Uniwersytet Wrocławski

Instrukcje asynchroniczne w JavaScript

Paweł Grabiński

Wydział Fizyki i Astronomii

Uniwersytet Wrocławski

2 maja 2018

Plan prezentacji

- Motywacja: synchroniczne vs asynchroniczne
- Callback Hell
- Promised Land
  - Łańcuchy obietnic (Promise chains)
  - Reduktory obietnic
  - *async/await*
- Strumienie reaktywne (Reactive streams) w **rxJS**
  - Observable
  - Subject

```
1 let user = getUserData(id);
2 // Request goes to the db
3 console.log(user);
4 // Logs undefined
```

Synchroniczne vs asynchroniczne

Sync vs Async
blocking vs non-blocking

Asynchroniczne zadania:

- setInterval & setTimeout
- eventy (“click”, “hover”, “custom”, itp.),
- zapytania API i baz danych.

Synchroniczne - blokujące zadania:

- operacje I/O,
- odczytywanie danych z dysku (NodeJS),
- synchroniczne zapytania HTTP.

Callbacks

```
1  // Callback evoked with delay
2  setTimeout( () => { console.log('I was waiting for 2
       seconds!'); }, 2000);
```

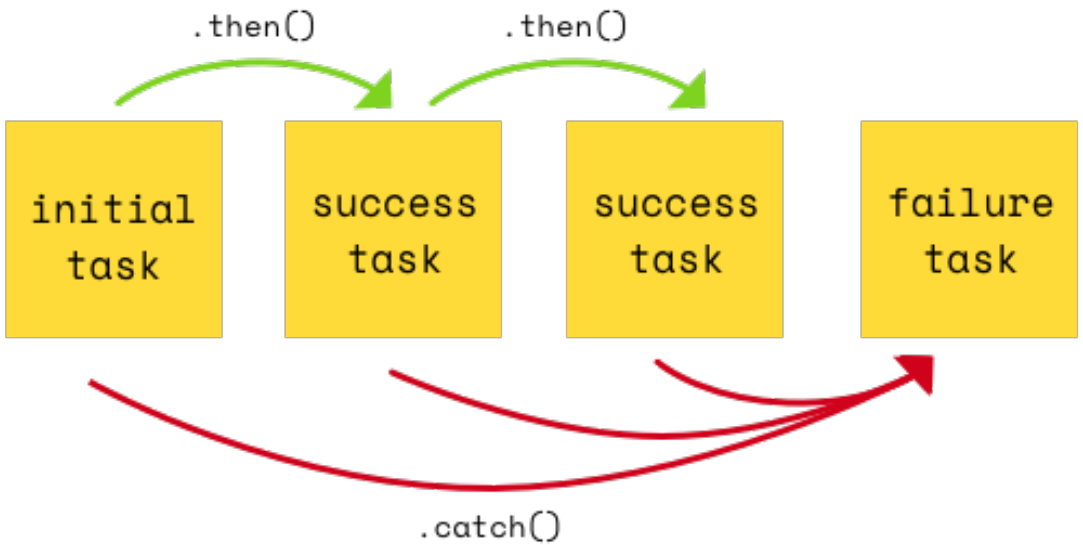
```
1  // GET request with JQuery
2  function getUserData(userId,callback) {
3    $.get('users/' + userId, callback);
4  }
5
6  // Logging callback function
7  getUserData('01',
8    (userData) => {
9        console.log(userData.name);
10     }
11 );
```

Callback Hell (from callbackhell.com)

```
1  fs.readdir(source, function (err, files) {
2    if (err) {
3      console.log('Error finding files: ' + err)
4    } else {
5      files.forEach(function (filename, fileIndex) {
6        console.log(filename)
7        gm(source + filename).size(function (err, values) {
8          if (err) {
9            console.log('Error identifying file size: ' + err)
10         } else {
11           console.log(filename + ' : ' + values)
12           aspect = (values.width / values.height)
13           widths.forEach(function (width, widthIndex) {
14             height = Math.round(width / aspect)
15             console.log('resizing ' + filename + 'to ' + height + 'x' + height)
16             this.resize(width, height).write(dest + 'w' + width + '_' + filename
                   , function(err) {
17               if (err) console.log('Error writing file: ' + err)
18             })
19           }.bind(this))
20         }
21       })
22     })
23   }
24 })
```

Promises - obietnica lepszej przyszłości

```
1  new Promise((resolve, reject) => {
2     ...
3       // resolve(value) - returns requested value
4       // reject(reason) - returns reason of failure
5  });
```



Łańcuchy obietnic (Promise chains)

```
1  // Promise style handling
2  getData(testOptions)
3    .then(...)
4      .catch(...)
```

```
1  sleep(1000)
2    .then(() => {
3        console.log('one');
4        return sleep(1000);
5    })
6    .then(() => {
7        console.log('two');
8        return sleep(1000);
9    })
10   .then(() => {
11       console.log('three');
12   })
13   .catch( err => console.error(err));
```

Reduktory obietnic - synchronizacja

Reduktory synchronizują równoległe instrukcje asynchroniczne.

- *all*
- *race*
- *any* (bluebird.js)
- *some* (bluebird.js)

```
1  Promise.all([p1, p2, p3, p4])
2    .then( ([v1, v2, v3, v4]) => {...})
3    .catch( ([err1, err2, err3, err4]) => {...});
4
5  Promise.some([p1, p2, p3, p4],2)
6    .then( ([v1, v2]) => {...})
7    .catch( ([err1, err2]) => {...});
```

yield i generatory funkcji

```
1  function* generatedFunction() {
2    let A = 1;
3    yield A;
4    let B = 2;
5    yield B;
6    let C = 3;
7    yield C;
8  }
9
10 let gen = generatedFunction();
11 consol.log(gen.next());
12 // logs 1
13 consol.log(gen.next());
14 // logs 2
15 consol.log(gen.next());
16 // logs 3
```

async/await

```
1  async function getUser(id) {
2    let user = await fetch('https://jsonplaceholder.
         typicode.com/users/' + id)
3    .then( response => response.json() );
4    return user;
5  }
6
7  getUser(1)
8    .then( user => console.log(user) );
```

rxJS - Observable

Zbiór wartości emitowany w czasie.

```
1 var result = Rx.Observable.fromPromise(fetch('http://
      myserver.com/'));
2 result.subscribe(x => console.log(x), e => console.
      error(e));
3
4 const timer = Rx.Observable.timer(0, 1000);
5 timers.subscribe(value => console.log(value));
```

Trzy możliwe zachowania *subscribe*:

- *next*
- *error*
- *complete*

rxJS - Subject

EventEmitter w postaci Observabli.

```
1  export class WishService {
2    wishlistById = new Subject<Wishlist>();
3      emmitCurrentWishlist() {
4        this.wishlistById
5          .next({...this.currentWishlist});
6      }
7  }
8
9  export class WishlistComponent {
10     ngOnInit() {
11       this.wisherSub = this.wishService.wishlistById
12       .subscribe(
13           (wishList: Wishlist) => {
14               ...
15           }
16       );
17     }
18 }
```

Materiały dodatkowe i kontakt

- YouTube: *5 Architectures of Asynchronous JavaScript*
- YouTube: *Async/Await: Modern Concurrency In Java Script*
- Pluralsight course: *Reasoning About Asynchronous JavaScript*
- callbackhell.com: *A guide to writing asynchronous JavaScript programs*
- Pluralsight blog: Introduction to asynchronous JavaScript

Paweł Grabiński

pawelrgrabinski@gmail.com

pgrabinski.pl