1
Dobrodošli na platformu Node.js

Postoje principi i projektni modeli koji bukvalno određuju iskustvo koje će projektanti aplikacija imati s platformom Node.js i pridruženim ekosistemom; među njima se verovatno najviše ističu asinhrona priroda platforme i njen stil programiranja koji, u svom najjednostavnijem obliku, obilno koristi povratne funkcije (engl. *callbacks*). Važno je da savladamo prvo te osnovne principe i modele, ne samo zato da bismo pisali ispravan kôd, nego i zato da bismo donosili efikasne odluke u vezi s projektovanjem aplikacija kada treba da rešavamo veće i složenije probleme.

Još jedan aspekt koji je karakterističan za Node.js jeste njegova filozofija. Izbor platforme Node.js podrazumeva više od samo učenja nove tehnologije; to takođe znači pihvatanje nove kulture i pristupanje novoj zajednici programera. Videćemo kako to značajno utiče na projektovanje naših aplikacija i komponenata i kako se one uklapaju među druge aplikacije čiji su autori članovi zajednice.

Osim navedenih aspekata, vredi znati da su najnovije verzije Node.js uvele podršku za mnoge mogućnosti koje opisuje specifikacija ES2015 (ranije ES6), što jezik čini još izrazitijim i prijatnijim za upotrebu. Važno je da upoznate te nove sintaksne i funkcionalne dopune jezika da biste pisali sažetiji i razumljiviji kôd i smislili alternativne načine implementiranja projektnih modela na koje ćete nailaziti u ovoj knjizi.

U ovom poglavlju, obradićemno sledeće teme:

- Node.js filozofija, odnosno „Node način“
- Node.js verzije 6 i ES2015
- Reaktorski model—mehanizam koji čini srce asinhrone arhitekture platforme Node.js

Node.js filozofija

Svaka platforma ima vlastitu filozofiju, a to je skup principa i pravila opšte prihvaćenih u zajednici programera, zatim „ideologija“ koja određuje kako treba raditi stvari i koja utiče na evoluciju platforme i kako se aplikacije razvijaju i projektuju. Neki među tim principima potiču iz same tehnologije, druge omogućuje njen ekosistem, neki su samo trendovi u zajednici programera, dok su drugi evolucije drugih ideologija. U platformi Node.js, neke od tih principa je postavio direktno njen tvorac, Ryan Dahl; druge su postavili ljudi koji su napravili jezgro platforme; neki potiču od harizmatičnih članova zajednice programera, a deo principa je i nasleđen iz JavaScript kulture ili je na njih uticala Unix filozofija.

Nijedno od tih pravila nije obavezno i treba ih uvek primenjivati uz zdrav razum; međutim, mogu biti izuzetno korisna ako nam zatreba izvor inspiracije dok projektujemo naše programe.

Na Wikipediji možete naći opsežnu listu filozofija čija tema je razvijanje softvera, na `http://en.wikipedia.org/wiki/List_of_software_development_philosophies`.

Malo jezgro

Samo jezgro platforme Node.js zasniva se na nekoliko principa; jedan od njih je da jezgro treba da sadrži najmanji mogući skup funkcionalnosti, a preostali deo funkcionalnosti se smešta u takozvani **korisnički prostor** (engl. *userland* ili *userspace*), što je ekosistem koji čine moduli koji ne pripadaju jezgru. Taj princip ima izuzetan uticaj na Node.js kulturu programiranja, budući da zajednici programera pruža slobodu da eksperimentišu i brzo istražuju širi skup rešenja unutar modula korisničkog prostora, umesto da im ruke budu vezane jednim sporo evoluirajućim rešenjem koje je ugrađeno u strožije kontrolisano i stabilno jezgro. Držanje funkcionalnosti jezgra na najosonovnijem minimumu, ne samo što pruža zgodan način za održavanje aplikacija, nego je pozitivno i sa aspekta kulturološkog uticaja na evoluciju celog ekosistema.

Mali moduli

Node.js koristi koncept *modula* kao osnovni način za strukturiranje koda jednog programa. To je gradivni blok za izradu aplikacija i višekratno upotrebljivih biblioteka koje se zovu *paketi* (reč paket se često koristi i za same module, budući da paket sadrži, najčešće, samo jedan modul koji služi kao ulazna tačka u paket). Na platformi Node.js, jedan od najpreporučenijih principa jeste projektovanje kratkih i sažetih modula, ne samo po veličini koda u njima, nego što je još važnije, i po opsegu vidljivosti tog koda.

Taj princip vuče korene iz Unixove filozifije, naročito iz sledeće dve ideje:

- „Malo je lepo.“
- „Svaki program treba da radi samo jednu jedinu stvar, ali da je radi dobro.“

Node.js je ove koncepte doveo na sasvim nov nivo. Pomoću zvaničnih alata za upravljanje paketima, npm, Node.js rešava problem *pakla međusobnih zavisnosti* tako što obezbeđuje da svaki instaliran paket ima zaseban vlastiti skup elemenata od kojih zavisi, što programu omogućava da bez skuoba zavisi od više paketa istovremeno. U stvari, Node način pruža izuzetan nivo višekratne upotrebljivosti koda, budući da se aplikacije sastoje od većeg broja malih i ciljanih zavisnosti od spoljnog koda. Mada se takvo nešto na drugim platformama može smatrati nepraktičnim, pa čak i potpuno neizvodljivim, na platformi Node.js se ta praksa preporučuje. Posledica toga je da nisu retki npm paketi koji sadrže manje od 100 redova koda, ili koji čak izlažu samo jednu jedinu funkciju.

Osim očiglednih prednosti po pitanju višekratne upotrebe istog koda, smatra se da mali modul pruža još i sledeće:

- Lako je razumljiv i upotrebljiv
- Lakše se testira i održava
- Savršen za deljenje s čitačem veba

Kraći i specijalizovaniji moduli omogućavaju svakome da deli ili višekratno koristi čak i najmanji delić koda; to je princip **Nemoj se ponavljati** (**NSP**) doveden na savim nov nivo.

Mali otisak

Osim što su sažeti po veličini i vidljivosti koda, karakteristično za Node.js module je i to da spoljnom svetu izlažu minimalni skup funkcionalnosti. Glavna prednost toga jeste da se tako proširuju mogućnosti za višekratnu upotrebu API-ja, što znači da API postaje lakši za upotrebu i manje podložan pogrešnoj upotrebi. U stvari, u najvećem broju slučajeva, korisnika određene komponente zanima samo vrlo ograničen i konkretan skup mogućnosti i nema potrebe da proširuje funkcionalnost komponente, niti da koristi neke njene naprednije aspekte.

Na platformi Node.js, veoma uobičajen model je da se moduli projektuju tako da spoljnom svetu izlažu samo jednu određenu funkcionalnost, kao što je funkcija ili konstruktor, a napredniji aspekti ili sekundarne mogućnosti postaju svojstva funkcije koju modul ili konstruktor izvozi. Korisnicima to omogućava da prepoznaju šta je važno a šta sekundarno. Neretko moduli izlažu samo jednu funkciju i ništa drugo samo zato što tako stavljaju na raspolaganje jedinstvenu i jasno prepoznatljivu ulaznu tačku s kojom korisnik ne može pogrešiti.

Još jedna karakteristika mnogih Node.js modula jeste činjenica da ih korisnik piše od početka, umesto da proširi neki postojeći. Ako onemogućimo pristup internim detaljima modula tako što zabranimo svaku mogućnost proširivanja modula, to će možda zvučati nefleksibilno, ali to zapravo pruža prednost smanjivanja broja mogućih slučajeva upotrebe modula, što pojednostavljuje njegovu implementaciju, olakšava održavanje i povećava upotrebljivost.

Jednostavnost i pragmatizam

Da li ste čuli za princip **KISS** (**Keep it simple and stupid**), neka bude jednostavno i glupo ili čuveni citat:

„Jednostavnost je najveća složenost."
– Leonardo da Vinci

Richard P. Gabriel, poznati naučnik u oblasti računarskih nauka, smislio je izraz „gore je bolje" (engl. *worse is better*) da bi opisao model u kojem se jednostavnija i ograničenija funkcionalnost smatra dobrim oblikom dizajna softvera. U svom tekstu, *The Rise of "Worse is Better"*, on kaže:

> *„Dizajn mora biti jednostavan, i u delu implementacije i u delu interfejsa.Važnije je da implementacija bude jednostavna nego interfejs. Jednostavnost je najvažniji element koji utiče na dizajn."*

Projektovanje jednostavnog, nasuprot savršenom softveru s mnogobrojnim osobinama, dobra je praksa iz više razloga: njegovo implementiranje zahteva manje truda, omogućava bržu isporuku naručiocu uz manji utrošak resursa, lakše se prilagođava i lakši je za održavanje i razumevanje. Te osobine podstiču doprinošenje zajednici programera a samom softveru omogućavaju da raste i da se poboljšava.

Na platformi Node.js, primenu tog principa omogućava i sam JavaScript, koji je vrlo pragmatičan jezik. U stvari, neretko viđamo da jednostavne funkcije, ograde i objektni literali zamenjuju složene hijerarhije klasa. U čistom objektno orijentisanom dizajnu autori često pokušavaju da oponašaju stvarni svet pomoću matematičkih izraza na računarskom sistemu, ali pri tome zanemaruju nesavršenost i složenost samog stvarnog sveta. Činjenica je da naš softver jeste uvek samo približna slika stvarnosti, a mi bismo verovatno imali više uspeha ako pokušamo da nešto osposobimo da radi što pre i uz razuman nivo složenosti, umesto da pokušavamo da stvorimo gotovo savršen softver ulažući ogroman napor i „masu" koda za održavanje.

Primenu tog principa viđaćemo na mnogim mestima u ovoj knjizi. Na primer, priličan broj projektnih modela, kao što su *singleton* ili *decorator*, mogu imati trivijalnu, a ponekad čak i ne sasvim pouzdanu, implementaciju i videćemo kako jedan praktičan pristup, bez komplikovanja, (u najvećem broju slučajeva) može biti bolji od čistog i besprekornog dizajna.

Uvod u Node.js, verzija 6 ili ES2015

U vreme pisanja ove knjige, najnovija značajnija izdanja pltaforme Node.js (verzije 4, 5 i 6) isporučuju se dopunjena sve širom jezičkom podrškom za nove mogućnosti definisane specifikacijom ECMAScript 2015 (skraćeno ES2015, a ranije poznata i kao ES6), čiji cilj je da jezik JavaScript načini još fleksibilnijim i pogodnijim za upotrebu.

U celoj ovoj knjizi, u našim primerima koda obilno ćemo koristiti neke od tih novih mogućnosti. Pošto su ti koncepti i dalje novina unutar Node.js zajednice programera, korisno je da ukratko razmotrimo najvažnije mogućnosti sa specifikacije ES2015 koje Node.js podržava u ovom trenutku. Naša referentna verzija je Node.js, verzija 6.