Node.js

veb razvoj

V izdanje

David Herron

Veb razvoj na strani servera je lakši na platformi Node 14 pomoću praktičnih primera

kompjuter biblioteka

Packt>

Node.js
veb razvoj

Skenirajte QR kod, registrujte knjigu i osvojite nagradu

Node.js je vodeći izbor platforme za veb razvoj na strani servera, koji omogućava programerima da koriste iste alatke i paradigme za softver na strani servera i za softver na strani klijenta. Ažurirano peto izdanje knjige „Node.js veb razvoj“ fokusirano je na nove funkcije radnih okruženja Node.js 14, Express 14.x i ECMAScript, a vodi vas kroz koncepte, tehnike i najbolju praksu za korišćenje radnog okruženja Node.js za kreiranje aplikacija na strani servera.

Na početku knjige ćete upoznati koncepte izrade veb aplikacija na strani servera, koristeći Node.js. Naučićete kako da razvijete kompletnu Node.js veb aplikaciju pomoću pozadinskih baza podataka da biste mogli da istražite još nekoliko baza podataka. Implementiraćete aplikaciju na pravi veb server, uključujući platformu hostovanu u „oblaku“ koja je izrađena na AWS EC2 pomoću Terraforma i Dockera, a integrisaćete i druge alatke, kao što su Redis i NGINX. Kako budete napredovali, naučićete jedinično i funkcionalno testiranje, zajedno sa implementacijom testa infrastrukture pomoću Dockera. Na kraju, otkrićete kako da ojačate bezbednost Node.js aplikacije, kako da koristite Let's Encrypt za obezbeđivanje HTTPS usluge i kako da primenite nekoliko oblika bezbednosti u aplikacijama, zahvaljujući stručnoj praksi. U svakom poglavlju knjiga će vam pomoći da svoje znanje primenite u praksi tokom celog „životnog ciklusa“ razvoja veb aplikacije.

Kada u celosti pročitate ovu knjigu o radnom okruženju Node.js, steći ćete praktično znanje o Node.js veb razvoju da biste mogli da izradite i implementirate svoje aplikacije na javni veb hosting.

Naučićete sledeće:

- Instalirajte i koristite Node.js 14 i Express 4.17 za veb razvoj i implementaciju.
- Implementirajte REST servise, koristeći radni okvir Restify.
- Razvijte, testirajte i implementirajte mikroservise, koristeći Kubernetes i Node.js.
- Upoznajte mehanizme za skladištenje, kao što su MySQL, SQLite3 i MongoDB.
- Osigurajte svoje veb aplikacije, koristeći testiranje bez pretraživača pomoću Puppeteera.
- Implementirajte HTTPS, koristeći Let’s Encrypt i aplikaciju za poboljšanje bezbednosti pomoću Helmeta.

ISBN: 978-86-7310-556-7

Packt›

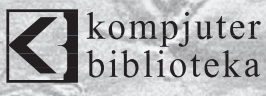
Prevod V izdanja

Node.js

veb razvoj

Veb razvoj na strani servera je lakši na platformi Node 14 pomoću praktičnih primera

David Herron



Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: David Herron

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2020.

Broj knjige: 533

Izdanje: Prvo

ISBN: 978-86-7310-556-7

Node.js Web Development

Fifth Edition

David Herron

ISBN 978-1-83898-757-2

O AUTORU

David Herron je softverski inženjer koji živi u mestu Silicon Valley, a učestvovao je u realizaciji značajnih projekata, kao što su X.400 server e-pošte, OpenJDK, Yahooova platforma za hosting Node.js aplikacija i servis za nadgledanje performansi solarnog panela. Zahvaljujući ovim projektima, David je sarađivao sa nekoliko kompanija dok mu nije dosadila komunikacija samo sa mašinama i dok nije počeo da čezne za ljudskom komunikacijom. Danas je on nezavisni pisac knjiga i blog postova koji se odnose na teme o tehnologiji, programiranju, električnim vozilima i čistim energetskim tehnologijama. Blog postovi se prikazuju na TechSparku, Mediumu, GreenTransportu i LongTailPipeu. Pomoću radnog okruženja Node.js on je kreirao statički generator veb sajtova AkashaCMS i alatku za generisanje e-knjiga AkashaEPUB.

O RECENZENTIMA

Esref Durna radi kao full stack inženjer od 2004. godine. Radio je kao prvi inženjer u nekoliko noovosnovanih preduzeća u regionu Sillicon Valley, a trenutno radi u American Expressu kao full stack inženjer koji je fokusiran na mikrofrontendove.

Migsar Navarro je full stack programer koji voli programiranje u JavaScriptu, jer smatra da je to jedan od najfleksibilnijih i najekspresivnijih programskih jezika. Uživa u izradi veb aplikacija koje su korisne i za korisnike i za programere koji ih kodiraju. Voli da deli znanje i da demistifikuje inženjerstvo, softversku arhitekturu i geografske informacione sisteme. Živi u Portu sa svojom devojkom i ćerkom.

„PACKT“ TRAŽI AUTORE KAO ŠTO STE VI

Ako ste zainteresovani da postanete autor za „Packt“, prijavite se na stranicu authors.packtpub.com. Sarađujemo sa hiljadama programera i tehničkih profesionalaca da bismo im pomogli da podele svoje mišljenje sa globalnom tehničkom zajednicom. Možete da podnesete osnovnu prijavu, da se prijavite za specifičnu temu za koju tražimo autore ili da pošaljete neke svoje ideje.

Predgovor

Node.js je JavaScript platforma na strani servera koja omogućava programerima da izrađuju brze i skalabilne aplikacije pomoću JavaScripta izvan veb pregledača. Ova platforma ima sve veću ulogu u svetu razvoja softvera. Prvo je korišćena kao platforma za serverske aplikacije, ali sada se često koristi u programerskim alatkama komandne linije, pa čak i u GUI aplikacijama, zahvaljujući kompletu alatki kao što je Electron. Node.js je „oslobodio" JavaScript od pregledača.

Node.js je pokrenut na vrhu ultrabrze JavaScript mašine u „srcu" „Googleovog" Chrome pregledača V8. Izvršavanje Node.js platforme prati genijalan model koji se zasniva na događajima i koji se često koristi za povećanje kapaciteta konkurentne obrade, bez obzira na upotrebu jednonitnog modela.

Primarni fokus Node.js su visokoperformantne i visokoskalabilne veb aplikacije, ali se primenjuje i u drugim oblastima. Na primer, Electron, tj. Node.js omotač oko Chromeovog mehanizma, omogućava Node.js programerima da kreiraju desktop GUI aplikacije i osnova je na kojoj su izrađene mnoge popularne aplikacije, uključujući uređivače Atom i Visual Studio Code, GitKraken, Postman, Etcher i desktop Slack klijent. Node.js je popularan na Internet of Things uređajima. Njegova arhitektura je posebno pogodna za razvoj mikroservisa i često pomaže da se formira serverska strana full-stack aplikacija.

„Ključ" za postizanje visoke propusne moći na jednonitnim sistemima je Node.js model za asinhrono izvršavanje. Ovaj model se znatno razlikuje od platformi koje se oslanjaju na programske niti za konkurentno programiranje, jer ti sistemi često troše dodatnu memoriju i složeni su. Nasuprot tome, Node.js koristi jednostavan model slanja događaja koji se prvobitno oslanjao na funkcije povratnog poziva, ali se danas oslanja na JavaScript Promise objekat i na asinhrone funkcije.

Pošto je Node.js platforma na vrhu Chromeove V8 mašine, ona može brzo da usvoji najnovija dostignuća u jeziku JavaScript. Node.js osnovni tim blisko sarađuje sa V8 timom, omogućavajući brzo usvajanje novih funkcija na jeziku Java

onako kako su one implementirane u V8. Node.js 14.x je najnovije izdanje za koje je i napisana ova knjiga.

Kome je namenjena ova knjiga

Serverski inženjeri će možda JavaScript smatrati odličnim alternativnim programskim jezikom. Zahvaljujući napretku JavaScripta, on je odavno prestao da bude jednostavan „igrački" jezik pogodan samo za animiranje dugmadi u pregledačima. Sada možemo da izradimo velike sisteme pomoću tog jezika, a Node.js ima mnogo ugrađenih funkcija (poput vrhunskog sistema modula) koje pomažu u većim projektima.

Programerima koji imaju iskustvo u JavaScriptu na strani pregledača može biti privlačno da prošire svoje horizonte i da uključe razvoj na strani servera, koristeći ovu knjigu.

Šta obuhvata ova knjiga

Poglavlje 1, „O radnom okruženju Node.js", uvodi vas u Node.js platformu. Obuhvata njenu upotrebu, izbor tehnološke arhitekture na platformi Node.js, njenu istoriju, istoriju JavaScript na strani servera, razloge zašto bi JavaScript trebalo da bude „oslobođen" od pregledača i važna nedavna dostignuća na JavaScript „sceni".

U Poglavlju 2, „Podešavanje radnog okruženja Node.js", razmatramo podešavanje Node.js programerskog okruženja. To uključuje instaliranje platforme Node.js na Windowsu, macOSu i Linuxu. Obuhvaćene su važne alatke, uključujući sisteme za upravljanje paketima `npm` i `yarn`i Babel, koji se koriste za prevođenje modernog JavaScripta u oblik koji se može pokrenuti na starijim JavaScript implementacijama.

U Poglavlju 3, „Istraživanje Node.js modula", razmatramo modul kao jedinicu modularnosti u Node.js aplikacijama. „Zaronićemo" duboko u razvoj Node.js modula i koristićemo `npm` za održavanje zavisnosti. Upoznaćete novi format modula, tj. ES6 modul, i saznaćete kako se koristi na platformi Node.js kada ga ona lokalno podržava.

U Poglavlju 4, „HTTP serveri i klijenti", započinjemo istraživanje veb razvoja pomoću platforme Node.js. Razvićemo nekoliko malih veb serverskih i klijentskih aplikacija u radnom okruženju Node.js. Koristićemo Fibonačijev algoritam u Node.js aplikaciji za istraživanje efekata teških izračunavanja, koja dugo traju. Naučićete i nekoliko strategija za smanjenje rizika i steći ćete osnovno iskustvo u razvoju REST servisa.

U Poglavlju 5, „Vaša prva aplikacija Express“, započinjemo glavno „putovanje“ ove knjige u kojoj razvijamo aplikaciju za kreiranje i uređivanje beležaka. U ovom poglavlju pokrećemo osnovnu aplikaciju za beleške i prvo ćemo koristiti radni okvir Express.

U Poglavlju 6, „Implementacija paradigme Mobile-First“, koristimo radni okvir Bootstrap V4 za implementaciju prilagodljivog veb dizajna u aplikaciji za beleške. Ovo uključuje integrisanje popularnog skupa ikona i korake neophodne za prilagođavanje Bootstrapa.

U Poglavlju 7, „Čuvanje i učitavanje podataka“, istražićemo nekoliko mehanizama baze podataka i metod za lako menjanje baza podataka po želji. Cilj je robusno čuvanje podataka na disku.

U Poglavlju 8, „Autentifikacija korisnika pomoću mikroservisa“, dodajemo autentifikaciju korisnika aplikaciji za beleške. Naučićete kako da upravljate prijavljivanjem i odjavljivanjem pomoću PassportJS-a. Autentifikacija je podržana i za lokalno uskladištena ovlašćenja korisnika i za upotrebu OAutha na Twitteru.

Poglavlje 9, „Dinamička klijent/server interakcija pomoću biblioteke Socket.IO“, omogućava korisnicima da razgovaraju jedni sa drugima u realnom vremenu. Koristićemo popularni radni okvir Socket.IO za dinamičku interakciju između klijenta i servera, radi podrške dinamičkim ažuriranjima sadržaja i jednostavnom sistemu komentarisanja. Svi korisnici se dinamički ažuriraju u pseudorealnom vremenu, pa možete da upoznate dinamičko ažuriranje u realnom vremenu.

U Poglavlju 10, „Implementacija Node.js aplikacija na Linux servere“, započinjemo „put“ implementacije. U ovom poglavlju koristićemo tradicionalne metode implementiranja pozadinskih servisa na operativni sistem Ubuntu pomoću sistema Sistemd.

U Poglavlju 11, „Implementacija Node.js mikroservisa pomoću Dockera“, započinjemo istraživanje implementacije u „oblaku“ pomoću Dockera, radi tretiranja aplikacije za beleške kao klastera mikroservisa.

Poglavlje 12, „Implementacija Docker Swarma na servisu AWS EC2 pomoću Terraforma“, doslovno nas vodi u „oblak“, pri čemu razmatramo izradu sistema za hosting u „oblaku“ pomoću AWS EC2 sistema. Koristićemo popularnu alatku Terraform za kreiranje i upravljanje EC2 klasterom i naučićete kako da gotovo u potpunosti automatizujete implementaciju klastera Docker Swarma pomoću Terraform funkcija.

U Poglavlju 13, „Testiranje koda i funkcionalno testiranje“, istražićemo tri načina testiranja: testiranje koda, REST testiranje i funkcionalno testiranje. Koristićemo popularne radne okvire za testiranje Mocha i Chai da bismo pokrenuli testne

slučajeve na sva tri načina. Za testiranje funkcionalnosti koristićemo Puppeteer popularni radni okvir za automatizaciju izvršenja testa u Chrome instanci.

U Poglavlju 14, „Bezbednost u Node.js aplikacijama“, integrisaćemo bezbednosne tehnike i alatke za smanjenje rizika od neovlašćenog pristupa. Prvo ćemo da primenimo HTTPS na AWS EC2 implementaciju pomoću Let's Encrypta. Zatim ćemo razmotriti nekoliko alatki u radnom okruženju Node.js za implementaciju bezbednosnih postavki i najbolju praksu za Docker i AWS okruženja.

Da biste dobili maksimum iz ove knjige

Osnovni zahtev je instaliranje platforme Node.js i uređivača teksta orijentisanog na programera. Nije vam potreban neki moderan uređivač; čak će i vi/vim uređivač biti dovoljan. Pokazaćemo vam kako da instalirate sve što vam je potrebno, a, pošto je sve otvorenog koda, nema prepreka za „ulazak“.

Najvažnija „alatka“ je vaš mozak.

SOFTVER/HARDVER KOJI JE OBUHVAĆEN U OVOJ KNJIZI	OS ZAHTEVI
Node.js i povezani radni okviri, kao što su Express, Sequelize i Socket.IO	bilo koji
alatke za upravljanje paketima `npm/yarn`	bilo koji
Python i C/C++ kompajleri	bilo koji
MySQL, SQLite3 i MongoDB baze podataka	bilo koji
Docker	bilo koji
Multipass	bilo koji
Terraform	bilo koji
Mocha i Chai	bilo koji

Svaki deo softvera je već dostupan. Za C/C++ kompajlere na Windowsu i macOS-u biće vam potrebno programsko okruženje Visual Studio (Windows) ili programsko okruženje Xcode (macOS); oba su dostupna besplatno.

Bilo bi korisno da imate iskustvo u JavaScript programiranju. JavaScript je prilično jednostavan jezik za učenje ako već poznajete druge programske jezike.

Preuzimanje datoteka sa primerima koda

Iako želimo da obezbedimo identične isečke koda u ovoj knjizi i u spremištu, na nekim mestima će biti manjih razlika. Spremište može da sadrži komentare, iskaze za debagovanje ili (komentarisane) alternativne implementacije koje nisu prikazane u ovoj knjizi.

Datoteke sa primerima koda za ovu knjigu možete da preuzmete sa našeg sajta:

https://bit.ly/3cUgp41

Kada je datoteka preuzeta, raspakujte ili ekstrahujte direktorijum, koristeći najnoviju verziju:

- WinRAR/7-Zip za Windows
- Zipeg/iZip/UnRarX za Mac
- 7-Zip/PeaZip za Linux

Paket koda za knjigu takođe se nalazi na GitHubu na adresi:

https://bit.ly/2EWOeVL

U slučaju da postoji njegovo ažuriranje, kod će biti ažuriran na postojećem GitHub spremištu.

Upotrebljene konvencije

Postoji veliki broj konvencija teksta koje su upotrebljene u ovoj knjizi.

`CodeInText` ukazuje na reči koda u tekstu, nazive tabele baze podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, nazive putanje, skraćene URL-ove, korisnički unos i Twitter postove. Evo i primera: „Prvo izmenite `package.json` da biste dobili sledeći odeljak `scripts`."

Blok koda je prikazan na sledeći način:

```
function readFile(filename) {
  return new Promise((resolve, reject) => {
    fs.readFile(filename, (err, data) => {
      if (err) reject(err);
      else resolve(data);
    });
  });
}
```

Kada želimo da privučemo pažnju na određeni deo bloka koda, relevantne linije ili stavke su ispisane zadebljanim slovima:

```
function asyncFunction(arg1, arg2) {
  return new Promise((resolve, reject) => {
    // perform some task or computation that's asynchronous
    // for any error detected:
    if (errorDetected) return reject(dataAboutError);
    // When the task is finished resolve(theResult);
  });
};
```

Svaki unos komandne linije ili ispis će biti prikazan na sledeći način:

```
$ mkdir notes
$ cd notes
```

Zadebljana slova ukazuju na novi termin, važnu reč ili reči koje vidite na ekranu. Na primer, reči u menijima ili okvirima za dijalog prikazane su u tekstu na sledeći način: „Kliknite na dugme **Submit**.“

Napomene ili važna obaveštenja prikazani su ovako.

Saveti i trikovi su prikazani ovako.

Kontaktirajte sa nama

Povratne informacije od naših čitalaca su uvek dobrodošle.

Štamparske greške - Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške su moguće. Ako pronađete grešku u ovoj knjizi, bili bismo zahvalni ako biste nam to prijavili. Posetite stranicu knjige na našem sajtu:

https://bit.ly/3naLS6S

i napišite komentar.

Piraterija - Ako pronađete ilegalnu kopiju naših knjiga na Internetu, u bilo kojoj formi, molimo vas da nas o tome obavestite i da nam pošaljete adresu lokacije ili naziv veb sajta. Molimo vas, kontaktirajte sa nama na adresi kombib@gmail.com i pošaljite nam link ka sumnjivom materijalu.

Recenzija

Kada pročitate i upotrebite ovu knjigu, zašto ne biste napisali vaše mišljenje o njoj na sajtu sa kojeg ste je poručili? Potencijalni čitaoci će tada moći da upotrebe vaše mišljenje da bi se odlučili o kupovini, mi u „Packtu“ ćemo znati šta mislite o našoj produkciji, a naši autori će moći da vide povratne informacije o svojoj knjizi.

Više informacija o „Packtu“ naći ćete na sajtu packt.com.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popusta i učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

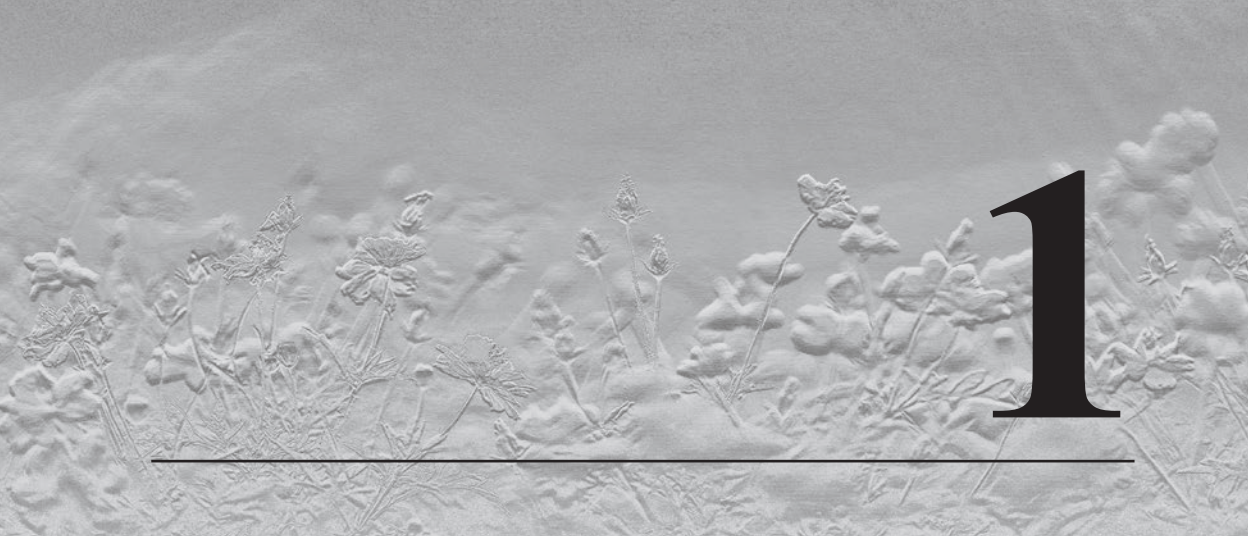
Deo 1

Uvod u radno okruženje Node.js

Ovo je pregled radnog okruženja Node.js na visokom nivou. Čitalac će preduzeti prve korake za upotrebu platforme Node.js.

Ovaj deo sadrži sledeća poglavlja:

- **Poglavlje 1**, „O radnom okruženju Node.js“
- **Poglavlje 2**, „Podešavanje radnog okruženja Node.js“
- **Poglavlje 3**, „Istraživanje Node.js modula“
- **Poglavlje 4**, „HTTP serveri i klijenti“



1

O radnom okruženju Node.js

JavaScript je na dohvat ruke svakog frontend veb programera, što ga čini vrlo popularnim programskim jezikom, tj. toliko popularnim da se pretpostavlja da je namenjen klijentskom kodu na veb stranama. Velike su šanse da ste, pošto što ste izabrali ovu knjigu, čuli za Node.js programsku platformu za kodiranje u JavaScriptu izvan veb pregledača. Sada star oko 10 godina, Node.js postaje platforma za „zrelo" programiranje, koja se često koristi i u velikim i u malim projektima.

U ovoj knjizi će biti predstavljen Node.js. Kada u celosti pročitate ovu knjigu, saznaćete o kompletnom „životnom ciklusu" razvoja veb aplikacija na strani servera pomoću platforme Node.js, od koncepta do implementacije i bezbednosti. U ovoj knjizi pretpostavljeno je sledeće:

- da već znate kako se piše softver
- da poznajete JavaScript
- da znate nešto o razvoju veb aplikacija u drugim programskim jezicima

Kada ocenjujemo neku novu alatku za programiranje, da li ćemo misliti da je ona bolja samo zato jer je nova i popularna? Možda neki ljudi na taj način ocenjuju neku novu alatku, ali zreli pristup je upoređivanje jedne alatke sa drugom. O tome se govori u ovom poglavlju, pri čemu se predstavlja tehnička osnova za korišćenje platforme Node.js. Pre nego što pređemo na kod, moramo razmotriti šta je Node.js i kako se uklapa u celokupno tržište alatki za razvoj softvera. Zatim ćemo „zaroniti" pravo u razvoj radnih aplikacija i shvatićete da je često najbolji način za učenje razvoja aplikacija istraživanje radnog koda.

U ovom poglavlju ćemo obraditi sledeće teme:

- −uvod u radno okruženje Node.js
- −šta možete da uradite pomoću platforme Node.js
- −zašto treba da koristite Node.js
- −arhitektura platforme Node.js
- −performanse, iskorišćenost i skalabilnost paltforme Node.js
- −Node.js, arhitektura mikroservisa i testiranje
- −implementacija modela aplikacije twelve-factor pomoću platforme Node.js

Pregled platforme Node.js

Node.js je uzbudljiva nova platforma za razvoj veb aplikacija, aplikacijskih servera, bilo koje vrste mrežnog servera ili klijenta i za programiranje opšte namene. Dizajnirana je za ekstremnu skalabilnost u umreženim aplikacijama pomoću genijalne kombinacije JavaScripta na strani servera, asinhronog I/O i asinhronog programiranja.

Iako ima samo 10 godina, Node.js je brzo „sazreo“ i sada „igra“ značajnu ulogu. Velike i male kompanije koriste ga i za velike i za male projekte. Na primer, „PayPal“ je konvertovao mnoge servise iz Jave u Node.js.

Node.js arhitektura odstupa od tipičnog izbora koji su napravile druge aplikacijske platforme. Tamo gde se programske niti koriste za skaliranje aplikacije radi punjenja CPU-a Node.js briše programske niti, zbog njihove inherentne složenosti. Tvrdi se da je u arhitekturama zasnovanim na jednoj programskoj niti memorijski otisak nizak, propusna moć je visoka, profil latencije pod opterećenjem je bolji, a programski model je jednostavniji. Platforma Node.js je u fazi brzog rasta i mnogi ljudi je posmatraju kao drugačiju alternativu tradicionalnim arhitekturama veb aplikacija koje koriste Java, PHP, Python ili Rubi on Rails.

U osnovi, Node.js je samostalna JavaScript mašina sa ekstenzijama, koja je pogodna za programiranje opšte namene i koja ima jasan fokus na razvoj serverskih aplikacija. Iako upoređujemo Node.js sa platformama aplikacijskog servera, on nije aplikacijski server. Umesto toga, Node.js je izvršavanje programiranja koje je slično Python, Go ili Java SE izvršavanju. Iako postoje radni okviri za veb aplikacije i aplikacijski serveri napisani na platformi Node.js, oni su samo sistemi za izvršavanje JavaScript programa.

Kada je reč o arhitekturi platforme, Node.js je vođen događajima, a ne višestrukim programskim nitima. Node.js arhitektura se zasniva na otpremanju blokirajućih operacija u jednonitnu petlju događaja, a rezultati se vraćaju pozivaocu kao događaju koji poziva funkciju hendlera događaja. U većini slučajeva se događaj pretvara u promis koji upravlja funkcijom `async`. Pošto je Node.js zasnovan na Chromeovoj V8 JavaScript mašini, poboljšanja performansi i funkcije implementirane u Chromeu brzo se prenose na platformu Node.js.

Osnovni Node.js moduli su dovoljno opšti za implementaciju bilo koje vrste servera koji izvršava neki TCP ili UDP protokol, bez obzira da li je to **Domain Name System (DNS)**, HTTP, **internet relay chat (IRC)** ili FTP. Iako podržava razvoj servera ili klijenata na Internetu, najčešće se koristi za redovnu izradu veb sajtova, umesto tehnologija kao što su Apache/PHP ili Rails stack, ili za dopunu postojećih veb sajtova - na primer, dodavanje ćaskanja u realnom vremenu ili nadgledanje postojećih veb sajtova lako se mogu izvesti pomoću biblioteke Socket.IO za Node.js. Zahvaljujući svojoj laganoj „prirodi“ i visokim performansama, Node.js se često smatra „**lepkom“**.

Posebno intrigantna kombinacija je razmeštanje malih servisa na modernoj infrastrukturi u „oblaku“ pomoću alatki, kao što su Docker i Kubernetes, ili pomoću funkcijskih platformi koje se ponašaju kao servis, kao što je AWS Lambda. Node.js dobro funkcioniše kada deli veliku aplikaciju na mikroservise koji se mogu lako implementirati na veći broj računara.

Pošto sada razumete šta je platforma Node.js, istražićemo je malo detaljnije.

Mogućnosti platforme Node.js

Node.js je platforma za pisanje JavaScript aplikacija izvan veb pregledača. Ovo nije JavaScript okruženje koje ste upoznali u veb pregledačima! Dok Node.js izvršava isti JavaScript jezik koji koristimo u pregledačima, on nema neke funkcije povezane sa pregledačem. Na primer, u radnom okruženju Node.js nije ugrađen HTML DOM.

Izuzev izvorne mogućnosti za izvršavanje JavaScripta, ugrađeni moduli obezbeđuju mogućnosti sledeće vrste:

- alatke komandne linije (u stilu skripta komandnog okruženja)
- interaktivno-terminalni stil programa - odnosno, **read-eval-print loop (REPL)**
- odlične funkcije kontrole procesa za nadgledanje podređenih procesa
- objekat bafera za obradu binarnih podataka

- TCP ili UDP socketi sa sveobuhvatnim povratnim pozivima koji su vođeni događajima
- DNS pretraživanje
- HTTP, HTTPS i HTTP/2 klijent-server na vrhu TCP pristupa bibliotečkom sistemu datoteke
- ugrađena podrška za testiranje rudimentarnog koda pomoću asertacija

Mrežni sloj platforme Node.js je niskog nivoa, a jednostavan je za upotrebu - na primer, HTTP moduli omogućavaju da napišete HTTP server (ili klijent) korišćenjem nekoliko linija koda. To je moćno, ali, kao programera, „stavlja" vas vrlo blizu zahtevima za protokol i omogućava da implementirate upravo ona HTTP zaglavlja u koja bi trebalo da vratite odgovore na zahteve.

„Obični" programeri veb aplikacija ne moraju da koriste HTTP niskog nivoa ili druge protokole; umesto toga, skloniji su produktivnijem radu pomoću interfejsa višeg nivoa - na primer, PHP koderi „pretpostavljaju" da veb serveri Apache/Nginx/ i slični već postoje da bi obezbedili HTTP i da ne moraju da primene HTTP deo servera steka. Nasuprot tome, Node.js programeri implementiraju HTTP server na koji je priključen njihov kod aplikacije.

Da bi situacija bila pojednostavljena, zajednica Node.js ima nekoliko radnih okvira veb aplikacija, poput Expressa, koji obezbeđuju interfejse višeg nivoa, a koji zahtevaju „obični" programeri. Možete brzo da konfigurišete HTTP server pomoću ugrađenih mogućnosti, kao što su sesije, „kolačići", opsluživanje statičkih datoteka i evidentiranje, ostavljajući programerima da se fokusiraju na svoju poslovnu logiku. Ostali radni okviri obezbeđuju podršku za OAuth 2 ili se fokusiraju na REST API-e i slično.

Zajednica ljudi koji koriste Node.js „izgradila" je neverovatnu raznolikost „stvari" na toj osnovi.

Šta ljudi rade pomoću platforme Node.js?

Node.js nije ograničen na razvoj aplikacija za veb servise; Node.js zajednica ga je usmerila u mnogim drugim pravcima:

- **izrada alatki** - Node.js je postao popularan izbor za razvoj alatki komandne linije koje se koriste u razvoju softvera ili u komunikaciji sa infrastrukturom usluga. Grunt, Gulp i Webpack široko koriste frontend programeri za izradu resursa za veb sajtove. Babel se često koristi za prenošenje modernog ES-2016 koda za pokretanje na starijim pregledačima. Popularni CSS optimizatori i procesori, poput PostCSS-a, napisani su u radnom okruženju Node.js. Statički sistemi za generisanje veb sajtova, kao što su Metalsmith, Punch i AkashaCMS, pokreću se u komandnoj liniji i generišu sadržaj veb sajta koji otpremite na veb server.

- **testiranje veb korisničkog interfejsa** - Biblioteka Puppeteer omogućava kontrolu nad „bezglavom“ (headless) Chrome instancom veb pregledača. Pomoću nje možete razvijati Node.js skriptove kontrolisanjem modernog kompletnog veb pregledača. Neki tipični primeri upotrebe biblioteke su veb skrepovanje i testiranje veb aplikacija.
- **desktop aplikacije** - I Electron i **node-webkit (NW.js)** su radni okviri za razvoj desktop aplikacija za Windows, macOS i Linux. Ovi radni okviri omotani bibliotekama Node.js koriste veliki deo Chromea za razvoj desktop aplikacija pomoću veb UI tehnologija. Aplikacije su napisane pomoću modernih HTML5, CSS3 i JavaScript programskih jezika i mogu da koriste napredne veb radne okvire, kao što su Bootstrap, React, VueJS i AngularJS. Mnogo popularnih aplikacija izrađeno je pomoću Electrona, uključujući Slack desktop klijentsku aplikaciju, Atom, Microsoft Visual Code programske uređivače, Postman REST klijent, GitKraken GIT klijent i Etcher, što omogućava neverovatno lako snimanje OS slika na fleš uređajima da bi bile pokrenute na računarima sa jednom pločom.
- **mobilne aplikacije** - Projekat „Node.js for Mobile System“ omogućava da razvijete aplikacije za pametne telefone ili tablet računare, koristeći Node.js za iOS i Android. Appleova App Store pravila onemogućavaju dodavanje JavaScript mašine pomoću JIT mogućnosti, što znači da se uobičajeni Node.js ne može koristiti u iOS aplikaciji. Za razvoj iOS aplikacija u projektu se koristi Node.js-on-ChakraCore da bi bila izbegnuta pravila App Storea. Za razvoj Android aplikacija u projektu se koristi uobičajeni Node.js na Androidu. U vreme pisanja ove knjige, projekat je u ranoj fazi razvoja, ali izgleda dobro.
- **Internet of Things (IoT)** - Node.js je veoma popularan jezik za projekte Internet-of-Things, a Node.js funkcioniše na većini računara zasnovanih na ARM-u. Najjasniji primer je NodeRED projekat koji obezbeđuje grafičko programsko okruženje, pri čemu omogućava da iscrtavate programe povezivanjem blokova. Sadrži mehanizme ulaza i izlaza orijentisane na hardver - na primer, mehanizme za interakciju sa **ulazom/izlazom opšte namene (GPIO - General Purpose I/O)** na računarima sa jednom pločom Raspberry Pi ili Beaglebone.

Možda već koristite Node.js aplikacije, a da to niste ni primetili! JavaScript ima mesto izvan veb pregledača, a ne samo na platformi Node.js.

JavaScript na strani servera

Prestanite da se češete po glavi i mrmljate u sebi: „Šta jezik pregledača radi na serveru?“. Istina, on ima dugu i uglavnom nepoznatu istoriju izvan pregledača. JavaScript je programski jezik, baš kao i bilo koji drugi programski jezik i bolje je da se zapitate: „Zašto on treba da ostane ’zarobljen’ u veb pregledačima?“.

Na početku nastanka Veba alatke za pisanje veb aplikacija bile su u početnoj fazi. Neki programeri su eksperimentisali sa pisanjem CGI skripta, koristeći Perlili TCL, a PHP i Java jezici su bili tek razvijeni. Čak i onda je JavaScript korišćen na strani servera. Jedan od prvih servera veb aplikacija bio je Netscapeov LiveWire server, koji je koristio JavaScript. Neke verzije „Microsoftovog" ASP-ja koriste JScript, tj. svoju verziju JavaScripta. Najnoviji JavaScript projekat na strani servera je aplikacijski radni okvir RingoJS u Java „univerzumu". Java 6 i Java 7 se isporučuju sa Rhino JavaScript mašinom. U Javi 8 Rhino je odbačen u korist novije Nashorn JavaScript mašine.

Drugim rečima, JavaScript izvan pregledača nije novina, čak i ako je neuobičajen.

Saznali ste da je Node.js platforma za pisanje JavaScript aplikacija izvan veb pregledača. Node.js zajednica koristi ovu platformu za ogroman niz vrsta aplikacija, tj. za mnogo više vrsta nego što je prvobitno zamišljeno. Ovo dokazuje da je Node.js popularan, ali moramo, ipak, razmotriti tehničku osnovu za njegovo korišćenje.

Zašto treba da koristite Node.js?

Zašto da izaberete Node.js među mnogim dostupnim platformama za razvoj veb aplikacija? Mnogo je stekova koje možete izabrati, a šta je to što Node.js uzdiže iznad ostalih? Odgovor na ovo pitanje saznaćete u narednim odeljcima.

Popularnost

Node.js brzo postaje popularna razvojna platforma koju usvaja mnogo velikih i malih „igrača". Jedan od tih „igrača" je „PayPal", koji svoj postojeći sistem zasnovan na Javi zamenjuje sistemom koji je napisan u radnom okruženju Node.js. Ostali veliki „usvojitelji" platforme Node.js uključuju Walmartovu platformu za e-trgovinu na mrežama LinkedIn i eBay.

Više o ovome možete pročitati na „PayPalovom" post blogu na adresi: https://bit.ly/3jJJlhG

Prema tvrdnjama NodeSourcea, upotreba platforme Node.js brzo raste (za više informacija posetite sajt https://bit.ly/3iyAZs6). Dokazi za ovaj rast uključuju povećanje propusne moći za preuzimanje Node.js izdanja, povećanje aktivnosti u GitHub projektima povezanim sa platformom Node.js i još mnogo štošta.

Interesovanje za sam JavaScript je i dalje veoma veliko, ali ne raste već godinama, mereno obimom pretrage (Google Insights) i njegovom upotrebom kao veštine programiranja (Dice Skills Center). Interesovanje za Node.js naglo raste, ali pokazuje znakove manjeg pada.

Za više informacija posetite https://bit.ly/33uMts7

Najbolje je da ne pratite mnoštvo ljudi, jer su oni različiti, a svaki od njih tvrdi da njihova softverska platforma radi odlične „stvari". Node.js radi neke super „stvari", ali ono što je još važnije je njegova tehnička zasluga.

JavaScript svuda

Imati isti programski jezik na serveru i klijentu dugogodišnji je san na Vebu. Ovaj san seže u rane dane Jave kada su Java apleti u pregledaču bili frontend za serverske aplikacije napisane u Java jeziku, a JavaScript je prvobitno bio zamišljen kao lak skriptni jezik za te aplete. Java nikada nije postala popularna kao programski jezik na strani klijenta, pa čak i fraza „Java apleti" bledi iz sećanja kao napušteni model aplikacije na strani klijenta. Zaključili smo da je osnovni jezik u pregledaču na strani klijenta JavaScript, a ne Java. Uobičajeno je da su frontend JavaScript programeri bili u različitom jezičkom „univerzumu" od tima na strani servera koji je verovatno pisao kodove u programskim jezicima PHP, Java, Ruby ili Python.

Vremenom su JavaScript mašine u pregledaču postale neverovatno moćne, pri čemu omogućavaju pisanje sve složenijih aplikacija na strani pregledača. Koristeći platformu Node.js, konačno možemo da implementiramo aplikacije pomoću istog programskog jezika na klijentu i serveru, tako što imamo JavaScript na oba kraja Veba - u pregledaču i na serveru.

Zajednički jezik za frontend i backend obezbeđuje nekoliko potencijalnih prednosti:

- Na oba kraja „žice" može raditi isto osoblje za programiranje.
- Kod se može lakše premestiti između servera i klijenta.
- Uobičajeni su formati podataka (JSON) između servera i klijenta.
- Uobičajene su softverske alatke za server i klijent.

- Uobičajene su alatke za testiranje ili izveštavanje o kvalitetu za server i klijent.
- Prilikom pisanja veb aplikacija šabloni za prikaz mogu se koristiti na obe strane.

JavaScript jezik je veoma popularan, zbog svoje sveprisutnosti u veb pregledačima. Ako ga uporedimo sa drugim programskim jezicima, možemo videti da JavaScript ima mnogo modernih, naprednih jezičkih koncepata. Zahvaljujući popularnosti JavaScripta, postoji veliki broj talentovanih i iskusnih JavaScript programera.

Iskorišćavanje „Googleovog" ulaganja u V8

Da bi Chrome postao popularan i odličan veb pregledač, „Google" je uložio u V8 da bi napravio superbrzu JavaScript mašinu. Zbog toga, on ima ogromnu motivaciju da nastavi da poboljšava V8. V8 je JavaScript mašina za Chrome, a može se izvršavati i samostalno.

Node.js je izrađen na vrhu V8 JavaScript mašine, pri čemu mu se omogućava da iskoristi sve što funkcioniše na V8. Node.js je mogao da brzo usvoji nove jezičke funkcije JavaScripta, jer ih je implementirao V8, i da ostvari „pobede" u performansama, zbog istog razloga.

Jednostavniji, asinhroni model koji je vođen događajima

Smatra se da arhitektura Node.js, koja je izgrađena na jednoj programskoj niti izvršenja pomoću genijalnog modela asinhronog programiranja koji je vođen događajima i brzom JavaScript mašinom, ima manje indirektne troškove nego arhitekture zasnovane na više programskih niti. Ostali sistemi koji koriste programske niti obično su složeni i imaju veće indirektne troškove, za razliku od platforme Node.js. O ovome će biti reči kasnije u ovom poglavlju.

Arhitektura mikroservisa

Nova senzacija u razvoju softvera su mikroservisi. Oni su fokusirani na deljenje velike veb aplikacije na male, usko fokusirane servise, koje mali timovi mogu lako razviti. Iako mikroservisi nisu baš nova ideja, već više predstavljaju ponovno definisanje starih računarskih modela klijent-server, uzorak mikroservisa dobro se uklapa u agilne tehnike upravljanja projektima i omogućava detaljniju implementaciju aplikacija.

Node.js je odlična platforma za implementaciju mikroservisa koje ćemo razmatrati kasnije.

Node.js je snažniji nakon velikog raskola i neprijateljskih kopija

Tokom 2014. i 2015. godine zajednica Node.js se suočila sa velikom podelom kada je reč o politici, smeru i kontroli. Projekat **io.js** bio je neprijateljska kopija koju je pokrenula grupa koja je želela da uključi nekoliko funkcija i izmeni proces donošenja odluka. Krajnji rezultat bio je spajanje Node.js i io.js spremišta, tj. nezavisna fondacija Node.js za pokretanje emisije i zajednica koje rade na Node.js unapređenju.

Konkretan rezultat izlečenja tih „rana" je brzo usvajanje novih funkcija jezika ECMAScript. V8 mašina je brzo prihvatila ove nove funkcije da bi unapredila stanje veb razvoja. Node.js tim prihvata ove funkcije onoliko brzo koliko se prikazuju u V8 mašini, što znači da promisi i funkcije `async` brzo postaju stvarnost za Node.js programere.

Suština je da zajednica Node.js ne samo da je „preživela" io.js kopiju i kasnije aio.js kopiju, već su zajednica i platforma koju je razvijala postajale snažnije kao rezultat toga.

U ovom odeljku ste saznali nekoliko razloga zbog kojih treba da koristite Node.js. Ne samo da je Node.js popularna platforma, koja ima snažnu zajednicu iza sebe, već postoje i ozbiljni tehnički razlozi zbog kojih treba da je koristite. Njena arhitektura ima neke ključne tehničke prednosti, pa ćemo ih sada detaljnije razmotriti.

Node.js arhitektura koja je vođena događajima

„Žestoke" performanse platforme Node.js su posledica asinhrone arhitekture koja je vođena događajima i upotrebe V8 JavaScript mašine. To joj omogućava istovremeno rešavanje više zadataka, poput „žongliranja" zahtevima iz više veb pregledača. Prvobitni tvorac platforme Node.js Ryan Dahl pratio je sledeće ključne stavke:

- Jednonitni model programiranja koji je vođen događajima ima jednostavan kod, manje složenosti i niže indirektne troškove, za razliku od aplikacijskih servera koji se oslanjaju na programske niti za obradu više konkurentnih zadataka.

- Pretvaranjem poziva blokirajućih funkcija u izvršavanje asinhronog koda možete konfigurisati sisteme tako da emituju događaj kada je zahtev za blokiranje ispunjen.
- V8 JavaScript mašinu možete koristiti iz pregledača Chrome, a sav posao je fokusiran na poboljšanje V8; sva poboljšanja performansi u V8, dakle, koriste platformu Node.js.

U većini aplikacijskih servera konkurentnost, odnosno mogućnost da se obrađuju višestruki konkurentni zahtevi, implementirana je pomoću višenitne arhitekture. U takvom sistemu svaki zahtev za podatke ili bilo koji drugi poziv blokirajuće funkcije uzrokuje suspenziju trenutne programske niti izvršavanja i čekanje na rezultat. Za upravljanje konkurentnim zahtevima potrebno je da postoji više programskih niti izvršavanja. Kada je jedna programska nit suspendovana, može se izvršiti i druga programska nit. To izaziva „bućkanje", jer server aplikacija pokreće i zaustavlja programske niti za obradu zahteva. Svaka suspendovana programska nit (koja, obično, čeka na završetak operacije ulaza/izlaza) troši puni stek poziva memorije, pri čemu programske niti dodaju složenost aplikacijskom serveru, ali i nameću nepotrebni posao serveru.

Da biste sve ovo shvatili, pogledajte u nastavku primer koji dao Ryan Dahl, tvorac platforme Node.js. U svojoj prezentaciji „Cinco de NodeJS" u maju 2010. godine (https://bit.ly/2HYkSHH) Dahl nas je pitao šta se dešava kada izvršimo liniju koda poput ove:

```
result = query('SELECT * from db.table');
// operate on the result
```

Naravno, program se zaustavlja u ovom trenutku dok sloj baze podataka šalje upit u bazu podataka i čeka rezultat ili grešku. Ovo je primer poziva blokirajuće funkcije. U zavisnosti od upita, pauza može biti prilično duga (nekoliko milisekundi, što je „večnost" u računarskom vremenu). Ova pauza je loša, jer programska nit izvršavanja ne može ništa učiniti dok čeka da stigne rezultat. Ako se vaš softver izvršava na platformi koja ima jednu programsku nit, čitav server će biti blokiran i neće reagovati. Ako je, umesto toga, vaša aplikacija pokrenuta na platformi servera zasnovanoj na programskim nitima, potreban je prekidač konteksta niti da bi bili zadovoljeni svi ostali zahtevi koji stignu. Što je veći broj neuobičajenih veza sa serverom, to je veći broj prekidača konteksta niti. Promena konteksta nije besplatna, jer za više programskih niti potrebno je više memorije po stanju niti i više vremena koje CPU troši na dodatni posao upravljanja nitima.

Ključna inspiracija koja je vodila originalni razvoj platforme Node.js bila je jednostavnost sistema koji ima jednu programsku nit. Jedna programska nit izvršavanja znači da server nema složenost višenitnih sistema. Ovaj izbor je značio da Node.js zahteva model koji je vođen događajima za upravljanje konkurentnim zadacima. Umesto koda koji čeka rezultate zahteva za blokiranje, poput preuzimanja podataka iz baze podataka, događaj se šalje u hendler događaja.

Upotreba programskih niti za implementaciju konkurentnosti često „dolazi" sa opomenama kao što su, *skupo i sklono greškama*, *sinhronizacione osnove operacije Jave* ili *dizajniranje konkurentnog softvera može biti složeno i podložno greškama*. Složenost potiče iz pristupa deljenim promenljivim i raznim strategijama za izbegavanje zastoja i konkurencije između programskih niti. *Sinhronizacione osnovne operacije Jave* su primer takve strategije, a očigledno je da su za mnoge programere komplikovane. Obično se kreira radni okvir, kao što je `java.util.concurrent`, da bi se „ukrotila" složenost konkurentnosti programskih niti, ali neki ljudi tvrde da skrivanje složenosti čini „stvari" samo još složenijim.

„Obični" Java programeri u ovom trenutku mogu prigovoriti. Možda je njihov kod aplikacije napisan za radni okvir, kao što je Spring, ili možda direktno koriste Java EE. U oba slučaja njihov kod aplikacije ne koristi konkurentne funkcije ili programske niti, pa gde je složenost koju smo upravo opisali? Samo zato što je ta složenost skrivena u Springu i Java EE-u ne znači da nema nje i indirektnih troškova.

U redu, shvatili ste: iako višeslojni sistemi mogu raditi neverovatne „stvari", postoji inherentna složenost. Šta obezbeđuje Node.js?

Node.js „odgovara" na složenost

Node.js zahteva da razmišljamo drugačije o konkurentnosti. Povratni pozivi koji su asinhrono otkazani iz petlje događaja mnogo su jednostavniji model konkurentnosti – jednostavniji su za razumevanje, za implementaciju, za razmišljanje i za debagovanje i održavanje.

Node.js ima jednu programsku nit izvršavanja bez čekanja na I/O ili promenu konteksta. Umesto toga, postoji petlja događaja koja šalje događaje funkcijama hendlera dok se nešto događa. Zahtev koji bi blokirao programsku nit izvršavanja izvršava se asinhrono, a rezultati ili greške aktiviraju događaj. Svaka operacija koja bi bila blokirana ili bi na drugi način trebalo da se završi mora da koristi asinhroni model.

Originalna Node.js paradigma je isporučila događaj anonimnoj funkciji. Sada, kada JavaScript ima funkcije `async`, paradigma Node.js se pomera da bi isporučila rezultate i greške pomoću promisa koji upravlja rezervisanom rečju `await`. Kada je pozvana asinhrona funkcija, kontrola se brzo prosleđuje petlji događaja, umesto da izazove blokiranje radnog okruženja Node.js. Petlja događaja nastavlja da obrađuje različite događaje dok snima gde da pošalje svaki rezultat ili grešku.

Korišćenjem asinhronog I/O koji je vođen događajima Node.js uklanja većinu ovih indirektnih troškova, istovremeno uvodeći vrlo malo svojih indirektnih troškova.

Jedna od stavki koju je Ryan Dahl izneo u prezentaciji „Cinco de Node" je hijerarhija vremena izvršenja različitih zahteva. Objektima u memoriji pristupa se brže (u nanosekundama) nego objektima na disku ili objektima koji su preuzeti sa Interneta (u milisekundama ili sekundama). Duže vreme pristupa spoljnim objektima meri se u milionima ciklusa sata, što može biti večnost za kupca koji sedi za veb pregledačem spreman da krene dalje, ukoliko je potrebno duže od dve sekunde za učitavanje stranice.

Stoga, konkurentna obrada zahteva podrazumeva korišćenje strategije za obradu zahteva kojima je potrebno duže vremena da se izvrše. Ako je cilj izbeći složenost višenitnog sistema, sistem mora koristiti asinhrone operacije, kao što to čini Node.js.

Kako izgledaju ovi pozivi asinhronih funkcija?

Asinhroni zahtevi u radnom okruženju Node.js

U radnom okruženju Node.js upit koji smo prethodno videli čita se na sledeći način:

```
query('SELECT * from db.table', function (err, result) {
    if (err) throw err; // handle errors
    // operate on result
});
```

Programer navodi funkciju koja se poziva (otuda i naziv *funkcija povratnog poziva*) kada su rezultat ili greška dostupni. Funkcija `query` i dalje traje isto toliko vremena. Umesto da blokira programsku nit izvršenja, ova funkcija se vraća u petlju događaja, a zatim je slobodna za upravljanje drugim zahtevima. Node.js će na kraju aktivirati događaj koji aktivira pozivanje ove funkcije povratnog poziva, pri čemu se ukazuje na rezultat ili grešku.

Slična paradigma se koristi u JavaScriptu na strani klijenta u kojem stalno pišemo funkcije hendlera događaja.

Napredak jezika JavaScript obezbedio je nove mogućnosti. Kada bi JavaScript bio korišćen zajedno sa promisima ES2015, ekvivalentni kod bi izgledao ovako:

```
query('SELECT * from db.table')
.then(result => {
    // operate on result
})
.catch(err => {
    // handle errors
});
```

Ovo je malo bolji kod, posebno u slučajevima duboko ugnežđene obrade događaja.

Veliki napredak obezbedila je `async` funkcija ES-2017:

```
try {
    const result = await query('SELECT * from db.table');
    // operate on result
} catch (err) {
    // handle errors
}
```

Izuzev rezervisanih reči `async` i `await`, ovaj kod izgleda kao kod koji bismo napisali u drugim programskim jezicima i mnogo je lakši za čitanje. Zbog onoga što funkcija `await` radi, ovo je još uvek asinhrono izvršenje koda.

Sva tri isečka koda izvršavaju isti upit koji smo ranije napisali. Umesto da `query` bude blokirajući poziv, on je asinhroni i ne blokira programsku nit izvršavanja.

U funkcijama povratnog poziva i asinhronom kodiranju promisa Node.js je imao svoj problem koji se odnosio na kompleksnost. Često pozivamo asinhronu funkciju jednu za drugom. U funkcijama povratnog poziva to je značilo duboko ugnežđene funkcije povratnog poziva, a u promisima to je značilo dug lanac funkcija hendlera `.then`. Izuzev složenosti kodiranja, imamo greške i rezultate na neuobičajenim mestima. Umesto da se greške i rezultati prikažu u sledećoj liniji koda, poziva se asinhrono izvršena funkcija povratnog poziva. Redosled izvršavanja nije linija po linija, kao u sinhronim programskim jezicima, već se redosled izvršavanja određuje redosledom izvršavanja funkcije povratnog poziva.

Pristup funkciji `async` rešava tu složenost kodiranja. Stil kodiranja je prirodniji, jer se rezultati i greške nalaze na uobičajenom mestu, tj. u sledećoj liniji koda. Rezervisana reč `await` integriše asinhrono upravljanje rezultatima, bez blokiranja programskih niti izvršavanja. Mnogo štošta je „zakopano“ pod okriljem funkcije `async/await`, a taj model ćemo detaljno razmotriti kasnije u ovoj knjizi.

Da li asinhrona arhitektura platforme Node.js poboljšava performanse?

Performanse i iskorišćenost

Ono što je uzbudljivo u radnom okruženju Node.js je propusna moć (tj. zahtevi u sekundi koje može da opsluži). Uporedna merenja performansi sličnih aplikacija - na primer, Apachea, pokazuju da Node.js ima odlične performanse.

Jedan od repera je sledeći jednostavni HTTP server (pozajmljen sa sajta https://nodejs.org/en/), koji jednostavno vraća poruku `Hello World` direktno iz memorije:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

Ovo je jedan od jednostavnijih veb servera koje možete da kreirate pomoću platforme Node.js. Objekat `http` enkapsulira HTTP protokol, a njegov metod `http.createServer` kreira kompletan veb server, „osluškujući" port koji je naveden u metodu `listen`. Svaki zahtev (`GET` ili `POST` u bilo kojem URL-u) na tom veb serveru poziva navedenu funkciju. Ovaj veb server je jednostavan i lak. U ovom slučaju, bez obzira na URL, vraća se `text/plain`, tj. odgovor `Hello World`.

Ryan Dahl je pokazao jednostavan referentni test u videu *„Ryan Dahl: Introduction to Node.js"* (na kanalu YUI biblioteke na YouTubeu, na adresi https://bit.ly/2HYkSHH). U tom testu se koristi sličan HTTP server kao u našem primeru, ali koji je vratio jednomegabajtni binarni bafer; Node.js je generisao 822 zahteva u sekundi, dok je Nginx generisao 708 zahteva u sekundi, što znači da je Node.js za 15 odsto bolji u odnosu na Nginx. Ryan je takođe napomenuo da je Nginx dostigao četiri megabajta memorije, dok je Node.js dostigao 64 megabajta.

Ključno zapažanje je bilo da je Node.js na kojem je pokrenut interpretirani JIT kompajlirani jezik visokog nivoa bio brz kao Nginx koji je izrađen od visokooptimizovanog C koda dok izvršava slične zadatke. Ta prezentacija je održana u maju 2010. godine, a Node.js je od tada u velikoj meri poboljšan, kao što je navedeno u govoru Chrisa Baileya, koji smo ranije pomenuli.

Fabian Frank, inženjer u kompaniji „Yahoo!", objavio je studiju slučaja performansi vidžeta za predlog pretraga u realnom svetu implementiran pomoću Apachea/PHP-ja i dve varijante Node.js stekova (https://bit.ly/3iyhpMk). Aplikacija je iskačući panel koji prikazuje predloge za pretragu kada korisnik upisuje izraze

pomoću HTTP upita zasnovanog na JSON-u. Verzija Node.js može obraditi osam puta više zahteva u sekundi, uz isto kašnjenje zahteva. Fabian Frank je izjavio da se oba Node.js steka linearno skaliraju sve dok upotreba CPU-a ne dostigne 100 odsto.

„LinkedIn" tim je izvršio veliku reviziju svoje mobilne aplikacije pomoću platforme Node.js za stranu servera da bi zamenio staru aplikaciju Ruby on Rails. Prekidač omogućava da „LinkedIn" pređe sa 30 servera na tri i omogućava spajanje tima za frontend i backend, jer je sve napisano u JavaScriptu. Pre nego što je „LinkedIn" tim odabrao Node.js, procenio je Rails pomoću biblioteke EventMachine, Python pomoću mašine Twisted, a procenjen je i Node.js. „LinkedIn" tim je izabrao Node.js zbog razloga koje smo upravo napomenuli. Šta je „LinkedIn" tim uradio pogledajte na adresi: https://bit.ly/2GD6rrM

Većina postojećih saveta za Node.js performanse obično je napisana za starije V8 verzije koje su koristile optimizator CrankShaft. V8 tim je potpuno izbacio CrankShaft i ima novi optimizator koji se zove TurboFan - na primer, u okviru CrankShafta je bilo sporije koristiti `try/catch`, `let/const`, generator funkcije i tako dalje. Obično se preporučuje da ne koristimo ove funkcije, što je razočavarajuće, jer želimo da koristimo nove JavaScript funkcije, pošto su one u velikoj meri poboljšale JavaScript jezik. Peter Marshall, inženjer iz V8 tima u kompaniji „Google", održao je govor na konferenciji „Node.js Interactive 2017", tvrdeći da bi pomoću TurboFana trebalo da se napiše „prirodni" JavaScript. Cilj je da se pomoću TurboFana unaprede performanse u V8 kompaniji. Da biste videli prezentaciju, pogledajte video u V8 pod nazivom „*High Performance JS*" na adresi https://bit.ly/2GD6y6G.

Činjenica je da JavaScript nije dobar za teške računske radove zbog svojih karakteristika. Neke ideje koje su vezane za ovo ćemo razmotriti u sledećem odeljku. U govoru Mikola Lysenkoa na konferenciji „Node.js Interactive 2016" razmotreni su neki problemi koji se odnose na numeričko računanje u JavaScriptu i neka moguća rešenja. Uobičajeno numeričko računanje uključuje velike numeričke nizove obrađene numeričkim algoritmima koje ste mogli naučiti u kalkulusnim ili linearnim klasama algebri. Ono što nedostaje JavaScriptu su višedimenzionalni nizovi i pristup određenim CPU instrukcijama. Rešenje koje je Lysenko predstavio je biblioteka u kojoj će se implementirati višedimenzionalni nizovi u JavaScriptu, zajedno sa drugom bibliotekom koja je puna numeričkih algoritama za računanje. Da biste videli prezentaciju, pogledajte video „*Numerical Computing in JavaScript*", čiji je autor Mikola Lysenko, na adresi: https://bit.ly/2SuXZ0l

Na konferenciji „Node.js Interactive 2017“ Bailey iz kompanije IBM je objasnio da je Node.js odličan izbor za visokoskalabilne mikroservise. Ključne karakteristike performansi su I/O performanse (merene transakcijama u sekundi), vreme pokretanja (jer ono ograničava koliko vaš servis može brzo da bude skaliran da bi bio zadovoljen zahtev) i memorijski otisak (jer on određuje koliko instanci aplikacije može biti raspoređeno na serveru). Node.js se odlikuje svim ovim merama; u svakom narednom izdanju Node.js se poboljšava, ali ostaje prilično stabilan. Bailey je predstavio brojke, upoređujući Node.js sa sličnim reperima napisanim u Spring Bootu, pokazujući da Node.js funkcioniše bolje. Da biste videli njegov govor, pogledajte video pod nazivom „*Node.js Performance and High Scalable Micro-Services-Chris Bailey, IBM*“ na adresi: https://bit.ly/3nisUeu

Poenta je da se Node.js odlikuje I/O propusnom moći zasnovanoj na događajima. Da li program Node.js može da se istakne u programima izračunavanja zavisi od vaše domišljatosti u rešavanju nekih ograničenja u jeziku JavaScript.

Veliki problem računskog programiranja je što ono sprečava izvršavanje petlje događaja. Kao što ćete videti u sledećem odeljku, to može učiniti da Node.js izgleda kao loš „kandidat“ za bilo šta.

Da li je Node.js „kancerozna“ skalabilna katastrofa?

U postu bloga „Node.js is a cancer“ iz oktobra 2011. godine (od kada je povučen sa bloga na kojem je objavljen) Node.js je nazvan skalabilna katastrofa. Primer prikazan za dokaz bila je CPU-ova implementacija Fibonačijevog algoritma niza. Dok je argument bio pogrešan, budući da Fibonačijev niz niko ne primenjuje na taj način, tačno je da programeri Node.js aplikacije moraju da razmotre gde postavljaju teške računske zadatke.

Ključ za održavanje visoke propusne moći Node.js aplikacija je obezbeđivanje brzog upravljanja događajima. Node.js koristi jednu programsku nit izvršavanja, pa ako je ta programska nit „zasuta“ velikim brojem proračuna, on neće moći da obradi događaje i biće smanjena propusna moć događaja.

Fibonačijev niz, koji služi kao zamena za teške računske zadatke, brzo postaje računski skup za izračunavanje „naivne“ implementacije poput ove:

```
const fibonacci = exports.fibonacci = function(n) {
   if (n === 1 || n === 2) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

Ovo je posebno pojednostavljen pristup izračunavanju Fibonačijevih brojeva. Postoji mnogo načina da se brže izračunaju Fibonačijevi brojevi. Prikazujemo ovo kao opšti primer onoga što se događa sa platformom Node.js kada su hendleri događaja spori i nećemo raspravljati o najboljim načinima izračunavanja matematičkih funkcija. Razmotrite sledeći server:

```
const http = require('http');
const url   = require('url');

http.createServer(function (req, res) {
  const urlP = url.parse(req.url, true);
  let fibo;
  res.writeHead(200, {'Content-Type': 'text/plain'});
  if (urlP.query['n']) {
    fibo = fibonacci(urlP.query['n']);      // Blocking
    res.end('Fibonacci '+ urlP.query['n'] +'='+ fibo);
  } else {
    res.end('USAGE: http://127.0.0.1:8124?n=## where ##
        is the Fibonacci number desired');
  }
}).listen(8124, '127.0.0.1');
console.log('Server running at http://127.0.0.1:8124');
```

Ovo je proširenje jednostavnog veb servera koji je prikazan ranije. U URL-u zahteva traži se argument `n` za koji će biti izračunat Fibonačijev broj. Kada se on izračuna, rezultat se vraća pozivaocu.

Za dovoljno velike vrednosti `n` (na primer, za vrednost `40`) server se uopšte ne odaziva, jer se petlja događaja ne pokreće. Umesto toga, ova funkcija je blokirala obradu događaja, jer petlja događaja ne može da šalje događaje dok se funkcija izračunava.

Drugim rečima, Fibonačijeva funkcija je zamena za svaku operaciju blokiranja.

Da li to znači da je Node.js neispravna platforma? Ne, to samo znači da programer mora voditi računa da kod bude pogodan za dugotrajna računanja i da razvije rešenja. Ovo obuhvata ponovno pisanje algoritma za upotrebu petlje događaja, ponovno pisanje algoritma za efikasnost, integrisanje izvorne biblioteke koda ili „podmetanje“ računski skupih izračunavanja na backend server.

Jednostavno ponovno upisivanje šalje izračunavanja kroz petlju događaja, ostavljajući serveru da dalje obrađuje zahteve u petlji događaja. Pomoću povratnih poziva i zatvaranja (anonimnih funkcija) možemo da održavamo asinhroni I/O i konkurentne promise, kao što je prikazano u sledećem kodu:

```
const fibonacciAsync = function(n, done) {
    if (n === 0) {
        return 0;
    } else if (n === 1 || n === 2) {
        done(1);
    } else if (n === 3) {
        return 2;
    } else {
        process.nextTick(function() {
            fibonacciAsync(n-1, function(val1) {
                process.nextTick(function() {
                    fibonacciAsync(n-2, function(val2) {
                    done(val1+val2); });
                });
        });
    });
  }
}
```

Ovo je podjednako čudan način izračunavanja Fibonačijevih brojeva, ali kada se koristi `process.nextTicka` petlja događaja ima priliku da bude izvršena.

Pošto je ovo asinhrona funkcija koja prihvata funkciju povratnog poziva, potrebno je malo refaktorisanje servera:

```
const http = require('http');
const url   = require('url');

http.createServer(function (req, res) {
  let urlP = url.parse(req.url, true);
  res.writeHead(200, {'Content-Type': 'text/plain'});
  if (urlP.query['n']) {
    fibonacciAsync(urlP.query['n'], fibo => {       // Asynchronous
        res.end('Fibonacci '+ urlP.query['n'] +'='+ fibo);
    });
    } else {
    res.end('USAGE: http://127.0.0.1:8124?n=## where ## is the
     Fibonacci number desired');
  }
}).listen(8124, '127.0.0.1'); console.log('Server running at
http://127.0.0.1:8124');
```

Dodali smo funkciju povratnog poziva da bismo dobili rezultat. U ovom slučaju server može da obrađuje više zahteva za Fibonačijev broj. Međutim, još uvek postoji problem koji se odnosi na performanse zbog neefikasnog algoritma.

Kasnije u ovoj knjizi ćemo malo detaljnije analizirati ovaj primer da bismo istražili alternativne pristupe.

U međuvremenu, možemo razgovarati o tome zašto je važno koristiti efikasne softverske stekove.

Upotreba servera, indirektni troškovi i uticaj na okolinu

Težnja za optimalnom efikasnošću (obrađivanjem više zahteva u sekundi) ne odnosi se samo na programersko zadovoljstvo koje potiče od optimizacije. Postoje stvarne poslovne i ekološke koristi. Obrada više zahteva u sekundi, kao što to mogu učiniti Node.js serveri, znači da neće morati da kupujete mnogo servera. Node.js potencijalno omogućava vašoj organizaciji da učini što više pomoću što manje sredstava.

Grubo rečeno, što više servera kupujete, to su veći novčani troškovi i troškovi zaštite životne sredine. Postoji čitavo polje ekspertize kada je reč o smanjenju troškova i uticaja na veb servere na okolinu za koje ta gruba smernica nije dovoljna. Cilj je prilično očigledan - manje servera, niži troškovi i manji uticaj na okolinu korišćenjem efikasnijeg softvera.

U „Intelovom“ radu „*Increasing Data Center Efficiency with Server Power Measurements*“ (https://intel.ly/36xpY83) dat je objektivni okvir za razumevanje efikasnosti i troškova centra za podatke. Mnogo je faktora, poput zgrada, sistema za hlađenje i dizajna računarskih sistema. Efikasan dizajn zgrade, efikasni sistemi za hlađenje i efikasni računarski sistemi (efikasnost centra za podatke, gustina podataka i gustina skladištenja) mogu sniziti troškove i smanjiti uticaj na okolinu. Međutim, ove dobitke možete uništiti primenom neefikasnog paketa softvera, jer ćete morati da kupujete više servera nego kada biste imali efikasan paket softvera. Osim toga, efekte od efikasnosti centra za podatke možete da poboljšate efikasnim paketom softvera koji omogućava da smanjite broj potrebnih servera.

Ovo razmatranje efikasnih paketa softvera nije samo za altruističke svrhe zaštite životne sredine. Ovo je jedan od slučajeva u kojima zaštita životne sredine može pomoći vašem preduzeću.

U ovom odeljku ste naučili mnogo o tome kako se Node.js arhitektura razlikuje od ostalih programskih platformi. Izbor isključivanja programskih niti radi implementacije konkurentnosti pojednostavljuje složenost i opsežne indirektne troškove koji nastaju upotrebom programskih niti. Čini se da je ovim ispunjeno obećanje koje se odnosi na efikasnost. Efikasnost ima brojne prednosti u mnogim aspektima poslovanja.

Prihvatanje napretka u jeziku JavaScript

Poslednjih nekoliko godina bilo je uzbudljivo vreme za JavaScript programere. Odbor TC-39 koji nadgleda ECMAScript standard, dodao je mnogo novih funkcija, od kojih su neke sintaktički „šećer“, ali neke od njih su nas odvele u potpuno novu eru JavaScript programiranja. Sama po sebi, funkcija `async/await` omogućava izlaz iz onoga što se zove „pad“ povratnog poziva, tj. iz situacije u kojoj se nalazimo kada ugnežđavamo povratne pozive unutar povratnih poziva. To je toliko značajna funkcija da bi trebalo ponovno osmisliti preovlađujuću paradigmu orijentisanu na povratni poziv u radnom okruženju Node.js i u ostatku JavaScript ekosistema.

Ranije ste u ovom poglavlju videli ovo:

```
query('SELECT * from db.table', function (err, result) {
    if (err) throw err; // handle errors
    // operate on result
});
```

Ovo je bio važan uvid Ryana Dahla i to je ono što je podstaklo popularnost platforme Node.js. Određene akcije traju dugo vremena, poput upita baze podataka, i ne treba ih tretirati isto kao i operacije koje brzo preuzimaju podatke iz memorije. Zbog „prirode“ jezika JavaScript, Node.js je morao da izrazi ovu asinhronu konstrukciju kodiranja na neuobičajeni način. Rezultati se ne prikazuju u sledećoj liniji koda, već se pojavljuju unutar funkcije povratnog poziva. Štaviše, greške se moraju rešavati na neuobičajeni način, tj. unutar funkcije povratnog poziva.

Konvencija u radnom okruženju Node.js je da je prvi parametar funkcije povratnog poziva indikator greške, a naredni parametri su rezultati. Ovo je korisna konvencija koju ćete naći u na čitavoj platformi Node.js; međutim, ona komplikuje upravljanje rezultatima i greškama, jer se prikazuju na neuobičajenoj lokaciji, tj. u funkciji povratnog poziva. „Prirodno“ mesto na kojem greške i rezultati treba da budu prikazani su naredne linije koda.

Spuštamo se dalje u „pakao“ povratnih poziva u svakom sloju ugnežđavanja funkcije povratnog poziva. Sedmi sloj umetanja povratnog poziva složeniji je od šestog sloja umetanja povratnog poziva. Zašto? Ako ništa drugo, posebni aspekti za upravljanje greškama postaju sve složeniji, jer su povratni pozivi dublje ugnežđeni.

Međutim, kao što ste videli ranije, ovo je novi preferirani način pisanja asinhronog koda u radnom okruženju Node.js:

```
const results = await query('SELECT * from db.table');
```

Alternativno, funkcije ES2017 async vraćaju vas ovom prirodnom izrazu programiranja. Rezultati i greške nalaze se na odgovarajućoj lokaciji, uz očuvanje odličnog asinhronog programskog modela koji je vođen događajima, a koji je Node.js učinio odličnim. Videćete kasnije u ovoj knjizi kako to funkcioniše.

TC-39 odbor dodao je JavaScriptu još mnogo novih funkcija, kao što su sledeće:

- poboljšana sintaksa za deklaracije klase, koja nasleđivanje objekata i funkcije `getter/setter` čini vrlo uobičajenim
- novi format modula koji je standardizovan za pregledače i Node.js
- novi metodi za znakovne nizove - na primer, notacija znakovnog niza šablona
- novi metodi za kolekcije i nizove - na primer, operacije `map/reduce/filter`
- rezervisana reč `const` za definisanje promenljivih koje se ne mogu promeniti i rezervisana reč `let` koja definiše promenljive čiji je opseg ograničen na blok u kome su deklarisane, a ne na prednju funkciju
- nove konstrukcije petlje i iteracijski protokol koji funkcioniše u novim petljama
- nova vrsta funkcije, tj. streličasta funkcija, koja je lagana po težini, što podrazumeva manje uticaja na memoriju i vreme izvršavanja
- Objekat `Promis` predstavlja rezultat koji „obećava“ da će biti isporučen u budućnosti. Promisi mogu, sami po sebi, smanjiti problem koji se odnosi na povratne pozive i čine deo osnova za funkcije `async`.
- Funkcije generatora su intrigantan način reprezentacije asinhrone iteracije u skupu vrednosti. Što je još važnije, one čine drugu polovinu osnova asinhronih funkcija.

Možda ćete videti novi JavaScript koji je opisan kao ES6 ili ES2017. Koji je preferirani naziv za opisivanje verzije JavaScripta koji se koristi?

Verzije ES1 do ES5 obeležile su različite faze razvoja JavaScripta. ES5, objavljen 2009. godine, u većoj meri se primenjuje u modernim pregledačima. Počev od ES6, odbor TC-39 odlučio je da promeni konvenciju o imenovanju zbog namere da se svake godine dodaju nove jezičke funkcije. Stoga, naziv verzije jezika sada uključuje godinu - na primer, ES2015 je objavljen 2015, ES2016 2016, a ES2017 2017. godine.

Implementacija JavaScript koda za ES2015/2016/2017/2018

Očigledno je da programeri JavaScripta često ne mogu da koriste najnovije funkcije. Programeri za frontend JavaScript ograničeni su implementiranim veb pregledačima i velikim brojem starih pregledača koji se koriste na uređajima čiji se OS godinama ne ažurira. Internet Explorer verzija 6 je, na svu sreću, skoro potpuno povučena, ali još uvek je mnogo starih pregledača instaliranih na starijim računarima koji i dalje obavljaju valjanu „ulogu" za svoje vlasnike. Stari pregledači imaju stare implementacije JavaScripta i ako želimo da se kod izvršava, potrebno je da on bude kompatibilan sa starim pregledačima.

Da bi kod bio kompatibilan sa starim pregledačima, možemo da koristimo Babel i druge alatke za prepisivanje koda. Mnogo proizvoda mora da bude upotrebljivo za ljude koji koriste stare pregledače. Programeri i dalje mogu da napišu svoj kod pomoću najnovijih JavaScript ili TypeScript funkcija, a zatim da pomoću Babela napišu svoj kod tako da može da bude pokrenut u starom pregledaču. Prema tome, frontend JavaScript programeri mogu usvojiti (neke) nove funkcije po ceni složenijih alatki za razvoj softvera i rizika da se pojave greške tokom procesa prepisivanja koda.

Node.js „svet" nema ovaj problem. Node.js je brzo usvojio funkcije ES2015/2016/2017, podjednako brzo kao što su implementirane u V8 mašinu. Počev od Node.js 8, mogla se slobodno koristiti funkcija `async` kao izvorna funkcija. Format novog modula prvi put je podržan u Node.js verziji 10.

Drugim rečima, frontend JavaScript programeri mogu tvrditi da moraju da sačekaju nekoliko godina pre usvajanja funkcija ES2015/2016/2017, dok Node.js programeri ne moraju da čekaju. Nove funkcije možemo jednostavno koristiti bez upotrebe alatki za prepisivanje koda, ali upotreba Babela se preporučuje ako vaši menadžeri insistiraju na podršci starijim Node.js izdanjima koja su prethodila usvajanju ovih funkcija.

Neka unapređenja u JavaScript „svetu" dešavaju se izvan TC-39 zajednice.

TypeScript i Node.js

TypeScript jezik je zanimljiv izdanak JavaScript okruženja. Pošto se JavaScript sve više može koristiti za složene aplikacije, za kompajler je sve korisnije da pomogne u „hvatanju“ grešaka u programiranju. Preduzetnički programeri koji koriste druge programske jezike, kao što je Java, navikli su na snažnu proveru tipa kao načina za sprečavanje određenih klasa grešaka.

Snažna provera tipa pomalo je anatema JavaScript programerima, ali je izuzetno korisna. TypeScript projekat ima za cilj da „donese“ dovoljno rigoroznosti iz jezika, kao što su Java i C#, a da, pri tom, ostavi dovoljno slobode zbog koje je JavaScript veoma popularan. Rezultat je provera tipa izvršenja u kompilaciji, bez teškog tereta koji su „nosili“ programeri u nekim drugim jezicima.

Iako nećemo koristiti TypeScript u ovoj knjizi, njegove alatke je vrlo lako usvojiti u Node.js aplikacijama.

U ovom odeljku ste saznali da platforma Node.js „ide u korak“ sa promenama JavaScript jezika.

Razvoj mikroservisa ili maksiservisa pomoću platforme Node.js

Nove mogućnosti, kao što su sistemi za implementaciju u „oblaku“ i Docker, obezbeđuju implementaciju nove vrste arhitekture servisa. Alatka Docker omogućava definisanje konfiguracije procesa na serveru u ponovljivom kontejneru koji milioni ljudi mogu lako da implementiraju u sistem hostovan u „oblaku“. Ona obezbeđuje male, jednonamenske serverske instance koje se mogu povezati da bi činile kompletan sistem. Docker nije jedina alatka koja pomaže u pojednostavljivanju implementacije „oblaka“; međutim, funkcije ove alatke su dobro prilagođene savremenim potrebama implementacije aplikacija.

Neki ljudi su popularizovali koncept mikroservisa kao način da se opiše ova vrsta sistema. Prema tvrdnjama na veb sajtu `microservices.io`, mikroservis se sastoji od skupa usko fokusiranih, nezavisno raspoloživih servisa. Koncept mikroservisa se na tom veb sajtu upoređuje sa monolitnim obrascem implementacije aplikacija u kojem je svaki aspekt sistema integrisan u jedan paket (kao što je jedna WAR datoteka za aplikaciju Java EE servera). Model mikroservisa obezbeđuje programerima neophodnu fleksibilnost.

Neke prednosti mikroservisa su sledeće:

- Svakim mikroservisom može upravljati mali tim.
- Svaki tim može da radi po svom rasporedu, pod uslovom da se zadrži kompatibilnost API servisa.
- Mikroservisi se mogu koristiti nezavisno ako je to potrebno - na primer, za jednostavnije testiranje.
- Lakša je promena tehnoloških stekova.

Gde se Node.js uklapa u ovo? Njegov dizajn se odlično uklapa u model mikroservisa:

- Node.js ohrabruje male, usko fokusirane jednonamenske module.
- Ovi moduli su sastavljeni u aplikaciji pomoću odličnog sistema za upravljanje paketima npm.
- Moduli za objavljivanje su neverovatno jednostavni, bilo da je reč o NPM spremištu ili o Git URL-u.
- Iako se aplikacijski radni okvir, kao što je Express, može koristiti u velikim servisima, veoma dobro funkcioniše za male lagane servise i podržava jednostavnu implementaciju.

Ukratko, Node.js možete lako koristiti na jednostavan i agilan način, izrađujući velike ili male servise, u zavisnosti od vaših arhitektonskih preferencija.

Rezime

U ovom poglavlju ste mnogo naučili. Konkretno, videli ste da JavaScript ima „život“ izvan veb pregledača i da je Node.js odlična programska platforma sa mnogim zanimljivim atributima. Iako je relativno mlad projekat, Node.js je postao veoma popularan i u velikoj meri se koristi ne samo za veb aplikacije, već i za programerske alatke komandne linije i za još mnogo štošta. Pošto je Node.js platforma zasnovana na Chromeovoj V8 JavaScript mašini, projekat može da prati brza unapređenja JavaScript jezika.

Node.js arhitektura se sastoji od asinhronih funkcija kojima upravlja petlja događaja koja aktivira funkcije povratnog poziva, umesto da koristi programske niti i blokirajući I/O. Ova arhitektura ima koristi od dobrih performansi koje obezbeđuju mnoge prednosti, uključujući mogućnost da se izvršava više posla pomoću manje hardvera. Takođe ste naučili da neefikasni algoritmi mogu izbrisati bilo koje prednosti performansi.

U ovoj knjizi se fokusiramo na razmatranje razvoja i implementacije Node.js aplikacija u realnom svetu. Obuhvatićemo što više aspekata razvoja, poboljšanja, testiranja i implementacije Node.js aplikacija.

Sada, kada ste pročitali ovaj uvod u radno okruženje Node.js, spremni ste da „zaronite“ u njega i da počnete da ga koristite. U Poglavlju 2, „*Podešavanje radnog okruženja Node.js*“, detaljno ćemo istražiti kako se podešava razvojno okruženje Node.js za Mac, Linux ili Windows, pa ćemo čak i napisati neki kod.