Sincronizando Código em JS

Stéfano Stypulkowski Zanata

Stéfano
Stypulkowski

Zanata

Código Assíncrono?

```
// Exemplo 1



requestPerson( person => {

  // I have a person

});
```

```
// Exemplo 2



requestPerson( person => {
  const dbPerson = new mongoose.models.Person( person );
  dbPerson.save( err => {
    // I have a db person
  });
});
```

```
// Exemplo 3

requestPersonBasicData( basic => {
  requestPersonExtas( extras => {
    const person = Object.assign( {}, basic, extras );
    const dbPerson = new mongoose.models.Person( person );
    dbPerson.save( err => {
      requestEnrichments(person, enrichments => {
        dbPerson.setEnrichments( enrichments );
        dbPerson.save( err2 => {
          // I have a complete person
        });
      });
    });
  });
});
```


CALLBACK HELL

Dificuldade de Entendimento

Profundidade de **Código**


Complicado
tratar erros

Inversion of Control


Difícil de
testar


Não modular

E agora?


Promises

Promises

Uma *promise* representa um valor que pode estar disponível **agora**, no **futuro**, ou pode **nunca** estar disponível

Mais que isso, uma promise é **confiável**

// Promises

```
Promise.all( [ requestPersonBasicData(), requestPersonExtas() ] ).then( values => {
  const person = Object.assign( {}, ...values );
  const dbPerson = new mongoose.models.Person( person );
  dbPerson.save().then( requestEnrichments ).then( enrichments => {
    dbPerson.setEnrichments( enrichments );
    return dbPerson.save();
  }).then( () => {
    // I have a complete person
  });
});
```


PROMISE
ALL THE THINGS

Melhor?


SIM

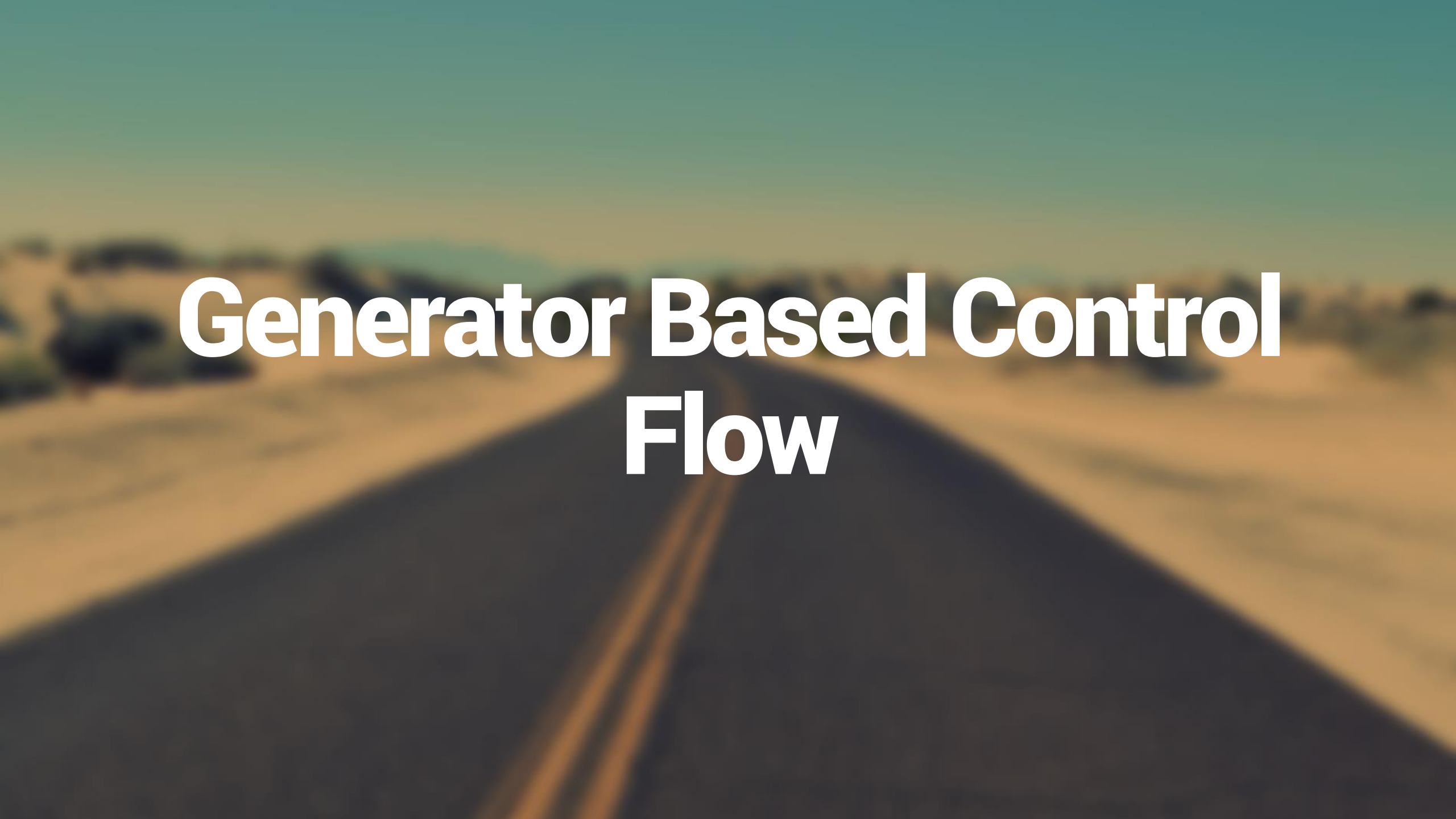

MAS AINDA NÃO ESTÁ BOM

Generators
+
Promises

Generators

Grosseiramente: São funções que podem ser **pausadas**

Operador ***yield*** dentro da função age como um ponto de entrada e saída


Generator Based Control
Flow

Generator based control flow

Controla o fluxo do código baseado na capacidade dos generators de **pausarem sua execução**

Usa **promises** como retorno das funções

Permite um **comportamento linear**

```
// Generator

run( function *() {
  const basic = requestPersonBasicData();
  const extras = requestPersonExtas();
  const person = Object.assign( {}, yield basic, yield person );
  const dbPerson = new mongoose.models.Person( person );
  yield dbPerson.save();
  const enrichments = requestEnrichments();
  dbPerson.setEnrichments( enrichments );
  yield dbPerson.save();
  // I have a complete person
});
```

```
// Run.js


const run = require( 'simplerunner' );

run(function *() {
  yield ...
  return value;
});
```

```
// Run.js chain

function runnable1() {
  return run(function *() {
    return 1;
  });
});

run(function *() {
  const v = yield runnable1();
  // v === 1
});
```

```
// Run.js error handling



run(function *() {
  const v = yield runnable1();
  // v === 1
}).catch(function (err) {

});
```

```
// Run.js error handling


run(function *() {
  try {
    yield runnable1();
  } catch (e) {
    // treat me
  }
});
```

```
// Run.js parallel



run(function *() {
  const v = runnable1();
  const v2 = runnable2();
  return { v: yield v, v2: yield v2 };
});
```

Run.js

Implementação original de **Kyle Simpson** e **Benjamin Gruenbaum**

You Don't Know JS: Async & Performance, de **Kyle Simpson**

Fonte no github: **github.com/madeinstefano/run**

Pacote do **npm**

```
npm install simplerunner
```

Implementações

co

github.com/tj/co

github.com/kriskowal/q

ASQ

github.com/getify/asynquence


Wow, vou começar a usar agor...
Quieto Robin, isso não é nativo!

async / await

```
// Async & await
async function() {
  const basic = requestPersonBasicData();
  const extras = requestPersonExtas();
  const person = Object.assign( {}, await basic, await person );
  const dbPerson = new mongoose.models.Person( person );
  await dbPerson.save();
  const enrichments = requestEnrichments();
  dbPerson.setEnrichments( enrichments );
  await dbPerson.save();
  // I have a complete person
});
```


Syntax Sugar

```
// Async & await



async function () {
  await ...;
  return result; // ou Promise
}
```

```
// Async & await chain


async function request() {
  return something;
}

async function process() {
  await request();
}
```

```
// Async & await parellel



async function proccess() {
  return { a: await one(), b: await two() };
}
```

```
// Async & await self executable



(async function () {
  await ...
}());
```

```
// Async & await error handling

async function request() {
  // do async stuff
}
async function process() {
  await request().catch( err => {
    // treat me
  });
}
```

```
// Async & await error handling

async function request() {
  // do async stuff
}
async function process() {
  try {
     request()
  } catch ( e ) {
    // treat me
  }
}
```

Aonde?

ES8 (2017)

Firefox 52

Google V8 5.5

Chrome 55

Node.js 7.6

Legal, mas **por que** devo usar?

Acaba com o Callback Hell

Código Simples e Legível

Modular e **testável**

```
// Mocha com promises


it('Should test a promise', function () {
  return myPromise().then( result => {
    expect( result.prop ).to.eql( 'value' )
  });
});
```

```
// Mocha com async & await



it('Should test a promise', async function (){
  await process();
  expect( assertion ).to.eql( 'value' );
});
```


TAKE MY MONEY!


OBRIGADO!

Referências

Simpson, K. You Don't Know JS: *Async & Performance*. Disponível em <https://github.com/yusufdoru/Sen-JavaScript-Bilmiyorsun/blob/master/async%20&%20performance/README.md>

_____. Going Async With ES6 Generators. Disponível em < https://davidwalsh.name/async-generators>

MDN. async function. Disponível em <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function>

Rauschmayer, Dr. A. ECMAScript 2017 (ES8): the final feature set. Disponível em <http://2ality.com/2016/02/ecmascript-2017.html>

_____. The final feature set of ECMAScript 2016 (ES7). Disponível em <http://2ality.com/2016/01/ecmascript-2016.html>

madein**stefano**

szanata.com