JUNIORDev DIGITALNA DALMACIJA

Složena stanja

React komponente u pravilu imaju više od jedne varijable stanja. Zbog toga je važno detaljnije se upoznati sa primjerima kada komponente imaju složenije unutarnje stanje.

Renderiranje

Vratimo se kratko na proces renderiranja komponente. Modificirati ćemo prethodni primjer na način da umjesto jednog brojača imamo dvije varijable stanja, svaku sa svojim brojačem:

App.jsx

```
function App() {
  const [broj, postaviBroj] = useState(5);
  const [drugi, postaviDrugi] = useState(99)

  function uvecajPrviBrojac() {
    postaviBroj(broj + 1);
  }

  return (
    <div>
      <p>Vrijednost prvog broja: {broj}</p>
      <button onClick={uvecajPrviBrojac}>Uvećaj prvi </button>
      <p>Vrijednost drugog broja: {drugi}</p>
      <button onClick={() => postaviDrugi(drugi + 1)}>Uvećaj drugi </button>
    </div>
  );
}
```

Možete uočiti da smo umjesto pisanja pomoćne funkcije jednostavno iskoristili ES6 mogućnost pisanja *arrow* funkcija i poslali anonimnu funkciju kao argument *onClick* svojstvu. U ovom slučaju imamo dva brojača i osvježavanjem bilo kojeg od njih osvježiti će cijela komponenta ali

će drugi brojač zadržati trenutnu vrijednost (neće se vratiti na početnu vrijednost) - to je jedna od ključnih karakteristika *useState*-a i korištenja internog stanja.

Dok je god komponenta aktivna i prikazana na ekranu, vrijednost varijable stanja će ostati sačuvana između višestrukih ciklusa renderiranja (osim naravno ako renderiranje nije uzrokovano promjenom baš te varijable stanja.)

Sinkronizacija osvježavanja

Još jedno bitno svojstvo Reacta je da pozivom funkcije za promjenu varijable stanja ne dolazi nužno do trenutne promjene stanja (i ponovnog renderiranja). Umjesto toga, možemo zamisliti da jednostavno zakazujemo promjenu za neki idući diskretni trenutak u kojem React odluči da je potrebno osvježiti stanje aplikacije.

Prednost ovog pristupa je što se na taj način osvježavanje stanja može optimizirati. To posebno dolazi do izražaja kada u kratkom razdoblju mijenjamo više varijabli stanja. Umjesto višestrukih pojedinačnih izmjena, koje bi svaka za sebe uzrokovala novo renderiranje, React jednostavno odrađuje sve promjene zajedno i na taj način je potreban samo jedan ciklus ponovnog renderiranja.

Ovo je najbolje samostalno probati na primjeru. Umjesto da prethodno definirana dva brojača povećavamo pojedinačno, napisati ćemo funkciju `uvecajOba` unutar koje ćemo pozvati obje funkcije za promjenu varijabli stanja. Kako bi bolje demonstrirali sinkronizirano osvježavanje, između dva poziva ćemo pozvati još jednu funkciju čija je svrha samo usporiti aplikaciju sa velikim brojem iteracija petlje. Pokušaje pokrenuti ovaj primjer:

App.jsx

```
function trosiVrijeme(){
  for (let i = 0; i < 1000000000; i ++){
    if (i % 10000000 == 0){
      console.log("Poruka")
    }
  }
}
```

```
function App() {
  const [broj, postaviBroj] = useState(5);
  const [drugi, postaviDrugi] = useState(99)

  function uvecajOba(){
    postaviBroj(broj + 1)
    trosiVrijeme()
    postaviDrugi(drugi + 1)
  }

  return (
    <div>
      <p>Vrijednost prvog broja: {broj}</p>
      <p>Vrijednost drugog broja: {drugi}</p>
      <button onClick={uvecajOba}>Uvećaj oba </button>
    </div>
  );
}
```

Funkcija "*uvecajOba*" sadrži unutar sebe tri poziva:

- uvećanje prvog brojača
- poziv funkcije sa "velikom" petljom
- uvećanje drugog brojača

Praćenjem ispisa u konzoli možemo vidjeti da se u istoj počinju prikazivati poruke, a da su na ekranu prikazane početne vrijednosti oba brojača. U trenutku kada se poruke počnu ispisivati, jasno je da smo došli do poziva funkcije "*trosiVrijeme*" i da je funkcija za povećanje prvog broja već pozvana. Unatoč tome, vrijednost brojača se ne uvećava sve dok se petlja ne izvrši do kraja i dođemo do poziva druge funkcije za promjenu stanja. Tek tada React izvršava obje promjene stanja i ponovno renderira komponentu - u tom trenutku vidimo da su oba brojača uvećana.

Bez ovog svojstva, prvi brojač bi se odmah uvećao što bi uzrokovalo ponovno renderiranje komponente i tada bi se našli u problemu - za vrijeme renderiranja bi se i dalje trebala izvršavati funkcija sa petljom te bi nam ostao još poziv druge funkcije za promjenu stanja koju treba pozvati. Zapravo bi imali na ekranu prikazano neko među-stanje aplikacije dok bi se još uvijek izvršavale naredbe iz prethodnog stanja. Tu dolazimo do pitanja što se događa sa korisničkim sučeljem dok još nisu sve promjene izvršene. Blokiranje sučelja nije dobra opcija jer bi korisnik stekao dojam neispravnog rada aplikacije. Interaktivno sučelje također nije dobra

ideja jer korisnik ponovno može pritisnuti tipku za uvećanje oba brojača - prije nego se drugi brojač stigao uvećati. To bi izazvalo dodatne probleme - kojim redoslijedom sada izvršiti promjene stanja i renderiranje? Nadalje, komponente često šalju svoje vrijednosti drugim komponentama i bez sinkroniziranih promjena možemo doći do situacije da nemamo konzistentne vrijednosti unutar naše aplikacije, što bi naravno gotovo sigurno dovelo do neispravnog rada same aplikacije.

Stanje kao objekt

Varijabla stanja ne mora nužno sadržavati samo jednu vrijednost. Kao i u "klasičnu" varijablu, u varijablu stanja možemo spremati različite tipove podataka. Prethodni primjer sa dva brojača možemo napisati na način da imamo samo jednu varijablu stanja koja sadrži objekt sa dva svojstva:

App.jsx

```
function App() {
  const [brojaci, postaviBrojace] = useState({prvi: 5, drugi: 10});

  return (
    <div>
      <p>Vrijednost prvog broja: {brojaci.prvi}</p>
      <p>Vrijednost drugog broja: {brojaci.drugi}</p>
    </div>
  );
}
```

Kod korištenja objekata za spremanje stanja potrebno je voditi računa o načinu na koji radimo promjene stanja. Za početak ćemo samo dodati nove *button* elemente i povezati ih sa praznim pomoćnim funkcijama:

App.jsx

```
function App() {
  const [brojaci, postaviBrojace] = useState({prvi: 5, drugi: 10});

  function uvecajPrvi(){
```

```
    //
  }

  function uvecajDrugi(){
    //
  }

  return (
    <div>
      <p>Vrijednost prvog broja: {brojaci.prvi}</p>
      <button onClick={uvecajPrvi}>Uvećaj prvi</button>
      <p>Vrijednost drugog broja: {brojaci.drugi}</p>
      <button onClick={uvecajDrugi}>Uvećaj drugi</button>
    </div>
  );
}
```

U ovom trenutku se potrebno podsjetiti da su stanja komponenti *immutable* tj. nepromjenjiva. U praksi to znači da ne možemo direktno mijenjati stanje objekta. Ovo ne bi bilo ispravno:

🚫 Ne smijemo direktno mijenjati varijablu stanja

```
function uvecajPrvi(){
  postavibrojace(brojaci.prvi = brojaci.prvi + 1)
}
```

Druga potencijalna pogreška može nastati ako šaljemo novi objekt kao iduće stanje, ali ne uključimo sva svojstva originalnog objekta stanja.

🚫 Prvi brojač će se ispravno uvećati ali ćemo izgubiti vrijednost drugog brojača.

```
function uvecajPrvi(){
  postaviBrojace({prvi: brojaci.prvi + 1})
}
```

Ispravni način je naravno poslati novi objekt koji sadrži sva svojstva kao i originalno stanje ali sa

novim vrijednostima:

```
function uvecajPrvi(){
  const novi = {
    prvi: brojaci.prvi + 1,
    drugi: brojaci.drugi
  }
  postaviBrojace(novi)
}
```

Kao što možete vidjeti, prvo smo napravili novi objekt sa istim svojstvima. Za vrijednost prvog svojstva smo uzeli vrijednost iz trenutnog stanja i uvećali je, dok smo za drugo svojstvo ostavili originalnu vrijednost (jer uvećavamo samo prvi brojač). Analogno ovome, možemo napisati funkciju i za uvećanje drugog brojača, samo ćemo ovaj put iskoristiti pogodnosti ES6 sintakse i pomoću *spread* operatora napraviti kopiju originalnog objekta - koju onda možemo mijenjati.

```
function uvecajDrugi(){
  let novi = {...brojaci}
  novi.drugi += 1
  postaviBrojace(novi)
}
```

Na ovaj način smo izbjegli potrebu za prepisivanjem svih svojstava koje ne želimo mijenjati (i njihovih vrijednosti). Jednostavno smo napravili kopiju postojećeg stanja, primijenili sve promjene na toj kopiji te je zatim iskoristili kako bi Reactu naznačili da je to novo stanje komponente koje želimo imati.

Last updated on March 7, 2023

‹ Komponente sa stanjem

Razmjena podataka ›