Promises

Róbert Novotný
UINF/KOPR
zima 2019

JavaScript v browseri beží v jednom vlákne

- ale chceme riešiť veci *na pozadí*
- nemôžeme čakať, kým dobehne HTTP/REST API request na pozadí, trvalo by to dlho, a vlákno by bolo zablokované

Node.js: asynchrónne programovanie na backende

- písanie výkonných a škálovateľných backendov
- syntax JavaScriptu + engine z Chrome
- event-driven (udalosti) filozofia bez vlákien

Riešenie asynchrónnosti

1. spustíme dlhotrvajúci proces
2. necháme sa upozorniť na dobehnutie
3. prípadne sa necháme upozorniť na chybu

Callbacks: tradičný spôsob

- funkcia berie parametre i callback
- po dobehnutí funkcie sa zavolá callback

```
setTimeout(ding, 3000);

function ding() {
  console.log("Ding!")
}
```

```
$.getJSON("entries", function(data, textStatus,
    data.data.forEach(function(entry) {
        var creator = entry.creator;
        $.getJSON(creator, function(data, textS
            console.log(data);
        });
    });
});
```

Reťazenie asynchrónnych volaní

- každý krok algoritmu vedie k ďalšej úrovni zanorenia a ďalším callbackom
- návratové hodnoty funkcií sa nepoužívajú!

Promises!

Promise

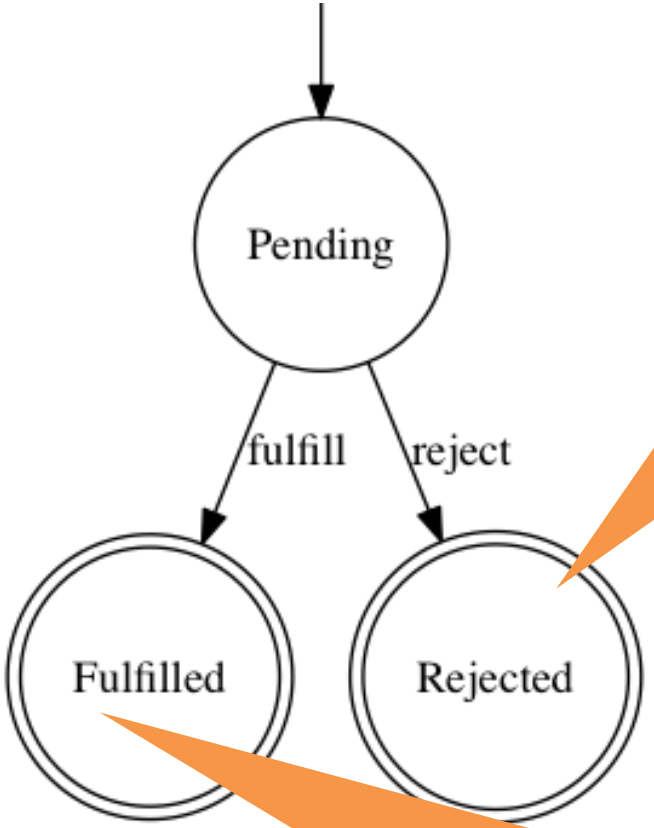
- **prísľub** budúceho výsledku z asynchrónnej operácie.
- objekty obaľujúce výsledok operácie, ktorý nemusí byť hneď dostupný

Java: CompletableFuture

Čo je Promise?

Promise je objekt/funkcia s metódou **then()**, ktorá sa správa podľa špecifikácie Promises/A+

Stavy promisu



zamietnutý. Nesie *reason*, dôvod zamietnutia

a.k.a. resolved = nesie v sebe hodnotu

Metóda then() má dva parametre

1. funkcia onFulfilled:
   - zavolaná po splnení prísľubu
   - prvým parametrom je hodnota prísľubu
2. funkciu onRejected
   - zavolaná po zamietnutí prísľubu
   - prvým parametrom je dôvod zamietnutia

Handlery/callbacky a návratové hodnoty

nič

- promise bude splnený

bežná hodnota

- promise bude splnený

iný promise

- promise si spriahne stav s týmto iným promisom

výnimka

- promise bude zamietnutý

Reťazenie then()

- then() vracia nový promise
  - splnený po dobehnutí onFullfilled
  - alebo zamietnutý po dobehnutí onRejected
- then() možno reťaziť
  - handlery sa volajú v poradí registrácie

then() je programovateľná bodkočiarka

Minimalistický príklad

ES6 AJAX

```
fetch(url)
    .then(res => showResult(res.statusText))
```

Reťazenie then()-ov

výsledkom je promise

```
fetch(url)
    .then(res => res.json())
    .then(res => showResult(res))
```

druhý promise spriahne stav s promisom z json()

Chybové stavy

```
fetch(url)
    .then(res => res.json())
    .then(json => showResult(json))
    .catch(e => showResult("Error!")
```

reťazenie troch handlerov!

cukor pre then(NIČ, chyba)

async-await

- v modernom JavaScripte (ES2017) a
- v C#
- moderný spôsob práce s prísľubmi

asynchrónny prístup bez callbackov!

await

- odbaľuje hodnotu / chybu z promisu
- umožňuje ju priradiť do premennej obvyklým spôsobom
- zamietnuté prísľuby prevádza na výnimky

```
let res = await fetch(url);
console.log(res.statusText)
```

async

- označuje funkciu ako **asynchrónnu**
- návratové hodnoty funkcie obalí do prísľubu, ak treba
- len v takejto funkcii možno používať **await**

```
async function findUserByAlbum(album) {
    let res = await fetch("http://...");
    return response.json()
}
```

json() vracia promise, netreba obaľovať

async-await

- klasický zápis promisov chápateľný programátormi
- priraďujem do premennej
  - žiaden callback
- používam výnimky
  - žiaden callback
- používam bodkočiarky
  - žiaden then()

Porovnanie prístupov

- "hlúpe" **callbacky**:
  - jednoduché, dostupné všade, štandard v Node.js
- **promises**
  - prehľadnejšia syntax
  - súčasť ES2015
- **async/await**
  - skoro "klasický" zápis asynchrónnych úloh
  - čakáme na masovú dostupnosť

Otázky?