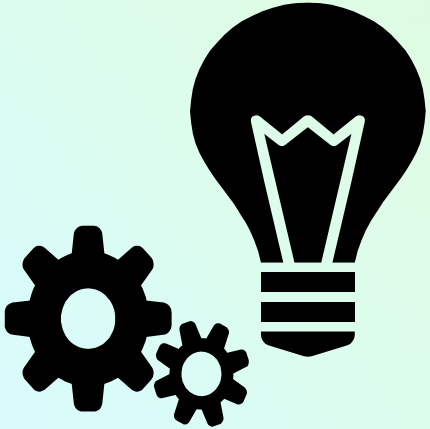
ES6 Generators

Asynchronous Flow Control

A Tech Talk by Fanny Jiang
April 25th, 2017

What are Generators?



A generator is a special function that you can pause and resume at many points.

Generators help us write asynchronously

Perform asynchronous functions inside existing control flow structures such as loops, conditionals, and try/catch blocks

JS basic-gen.js demo

```js
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

Anatomy of a Generator

JS basic-gen.js demo

```js
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

Anatomy of a Generator

JS basic-gen.js demo

```js
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

Anatomy of a Generator

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next()
{ value: 1, done: false }
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next()
{ value: 1, done: false }
> iterator.next()
first undefined
{ value: 2, done: false }
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next()
{ value: 1, done: false }
> iterator.next()
first undefined
{ value: 2, done: false }
> iterator.next()
second undefined
{ value: 3, done: false }
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next()
{ value: 1, done: false }
> iterator.next()
first undefined
{ value: 2, done: false }
> iterator.next()
second undefined
{ value: 3, done: false }
> iterator.next()
third undefined
{ value: undefined, done: true }
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 3 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next()
{ value: 1, done: false }
> iterator.next()
first undefined
{ value: 2, done: false }
> iterator.next()
second undefined
{ value: 3, done: false }
> iterator.next()
third undefined
{ value: undefined, done: true }
> iterator.next()
{ value: undefined, done: true }
>
```

basic-gen.js - Tech-Talk

basic-gen.js

```javascript
// declare generator with function* keyword

function *firstGen() {
  console.log('first', yield 1)
  console.log('second', yield 2)
  console.log('third', yield 3)
}
```

master* 0 4 Ln 27, Col 1 Spaces: 2 UTF-8 LF Javascript (Babel) ESLint

1. node

```
Fanny Tech-Talk $ node
> var myGen = require('./demo/basic-gen')
undefined
> var iterator = myGen()
undefined
> iterator.next('FIRST')
{ value: 1, done: false }
> iterator.next('SECOND')
first SECOND
{ value: 2, done: false }
> iterator.next('THIRD')
second THIRD
{ value: 3, done: false }
> iterator.next('WHERE\'S FIRST?')
third WHERE'S FIRST?
{ value: undefined, done: true }
>
```

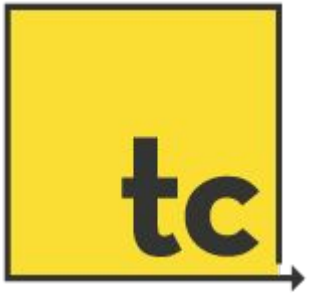
“ Simple enough, but what’s the point? How can it help me write asynchronous functions?

When a generator yields a Promise

- Generator will wait for promise to resolve
- Resolved promise gets passed back to the generator and to its assigned variable
- Make asynchronous calls to servers and databases

Generator Libraries

- Co
- Bluebird
- Google Traceur



JS demo.js demo

```javascript
// Example:  make asynchronous requests to create new instances in database
const co = require('co')
const User = require('User')
const Hobby = require('Hobby')

/* <------------ with promises only -----------------> */

function makeHobbies() {
  // creating new instances in db return promises
  const jessDay = User.create({firstName: 'Jess', lastName: 'Day'}),
        crafting = Hobby.create({name: 'crafting'}),
        singing = Hobby.create({name: 'singing'})

  return Promise.all([jessDay, crafting, singing])
    .then(([jessDay, crafting, singing]) => {
      return Promise.all([
        jessDay.addHobby(crafting),
        jessDay.addHobby(singing)
      ])
    })
}

/* <--------------- with generator -----------------> */

co(function *makeHobbs() {
  const jessDay = yield User.create({firstName: 'Jess', lastName: 'Day'}),
        crafting = yield Hobby.create({name: 'crafting'}),
        singing = yield Hobby.create({name: 'singing'})

  jessDay.addHobby(yield crafting)
  jessDay.addHobby(yield singing)
})
```

Example: make async requests to database using Sequelize

Promises Only

- More lines of code
- Passing several promises down chain

Generator

- Cleaner code
- Looks synchronous

Learn more!

Videos:

- **funfunfunction: Generators in JS - What, Why and How**
  https://www.youtube.com/watch?v=QO07THdLWQo
- **LearnCode.academy: JS Generators - They Change Everything**
  https://www.youtube.com/watch?v=QO07THdLWQo

Articles:

- http://exploringjs.com/es6/ch_async.html
- https://blog.risingstack.com/asynchronous-javascript/
- https://medium.com/javascript-scene/the-hidden-power-of-es6-generators-observable-async-flow-control-cfa4c7f31435
- https://davidwalsh.name/es6-generators
- https://davidwalsh.name/async-generators
- https://kaye.us/javascript-async-control-flow/

Resources

Tools and Docs:

- **MDN - Iterators and Generators:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators_and_Generators
- **MDN - Promises** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- **co:** https://github.com/tj/co
- **Bluebird:** http://bluebirdjs.com/docs/getting-started.html
- **Traceur:** https://github.com/google/traceur-compiler

Thanks!

Any questions?

Email: fanny.jiang@outlook.com

Blog: medium.com/@heygirlcode