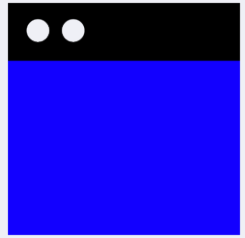
Code Academy



Gytis Juozenas

Asinchroninis programavimas (promises, callbacks)

2022

JavaScript programavimo kalba

Šiandien išmoksite

01 Asinchroninis programavimas (promises, callbacks)

Asinchroninis programavimas

Javascript yra sinchroninis programavimas (skaito vieną eilutę vienu metu).

JavaScript yra vienos gijos (angl. *single-threaded*) programavimo kalba, o tai reiškia, kad vienu metu gali įvykti tik vienas dalykas. Trumpai tariant, JavaScript variklis vienu metu gali apdoroti tik vieną teiginį vienu metu.

Nors vienos gijos programavimo kalbos supaprastina kodo rašymą, nes jums nereikia jaudintis dėl sutapimo/konkuravimo problemų, tai taip pat reiškia, kad negalite atlikti ilgų operacijų, tokių kaip prieiga prie tinklo, neužblokuodami pagrindinės gijos atlikimo.

Pavyzdžiui, įsivaizduokite, kad kreipiatės duomenų į API. Priklausomai nuo situacijos, serveris gali užtrukti, kol apdoros užklausą, užblokuodamas pagrindinę giją, kad tinklalapis neatsakytų.

Čia atsiranda **asinchroninis JavaScript**. Naudodami asinchroninį JavaScript (pvz., callbacks, promises, arba async/await), galite atlikti ilgas tinklo užklausas neužblokuodami pagrindinės gijos.

Asinchroninis programavimas

Kaip veikia sinchroninis JavaScript?

Svarbios sąvokos:

- **Execution Context** yra abstrakti aplinkos, kurioje įvertinamas ir vykdomas JavaScript kodas, samprata.
- **Call stack** yra krūva su LIFO (Last in, First out) struktūra, naudojama visam vykdymo kontekstui (execution context), sukurtam vykdant kodą, saugoti.

Paanalizuokime šį kodą (jo call stack'as pateiktas dešinėje):

```
const second = () => {
  console.log('Hello there!');
}
const first = () =>
  { console.log('Hi
  there!'); second();
  console.log('The End');
}
first();
```

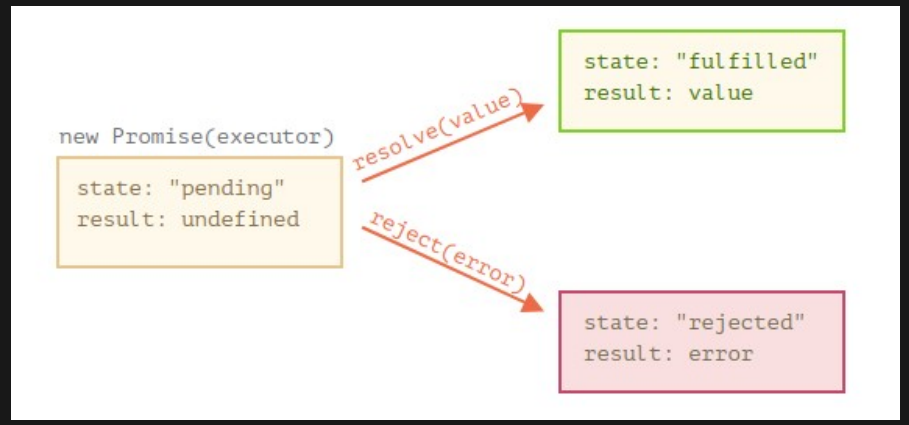
Asinchroninis programavimas

Kaip veikia asinchroninis JavaScript?

Paanalizuokime šį kodą (atlikimas dešinėje):

```
const networkRequest = () => {
  setTimeout(() =>
    { console.log('Async
    Code');
  }, 2000);
};

console.log('Hello World');
networkRequest();
console.log('The End');
```



Asinchroninis programavimas (Promises)

Promises

New Promise konstruktoriaus grąžintas *Promise* objektas turi šias savybes:

- *state*:
  - iš pradžių "pending";
  - tada pasikeičia į "fulfilled" kai *resolve* yra iškviestas arba "rejected", kai *reject* yra iškviestas;

- *result*:
  - iš pradžių *undefined*, tada keičiasi į
  - *value*, kai *resolve(value)* iškviestas arba *error* kai reject(error) yra iškviestas..

Practice Time

Laisva užduotis!
Išbandykite Async programavimą pasirašydami bent tris funkcijas ir panaudodami timeout taip kad jums būtų aišku.

```
const networkRequest = () => {
  setTimeout(() =>
  { console.log('Async Code');
  }, 2000);
};
console.log('Hello World');
networkRequest();
console.log('The End');
```

Practice Time

1. Susikurkite HTML failą kuris iš esmės empty bus apart tuščio p tag‘o;
2. JS apsirašykite tą patį kodą dviem būdais: sinchroniškai ir asinchroniškai;
3. Kodas pasieks tuščia p tag‘ą nustatys jam kažkokį tekstą iš‘alert‘ins tą tekstą ir pakeis jo spalvą. Vienu atveju tai atliks sinchroniškai, kitu atveju timeout bus uždėtas alert‘ui.
4. Pastebėsite keistą dalyką su alert‘u ir teksto atsiradimu, pasibandykite kad tai įvyktų atvirkščiai nei gausis, t.y. Susikeistų vietomis alert‘as su teksto atsiradimu puslapyje.

Asinchroninis programavimas (Promises)

Promises

Tai objektas reprezentuojantis kažko sėkmingą įvykdymą arba kaip tik žlugimą/neįvykdymą asinchroninėje operacijoje. Kitaip tariant tai yra pažadas kaip ir realybėje, tik šiuo atveju jums kodas panaudojus promise prižada kažką padaryti, tačiau visuomet yra dvi baigtys, pažado tesėjimas (**resolve**) ir pažado netesėjimas (**reject**).

Resolve bus vykdoma tuomet kai pažadas yra tęsiamas (žinau, labai smagiai skamba, bet tikiuosi suprantate) ir jis iškviečiamas su .then(). .then() paima metodą ir pasiima parametrą/us.

Reject bus vykdoma tuomet kai pažadas yra netęsiamas kitaip tariant prieinamas error o ne rezultatas kurio norima ir jis iškviečiamas su .catch(). .catch() paima metodą ir pasiima parametrą/us.

PAVYZDYS:

```
let prom = new Promise((resolve, reject) => {
  let num = Math.random();
  if (num < 0.5) {
    resolve("Yay! Resolved: " + num);
  } else {
    reject("Ohhh noooo!: " + num);
  }
});

prom.then((res) => console.log(res)).catch((err) => console.log(err));
```

```
new Promise((resolve, reject) => {
    console.log('Initial');

    resolve();
})
.then(() => {
    throw new Error('Something failed');

    console.log('Do this');
})
.catch(() => {
    console.error('Do that');
})
.then(() => {
    console.log('Do this, no matter what happened before');
});
```

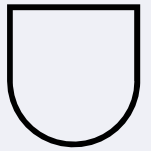
Asinchroninis programavimas (Promises) ir callback

Promises vs callbacks

Jeigu atkreipsite dėmesį, suprasite jog promises tai yra tiesiog gražesnis callback būdas.

```
const makePB&J = () => {
makeBread(function() {
 putPeanutButter(function(...args) {
   spreadJelly(function(...args) {
     sandwichThem(bread,peanutButter,jelly){
    // welcome to hell
     });
   });
  });
});
};
```

```
const makePb&J = () => {
 return makeBread()
   .then(peanut => putPeanutButter(peanut))
   .then(jelly => spreadJelly(jelly))
   .then(sandwich => sandwichThem(sandwich));
 catch((ewww crunchyPeanutButter));
};
```



callback hell

Callback hell

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Practice Time

Naudodami promise parašykite programą, kuri gamina picas:

1. Sukurkite anonymous funkciją kuri turi Promise su resolve ir reject bei pasiima vieną property - toppings;
2. Promise tikrina ar picos ingridientuose yra ananasas, jei yra panaudojamas reject ir išmetama kažkokia pikta žinutė. Kitu atveju suveikia resolve kuris pasako štai tavo pica su tokiais ir tokiais ingridientais. Nepamirškit returninti šio Promise!
3. Susikurkite du kintamuosius kurie iškviečia picos pagaminimo funkciją ir ją iškviesdami pateikia bent po du ingridientus, tik kadangi jūs panaudojote vieną property aprašydami funkciją, pagalvokite kaip reikės tai padaryti;
4. Išsikvieskite vieną iš kintamųjų ir panaudokite .then() ir .catch() gauti rezultatui. Išbandykite abudu kintamuosius.

```
let prom = new Promise((resolve, reject) => {
  let num = Math.random();
  if (num < 0.5) {
    resolve("Yay! Resolved: " + num);
  } else {
    reject("Ohhh noooo!: " + num);
  }
});

prom.then((res) => console.log(res)).catch((err) => console.log(err));
```