UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

A J A X T E H N O L O G I J A

DIPLOMSKI MASTER RAD

Mentor: Prof. Dr. Dušan Tošić
Kandidat: Aleksandra M. Janković
Katedra: Računarstvo i informatika

Beograd, jul 2008.

SADRŽAJ:

1. UVOD

U ovom radu je dat prikaz Ajax tehnologije, koja omogućuje dinamičko kreiranje Web stranica, uz objašnjenje na primerima primene svih programskih i skript jezika koje ova tehnologija uključuje, i to:

- osnovne ideje Ajax-a, produktivan pristup konstrukcije Web sajtova, princip funkcionisanja, prednosti i mane ove tehnologije;
- kako se prave asinhroni zahtevi sa JavaScript-om i Ajax-om, detaljan opis XMLHttpRequest objekta, njegovi metodi i osobine; HTTP status codes, ready states;
- opisan je DOM, konvertovanje HTML-a u objektni model za pravljenje interaktivnijih Web strana, struktura DOM drveta, osobine i metode čvorova, primer pravljenja DOM aplikacije;
- opisana je primena XML-a u zahtevima serveru i odgovorima;
- dat je primer korišćenja Google Ajax pretrage uz pomoć javnog API;
- objašnjeno je korišćenje JSON-a za transfer podataka.

U zaključku su date karakteristike novog Web modela Web 2.0., koji obuhvata i Ajax.

Pun naziv korišćenih skraćenica u radu nalazi se u prilogu.

Posebnu zahvalnost dugujem svom mentoru Prof. Dr. Dušanu Tošiću na korisnim sugestijama.

2. KRATAK PRIKAZ AJAX TEHNOLOGIJE

Termin Ajax prvi put je upotrebljen u februaru 2005. godine, kada je Džesi Džejms Garet (Jesse James Garret), dizajner informacionih sistema i direktor kompanije Adaptive Path, pokušao da nađe odgovarajuću skraćenicu za grupu tehnologija koju je predlagao svom klijentu. Ova skraćenica nastala je od fraze „Asynchronous JavaScript and XML", koja nabraja tehnologije koje su u osnovi onoga što se smatra Ajax-om. Dakle, Ajax nije poseban programski jezik, već se radi o skupu tehnologija namenjenom dinamičkom kreiranju Web stranica. Popularnosti Ajax-a je najviše pogodovao trenutak na tržištu u kojem su veliki proizvođači Internet industrije želeli da težište korišćenja računara prenesu sa desktopa na Web stranice, za šta im je bilo potrebno upravo ovako nešto.

Klasičan model Web aplikacija funkcioniše na sledeći način: većina akcija korisnika (na primer klik na link ili slanje podataka preko forme) proizvodi slanje HTTP zahteva serveru. Server poslate podatke obrađuje i korisniku vraća nazad novu HTML stranicu sa rezultatima obrade koji se prikazuju u pretraživaču. Dok server radi, korisnik čeka. U slučaju da postoji intenzivnija interakcija sa korisnikom, to dovodi do niza uzastopnih učitavanja stranica što rezultuje čekanjem na prenos podataka od servera do klijenta – a to je neprijatno iskustvo za korisnika, s obzirom da se ovakve stvari u klasičnim desktop aplikacijama ne dešavaju.
Razlike između desktop i Web aplikacija:

- desktop aplikacije su obično jako brze (one se pokreću na korisničkom računaru), imaju sjajan korisnički interfejs (koji obično komunicira sa operativnim sistemom), i potpuno su dinamične tj. bez čekanja,
- Web aplikacije obezbeđuju servise koje nikad ne bismo mogli da dobijemo na desktopu, ali sa snagom Web-a dolazi i čekanje na odgovor servera, čekanje na osvežavanje ekrana, čekanje da se zahtev vrati i generiše nova strana.

Ajax pokušava da premosti razliku između funkcionalnosti i interaktivnosti desktop aplikacija i uvek ažuriranih Web aplikacija.

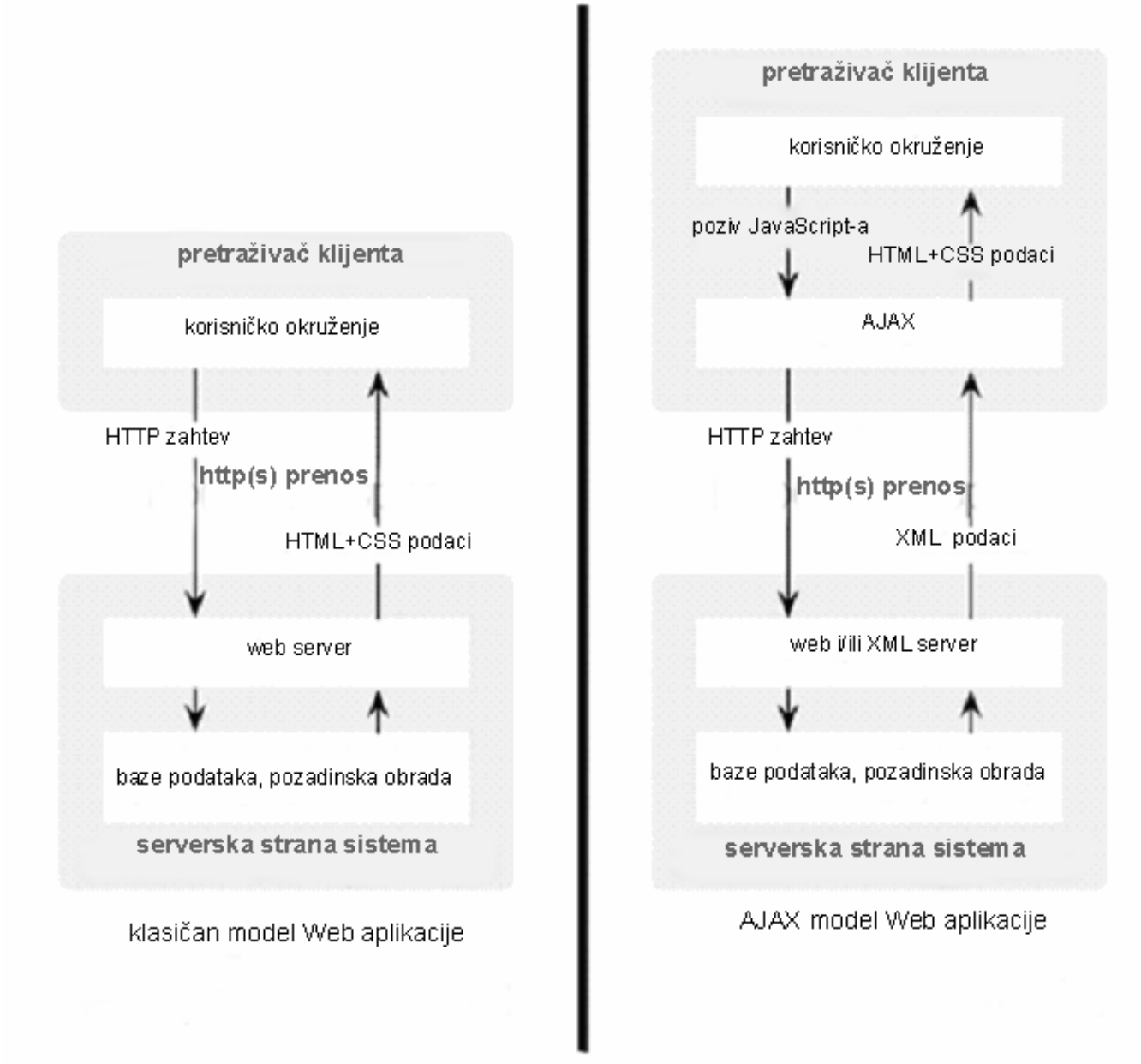
Ideja na kojoj se zasniva Ajax jeste da se stranica na kojoj se odvija Web aplikacija učita samo jednom, a da se svaka dalja komunikacija sa serverom sakrije od korisnika i obavlja bez ponovnog učitavanja čitave stranice. Svaki prenos podataka između servera i klijenta (u slučaju Ajax-a to je pretraživač) vrši se u pozadini. Ovo je moguće korišćenjem JavaScript-a, koji je zadužen za komunikaciju sa serverom – slanje HTTP zahteva, prijem podataka sa servera i njihov prikaz na stranici, kao i za interakciju sa korisnikom.

Ajax tehnologija omogućava da se korisnička interakcija sa aplikacijom dešava asinhrono, nezavisno od komunikacije sa serverom. Kod klasičnog pristupa, akcija korisnika izaziva trenutnu reakciju, dakle, podaci se šalju na server, što blokira sve dalje akcije korisnika dok server ne odgovori na zahtev i pošalje novu stranicu. Kod asinhronog pristupa, sve akcije korisnika prihvata JavaScript, koji u pozadini šalje zahteve serveru prikazujući rezultate kada budu raspoloživi, dok korisnik u međuvremenu može da nastavi sa radom.

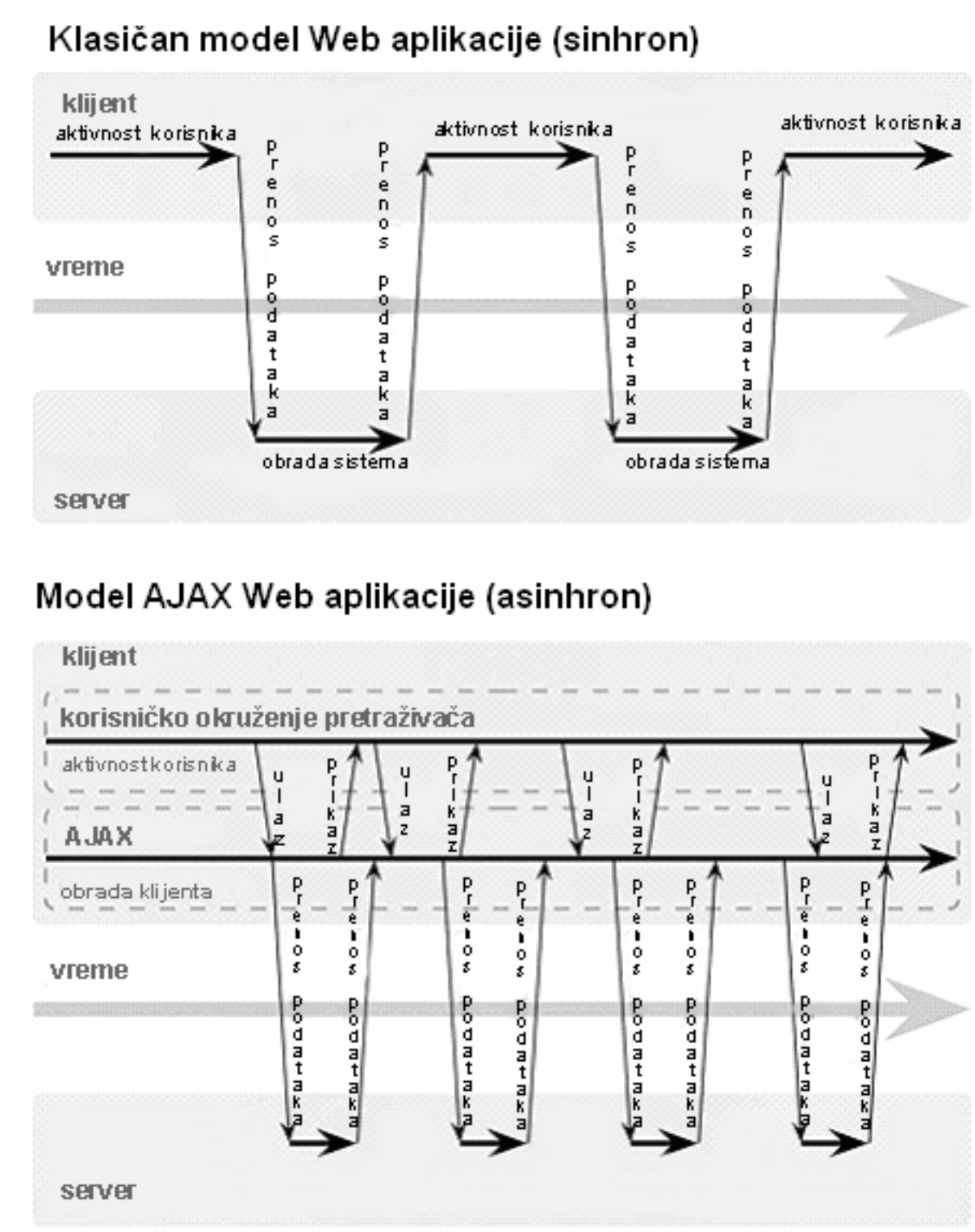
XML se najčešće koristi za prenos podataka sa serverske strane na klijentsku, mada se u Ajax aplikacijama podaci mogu prenositi i u JSON, HTML ili tekstualnom formatu.

Ajax tehnologija kombinuje više programskih i skript jezika:

- za definisanje izgleda stranice koristi se HTML ili XHTML i CSS,
- za dinamičko menjanje delova stranice, bez njenog ponovnog učitavanja u celosti, i za kontrolu interakcije sa korisnikom, koriste se JavaScript i DOM,
- kod komunikacije sa serverom, podaci se prenose pomoću XML-a, JSON-a, čistog teksta ili HTML-a.



Slika 1. Tradicionalan model Web aplikacije u poređenju sa Ajax modelom.



Slika 2. Model sinhrone interakcije tradicionalnih aplikacija (gore) u poređenju sa asinhronim modelom Ajax aplikacije (dole).

U razvoju Ajax pristupa, veliki progres je napravio Google sa njegovim proizvodima: Gmail, Google Groups, Google Suggest i Google Maps, koji su sve Ajax aplikacije. Ostale aplikacije koje koriste AJAX su: Flickr, a9.com pretraživač,...

Prednosti korišćenja Ajax-a:

- daleko bolja interakcija sa korisnikom,
- asinhroni način komunikacije sa serverom, pored bolje interaktivnosti stranice, donosi i smanjenje količine prenetih podataka između servera i klijenta (bandwidth usage), što kod dobro posećenih sajtova može da predstavlja veliki dobitak.

Kako JavaScript predstavlja temelj Ajax-tehnologije, u isto vreme predstavlja i izvor nekih nedostataka. Naime, i pored postojanja standarda, različiti proizvođači pretraživača ne poštuju standarde i dolazi do nekompatibilnosti. Takođe, budući da se Ajax stranice samo jednom učitavaju, „istorija" u pretraživaču postaje neupotrebljiva, tako da programeri moraju na razne načine da se snalaze da ovaj problem reše. Asinhrona komunikacija sa serverom pruža mnoge pogodnosti, ali zahteva i uzimanje u obzir mrežnog kašnjenja i problema u komunikaciji. Pored ovih tehničkih poteškoća, ono što za sajtove može da bude najveći problem jeste njihova optimizacija za pretraživače – budući da se Ajax stranice dinamički generišu, pretraživači često ne mogu da ih protumače, što dovodi do problema u indeksiranju sajtova. Sličan problem postoji i sa alatima za analizu posećenosti stranica – korisnik može i ceo dan da provede na jednoj Ajax stranici, a klasični alati za analizu posećenosti to će protumačiti kao jedno prikazivanje stranice. Međutim, pošto je Ajax podržan od dosta velikih sajtova (kao na primer Google-a), ne treba sumnjati da će se ovi problemi ubuduće brže rešavati.

3. PRAVLJENJE ASINHRONIH ZAHTEVA

3.1. XMLHttpRequest objekat

XMLHttpRequest objekat je JavaScript objekat koji je ključan za Ajax aplikacije. Pomoću njega se šalju zahtevi serverskoj strani aplikacije, kao i primaju povratni podaci od servera. XMLHttpRequest objekat omogućuje direktnu komunikaciju sa serverom, bez ponovnog učitavanja cele strane, što obezbeđuje asinhronost Ajax-a. Kreira se na sledeći način:

Kod 1. Primer kreiranja novog XMLHttpRequest objekta

```
<script language="javascript" type="text/javascript">
var xmlHttp = new XMLHttpRequest();
</script>
```

U normalnim Web aplikacijama, korisnik popunjava polja forme i klikne Submit dugme. Zatim se cela forma šalje serveru, server prosleđuje obradu skript jeziku (PHP, Java, CGI, ..), i kada je skript gotov, vraća kompletno novu stranu. Ta strana može biti HTML sa novom formom sa nekim popunjenim podacima, potvrda ili strana sa nekim opcijama selektovanim bazirano na podacima unetim u orginalnoj formi. Naravno, dok skript ili program na serveru obrađuje i vraća novu formu, korisnik treba da čeka. Ekran postaje prazan i biće ponovo učitan kako se podaci vraćaju sa servera. Ovde dolazi do izražaja slaba interaktivnost - korisnik ne dobija brzo povratnu informaciju, i sigurno se ne oseća kao da radi na desktop aplikaciji.

Ajax stavlja JavaScript tehnologiju i XMLHttpRequest objekat između Web forme i servera. Kada korisnik popuni formu, ti podaci su poslati JavaScript kodu (a ne direktno serveru), koji šalje zahtev serveru. Dok se to dešava, forma na korisničkom ekranu ne nestaje. Zahtev je poslat asinhrono, što znači da JavaScript kod i korisnik ne čekaju na server da odgovori, tako da korisnik može da nastavi da unosi podatke, skroluje i koristi aplikaciju. Server vraća podatke nazad JavaScript kodu koji odlučuje šta da radi sa tim podacima. Može da ažurira polja forme odmah, stvarajući utisak da je brza aplikacija - korisnici dobijaju nove podatke bez slanja njihovih formi ili osvežavanja.

Korišćenje JavaScript koda se svodi na sledeće zadatke:

1. Uzimanje podataka iz HTML forme i slanje serveru,
2. Promena vrednosti na formi,
3. Parsiranje HTML i XML-a: JavaScript se koristi da manipuliše DOM-om i da radi sa strukturom HTML forme i XML podacima koje server vrati.

Za prva dva zadatka koristi se **getElementById()** metod:

Kod 2. Primer uzimanja i postavljanja vrednosti forme sa JavaScript kodom

```
// Uzimanje vrednosti polja "telefon" i smeštanje u promenljivu  sa imenom "telefon"
var telefon = document.getElementById("telefon").value;
// Postavljanje nekih vrednosti forme korišćenjem niza koji se zove "odgovor"
document.getElementById("narudzbina").value = odgovor[0];
document.getElementById("adresa").value = odgovor[1];
```

3.2. Rad sa različitim pretraživačima

Internet Explorer koristi MSXML parser za rukovanje XML-om tako da, kada se koristi Internet Explorer, mora da se kreira objekat na poseban način:

Kod 3. Primer kreiranja XMLHttpRequest objekta u Microsoft pretraživaču

```
var xmlHttp = false;
try {
xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
try {
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
} catch (e2) {
xmlHttp = false;
}
}
```

Kada se koristi Mozilla i pretraživači koji nisu Microsoft-ovi dovoljno je napisati: var xmlHttp = new XMLHttpRequest object .

Da bi XMLHttpRequest objekat bio podržan na svim pretraživačima piše se:

Kod 4. Primer kreiranja XMLHttpRequest objekta za više pretraživača

```
/* Kreiranje novog XMLHttpRequest objekta za komunikaciju sa Web serverom */
var xmlHttp = false;
try {
xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
try {
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
} catch (e2) {
xmlHttp = false;
}
}
if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
xmlHttp = new XMLHttpRequest();
}
```

U ovom kodu se kreira promenljiva xmlHttp da referiše na XMLHttpRequest objekat, kreira se objekat u Microsoft pretraživaču. Ako posle toga xmlHttp i dalje nije postavljen, kreira se objekat na ne-Microsoft način. Na kraju xmlHttp bi trebalo da referiše na ispravan **XMLHttpRequest** objekat.

Generalna procedura korišćenja XMLHttpRequest:

1. Uzimaju se svi podaci koji su nam potrebni iz Web forme,
2. Gradi se URL za konektovanje,
3. Otvara se konekcija za server,
4. Podešava se funkcija koju će sever pokrenuti kad je gotov,
5. Slanje zahteva.

Kod 5. Primer pravljenja zahteva pomoću Ajax-a

```
function callServer() {
// Uzimanje grada i države iz Web forme:
var grad = document.getElementById("grad").value;
var drzava = document.getElementById("drzava").value;
// Jedino ako postoje vrednosti za oba polja, nastavlja
if ((grad == null) || (grad == "")) return;
if ((drzava == null) || (drzava == "")) return;
// Konstruisanje URL na koji se konektuje
var url = "/scripts/getZipCode.php? grad=" + escape(grad) + "& drzava=" + escape(drzava);
// Otvara se konekcija ka serveru
xmlHttp.open("GET", url, true);
// Podešavanje funkcije koji server pokreće kada je gotov
xmlHttp.onreadystatechange = updatePage;
// Slanje zahteva
xmlHttp.send(null);
}
```

Uzimaju se vrednosti sa nekoliko polja forme, kod postavlja PHP skript kao destinaciju na koju da se konektuje. Konekcija je otvorena. Metod konekcije je određen (GET), kao i URL na koji da se konektuje. Kada je treći parametar postavljen na true, zahteva se asinhrona konekcija, pa korisnici i dalje mogu da koriste formu dok server obrađuje zahtev u pozadini. Kada je postavljen na false, kod bi čekao na serveru kada je zahtev napravljen i ne bi nastavljao dok odgovor ne bi bio primljen.

Onreadystatechange opcija xmlHttp (instanca XMLHttpRequest objekta) omogućava da se serveru kaže šta da radi kad završi obradu. Na kraju je pozvan **send()** sa vrednošću null.

Rukovanje odgovorom servera:

- ništa se ne radi dok **xmlHttp.readyState** vrednost nije 4,
- server će odgovor smestiti u **xmlHttp.responseText** osobinu.

Kod 6. Primer rukovanja odgovorom servera

```
function updatePage() {
if (xmlHttp.readyState == 4) {
var response = xmlHttp.responseText;
document.getElementById("zipCode").value = response;
}
}
```

Ovaj kod čeka da ga server pozove sa ispravnim “ready state” i onda koristi vrednost koju je server vratio (ZIP kod za unet grad i zemlju) da postavi vrednost drugog polja forme. Rezultat je iznenadna pojava **zipCode** polja sa ZIP kodom, a korisnik nikad nije kliknuo dugme. Korisnik će moći i da ponovo upiše novu vrednost. Znači, JavaScript metod uzima informacije koje korisnik stavlja u formu, šalje serveru, obezbeđuje drugi JavaScript metod da sluša i obrađuje odgovor, i postavlja vrednost polja kad taj odgovor dođe nazad.

Kod 7. Primer pozivanja JavaScript metoda

```
<form>
<p>Grad: <input type="text" name="grad" id="grad" size="25" onChange="callServer();" /></p>
<p>Država: <input type="text" name="drzava" id="drzava" size="25" onChange="callServer();"
/></p>
<p>Zip Code: <input type="text" name="zipCode" id="zipCode" size="5" /></p>
</form>
```

Kada korisnik stavi novu vrednost za grad ili državu, **callServer()** metod se pokreće i Ajax počinje.

Najvažniji metodi i osobine objekta XMLHttpRequest su:

- **open()**: metod pomoću koga se postavlja novi zahtev serveru,
- **send()**: metod pomoću koga se šalje zahtev serveru,
- **abort()**: metod pomoću koga se odustaje od trenutnog zahteva,
- **readyState**: osobina koja obezbeđuje trenutno HTML ready state,
- **responseText**: osobina - tekst koji server šalje nazad kao odgovor zahtevu.

3.3. Kreiranje nove promenljive i dodeljivanje instance XMLHttpRequest objektu

U sledećem primeru se kreira nova promenljiva pod imenom request i dodeljuje joj se vrednost false. Vrednost false će značiti da XMLHttpRequest objekat nije još kreiran. U try/catch bloku se pokušava i kreira XMLHttpRequest objekat. Ako je to neuspešno (catch (trymicrosoft)), pokušava se i kreira Microsoft-kompatibilan objekat korišćenjem novije verzije Microsoft-a (Msxml2.XMLHTTP). Ako je to neuspešno (catch (othermicrosoft)), pokušava se i kreira Microsoft-kompatibilan objekat korišćenjem starije verzije Microsoft-a (Microsoft.XMLHTTP). Ako je i to neuspešno (catch (failed)), request se postavlja na false. Zatim se proverava da li je request i dalje false, i ako jeste korisnik se obaveštava da postoji problem. Ako je true, inicijalizacija je prošla uspešno.

Kod 8. Primer dodavanja podrške za Microsoft pretraživače

```
<script language="javascript" type="text/javascript">
var request = false;
try {
request = new XMLHttpRequest();
} catch (trymicrosoft) {
try {
request = new ActiveXObject("Msxml2.XMLHTTP");
} catch (othermicrosoft) {
try {
request = new ActiveXObject("Microsoft.XMLHTTP");
} catch (failed) {
request = false;
}
}
}
if (!request)
alert("Greška u inicijalizaciji XMLHttpRequest!");
</script>
```

Kod je ugrađen direktno unutar script tagova, što znači da se radi o statičkom JavaScript-u i da se kod izvršava malo pre nego što je strana prikazana korisniku. Kod se može smestiti u metod, ali to se ređe koristi zbog kašnjenja obaveštenja o greški.

3.4. Slanje zahteva sa XMLHttpRequest

Jednom kada se formira objekat, može da započne komunikacija zahtev-odgovor. Jedina namena **XMLHttpRequest** je da dozvoli da se prave zahtevi i primaju odgovori. Sve ostalo: menjanje korisničkog okruženja, zamena slika, tumačenje podataka koje sever šalje nazad je posao JavaScripta, CSS ili drugog koda na strani klijenta. Ajax ima sigurnosni model, koji obezbeđuje da Ajax kod može da pravi zahteve samo istim domenima na kojima je pokrenut.

3.5. Konstruisanje URL adrese servera na koju se konektujemo

URL se najčešće konstruiše kao set statičkih podataka kombinovanih sa podacima iz forme sa kojom korisnik radi. U sledećem primeru je prikazan JavaScript koji uzima vrednost polja telefon a zatim konstruiše URL adresu koristeći taj podatak.

Kod 9. Primer kontruisanja zahtevanog URL

```
<script language="javascript" type="text/javascript">
var request = false;
try {
request = new XMLHttpRequest();
} catch (trymicrosoft) {
try {
request = new ActiveXObject("Msxml2.XMLHTTP");
} catch (othermicrosoft) {
try {
request = new ActiveXObject("Microsoft.XMLHTTP");
} catch (failed) {
request = false;
}
}
}
if (!request)
alert("Greška u inicijalizaciji XMLHttpRequest-a!");
function getCustomerInfo() {
var telefon = document.getElementById("telefon").value;
var url = "/cgi-local/lookupCustomer.php?telefon=" + escape(telefon);
}
</script>
```

U sledećem kodu se vidi XHTML za ovu formu.

Kod 10. Primer XHTML forme

```
<body>
<p><img src="logo.gif" alt="Slika 1" /></p>
<form action="POST">
<p>Unesite Vaš broj telefona:
<input type="text" size="14" name="telefon" id="telefon" onChange="getCustomerInfo();" />
</p>
<p>Vaša porudžbina će biti isporučena na:</p>
<div id="adresa"></div>
<p>Ukucajte Vašu porudžbinu ovde:</p>
```

```
<p><textarea name="porudžbina" rows="6" cols="50" id="porudžbina"></textarea></p>
<p><input type="submit" value="Poruči hranu" id="submit" /></p>
</form>
</body>
```

Kada korisnik unese broj telefona ili promeni broj, aktivira se **getCustomerInfo()** metod. Taj metod uzima broj i koristi ga da konstruiše URL string koji se čuva u **url** promenljivoj.

3.6. Otvaranje zahteva

Zahtev može da se konfiguriše sa URL-om konekcije, pomoću **open()** metoda XMLHttpRequest objekta. Ovaj metod ima 5 parametara:

1. **Request-type** - tip zahteva za slanje; standardne vrednosti su GET ili POST, ali može da se šalje i HEAD zahtev,
2. **Url** - URL na koji se konektujemo,
3. **Asynch** - true ako treba da zahtev bude asinhron ili false ako treba da bude sinhron (opcioni parametar, podrazumevana vrednost je true),
4. **Username** - ako se zahteva identifikacija(opcion je i nema podrazumevanu vrednost),
5. **Password** - ako se zahteva identifikacija (opcion je i nema podrazumevanu vrednost).

U sledećem primeru se poziva open()-metod sa argumentima: GET zahtev za slanje, URL adresa i true vrednost za asinhron zahtev.

Kod 11. Primer otvaranja zahteva

```
function getCustomerInfo() {
var phone = document.getElementById("telefon").value;
var url = "/cgi-local/lookupCustomer.php?telefon=" + escape(telefon);
request.open("GET", url, true);
}
```

3.7. Slanje zahteva

Kada se zahtev jednom konfiguriše **open()** metodom, može da se šalje metodom **send()**. **Send()** uzima jedan parametar, a to je sadržaj za slanje.

Umesto toga, podaci mogu da se šalju i pomoću GET zahteva i njegovog parametra URL, koji sadrži podatke, a argument **send()** metoda u ovom slučaju je **null**. Ovakva upotreba send()-metoda može se videti u sledećem primeru.

Kod 12. Primer slanja zahteva

```
function getCustomerInfo() {
var telefon = document.getElementById("telefon").value;
var url = "/cgi-local/lookupCustomer.php?telefon=" + escape(telefon);
request.open("GET", url, true);
request.send(null);
}
```

Kada se šalju poverljive informacije ili XML-podaci, onda se sadržaj šalje pomoću send() metoda.

Pomoću **onreadystatechange** osobine se definiše povratni metod, koji se pokreće kada server završi obradu zahteva. Time se dopušta serveru da se vrati u kod Web strane, kada završi obradu zahteva. Ovde dolazi do izražaja asinhronost: korisnik rukuje formom na jednom nivou, dok na drugom nivou server odgovara zahtevu i poziva povratni metod koji je specificiran kao vrednost svojstva **onreadystatechange.** U sledećem primeru se definiše povratni metod updatePage().

Kod 13. Primer podešavanja povratnog metoda

```
function getCustomerInfo() {
var telefon = document.getElementById("telefon").value;
var url = "/cgi-local/lookupCustomer.php? telefon=" + escape(telefon);
request.open("GET", url, true);
request.onreadystatechange = updatePage;
request.send(null);
}
```

Ova vrednost je postavljena pre nego što je send() pozvan.

3.7.1. HTTP ready state

HTTP ready state određuje status ili stanje zahteva. Koristi se da se utvrdi da li je zahtev počeo, da li je odgovoreno, ili je zahtev/odgovor završen. Takođe se koristi u određivanju da li je sigurno čitanje podataka poslatih od strane servera. Funkcija koja je dodeljena **onreadystatechange** vrednosti se poziva kada server završi sa zahtevom i kada se **HTTP ready state** promeni.

Postoji 5 **ready state**:

- **0** : zahtev nije inicijalizovan (pre nego što pozovemo open()),
- **1** : zahtev je postavljen, ali nije poslat (pre nego što se pozove send()),
- **2** : zahtev je poslat i obrađuje se,
- **3** : zahtev se obrađuje; često su neki delimični podaci dostupni iz odgovora, ali sever nije završio sa odgovorom,
- **4** : zahtev je gotov; može da se uzme odgovor servera i da se koristi.

Najbitnije je stanje 4, koje govori da je odgovor servera kompletiran, da je sigurno da se provere podaci iz odgovora i da se koriste. U sledećem primeru, na početku povratnog metoda updatePage() se proverava ready stanje.

Kod 14 Primer provere ready stanja

```
function updatePage() {
if (request.readyState == 4)
alert("Server je gotov!");
}
```

3.7.2. HTTP status kodovi

Prilikom rada mogu se javiti različite greške, na primer pogrešno ukucana URL adresa, kada se dobija kod greške 404 koji govori da strana ne postoji, ili da se pristupa osiguranim (secured) ili zabranjenim podacima (kod 403 i 401). Ovo su sve kodovi koji rezultiraju iz kompletiranog odgovora, tj server je izvršio zahtev (HTTP ready state je 4), ali verovatno ne vraća podatke koje klijent očekuje. Zato se mora proveriti i HTTP status i ako je on 200, to govori da je sve u redu. Znači sa **ready state** 4 i **status code** 200, mogu da se obrađuju podaci sa servera i ti podaci bi trebalo da budu ono što je traženo. Takva provera se može videti u sledećem primeru.

Kod 15 Primer provere HTTP status koda

```
function updatePage() {
if (request.readyState == 4)
if (request.status == 200)
alert("Server je gotov!");
}
```

Detaljnija provera greške može se izvšiti na sledeći način:

Kod 16 . Primer detaljnije provera greške

```
function updatePage() {
if (request.readyState == 4)
if (request.status == 200)
alert("Server je gotov!");
else if (request.status == 404)
alert("Zahtevani URL ne postoji");
else
alert("Greska: status code je " + request.status);
}
```

3.8. Čitanje teksta odgovora

Rukovanje podacima poslatim od servera vrši se pomoću **responseText** svojstva XMLHttpRequest objekta. Tekst koji server vraća može biti u bilo kom formatu: CSV(comma separated values), PSV(pipe separated values), dugačak string. U sledećem primeru vraćaju se porudžbina i adresa razdvojene ‘pipe’ simbolom, koje se zatim koriste da se postave vrednosti elemenata forme.

Kod 17. Primer rukovanja odgovorom servera

```
function updatePage() {
if (request.readyState == 4) {
        if (request.status == 200) {
                var odgovor = request.responseText.split("|");
                document.getElementById("porudzbina").value = odgovor [0];
                document.getElementById("adresa").innerHTML = odgovor [1].replace(/\n/g, "");
                } else
        alert("status je " + request.status);
        }
}
```

Još jedna važna osobina XMLHttpRequest objekta se zove **responseXML**. Ta osobina sadrži XML odgovor u događaju (event) koji server izabere da odgovori preko XML. Rad sa XML odgovorom zahteva parsiranje, DOM i stoga je malo komplikovaniji.

3.9. Pravljenje HEAD zahteva

Umesto korišćenja GET i POST zahteva kada server vraća tražene podatke, može se koristiti HEAD zahtev kada server vraća informacije o podacima pre nego što su oni poslati, kao na primer:

- poslednje vreme kada je sadržaj odgovora promenjen,
- da li zahtevan odgovor postoji,
- dužina odgovora,
- tip odgovora.

Evo kako se to može uraditi:

Kod 18. Primer pravljanja HEAD zahteva pomoću Ajax-a

```
function getSalesData() {
createRequest();
var url = "/boards/servlet/UpdateBoardSales";
request.open("HEAD", url, true);
request.onreadystatechange = updatePage;
request.send(null);
}
```

Na primer, štampanje svih informacija o odgovoru:

Kod 19. Štampanje svih zaglavlja odgovora iz HEAD zahteva

```
function updatePage() {
if (request.readyState == 4) {
alert(request.getAllResponseHeaders());
}
}
```

Svako od ovih zaglavlja može da se koristi nezavisno kako bi se obezbedile dodatne informacije ili funkcionalnosti unutar Ajax aplikacije. Na primer, informacija o dužini odgovora se dobija pomoću:

```
request.getResponseHeader("Content-Length")
```

4. DOM (Document Object Model)

4.1. Korišćenje DOM-a za Web odgovor

DOM je efektivan model koji omogućava da se radi istovremeno sa XML-om, na jednoj strani, i HTML-om sa druge strane.

Razmotrimo ukratko postupak koji se koristi pri pri dizajniranju, odnosno opsluživanju Web-strane:

1. Neko kreira HTML u tekst editoru ili u nekom specijalizovanom editoru,
2. Postavi se HTML na Web server, čime postaje javan na Internetu,
3. Korisnici zahtevaju tu stranu pomoću pretraživača kao što su Firefox, Internet Explorer itd.
4. Pretraživač korisnika pravi zahtev za HTML-dokumentom Web serveru,
5. Pretraživač generiše stranu koju dobije od servera grafički i tekstualno; korisnik gleda i aktivira Web stranu.

Struktura Web strane apsolutno može da se kontroliše. Stil i ponašanje (event handler i JavaScript) se primenjuju na ovu organizaciju naknadno. Pretraživač uzima svu tekstualnu organizaciju i pretvara je u set objekata, od kojih svaki može biti promenjen, dodat ili izbrisan.

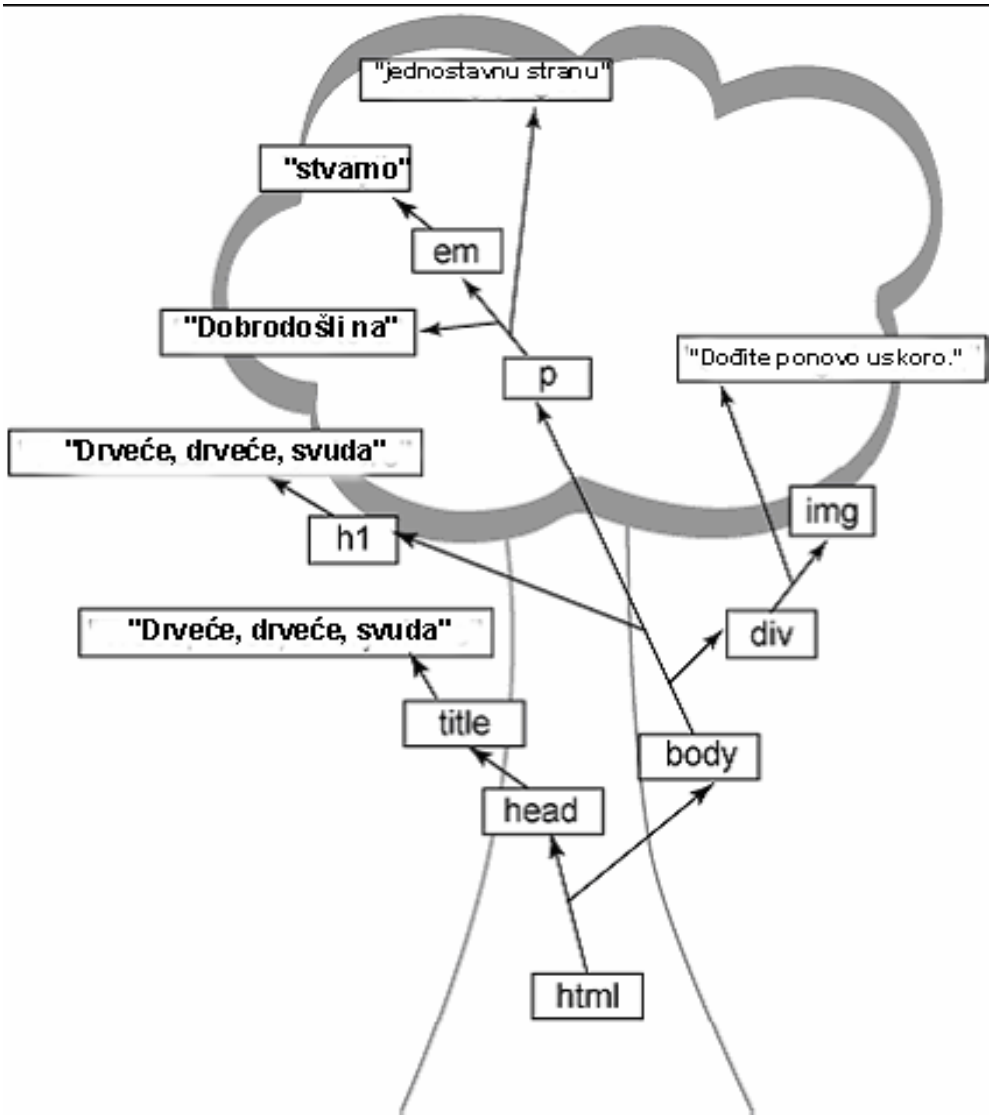
Reprezentacija objektima je DOM drvo. DOM specifikacija je podržana od strane World Wide Web Consortium (W3C). DOM definiše tipove objekata i osobine koje dopuštaju pretraživaču da prikaže obeležavanje (markiranje).

Neka imamo sledeću jednostavnu HTML stranu:

Kod 20. Jednostavna HTML strana u tekstualnom obeležavanju

```
<html>
<head>
<title>Drvece, drvece, svuda</title>
</head>
<body>
<h1> Drvece, drvece, svuda </h1>
<p>Dobrodosli na <em>stvarno</em> jednostavnu stranu.</p>
<div>
Dođite ponovo uskoro.
<img src="come-again.gif" />
</div>
</body>
</html>
```

Njena drvoidna reprezentacija može izgledati ovako:



Slika 4. Prikaz drveta Koda 20.

Sve prezentacije počinju <html> elementom koji predstavlja koren drveta. <head> i <body> elementi su deca <html> korenog elementa. Oni mogu biti roditeljski elementi za druge tj. mogu imati decu-elemente. Ekstremiteti drveta su listovi.

Svaka komponenta u prezentaciji dobija svoj tip objekta:

- elementi u HTML-u su predstavljeni **Element** tipom objekata,
- tekst u dokumentu je predstavljen **Text** tipom, atributi **Attribute** tipom.

HTML dokument je parsiran i pretvoren u set objekata kao na slici. Sada Web pretraživač može lako da promeni pojavljivanje i strukturu drveta korišćenjem osobina ovih objekata, za razliku od predstavljanja tekstom kada svaka promena osobine ili strukture zahteva da pretraživač prepiše statički fajl, parsira ga, i ponovo prikaže na ekranu. Sve u markiranju mora da bude pretvoreno u objekat nekog tipa. Pretraživač mora da zadrži redosled elemenata i teksta.

Atributi se smešteni u model objekta koji pretraživač koristi, ali su oni specijalni slučaj. Svaki element ima listu atributa koja je moguća, odvojena od liste dece objekata. Atributi (na primer, id, class) za element moraju da imaju jedinstvena imena tj. element ne moze da ima dva **id** ili dva **class** atributa. To takođe čini ovu listu različitu od liste objekata dece.

Dobro formatiran HTML-dokument olakšava konvertovanje označavanja u reprezentaciju drveta, i on je to ako je zadovoljeno:

- svaki otvarajući tag ima odgovarajući zatvarajući tag,
- poslednji otvarajući tag se poklapa sa prvim zatvarajućim tagom.

Modelu objekta se pristupa pomoću promenljive `document` u JavaScript kodu u Web strani: var domTree=document. Ovim Web pretraživač čini `document` objekat dostupan JavaScript kodu i objekat predstavlja kompletno drvo markiranja.

Svaki deo označavanja je predstavljen objektom, ali to je više od objekta - specifičan tip objekta, DOM čvor. Ostali tipovi(tekst, elementi, i atributi) su nasleđeni od osnovnog tipa čvora, pa postoje čvorovi teksta, čvorovi elementa, i čvorovi atributa.

Na primer, u liniji koda

```
var number=document.getElementById(''telefon'').value
```

se koristi DOM da se nađe specifični element i dobije vrednost tog elementa (u ovom slučaju polje forme). DOM drvo je drvo objekata tj. drvo čvorova objekata. U Ajax aplikacijama radom sa čvorovima mogu da se kreiraju efekti kao na primer, uklanjanje elementa i njegovog sadržaja, osvetljavanje odredjenog dela teksta, ili dodavanje novog slikovnog elementa. Pošto se ovo sve dešava na strani klijenta (kod koji se pokreće u Web pretraživaču), ovi efekti se dešavaju odmah, bez komunikacije sa serverom. Krajnji rezultat je često aplikacija koja omogućava brži odgovor jer se stvari na Web strani menjaju bez dugih pauza koje bi nastale usled slanja zahteva serveru i potrebe da se odgovor interpretira.

4.2. Manipulisanje DOM-om

Kada je HTML-dokument i CSS poslat Web pretraživaču, prevodi se iz tekstualnog modela u model objekata. Pretraživač dalje radi direktno sa modelom objekata.

DOM je W3C standard. DOM je cross-language specifikacija, što znači da može da se koristi iz većine popularnih programskih jezika. W3C definiše nekoliko jezičkih veza (bindings) za DOM. To je ustvari API definisan da nam omogući da se koristi DOM za specifičan jezik (postoji za jezike: C, Java, JavaScript).

Čvor je najosnovniji tip objekta u DOM-u. Svaki drugi tip objekta definisan pomoću DOM-a nasleđuje objekat čvor. Svaki element i tekst u DOM drvetu ima roditelja; roditelj je ili dete drugog elementa, ili je najvišlji element u DOM drvetu.

Osobine čvora su:

- **nodeName** - ime čvora,
- **nodeValue** - daje vrednost čvora,
- **parentNode** - vraća roditelja čvora (svaki element sem korenog, atribut, i tekst ima roditeljski čvor),
- **childNodes** - lista dece čvora (korisno samo kada se radi sa elementom),
- **firstChild** - prečica za prvi čvor u **childNodes** listi,
- **lastChild** - prečica za poslednji čvor u **childNodes** listi,
- **previousSibling** - vraća čvor pre trenutnog čvora (vraća čvor koji prethodi trenutnom, u **childNodes** listi roditelja),
- **nextSibling** - naredni čvor u roditeljevoj **childNodes** listi
- **attributes** - vraća listu atributa elementa.

NodeName i **nodeValue** nisu primenjivi na sve tipove čvorova:

- tekstualni čvor:
  NodeName - null
  NodeValue - tekst čvora
- čvor elementa:
  NodeName - ime elementa
  NodeValue - null
- čvor atributa:
  ima i **NodeName** i **NodeValue.**

Sledeći primer pokazuje upotrebu nekoliko osobina čvora.

Kod 21. Primer korišćenja osobina čvorova u DOM-u:

```
// Prve dve linije uzimaju DOM drvo trenutne Web strane, a zatim i <html> element za to DOM drvo
var myDocument = document;
var htmlElement = myDocument.documentElement;
// Koje je ime <html> elementa? "html"
alert("Koreni element strane je " + htmlElement.nodeName );
// Traži <head> element
var headElement = htmlElement.getElementsByTagName("head")[0];
if (headElement != null) {
alert("Nađen je head element strane " + headElement.nodeName );
// Štampa naslov strane
var titleElement = headElement.getElementsByTagName("title")[0];
if (titleElement != null) {
// Tekst će biti prvo dete čvora <title> elementa
var titleText = titleElement.firstChild;
// Možemo preuzeti tekst tektualnog čvora sa nodeValue
alert("Naslov strane je '" + titleText.nodeValue + "'");
}
// Posle <head> je <body>
var bodyElement = headElement.nextSibling ;
while (bodyElement.nodeName .toLowerCase() != "body") {
bodyElement = bodyElement.nextSibling ;
}
// Nađen je <body> element...
}
```

Metodi čvora, dostupni za sve tipove čvorova su:

- **insertBefore(newChild,referenceNode)** - ubacuje **newChild** čvor pre **referenceNode**; ovaj metod se poziva iz namenjog roditelja **newChild**,
- **replaceChild(newChild, oldChild)**- zamenjuje **oldChild** čvor **newChild** čvorom,
- **removeChild(oldChild)** - uklanja **oldChild** iz čvora funkcije koja je pokrenuta,
- **appendChild(newChild)** - dodaje **newChild** čvor čvoru funkcije koja je pokrenuta; **newChild** je dodat kraju dece ciljnog čvora,
- **hasChildNodes()**- vraća true ako čvor koji ga je pozvao ima decu, false ako nema,
- **hasAttributes()**- vraća true ako čvor koji ga je pozvao ima atribute, false ako nema.

Svi ovi metodi odnose se na decu čvorova.
Neki od prethodno opisanih metoda koriste se u sledećem primeru.

Kod 22. Primer korišćenja metoda čvorova u DOM-u

```
// Prve dve linije uzimaju DOM drvo trenutne Web strane, a zatim i <html> element za to DOM drvo
var myDocument = document;
var htmlElement = myDocument.documentElement;
// Ime <html> elementa? "html"
alert("Koreni element strane je " + htmlElement.nodeName);
// Traži <head> element
var headElement = htmlElement.getElementsByTagName("head")[0];
if (headElement != null) {
alert("Nađen je head element pod imenom" + headElement.nodeName);
// Štampa naslov strane
var titleElement = headElement.getElementsByTagName("title")[0];
if (titleElement != null) {
// Tekst će biti prvo dete čvora <title> elementa
var titleText = titleElement.firstChild;
// Možemo preuzeti tekst tektualnog čvora sa nodeValue
alert("Naslov strane je '" + titleText.nodeValue + "'");
}
// Posle <head> je <body>
var bodyElement = headElement.nextSibling;
while (bodyElement.nodeName.toLowerCase() != "body") {
bodyElement = bodyElement.nextSibling;
}
// Našli smo <body> element...
// Uklanja sve  <img> elemente najvišljeg nivoa u telu
if (bodyElement.hasChildNodes() ) {
for (i=0; i<bodyElement.childNodes.length; i++) {
var currentNode = bodyElement.childNodes[i];
if (currentNode.nodeName.toLowerCase() == "img") {
bodyElement.removeChild(currentNode) ;
}
}
}
}
```

U sledećem kodu se može videti kompletan HTML fajl za testiranje prethodna dva primera.

Kod 23. HTML fajl sa JavaScript kodom sa korišćenjem DOM-a

```
<html>
<head>
<title>JavaScript and the DOM</title>
<script language="JavaScript">
function test() {
// Prve dve linije uzimaju DOM drvo trenutne Web strane, a zatim i <html> element za to DOM drvo
var myDocument = document;
var htmlElement = myDocument.documentElement;
alert("Koreni element strane je " + htmlElement.nodeName);
var headElement = htmlElement.getElementsByTagName("head")[0];
if (headElement != null) {
alert("Nađen je head element pod imenom " + headElement.nodeName);
var titleElement = headElement.getElementsByTagName("title")[0];
if (titleElement != null) {
var titleText = titleElement.firstChild;
alert("Naslov strane je '" + titleText.nodeValue + "'");
}
var bodyElement = headElement.nextSibling;
while (bodyElement.nodeName.toLowerCase() != "body") {
bodyElement = bodyElement.nextSibling;
}
if (bodyElement.hasChildNodes()) {
for (i=0; i<bodyElement.childNodes.length; i++) {
var currentNode = bodyElement.childNodes[i];
if (currentNode.nodeName.toLowerCase() == "img") {
bodyElement.removeChild(currentNode);
}
}
}
}
}
</script>

</head>
<body>
<p>Neki tekst.</p>
<img src="http://www.matf.bg.ac.yu/slika1.jpg" />
<input type="button" value="Testiraj me!" onClick="test();" />
</body>
</html>
```

4.3. Tipovi čvorova

U većini Web aplikacija, najčešće se radi sa 4 tipa čvorova:

- čvor dokumenta: predstavlja ceo HTML document,
- čvor elementa: predstavlja HTML element (kao na primer, a ili img),
- čvor atributa: predstavlja atribute HTML elementa (na primer, href za a element, ili src za img element),
- čvor teksta: predstavlja tekst u HTML dokumentu (tekst koji se pojavljuje unutar elemenata kao što su p, a ili h2).

4.3.1. Čvor dokumenta

Čvor dokumenta zapravo nije element HTML (ili XML) strane, već same strane. Na HTML Web strani čvor dokumenta je celo DOM drvo. U JavaScriptu može se pristupiti čvoru dokumenta korišćenjem ključne reči document.

```
// Prve dve linije uzimaju DOM drvo trenutne Web strane, a zatim i <html> element za to DOM drvo
var myDocument = document ;
var htmlElement = myDocument.documentElement;
```

Document je ključna reč u JavaScript-u, vraća DOM drvo za trenutnu Web stranu, odakle može da se radi sa svim čvorovima u drvetu. **Document** objekat može da se koristi za pravljenje novih čvorova, korišćenjem metoda kao što su:

- **createElement(elementName)** - kreira element sa datim imenom,
- **createTextNode(text)** - kreira novi tekstualni čvor sa datim tekstom,
- **createAttribute(attributeName)** - kreira novi atribut sa datim imenom.

Ovi metodi kreiraju čvorove, ali ih ne dodaju ili ne ubacuju u odredjen dokument. Za to se koriste metodi **insertBefore()** ili **appendChild()**.

Na primer, kreiranje i dodavanje novog elementa dokumentu se može uraditi na sledeći način:

```
var pElement = myDocument.createElement("p");
var text = myDocument.createTextNode("Neki tekst u p elementu.");
pElement.appendChild(text);
bodyElement.appendChild(pElement);
```

4.3.2. Čvor elementa

Postoje dva seta metoda koji su specifični za čvor elementa (ostali su zajednički svim čvorovima):

1. Metodi koji se odnose na rad sa atributima:

- **getAttribute(name)** - vraća vrednost atributa **name**,
- **removeAttribute(name)** - vraća atribut koji se zove **name**,
- **setAttribute(name, value) -** kreira atribut koji se zove **name,** i postavlja mu vrednost na **value**,
- **getAttributeNode(name)** - vraća čvor atributa koji se zove **name,**
- **removeAttributeNode(node)** - uklanja čvor atributa koji se poklapa sa datim čvorom.

U sledećem primeru kreira se novi img element, postavljaju se atributi i dodaju telu HTML strane:

```
var imgElement = document.createElement("img");
imgElement.setAttribute("src", "http://www.headfirstlabs.com/Images/hraj_cover-150.jpg");
imgElement.setAttribute("width", "130");
imgElement.setAttribute("height", "150");
bodyElement.appendChild(imgElement);
```

2. Metodi koji se odnose na traženje ugnježdenih elemenata:

- **getElementsByTagName(elementName)** - vraća listu čvorova elemenata sa datim imenom.

Primer pronalaženja i uklanjanja svih img elemenata u HTML strani iz Koda 23 :

```
// Brisanje svih <img> elemenata najvišljeg nivoa u telu
if (bodyElement.hasChildNodes()) {
for (i=0; i<bodyElement.childNodes.length; i++) {
var currentNode = bodyElement.childNodes[i];
if (currentNode.nodeName.toLowerCase() == "img") {
bodyElement.removeChild(currentNode);
}}}
```

Isti efekat se može postići korišćenjem getElementsByTagName():

```
// Brisanje svih <img> elemenata najvišljeg nivoa u telu
var imgElements = bodyElement.getElementsByTagName("img");
for (i=0; i<imgElements.length; i++) {
var imgElement = imgElements.item[i];
bodyElement.removeChild(imgElement);
}
```

4.3.3. Čvorovi atributa

DOM predstavlja atribute kao čvorove i uvek mogu da se uzmu atributi elementa korišćenjem **attributes** osobine elementa, kao što je pokazano u primeru:

```
// Brisanje svih <img> elemenata najvišljeg nivoa u telu
var imgElements = bodyElement.getElementsByTagName("img");
for (i=0; i<imgElements.length; i++) {
var imgElement = imgElements.item[i];
//Štampanje nekih informacija o ovom elementu
var msg = " Nađen je img element!";
var atts = imgElement.attributes;
for (j=0; j<atts.length; j++) {
var att = atts.item(j);
msg = msg + "\n " + att.nodeName + ": '" + att.nodeValue + "'";
}
alert(msg);
bodyElement.removeChild(imgElement);}
```

Attributes osobina je na tipu čvora, a ne specifično na tipu elementa. Atributi nisu deca elemenata kao drugi elementi ili tekst, tj ne pojavljuju se kao članovi elementa. Ali, atributi imaju vezu sa elementom: element poseduje svoje atribute. DOM koristi čvorove da predstavi atribute i učini ih dostupnim elementu preko specijalne liste. Znači, atributi su deo DOM drveta, ali se često ne pojavljuju na drvetu.

Moguće je raditi sa čvorovima atributa, ali je lakše koristiti metode dostupne u klasi elemenata da se radi sa atributima:

- **getAttribute(name)** - vraća vrednost atributa **name**,
- **removeAttribute(name)** - uklanja atribut sa imenom **name,**
- **setAttribute(name, value)** - kreira atribut koji se zove **name** i postavlja mu vrednost na **value.**

4.3.4. Čvorovi teksta

Skoro sve osobine sa kojima se obično radi sa čvorovima teksta su dostupni na objektu čvora. Najčešće se koristi **nodeValue** osobina da se uzme tekst iz čvora teksta:

```
var pElements = bodyElement.getElementsByTagName("p");
for (i=0; i<pElements.length; i++) {
var pElement = pElements.item(i);
var text = pElement.firstChild.nodeValue;
alert(text);
}
```

Neke od metoda specifičnih za čvorove teksta (za dodavanje ili podelu podataka u čvoru):

- **appendData(text)** - dodaje tekst kraju postojećeg teksta tekstualnog čvora,
- **insertData(position, text**) - dopušta da se ubace podaci u sredinu tekstualnog čvora; ubacuje tekst u poziciju koja se naznači,
- **replaceData(position, length, text)** - uklanja karaktere počev od date pozicije, naznačene dužine, i stavlja dati tekst umesto uklonjenog.

Ukoliko se prolazi kroz DOM drvo, a ne zna se o kom tipu čvora se radi, to se može proveriti pomoću konstanti:

1. **Node.ELEMENT_NODE** je konstanta za tip čvora elementa.
2. **Node.ATTRIBUTE_NODE** je konstanta za tip čvora atributa.
3. **Node.TEXT_NODE** je konstanta za tip čvora teksta.
4. **Node.DOCUMENT_NODE** je konstanta za tip čvora dokumenta.

Može se koristiti i osobina **nodeType** kao u sledećem primeru:

```
var someNode = document.documentElement.firstChild;
if (someNode.nodeType == Node.ELEMENT_NODE) {
alert("Nađen je čvor element pod imenom" + someNode.nodeName);
} else if (someNode.nodeType == Node.TEXT_NODE) {
alert("To je tekstualni čvor; tekst je " + someNode.nodeValue);
} else if (someNode.nodeType == Node.ATTRIBUTE_NODE) {
alert("Ovo je atribut pod imenom " + someNode.nodeName+ " sa vrednošću '" +
                                                    someNode.nodeValue + "'");
}
```

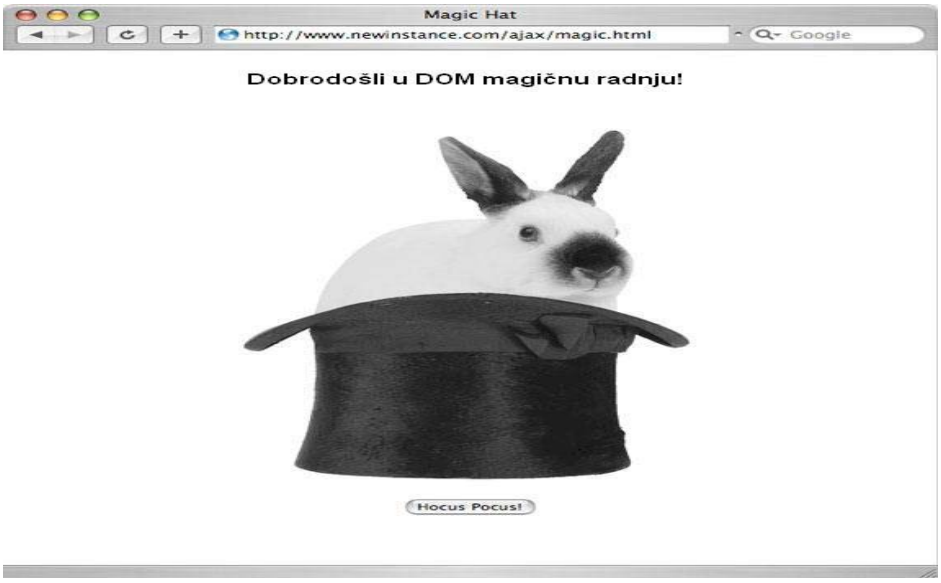
Gornje konstante nisu podržane od strane Internet Explorer-a.

4.4. Pravljenje DOM baziranih aplikacija

Pravljenje DOM aplikacija može se pokazati na jednostavnom primeru HTML strane koja prikazuje sliku šešira i dugmeta Hokus Pokus. Ako se klikne na ovo dugme prikazuje se nova slika šešira i zeca u njemu. Ideja je da DOM može da menja stvari na Web strani bez slanja forme.

Kod 24. Osnovna strana

```
<html>
<head>
<title>Čarobni šešir</title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
}
</script>
</head>
<body>
<h1 align="center">Dobrodošli u DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hocus Pocus!" />
</p>
</form>
</body>
</html>
```



Nekoliko napomena:

- dugme u kodu je tipa button a ne submit button. Korišćenjem normalnog dugmeta i izbegavanjem submit dugmeta, JavaScript funkcija se vezuje za dugme i obavlja komunikaciju sa pretraživačem bez slanja forme,
- id atribut je dodat u HTML element <img> kako bi kasnije moglo da se pristupa tom elementu,
- korišćenjem getElementById() metoda, dostupnom na objektu document koji predstavlja Web stranu, pronalazi se DOM čvor koji predstavlja img element.

Zamena slike na komplikovaniji način, se sastoji iz sledećih koraka:

1. Kreiranje novog img elementa:
   ```
   var newImage = document.createElement("img");
   newImage.setAttribute("src", "rabbit-hat.gif");
   ```
2. Pristupanje elementu koji je njemu roditelj trenutnog img elementa:
   ```
   var imgParent = hatImage.parentNode;
   ```
3. Ubacivanje novog img elementa kao dete roditelja, tačno pre postojećeg img elementa:
   ```
   imgParent.insertBefore(newImage, hatImage);
   ```
4. Uklanjanje starog img elementa:
   ```
   imgParent.removeChild(hatImage);
   ```
5. Podešavanje da se klikom na dugme pozove JavaScript funkcija:
   ```
   <input type="button" value="Hocus Pocus!" onClick="showRabbit();" />
   ```

Navedeni postupak se može videti u sledećem kodu.

Kod 25. Zamena stare slike sa novom:

```
<html>
<head>
<title> Čarobni šešir </title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
var newImage = document.createElement("img");
newImage.setAttribute("src", "rabbit-hat.gif");
var imgParent = hatImage.parentNode;
imgParent.insertBefore(newImage, hatImage);
imgParent.removeChild(hatImage);
}
</script>
</head>
<body>
<h1 align="center"> Dobrodošli na DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hokus Pokus!" onClick="showRabbit();/>
</p>
</form>
</body>
</html>
```

Zamena slike može se izvršiti na lakši način, uz pomoć replaceChild() metoda:

1. Kreira se novi img element.
2. Pristupa se elementu koji je roditelj trenutnog img elementa.
3. Zamenjuje se stari img element sa novim tek kreiranim.
4. Vrši se povezivanje sa JavaScript kodom.

Ovaj pristup je korišćen u sledećem kodu.

Kod 26. Zamena stare slike novom (u jednom koraku)

```
<html>
<head>
<title>Magični šešir</title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
var newImage = document.createElement("img");
newImage.setAttribute("src", "rabbit-hat.gif");
var imgParent = hatImage.parentNode;
imgParent.replaceChild(newImage, hatImage);
}
</script>
</head>
<body>
<h1 align="center"> Dobrodošli na DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hocus Pocus!" onClick="showRabbit();" />
</p>
</form>
</body>
</html>
```

Slika se najlakše može zameniti samo promenom src atributa postojeće slike.

```
hatImage.setAttribute("src", "rabbit-hat.gif");
```

To se može videti u sledećem kodu.

Kod 27. Promena src atributa

```
<html>
<head>
<title>Magični šešir</title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
hatImage.setAttribute("src", "rabbit-hat.gif");
}
</script>
</head>
<body>
<h1 align="center"> Dobrodošli na DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hokus Pokus!" onClick="showRabbit();" />
</p>
```

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

Ako se želi promena ispisa na dugmetu u zavisnosti od toga da li je zec u šeširu ili izvan njega, dodaje se id dugmetu da bi se referisalo dugme i tako mu se pristupalo, što se može videti u sledećem kodu.

Kod 28. Kompletirana Web strana

```
<html>
<head>
<title>Magični šešir</title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
hatImage.setAttribute("src", "rabbit-hat.gif");
var button = document.getElementById("hocusPocus");
button.setAttribute("value", "Vrati se u šešir!");
}
</script>
</head>
<body>
<h1 align="center"> Dobrodošli na DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hokus Pokus!" id="hocusPocus" onClick="showRabbit();" />
</p>
</form>
</body>
</html>
```

Proces je obrnut za vraćanje zeca:

```
function hideRabbit() {
var hatImage = document.getElementById("topHat");
hatImage.setAttribute("src", "topHat.gif");
var button = document.getElementById("hocusPocus");
button.setAttribute("value", "Hokus Pokus!");
}
```

Treba da se promeni i akcija koja se dešava kada piše nešto drugo na dugmetu(kada piše "Vrati se u šešir!" poziva se hideRabbit(), a kada je zec sakriven onda showRabbit()). Rukovodilac događajem(hendler) sa kojim se ovde radi je onClick. U JavaScript-u se referiše događaj “klik na dugme” korišćenjem onClick hendlera dugmeta. Događaj pokrenut dugmetom menja se definisanjem nove funkcije i povezivanjem sa onClick hendlerom.

```
button.onclick = myFunction;
```

Upotreba onClick hendlera dugmeta je prikazana u sledećem primeru.

Kod 29. Menjanje onClick funkcije dugmeta

```
<html>
<head>
<title>Magični šešir</title>
<script language="JavaScript">
function showRabbit() {
var hatImage = document.getElementById("topHat");
hatImage.setAttribute("src", "rabbit-hat.gif");
var button = document.getElementById("hocusPocus");
button.setAttribute("value", "Vrati se u šešir!");
button.onclick = hideRabbit;
}
function hideRabbit() {
var hatImage = document.getElementById("topHat");
hatImage.setAttribute("src", "topHat.gif");
var button = document.getElementById("hocusPocus");
button.setAttribute("value", "Hokus Pokus!");
button.onclick = showRabbit;
}
</script>
</head>
<body>
<h1 align="center"> Dobrodošli na DOM magičnu radnju!</h1>
<form name="magic-hat">
<p align="center">
<img src="topHat.gif" id="topHat" />
<br /><br />
<input type="button" value="Hokus Pokus!" id="hocusPocus" onClick="showRabbit();" />
</p>
</form>
</body>
</html>
```

5. KORIŠĆENJE XML-a (Extensible Markup Language) U ZAHTEVIMA I ODGOVORIMA

Iza imena XMLHttpRequest objekat se krije čisti HTTP protokol prenosa (Hypertext Transfer Protocol), dok je XML format podataka.

U asinhronim aplikacijama, postoje dve primene XML-a:

- slanje zahteva od strane Web strane ka serveru u XML formatu,
- primanje zahteva od servera u Web strani u XML formatu.

Prvo, slanje zahteva u XML-u zahteva da se formatira zahtev kao XML korišćenjem API-ja ili spajanjem teksta i slanjem rezultata serveru. Glavni posao je konstrukcija zahteva na način koji zadovoljava pravila XML-a tako da može biti razumljiv za server. Fokus je na XML formatu. Postoje podaci koji se šalju i samo treba da se prebace u XML zapis.

Druga upotreba je za primanje zahteva u XML-u. To zahteva da se uzme odgovor od servera i ekstrahuju podaci iz XML-a. Fokus je na podacima sa servera.

U 90% Web aplikacija koje se pišu, koristi se par ime/vrednost za slanje serveru. Na primer, ako korisnik unosi svoje ime i adresu u formu na Web strani, imaćemo sledeće podatke iz forme:

```
ime=Marko
prezime=Markovic
ulica=Knez Mihajlova 15
grad=Beograd
država=Srbija
poštanskiBroj=11000
```

Ako se koristi običan tekst za slanje ovih podataka serveru, može se koristiti kod iz sledećeg primera.

Kod 30. Slanje parova kao običan tekst.

```
function callServer() {
// Uzima grad i državu iz Web forme
var ime = document.getElementById("ime ").value;
var prezime = document.getElementById("prezime").value;
var ulica = document.getElementById("ulica").value;
var grad = document.getElementById("grad").value;
var država = document.getElementById("država").value;
var poštanskiBroj = document.getElementById("poštanskiBroj").value;
// Gradi Url za konekciju
var url = "/scripts/saveAddress.php? ime =" + escape(ime) +
"& prezime =" + escape(prezime) + "& ulica =" + escape(ulica) +
"& grad =" + escape(grad) + "& država =" + escape(država) +
"& poštanskiBroj=" + escape(poštanskiBroj);
// Otvara konekciju ka serveru
xmlHttp.open("GET", url, true);
// Postvlja funkciju koju server pokreće kad završi
xmlHttp.onreadystatechange = confirmUpdate;
// Šalje zahtev
xmlHttp.send(null);
}
```

Kada se konvertuju parovi ime/vrednost u XML, podatak treba da se zapiše u XML-formatu, gde je ime elementa ime para, a sadržaj elementa njegova vrednost. Neophodan je i koreni element:

```
<adresa>
<ime>Marko</ime>
<prezime>Markovic</prezime>
<ulica> Knez Mihajlova 15</ulica>
<grad> Beograd </grad>
<drzava>Srbija</drzava>
< poštanskiBroj >11000</ poštanskiBroj ></adresa>
```

Da bi skript prihvatio specifičan XML format i strukturu preko koje se šalju podaci, treba obezbediti da skript, kojem se šalje XML, prihvata XML kao format podataka. Bolje je koristiti POST zahtev za XML zbog ograničenja dužine stringa.

Dodavanje se vrši na sledeći način:

```
xmlHttp.setRequestHeader("Content-Type", "text/xml");
```

što govori da se šalje XML, a ne običan ime/vrednost par kao u slučaju sledeće komande:

```
xmlHttp.setRequestHeader("Content-Type", "text/plain");
```

Da bi se realizovalo slanje poziva se **send()** metod sa XML stringom. Server će dobiti XML zahtev, prihvatiti XML, parsirati ga i poslati nazad odgovor.

Kod 31. Slanje ime/vrednost parova u XML-u

```
function callServer() {
// Uzima grad i drzava iz Web forme
var fime = document.getElementById("ime").value;
var prezime = document.getElementById("prezime").value;
var ulica = document.getElementById("ulica").value;
var grad = document.getElementById("grad").value;
var drzava = document.getElementById("drzava").value;
var postanskiBroj = document.getElementById("postanskiBroj").value;
var xmlString = "<profile>" +
" <ime>" + escape(ime) + "</ime>" +
" <prezime>" + escape(prezime) + "</prezime>" +
" <ulica>" + escape(ulica) + "</ ulica >" +
" <grad>" + escape(grad) + "</ grad >" +
" <drzava>" + escape(drzava) + "</ drzava >" +
" <postanskiBroj>" + escape(postanskiBroj) + "</ postanskiBroj >" +
"</profile>";
// Gradi URL na koji se konektuje
var url = "/scripts/saveAddress.php";
// Otvara konekciju ka serveru
xmlHttp.open("POST", url, true);
// Govori serveru da šaljemo XML
xmlHttp.setRequestHeader("Content-Type", "text/xml");
// Postvlja funkciju koju server pokreće kad završi
xmlHttp.onreadystatechange = confirmUpdate;
// Šalje zahtev
xmlHttp.send(xmlString);
}
```

Mane korišćenja XML-a su:

- usporavanje i dodavanje kompleksnost,
- nije ga lako konstruisati - veća mogućnost greške,
- ne nudi mnogo prednosti nad običnim tekstom i ime/vrednost parom.

XML-podaci će se slati serveru samo u dva slučaja:

- ako se komunicira sa serverom koji jedino prihvata XML,
- ako se poziva remontni API koji prihvata samo XML ili SOAP zahteve.

U ostalim slučajevima bolje je koristiti običan tekst.

Prednost upotrebe XML-a dolazi prilikom vraćanja odgovora klijentu, pošto server ne može da šalje parove ime/vrednost na standardan način. Ako server odgovori aplikaciji slanjem stringa klijent nema standardizovan način da razdvoji parove i da unutar para razdvoji ime i vrednost. Podaci bi morali ručno da se parsiraju.

XML odgovor sa servera može se tretirati na 2 osnovna načina:

- kao običan tekst u XML formatu
- kao XML dokument, prezentovan kao DOM Document objekat.

U sledećem primeru je prikazano korišćenje XML odgovora kao običnog teksta, i zato se koristi responseText osobina **request** objekta.

Kod 32. Tretiranje XML-a kao normalan odgovor servera

```
function updatePage() {
if (request.readyState == 4) {
if (request.status == 200) {
var response = request.responseText;
// response ima XML odgovor od servera
alert(response);
}
}
}
```

Neka na primer, server vraća gledanost nekoliko emisija. Ako bi se štampala vrednost **response** promenljive dobilo bi se sledeće:

```
<gledanost><emisija><naslov>Naslov 1</naslov ><gledanost >6.5</ gledanost >
</emisija><emisija><naslov> Naslov 2</naslov><gledanost>14.2</gledanost></emisija><
emisija>
<naslov> Naslov 3</naslov><gledanost >9.1</ gledanost ></ emisija ></gledanost >
```

Rezultat u takvom formatu je prilično komplikovan za dalju upotrebu, pa se retko i koristi.

Druga opcija je da se umesto osobine **responseText** koristi osobina **responseXML** koja je dopuštena na **XMLHttpRequest.** Ovim se vraca odgovor servera u formi DOM dokumenta.

Kod 33. Tretiranje XML kao XML

```
function updatePage() {
if (request.readyState == 4) {
if (request.status == 200) {
var xmlDoc = request.responseXML;
// rad sa xmlDoc korišćenjem DOM
}}}
```

Sada postoji DOM Document i može da se radi sa njim kao sa bilo kojim drugim XML-dokumentom.

Kod 34. Tražimo sve "emisija" elemente

```
function updatePage() {
if (request.readyState == 4) {
if (request.status == 200) {
var xmlDoc = request.responseXML;
var showElements = xmlDoc.getElementsByTagName("emisija");
}
}
}
```

Sledeći kod pokazuje upotrebu DOM-a, XML-a i JavaScript-a za jednostavnu manipulaciju XML-om primljenog od servera.

Kod 35. Kretanje kroz sve "emisija" elemente

```
function updatePage() {
if (request.readyState == 4) {
if (request.status == 200) {
var xmlDoc = request.responseXML;
var showElements = xmlDoc.getElementsByTagName("emisija");
for (var x=0; x<showElements.length; x++) {
        var naziv = showElements[x].childNodes[0].value;
        var gledanost = showElements[x].childNodes[1].value;
}}}}
```

6. KORIŠĆENJE GOOGLE AJAX PRETRAGE POMOĆU API-ja

Javni API dozvoljavaju da se koriste programi koji su smešteni na nekom serveru uz korišćenje nečijih podataka. Sam API definiše kako se komunicira sa programom. Javni API za Google-ov motor pretrage omogućava zahteve pretrage, a Google-ov kod pretražuje Google-ove podatke i vraća rezultat programu.

Prvo se preuzme 'developer' ključ od Google-a sa Web adrese: code.google.com/apis/ajaxsearch. Klikne se na link "Sign up for a Google Ajax Search API key". Overi se saglasnost, ukuca adresa i klikne Generate API Key, sačeka par sekundi i dobije ključ. Svi detalji se mogu saznati iz Google API dokumentacija.

Konstrukcija, koja omogućava pretragu, je GSearchControla:

Kod 36. Kreiranje nove GSearchControl

```
function OnLoad() {
// Kreiranjee Google search kontrole
var searchControl = new GSearchControl();
...
}
```

Nova lokalna pretraga se podešava korišćenjem GlocalSearch - to je specijalna konstrukcija Google-a koja omogućava da se izvode bazične pretrage na određenoj lokaciji. Način korišćenja se može videti u sledećem primeru.

Kod 37. Podešavanje nove lokalne pretrage

```
function OnLoad() {
// Kreiranje kontrole za google pretragu
var searchControl = new GSearchControl();
// ovo omogucava da se podesi sta se pojavljuje u rezultatima pretrage
var localSearch = new GlocalSearch();
...
// precizira se Google lokacija gde želimo da lociramo pretragu
localSearch.setCenterPoint("Serbia");
...
}
```

Kod 38. Tipovi pretrage koji su omogućeni

```
function OnLoad() {
// Kreira se Google search kontrola
var searchControl = new GSearchControl();
// Ovo nam omogućava da podesimo šta se pojavljuje u rezultatima pretrage
var localSearch = new GlocalSearch();
searchControl.addSearcher(localSearch);
searchControl.addSearcher(new GwebSearch());
searchControl.addSearcher(new GvideoSearch());
searchControl.addSearcher(new GblogSearch());
// Specificiramo Google našu lokaciju da bazira pretragu u okolini
localSearch.setCenterPoint("Serbia");
...
}
```

Najčešći tipovi pretrage su:

- **GwebSearch**: objekat za pretragu Web-a,
- **GvideoSearch**: objekat koji traži video povezan za naš pojam pretrage,
- **GblogSearch**: pretražuje blogove.

Na kraju se poziva draw() metod. Ovom pozivu se dodeljuje DOM element u HTML-u.

```
searchControl.draw(document.getElementById("searchcontrol"));
```

Posle toga će u pretraživaču biti prikazana forma za Google pretragu.

Sledi ceo kod za jednostavnu Web aplikaciju Google-ove pretrage i kreiranje Search box-a.

Kod 39. HTML za jednostavnu Google aplikaciju pretrage

```
<html>
<head>
<title> Google Ajax API Aplikacija za pretragu</title>
<link href="http://www.google.com/uds/css/gsearch.css"
type="text/css" rel="stylesheet" />
<script
src="http://www.google.com/uds/api?file=uds.js&v=1.0&key=NAS KLJUC"
type="text/javascript"> </script>
<script language="Javascript" type="text/javascript">
function OnLoad() {
// Kreira se Google search kontrola
var searchControl = new GSearchControl();
// Ovo nam omogućava da podesimo šta se pojavljuje u rezultatima pretrage
var localSearch = new GlocalSearch();
searchControl.addSearcher(localSearch);
searchControl.addSearcher(new GwebSearch());
searchControl.addSearcher(new GvideoSearch());
searchControl.addSearcher(new GblogSearch());
// Specificiramo Google našu lokaciju da bazira pretragu u okolini
localSearch.setCenterPoint("Serbia");
// "Crta" kontrolu na HTML formi
searchControl.draw(document.getElementById("searchcontrol"));
}
</script>
</head>
<body onload="OnLoad()">
<div id="searchcontrol" />
</body>
</html>
```

Aktiviranjem prethodne aplikacije ukuca se ključ i dobija se najjednostavnija strana Google pretrage.

Kod 40. Dodavanje inicijalnog pojma za pretragu:

```
function OnLoad() {
// Kreira se Google search kontrola
var searchControl = new GSearchControl();
// Ovo nam omogućava da podesimo šta se pojavljuje u rezultatima pretrage
var localSearch = new GlocalSearch();
searchControl.addSearcher(localSearch);
searchControl.addSearcher(new GwebSearch());
searchControl.addSearcher(new GvideoSearch());
searchControl.addSearcher(new GblogSearch());
```

```
// Specificiramo Google našu lokaciju da bazira pretragu u okolini
localSearch.setCenterPoint("Serbia");
// "Crta" kontrolu na HTML formi
searchControl.draw(document.getElementById("searchcontrol"));
searchControl.execute("Christmas Eve");
}
```

Kada se ukuca pojam za pretragu i klikne Search dugme, primetiće se Ajax-ov odgovor: rezultati pretrage dolaze bez ponovnog učitavanja strane. XMLHttpRequest i getElementById() metod, DOM i manipulacija strane je skriveno u dve linije unutar HTML-a. Google upravlja fajlom koji se zove uds.js i koji ima unutar sebe ceo JavaScript neophodan da search box funkcioniše. U sledećem primeru se vidi korišćenje tog fajla.

Kod 41. Najvažniji JavaScript fajl

```
<script
src="http://www.google.com/uds/api?file=uds.js &v=1.0&key=[NAS GOOGLE KLJUC]"
type="text/javascript"> </script>
```

Takođe, skriveni kod za GSearchControl() objekat, kreiran je u onLoad() JavaScript funkciji. Sve što treba da se uradi je da se uključi kod iz sledećeg listinga:

Kod 42. Kreiranje GSearchControl objekta

```
// Kreiranje Google search kontrole
var searchControl = new GSearchControl();
```

HTML-sadržaj koji je neophodan je samo div tag sa ID elementom na koji naš JavaScript može da se referiše:

Kod 43. Ceo HTML neohodan za kreiranje kontrole za pretragu

```
<div id="searchcontrol" />
```

U pozadini, Google-ov kod radi kompleksnije stvari. Uds.js JavaScript fajl pravi lokalna podešavanja, rukuje nekim Google-ovim specifičnim zadacima, verifikuje Google ključ i puni dva druga skripta:

http://www.google.com/uds/js/locale/en/uistrings.js
http://www.google.com/uds/js/uds_compiled.js.

Ajax aplikacije nisu samo korišćenje XMLHttpRequest, već je to i način da se pristupi Web aplikacijama zasnovanim na asinhronosti.

7. KORIŠĆENJE JSON-a (JavaScript Object Notation) ZA TRANSFER PODATAKA

Kada se koristi ime/vrednost par ili XML, u osnovi se uzimaju podaci iz aplikacije korišćenjem JavaScripta i smeštaju se u format podataka. U ovim slučajevima JavaScript funkcioniše kao jezik za manipulaciju podacima, pomeranje i manipulisanje podacima sa Web forme i stavljanje u format koji jednostavno može biti poslat serverskom programu.

Međutim, ponekad je JavaScript više od jezika za formatiranje. U ovim slučajevima se koriste objekti u JavaScript jeziku da se predstave podaci i da se u pozadini pretvore podaci od Web podataka u zahteve. U ovim slučajevima ima više posla jer je potrebno estrahovati podatke iz objekata u JavaScriptu i zatim staviti te podatke u ime/vrednost parove XML-a. To je mesto gde JSON dolazi do izražaja: dozvoljava da se lako prebace JavaScript objekti u podatke koji mogu biti poslati kao deo zahteva (sinhronog ili asinhronog).

Pojednostavljeno, JSON dozvoljava da se transformiše set podataka predstavljenih JavaScript objektom u string koji se može lako proslediti od jedne funkcije drugoj, ili u slučaju asinhronih aplikacija, od Web klijenta programu na serverskoj strani JSON omogućava da se strukture predstave mnogo kompleksnije od parova ime/vrednost. Na primer, mogu se predstaviti nizovi i kompleksni objekti, za razliku od jednostavne liste ključeva i vrednosti.

JSON dolazi do izražaja kada postoji zajedno više ime/vrednost parova.
Prvo, može se kreirati zapis podataka, na primer,

```
{ "ime": "Marko", "prezime":"Markovic", "email": "mm@gmail.com " }
```

Sve tri vrednosti su deo istog zapisa.

U sledećem kodu može se videti dodeljivanje varijabli JSON podataka niza vrednosti:

```
var ljudi =
{ "programeri": [
{ "ime": " Marko ", "prezime":" Markovic ", "email": " mm@gmail.com " },
{ "ime ": "Petar", "prezime ":"petrovic", "email": " pp@gmail.com " }
],
"autori": [
{ " ime ": "Isaac", "lastName": "Asimov", "tip": "naucna fantastika" },
{ " ime ": "Tad", "lastName": "Williams", "tip": "fantazija" }
],
"muzicari": [
{ "ime": "Eric", "prezime": "Clapton", "instrument": "gitara" },
{ "ime": "Sergei", "prezime": "Rachmaninoff", "instrument": "klavir" }
]
}
```

Pristup podacima je jednostavan, kao na primer:

```
ljudi.programeri[0].prezime;
ljudi.autori[1].tip // Vrednost je "fantazija"
ljudi.muzicari[3].prezime // Nedefinisano
ljudi.programeri[2].ime // Vrednost je "Petar"
```

JSON podaci se mogu promeniti na sledeći način:

```
people.muzicari[1].prezime = "Rachmaninov";
```

Nekad može biti korisno imati JSON podatke u formi stringa. Konverzija se može izvršiti na sledeći način:

```
String newJSONtext = ljudi.toJSONString();
```

Kada postoji string, može se koristiti na primer kao string zahteva u Ajax aplikaciji.

Može se konvertovati bilo koji JavaScript objekat u JSON tekst:
String myObjectInJSON = myObject.toJSONString();

To je najveća razlika između JSON i ostalih tipova formata podataka. Sa JSON-om poziva se jednostavna funkcija i dobijaju se podaci, formatirani i spremni za upotrebu. Sa ostalim formatima podataka, mora da se vrši konverzija između sirovog podatka i formatiranog podatka. Krajnji rezultat je da, kad se već radi sa mnogo JavaScript objekata, JSON je dobra opcija za laku konverziju podataka u format koji je jednostavno poslati serveru.

Slanje JSON podataka na server može se izvršiti na dva načina:

1. Slanje JSON u ime/vrednost paru preko GET- zahteva

Najjednostavniji način za slanje JSON podataka serveru je da se konvertuju u tekst i pošalju kao ime/vrednost par. Da bi se izbegla pogrešna interpretacija praznina i ostalih karaktera, koristi se escape()-metod, koji rukuje praznim karakterima, i konvertuje ih u Web sigurne karaktere:

```
var url = "organizePeople.php?people=" + escape(people.toJSONString());
request.open("GET", url, true);
request.onreadystatechange = updatePage;
request.send(null);
```

Ovde je korišćen GET zahtev i on ima sledeće nedostatke:

- dužina podataka je možda veća od limita dužine stringa URL-a
- šalju se svi podaci kroz mrežu kao čist tekst, što može biti nesigurno.

2. Slanje JSON podataka preko POST zahteva

```
var url = "organizePeople.php?timeStamp=" + new Date().getTime();
request.open("POST", url, true);
request.onreadystatechange = updatePage;
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
request.send(people.toJSONString());
```

ContentType zaglavlje se podešava da kaže serveru koju vrstu podataka da očekuje. U ovom slučaju **application/x-www-form-urlencoded** obaveštava server da se šalje tekst koji se dobija iz normalne HTML forme.

Za interpretaciju JSON-podataka na serveru:

1. Treba naći JSON parser/toolkit/helper API za jezik koji se koristi za pisanje programa na strani servera.
2. Treba koristiti JSON parser/toolkit/helper API za uzimanje zahtevanih podataka od klijenta i prebacivanje u nešto što skript može da razume.

8. ZAKLJUČAK

Velika primena Ajax-a u mnoštvu novih aplikacija(Google, Gmail, Google Suggest, Google Maps, Flickr,...), vode ka zaključku da ova grupa tehnologija čini novi web model „Web 2.0.”. To je novogenerisani termin koji obuhvata i Ajax, a koji čine bogate Internet aplikacije.

Sajtovi Web 2.0 uključujuju neke od sledećih tehnika:

- CSS radi razdvajanja prezentacije od sadržaja,
- softver za wiki-je i forume za podršku korisnički definisanih sadržaja,
- alatke za blogove,
- objedinjavanje, spajanje i obaveštavanje pomoću RSS polja,
- XML i JSON bazirane API-je,
- tehnike za dinamične internet aplikacije, često zasnovane na Ajax-u.

Pojam Web 3.0. se koristi za hipoteze o budućem razvoju Web-a(World Wide Web). Pogledi na razvoj novog Web-a su različiti: neki smatraju da će tehnologija Semantic Web izmeniti način na koji se Web koristi i dovesti do novih mogućnosti veštačke inteligencije. Drugi sugerišu da će povećanje brzina Internet konekcija, modularne aplikacije, ili napredak u računarskoj grafici igrati ključnu ulogu u evoluciji Web-a.

Ono što je najzanimljivije kod Web-a je da bilo ko može da doprinese njegovom razvoju, a u kom pravcu će se razvijati zavisi od njegovih korisnika.

9. LITERATURA

1. Bruce W. Perry;
   Ajax trikovi; O'Reilly; 2006

2. Brett McLaughlin;
   Mastering Ajax Part 1-11; http://www.ibm.com/developerworks/webservices/

3. http://www.w3schools.com/default.asp

4. Svet kompjutera; Mala škola Ajax-a;
   http://www.sk.co.yu/2007/10/skum01.html

5. http://en.wikipedia.org/wiki/Ajax

PRILOG: Spisak skraćenica korišćenih u radu

API - Application Programming Interface
CSS - Cascade Style Sheet
DOM - Document Object Model
HTTP - Hypertext Transfer Protocol
HTML - Hypertext Markup Language
JSON - JavaScript Object Notation
RSS - Really Simple Syndication
URL - Uniform Resource Locator
W3C - World Wide Web Consortium
XML - Extensible Markup Language