Brad Dayley
Brendan Dayley
Caleb Dayley

Node.js, MongoDB i Angular

integrisane alatke za razvoj veb strana

DRUGO IZDANJE

Definitivni vodič za upotrebu steka MEAN za izradu veb strana

Node.js, MongoDB i Angular

Node.js je vodeće programsko okruženje, MongoDB je najpopularnija NoSQL baza podataka, a Angular je vodeći radni okvir za izloženi (front-end) razvoj koji je zasnovan na MVC-u. Zajedno čine potpuno integrisane alatke za razvoj veba, koje se jednostavno implementiraju i omogućavaju veb programerima da kreiraju sajtove visokih performansi i aplikacije koje su potpuno ugrađene u JavaScript, od servera do klijenta.

U ovom novom izdanju knjige *Node.Js, MongoDB i Angular integrisane alatke za razvoj veb strana*, koja je ažurirana za Angular 2 i naredne verzije, prikazano je kako se integrišu ove tri tehnologije u potpuna radna rešenja. Knjiga počinje konciznim, kristalno jasnim uputstvima za upotrebu svake tehnologije, a zatim se brzo prelazi na izradu uobičajenih veb aplikacija.

Naučićete kako da koristite Node.js i MongoDB da biste napravili više skalabilnih sajtova visokih performansi, kako da primenite Angularov inovativni MVC pristup za strukturiranje efikasnijih stranica i aplikacija i kako da koristite sve tri tehnologije zajedno da biste isporučili sledeću izvanrednu generaciju veb rešenja.

- Implementirajte visokoskalabilni i dinamički veb server, koristeći Node.js i Express.
- Implementirajte MongoDB skladište podataka za svoje veb aplikacije.
- Pristupite bazi podataka MongoDB iz Node.js JavaScript koda i komunicirajte sa njom.
- Naučite osnove TypeScripta.
- Definišite prilagođene Angular direktive koje proširuju HTML jezik.
- Izradite veb usluge na strani servera u JavaScriptu.
- Implementirajte usluge na strani klijenta koje mogu da komuniciraju sa Node.js veb serverom.
- Izradite dinamičke prikaze pregledača koji omogućavaju bogatu korisničku interakciju.
- Dodajte autentifikovane korisničke naloge i ugnežđene komponente komentara u svoje veb aplikacije i stranice.

Brad Dayley je viši softverski inženjer sa više od 20 godina iskustva u razvoju poslovnih aplikacija i veb interfejsa. On je dizajnirao i implementirao veliki broj aplikacija i servisa, od aplikacijskih servera, do kompleksnih veb aplikacija.
Brendan Dayley je programer veb aplikacija koji je napisao veliki broj aplikacija, koristeći JavaScript, TypeScript i Angular.
Caleb Dayley je student kompjuterskih nauka. Programira u jezicima JavaScript, Python i C# i veliki je ljubitelj platforme Unity.

Razvoj veb strana / Angular 2, Angular 4, Node.js 6, MongoDB 3

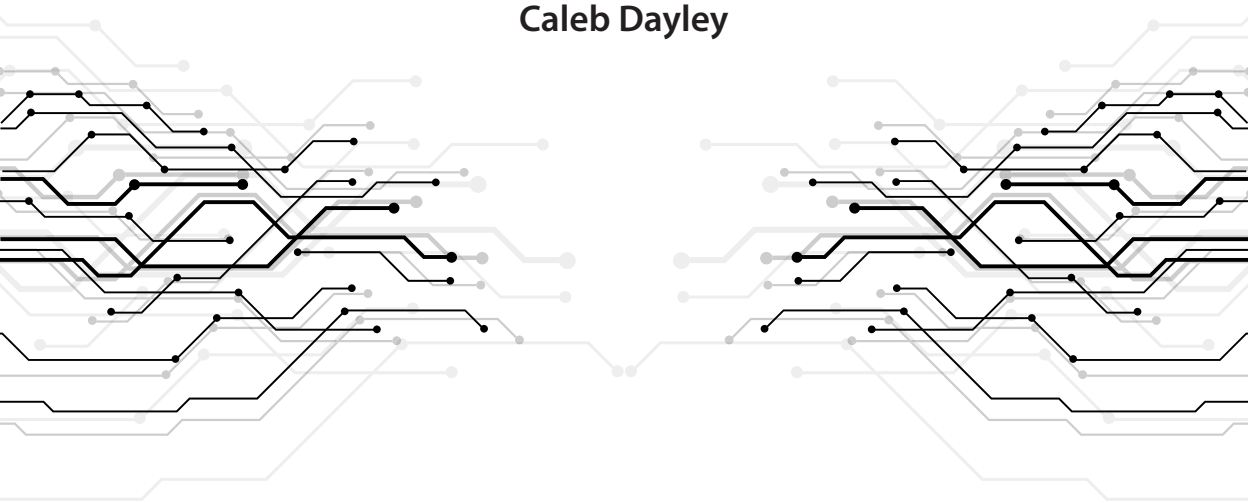
ISBN: 978-86-7310-521-5

Node.js, MongoDB i Angular

Integrisane alatke za razvoj veb strana

Prevod II izdanja

Brad Dayley
Brendan Dayley
Caleb Dayley



kompjuter biblioteka

Addison-Wesley

Izdavač:

kompjuter biblioteka

Obalskih radnika 15, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Brad Dayley
Brendan Dayley
Caleb Dayley

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2018.

Broj knjige: 498

Izdanje: Prvo

ISBN: 978-86-7310-521-5

Node.js, MongoDB and Angular Web Development, Second Edition

ISBN 978-0-13-465553-6

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд,
се добија на захтев

O AUTORIMA

Brad Dayley je viši softverski inženjer, sa više od 20 godina iskustva u razvoju poslovnih aplikacija i veb interfejsa. Godinama koristi JavaScript i jQuery i autor je knjiga *Learning Angular, jQuery and JavaScript Phrasebook* i *Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One*. Dizajnirao je i implementirao veliki broj aplikacija i usluga, od aplikacijskih servera, do kompleksnih veb aplikacija.

Brended Dayley je programer veb aplikacija, koji voli da uči i implementira najnovije i najbolje tehnologije. Koautor je knjiga *Learning Angular* i *Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One*. Napisao je veliki broj veb aplikacija, koristeći JavaScript, TypeScript i Angular, a istražuje nove veb i mobilne tehnologije, kao što je proširena realnost, i njihovu primenu u inovativnim rešenjima.

Caleb Dayley je student kompjuterskih nauka. Samostalno je naučio veći deo onoga što zna o programiranju i nekoliko programskih jezika, uključujući JavaScript i C#, pri čemu je koristio prvo izdanje ove knjige. Uzbuđen je zbog onoga što donosi budućnost i zbog šanse da pomogne u dizajniranju i kreiranju sledeće generacije inovativnog softvera koji će nastaviti da poboljšava način na koji živimo, radimo i zabavljamo se.

ZAHVALNICE

Koristim ovu stranicu da se zahvalim svima koji su mi pomogli u pisanju ove knjige. Prvo se zahvaljujem mojoj divnoj supruzi za inspiraciju, ljubav i podršku koju mi pruža. Nikada ne bih uspeo bez nje. Takođe se zahvaljujem mojim sinovima koji su mi pomogli u pisanju. Zahvaljujem se Marku Taberu što se pobrinuo da pisanje ove knjige ostane na pravom putu.

Brad Dayley

Zahvaljujem se svima koji su mi pomogli u pisanju ove knjige: pre svega, mojoj supruzi, koja me ohrabruje da napredujem i pruža mi svu svoju ljubav, mom ocu, koji je moj mentor u pisanju i programiranju, ali i u životu, i mojoj majci, koja je uvek bila uz mene kada mi je bila potrebna. I, na kraju, zahvaljujem se Marku Taberu, koji mi je pružio priliku da učestvujem u realizaciji ovog projekta.

Caleb Dayley

UVOD

Ova knjiga treba da vas uvede u svet korišćenja JavaScripta u projektima za razvoj veba - od servera i usluga, do klijenta pregledača. Knjiga „pokriva“ implementaciju i integraciju Node.js, MongoDB-a i Angulara, nekih od najuzbudljivijih i inovativnih tehnologija koje su se pojavile u svetu razvoja veba.

U ovom uvodu odgovorićemo na sledeća pitanja:

- Ko treba da čita ovu knjigu?
- Zašto treba da pročitate ovu knjigu?
- Šta ćete moći da postignete pomoću ove knjige?
- Šta su Node.js, MongoDB i Angular i zašto su to odlične tehnologije?
- Kako je organizovana ova knjiga?
- Gde da pronađete primere koda?
- Ko treba da čita ovu knjigu?

Ova knjiga je namenjena čitaocima koji već poznaju osnove HTML-a i koji su napisali neke programe na savremenom programskom jeziku. Poznavanje JavaScripta će olakšati razumevanje ove knjige, ali nije neophodno, zato što knjiga „pokriva“ osnove JavaScripta.

Zašto treba da pročitate ovu knjigu?

U ovoj knjizi ćete naučiti kako da kreirate moćne, interaktivne veb stranice i veb aplikacije - od veb servera i usluga na serveru, do interaktivnih veb aplikacija koje se zasnivaju na pregledaču. Tehnologije koje su ovde predstavljene su otvorenog koda i moći ćete da koristite JavaScript kako za komponente servera, tako i za pregledače.

Čitaoci ove knjige žele verovatno da nauče Node.js i MongoDB da bi napravili visokoskalabilne veb stranice visokih performansi. Oni takođe žele da koriste MVC/MVVM (Model-View-Controller/Model-View-View-Model) pristup Angulara za implementa-

ciju veb stranica koje su dobro dizajnirane i strukturirane. Node.js, MongoDB i Angular omogućavaju potpuno integrisani stek za razvoj veba, koji se lako implementira i omogućava implementiranje fenomenalnih veb aplikacija.

Šta ćete naučiti u ovoj knjizi?

U ovoj knjizi ćete naučiti kako da napravite dinamične veb sajtove i veb aplikacije u stvarnom svetu. Veb sajtovi više ne sadrže jednostavan statički sadržaj na HTML stranicama sa integrisanim slikama i formatiranim tekstom. Umesto toga, postali su mnogo dinamičniji, sa jednom stranicom koja često služi kao ceo sajt ili aplikacija.

Angular tehnologiju možete da ugradite u logiku veb stranice koja može da komunicira sa Node.js serverom i pribavi potrebne podatke iz MongoDB baze podataka. Kombinacija tehnologija Node.js, MongoDB i Angular omogućava da implementirate interaktivne i dinamičke stranice. Evo samo nekih od tema koje ćemo obraditi u ovoj knjizi:

- kako da implementirate visokoskalabilne i dinamičke veb servere, koristeći Node.js i Express
- kako da izradite veb usluge na serveru u JavaScriptu
- kako da implementirate MongoDB skladište podataka za veb aplikacije
- kako da pristupite MongoDB-u iz Node.js JavaScript koda i da komunicirate sa njim
- kako da definišete statičko i dinamičko rutiranje na vebu i da implementirate serverske skriptove
- kako da definišete prilagođene Angular komponente koje proširuju HTML jezik
- kako da implementirate usluge na strani klijenta koje mogu da komuniciraju sa Node.js veb serverom
- kako da napravite dinamičke prikaze pregledača koji omogućavaju bogatu korisničku interakciju
- kako da dodate ugrađene komponente na svoje veb stranice
- kako da implementirate Angular rutiranje za upravljanje navigacijom između prikaza klijentskih aplikacija

Šta je Node.js?

Node.js, koji se ponekad naziva samo Node, predstavlja razvojni radni okvir koji se zasniva na „Googleovoj“ V8 JavaScript mašini. Pišete Node.js kod u JavaScriptu, a zatim ga V8 kompajlira u mašinski kod koji se izvršava. Možete da napišete veći deo koda ili čak i ceo kod na strani servera u okruženju Node.js, uključujući skriptove veb servera, klijentske skriptove i skriptove bilo koje prateće funkcije veb aplikacije. Pošto se veb server

i skriptovi prateće veb aplikacije pokreću zajedno u istoj aplikaciji na strani servera, to omogućava mnogo veću integraciju između veb servera i skriptova.

Ovo su samo neki od razloga zbog kojih je Node.js odličan radni okvir:

- **JavaScript od jednog do drugog kraja** - Jedna od najvećih prednosti okruženja Node.js je što ono omogućava pisanje serverskih i klijentskih skriptova u JavaScriptu. Uvek je bilo poteškoća u odlučivanju da li logiku smestiti u klijentske ili serverske skriptove. Zahvaljujući okruženju Node.js, možete preuzeti JavaScript napisan na klijentu i lako prilagoditi serveru i obratno. Dodatna prednost je što klijentski i serverski programeri govore istim jezikom.
- **skalabilnost koja je vođena događajem** - Node.js primenjuje jedinstvenu logiku za upravljanje veb zahtevima. Umesto da koristi više programskih niti koje čekaju da obrade veb zahteve, okruženje Node.js obrađuje zahteve na istoj niti, koristeći osnovni model događaja. To omogućava Node.js veb serverima da skaliraju na način koji na tradicionalnim veb serverima nije moguć.
- **proširivost** - Novosti o Node.js prati veliki broj ljudi i postoji aktivna zajednica Node.js programera u kojoj se stalno nude novi moduli za proširenje funkcionalnosti okruženja Node.js. Osim toga, možete jednostavno da instalirate i uključite nove module u okruženju Node.js. Možete da proširite Node.js projekat da biste uključili nove funkcije za samo nekoliko minuta.
- **brza implementacija** - Podešavanje okruženja Node.js i programiranje u njemu su veoma jednostavni. Za samo nekoliko minuta možete da instalirate Node.js i da dobijete radni veb server.

Šta je MongoDB?

MongoDB je agilna i skalabilna NoSQL baza podataka. Naziv potiče od reči „hu**mongo**us“ (ogromno), kojom se naglašavaju skalabilnost i performanse MongoDB-a. MongoDB omogućava odlično pozadinsko skladište za veb sajtove gustog saobraćaja na kojima je potrebno uskladištiti podatke, kao što su korisnički komentari, blogovi ili druge stavke. Pozadinsko skladište se može brzo skalirati i lako implementirati.

Ovo su samo neki od razloga zbog kojih se MongoDB zaista uklapa u Node.js stek:

- **orijentacija dokumenta** - Pošto je MongoDB orijentisan ka dokumentaciji, podaci se čuvaju u bazama podataka u formatu veoma sličnom onome koji se koristi na serverskim i klijentskim skriptovima. Ovo eliminiše potrebu prenosa podataka iz redova na objekte i nazad.
- **visoke performanse** - MongoDB je jedna od dostupnih baza podataka najboljih performansi. U današnje vreme, kada sve više ljudi može da ostvari interakciju sa veb sajtovima, naročito je važno imati „pozadinu“ koja može da podrži gust mrežni saobraćaj.

- **visoka dostupnost** - Model replikacije MongoDB-a olakšava održavanje skalabilnosti, uz zadržavanje visokih performansi.
- **visoka skalabilnost** - Struktura MongoDB-a olakšava horizontalno skaliranje deljenjem podataka na više servera.
- **bez SQL injektovanja** - MongoDB baza podataka nije podložna SQL injektovanju (tj. stavljanju SQL iskaza u veb obrasce ili u druge ulazne podatke iz pregledača, čime se ugrožava bezbednost baze podataka). Razlog je činjenica da se objekti skladište kao objekti, bez upotrebe SQL stringova.

Šta je Angular?

Angular je JavaScript radni okvir na strani klijenta, koji je razvila kompanija „Google“. On omogućava okruženje koje olakšava implementaciju dobro dizajniranih i strukturiranih veb stranica i aplikacija, pri čemu se koristi MVC/MVVM radni okvir.

Angular obezbeđuje funkcije za upravljanje korisničkim unosom u pregledaču, manipuliše podacima na strani klijenta i kontroliše kako su elementi prikazani u pregledaču.

Ovo su samo neke od prednosti Angulara:

- **povezivanje podataka** - Angular na jednostavan način povezuje podatke sa HTML elementima, koristeći moćni mehanizam oblasti važenja.
- **proširivost** - Angular arhitektura dozvoljava lako proširenje skoro svakog aspekta jezika da bi bile omogućene prilagođene implementacije.
- **čist kod** – U Angularu ste primorani da napišete čist, logički kod.
- **kod za ponovnu upotrebu** - Kombinacija proširivosti i čistog koda olakšava pisanje koda za ponovnu upotrebu. U stvari, u Angularu ste primorani da napišete kod za ponovnu upotrebu prilikom kreiranja prilagođene usluge.
- **podrška** - Kompanija „Google“ mnogo ulaže u Angular projekat i zato je on u prednosti nad sličnim inicijativama koje nisu uspele.
- **kompatibilnost** – Angular je zasnovan na JavaScriptu i usko je povezan sa JavaScript standardom. To olakšava početak integracije Angulara u razvojno okruženje i ponovno korišćenje delova postojećeg koda unutar strukture radnog okvira Angular.

Organizacija ove knjige

Ova knjiga je podeljena na šest glavnih delova:

Deo I, „*Početak rada*“, sadrži pregled interakcije između alatki Node.js, MnogoDB i Angular, a predstavljen je i način na koji ove tri alatke formiraju kompletan stek za razvoj veba. U Poglavlju 2 opisane su osnove JavaScript jezika, koje su potrebne prilikom implementiranja koda za Node.js i Angular.

Deo II, „*Učenje okruženja Node.js*“, „pokriva“ Node.js jezičku platformu, od instalacije, do implementacije Node.js modula. U ovom delu opisan je osnovni radni okvir koji je potreban da biste implementirali prilagođene Node.js module i veb server i skriptove na strani servera.

Deo III, „*Učenje baze podataka MongoDB*“, „pokriva“ MongoDB bazu podataka, od instalacije, do integracije pomoću Node.js aplikacija. U ovom delu ćete naučiti kako da planirate model podataka koji će se uklopiti u potrebe vaše aplikacije, kako da pristupite MongoDB-u iz Node.js aplikacija i kako da komunicirate sa njim.

U Delu IV, „*Upotreba Expressa za lakši rad*“, razmotreni su Express modul za Node.js i način kako se primenjuje taj modul kao veb server za aplikaciju. Naučićete kako da podesite dinamičko i statičko rutiranje podataka i kako da implementirate bezbednost, keširanje (caching) i druge osnovne funkcije veb servera.

U Delu V, „*Učenje Angulara*“, razmotreni su arhitektura radnog okvira Angular i način na koji ona može da se integriše u Node.js stek. Ovaj deo „pokriva“ kreiranje prilagođenih HTML komponenata i usluga na strani klijenta koje se mogu primeniti u pregledaču.

Deo VI, „*Napredni Angular*“, „pokriva“ naprednije programiranje pomoću Angulara, kao što je kreiranje prilagođenih direktiva i servisa. U njemu ćete takođe naučiti kako se koriste Angularovi ugrađeni HTTP servisi i servisi rutiranja. Na kraju ovog dela dati su neki dodatni primeri bogatog korisničkog interfejsa, kao što su izrada drag-and-drop komponenata i implementiranje animacija.

Primeri koda za ovu knjigu

Primere koda za ovu knjigu ćete pronaći u listinzima. Naslov svakog listinga sadrži naziv datoteke za izvorni kod. Izvorni kod je dostupan za preuzimanje na veb sajtu izdavača.

Završna reč

Nadamo se da ćete uživati u učenju tehnologija Node.js, MongoDB i Angular onoliko koliko smo i mi uživali. Ovo su odlične inovativne tehnologije koje je zabavno koristiti. Uskoro ćete moći da se pridružite mnogim drugim veb programerima koji koriste veb stek od Node.js do Angulara za izradu interaktivnih veb sajtova i veb aplikacija.

1

Upoznavanje steka od Node.js do Angulara

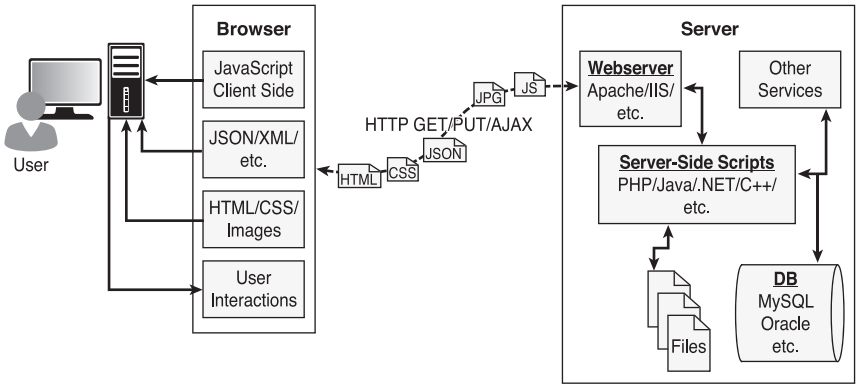
U ovom poglavlju se fokusiramo na osnovne komponente radnog okvira za razvoj veba, a zatim opisujemo komponente steka od Node.js do Angulara, koje će biti osnova za ostatak knjige. U prvom odeljku razmatramo različite aspekte radnog okvira za razvoj veb sajtova/aplikacija, od korisnika, do pozadinskih servisa. Prvo ćemo razmotriti komponente radnog okvira za razvoj veba da biste lakši razumeli kako su komponente steka od Node.js do Angulara povezane sa delovima opšteg radnog okvira. Ovo će vam pomoći da bolje uvidite prednosti korišćenja komponente steka od Node.js do Angulara nad korišćenjem tradicionalnih tehnologija.

Razumevanje osnovnog radnog okvira za razvoj veba

Da biste pravilno razumeli prednosti korišćenja tehnologija Node.js, MongoDB i Angular kao veb okvira, u ovom odeljku dajemo pregled osnovnih komponenata većine veb sajtova. Ako poznajete kompletno veb okruženje, sadržaj u ovom odeljku će za vas biti već poznat, ali ako razumete samo serversku ili klijentsku stranu veb okvira, ovaj odeljak vam daje kompletniju „sliku".

Glavne komponente svakog veb okvira su korisnik, pregledač, veb server i pozadinski servisi. Iako se sajtovi značajno razlikuju po izgledu i ponašanju, svi oni sadrži ove osnovne komponente u nekom obliku.

Ovaj odeljak neće biti detaljan, sveobuhvatan, niti tehnički precizan, već će u njemu biti predstavljeni delovi koji su uključeni u funkcionalni veb sajt. Komponente su opisane od korisnika do pozadinskog servisa. U narednom odeljku razmatramo stek od Node.js do Angulara odozdo prema gore da biste razumeli gde koji deo pripada i zašto. Na slici 1.1 je prikazan osnovni dijagram da biste lakše vizuelizovali komponente na veb sajtu/aplikaciji.



Slika 1.1 Dijagram na kome su prikazane komponente osnovnog veb sajta/aplikacije

Korisnik

Korisnici su osnovni deo svih veb sajtova. Na kraju krajeva, veb sajtovi postoje zbog njih. Očekivanja korisnika koja definišu uslove za razvoj dobrog veb sajta mnogo su se promenila tokom godina. Korisnici su ranije prihvatali spor i glomazan World Wide Wait, ali više ga ne prihvataju. Oni očekuju da se veb sajtovi „ponašaju“ slično kao aplikacije koje su instalirane na njihovim računarima i mobilnim uređajima.

Korisnici očekuju vizuelni izlaz i unos interakcija na veb stranice. To znači da oni pregledaju rezultate obrade veb okruženja, a zatim ostvaruju interakciju, koristeći klikove miša, unos sa tastature i brzo prevlačenje prstom i dodire na ekranima mobilnih uređaja.

Pregledač

Pregledač „igra“ tri uloge u veb okruženju. Prvo, omogućava komunikaciju ka veb serveru i od veb servera. Drugo, interpretira podatke sa servera i prikazuje ih na ekranu koji korisnik vidi. Na kraju, pregledač obrađuje korisničku interakciju pomoću tastature, miša, ekrana osetljivog na dodir ili drugih ulaznih uređaja i preduzima odgovarajuće radnje.

Komunikacija od pregledača do veb servera

Komunikacija od pregledača do veb servera se sastoji od niza zahteva pomoću protokola HTTP i HTTPS. Protokol za prenos hiperteksta (HTTP - Hypertext Transfer Protocol) definiše komunikaciju između pregledača i veb servera. On definiše tipove zahteva koji mogu biti podneti, format tih zahteva i HTTP odgovor.

HTTPS primenjuje dodatni bezbednosni sloj SSL/TLS da bi bile omogućene bezbedne veze, tako što od veb servera zahteva da obezbedi sertifikat za pregledač. Korisnik može da odluči da li će prihvatiti sertifikat pre nego što omogući vezu.

Pregledač podnosi tri glavna tipa zahteva serveru:

- **GET** - GET zahtev se, obično, koristi za preuzimanje podataka sa servera, kao što su .html datoteke, slike ili JSON podaci.
- **POST** - POST zahtevi se koriste prilikom slanja podataka na server – na primer, prilikom dodavanja artikala na kupovnu karticu ili slanja veb obrazaca.
- **AJAX** - Asinhroni JavaScript i XML (AJAX) je, zapravo, samo GET ili POST zahtev koji direktno šalje JavaScript zahteve pregledaču. Uprkos nazivu, AJAX zahtev može da u odgovoru primi XML, JSON ili neobrađene podatke.

Vizuelizacija prikaza u pregledaču

Ekran koji korisnik vidi i sa kojim često komunicira sastoji se od nekoliko različitih podataka koji su preuzeti sa veb servera. Pregledač čita podatke iz početnog URL-a, a zatim prikazuje HTML dokument da bi izradio objektni model dokumenta (DOM - Document Object Model). DOM je objekat strukture stabla sa HTML dokumentom kao korenom. Struktura stabla u osnovi odgovara strukturi HTML dokumenta. Na primer, dokument će imati html kao podređeni element, html će imati „glavu“ i telo kao podređene elemente, a telo može sadržati `div`, `p` ili druge elemente kao podređene elemente na sledeći način:

```
document
  + html
    + head
    + body
      + div
        + p
```

Pregledač interpretira svaki DOM element i prikazuje ga na ekranu korisnika da bi vizuelizovao prikaz veb stranice.

Pregledač često dobija različite vrste podataka iz više zahteva veb servera da bi bila izrađena veb stranica. Ovo su tipovi podataka koje pregledač najčešće koristi za vizuelizovanje krajnjeg korisničkog prikaza i definisanje ponašanja veb stranice.

- **HTML datoteke** - Ove datoteke omogućavaju osnovnu strukturu DOM-a.
- **CSS datoteke** - Ove datoteke definišu kako treba da se stilizuje svaki element na stranici, kao što su font, boja, ivice i razmak.
- **klijentski skriptovi** - Ovo su, obično, datoteke JavaScripta. One mogu da omoguće funkcije veb stranice, da manipulišu DOM-om radi promene izgleda veb stranice i da obezbede svu neophodnu logiku potrebnu za prikaz stranice i za funkcionalnost.
- **medijske datoteke** - Slike, video i zvučne datoteke se prikazuju kao deo veb stranice.
- p**odaci** - Svi podaci, kao što su XML, JSON ili neobrađeni tekst, mogu biti dostupni na veb serveru kao odgovor na AJAX zahtev. Umesto da zahtev pošaljete nazad serveru radi ponovne izrade veb stranice, nove podatke možete da preuzmete pomoću AJAX-a i da ih umetnete na veb stranicu pomoću JavaScripta.

- **HTTP zaglavlja** - HTTP protokol definiše skup zaglavlja koje mogu da koriste pregledač i klijentski skriptovi za definisanje „ponašanja“ veb stranice. Na primer, „kolačići“ su sadržani u HTTP zaglavljima. HTTP zaglavlja definišu takođe tip podataka u zahtevu i tip podataka koji se očekuje da bude vraćen nazad u pregledač.

Korisnička interakcija

Korisnik komunicira sa pregledačem pomoću ulaznih uređaja, kao što su miš, tastatura i ekrani osetljivi na dodir. Pregledač ima izgrađen sistem događaja koji beleži ove događaje korisničkog unosa i onda preduzima odgovarajuće radnje. Te radnje mogu biti prikazivanje iskačućeg menija, učitavanje novog dokumenta sa servera ili izvršavanje JavaScripta na strani klijenta.

Veb server

Glavni fokus veb servera je na upravljanju zahtevima iz pregledača. Kao što je ranije opisano, pregledač može da zahteva dokument, da objavi podatke ili da izvrši AJAX zahtev, radi dobijanja podataka. Veb server koristi HTTP zaglavlja i URL adresu da bi utvrdio koje radnje treba da preduzme. Radnje zavise od veb servera, konfiguracije i tehnologija koje se koriste.

Većina uobičajenih veb servera, kao što su Apache i IIS, napravljena je da služi statičkim datotekama, kao što su .html, .css i medijske datoteke. Radi upravljanja POST zahtevima koji modifikuju podatke servera i AJAX zahtevima za interakciju sa pozadinskim servisima, veb servere treba proširiti pomoću serverskih skriptova.

Serverski programi predstavljaju sve što veb server može da izvrši da bi obavio zadatak koji pregledač zahteva. Mogu biti napisani u jezicima PHP, Python, C, C ++, C#, Java itd. Veb serveri, kao što su Apache i IIS, sadrže mehanizme za uključivanje serverskih skriptova, koje, zatim, povezuju sa određenim URL lokacijama koje zahteva pregledač.

Stabilan radni okvir veb servera može da bude od velikog značaja. Često je potrebno sasvim malo konfiguracije da bi bili omogućeni različiti skriptni jezici i povezivanje serverskih skriptova tako da veb server može uputiti odgovarajući zahtev odgovarajućem skriptu.

Serverski skriptovi generišu odgovor direktno izvršavanjem koda ili povezivanjem sa drugim pozadinskim serverima, kao što su baze podataka za dobijanje potrebnih informacija, a zatim upotrebljavaju te informacije da izrade i pošalju odgovarajući odgovor.

Pozadinski servisi

Pozadinski servisi se pokreću u pozadini veb servera i sadrže podatke koji se koriste za kreiranje odgovora u pregledaču. Tip pozadinskih servisa koji se najčešće koristi je baza podataka koja skladišti informacije. Kada pregledač pošalje zahtev u kome se zahtevaju informacije iz baze podataka ili drugog pozadinskog servera, serverski skript se povezuje sa bazom podataka, preuzima i formatira informacije, a zatim ih šalje nazad pregledaču. I obratno - kada podaci stignu iz zahteva na vebu, koji se moraju uskladištiti u bazu podataka, serverski skript se povezuje sa bazom podataka i ažurira podatke.

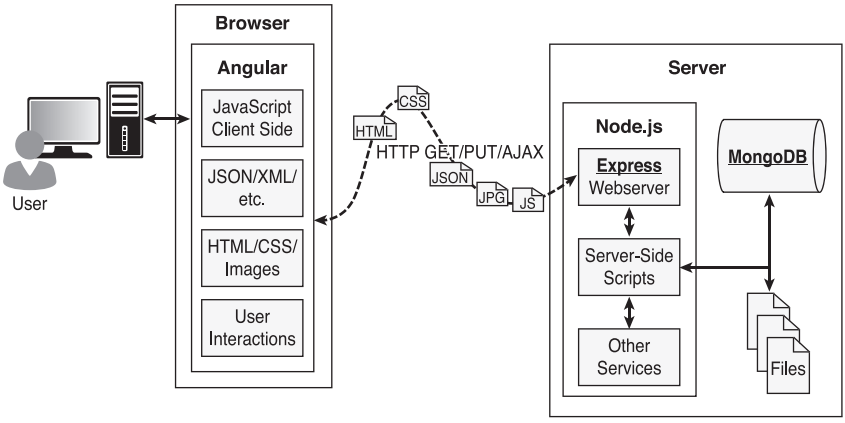
Razumevanje komponenata steka od Node.js do Angulara

Sada, kada je osnovna struktura veb okvira još uvek sveža u vašem pamćenju, vreme je da razmotrimo stek od Node.js do Angulara. Najzastupljenija i najbolja verzija ovog steka je stek od Node.js do Angulara, koji je sastavljen od alatki MongoDB, Express, Angular i Node.js.

U steku od Node.js do Angulara Node.js omogućava osnovnu platformu za razvoj. Pozadinski servisi i serverski skriptovi su napisani u okruženju Node.js. Baza podataka MongoDB omogućava skladištenje podataka veb sajta, ali pristupa joj se pomoću MongoDB drajvera modula Node.js. Veb server se definiše pomoću Expressa, koji je, takođe, Node.js modul.

Prikaz u pregledaču se definiše i kontroliše pomoću radnog okvira Angular. Angular je MVC radni okvir u kome je model sastavljen od JSON ili JavaScript objekata, prikaz je HTML/CSS, a kontroler se sastoji se od Angular JavaScripta.

Na slici 1.2 prikazan je osnovni dijagram uklapanja steka od Node.js do Angulara u osnovni model veb sajta/aplikacije. U sledećim odeljcima je opisana svaka od ovih tehnologija i objašnjeno je zašto su one izabrane kao deo steka od Node.js do Angulara. Naredna poglavlja u ovoj knjizi „pokrivaju“ svaku od ovih tehnologija mnogo detaljnije.



Slika 1.2 Osnovni dijagram na kojem je prikazano gde se Node.js, Express, MongoDB i Angular uklapaju u veb paradigmu

Node.js

Node.js je razvojno okruženje koje se zasniva na „Googleovoj“ V8 JavaScript mašini. Zbog toga se Node.js kod piše u JavaScriptu, a zatim se pomoću te mašine kompajlira u mašinski kod koji se izvršava.

Mnogi pozadinski servisi mogu biti napisani u okruženju Node.js, kao i serverski skriptovi i sve podržane funkcije veb aplikacija. Dobro je što se Node.js, u stvari, zasniva na JavaScriptu, pa možete jednostavno preuzeti funkcije iz klijentskog skripta i smestiti ih u serverski skript. Osim toga, veb server može da funkcioniše direktno u okviru platforme Node.js kao Node.js modul, pa mnogo jednostavnije povezuje nove servise ili skriptove sa servera nego Apache.

Ovo su samo neki od razloga zbog kojih je Node.js odličan radni okvir:

- **JavaScript od jednog do drugog kraja** - Jedna od najvećih prednosti okruženja Node.js je što omogućava pisanje serverskih i klijentskih skriptova u JavaScriptu. Uvek je bilo poteškoća u odlučivanju da li logiku smestiti u klijentske ili serverske skriptove. Kada klijent izvršava previše zadataka, postaje glomazan i nezgrapan, ali isto tako kada server izvršava previše zadataka, usporava veb aplikacije i veoma mnogo opterećuje server. Zahvaljujući okruženju Node.js, možete preuzeti JavaScript napisan na klijentu i lako ga prilagoditi serveru i obratno. Osim toga, klijentski i serverski programeri govore istim jezikom.
- **skalabilnost koja je vođena događajem** - Node.js primenjuje jedinstvenu logiku za upravljanje veb zahtevima. Umesto da koristi više programskih niti koje čekaju da obrade veb zahteve, okruženje Node.js obrađuje zahteve na istoj niti, koristeći osnovni model događaja. Ovo omogućava Node.js veb serverima da izvrše skaliranje na način koji nije moguć na tradicionalnim veb serverima. Skalabilnost koja je vođena događajima će biti razmotrena detaljnije u sledećim poglavljima.
- **proširivost** - Novosti o Node.js prati veliki broj ljudi i postoji aktivna zajednica Node.js programera, u kojoj se stalno nude novi moduli za proširenje funkcionalnosti okruženja Node.js. Osim toga, možete jednostavno da instalirate i uključite nove module u okruženju Node.js. Možete da proširite Node.js projekat da biste uključili nove funkcije za samo nekoliko minuta.
- **vreme** - Za samo nekoliko minuta možete da instalirate Node.js i da dobijete radni veb server.

MongoDB

MongoDB je agilna i skalabilna NoSQL baza podataka. Naziv potiče od reči „hu**mongo**us“ (ogromno) – njome se naglašavaju skalabilnost i performanse MongoDB-a, koji se zasniva na NoSQL modelu skladištenja dokumenta, što znači da se podaci skladište u bazi podataka u obliku JSON objekata, umesto u obliku tradicionalnih kolona i redova relacione baze podataka.

MongoDB omogućava odlično pozadinsko skladište za veb sajtove sa gustim saobraćajem na kojima je potrebno uskladištiti podatke, kao što su korisnički komentari, blogovi ili druge stavke. Pozadinsko skladište se može brzo skalirati i lako implementirati. Ova knjiga „pokriva“ biblioteku MongoDB drajvera za pristup MongoDB-a iz okruženja Node.js.

Node.js podržava različite upravljačke programe za pristup DB drajverima, tako da spremište podataka isto tako lako može da bude MySQL ili neka druga baza podataka. Ovo su neki od razloga zbog kojih se MongoDB zaista dobro uklapa u Node.js stek:

- **orijentacija dokumenta** - Pošto je MongoDB orijentisan ka dokumentu, podaci se čuvaju u bazama podataka u formatu koji je veoma sličan onome koji se koriste na serverskim i klijentskim skriptovima. To eliminiše potrebu prenosa podataka iz redova na objekte i nazad.
- **visoke performanse** - MongoDB je jedna od dostupnih baza podataka sa najvišim performansama. U današnje vreme, kada sve više ljudi može da ostvari interakciju sa

veb sajtovima, veoma je važno imati „pozadinu“ koja može da podrži gust mrežni saobraćaj.

- **visoka dostupnost** - Model replikacije MongoDB-a olakšava održavanje skalabilnosti, uz zadržavanje visokih performansi.
- v**isoka skalabilnost** - Struktura MongoDB-a olakšava horizontalno skaliranje deljenjem podataka na više servera.
- b**ez SQL injektovanja** - MongoDB baza podataka nije podložna SQL injektovanju (tj. stavljanju SQL iskaza u veb obrasce ili u druge ulazne podatke iz pregledača, čime se ugrožava bezbednost baze podataka). Razlog je činjenica da se objekti skladište kao objekti, bez upotrebe SQL stringova.

Express

Modul Express se ponaša kao veb server u steku od Node.js do Angulara. Express funkcioniše u okruženju Node.js, čime se olakšavaju konfiguracija, implementacija i kontrola. Modul Express proširuje Node.js da bi obezbedio nekoliko važnih komponenata za upravljanje veb zahtevima. To omogućava da se implementira radni veb server u okruženju Node.js pomoću nekoliko redova koda.

Na primer, modul Express obezbeđuje jednostavno podešavanje odredišnih ruta (URL-ova) sa kojima će se korisnici povezati. Takođe omogućava odlične funkcije prilikom korišćenja HTTP zahteva i objekata odgovora, uključujući „kolačiće“ i HTTP zaglavlje.

Sledi deo liste dragocenih funkcija Expressa:

- **upravljanje rutiranjem -** Express omogućava lako definisanje rute (krajnje tačke URL-a) koja se povezuje direktno sa funkcijama Node.js skripta na serveru.
- **obrada grešaka** - Express ima ugrađenu obradu greške koja se javlja kada dokument nije pronađen i drugih grešaka.
- **jednostavna integracija** - Express server se može lako implementirati u pozadini postojećeg reverznog proksi sistema, kao što su Nginx ili Varnish. To omogućava Express serveru jednostavnu integraciju u postojeći bezbednosni sistem.
- **„kolačići“** - Express omogućava jednostavno upravljanje „kolačićima“.
- **upravljač sesijama i keširanjem** - Express omogućava upravljanje sesijama i keširanjem.

Angular

Angular je JavaScript okruženje na strani klijenta, koje je razvila kompanija „Google“. On obezbeđuje sve funkcije koje su potrebne za upravljanje korisničkim unosom u pregledaču, za manipulisanje podacima na strani klijenta i za kontrolisanje kako su elementi prikazani u pregledaču. Angular, koji se piše pomoću JavaScripta, omogućava okruženje koje olakšava implementaciju veb aplikacija, koristeći MVC/MVVM radni okvir.

Na Node.js platformi mogu da se koriste i druga JavaScript okruženja, kao što su Backbone, Ember i Meteor. Međutim, Angular trenutno ima najbolji dizajn, skup funkcija i trajektorijum. Ovo su neke od njegovih prednosti:

- **povezivanje podataka** - Angular na jednostavan način povezuje podatke sa HTML elementima, koristeći moćni mehanizam oblasti važenja.
  proširivost - Angular arhitektura dozvoljava lako proširenje skoro svakog aspekta jezika da bi bile omogućene prilagođene implementacije.
- **čist kod** – U Angularu ste prinuđeni da napišete čist, logički kod.
- **kod za ponovnu upotrebu** - Kombinacija proširivosti i čistog koda olakšava pisanje koda za ponovnu upotrebu. U stvari, Angular vas primorava da napišete kod za ponovnu upotrebu prilikom kreiranja prilagođene usluge.
- **podrška** - Kompanija „Google“ mnogo ulaže u Angular projekat i zato je on u prednosti nad sličnim inicijativama koje nisu uspele.
- **kompatibilnost** - Angular je zasnovan na JavaScriptu i usko je povezan sa JavaScript standardom. To olakšava početak integracije Angulara u razvojno okruženje i ponovno korišćenje delova postojećeg koda unutar strukture radnog okvira Angular.

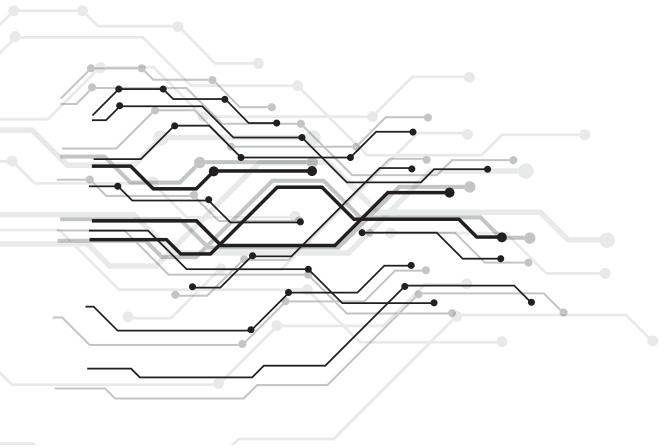
Rezime

U ovom poglavlju su prikazane osnove okruženja za razvoj veba. Razmatrane su interakcija između veb servera i pregledača i funkcije koje su potrebne za izradu modernih veb sajtova.

Takođe je opisan stek od Node.js do Angulara, koji sadrži alatke Node.js, MongoDB, Express i Angular. Node.js obezbeđuje platformu za radni okvir, MongoDB pozadinsko skladište podataka, Express veb server, a Angular radni okvir na strani klijenta za moderne veb aplikacije.

Sledeće

Sledeće poglavlje sadrži osnove JavaScript jezika. Pošto se ceo stek od Node.js do Angulara zasniva na JavaScriptu, potrebno je da poznajete JavaScript jezik da biste mogli da pratite primere u ostatku knjige.



2

Osnove JavaScripta

Svaka komponenta koju ćete koristiti iz ove knjige, Node.js, Express, TypeScript i Angular, zasniva se na jeziku JavaScript. To olakšava implementaciju i ponovno korišćenje koda na svim nivoima steka za razvoj veba.

Svrha ovog poglavlja je upoznavanje nekih osnova jezika JavaScript, kao što su promenljive, funkcije i objekti. Nije predviđeno da bude kompletan vodič za učenje jezika, već su u njemu ukratko predstavljene važne sintakse i idiomi. Ako ne poznajete JavaScript, ovo poglavlje bi trebalo da vam omogući dovoljno informacija da razumete primere u ostatku knjige. Ako već dobro poznajete JavaScript, možete da preskočite ovo poglavlje ili da ga pregledate kao podsetnik.

Definisanje promenljivih

U JavaScriptu ćemo prvo definisati promenljive. One predstavljaju način za dodelu naziva podacima, koje možete da koristite da biste privremeno uskladištili podatke i pristupili podacima iz vaših JavaScripta datoteka. Promenljive mogu ukazivati na jednostavne tipove podataka, kao što su brojevi ili stringovi, ili mogu ukazivati na složenije tipova podataka, kao što su objekti.

Da biste definisali promenljivu u JavaScriptu, upotrebite rezervisanu reč `var`, a zatim dodelite naziv promenljivoj - na primer:

```
var myData;
```

Možete, takođe, dodeliti vrednost promenljivoj u istoj liniji koda. Na primer, sledeća linija koda kreira promenljivu `myString` i dodeljuje joj vrednost „`Some Text`“:

```
var myString = "Some Text";
```

Mogu se koristiti i sledeće linije koda:

```
var myString;
myString = "Some Text";
```

Nakon što deklarišete promenljivu, možete koristiti naziv da biste dodelili vrednost promenljivoj i pristupili toj vrednosti. Na primer, sledeći kod sadrži string u promenljivoj `myString` i koristi ga za dodelu vrednosti promenljivoj `newString`:

```
var myString = "Some Text";
var newString = myString + " Some More Text";
```

Nazivi opisuju podatke koji su uskladišteni u promenljivim da biste lako mogli da ih upotrebite kasnije u programu. Jedina pravila za kreiranje naziva promenljivih su da nazivi moraju započeti slovom ili znakovima `$` ili _ i da ne mogu sadržati razmake. Osim toga, zapamtite da nazivi promenljivih razlikuju mala i velika slova, pa se naziv `myString` razlikuje od naziva `MyString`.

Razumevanje JavaScript tipova podataka

JavaScript koristi tipove podataka da bi odredio kako da upravlja podacima koji su dodeljeni promenljivoj. Tip promenljive određuje koje operacije možete izvršiti na promenljivoj, kao što su petlja ili izvršavanje koda. U sledećoj listi opisani su tipovi promenljivih koji se najčešće koriste u ovoj knjizi.

- **string** - Skladišti podatke znakova kao string. Podaci o znakovima se navode u jednostrukim ili dvostrukim navodnicima. Svi podaci koji su sadržani u navodnicima biće dodeljeni stringu promenljive - na primer:

  ```
  var myString = 'Some Text';
  var anotherString = 'Some More Text';
  ```

- **broj** - Skladišti podatke kao numeričku vrednost. Brojevi se koriste za brojanje, izračunavanja i upoređivanja. Ovo su neki primeri:

  ```
  var myInteger = 1;
  var cost = 1.33;
  ```

- **Bulova vrednost** - Skladišti jedan bit koji je tačan ili netačan. Bulovi se često koriste kao indikatori. Na primer, možete da postavite promenljivu na vrednost false na početku nekog koda i da je proverite kada se izvrši (pogledajte da li je kod izvršen na određenom mestu). Sledeći primeri definišu promenljive `true` i `false`.

  ```
  var yes = true;
  var no = false;
  ```

- **niz** - Indeksirani niz predstavlja niz odvojenih različitih stavki podataka koji su smešteni pod jednim nazivom promenljive. Stavkama u nizu se može pristupiti korišćenjem nultog indeksa pomoću iskaza `array[index]`. U sledećem primeru kreiran je jednostavan niz, a zatim se pristupa prvom elementu, koji je na indeksu `0`.

  ```
  var arr = ["one", "two", "three"]
  var first = arr[0];
  ```

- **literal objekta** - JavaScript podržava mogućnost kreiranja i korišćenja literala objekta. Kada koristite literal objekta, možete pristupiti vrednostima i funkcijama u objektu pomoću sintakse `object.property`. U sledećem primeru prikazano je kako da kreirate svojstva literala objekata i kako da im pristupite:

```
var obj = {"name": "Brendan", "Hobbies":["Video Games", "camping"], "age",
"Unknown"};
var name = obj.name;
```

- **Null** - Ponekad nemate vrednost za skladištenje u promenljivoj zbog toga što nije kreirana ili je više ne koristite. U ovom trenutku možete podesiti promenljivu na vrednost `null`. Korišćenje te vrednosti je bolje od dodeljivanja vrednosti `0` promenljivoj ili praznog niza „", zato što ove vrednosti mogu biti važeće za promenljivu. Dodelom vrednosti `null` promenljivoj omogućavamo da ta promenljiva, u stvari, bude bez vrednosti i možemo da proverimo `null` unutar koda.

```
var newVar = null;
```

> **NAPOMENA**
>
> JavaScript je jezik bez tipa. U skriptu ne morate navesti koji je tip podataka promenljiva, jer interpretator automatski identifikuje tačan tip. Osim toga, možete da promenljivoj jednog tipa dodelite vrednost drugog tipa. Na primer, u sledećem kodu se definiše promenljiva stringa, koja se, zatim, dodeljuje tipu celog broja:
> var id = „testID";
> id = 1;

Upotreba operatora

JavaScript operatori omogućavaju da promenite vrednost promenljive. Već poznajete operator =, koji se koristi za dodeljivanje vrednosti promenljivim. JavaScript omogućava nekoliko različitih operatora, koji se mogu grupisati u dve vrste: aritmetički operatori i operatori dodele.

Aritmetički operatori

Aritmetički operatori se koriste za obavljanje operacija između promenljivih i direktnih vrednosti. Tabela 2.1 sadrži listu aritmetičkih operacija, zajedno sa rezultatima koji se primenjuju.

Tabela 2.1 JavaScript aritmetički operatori, sa rezultatima koji se zasnivaju na izrazu y=4

OPERATOR	OPIS	PRIMER	DOBIJENI X
+	Sabiranje	x=y+5 x=y+"5" x="Four"+y+"4"	9 „45" „Four44"

OPERATOR	OPIS	PRIMER	DOBIJENI X
-	Oduzimanje	x=y-2	2
++	Inkrementiranje	x=y++ x=++y	4 5
--	Dekrementiranje	x=y-- x=--y	4 3
*	Množenje	x=y*4	16
/	Deljenje	x=10/y	2.5
%	Modulo (ostatak deljenja)	x=y%3	1

NAPOMENA

Operator + može da se koristi i za dodavanje stringova ili za kombinacije stringova i brojeva. On omogućava da brzo spojite stringove i da dodate numeričke podatke izlaznim nizovima. U tabeli 2.1 prikazano je da se prilikom dodavanja numeričke vrednosti i vrednosti stringa numerička vrednost pretvara u niz, a zatim se dva stringa spajaju.

Operatori dodele

Operatori dodele se koriste za dodelu vrednosti promenljivoj. Pored operatora =, postoji nekoliko različitih obrazaca koji omogućavaju manipulaciju podacima dok dodeljujete vrednost. U tabeli 2.2 prikazana je lista operacija dodele, zajedno sa rezultatima koji se primenjuju.

Tabela 2.2 JavaScript operatori dodele i rezultati koji se zasnivaju na izrazu x=10

OPERATOR	PRIMER	EKVIVALENTNE ARITMETIČKE OPERACIJE	DOBIJENI X
=	x=5	x=5	5
+=	x+=5	x=x+5	15
-=	x-=5	x=x-5	5
*=	x*=5	x=x*5	50
/=	x/=5	x=x/5	2
%=	x%=5	x=x%5	0

Primena operatora poređenja i uslovnih operatora

Pomoću uslovnih operatora možete da primenite logiku na vaše aplikacije tako da određeni kod bude izvršen samo pod određenim uslovima. To ćete uraditi primenom logike poređenja na vredno-

sti promenljive. U sledećim odeljcima su opisani poređenja koja su dostupna u JavaScriptu i njihova primena u uslovnim iskazima.

Operatori poređenja

Operator poređenja procenjuje dva dela podataka i vraća vrednost `true` ako je procena tačna, ili vrednost `false` ako je procena netačna. Operatori poređenja upoređuju vrednost na levoj strani operatora sa vrednošću na desnoj strani.

Najjednostavniji način na koji možemo da vam pomognemo da razumete sintaksu JavaScript poređenja je da prikažemo listu sa nekoliko primera. Tabela 2.3 sadrži listu operatora poređenja sa nekoliko primera.

Tabela 2.3 JavaScript operatori poređenja i rezultati koji se zasnivaju na izrazu x=10

OPERATOR	OPIS	PRIMER	REZULTAT
==	Jednako (koristi se samo za vrednost)	x==8 x==10	false true
===	Vrednost i tip su jednaki	x===10 x===“10“	true false
!=	Nije jednako	x!=5	true
!==	Vrednost i tip nisu jednaki	x!==“10“ x!==10	true false
>	Veće od	x>5	true
>=	Veće ili jednako	x>=10	true
<	Manje od	x<5	false
<=	Manje ili jednako	x<=10	true

Možete kombinovati više poređenja pomoću logičkih operatora i standardnih zagrada. U tabeli 2.4 prikazani su lista logičkih operatora i način kako se oni koriste za kombinovanje sa operatorima poređenja.

Tabela 2.4 JavaScript logički operatori i rezultati koji se zasnivaju na izrazima y=10 i y=5

OPERATOR	OPIS	PRIMER	REZULTAT
&&	And	(x==10 && y==5) (x==10 && y>x)	true false

OPERATOR	OPIS	PRIMER	REZULTAT
||	Or	(x>=10 || y>x) (x<10 || y>x)	true false
!	Not	!(x==y) !(x>y)	true true
	Mix	(x>=10 && y<x || x==y) ((x<y || x>=10) && y>=5) (!(x==y) && y>=10)	true true false

Upotreba iskaza if

Iskaz `if` omogućava da odvojite izvršenje koda na osnovu procene koja je izvršena poređenjem. U sledećim linijama koda uslovni operatori su u zagradama `()`, a kod koji će biti izvršen ako uslovni operator ima vrednost `true` je u zagradama `{}`:

```
if(x==5){
  do_something();
}
```

Pored toga što možete da izvršite kod unutar bloka iskaza `if`, možete odrediti blok `else` koji se izvršava samo ako je uslov `false` - na primer:

```
if(x==5){
  do_something();
} else {
  do_something_else();
}
```

Takođe možete izvršiti ulančavanje iskaza `if`. Da biste to izvršili, dodajte uslovni iskaz zajedno sa iskazom `else` - na primer:

```
if(x<5){
  do_something();
} else if(x<10) {
  do_something_else();
} else {
  do_nothing();
}
```

Implementiranje iskaza switch

Još jedan tip uslovne logike je iskaz `switch`. On omogućava da jednom procenite izraz, a zatim da na osnovu vrednosti izvršite jedan od mnogih delova koda.

Sintaksa za iskaz `switch` je sledeća:

```
switch(expression){
  case value1:
```

```
    <code to execute>
    break
  case value2:
    <code to execute>
    break;
  default:
    <code to execute if not value1 or value2>
}
```

Iskaz `switch` u potpunosti procenjuje izraz i daje vrednost. Vrednost može biti niz, broj, Bulova vrednost ili, čak, i objekat. Iskaz `switch` se, zatim, upoređuje sa svakom vrednošću koja je navedena u iskazu `case`. Ako se vrednosti podudaraju, izvršava se kod u iskazu `case`. Ako se vrednosti ne podudaraju, izvršava se podrazumevani kod.

> **NAPOMENA**
>
> Svaki iskaz break sadrži, obično, komandu na kraju, koja ukazuje na prekid iskaza switch. Ako iskaz break ne postoji, izvršenje koda se nastavlja pomoću sledećeg iskaza case.

Implementiranje petlje

Petlja služi za izvršavanje istog segmenta koda više puta. Ovo je korisno kada morate da izvršavate iste zadatke u nizu ili skupu objekata.

JavaScript može da izvršava petlje `for` i `while`. U sledećim odeljcima je opisano kako se implementiraju petlje u JavaScriptu.

Petlje while

Najosnovniji tip petlje u JavaScriptu je petlja `while`. Ona testira izraz i nastavlja da izvršava kod koji se nalazi u zagradama {}, sve dok izraz ne bude imao vrednost `false`.

Na primer, sledeća petlja `while` se izvršava sve dok vrednost promenljive `i` nije jednaka broju `5`:

```
var i = 1;
while (i<5){
  console.log("Iteration " + i + "<br>");
  i++;
}
```

Rezultat koji je dobijen u konzoli je sledeći:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

Petlje do/while

Drugi tip petlje `while` je petlja `do/while`. Ona je korisna ako uvek želite da izvršite kod u petlji najmanje jednom, a izraz ne može da bude testiran dok se kod ne izvrši najmanje jednom.

Na primer, sledeća petlja `do/while` se izvršava sve dok vrednost `day` ne bude jednaka vrednosti `Wednesday`:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i=0;
do{
  var day=days[i++];
  console.log("It's " + day + "<br>");
} while (day != "Wednesday");
```

Rezultat koji je dobijen u konzoli je sledeći:

```
It's Monday
It's Tuesday
It's Wednesday
```

Petlje for

JavaScript petlja `for` omogućava da izvršite kod određeni broj puta tako što ćete koristiti iskaz `for`, koji kombinuje tri iskaza u jedan blok izvršavanja pomoću sledeće sintakse:

```
for (assignment; condition; update;){
  code to be executed;
}
```

Iskaz `for` koristi ta tri iskaza na sledeći način prilikom izvršenja petlje:

- **assignment** - Ovaj iskaz je izvršen pre nego što petlja započne i više se ne izvršava. Koristan je za inicijalizaciju promenljivih koje će se koristiti u petlji kao uslovi.
- **condition** - Izraz je procenjen pre svake iteracije petlje. Ako izraz ima vrednost `true`, petlja `for` se izvršava - u suprotnom, petlja se prekida.
- **update**: Izvršena je svaka iteracija nakon što je izvršen kod u petlji. Iskaz update se obično koristi za povećanje brojača, koji je upotrebljen u iskazu condition.

Sledeći primer ilustruje ne samo osnovnu petlju, već i mogućnost ugnežđavanja jedne petlje unutar druge:

```
for (var x=1; x<=3; x++){
  for (var y=1; y<=3; y++){
    console.log(x + " X " + y + " = " + (x*y) + "<br>");
  }
}
```

Rezultat koji je dobijen u konzoli je sledeći:

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
```

Petlje for/in

Još jedan tip petlje je `for/in`. Ona se izvršava na bilo koji tip podataka na koji se može ponoviti. U većini slučajeva petlju `for/in` ćete koristiti na nizovima i objektima. U sledećem primeru ilustrovani su sintaksa i „ponašanje“ `for/in` petlje na jednostavnom nizu:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  console.log("It's " + days[idx] + "<br>");
}
```

Obratite pažnju da je promenljiva `idx` prilagođena svakom ponavljanju petlje, od početka indeksa niza, do poslednjeg indeksa niza. Dobijeni rezultat je sledeći:

```
It's Monday
It's Tuesday
It's Wednesday
It's Thursday
It's Friday
```

Prekid petlji

Kada koristite petlju, ponekad će biti potrebno da prekinete izvršavanje koda unutar samog koda, bez čekanja sledeće iteracije. Postoje dva različita načina da to uradite: pomoću rezervisane reči `break` i pomoću rezervisane reči `continue`.

Rezervisana reč `break` zaustavlja izvršenje petlje `for` ili `while` u potpunosti. Sa druge strane, rezervisana reč `continue` zaustavlja izvršenje koda unutar petlje, a nastavlja se u sledećoj iteraciji. Razmotrite sledeće primere:

Upotreba reči `break` ako je dan u nedelji Wednesday (sreda):

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    break;
  console.log("It's " + days[idx] + "<br>");
}
```

Kada je vrednost Wednesday, izvršavanje petlje se u potpunosti zaustavlja:

```
It's Monday
It's Tuesday
```

Upotreba reči `continue` ako je dan u nedelji Wednesday:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    continue;
  console.log("It's " + days[idx] + "<br>");
}
```

Obratite pažnju da se Wednesday ne ispisuje, jer je upotrebljen iskaz `continue`. Međutim, izvršenje petlje je u potpunosti dovršeno:

```
It's Monday
It's Tuesday
It's Thursday
It's Friday
```

Kreiranje funkcija

Jedan od najvažnijih delova JavaScripta je pisanje koda koji može ponovo koristiti drugi kod. Da biste napisali taj kod, podelite svoj kod na funkcije koje obavljaju određene zadatke. Funkcija je niz iskaza u kodu koji su kombinovani u jednom bloku i kojima je dodeljen naziv. Kod u bloku se može izvršiti tako što se upućuje na taj naziv.

Definisanje funkcija

Funkcije se definišu pomoću rezervisane reči `function` - ona sadrži naziv koji opisuje upotrebu ove funkcije, liste bez ijednog argumenta ili sa više argumenata u zagradama () i bloka sa jednim ili više iskaza koda u zagradama {}. Na primer, ovo je definicija funkcije koja ispisuje tekst „`Hello World`" u konzolu.

```
function myFunction(){
  console.log("Hello World");
}
```

Da biste izvršili kod u iskazu `myFunction()`, potrebno je da dodate sledeću liniju koda u glavni JavaScript kod ili unutar druge funkcije:

```
myFunction();
```

Prosleđivanje promenljivih funkciji

Često ćete morati da prosledite određene vrednosti funkcijama koje će biti upotrebljene prilikom izvršavanja koda. Vrednosti se prosleđuju funkciji u obliku liste koja je razdvojena zarezima. Za definiciju funkcije potrebna je lista naziva promenljivih u zagradama (), koje odgovaraju broju naziva koji se prosleđuju. Na primer, sledeća funkcija prihvata dva argumenta `name` i `city`, koje koristi za izradu izlaznog stringa:

```
function greeting(name, city){
  console.log("Hello " + name);
  console.log(". How is the weather in " + city);
}
```

Da biste pozvali funkciju `greeting()`, morate je proslediti vrednostima `name` i `city`. Vrednost može biti direktna vrednost ili prethodno definisana promenljiva. Sledeći kod izvršava funkciju `greeting()`, koja sadrži promenljivu `name` i direktan string iz vrednosti `city`:

```
var name = "Brad";
greeting(name, "Florence");
```

Vraćanje vrednosti iz funkcija

Često se događa da funkcije moraju vratiti vrednost kodu koji se poziva. Dodavanje rezervisane reči `return`, koja je praćena promenljivom ili vrednošću, vraća vrednost iz funkcije. Na primer, sledeći kod poziva funkciju za formatiranje stringa, dodeljuje vrednost koja se vraća iz funkcije u promenljivu, a zatim ispisuje vrednost u konzolu:

```
function formatGreeting(name, city){
  var retStr = "";
  retStr += "Hello <b>" + name +"<b>,<br>);
  retStr += "Welcome to " + city + "!";
 return retStr;
}
var greeting = formatGreeting("Brad", "Rome");
console.log(greeting);
```

Možete da dodate više iskaza `return` funkciji. Kada funkcija naiđe na iskaz `return`, izvršavanje koda se odmah zaustavlja. Ako iskaz `return` sadrži vrednost koju treba da vrati, vrednost će biti vraćena. U sledećem primeru je prikazana funkcija koja proverava unos i koja se odmah vraća ako je vrednost nula.

```
function myFunc(value){
  if (value == 0)
    return value;
  <code_to_execute_if_value_nonzero>
  return value;
}
```

Upotreba anonimnih funkcija

Do sada su svi primeri koje ste videli primeri imenovanih funkcija. JavaScript omogućava kreiranje anonimnih funkcija. U funkcionalnom jeziku, kao što je JavaScript, anonimne funkcije se mogu koristiti kao parametri funkcija, kao svojstva objekta ili za vraćanje vrednosti iz funkcije. Prednost anonimnih funkcija je što se definišu direktno u skupovima parametara kada se pozivaju druge funkcije. Dakle, nije vam potrebna formalna definicija.

Na primer, u sledećem kodu se definiše funkcija `doCalc()`, koja prihvata tri parametra. Prva dva parametra treba da budu brojevi, a treći je funkcija koja se poziva i prosleđuje dva broja kao argumente:

```
function doCalc(num1, num2, calcFunction){
    return calcFunction(num1, num2);
}
```

Možete definisati funkciju i potom proslediti njen naziv bez prosleđivanja parametara funkciji `doCalc()` - na primer:

```
function addFunc(n1, n2){
    return n1 + n2;
}
doCalc(5, 10, addFunc);
```

Međutim, takođe možete da koristite anonimnu funkciju direktno u pozivu funkciji `doCalc()`, kao što je prikazano u sledećim iskazima:

```
console.log( doCalc(5, 10, function(n1, n2){ return n1 + n2; }) );
console.log( doCalc(5, 10, function(n1, n2){ return n1 * n2; }) );
```

Kao što vidite, prednost korišćenja anonimnih funkcija je što vam nije potrebna formalna definicija, jer se one neće koristiti na drugom mestu u kodu. Kada se u JavaScriptu upotrebe anonimne funkcije, on postaje koncizniji i čitljiviji.

Razumevanje oblasti važenja

Da biste počeli da dodajete uslove, funkcije i petlje u JavaScript aplikacije, treba da razumete oblast važenja promenljive. Oblast važenja promenljive je, jednostavno, „vrednost određene promenljive u liniji koda koji se izvršava“.

JavaScript omogućava da definišete i globalnu i lokalnu verziju promenljive. Globalna verzija je definisana u glavnom JavaScript kodu, a lokalne verzije su definisane unutar funkcija. Kada definišete lokalnu verziju u funkciji, biće kreirana nova promenljiva u memoriji. U okviru te funkcije ukazaćete na lokalnu verziju. Izvan te funkcije ukazaćete na globalnu verziju.

Da biste bolje razumeli oblast važenja promenljive, razmotrite sledeći kod:

```
var myVar = 1;
function writeIt(){
  var myVar = 2;
  console.log("Variable = " + myVar);
  writeMore();
}
function writeMore(){
  console.log("Variable = " + myVar);
}
writeIt();
```

Globalna promenljiva `myVar` se definiše u liniji 1, a u liniji 3 se definiše lokalna verzija u funkciji `writeIt()`. Linija 4 ispisuje „`Variable = 2`" u konzolu. Zatim se u liniji 5 poziva funkcija `writeMore()`. Pošto u funkciji `writeMore()` nije definisana lokalna verzija promenljive `myVar`, vrednost globalne promenljive `myVar` se ispisuje u liniji 8.

Upotreba JavaScript objekata

JavaScript ima nekoliko ugrađenih objekata, kao što su Number, Array, String, Date i Math. Svaki od tih objekata ima svojstva i metode članova. Pored JavaScript objekata, Node.js, MongoDB, Express i Angular dodaju svoje ugrađene objekte.

JavaScript omogućava dobru strukturu objektno-orijentisanih programa za kreiranje prilagođenih objekata. Korišćenje objekata, umesto kolekcija funkcija, je važno za pisanje čistog i efikasnog JavaScript koda koji se može ponovo upotrebiti.

Upotreba sintakse objekta

Da biste efikasno koristili objekte u JavaScriptu, morate da razumete njihove strukture i sintaksu. Objekat je, zapravo, samo kontejner za grupisanje više vrednosti i, u nekim slučajevima, funkcioniše zajedno sa vrednostima. Vrednosti objekta se zovu *svojstva*, a funkcije se zovu *metodi*.

Da biste koristili objekat JavaScripta, morate prvo da kreirate instancu objekta. Instance objekata se kreiraju pomoću rezervisane reči `new`, koja sadrži naziv konstruktora. Na primer, koristite sledeću liniju koda da biste kreirali instancu ugrađenog objekta `Number` u JavaScriptu:

```
var x = new Number ("5");
```

Sintaksa objekta je veoma jednostavna: koristite naziv objekta, iza kojeg sledi tačka, pa svojstvo ili naziv metoda. Na primer, sledeće linije koda dobijaju i postavljaju svojstvo `name` objekta pod nazivom `myObj`:

```
var s = myObj.name;
myObj.name = "New Name";
```

Isto tako možete da dobijete i postavite metode objekta. Na primer, sledeće linije koda pozivaju metod `getName()` i menjaju funkciju metoda na objektu pod nazivom `myObj`:

```
var name = myObj.getName();
myObj.getName = function() { return this.name; };
```

Možete da kreirate i objekte kojima ćete dodeliti promenljive i funkcije direktno pomoću sintakse `{}`. Na primer, sledeći kod definiše novi objekat i dodeljuje vrednosti i funkciju metoda:

```
var obj = {
    name: "My Object",
    value: 7,
    getValue: function() { return this.value; }
};
```

Takođe možete pristupiti članovima objekta JavaScripta pomoću sintakse `[propertyName]` objekta. Ovo je korisno kada koristite dinamičke nazive svojstava ili ako naziv svojstva sadrži znakove koje JavaScript ne podržava. Sledeći primer koda pristupa svojstvima „`User Name`" i „`Other Name`" objekta pod nazivom `myObj`:

```
var propName = "User Name";
var val1 = myObj[propName];
var val2 = myObj["Other Name"];
```

Kreiranje prilagođenih objekata

Kao što ste videli do sada, upotreba ugrađenih JavaScript objekata ima nekoliko prednosti. Kada budete počeli da pišete kod koji koristi sve više i više podataka, želećete da kreirate sopstvene prilagođene objekte, sa specifičnim svojstvima i metodama.

JavaScript objekti mogu da se definišu na nekoliko različitih načina. Najjednostavniji način je „u pokretu": kreirajte generički objekat, a zatim dodajte svojstva po potrebi.

Na primer, da biste kreirali korisnički objekat i dodelili ime i prezime i definisali funkciju da biste vratili ime, možete da koristite sledeći kod:

```
var user = new Object();
user.first="Brendan";
user.last="Dayley";
user.getName = function() { return this.first + " " + this.last; }
```

Isto to možete postići i direktnom dodelom, koristeći sledeću sintaksu u kojoj je objekat zatvoren u zagradama `{}`, a svojstva se definišu pomoću sintakse `property:value`:

```
var user = {
  first: Brendan,
  last: 'Dayley',
  getName: function() { return this.first + " " + this.last; }};
```

Prve dve opcije dobro funkcionišu za jednostavne objekte koje ne treba da ponovo kasnije koristite. Bolji metod za ponovno korišćenje objekata je da, zapravo, ugradite objekat unutar funkcijskog bloka. To će vam omogućiti da zadržite ceo kod koji se odnosi na objekat koji je lokalni - na primer:

```
function User(first, last){
  this.first = first;
  this.last = last;
  this.getName = function( ) { return this.first + " " + this.last; }};
var user = new User("Brendan", "Dayley");
```

Krajnji rezultat ovih metoda je, u suštini, isti: imate objekat sa svojstvima koja se mogu referencirati pomoću sintakse sa tačkom, kao što je prikazano ovde:

```
console.log(user.getName());
```

Upotreba prototipskog obrasca objekta

Napredniji način za kreiranje objekata je upotreba prototipskog obrasca. Ovaj obrazac se implementira tako što se funkcije definišu unutar atributa `prototype` objekta, umesto u samom objektu. Prednost izrade prototipa je u tome što se funkcije koje su definisane u prototipu kreiraju samo jednom kada se učita JavaScript, umesto svakog puta kada se kreira novi objekat.

U sledećem primeru prikazan je kod koji je potreban za implementiranje prototipskog obrasca. Primetićete da se objekat `UserP` definiše i zatim se postavlja `UserP.prototype` da biste mogli da dodate funkciju `getFullName()`. Možete da dodate koliko god funkcija želite u prototip. Kada se kreira novi objekat, te funkcije će biti dostupne.

```
function UserP(first, last){
  this.first = first;
  this.last = last;
}
UserP.prototype = {
  getFullName: function(){
      return this.first + " " + this.last;
    }
};
```

Manipulisanje stringovima

`String` je objekat koji se najčešće koristi u JavaScriptu. JavaScript automatski kreira taj objekat uvek kada definišete promenljivu koja sadrži tip podataka stringa - na primer:

```
var myStr = "Teach Yourself jQuery & JavaScript in 24 Hours";
```

Prilikom kreiranja stringa nekoliko specijalnih znakova nije moguće dodati direktno u string. Za ove znakove JavaScript obezbeđuje skup izlaznih sekvenci, koje su opisane u tabeli 2.5.

Tabela 2.5 Izlazne sekvence objekta String

IZLAZNA SEKVENCA	OPIS	PRIMER	IZLAZNI STRING
\‘	jednostruki znak navoda	„couldn\‘t be“	couldn‘t be
\“	dvostruki znak navoda	„I \“think\“ I \“am\““	I „think“ I „am“
\\	obrnuta kosa crta	„one\\two\\three“	one\two\three
\n	novi red	„I am\nI said“	I am I said
\r	znak za početak reda	„to be\ror not“	to be or not

IZLAZNA SEKVENCA	OPIS	PRIMER	IZLAZNI STRING
\t	tabulator	„one\ttwo\tthree“	one two three
\b	brisanje unazad	„correctoin\b\b\bion“	correction
\f	prelazak na sledeću stranicu	„Title A\fTitle B“	Title A then Title B

Da biste utvrdili dužinu stringa, možete da upotrebite svojstvo `length` objekta `String` - na primer:

```
var numOfChars = myStr.length;
```

Objekat `String` ima nekoliko funkcija koje omogućavaju da pristupite i stringu i manipulišete njime na različite načine. Metodi za manipulisanje stringom su opisani u tabeli 2.6.

Tabela 2.6 Metodi za manipulisanje objektima String

METOD	OPIS
charAt(index)	Vraća znak na određenom indeksu.
charCodeAt(index)	Vraća vrednost unicode znaka na određenom indeksu.
concat(str1, str2, ...)	Spaja jedan ili više stringova i vraća kopiju spojenih stringova.
fromCharCode()	Konvertuje vrednosti unicode u stvarne znakove.
indexOf(subString)	Vraća poziciju prvog ponavljanja zadate vrednosti SubString. Vraća -1 ako podstring nije pronađen.
lastIndexOf(subString)	Vraća poziciju poslednjeg ponavljanja zadate vrednosti SubString. Vraća -1 ako podstring nije pronađen.
match(regex)	Pretražuje string i vraća sva podudaranja regularnog izraza.
replace(subString/regex, replacementString)	Traži podudaranja vrednosti SubString ili regularnog izraza u stringu i zamenjuje postojeći podstring novim.
search(regex)	Pretražuje string na osnovu regularnog izraza i vraća poziciju prvog podudaranja.
slice(start, end)	Vraća novi string, čiji je deo između pozicija start i end uklonjen.
split(step, limit)	Razdvaja string na niz podnizova na osnovu graničnog znaka ili regularnog izraza. Opcioni argument limit definiše maksimalni broj razdvajanja da bi razdvajanje bilo započeto od početka.
substr(start,length)	Ekstrahuje znakove iz stringa, počev od navedene pozicije start do pozicije znakova lenght.
substring(from, to)	Vraća podstring znakova između indeksa from i to.

METOD	OPIS
toLowerCase()	Konvertuje velika slova stringa u mala.
toUpperCase()	Konvertuje mala slova stringa u velika.
valueOf()	Vraća vrednosti osnovnog stringa.

Da biste počeli da koristite funkciju koja je data u objektu `String`, u sledećem odeljku opisaćemo neke od uobičajenih zadataka koji se mogu izvršiti pomoću metoda objekta `String`.

Kombinovanje stringova

Više stringova se može kombinovati pomoću operacije + ili primenom funkcije `concat()` na prvi string. Na primer, u sledećem kodu stringovi `sentence1` i `sentence2` će dati isti rezultat:

```
var word1 = "Today ";
var word2 = "is ";
var word3 = "tomorrow\'s";
var word4 = "yesterday.";
var sentence1 = word1 + word2 + word3 + word4;
var sentence2 = word1.concat(word2, word3, word4);
```

Traženje podstringa u stringu

Da biste proverili da li je string podstring drugog stringa, možete da koristite metod `IndexOf()`. Na primer, sledeći kod ispisuje string u konzolu samo ako sadrži reč „think“:

```
var myStr = "I think, therefore I am.";
if (myStr.indexOf("think") != -1){
  console.log (myStr);
}
```

Zamena reči u stringu

Još jedan uobičajeni zadatak objekta `String` je zamena jednog podstringa drugim. Da biste zamenili reč ili frazu u stringu, upotrebite metod `replace()`. Sledeći kod zamenjuje tekst „`<username>`“ vrednošću promenljive `username`:

```
var username = "Brendan";
var output = "<username> please enter your password: ";
output.replace("<username>", username);
```

Podela stringa na niz

Stringove možete, obično, podeliti na nizove, koristeći granični znak. Na primer, sledeći kod deli vremenski string na niz osnovnih delova pomoću metoda `split()` u „`:`" razdelniku:

```
var t = "12:10:36";
var tArr = t.split(":");
var hour = t[0];
var minute = t[1];
var second = t[2];
```

Upotreba nizova

Objekat Array omogućava skladištenje skupa drugih objekata i upravljanje njime. U nizovima mogu da se uskladište brojevi, stringovi i drugi JavaScript objekti. Postoji nekoliko različitih metoda za kreiranje JavaScript nizova. Na primer, sledeći iskaz kreira tri identične verzije jednog niza:

```
var arr = ["one", "two", "three"];
var arr2 = new Array();
arr2[0] = "one";
arr2[1] = "two";
arr2[2] = "three";
var arr3 = new Array();
arr3.push("one");
arr3.push("two");
arr3.push("three");
```

Prvi metod definiše objekat `arr` i postavlja sadržaj u jedan iskaz pomoću zagrada `[]`. Drugi metod kreira objekat `arr2` i dodaje stavke u njemu pomoću direktne dodele indeksa. Treći metod kreira objekat `arr3`, pa koristi najbolju opciju za proširenje nizova, a to je upotreba metoda `push()` za smeštanje stavki u niz.

Da biste dobili elemente u nizu, možete da koristite svojstvo `length` objekta `Array` - na primer:

```
var numOfItems = arr.length;
```

Nizovi su nulta indeksni, što znači da je prva stavka na indeksu 0. Na primer, u sledećem kodu vrednost promenljive `first` će biti `Monday`, a vrednost promenljive `last` će biti `Friday`:

```
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var first = w [0];
var last = week[week.length-1];
```

Objekat `Array` ima nekoliko ugrađenih funkcija koje omogućavaju da pristupite nizu i da manipulišete njime na različite načine. U tabeli 2.7 opisani su metodi koji su povezani sa objektom `Array`, koji omogućava da manipulišete sadržajem niza.

Tabela 2.7 Metodi za manipulisanje objektima Array

METOD	OPIS
concat(arr1, arr2, ...)	Vraća pridruženu kopiju nizova koji su prosleđeni kao argumenti.
indexOf(value)	Vraća prvi indeks vrednosti value u nizu ili -1 ako stavka nije pronađena.
join(separator)	Spaja sve elemente niza koje je granični znak razdvojio u jedan string. Ako granični znak nije naveden, koristi se zarez.
lastIndexOf(value)	Vraća poslednji indeks vrednosti value u nizu ili -1 ako vrednost nije pronađena.
pop()	Uklanja poslednji element iz niza i vraća taj element.
push(item1, item2, ...)	Dodaje jedan ili više novih elemenata na kraju niza i vraća novu vrednost dužine.
reverse()	Inverzuje redosled svih elemenata u nizu.
shift()	Uklanja prvi element niza i vraća taj element.
slice(start, end)	Vraća elemente između indeksa start i end.
sort(sortFunction)	Sortira elemente niza. Funkcija sortFunction je opciona.
splice(index, count, item1, item2...)	Ako je index određen, stavke koje se nabrajaju se uklanjaju, a zatim se opcione stavke prosleđuju kao argumenti u index.
toString()	Vraća string niza.
unshift()	Dodaje nove elemente na početku niza i vraća novu vrednost dužine.
valueOf()	Vraća osnovnu vrednost objekta niza.

Da biste počeli da koristite funkciju koja je data u objektu `Array`, u sledećem odeljku opisaćemo neke od uobičajenih zadataka koji se mogu izvršiti pomoću metoda objekta `Array`.

Kombinovanje nizova

Možete da kombinujete nizove na isti način na koji kombinujete objekte `String` pomoću operacije + ili pomoću metoda `concat()`. U sledećem kodu niz `arr3` će biti isti kao i niz `arr4`:

```
var arr1 = [1,2,3];
var arr2 = ["three", "four", "five"]
var arr3 = arr1 + arr2;
var arr4 = arr1.concat(arr2);
```

> **NAPOMENA**
>
> Možete da kombinujete niz brojeva i niz stringova. Svaka stavka u nizu zadržava svoj tip objekta. Međutim, dok koristite stavke u nizu, potrebno je da obratite pažnju na nizove koji imaju više tipova podataka da ne bi došlo do problema.

Ponavljanje nizova

Možete da ponovite niz pomoću petlje `for` ili `for/in`. U sledećem kodu prikazano je ponavljanje svake stavke u nizu pomoću obe petlje:

```
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var i=0; i<week.length; i++){
  console.log("<li>" + week[i] + "</li>");
}
for (dayIndex in week){
  console.log("<li>" + week[dayIndex] + "</li>");
}
```

Pretvaranje niza u string

Korisna funkcija objekta `Array` je kombinovanje elemenata stringa, radi razdvajanja objekta `String` posebnim separatorom pomoću metoda `join()`. Na primer, sledeći kod pretvara vremenske komponente u format 12:10:36:

```
var timeArr = [12,10,36];
var timeStr = timeArray.join(":");
```

Provera stavke u nizu

Često ćete morati da proverite da li niz sadrži određenu stavku. To možete uraditi pomoću metode `indexOf()`. Ako stavka nije pronađena u listi, biće vraćen `-1`. Ako se stavka nalazi u nizu `week`, sledeća funkcija ispisuje poruku u konzolu:

```
function message(day){
  var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
  if (week.indexOf(day) == -1){
    console.log("Happy " + day);
  }
}
```

Dodavanje i uklanjanje stavki nizova

Postoji nekoliko metoda za dodavanje i uklanjanje stavki iz objekta `Array` pomoću različitih ugrađenih metoda. U tabeli 2.8 prikazani su različiti metodi koji se koriste u ovoj knjizi.

Tabela 2.8 Metodi niza Array koji se koriste za dodavanje i uklanjanje elemenata iz nizova

ISKAZ	VREDNOST PROMENLJIVE X	VREDNOSTI NIZA ARR
var arr = [1,2,3,4,5];	undefined	1,2,3,4,5
var x = 0;	0	1,2,3,4,5
x = arr.unshift(„zero“);	6 (length)	zero,1,2,3,4,5
x = arr.push(6,7,8);	9 (length)	zero,1,2,3,4,5,6,7,8
x = arr.shift();	zero	1,2,3,4,5,6,7,8
x = arr.pop();	8	1,2,3,4,5,6,7
x=arr.splice(3,3,“four“, „five“,“six“);	4,5,6	1,2,3,four,five,six,7
x = arr.splice(3,1);	four	1,2,3,five,six,7
x = arr.splice(3);	five,six,7	1,2,3

Dodavanje obrade grešaka

Važan deo pisanja koda u JavaScriptu je dodavanje obrade grešaka u slučajevima kada postoje problemi. Ako se izuzetak u kodu javlja zbog problema u vašem JavaScriptu, skript podrazumevano neće uspeti i učitavanje neće biti dovršeno. To nije željeno ponašanje, već je, u stvari, veoma štetno. Da biste sprečili ove vrste problema, stavite kod u blok `try/catch`.

Blokovi try/catch

Da biste sprečili da se vaš kod potpuno prekine, koristite `try/catch` blokove koji mogu da obrade greške unutar koda. Ako JavaScript naiđe na grešku prilikom izvršavanja koda u bloku `try`, prelazi na izvršavanje dela `catch`, umesto da zaustavi ceo skript. Ako nema greške, izvršavaju se svi blokovi `try`, ali ne i blokovi `catch`.

Na primer, sledeći blok `try/catch` pokušava da dodeli promenljivu `x` vrednosti nedefinisane promenljive pod nazivom `badVarNam`.

```
try{
    var x = badVarName;
} catch (err){
    console.log(err.name + ': "' + err.message +  '" occurred when assigning x.');
}
```

Imajte na umu da iskaz `catch` prihvata parametar `err`, koji je objekat greške. Objekat greške omogućava svojstvo `message` koje sadrži opis greške. On omogućava i svojstvo `name`, koje predstavlja naziv tipa generisane greške.

Prethodni kod daje izuzetak i ispisuje sledeću poruku:

```
ReferenceError: "badVarName is not defined occurred when assigning x."
```

Generisanje grešaka

Možete da generišete greške pomoću iskaza `throw`. U sledećem kodu prikazano je kako treba da dodate iskaze `throw` funkciji da biste generisali grešku, čak i ako nije došlo do greške u skriptu. Funkcija `sqrRoots()` prihvata jedan argument `x`, a zatim testira x da proveri da li je broj pozitivan (`positive`) i vraća string pomoću kvadratnog korena argumenta `x`. Ako `x` nije pozitivan broj, onda se generiše odgovarajuća greška i blok `catch` vraća grešku:

```
function sqrRoot(x) {
    try {
        if(x=="")    throw {message:"Can't Square Root Nothing"};
        if(isNaN(x)) throw {message:"Can't Square Root Strings"};
        if(x<0)      throw {message:"Sorry No Imagination"};
        return "sqrt("+x+") = " + Math.sqrt(x);
    } catch(err){
        return err.message;
    }
}
function writeIt(){
    console.log(sqrRoot("four"));
    console.log(sqrRoot(""));
    console.log(sqrRoot("4"));
    console.log(sqrRoot("-4"));
}
writeIt();
```

Ovo je izlaz konzole u kome su ispisane različite greške, koje su generisane na osnovu ulaznih podataka u funkciji `sqrRoot()`:

```
Can't Square Root Strings
Can't Square Root Nothing
sqrt(4) = 2
Sorry No Imagination
```

Upotreba rezervisane reči finally

Još jedna važna alatka za upravljanje izuzetkom je rezervisana reč `finally`. Ona se može dodati na kraju bloka `try/catch`. Nakon izvršenja blokova `try/catch`, ta rezervisana reč se uvek izvršava, bez obzira da li se greška dogodila, da li je generisana ili je u potpunosti izvršen blok `try`.

Ovo je primer korišćenja bloka `finally` unutar veb stranice:

```
function testTryCatch(value){
  try {
    if (value < 0){
      throw "too small";
```

Rezime

Razumevanje JavaScripta je od ključnog značaja kada se koriste okruženja Node.js, MongoDB, Express i Angular. U ovom poglavlju razmatrali smo osnovnu sintaksu JavaScript jezika (da biste shvatili koncepte u ostatku knjige), kreiranje objekata i funkcija i upotrebu stringova i nizova. Takođe ste naučili kako da primenite obradu grešaka u vašim skriptovima, koja je važna u okruženju Node.js.

Sledeće

U sledećem poglavlju prelazite direktno na podešavanje projekta Node.js. Takođe ćete naučiti nekoliko jezičkih idioma i videti jednostavne praktične primere.