeg prozora
prompt()	Prikazuje dijalog koji zahteva unos korisnika
resizeBy()	Smanjuje prozor za zadati broj piksela
resizeTo()	Smanjuje prozor na zadatu širinu i visinu
scrollBy()	Pomera sadržaj za zadati broj piksela
scrollTo()	Skroluje sadržaj na zadate kordinate

setInterval()	Evaluira izraz u zadatom vremenskom intervalu
setTimeout()	Evaluira izraz posle zadatog broja milisekundi

Tabela 15 Metode objekta Window

Osobine i metodi objekta prozor mogu se u skriptu referencirati na nekoliko načina. Najčešći način je da se u referencu uključi objekat prozora:

```
window.imeOsobine
window.imeMetoda
```

Kada pri referenciranju skript pokazuje na prozor u kome je smešten dokument, za objekat prozora postoji i sinonim self.

```
self.imeOsobine
self.imeMetoda
```

Nije uvek potrebno praviti novi prozor koristeći kod JavaScript-a.

Skript ne pravi glavni prozor brauzera, to čini korisnik aktiviranjem brauzera ili otvaranjem URL-a ili datoteke iz menija brauzera. Taj prozor je automatski valjan objekat Window, čak iako je prazan. Ali skript može generisati bilo koji broj podprozora ako je otvoren glavni prozor (i ako sadrži dokument čiji će skript otvoriti podprozor).

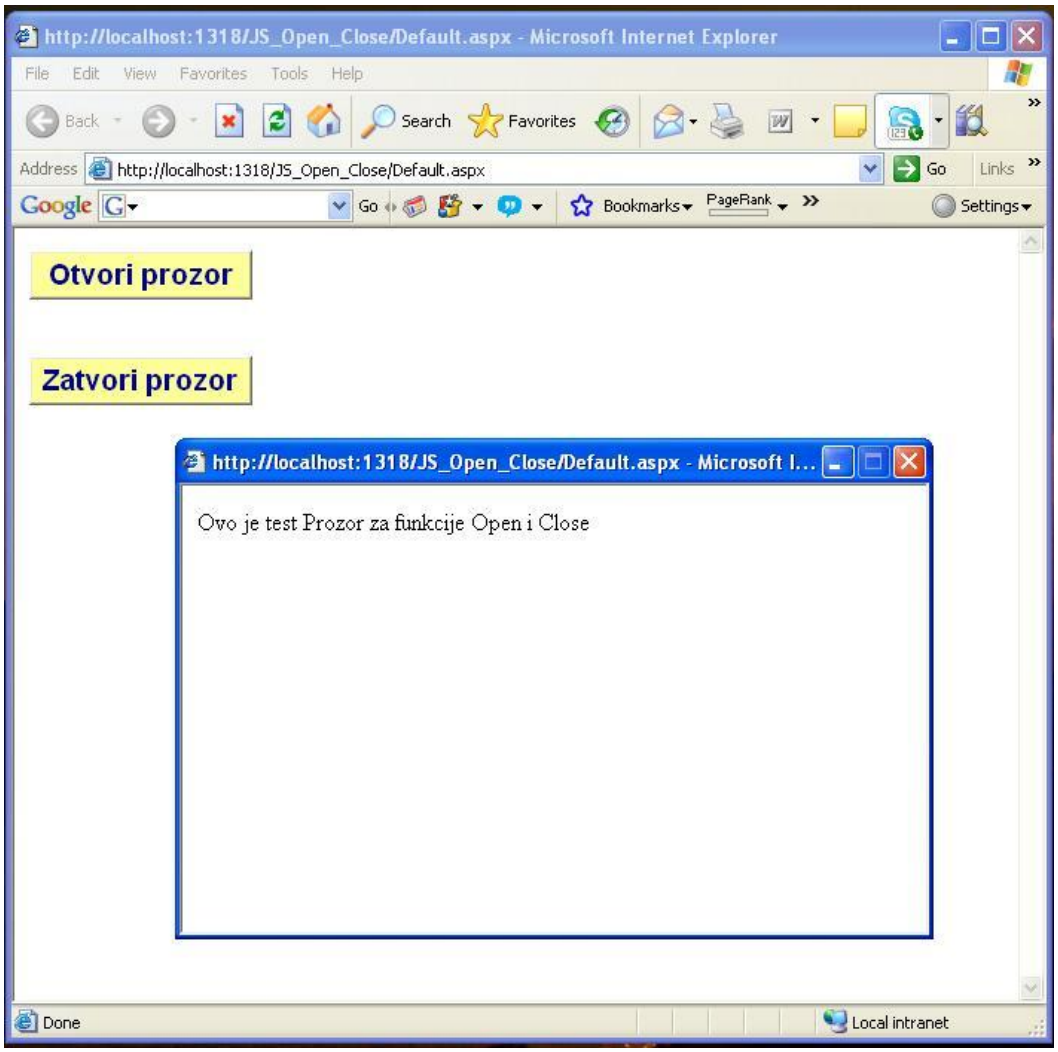
Metod koji pravi novi prozor je window.open() koja sadrži više parametara koji definišu karakteristike prozora.

window.open(URL,name,specs,replace)

U sledećem primeru se može videti kako skript definiše, otvara i zatvara manji prozor:

```
<html>
<body>
<form id="form1" runat="server">
<div>

<script type="text/javascript">
function zatvoriProzor()
    {
    mojProzor.close()
    }
</script>

<script type="text/javascript">
function otvoriProzor()
    {
    mojProzor=window.open('','','width=500,height=300')
    mojProzor.document.write("Ovo je test Prozor za funkcije Open i Close")
    }
</script>

<input type=button value="Otvori prozor" onclick="otvoriProzor()"
style="font-weight: bold; font-size: 14pt;
       width: 150px; color: #000099; background-color: #ffff99" />
```

```
        <br />
        <br />
        <br />
<input type="button" value="Zatvori prozor" onclick="zatvoriProzor()"
style="font-weight: bold;
font-size: 14pt; width: 150px; color: #000099; background-color: #ffff99"
/>

</div>
</form>
</body>
</html>
```

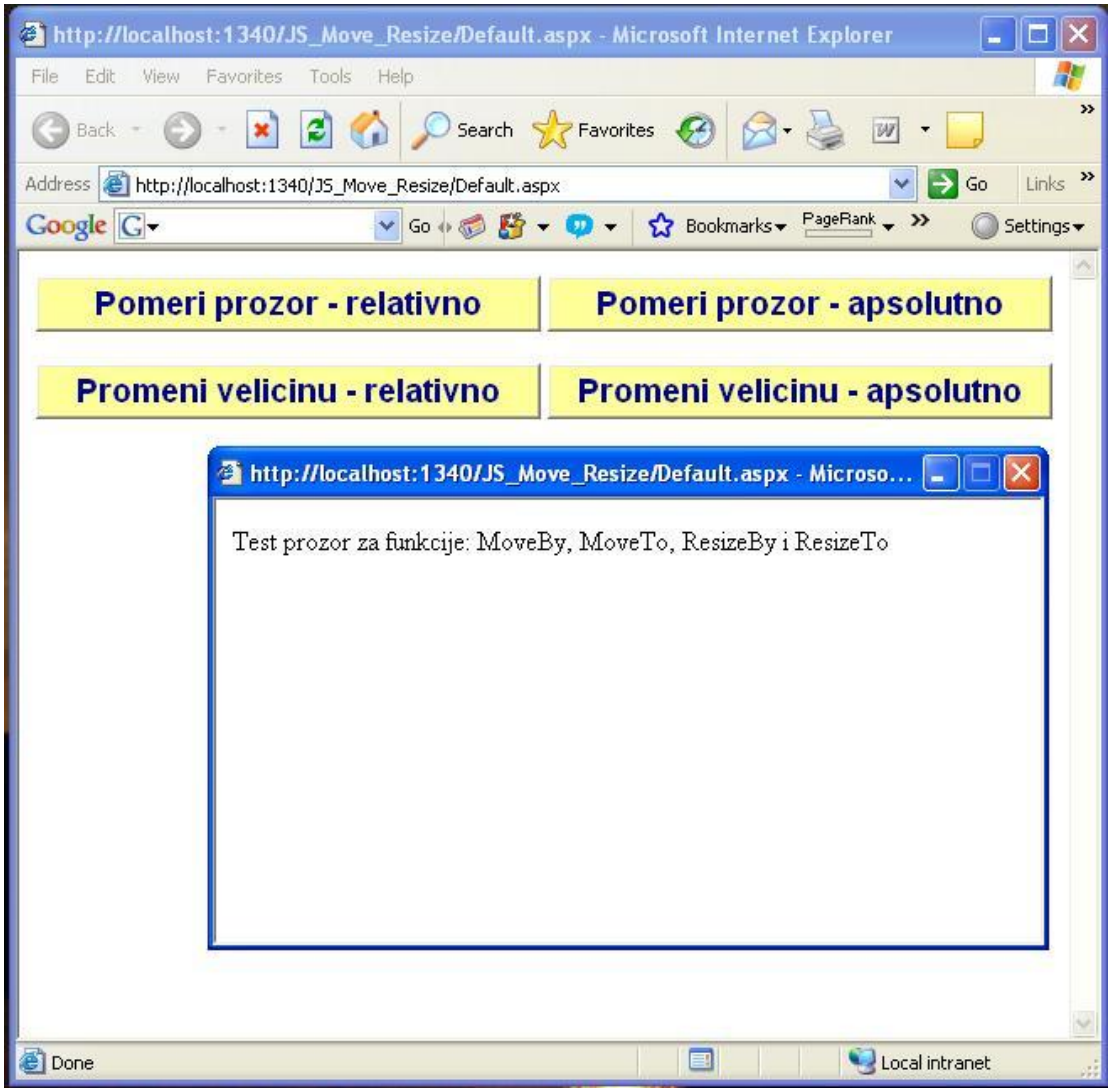


Slika 4 Rezultat reakcije na događaj otvori prozor

U sledećem primeru se može videti kako skript definiše, pomera i menja veličinu manjeg prozora:

```
<html>
<body>
<form id="form1" runat="server">
<div>

<script type="text/javascript">
```

```
mojProzor=window.open('','','width=200,height=100')
mojProzor.document.write("Test prozor za funkcije: MoveBy, MoveTo, ResizeBy
i ResizeTo")
</script>

<script type="text/javascript">
function pomeriProzorRelativno()
    {
    mojProzor.moveBy(50,50)
    }
</script>

<script type="text/javascript">
function pomeriProzorApsolutno()
  {
  mojProzor.moveTo(50,50)
  }
</script>

<script type="text/javascript">
function promeniVelicinuRelativno()
  {
  mojProzor.resizeBy(-10,-10)
  }
</script>
<script type="text/javascript">
function promeniVelicinuApsolutno()
  {
  mojProzor.resizeTo(500,300)
  }
</script>

<input type="button" value="Pomeri prozor - relativno" style="font-weight:
bold;
font-size: 14pt; width: 300px; color: #000099; background-color: #ffff99"
onclick="pomeriProzorRelativno()" />
<input type="button" value="Pomeri prozor - apsolutno" style="font-weight:
bold;
font-size: 14pt; width: 300px; color: #000099; background-color: #ffff99"
onclick="pomeriProzorApsolutno()" />
<br />
<input type="button" value="Promeni velicinu - relativno" style="font-
weight: bold;
font-size: 14pt; width: 300px; color: #000099; background-color: #ffff99"
onclick="promeniVelicinuRelativno()" />
<input type="button" value="Promeni velicinu - apsolutno" style="font-
weight: bold;
font-size: 14pt; width: 300px; color: #000099; background-color: #ffff99"
onclick="promeniVelicinuApsolutno()" />
</div>
</form>
</body>
</html>
```



Slika 5 Prikaz stranice sa demonstracijom pomeranja i promene veličine prozora

setInterval() metoda poziva funkciju ili evaluira izraz u zadatom intervalu (u milisekundama).

setInterval(code,millisec,["lang"])

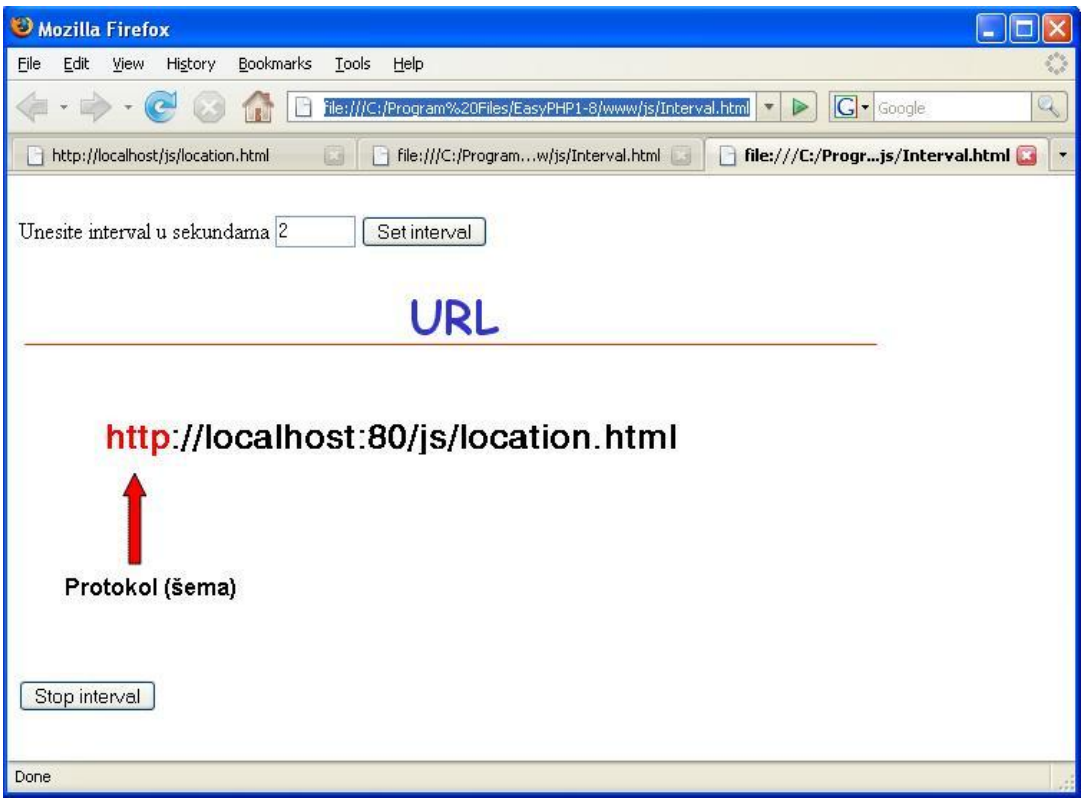
Parametar	Opis
code	Pokazuje na funkciju ili kod koji treba da se izvrši
millisec	Broj u milisekundama
lang	JScript | VBScript | JavaScript

Tabela 16 Parametri metode setInterval()

U našem primeru pritiskom na prvo dugme se poziva funkcija "set_interval()" koja poziva metodu setInterval() sa parametrima nazivom funkcije promeni_sliku() i unetim brojem sekundi pomnoženih sa 1000. Funkcija promeni_sliku() postavlja redom, u krug src parametar taga img sa id-em slika. Metoda window.clearInterval() briše postavljeni interval.

```
<html>
<head>
<script language=javascript>
j=document.getElementById("interval")*1000;
```

```
function postavi_interval()
{
      j=document.getElementById("interval")*1000;
}
</script>
</head>
<body onload="postavi_interval">
<br/>
Unesite interval u sekundama
<input type="text" id="interval" size="5" value="2" />
<button onClick="promeni_interval()">Postavi(promeni)
interval</button><br/>
<script language=javascript>
var int;
var i=0;
function promeni_interval()
{
var j=document.getElementById("interval").value*1000;
      int=window.setInterval("promeni_sliku()",j)
}
function promeni_sliku()
  {var s
  i++;i%=4; s=i+1
  document.getElementById("slika").src="0"+s+".jpg"
  }
</script>
<br/>
<img src="01.jpg" id="slika"/>
<br/>
<button onClick="int=window.clearInterval(int)">Poništi interval</button>
</body>
</html>
```



Slika 6 Prikaz sekvence izvršavanja metode "setInterval()“

10.4.2 Objekat Navigator

Navigator objekat je JavaScript objekat, ne HTML DOM objekat. Ovaj objekat je kreiran od strane JavaScript runtime mašine i sadrži informacije o brauzeru klijenta.

Osobine objekta Navigator

Osobina	Opis
appCodeName	Vraća kod imena brauzera.
appMinorVersion	Vraća minor verziju brauzera.
appName	Vraća ime brauzera.
appVersion	Vraća platformu i verziju brauzera.
browserLanguage	Vraća tekući jezik browsera.
cookieEnabled	Vraća logičku vrednost koja označava da li su kukuji dostupni u brauzeru.
cpuClass	Vraća CPU klasu brauzerovog sistema.
onLine	Vraća logičku vrednost koja označava da li je sistem u offline modu.
platform	Vraća platformu operativnog sistema.
systemLanguage	Vraća jezik koji koristi operativni sistem.
userAgent	Vraća vrednost useragent zaglavlja koji klijent šalje serveru.
userLanguage	Vraća prirodno podešavanje jezika operativnog sistema.

Tabela 17 Svojstva objekta Navigator

Metode objekta Navigator

Metoda	Opis
javaEnabled()	Označava da li brauzer podržava Javu.
taintEnabled()	Označava da li brauzer podržava data tainting.

Tabela 18 Metode objekta Navigator

10.4.3 Objekat Location

Location objekat je JavaScript objekat, ne HTML DOM objekat. Kreiran je od strane JavaScript runtime mašine i sadrži informacije o tekućem URL-u. Svaka veb stranica koja je prikazana u brauzeru, naziva se lokacija i ima svoj URL. Brauzer ovu lokaciju smešta u objekat location. Location objekat je deo Window objekata i dostupan je kroz window.location svojstva.

Osobine objekta Location

Svojstvo	Opis
hash	Postavlja ili vraća deo URL-a od znaka #
host	Postavlja ili vraća hostname i broj porta tekućeg URL-a
hostname	Postavlja uili vraća hostname tekućeg URL-a
href	Postavlja ili vraća ceo URL
pathname	Postavlja ili vraća putanju tekućeg URL-a
port	Postavlja ili vraća broj porta tekućeg URL-a
protocol	Postavlja ili vraća protokol tekućeg URL-a
search	Postavlja ili vraća URL od znaka pitanja

Tabela 19 Metode objekta Location

Metode objekta location

Metode	Opis
assign()	Učitava novi dokument
reload()	Oslobađa tekući dokument
replace()	Menja tekući dokument novim

Tabela 20 Metode objekta Location

Ponekad objekat predstavlja nešto što nema fizičku reprezentaciju, kao što je prozor i dugme. To je slučaj sa objektom lokacije. Ovaj objekat predstavlja URL koji je učitan u dokument. Za kretanje skriptova kroz druge stranice, treba zadati naredbe. Dakle, ako skript otvara novi prozor i dodeljuje njegovu referencu promenljivoj noviProzor, naredba koja učitava stranicu u prozor biće:

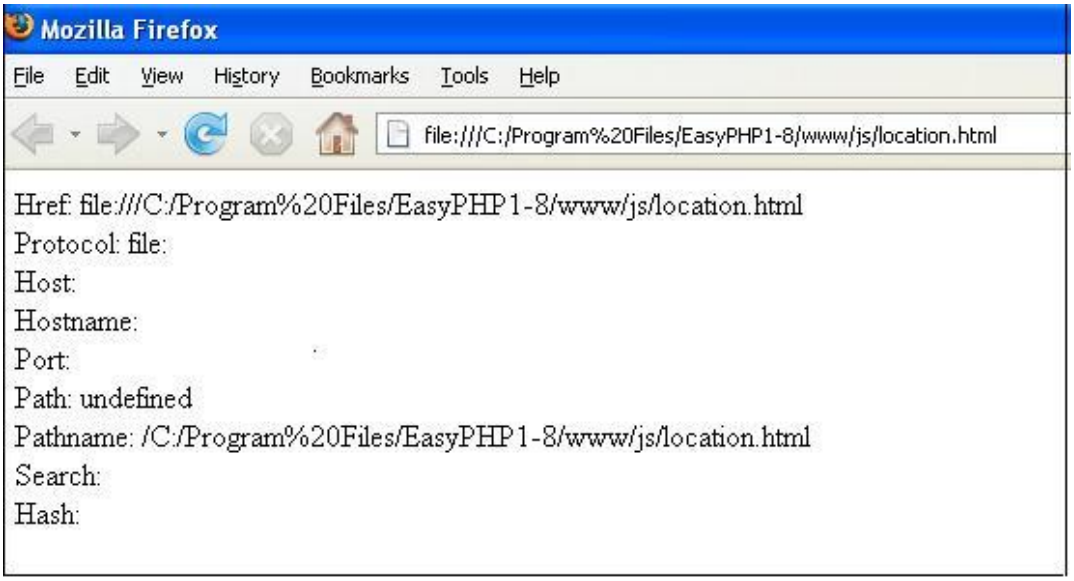
```
noviProzor.location=http://www.mojSajt.com
```

Znak taraba (#) je URL konvencija koja usmerava veb čitač na sidro koje se nalazi u dokumentu. Bilo koje ime da se dodeli sidru (pomoću oznaka <a name “…”>…) postaje deo URL-a posle tarabe.

Svojstvo location.href obezbeđuje znakovni niz kompletnog URL-a za navedeni objekat Window.

Primer

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<body>

<script type="text/javascript">
document.write("Href: "+location.href);
document.write("</br>Protocol: "+location.protocol);
document.write("</br>Host: "+location.host);
document.write("</br>Hostname: "+location.hostname);
document.write("</br>Port: "+location.port);
document.write("</br>Path: "+location.path);
document.write("</br>Pathname: "+location.pathname);
document.write("</br>Search: "+location.search);
document.write("</br>Hash: "+location.hash);

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



Slika 7 Rezultat učitavanja prethodnog primera u brauzer

10.4.4 Objekat History

History objekat je kreiran od strane JavaScript runtime mašine i sastoji se od niza URL-ova. Ovi URL-ovi su URL-ovi koje korisnik može posetiti unutar brauzerovog prozora. History objekat je deo Window objekta i dostupan je kroz window.history svojstva.

Osobina objekta history

Osobina	Opis

length	Vraća broj elemenata u history listi

Tabela 21 Svojstvo objekta history

Metode objekta history

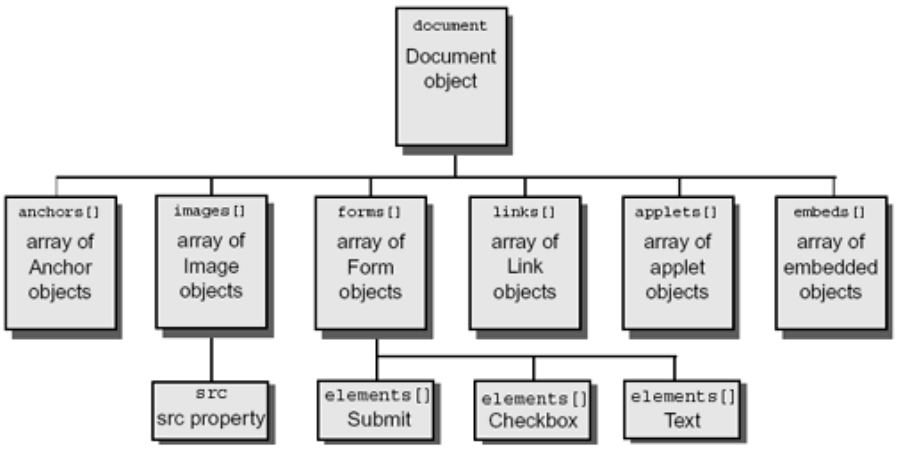
Metode	Opis
back()	Učitava prethodni URL iz history liste
forward()	Učitava sledeći URL u history listu
go()	Učitava zadatu stranicu u history listu

Tabela 22 Metode objekta history

10.4.5 Objekat Document

Document objekat predstavlja ceo HTML document i može se koristiti za pristup svim elementima na stranici. On je deo Window objekta i dostupan je kroz window.document svojstva.

Objekat dokumenta sadrži stvarni sadržaj stranice. Svojstva i metode ovog objekta utiču na izgled i sadržaj dokumenta koji se nalazi u prozoru. Sve informacije koje se nalaze u *head* delu stranice su takođe sadržaj objekta *document*. Sve reference do dokumenta moraju biti potpune, odnosno deo reference *document* koji ukazuje na objekat dokumenta ne sme biti izostavljen pri pisanju skriptova, za razliku od objekta *window*.



Slika 8 Šematski prikaz objekta document

Objekat dokumenta je dosta kompleksan što je i očekivano, s obzirom da objekat dokumenta sadrži sve elemente koji se pojavljuju na stranici. Neka svojstva ovog objekta uspostavljaju se pomoću atributa <*body*

Kolekcije objekta Document

Kolekcija	Opis
anchors[]	Vraća referencu svih Anchor objekata u dokumentu
forms[]	Vraća referencu svih Form objekata u dokumentu
images[]	Vraća referencu svih Image objekata u dokumentu
links[]	Vraća referencu svih Area i Link objekata u dokumentu

Tabela 23 Kolekcije objekta Document

Svojstva objekta Document

Svojstva	Opis
body	Daje direktan pristup <body> elementima
cookie	Postavlja ili vraća sve kukije koji su povezani sa tekućim dokumentom
domain	Vraća ime domena tekućeg dokumenta
lastModified	Vraća datum i vreme dokumenta kada je dokument poslednji put modifikovan
referrer	Vraća URL dokumenta sa koga je učitao tekući dokument
title	Vraća naslov tekućeg dokumenta
URL	Vraća URL tekućeg dokumenta

Tabela 24 Svojstva objekta Document

Metode objekta Document

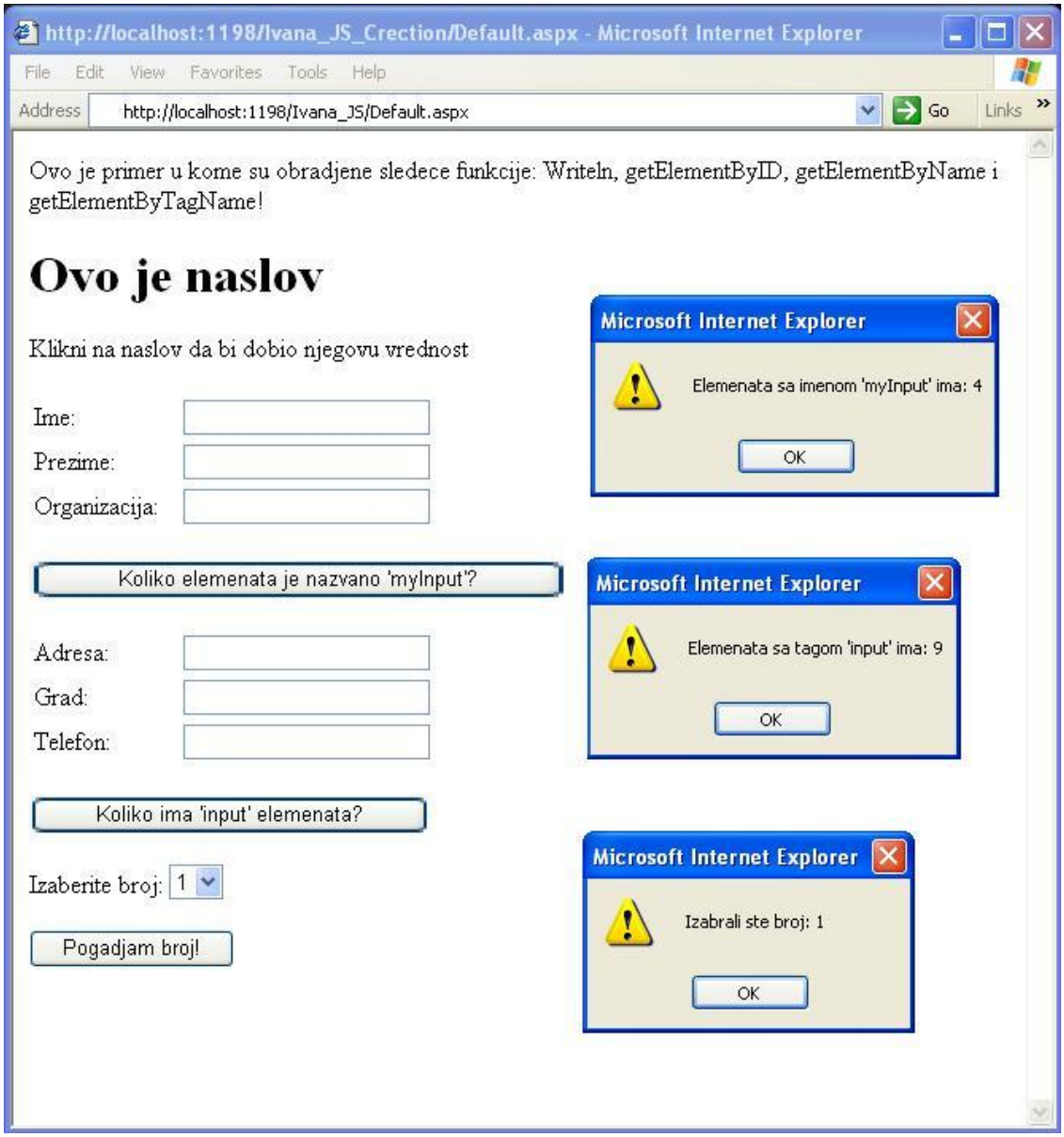
Metode	Opis
close()	Zatvara izlazni strim koji je otvoren pomoću open metode
getElementById()	Vraća referencu na prvi objekat sa zadatim id-em
getElementsByName()	Vraća kolekciju objekata sa zadatim imenom
getElementsByTagName()	Vraća kolekciju objekata sa zadatim imenom taga
open()	Otvara strim da bi sakupio izlaze iz bilo koje metode
write()	Piše HTML izraze ili JavaScript kod u dokumentu
writeln()	Identično kao write metoda sa mogućnošću da se doda jedan karakter za novi red posle svakog izraza

Tabela 25 Metode objekta Document

Primer:

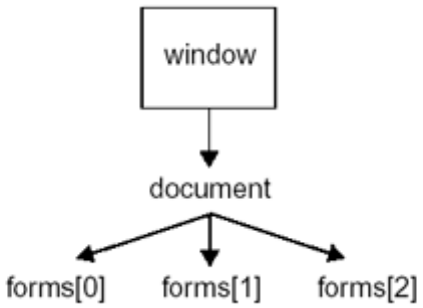
```
<html>
<body>
<div>
<script type="text/javascript">
function getValue()
  {
  var x=document.getElementById("myHeader")
  alert(x.innerHTML)
  }
function getElements_ByName()
  {
  var x=document.getElementsByName("myInput");
  alert("Elemenata sa imenom 'myInput' ima: "+x.length);
  }
function getElements_ByTag()
  {
  var x=document.getElementsByTagName("input");
  alert("Elemenata sa tagom 'input' ima: "+x.length);
  }

function getElements_ById()
  {
  var x=document.getElementById("I3");
  alert("Izabrali ste broj: "+x.value);
  }
</script>
<script type="text/javascript">
document.writeln("Ovo je primer u kome su obradjene sledece funkcije:
Writeln, getElementByID, getElementByName i getElementByTagName!")
</script>
<br />
<br />

<h1 id="myHeader" onclick="getValue()">Ovo je naslov</h1>
<p>Klikni na naslov da bi dobio njegovu vrednost</p>
<table>
    <tr>
        <td width="90px">
            Ime:
        </td>
        <td>
            <input name="myInput" type="text" size="20" id="Ime"/>
        </td>
    </tr>
    <tr>
        <td width="90px">
            Prezime:
        </td>
        <td>
            <input name="myInput" type="text" size="20" id="Prezime"/>
        </td>
    </tr>
    <tr>
        <td width="90px">
            Organizacija:
        </td>
        <td>
           <input name="myInput" type="text" size="20" id="Organizacija" />
        </td>
```

```
    </tr>
</table>
<br />
<input type="button" onclick="getElements_ByName()" value="Koliko elemenata
je nazvano 'myInput'?" id="I1" />
<br /><br />
<table>
    <tr>
        <td width="90px">
            Adresa:
        </td>
        <td>
            <input name="other" type="text" size="20" id="Adresa"/>
        </td>
    </tr>
    <tr>
        <td width="90px">
            Grad:
        </td>
        <td>
            <input name="other" type="text" size="20" id="Grad"/>
        </td>
    </tr>
    <tr>
        <td width="90px">
            Telefon:
        </td>
        <td>
            <input name="other" type="text" size="20" id="Telefon"/>
        </td>
    </tr>
</table>
<br />
<input type="button" onclick="getElements_ByTag()" value="Koliko ima
'input' elemenata?" id="I2"/><br />
<br />
Izaberite broj:
<select name="myInput" id="I3">
      <option value="1">1</option>
      <option value="2">2</option>
      <option value="3">3</option>
</select><br /><br />
<input type="button" onclick="getElements_ById()" value="Pogadjam broj!"
id="I4"/><br />
</div>
</body>
</html>
</body>
</html>
```



Slika 9 Demonstracija metoda objekata Document

10.5. Objekat forme



Slika 10 Objekat forms u DOM hijerarhiji

Interakcija između Web stranice i korisnika odvija se uglavnom unutar forme. To je mesto gde se nalazi dosta korisnih HTML elemenata: polja za tekst, dugmad, polja za potvrdu, liste za izbor itd. Sa mape objekata u modelu objekata dokumenta može se videti da objekat forme ulazi u sastav objekta dokumenta. Iako je uvek sadržan u objektu, pri referenciranju objekta forme uvek se mora navesti i objekat dokumenta u referenci.

Forma ima relativno mali broj svojstava, metoda i procedura za obradu događaja. Skoro sva svojstva su ista kao i atributi oznake *<form>*. Dokument sadrži više formi, a svaka forma proizvoljan broj elemenata. Ovakav odnos odgovara organizaciji HTML oznaka, gde su elementi kao što su oznake *<input>* ugnježdeni između *<form>* i *</form>* tagova.

Forme se prave isključivo od standardnih HTML oznaka na stranici. Može se zadati neki (ili svi) od sledećih atributa: *name, target, action, method* i *enctype*. Svaki od njih je svojstvo objekta forme i može mu se pristupiti korišćenjem samo malih slova u nazivu, kao na primer:

```
document.ImeForme.action;
```

Da bi se promenila vrednost bilo kome od ovih svojstava, jednostavno mu treba dodeliti valjanu vrednost uz pomoć operatora dodele.

10.5.1 Svojstva objekta forme

Svojstvo	Opis
action	URL servera na koji se šalje forma
button	Objekat koji predstavlja generičko dugme
checkbox	Objekat koji predstavlja *checkbox*
elements	Niz koji sadrži element za svako polje forme (*radio* dugme, *checkbox*, dugme)
encoding	MIME tip
fileUpload	Objekat koji predstavlja file-upload polje forme
hidden	Objekat koji predstavlja *hidden* polje forme
length	Broj polja definisan formom
method	*get* ili *post* metoda kojim se forma šalje na server
name	Ime forme
password	Objekat koji predstavlja *password* polje
radio	Objekat koji predstavlja *radio* polje
reset	Objekat koji predstavlja *reset* dugme
select	Objekat koji predstavlja listu selekcije
submit	Objekat koji predsatvlja *submit* dugme
target	Referencira HTML target atribut, ime frejma gde će se korisnički odgovor na poslate podatke biti prikazan
text	Objekat koj predstavlja *text* polje
textarea	Objekat koji predstavlja *text area* polje

Tabela 26 Svojstva objekta forme

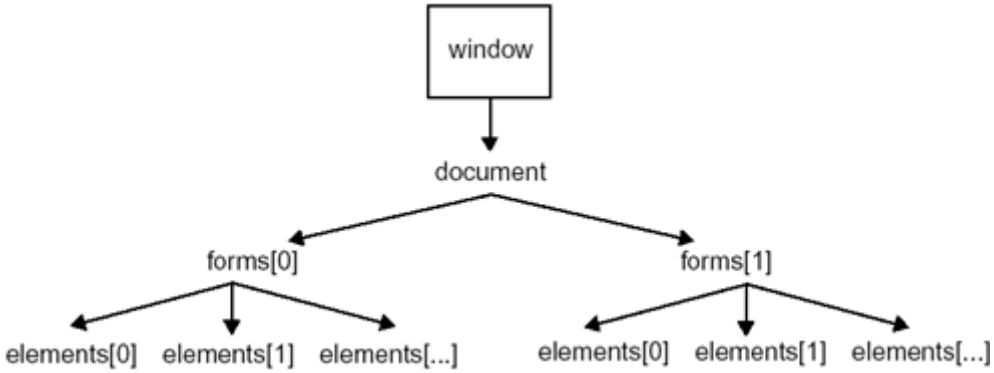
form.action

Ovo svojstvo sadrži URL (relativni ili apsolutni) kome će se pristupiti kada se forma pošalje. Najčešće je to neka stranica na serveru na kojoj se obrađuju podaci koji su poslati putem forme. Ovo svojstvo može samo da se čita.

form.length

Ovo svojstvo vraća broj kontrola (dugmadi, polja za unos teksta...) koje se nalaze unutar forme. Svojstvo *length* se može iskoristiti u petlji za postavljanje granične vrednosti za brojač, kada se treba izvršiti neka akcija za svaki element forme. Ovo svojstvo ima istu vrednost kao i svojstvo *form.elements.length*.

form.elements[]



Slika 11 Objekat elements[] u DOM hijerarhiji

Svojstvo *form.elements[]* vraća listu svih elemenata koji se nalaze unutar forme. Ova lista je poseban niz, sa elementima koju su navedeni po redosledu pojavljivanja njihovih HTML oznaka u programskom kôdu.

10.5.1.1 Svojstva element objekta

Svojstvo	Opis
form	Ime form objekta gde je element definisan (read only)
name	Ime ulaznog uređaja specificirano u HTML-ovom *name* atributu (readonly)
type	Vrsta ulaznog uređaja (*checkbox*, *radio*...)
value	Tekst koji je pridružen ulaznom uređaju, kao na primer tekst pridružen tekst polju itd.

Tabela 27 Svojstva objekta element

Ponekad je potrebno da skript pretraži sve elemente u formi. Pristupajući elementima ovog niza preko indeksa, pristupamo svakoj kontroli unutar forme. Sledeći primer demonstrira upotrebu svojstva *form.elements[]*. U *for* petlji pretražuje se svaki element u formi da bi sadržaju polja za tekst dodelio vrednost praznog niza znakova. Ovaj skript ne može jednostavno da zareda i postavi sadržaj svakog elementa na prazan znakovni niz, jer neki elementi mogu biti dugmad koja nemaju svojstvo kome se može dodeliti prazan znakovni niz. Zbog toga se najpre vrši provera da li je trenutna kontrola tipa *text*.

```
var forma = window.document.forms[0];
for (var i = 0; i < forma.elements.length; i++)
{
    if (forma.elements[i].type == "text ")
        forma.elements[i].value = " ";
}
```

10.5.2 Metode objekta forme

Objekat forme ima dve veoma korisne metode. To su metode *submit()* i *reset()*.

form.submit()

Ova metoda ima isti efekat kao kad se pritisne dugme u okviru forme koje je tipa *submit*. Metoda ne prima nikakve parametre i ne vraća vrednost.

form.reset()

Ova metoda ima isti efekat kao kada se pritisne dugme u okviru forme koje je tipa *reset*. Ni ova metoda ne prima parametre niti vraća vrednost.

10.6 Objekat za tekst (text)

Slika 12 Objekat text u DOM hijerarhiji

10.6.1 Svojstva objekta text

Svojstvo	Opis
defaultValue	Vrednost dodeljena *value* atributu i vrednost koju korisnik vidi u tekst polju čim se učita strana
form	Ime forme gde je tekst polje definisano
name	Ime koje se koristi za referenciranje tekst polja
type	Vrsta input polja (*text*)
value	Value atribut koji će biti dodeljen bez obzira šta korisnik napiše u tekst polju

Tabela 28 Svojstva objekta text

10.6.2 Metode objekta text

Metoda	Opis
blur()	Sklanja fokus sa objekta
focus()	Stavlja fokus na objekat
handleEvent()	Poziva obrađivač događaja za određeni događaj

select()	Selektuje i označava tekst u boxu
unwatch()	Iskljucuje watch za određeno svojstvo
watch()	Posmatra i kad je svojstvo promenjeno poziva funkciju

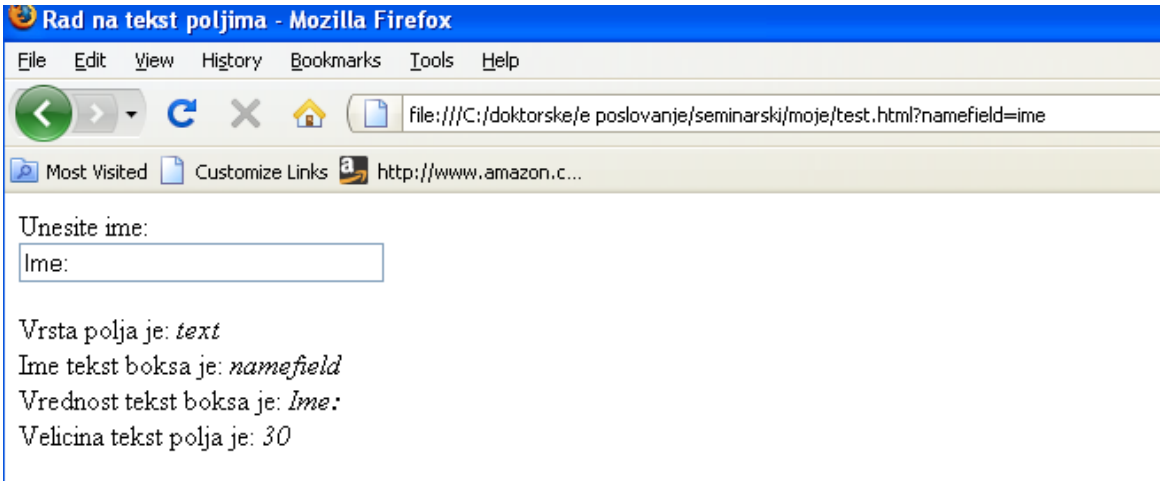
Tabela 29 Metode objekta text

Svaki od četiri elementa forme koji se odnose na tekst (*text*), oblast za tekst (*textarea*), lozinka (*password*) i sakriveni (*hidden*), predstavljaju objekte koji mogu da sadrže tekst. Svi osim sakrivenog objekta prikazuju se na stranici, omogućavajući korisniku da unese informaciju. Imaju ista svojstva i iste metode.

Za sve vidljive objekte u ovoj kategoriji, procedure za obradu događaja se aktiviraju pomoću mnogih akcija korisnika, kao što su fokusiranje polja (pojava kursora u polju) i menjanje teksta (unošenje novog teksta i napuštanje polja). Procedura koja obrađuje događaj promene teksta u polju je *onChange*= procedura.

Bez sumnje, najčešće korišćeno svojstvo objekta tekstualnog polja jeste svojstvo *value*. Ovo svojstvo predstavlja tekući sadržaj elementa teksta. Skript može pozvati i podesiti njegov sadržaj u svakom trenutku. Sadržaj svojstva *value* uvek je znakovni niz. To može zahtevati konverziju u brojeve ukoliko se polja sa tekstom koriste kao ulazne veličine za matematičke operacije.

```
<html>
   <head><title>Rad na tekst poljima</title></head>
   <body>
      <form name="form1">
       Unesite ime:<br>
         <input type="text"
            name="namefield"
            size=30 value="Ime: "
           onFocus="document.form1.namefield.select()">
       </form>
   <script language="JavaScript">
   // Kako referencirati formu kroz JavaScript?
   //  document.form[].element.property
  document.write( "Vrsta polja je:<em> "+
      document.form1.namefield.type);
   document.write( "<br></em>Ime tekst boksa je:<em> "+
      document.form1.namefield.name);
   document.write( "<br></em>Vrednost tekst boksa je:<em> "+
      document.form1.namefield.value);
   document.write( "<br></em>Velicina tekst polja je:<em> "+
      document.form1.namefield.size);
   </script>
   </body></html>
```



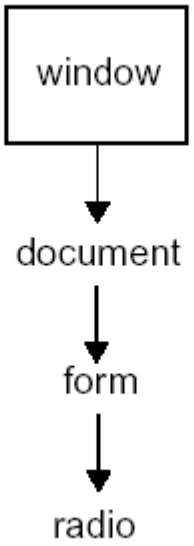
Slika 13 Rezultat izvršavanja gornjeg programa

10.7 Objekat dugme (button)

Dugme je jedan od najjednostavnijih objekata za skriptovanje. Ono ima samo nekoliko svojstava kojima se retko pritupa ili se retko modifikuju u svakodnevnom skriptovanju. Kao i kod tekstualnih objekata, vizuelni efekat dugmeta ne proizvodi HTML ili skript, već operativni sistem i čitač, koje koristi posetilac stranice. Daleko najkorisnija procedura za obradu događaja objekta dugme jeste procedura za obradu događaja *onClick*. Ona se aktivira svaki put kada korisnik kursorom miša pritisne dugme.

Iako je izgled dugmeta diktiran od strane operativnog sistema, njegov izgled se uz pomoć CSS-a može prilagoditi dizajnu stranice, tako što će se pomoću *class* selektora redefinisati njegov izgled na željeni način.

10.8 Objekat radio



Slika 14 Objekat radio u DOM hijerarhiji

10.8.1 Svojstva radio objekta

Svojstvo	Opis
checked	Vraća vrednost *true* ako je selektovana neka vrednost u suprotnom vraća *false*
defaultChecked	Odnosi se na čekirani atribut – ono polje koje korisnik vidi kao čekirano prilikom učitavanja strane
form	Ime forme gde je *radio* polje definisano
name	Ime koje se koristi za referenciranje *radio* polja
type	Odnosi se na vrstu atributa *radio input* polja
value	Odnosi se na vrednost atributa

Tabela 30 Svojstva radio objekta

10.8.2 Metode radio objekta

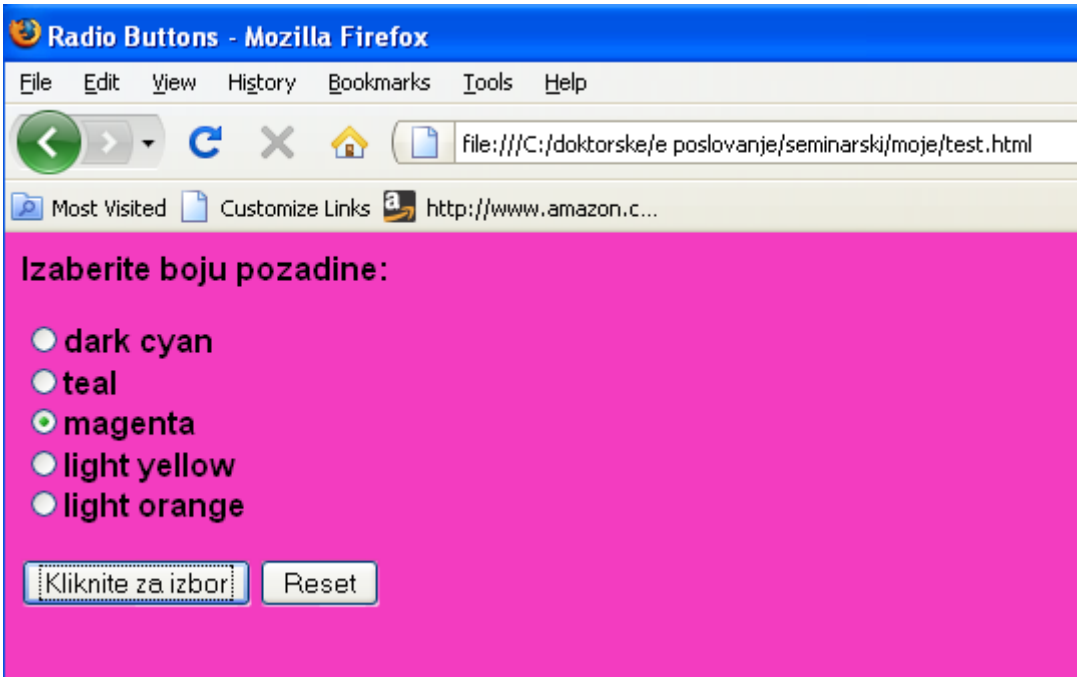
Radio objekat ima iste metode kao i tekst polje.

Primer korišćenja *radio* objekta i njegovih svojstava :

```
<html>
   <head><title>Radio Buttons</title>
   <script name="JavaScript">
     function changeBg(f){
        for (var i = 0; i < f.color.length;i++){
          if(f.color[i].checked){
            document.bgColor= f.color[i].value;
           }
         }
      }
   </script>
   </head>
   <body bgcolor="#CCFFFF">
   <font face="arial"><b>
  <form name="formradio">
   Izaberite boju pozadine:<p>
  <input type=radio
     name="color"
     value="#0099CC">dark cyan<br>
   <input type=radio
     name="color"
     value="#339966">teal<br>
   <input type=radio
     name="color"
     value="#F33CC">magenta<br>
   <input type=radio
     name="color"
     value="#FFFF66">light yellow<br>
   <input type=radio
     name="color"
     value="#FF9933">light orange<br>
   <p>
   <input type=button
     value="Kliknite za izbor" onClick="changeBg(this.form);">
   <input type=reset>
```

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

U gornjem primeru biramo boju pozadine pa kad kliknemo na dugme pozadina dokumenta se menja u novoizabranu boju.



Slika 15 Rezultat izvršavanja gornjeg primera

10.9 Objekat polje za potvrdu (checkbox)

Polje za potvrdu je takođe jednostavan objekat forme, ali neka od svojstava možda nisu baš intuitivna. Za razliku od svojstva *value* jednostavnog objekta dugme, svojstvo *value* polja za potvrdu je tekst koji želimo da povežemo sa tim objektom. Ovaj tekst se ne pojavljuje na stranici u bilo kom obliku, ali ovo svojstvo može biti važno skriptu koji želi da zna više o polju.

Slika 16 Objekat checkbox u DOM hijerarhiji

10.9.1 Svojsta objekta checkbox

Svojstvo	Opis
checked	Vraća vrednost *true* ako je čekirana neka vrednost
defaultChecked	Vraća *true* ako input tag sadrži *checked* atribut, u suprotnom *false*
form	Ime forme gde je *checkbox* polje definisano
name	Ime koje se koristi za referenciranje *checkbox* polja
type	Odnosi se na vrstu atributa *input* polja
value	Odnosi se na tekst dodeljen vrednosti atributa

Tabela 31 Svojstva objekta checkbox

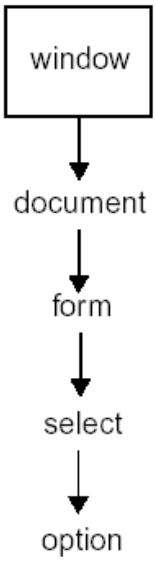
Ključno svojstvo objekta za potvrdu jeste ono koje govori da li je polje potvrđeno ili nije. Svojstvo *checked* je logička vrednost *true* ako je polje potvrđeno, *false* ako nije. Kao i svako svojstvo koje ima logičku vrednost, tako i ovo, može biti iskorišćeno za postavljanje i proveru uslova. Na primer ako imamo jednu formu i jedno polje za potvrdu sa imenom *potvrda*, možemo proveriti da li je ono potvrđeno na sledeći način:

```
if (document.forms[0].potvrda.checked)
  {
    alert('Polje je potvrđeno');
  }
else
  {
    alert('Polje nije potvrđeno')
  }
```

10.9.2 Metode objekta checkbox

Objekat *checkbox* ima iste metode kao objekat *text.*

10.10 Objekat izbora (select)



Slika 17 Select objekat u DOM hijerarhiji

Još jedan objekat forme koji je interesantan za skriptovanje je objekat izbora. Ovaj objekat je prilično složen čemu u prilog govori činjenica da je moguće izabrati više stavki u okviru ovog objekta.

10.10.1 Svojstva objekta select

Svojstvo	Opis
form	Ime forme gde je *select* objekat definisan
length	Broj elemenata select objekta
name	Ime koje se koristi prilikom referenciranja select objekta
options[]	Niza objekata koji opisuju svaku od opcija. Ti objekti imaju svojstva: *index*, *text, selected*, *value*
selectedIndex	Vrednost indeksa selektovane opcije, -1 ako nijedna nije selektovana
type	Može da ima dve moguće vrednosti: da se bira više vrednosti ili samo jedna

Tabela 32 Svojstva objekta select

Neka svojstva pripadaju celom objektu *select*, dok druga pripadaju pojedinačnim opcijama unutar njega. Ako je cilj skripta da se njime odredi koju stavku je korisnik izabrao, moraju da se koriste oba svojstva, izabranog objekta i izabrane opcije.

Najvažnije svojstvo objekta *select* jeste svojstvo *selectedIndex* kome se prisupa na sledeći način:

```
document.form[0].imeSelectObjekta.selectedIndex;
```

Ova vrednost je indeks elementa koji je trenutno izabran.

Dva važna svojstva stavke (option) su *text* i *value*, kojima se pristupa na sledeći način:

```
document.form[0].imeSelectObjekta.options[n].text;
document.form[0].imeSelectObjekta.options[n].value;
```

10.10.2 Metode objekta select

Metode su iste kao kod *text* objekta.

10.11 Provera unetih podataka

Kao što je na samom početku rečeno, jedna od glavnih primena JavaScript-a je provera podataka koje je korisnik uneo u elemente forme. Gotovo je izvesno da će korisnik uneti pogrešnu vrstu podataka u neki element forme. To se može desiti slučajno a nekada je unos namerno netačan kako bi se ispitala imunost aplikacije ili izbeglo davanje potrebnih informacija.

Postoje dva načina provere unetih podataka: u toku unosa podataka i neposredno pre slanja forme na server.

Najbolji trenutak za hvatanje greške je čim je korisnik napravi. Naročito kada je u pitanju dugačka forma sa obiljem raznolikih informacija. Korisniku će biti vrlo prijatno ako odmah pri unosu pogrešne informacije o tome bude i obavešten.

Kako se vrši provera u realnom vremenu? Polja za unos teksta imaju dve procedure za obradu događaja koje se koriste u te svrhe – *onBlur* i *onChange*.

Primena *onBlur* procedure za obradu događaja napuštanja polja može nekada da iritira korisnika. U kombinaciji sa metodom polja za unos *focus()*, može se desiti da nastane naizgled beskonačna petlja sve dok korisnik ne unese tačan podatak.

Problem koji se javlja prilikom korišćenja procedure *onChange* kao okidača za proveru podataka je u tome što korisnik može da ga pobedi. Promena se registruje samo kada se tekst polja zaista i promeni, kada korisnik izađe iz polja pritiskom na taster Tab ili kada pritisne mišem izvan polja. Ako je korisnik upozoren o nekom pogrešnom unosu u polje i ne promeni grešku, događaj promene se neće opet oglasiti. Zbog ovog nedostatka procedure *onChange*, preporuka je da se pre slanja forme ka serveru, podaci opet provere.

U svim čitačima za koje može da se napiše skript, procedura za obradu događaja *onSubmit* otkazuje slanje ako dobije *false* kao krajnji rezultat. Shodno tome, ako se desi da uslov koji je postavljen vrati vrednost *false* to je znak da podaci nisu ispravni i slanje forme se odlaže.

Kako bismo mogli da vršimo proveru podataka unetih u elemente forme, potrebno je da napišemo funkcije koje će izvršiti proveru parametra kojeg smo prosledili funkciji. Moguće je napisati funkcije kojima će se proveravati različiti tipovi podataka i te funkcije snimiti u poseban fajl *.js* i taj fajl koristiti kao biblioteku funkcija.

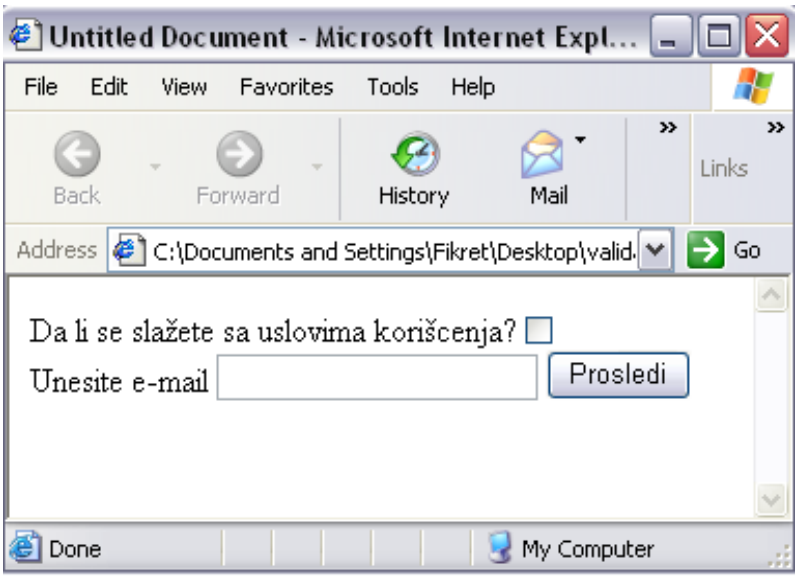
Najčešće provere se odnose na ispitivanje da li je u tekstualno polje unet neki konkretan znak. To je slučaj sa poljem koje treba da sadrži e-mail korisnika. Zatim, često se proverava da li u polju za unos postoji bilo kakav znak, tj. da li je polje prazno. Nekada je obavezno da korisnik čekira polje za tu svrhu (*checkbox* polje forme), pa se tada proverava da li je korisnik to i uradio. U nastavku je dat primer validacije podataka forme.

Ako na stranici imamo formu koja sadrži jedno polje za unos e-mail adrese i jedno polje tipa *checkbox* koje korisnik treba da obeleži, možemo proveriti da li je u polje za e-mail zaista unet e-mail i da li je polje *checkbox* obeleženo. Ako su oba uslova zadovoljena, funkcija će vratiti vrednost *true*, što će značiti da forma može da se pošalje na server. U suprotnom će funkcija vratiti vrednost *false* i podaci neće biti prosleđeni do servera.

Primer HTML kôda kojim je definisana forma i njeni elementi:

```
<form name=”forma” method=”post” onsubmit="return Proveri(this)">
<label> Da li se slažete sa uslovima korišćenja? </label>
<input name="usloviKoriscenja" type="checkbox" value=""  /><br />
<label> Unesite e-mail </label>
<input name="email"  type="text" />
<input type="submit" value="Prosledi" />
</form>
```

Izgled stranice je ovakav:



Slika 18 Izgled stranice za gornji primer

Funkcija za validaciju ove jednostavne forme bi mogla da se napiše na sledeći način:

```
function proveri(forma)
{
        if ((forma.email.value.indexOf('@') == -1) ||   (forma.usloviKoriscenja.checked == false))
{
          alert('Podaci nisu ispravno uneti');
          return false;
}
         return true;
}
```

Funkciji *proveri(),* kao parametar se prosleđuje forma. Zatim se proverava da li polje za unos e-mail adrese sadrži znak @. Ako ne sadrži, metoda *indexOf()* vraća vrednost *-1*, odnosno nepostojeću poziciju u stringu. Tada prvi uslov dobija vrednost *true*. Ako drugi uslov dobije vrednost *true* (odnosno, korisnik je obeležio polje), tada funkcija *proveri()* vraća vrednost *true*. U suprotnom, funkcija obaveštava korisnika o grešci i podaci se ne šalju na server.

10.12 Objekat slike (image)

Jedan od osnovnih objekata koji omogućavaju interakciju korisnika sa stranicom jeste objekat slike. Prostor na stranici (zadat pomoću taga <*img*> u HTML dokumentu) koji je predviđen za smeštanje slike, može dinamički, uz pomoć skripta biti ispunjen nekom drugom

slikom istih dimenzija. Ovo se najčešće koristi kada se kao dugme postavi neka slika, pa pri prelasku kursorom miša preko dugmeta treba korisniku naglasiti da se radi o dugmetu.

Još jedna prednost koju dobijaju autori skriptova jeste ta što se prilikom učitavanja stranice slike mogu prethodno učitati u čitačev keš za slike. Ako nameravamo da menjamo slike kao odgovor na korisnikovu akciju, neće biti kašnjenja pri izmeni slike, jer se ona već nalazi u memoriji čitača.

U JavaScript-u je moguće napraviti novi objekat slike na sledeći način:

```
var slika = new Image(100, 100);
```

U gornjem primeru se vidi da je za pravljenje nove slike korištena ključna reč new i pozvan konstruktor za sliku, koji kao prvi parametar prima širinu slike u pikselima, a kao drugi visinu slike u pikselima.

Da bismo sliku učitali unapred, potrebno je da na prethodno opisani način definišemo nov, prazan objekat *image* koji će biti globalna promenljiva. Slike možemo da učitamo unapred ili naredbama skripta koje se izvršavaju kao odgovor na događaj prozora *onLoad*. Slika koja treba da zauzme prostor u oznaci <*img*> treba da bude istih dimenzija koje su navedene atributima *height* i *width*. Zatim se svojstvu *src* slike dodeli putanja do slike, na primer:

```
slika.src = 'slika2.gif';
```

10.12.1 Svojstva objekta slike

Svojstvo	Opis
border	Ceo broj koji određuje širinu ivice slike u pikselima
complete	Logička vrednost koja vraća *true* ako je završeno učitavanje slike
height	Ceo broj koji prikazuje visinu slike u pikselima
hspace	Ceo broj koj prikazuje horizontalni prostor u pikselima oko slike
lowsrc	Specificira sliku koja se prikazuje u manjoj rezoluciji
name	String koji sadrži ime slike
prototype	Koristi se za dodavanje svojstava image objektu od strane korisnika
src	String koji sadrži putanju i ime slike
vspace	Ceo broj koji pokazuje vertikalni prostor oko slike (u pikselima)
width	Ceo broj koji prikazuje širinu slike u pikselima

Tabela 33 Svojstva objekta slike

10.12.1.1 Svojstvo image.complete

Nekada se pri pisanju skriptova za slike javlja potreba da se utvrdi da li je slika u potpunosti učitana ili ne, odnosno da li je još uvek u toku proces učitavanja slike. Ovo svojstvo ima logičku vrednost *true* ako je proces učitavanja slike završen.

Međutim, treba naglasiti da će svojstvo imati vrednost *true* čak i ako je učitana samo slika koja je definisana atributom *lowsrc*. Zbog toga se ne treba u potpunosti oslanjati na ovo svojstvo, već po proveri vrednosti koju svojstvo ima treba uporediti i vrednost src atributa sa imenom odgovarajuće datoteke. Ako je vrednost *src* attributa u nekom trenutku jednaka vrednosti atributa *lowsrc*, to znači da slika još uvek nije učitana već da je učitana samo slika specificirana atributom *lowsrc*.

Pošto ovo svojstvo ima logičku vrednost, ono se može koristiti za formiranje i proveravanje uslova u *if i if … else* konstrukcijama na sledeći način:

```
if (document.images[3].complete)
{
  // uradi nešto
}
```

10.12.1.2 Svojstvo image.lowsrc

Za slike kojima je potrebno više vremena da se učitaju, čitači nam omogućuju da izaberemo istu sliku sa manjom rezolucijom i 'težinom' koja stoji privremeno dok se ne učita velika slika. Alternativnu sliku dodeljujete pomoću atributa *lowsrc* unutar oznake <*img*>. Ovaj atribut je pridružen svojstvu objekta slike *lowsrc*.

Treba obratiti pažnju na to, da ako se navede *lowsrc* datoteka slike, svojstvo *complete* će dobiti vrednost *true* i procedura za obradu događaja *onLoad* aktiviraće se kada se alternativna datoteka učita: neće se čekati učitavanje glavne *src* datoteke.

10.12.1.3 Svojstvo image.src

Za najefikasniju upotrebu objekta image iz JavaScript-a, ključno je svojstvo *src* koje omogućava da se dodeli URL, čime se učitava nova slika u postojeću oblast prikaza slike.

Treba imati na umu da su visina i širina slike definisani pomoću atributa u oznaci <*img*>. Ako jednostavno stavimo ime nove datoteke koja sadrži sliku u svojstvo objekta slike, JavaScript će podesiti sliku tako da se uklopi u dimenzije definisane atributima oznake <*img*> ili (ako ih nema) u dimenzije prve slike koja je učitana u objekat.

10.13 *Links* objekat

Link objekat je svojstvo document objekta i omogućuje pristup linkovima koji su učitani u dokument. Odgovara HTML-ovom <*a href*> tagu. Svaki link na stranici se nalazi u *links[]* nizu u onom redosledu u kom se link pojavljuje u dokumentu. Prvi link će biti predstavljen kao *document.links[0]*.

Links objekat sadrži URL i sličan je *windows*-ovom *location* objektu i imaju ista svojstva. Postoji i devet događaja koji mogu biti okinuti u radu sa linkom, a to su: *onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp, onMouseOver* i *onMouseOut*.

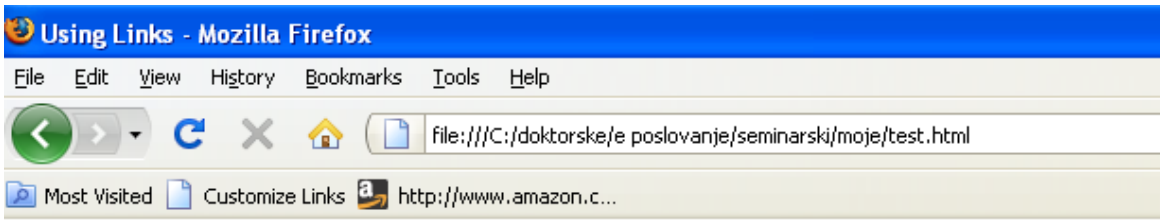
10.13.1 Svojstva Links objekta

Svojstvo	Opis
hash	Deo urla koji se odnosi na anchor
host	Deo url-a koji se odnosi na ime hosta
hostname	Deo url-a koji se odnosi na ime host mašine
href	Ceo url
pathname	Deo urla koji se odnosi na putanju
port	Deo urla koji se odnosi na port
protocol	Deo urla koji se odnosi na protokol
search	Deo urla koji se odnosi na query string
target	Html-ov target atrbitut

Tabela 34 Svojsta objekta Links

```
<html><title>Using Links </title>
   </head>
   <h2>Links i njegova svojstva</h2>
   <body>
  <a href="http://search.yahoo.com/bin/search?p=javascript">
   Search for JavaScript Stuff</a>
   <p>
  <a href="http://google.com" >Go to Google</a>
   <p>
   Click here for Yahoo <br>
  <a href="http://www.yahoo.com">
   <img src="yahoo.bmp">
   </a>
   <script language = "JavaScript">
      document.write("<br><b>Ovaj dokument ima " +document.links.length + "
linka.<br></b>");
         for (i = 0; i< document.links.length; i++){
document.write("<u>document.links["+i+"]:</u><br>");
           document.write("<b>hostname:</b> "+ document.links[i].hostname +"<br>");
           document.write("<b>href: </b>" + document.links[i].href +"<br>");
           document.write("<b>pathname:</b>" + document.links[i].pathname +"<br>");
           document.write("<b>port:</b> " + document.links[i].port +"<br>");
           document.write("<b>query:</b> " + document.links[i].search +"<br>");
          document.write("<b>protocol:</b> "+document.links[i].protocol +"<br><br>");
          }
   </script>
  </body>
</html>
```

Kao rezultat izvršavanja gornjeg programa dobijamo sledeći izlaz:



Links i njegova svojstva

Search for JavaScript Stuff

Go to Google

Click here for Yahoo

Ovaj dokument ima 3 linka.
document.links[0]:
hostname: search.yahoo.com
href: http://search.yahoo.com/bin/search?p=javascript
pathname:/bin/search
port:
query: ?p=javascript
protocol: http:

document.links[1]:
hostname: google.com
href: http://google.com/
pathname:/
port:
query:
protocol: http:

document.links[2]:
hostname: www.yahoo.com
href: http://www.yahoo.com/
pathname:/
port:
query:
protocol: http:

Slika 19 Rezultat izvršavanja gornjeg programa

11 JavaScript Custom objekti

Postoje dva tipa: osnovni i kompozitni. Objekti spadaju u kompozitne tipove. Oni pružaju način da se organizuje kolekcija podataka u jedinstvenu celinu.

Kada se priča o objektima obično se porede sa realnim stvarima u prirodi: knjiga, automobil itd. Ako bismo objekat poredili sa vrstom reči, bili bi imenice. Imenice se opisuju pridevima, na primer: knjiga je stara, ima 400 strana, sadrži pesme. Pridevi koji opisuju objekte zovu se svojstva ili atributi. Objekat sadrži kolekciju svojstava koja ga opisuju. Takođe u jeziku postoje i glagoli koji opisuju šta dati objekat može da radi i šta se može uraditi s njim. Knjiga može da se čita, stranice se mogu listati, može se otvoriti i zatvoriti. Ekvivalent glagola u OOP su metode. Objektni model je hijerarhijska drvolika struktura koja se koristi da bi se opisale komponente nekog objekta. Kada se pristupa objektu u nekom stablu, objekat na vrhu je *koreni* element ili *roditelj* svih drugih objekata. Ako postoji objekat ispod njega on je njegovo dete, a ako postoji još neki objekat na istom nivou za te objekte kažemo da su braća. Objekat dete može takođe imati potomke. Prilikom spuštanja niz drvo tačka se koristi za međusobno odvajanje objekata. Neka je ljubimac objekat roditelj koji ima dvoje dece *mačka* i *pas*. I objekti mačka i pas imaju svojstva koja su im pridružena. Da bismo stigli do svojstva *ime* objekta mačka ili do svojstva *rasa* objekta pas koristićemo sledeću sintaksu: *ljubimac.mačka.ime* tj. *ljubimac.pas.rasa*.

11.1 Kreiranje objekta korišćenjem konstruktora

JavaScript dozvoljava da se objekat kreira na puno načina, a jedan od njih je korišćenjem konstruktora. Konstruktor je posebna vrsta metode koja kreira instancu objekta. JavaScript dolazi sa nekoliko ugrađenih konstruktora. Ispred imena konstruktora stavlja se ključna reč *new* da bi se kreirao objekat.

```
var myNewObject = new Object(argument, argument...);
```

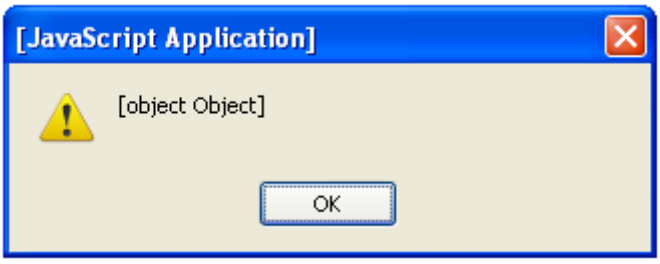
Ako bismo želeli da kreiramo objekat *ljubimac* potrebno je napisati sledeće:

```
var ljubimac = new Object();
```

U ovom slučaju smo koristili predefinisani konstruktor *Object()* koji vraća objekat koji se zove ljubimac. Na taj način je kreiran i instanciran objekat *ljubimac* i spreman je da mu se dodele svojstva i metode.

Sledi primer kreiranja objekta *ljubimac*.

```
<html>
   <head><title>The Object() Constructor</title>
   <script language = "javascript">
      var ljubimac = new Object();
      alert(ljubimac);
   </script>
   </head>
   <body></body>
</html>
```



Slika 20 Rezultat izvršavanja gornjeg programa

Na sličan način se mogu kreirati i objekti *pas* i *macka*.

11.2 Svojstva objekta

Svojstva opisuju objekat i povezana su sa objektom koji opisuju tačkom. Objekat najviši u hijerahiji je *ljubimac* sa decom *macka* i *pas.* Iako je *macka* objekat može se posmatrati i kao svojstvo objekta *ljubimac*. I objekti *pas* i *macka* su objekti koji imaju svoja svojstva kao što su *ime, boja, velicina* itd.

Ako želimo da dodelimo neko svojstvo objektu *macka* sintaksa je sledeća:

```
ljubimac.macka.name = „pegi“;
ljubimac.macka.boja = „sarena“;
ljubimac.macka.velicina = „debela“;
```

11.3 Metode objekta

Metode su specijalne funkcije koje objektno-orijentisani jezici koriste da opišu kako se neki objekat ponaša. Mačka prede, a pas laje. Metode su, kao glagoli, reči koje opisuju neku radnju na objektu. Na primer, objekat *macka* može imati metodu *spavaj()* ili *igraj()* a objekat *pas* može imati metode *ustani()* ili *sedi()*, a oba objekta mogu imati metodu *jedi()*.

Isto kao što se koristi tačka kod odvajanja objekata od njihovih svojstava, koristi se i u pozivu metoda. Za razliku od svojstva, iza metode se piše par zagrada. Ako želimo da pozovemo metodu *igraj()* objekta *macka* koji je dete objekta *ljubimac* to možemo uraditi na sledeći način:

```
ljubimac.macka.igraj();
```

Metode, isto kao i funkcije mogu imati argumente ili poruke koje će biti poslate objektu kao na primer:

```
ljubimac.pas.uhvati(„lopta“);
```

11.4 Objekti definisani od strane korisnika

Svi objekti koji se kreiraju od strane korisnika su potomci objekta nazvanog *Object()*. Operator *new* se koristi da bi se kreirala instanca objekta, iza čega sledi konstruktor. Referenca objekta se vraća i biva dodeljena varijabli.

```
var kola = new Object();
```

```
<html>
<head><title>User-defined objekti</title>
<script language = "javascript">
   var igracka = new Object();   // kreiranje objekta
   igracka.ime = "Lego";      // dodela svojstava objektu
   igracka.boja = "crvena";
   igracka.oblik = "kvadar";
</script>
</head>
<body>
<script language = "javascript">
  document.write("<b>Igracka je " + igracka.ime + ".");
  document.write("<br>Boja je " + igracka.boja + ", a oblik je " + igracka.oblik+ ".");
</script>
</body>
</html>
```

U gornjem primeru je prvo kreiran objekat *igracka* korišćenjem konstruktora *Object()*, zatim su dodeljena svojstva *ime, boja, oblik* objektu *igracka*. Kao rezultat gornjeg primera dobijamo sledeće:

```
Igracka je Lego.
Boja je crvena, a oblik je kvadar.
```

11.4.1 Kreiranje objekta korišćenjem funkcije

Da bismo kreirali svoje objekte možemo da kreiramo funkciju koja specificira ime objekta, svojstva i metode. Ta funkcija će da predstavlja templejt ili prototip objekta. Kada se zove pomoću ključne reči *new* ponaša se kao konstruktor i pravi novi objekat i vraća njegovu referencu.

Ključna reč *this* se koristi da se referencira objekat koji je prosleđen funkciji.

```
<html>
<head><title>User-defined objekat knjiga</title></head>
<script language = "javascript">
  function knjiga(naslov, autor, izdavac){
    // Definisanje svojstava
    this.naslov = naslov;
    this.autor = autor;
    this.izdavac = izdavac;
}
</script>
<body></body>
<script language = "javascript">
  var mojaKnjiga = new knjiga("Zovem se crveno", "Orhan Pamuk", "Geopoetika");
  document.writeln("<b>"   + mojaKnjiga.naslov +"<br>" + mojaKnjiga.autor +"<br>" +
mojaKnjiga.izdavac);
</script>
</body>
</html>
```

U gornjem primeru je kreirana funkcija *knjiga* koja specificira ime objekta i njegova svojstva. Kada želimo da kreiramo objekat tipa *knjiga* pozvaćemo funkciju *knjiga* iza ključne reči *new* sa odgovarajućim prametrima. Rezultat tog poziva će biti napravljen objekat i biće vraćena njegova referenca. Rezultat izvršavanja gornjeg primera je:

```
Zovem se crveno
Orhan Pamuk
Geopoetika
```

11.5 Definisanje metoda za objekte

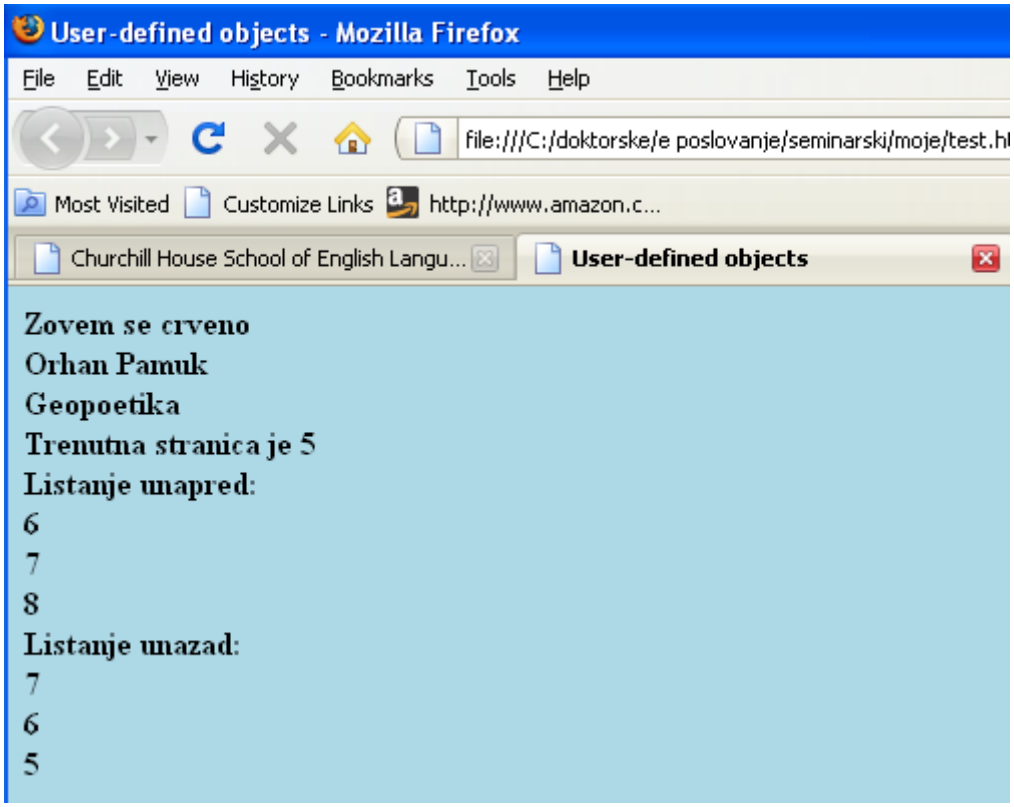
Pored dodeljivanja svojstava datom objektu, neophodno je definisati i njegove metode. Metode su funkcije koje omogućuju da se nešto uradi na objektu. Mala je razlika između funkcije i metode: funkcija je nezavisni deo niza naredbi, a metoda je vezana za objekat i može joj se pristupiti korišćenjem reči *this.*

```
<html>
<head><title>Jednostavne metode</title>
<script language = "javascript">
 function distance(r, t){  // definisanje objekta
   this.rate = r;        // dodeljivanje svojstava
   this.time = t;
 }
 function calc_distance(){  // definisanje funkcije
                      // koja ce se koristiti kao metoda
  return this.rate * this.time;
 }
 </script>
 </head>
 <body>
 <script language="javascript">
  var speed=eval(prompt("Kojom brzinom ste vozili (km/h)? ",""));
  var elapsed=eval(prompt("Koliko dugo ste vozili (h)?" ,""));
  var howfar=new distance(speed, elapsed);
  howfar.distance=calc_distance;  // Kreira svojstvo
  var d = howfar.distance();      // poziva metodu
  alert("Predjeno rastojanje je " + d + " km.");
 </script>
 </body>
</html>
```

Funkcija *distance()* kreira i vraća referencu objekta *distance*. Funkcija *distance* ima dva parametra *r* i *t*. U njoj korišćenjem ključne reči *this* dodeljujemo svojstva *r* i *t* objektu *distance()*. Funkcija *calc_distance()* će se koristiti kasnije kao metoda objekta. Korisnik će biti putem *prompt* prozora pitan za ukupno pređeno rastojanje, kao i dužinu vožnje. Pomoću te dve vrednosti će biti kreiran novi objekat *howfar*. Objekat *howfar* će imati pridruženu metodu *calc_distance*. Zatim pozivamo da se izvrši metoda *distance()* za objekat *howfar*, a vraćenu vrednost dodeljujemo promenljivoj *d.* Na samom kraju ukupno pređeno rastojanje prikazujemo u prozoru *alert.*

```
<html>
   <head><title>User-defined objects</title>
   <script language ="javascript">
     function book(title, author, publisher){
        this.pagenumber=0;      // Svojstva
        this.title = title;
        this.author = author;
        this.publisher = publisher;
        this.uppage = pageForward;   // Dodeljivanje imena funkcije nekom svojstvu
        this.backpage = pageBackward;
      }
     function pageForward(){   // Funkcije koje ce se koristiti kao metode
         this.pagenumber++;
         return this.pagenumber;
      }
     function pageBackward(){
         this.pagenumber--;
         return this.pagenumber;
      }
   </script>
   </head>
   <body bgcolor="lightblue">
   <script language = "javascript">
     var myBook = new book("Zovem se crveno", "Orhan Pamuk","Geopoetika" );    //
kreiranje //novog objekta
     myBook.pagenumber=5;
     document.write( "<b>"+ myBook.title + "<br>" + myBook.author + "<br>" +
myBook.publisher +"<br>Trenutna stranica je " + myBook.pagenumber );
    document.write("<br>Listanje unapred: " );
    for(i=0;i<3;i++){
       document.write("<br>" + myBook.uppage());
        // Listanje unapred
     }
     document.write("<br>Listanje unazad: ");
     for(;i>0; i--){
       document.write("<br>" + myBook.backpage());
        // Listanje unapred
      }
   </script>
   </body>
   </html>
```

U ovom primeru imamo konstruktor funkciju koja se koristi za kreiranje objekta dodeljujući mu i svojstva i metode. U listi parametara se nalaze vrednosti za svojstva *title, author* i *publisher.* U funkciji *book* koristimo ključnu reč *this* da bismo referencirali objekat *book.* Takođe kroz *book* konstruktor inicijalizujemo *pagenumber* svojstvo objekta *book* na *0*. Funkcija *pageForward* predstavlja metodu objekta *book*. Metodu definišemo tako što dodeljujemo funkciju svojstvu *book* objekta. Samo ime metoda (bez zagrada) je dodeljeno svojstvu. Ako bismo stavili zagrade dobili bismo poruku o grešci. Funkcija *pageForward()* ima ulogu da povećava broj strana za jedan, a funkcija *pageBackward()* da ga smanjuje. U primeru smo zatim kreirali objekat *myBook*, postavili *pagenumber* svojstvo na *5*. Kao rezultat izvršavanja gornjeg programa dobijamo sledeći izlaz:



Slika 21 Rezultat izvršavanja gornjeg programa

11.6 Svojstva kao objekti

Već je rečeno da je moguće da jedan objekat bude svojstvo drugog objekta. Taj objekat koji je svojstvo drugog može se kreirati kao što se kreira bilo koji drugi objekat korišćenjem metode konstruktora. Ako je objekat koji se kreira svojstvo nekog drugog objekta onda se ispred njegovog imena ne sme koristiti ključna reč *var*. Na primer: *var ljubimac.macka = new Object()* će izazvati grešku jer je *mačka* svojstvo objekta *ljubimac* a svojstva nikad nisu promenljive.

```
<html>
   <head><title>Svojstva mogu biti objekti</title>
      <script language = "javascript">
         var ljubimac = new Object();  // ljubimac objekat
         ljubimac.macka = new Object();  // macka je svojstvo objekta ljubimac
         // macka je takođe i objekat
         ljubimac.macka.ime="Sylvester";  // dodeljivanje svojstava objektu macka
         ljubimac.macka.boja="black";
         ljubimac.pas = new Object();
         ljubimac.pas.rasa = "Shepherd";
         ljubimac.pas.ime = "Lassie";
      </script>
   </head>
   <body>
      <script language = "javascript">
         document.write("<b>Ime macke je " + ljubimac.macka.ime + ".");
         document.write("<br>Ime psa je " + ljubimac.pas.ime + ".");
      </script>
   </body>
   </html>
```

U ovom primeru kreiramo objekat *ljubimac* korišćenjem konstruktora *Object()*. Zatim pomoću istog konstruktora *Object()* kreiramo objekat *macka* koji je potomak objekta *ljubimac*, a takođe i njegovo svojstvo. Objekat *macka* ima svojstvo *ime* i dodeljujemo mu vrednost *Sylvester*. Zatim kreiramo objekat *pas* koji je takože potomak objekta *ljubimac*, a takođe i njegovo svojstvo. Zatim prikazujemo svojstvo *ime* prvo objekta *macka*, a zatim objekta *pas.* U prvom slučaju je izlaz *Sylvester*, a u drugom *Lassie*.

11.7 Literali objekta

Kada se objekat kreira dodeljivanjem liste svojstava odvojenih tačka-zarezom i oivičenim velikim zagradama, to se zove literal objekta. Svako svojstvo se sastoji od imena svojstva iza koga sledi dvotačka i vrednost svojstva. Literal objekta se može direktno ugnjezditi u JavaScript kod.

```
var object = { property1: value, property2: value };
var area = { length: 15, width: 5 };
```

```
<html>
   <head><title>Object Literals</title>
   </head>
   <body>
      <script language = "javascript">
         var car = {
         make: "Honda",
          year: 2002,
          price: "30,000",
          owner: "Henry Lee",
         };
         var details=car.make + "<br>";
          details += car.year + "<br>";
          details += car.price + "<br>";
          details += car.owner + "<br>";
          document.write(details);
      </script>
   </body>
   </html>
```

U ovom primeru smo prvo kreirali i inicijalizovali literal *car*. Zatim smo mu dodelili svojstva *make, year, price, owner*. Promenljivoj *details* smo dodelili svojstva objekta *car* i prikazali ih. Kao rezultat izvršavanja gornjeg programa dobijamo:

```
Honda
2002
30,000
Henry Lee
```

11.8 Rad sa objektima

11.8.1 Ključna reč *with*

Ključna reč *with* se koristi kao skraćenica za referenciranje svojstava i metoda nekog objekta. Ako stavimo objekat kao njegov parametar sve što se nalazi između velikih zagrada se odnosi na taj objekat. Svojstva i metode se u tom bloku mogu koristiti bez naziva objekta kao u donjem primeru.

```
with(employee){
    document.write(name, ssn, address);
}
<html>
<head><title>Kljucna rec width</title>
<script language = "javascript">
  function book(title, author, publisher){
    this.title = title;   // Svojstva
    this.author = author;
    this.publisher = publisher;
    this.show = display;  // Metoda
 }
 function display(anybook){
 with(this){   // Primer koriscenja with kljucne reci
   var info = "Naslov je " + title;
   info += "\nAutor je " +  author;
   info += "\nIzdavac je " + publisher;
   alert(info);
   }
  }
 </script>
 </head>
 <body>
 <script language = "javascript">
    var childbook = new book("Hari Poter i kamen mudrosti", "J.K Rowling", "Narodna
knjiga");
    var adultbook = new book("Zovem se crveno", "Orhan  Pamuk", "Geopoetika");
    childbook.show(childbook); // poziv metoda decije knjige
    adultbook.show(adultbook);  // poziv metoda za knjigu za odrasle
</script>
</body>
</html>
```

Prvo smo definisali konstruktor *book* sa njegovim svojstvima: *title, author, publisher*, kao i metodu *display*. Kod definisanja funkcije *display* koristili smo ključnu reč *with* pomoću koje smo referencirali svojstva objekta bez korišćenja njegovog imena ili ključne reči *this*. Promenljivoj *info* smo dodelili vrednosti svojstava *book* objekta i na kraju funkcije tu vrednost prikazujemo u *alert* prozoru. U samom kodu pozivamo konstruktor i pravimo dva *book* objekta: *childbook* i *adultbook*, a potom na tim objektima pozivamo metodu *show()*. Kao rezultat izvršavanja ovog programa dobićemo dva alert prozora u kojima su nam ispisana svojstva ova dva kreirana objekta.

11.9 Proširivanje objekata korišćenjem prototipova

Objektno orjentisani programski jezici podržavaju nasleđivanje. Kod nasleđivanja jedan objekat nasleđuje svojstva drugog. U JavaScriptu je podržano nasleđivanje kroz prototip. Moguće je dodati svojstva objektima nakon njihovog kreiranja korišćenjem objekta *prototype*.

JavaScript funkcijama je automatski dodeljen prazan *prototype* objekat. Ako je funkcija u stvari konstruktor objekta onda se prototype objekat može iskoristiti za implementiranje nasleđivanja. Svaki put kada je novi objekat iste klase kreiran taj objekat takođe nasleđuje objekat *prototype* i ista svojstva. Onda će svaki objekat kreiran posle automatski nasleđivati nova svojstva.

11.9.1 Pojam klase u JavaScriptu

Objekat je zajedno sa svojim metodama i svojstvima smešten u kontejner koji se zove klasa, a jedna klasa može da nasledi drugu klasu itd. Iako JavaScript nema ugrađene klase, koncept klasa se može podražavati korišćenjem konstruktora i objekta *prototype*.

Svaka JavaScript klasa ima objekat *prototype* i jedan skup svojstava. Svaki objekat kreiran u okviru klase će naslediti svojstva *prototype*. Recimo da smo definisali konstruktor funkciju koja se zove *Zaposleni()* sa skupom svojstava. *Prototype* objekat ima sva ista svojstva. Konstruktor funkcija *Zaposleni()* predstavlja klasu. Konstruktor se može pozvati i može se instancirati objekat *domar* a zatim ponovo pozvati konstruktor i instancirati novi objekat *menadzer* itd. Svaka instanca klase *Zaposleni()* automatski nasleđuje i sva svojstva definisana za objekat *Zaposleni* kroz njegov *prototype.*

Nakon kreiranja objekta, nova svojstva se mogu dodati korišćenjem svojstva *prototype*. Na taj način je implementirano nasleđivanje u JavaScript.

```
<html><head><title>Prototypes</title>
   <script language = "javascript">
   // Customize String Functions
      function uc(){
         var str=this.big();
         return( str.toUpperCase());
       }
      function lc(){
         var str=this.small();
         return( str.toLowerCase());
       }
      String.prototype.bigUpper=uc;
      String.prototype.smallLower=lc;
      var string="Ovo je string za testiranje";
     string=string.bigUpper();
     document.write(string+"<br>");
     document.write(string.bigUpper()+"<br>");
     document.write(string.smallLower()+"<br>");
   </script>
   </head>
   <body ></body>
   </html>
```

U gornjem primeru smo prvo definisali funkciju *uc()*. Metoda *big()* povećava veličinu fonta, pa ova funkcija kao rezultat vraća *string* napisan svim velikim slovima i većim fontom. Zatim smo definisali funkciju *lc()*. Ova funkcija koristi metodu *small()* pomoću koga smanjuje font i pretvara sva slova stringa u mala. Funkciju *uc()* dodeljujemo svojstvu *String.property.bigUpper*, kreirajući na taj način novu metodu za *String* objekat. Slično radimo i sa funkcijom *lc()* i svojstvom *String.property.smallLower*. Zatim definišemo promenljivu *string* nad kojom ćemo primeniti ove metode. Kao krajnji rezultat izvršenja ove metode dobićemo izlaz kao na slici 22.

Slika 22 Rezultat izvršavanja gornjeg programa

11.10 Različiti načini kreiranja custom objekata

Kreiranje našeg objekta zahteva kreiranje konstruktor funkcije čija uloga će biti da definiše inicijalnu strukturu objekta. Vrednosti koje će biti dodeljene svojstvima objekta se prenose kao parametri funkcije a u samoj funkciji se ove vrednosti dodeljuju svojstvima. Sledeći konstruktor definiše objekat koji ima dva svojstva *ime* i *godine*:

```
function kolega(ime, godine)
{
   this.ime = ime;
   this.godine = godine;
}
```

Ako želimo da kreiramo objekat pomoću konstruktora pozivamo konstruktor korišćenjem ključne reči *new*.

```
var zap1 = new kolega("Mirko", 23);
var zap2 = new kolega("Slavko", 32);
```

Ključna reč *this* u konstruktoru lokalizuje kontekst funkcije na konkretni kreirani objekat. Kako se funkcija koristi za svaki objekat koji se kreira, kontekst je ograničen samo na objekat koji se pravi.

Objekti se mogu kreirati korišćenjem i skraćene sintakse koja definiše objekat unutar vitičastih zagrada. Svojstva imena i vrednosti su definisane unutar vitičastih zagrada kao *ime/vrednost* parovi pri čemu su *ime* i *vrednost* razdvojeni dvotačkom, a svaki par međusobno zarezom. Imena svojstava ne mogu da počnu brojem. Prethodno kreirani objekti mogu biti kreirani i na sledeći način.

```
var zap1 = {ime:"Mirko", godine:23};
var zap2 = {ime:"Slavko",  godine:32};
```

Nakon kreiranja objekata moguće je pristupiti vrednosti svojstva na isti način kao što pristupamo ugrađenim JavaScript svojstvima i objektima. Ako, na primer, želimo da prikažemo podatke iz objekta *zap2* u alert prozoru, to možemo uraditi na sledeći način:

```
alert("Zaposleni " + zap2.ime + " ima " + zap2.godina + " godina.");
```

Kada objekat postoji možemo da dodamo novo svojstvo toj instanci tako što jednostavno dodelimo vrednost proizvoljnom nazivu svojstva. Ako želimo, na primer, da dodamo svojstvo koje opisuje koliko Slavko ima dece možemo da koristimo sledeći iskaz:

```
zap2.brojDece = 2;
```

Posle ovog iskaza, samo *zap2* ima to svojstvo. Ne postoji zahtev da se to svojstvo deklariše u konstruktoru. To takođe znači da možemo da kreiramo prazan objekat i onda eksplicitno popunimo njegova svojstva.

```
var zap1 = new Object( );
zap1.ime = "Mirko";
zap1.godine = 23;
```

Ovakvo kreiranje objekata je obično mnogo teže za održavanje i zauzima mnogo više prostora pogotovu ako je neophodno da se kreira veći broj sličnih objekata.

Ako gore kreiranom objektu želimo da dodamo metode to možemo uraditi na sledeći način. Neophodno je da da ta metoda bude incijalno definisana u kodu kao JavaScript funkcija, a potom ćemo dodeliti referencu toj funkciji kao vrednost za ime metode bilo u konstruktoru ili u *ime/vrednost* paru unutar vitičastih zagrada. Ako želimo da dodamo metodu objektu koji prikazuje alert prozor sa imenom zaposlenog i njegovim godinama neophodno je da prvo napravimo funkciju koja to radi.

```
function prikaziSve( ) {
alert("Zaposleni " + this.ime + " ima " + this.godina + " godina.");
}
```

Potom dodeljujemo funkciju imenu metode u konstruktor funkciju. Izmenjena konstruktor funkcija izgleda ovako:

```
function kolega(ime, godina) {
this.ime = ime;
this.godina = godina;
this.prikazi = prikaziSve;
}
```

Ili ako koristimo drugu sintaksu:

```
var zap1 = {ime:"Mirko", godina:23, prikazi:prikaziSve};
var zap2 = {ime:"Slavko", godina:32, prikazi:prikaziSve};
```

Metodu možemo da pozovemo kroz bilo koji od objekata:

```
zap1.prikazi();
```

JavaScript pruža i neke dodatne prečice u konstruktor funkciji koje omogućavaju automatsku dodelu podrazumevane vrednosti bilo kojem svojstvu koje ima *null* vrednost prosleđenu kao parametar. Na primer ako u konstruktoru objekta *kolega* u pozivu ostavimo drugi parametar prazan, parametar *godine* će biti inicijalizovan sa *null*. Ako želimo da dodelimo validnu predefinisanu vrednost tom svojstvu koristimo sledeću sintaksu.

```
function kolega(ime, godina) {
this.ime = ime;
this.godina = godina || 0;
this.prikazi = prikaziSve;
}
```

Operator koji koristimo je standardni JavaScript *OR* operator. Ako je prva vrednost logički *false* druga vrednost će biti dodeljena svojstvu.

Korišćenje duže sintakse za kreiranje konstruktor funkcije ima tu prednost da je moguće uključiti pozive drugim funkcijama unutar samog konstruktora. Na primer ako želimo da uključimo neke inicijalizacije na tek kreiranom objektu to činimo tako što dodajemo poziv funkciji kao naredbu unutar konstruktor funkcije. Takođe možemo proslediti referencu objektu koji se kreira prosleđivanjem ključne reči *this* kao parametra.

Sledeća funkcija prikazuje *alert* prozor svaki put kad se kreira objekat.

```
function verify(obj) {
alert("Upravo je dodat:  " + obj.ime + ".");
}
function kolega(ime, godina) {
this.ime = ime;
this.godina = godina;
this.prikazi = prikaziSve;
verify(this);
}
```

12 JSON

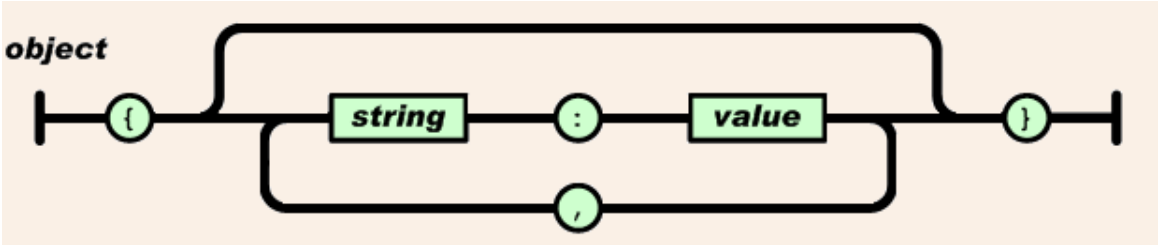
JSON je skraćenica od **Ja**va**S**cript **O**bject **N**otation i jedan od načina korišćenja Ajax tehnologije prilikom pristupa serverskim podacima. Koncept JSON-a je jednostavan: prosleđuje podatke do klijenta kao string i onda korišćenjem JavaScript funkcije *eval()* konvertuje taj string u nizove i objekte.

JSON je "lightweight", otvoreni format za razmenu tekstualno baziranih podataka koji je pogodan za upotrebu u web aplikacijama koje koriste Ajax. Jednostavan je za čitanje i pisanje ljudima, a i mašine ga lako parsiraju i generišu. JSON je tekstualni format koji je kompletno nezavistan jezik ali koristi konvencije koje su poznate programerima koji su programirali u C-olikim jezicima. Ovo čini JSON idelanim jezikom za prenos podataka. JSON je sagrađen od dve strukture:

- Kolekcija ime/vrednost parova. U različitim jezicima ovo je realizovano kroz objekte, strukture, hash tabele ili asocijativne nizove.
- Uređena lista vrednosti. U mnogim jezicima ovo je realizovano kroz nizove, liste, vektore.

12.1 Objekti u JSON-u

Objekat je neuređen skup ime/vrednost parova. Objekat počinje i završava se vitičastim zagradama. Iza svakog imena sledi dvotačka i *ime/vrednost* par odvojen zarezom:



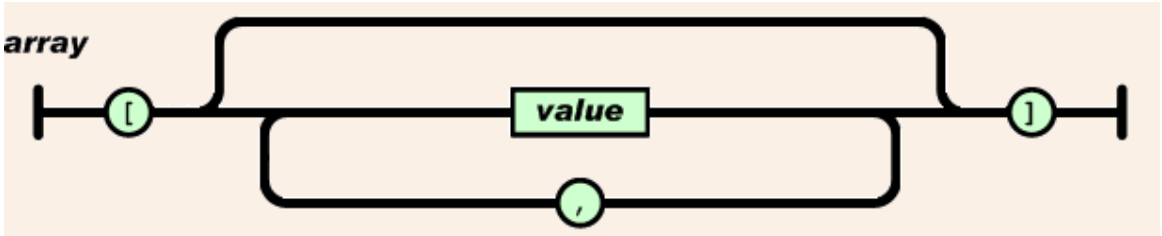
Slika 23 **Definicija JSON objekta**

Primeri objekata:

```
{}
{ članovi }
```

12.2 Nizovi u JSON-u

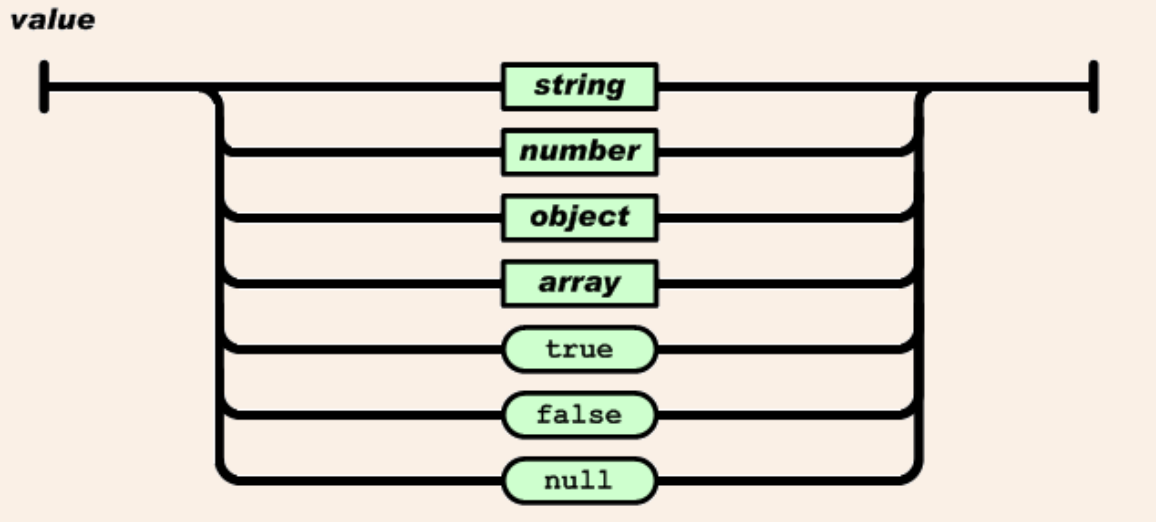
Niz je uređena kolekcija vrednosti. Niz je oivičen srednjim zagradama, a vrednosti su razdvojene znakom zareza.



Slika 24 Definicija JSON niza

12.3 Definicija vrednosti u JSON-u

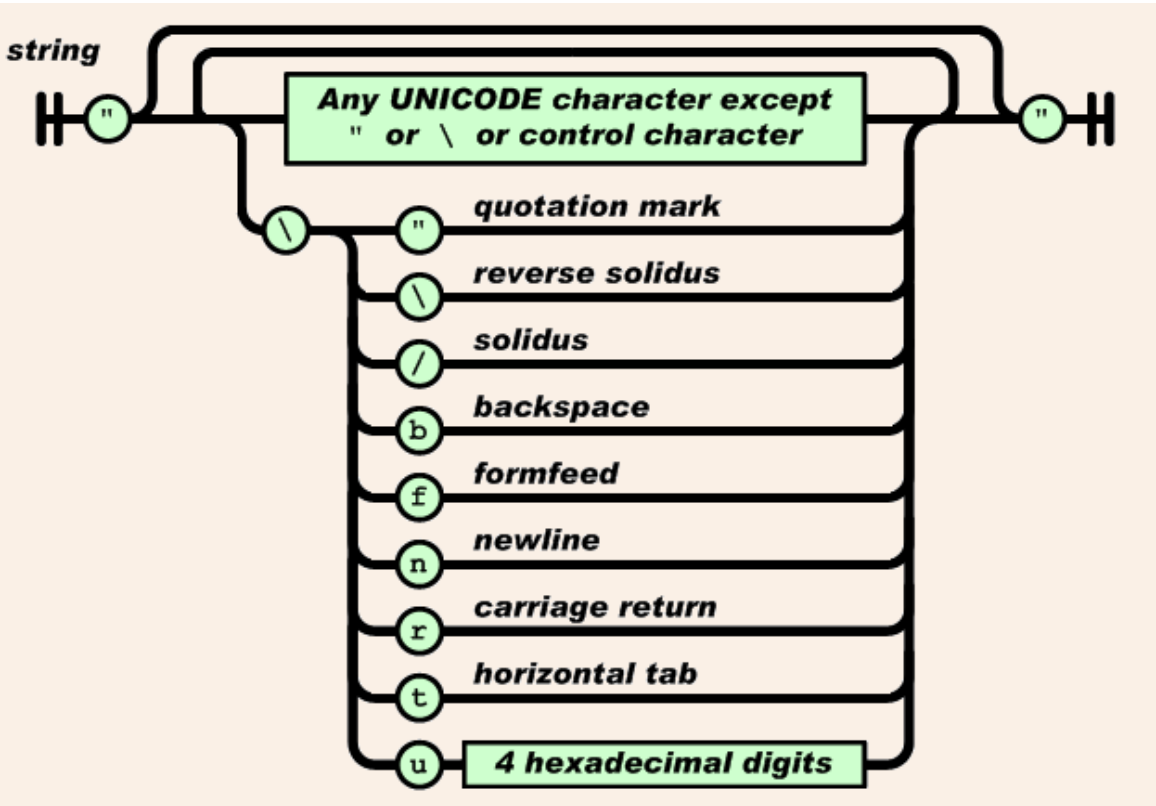
Vrednost može biti string pod znacima navoda ili broj, ili logička vrednost (true, false), objekat ili niz. Ova struktura može biti ugnježdena.



Slika 25 Definicija vrednosti u JSON-u

12.4 Definicija stringa u JSON-u

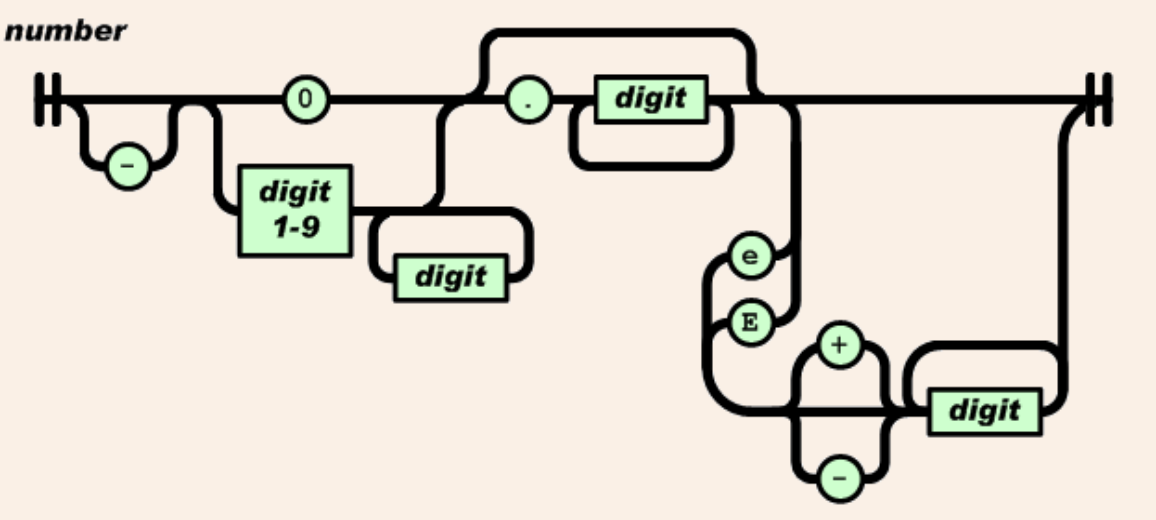
String je kolekcija nula ili više unicode karaktera oivičenih duplim navodnicima. Karakter je predstavljen kao jedinstven karakter. String je predstavljen kao u C-u, tj. Javi.



Slika 26 String u JSON-u

12.5 Definicija broja u JSON-u

Broj je sličan kao broj u C-u ili Javi, osim što se ne koriste oktalni ni heksadecimalni formati.



Slika 27 Broj u JSON-u

Sledeći primer predstavlja JSON reprezentaciju objekta koji opisuje *contact*. Objekat ima string polja za ime i prezime, sadrži objekat koji predstavlja adresu osobe i sadrži listu telefonskih brojeva:

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

Pretpostavimo da se ovaj tekst nalazi u JavaScript promenljivoj *contact*. Na ovaj način se može rekreirati objekat korišćenjem JavaScript ugrađene funkcije *eval().*

```
var p = eval("(" + contact + ")");
```

Polja kao *p.firstName, p.address.city, p.phoneNumbers[0]* su onda dostupna. Promenljiva *contact* mora biti u okviru vitičastih zagrada da bi se uklonila dvosmislenost.

Kao što postoji XML Schema za proveru strukture XML dokumenta, tako postoji i JSON Schema koja radi na sličan način.

12.6 JSON i Ajax

Sledeći JavaScript kod prikazuje kako se može koristiti objekat *XMLHttpRequest* za zahtevanja objekta u JSON formatu od servera.

```
var the_object;
var http_request = new XMLHttpRequest();
http_request.open( "GET", url, true );
http_request.send(null);
http_request.onreadystatechange = function () {
    if ( http_request.readyState == 4 ) {
        if ( http_request.status == 200 ) {
            the_object = eval( "(" + http_request.responseText + ")" );
        } else {
            alert( "There was a problem with the URL." );
        }
        http_request = null;
    }
};
```

Korišćenje *XMLHttpRequest* objekta u ovom primeru nije kompatibilno kroz različite čitače. Čitači mogu da koriste *<iframe>* element da asinhrono zahtevaju JSON podatke da bi bili kompatibilni kroz razne čitače ili jednostavno da koriste *<form action="url_to_cgi_script" target="name_of_hidden_iframe">*. Ovi pristupi su bili prisutni pre širenja *XMLHttpRequest* objekta.

12.7 Sigurnost JSON-a

Iako JSON predstavlja format serijalizacije podataka, njegov dizajn je podskup JavaScript jezika i poteže pitanja zaštite. Centar brige je korišćenje JavaScript interpretatora da dinamički izvrši JSON-ov tekst kao JavaScript što izlaže program nekom zlonamernom skriptu da zaluta. Iako nije jedini način procesiranja JSON-a, to je najlakše i najpopularnije rešenje.

Iako je najveći deo JSON formatiranog teksta takođe JavaScript kod, lak način da JavaScript program parsira podatke formatirane JSON-om je korišćenjem JavaScriptove *eval()* funkcije, koja je dizajnirana da izračuna JavaScript izraze, tako da se JavaScript interpretator koristi za izvršavanje JSON podataka i pravljenje JavaScript objekata. Ova funkcija može biti uzrok raznih zlobnih napada pa se preporučuje da se validira pre korišchenja *eval()* funkcije.

```
var my_JSON_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(text.replace(/"(\\.|[^"\\])*"/g, ''))) && eval('(' + text + ')');
```

Takođe je razvijena i funkcija *parseJSON()* i predstavlja alternativu *eval()* funkciji i procesira JSON podatke ali ne i JavaScript.

12.8 JSON i XML

JSON i XML se mogu koristiti za predstavljanje objekata iz memorije i to u tekstualnom formatu koji je čitljiv od strane ljudi. Štaviše, ova dva formata za razmenu su izomorfna – za dati tekst u jednom formatu postoji ekvivalentan u drugom. Prilikom odlučivanja koji format će se koristiti nije jednostavno napraviti izbor. XML ima korene u označavanju tekstova dokumenta i dobar je na tom polju, dok je JSON s druge strane s korenima u programerskim jezicima i strukturama koje usled toga pružaju prirodnije i dostupnije preslikavanje struktuiranim podacima za razmenu.

Karakteristika	**XML**	**JSON**
Tipovi podataka	Nema pojam tipa podataka. Tipovi podataka se mogu dodati kroz XML šemu.	Pruža skalarne tipove podataka i mogućnost izražavanja struktuiranih podataka kroz nizove i objekte.
Podrška nizovima	Podrška nizovima je moguća ali komplikovana	Ima podršku za nizove
Podrška objektima	Objekti moraju biti izraženi konvencijama često korišćenjem atributa i elemenata	Ima podršku za objekte
Podrška null vrednosti	Zahteva korišćenje *xsi:null* na elementima u XML instanci dokumenta plus importovanje odgovarajucheg namespace-a	Prirodno prepoznaje *null* vrednosti
Komentari	Ima podršku i obično je dostupna kroz API	Nisu podržani
Namespaces	Podržava čim se eliminiše rizik kolizije imena prilikom kombinacije dokumenata. Takođe omogućuju postojećim standardnim XML-ovima da budu proširivi.	Ne postoji koncept namespace-a. Kolizije oko imenovanja se obično izbegavaju ugnježdavanjem objekata ili korišćenjem prefiska.
Odluke formatiranja	Kompleksno. Zahteva veliki napor da se odluči kako se preslikavaju tipovi aplikacije u XML elemente i atribute. Postoje debate o tome koji je pristup bolji da li onaj koji se zasniva na elementima ili na atributima o čemu je već bilo reči.	Jednostavno. Pruža direktnije mapiranje za podatke aplikacije. Jedini nedostatak je odsustvo tipa datuma.
Veličina	Dokumenti su dugački naročito ako se koristi pristup zasnovan na elemetima	Sintaksa je veoma elegantna i to vodi formatiranom tekstu gde je najveći deo prostora iskorišćen za prezentaciju podataka.
Parsiranje u JavaScript-u	Zahteva XML DOM implementaciju i dodatan	Ne zahteva dodatan kod za parsiranje, koristi se

	kod za mapiranje teksta ponovo u JavaScript objekte	JavaScript-ova *eval()* funkcija
Brzina učenja	Zahteva korišćenje nekoliko tehnologija: Xpath, XML Schema, XSLT, XML Namespaces, DOM itd.	Veoma jednostavna tehnologija koja je poznata programerima koji imaju znanje JavaScripta ili nekog programskog jezika.

Tabela 35 Uporedni prikaz XML-a i JSON-a