Predavanje

Nedelja 1, cas 2

Kreiranje lokalnih paketa/modula i njihovo pozivanje

U današnjem digitalnom dobu, organizacija koda je ključna za efikasan razvoj softvera. Kreiranje modularnog koda omogućava lakše održavanje, ponovno korišćenje i skaliranje aplikacija. U ovoj prezentaciji istražićemo proces kreiranja lokalnih paketa/modula u JavaScript-u i kako ih pozivati u našim projektima.

Kreiranje lokalnih paketa/modula

Lokalni paketi/moduli predstavljaju organizacione jedinice koda u JavaScript-u. Oni omogućavaju razdvajanje funkcionalnosti aplikacije na manje, samostalne delove. Kreiranje lokalnih paketa/modula olakšava upravljanje složenim projektima i omogućava ponovno korišćenje koda. Proces kreiranja lokalnih paketa/modula uključuje:

Dodavanje funkcionalnosti/modula u direktorijum:

Razvijamo funkcionalnost koju želimo da uključimo u naš paket/modul.

Definišemo funkcije, klase, ili druge komponente koje želimo izvoziti.

Korišćenje module.exports za izvoz funkcionalnosti/modula:

Koristimo module.exports ili exports za izvoz funkcionalnosti/modula iz našeg paketa/modula.

Pozivanje lokalnih paketa/modula

Nakon što smo kreirali naš lokalni paket/modul, sada je vreme da ga pozovemo i iskoristimo u našim projektima. To možemo uraditi koristeći require funkciju:

```
// Uvozimo lokalni paket/modul
const myModule = require('./myModule');

// Koristimo funkcionalnosti/modul iz našeg paketa/modula
myModule.myFunction();
```

Instalacija i deinstalacija paketa/modula

U današnjem razvoju softvera, korišćenje spoljnih paketa/modula je neizbežno. Paketi/moduli nam pružaju gotove funkcionalnosti i olakšavaju razvoj aplikacija.

Paketi/moduli predstavljaju biblioteke ili komponente koje se koriste za dodavanje funkcionalnosti u JavaScript projekte.
Instalacija paketa/modula obično se obavlja pomoću npm-a, koji je ugrađeni alat za upravljanje paketima/modulima u Node.js ekosistemu.

Instalacija paketa/modula

Koraci za instalaciju paketa/modula uključuju:

Korišćenje npm install komande:

Otvorite terminal i navigirajte do radnog direktorijuma projekta.
Pokrenite npm install ime_paketa komandu za instalaciju paketa/modula.
Instalacija globalnih paketa/modula:

- Dodajte -g opciju uz npm install komandu za instalaciju paketa/modula globalno na sistemu.
- Specifikacija verzije paketa/modula:
- Možete specificirati tačnu verziju paketa/modula koji želite instalirati dodavanjem odgovarajućih prefiksa (+, ^) uz naziv paketa/modula.

Deinstalacija paketa/modula

Nakon što završimo sa korišćenjem određenog paketa/modula, možemo ga deinstalirati kako bismo smanjili veličinu projekta i održali ga čistim. Koraci za deinstalaciju paketa/modula uključuju:

- Korišćenje npm uninstall komande:
- Otvorite terminal i navigirajte do radnog direktorijuma projekta.
- Pokrenite npm uninstall ime_paketa komandu za deinstalaciju paketa/modula.
- Deinstalacija globalnih paketa/modula:
  - Dodajte -g opciju uz npm uninstall komandu za deinstalaciju globalnih paketa/modula.
  - Opcioni parametri za deinstalaciju:
  - Možete koristiti dodatne parametre kao što su --save, --save-dev za ažuriranje package.json fajla nakon deinstalacije paketa/modula.

Korišćenje ugrađenih paketa/modula

Node.js je moćan okvir za izgradnju serverskih aplikacija, a jedna od njegovih ključnih karakteristika je bogata biblioteka ugrađenih modula. U ovoj prezentaciji istražićemo kako koristiti ugrađene pakete/module u Node.js-u, sa posebnim fokusom na fs modul za rad sa filesystem-om.

Korišćenje ugrađenih paketa/modula

Jedna od najčešćih operacija u serverskim aplikacijama je manipulacija fajlovima i direktorijumima. Za to koristimo fs modul, koji je ugrađen u Node.js. Ključne funkcije fs modula uključuju:

- Čitanje fajlova (fs.readFile)
- Pisanje u fajlove (fs.writeFile)
- Kreiranje direktorijuma (fs.mkdir)
- Brisanje fajlova ili direktorijuma (fs.unlink, fs.rmdir)
- Preimenovanje fajlova (fs.rename)

Ovaj modul omogućava nam da radimo sa filesystem-om na način koji je prilagođen Node.js-u, što je posebno korisno u serverskim aplikacijama.

Konverzija callback funkcija u promise

U današnjem svetu asinhronog programiranja, callback funkcije su često korišćene za upravljanje asinhronim operacijama. Međutim, sa pojavom promise-a, mnogi programeri prelaze na korišćenje promise-a zbog njegove jasnoće i lakše upotrebe. Istražićemo proces konverzije funkcija koje koriste callback u promise funkcije i zašto je to korisno.

Konverzija callback funkcija u promise

Jedan od glavnih razloga za konverziju callback funkcija u promise funkcije je olakšanje upravljanja asinhronim operacijama i izbegavanje "callback hell" problema. Evo kako možemo konvertovati klasične callback funkcije u promise funkcije:

Konverzija callback funkcija u promise

- Ručno kreiranje promise-a:

Koristimo new Promise konstruktor da bismo ručno kreirali promise. Definišemo izvršni kod unutar promise-a i pozivamo resolve/reject funkcije na osnovu rezultata operacije.

- Konvertovanje postojećih funkcija:

Postojeće funkcije koje koriste callback možemo konvertovati u promise funkcije tako što ćemo obaviti njihovu logiku unutar promise-a.

- Korišćenje util.promisify:

Node.js nudi ugrađeni modul util sa funkcijom promisify koja automatski konvertuje funkcije koje koriste callback u promise funkcije.

Konverzija callback funkcija u promise

Kroz praktične primere, možemo jasno videti prednosti korišćenja promise funkcija u odnosu na callback funkcije.

Evo nekoliko primera:

- Čitanje fajlova: Korišćenje fs.readFile sa callback-om i promise-om za čitanje fajlova sa filesystem-a.
- Izvršavanje HTTP zahteva: Upotreba http.get sa callback-om i promise-om za izvršavanje HTTP zahteva ka spoljnim resursima.
- Korišćenje asinhronih funkcija: Korišćenje asinhronih funkcija u okviru modernih JavaScript okruženja, gde je korišćenje promise-a preporučeno.

Pozivanje promise funkcija sa then/catch i async/await u try/catch blok

U svetu asinhronog programiranja, promise-ovi su postali standardni način rukovanja asinhronim operacijama. Dodatak then/catch lanca i async/await sintakse olakšavaju upravljanje asinhronim kodom i hvatanje grešaka.

Pozivanje promise funkcija sa then/catch i async/await u try/catch blok

Korišćenje then/catch lanca za obradu promise-ova:

then/catch lanac je način za obradu rezultata i grešaka koji se javljaju pri izvršavanju promise funkcija. Evo kako ga koristimo:

Korišćenje then metode:

then metoda se koristi za rukovanje uspešnim izvršavanjem promise-a.

Unutar then metode možemo izvršiti željene akcije nad rezultatom promise-a.

Pozivanje promise funkcija sa then/catch i async/await u try/catch blok

Korišćenje catch metode:

catch metoda se koristi za hvatanje grešaka koje se jave pri izvršavanju promise-a.

Unutar catch metode možemo definisati način kako ćemo rukovati greškom i preduzeti odgovarajuće akcije.

Pozivanje promise funkcija sa then/catch i async/await u try/catch blok

async/await sintaksa omogućava nam da pišemo asinhroni kod na način koji izgleda kao klasičan, sinkroni kod. Evo kako to funkcioniše:

Korišćenje async funkcije:

async ključna reč se koristi za definisanje asinhronih funkcija. Kada koristimo async funkciju, ona implicitno vraća promise.

Korišćenje await operatora:

await operator se koristi unutar async funkcije za čekanje na rezultat asinhronih operacija.

Kada koristimo await, izvršavanje koda se pauzira dok se ne završi asinhrona operacija.

Korišćenje try/catch bloka:

try/catch blok se koristi za hvatanje grešaka koje se javljaju pri izvršavanju asinhronih operacija.

Unutar try bloka stavljamo kod koji želimo izvršiti, dok se greške hvataju u catch bloku.

Pozivanje promise funkcija sa then/catch i async/await u try/catch blok

Korišćenje await operatora:

await operator se koristi unutar async funkcije za čekanje na rezultat asinhronih operacija.
Kada koristimo await, izvršavanje koda se pauzira dok se ne završi asinhrona operacija.

Korišćenje try/catch bloka:

try/catch blok se koristi za hvatanje grešaka koje se javljaju pri izvršavanju asinhronih operacija.
Unutar try bloka stavljamo kod koji želimo izvršiti, dok se greške hvataju u catch bloku.