Koa

эволюция middleware

Цепочка асинхронных операций

Callbacks

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  action1(function(err, result1) {
    if (err) { /** handle error */}
    action2(result1, function(err, result2) {
      if (err) { /** handle error */}
      action3(result2, function(err, result3) {
        if (err) { /** handle error */}
        res.send(JSON.stringify(result3))
      })
    })
  })
})
```

Проблемы

– Трудночитаемый код
– Дублирование обработки ошибок

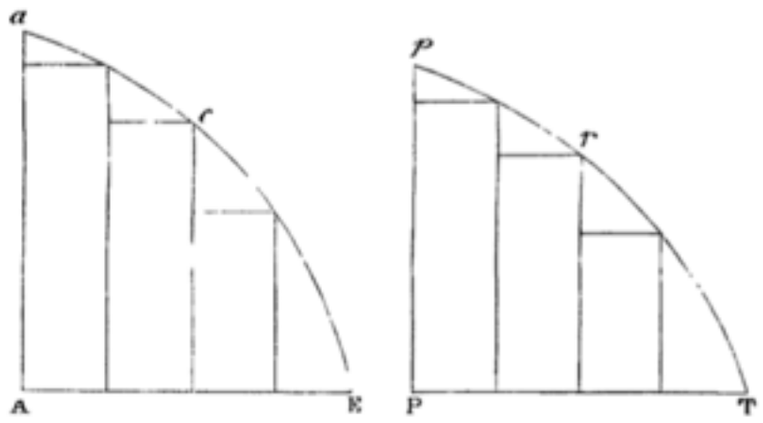
Async/await

```
import Koa from 'koa'
const app = new Koa()

app.use(async (ctx, next) => {
  let result1 = await action1()
  let result2 = await action2(result1)
  let result3 = await action3(result2)
  ctx.body = JSON.stringify(result3)
})
```

Что нас удерживает?

– Новый синтаксис

ad singula, sint eædem; dico quod figuræ duæ A a c E, P p r T, *sunt ad invicem in eadem illa ratione.*



Etenim ut sunt parallelogramma singula ad singula, ita (componendo) fit summa omnium ad summam omnium, & ita figura ad figuram; existente nimirum figura priore (per lemma 111) ad summam priorem, & figura posteriore ad summam posteriorem in ratione æqualitatis. *Q. E. D.*

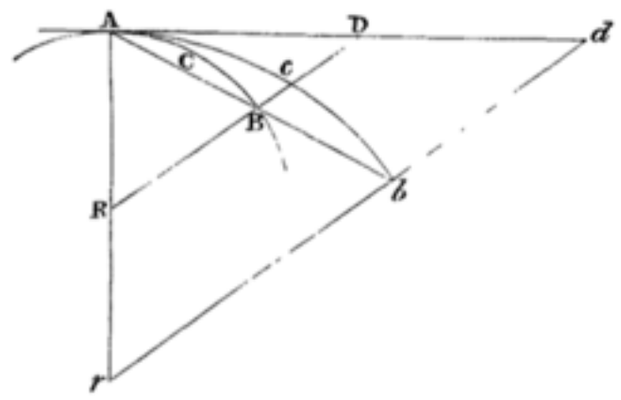
Corol. Hinc si duæ cujuscunque generis quantitates in eundem partium numerum utcunque dividantur; & partes illæ, ubi numerus earum augetur & magnitudo diminuitur in infinitum, datam obtineant rationem ad invicem, prima ad primam, secunda ad secundam, cæteræque suo ordine ad cæteras: erunt tota ad invicem in eadem illa data ratione. Nam si in lemmatis hujus figuris sumantur parallelogramma inter se ut partes, summæ partium semper erunt ut summæ parallelogrammorum; atque ideo, ubi partium & parallelogrammorum numerus augetur & magnitudo diminuitur in infinitum, in ultima ratione parallelogrammi ad parallelogrammum, id est (per hypothesin) in ultima ratione partis ad partem.

LEMMA V.

Similium figurarum latera omnia, quæ sibi mutuo respondent, sunt proportionalia, tam curvilinea quam rectilinea; & areæ sunt in duplicata ratione laterum.

LEMMA VI.

Si arcus quilibet positione datus A C B *subtendatur chorda* A B, *& in puncto aliquo* A, *in medio curvaturæ continuæ, tangatur a recta utrinque producta* A D; *dein puncta* A, B *ad invicem accedant & coëant; dico quod angulus* B A D, *sub chorda & tangente contentus, minuetur in infinitum & ultimo evanescet.*



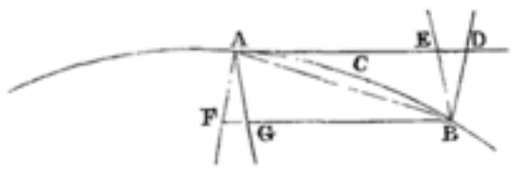
Nam si angulus ille non evanescit, continebit arcus *A C B* cum tangente *A D* angulum rectilineo æqualem, & propterea curvatura ad punctum *A* non erit continua, contra hypothesin.

LEMMA VII.

Iisdem positis; dico quod ultima ratio arcus, chordæ, & tangentis ad invicem est ratio æqualitatis.

Nam dum punctum *B* ad punctum *A* accedit, intelligantur semper *A B* & *A D* ad puncta longinqua *b* ac *d* produci, & secanti *B D* parallela agatur *b d*. Sitque arcus *A c b* semper similis arcui *A C B*. Et punctis *A*, *B* coeuntibus, angulus *d A b*, per lemma superius, evanescet; ideoque rectæ semper finitæ *A b*, *A d*, & arcus intermedius *A c b* coincident, & propterea æquales erunt. Unde & hisce semper proportionales rectæ *A B*, *A D*, & arcus intermedius *A C B* evanescent, & rationem ultimam habebunt æqualitatis. *Q. E. D.*

Corol. 1. Unde si per *B* ducatur tangenti parallela *B F*, rectam quamvis *A F* per *A* transeuntem perpetuo secans in *F*, hæc *B F* ultimo ad arcum evanescentem *A C B* rationem habebit æqualitatis, eo quod completo parallelogrammo *A F B D* rationem semper habet æqualitatis ad *A D*.



Lauros-Giraudon

$x:y+\overline{\overline{a+bxc-xx}}:$quadrat. ex $ex+fxx+ax\sqrt{gg+yy}+yy:\sqrt{hh+lx+mxx}$: aequ. 0, exprimens relationem inter x et y seu inter AX et XY, posito ipsas $a, b, c, e, f, g, h, l, m$ esse datas; quaeritur modus ex dato puncto Y educendi YD, quae curvam tangat, seu quaeritur ratio rectae DX ad rectam datum XY. Compendii causa pro $a+bx$ scribamus n; pro $c-xx$, p ; pro $ex+fxx$, q ; pro $gg+yy$, r ; pro $hh+lx+mxx$, s ; fiet

$x:y+np:qq+ax\sqrt{r}+yy:\sqrt{s}$ aeqti. 0, quae sit aequatio *secunda.* Jam ex calculo nostro constat $d,x:y$ esse $\pm xdy+ydx:yy$; et similiter $d,np:qq$ esse

$(\pm)2npdq(\mp)qndp+pdn,:q^3$ et $d,ax\sqrt{r}$ esse $+axdr:2\sqrt{r}+adx\sqrt{r}$; et

$d,yy:\sqrt{s}$ esse$\big((\pm)\big)yyds\big((\mp)\big)4ysdy:2s\sqrt{s}$, quae omnes quantitates differentiales inde ab ipso $d,x:y$ usque, ad $d,yy:\sqrt{s}$ in unum additae facient 0, et dabunt hoc modo aequationem *tertiam,* ita enim pro membris secundae aequationis substituuntur quantitates eorum differentiales. Jam dn est bdx, et dp est $-2xdx$, et dq est $edx+2fxdx$, et dr est $2ydy$, et ds est $ldx+2mxdx$. Quibus valoribus in aequatione tertia substitutis habebitur aequatio *quarta,* ubi quantitates differentiales, quae solae supersunt, nempe dx, dy, semper reperiuntur extra nominatores et vincula, et unumquodque membrum afficitur vel per dx, vel per dy, servata semper lege homogeneorum quoad has duas quantitates, quomodocunque implicatus sit calculus : unde semper haberi potest valor ipsius $dx:dy$ seu rationis dx ad dy, hoc est DX quaesitae ad XY datam, quae ratio in hoc nostro calculo (mutando aequationem quartam in Analogiam) erit ut

$$\mp x:yy-axy:\sqrt{r}\big((\mp)\big)2y:\sqrt{s} \text{ est ad}$$

$$\mp 1:y(\pm)2npe+2fx,:q^3(\mp)-2nx+pb:qq+a\sqrt{r}\big((\pm)\big)yyl+2mx:2s\sqrt{s}.$$

Dantur autem x et y ex dato puncto Y. Dantur et valores supra scripti literarum n, p, q, r, s per x et y. Habetur ergo quaesitum. Atque hoc exemplum satis implicatum ideo tantum ascripsimus, ut modus superioribus regulis in calculo etiam difficiliore utendi appareret. Nunc praestat usum in exemplis intellectui magis obviis ostendere.

Data sint duo puncta C et E (fig.112), et recta SS in eodem cum ipsis plano; quaeritur punctum F in SS ita sumendum, ut junctis CF, EF, sit aggregatum rectangulorum CF in datam h, et FE in datam r , omnium possibilium minimum, hoc est si SS sit mediorum separatrix , et h repraesentet densitatem medii ut aequae a parte C , et r densitatem medii ut aeris a parte E, quaeritur punctum F tale, ut via a C ad E per F sit omnium possibilium facillima. Ponamus omnia ista rectangulorum aggregata possibilia, vel omnes viarum possibilium difficultates, repraesentari per ipsas KV, curvae VV ordinatas ad rectam GK normales, quas vocabimus ω, quaerique minimam earum NM. Quia dantur puncta C et E, dabuntur et perpendiculares ad SS, nempe CP (quam

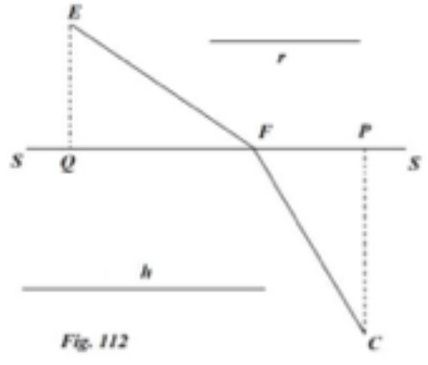


Fig. 112

vocabimus c) et EQ (quam e) et praeterea PQ (quam p), ipsa autem QF, quae sit aequalis ipsi GN (vel AX), vocabimus x et CF, f, et EF, g; fiet FP, $p-x, f$ aequ. $\sqrt{cc+pp-2px+xx}$ seu compendio $\sqrt{l}$, et g aequ. $\sqrt{ee+xx}$ seu compendio $\sqrt{m}$. Habemus ergo ω aequ. $h\sqrt{l}+r\sqrt{m}$, cujus aequationis aequatio differentialis (posito $d\omega$ esse 0, in casu minimae) est 0 aequ. $+hdl:2\sqrt{l}+rdm:2\sqrt{m}$ per regulas calculi nostri traditas; jam dl est $-2dx\overline{p-x}$, et dm est $2xdx$, ergo fit: $h\overline{p-x}:f$ aequ. $rx:g$. Quodsi jam haec accommodentur ad dioptricam, et ponantur f et g seu CF et EF aequales, quia eadem manet refractio in puncto F, quantacunque ponatur longitudo rectae CF, fiet $h\overline{p-x}$ aequ. rx, seu $h:r::x:p-x$, seu h ad r ut QF ad FP, hoc est sinus angulorum incidentiae et refractionis FP et QF erunt reciproce ut r et h, densitates mediorum, in quibus fit incidentia et refractio. Quae densitas tamen non respectu nostri, sed respectu resistentiae quam radii lucis faciunt, intelligenda est. Et habetur ita demonstratio calculi, alibi a nobis in his ipsis Actis exhibiti, quando generale Opticae, Catoptricae et Dioptricae fundamentum exponebamus, cum alii doctissimi Viri multis ambagibus venati sint quae hujus calculi peritus tribus lineis imposterum praestabit. Quod alio adhuc exemplo docebo. Sit curva 133 (fig.113) talis naturae: ut a puncto ejus quocunque ut 3 ductae ad sex puncta fixa in axe posita 4, 5, 6, 7, 8, 9, sex rectae 34, 35, 36, 37, 38, 39 simul additae, sint rectae datae g aequales. Sit axis T 14526789, et 12 sit abscissa, 23 ordinata, quaeritur tangens 3 T; dico fore T 2 ad 23 ut

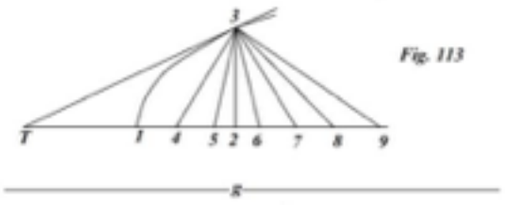


Fig. 113

$$\frac{23}{34}+\frac{23}{35}+\frac{23}{36}+\frac{23}{37}+\frac{23}{38}+\frac{23}{39} \text{ est ad } -\frac{24}{34}-\frac{25}{35}+\frac{26}{36}+\frac{27}{37}+\frac{28}{38}+\frac{29}{39}.$$

Eademque erit regula, continuatis tantum terminis, si non sex, sed decem, vel plura puncta fixa supponerentur, qualia secundum methodos tangentium editas calculo praestare sublatis irrationalibus, taediosissimae et aliquando insuperabilis operae foret, ut si rectangula plana vel solida secundum omnes biniones vel terniones possibiles ex rectis illis composita datae quantitati aequari deberent, in quibus omnibus, et multo implicatioribus, methodi nostrae eadem est opinione multo major rarissimique exempli facilitas. Et haec quidem initia sunt tantum Geometriae cujusdam multo sublimioris, ad difficillima et pulcherrima quaeque etiam mistae Matheseos problemata pertingentis, quae sine calculo nostro differentiali, aut simili, non temere quisquam pari facilitale tractabit. Appendicis loco placet adjicere solutionem Problematis, quod *Cartesius* a *Beaunio* sibi propositum Tom. 3. Epist. tentavit, sed non solvit : Lineam invenire WW talis naturae, ut ducta ad axem tangente WC, sit XC semper aequalis eidem rectae constanti a. Jam XW seu w ad XC seu a, ut dw ad dx; ergo si dx (quae assumi potest pro arbitrio) assumatur constans sive semper eadem, nempe b, seu si ipsae x sive AX crescant

Callbacks

Async

XMLHttpRequest

```
xhr.open('get', '/json-route', true)

xhr.onreadystatechange = function() {
  var status
  if (xhr.readyState == 4) {
    console.log(JSON.parse(xhr.responseText))
  }
}

xhr.send()
```

Fetch

```
fetch('/json-route').then(function(response) {
  console.log(response.json())
})
```

Что нас удерживает?

– Новый синтаксис
– Нужен Babel

Async/await скоро будут доступны нативно в nodejs

Question: New project on v1 v ×

Default Profile

GitHub, Inc. [US] https://github.com/koajs/koa/issues/826

This repository Search Pull requests Issues Gist

koajs / koa

Watch 671 Star 12,853 Fork 1,121

Code Issues 15 Pull requests 12 Projects 0 Wiki Pulse Graphs

Question: New project on v1 vs. v2 #826

New issue

Closed **hrmoller** opened this issue on 30 Sep · 3 comments

hrmoller commented on 30 Sep

Hi,

We're about to start up a new project and we're a little unsure whether we should choose to go with v1 versus v2-alpha.

It's a relatively small project which will hit our production lines within this year. It's our first project based on koa but we have several projects running on Express already.

Our main concern is whether v2 will be held up to date against v1. I.e. if a new feature is implemented in v1 will it then also be in v2?

In other parts of our project we're expecting to be use `async/await` why our babel setup will be able to handle that syntax no matter if we're on v1 or v2 anyways.

Any advise would be appreciated.

hrmoller changed the title from **v1 vs. v2** to **Question: New project on v1 vs. v2** on 30 Sep

Projects

None yet

Labels

None yet

Milestone

No milestone

Assignees

No one assigned

3 participants

Notifications

План

No middleware

Callbacks

Generators

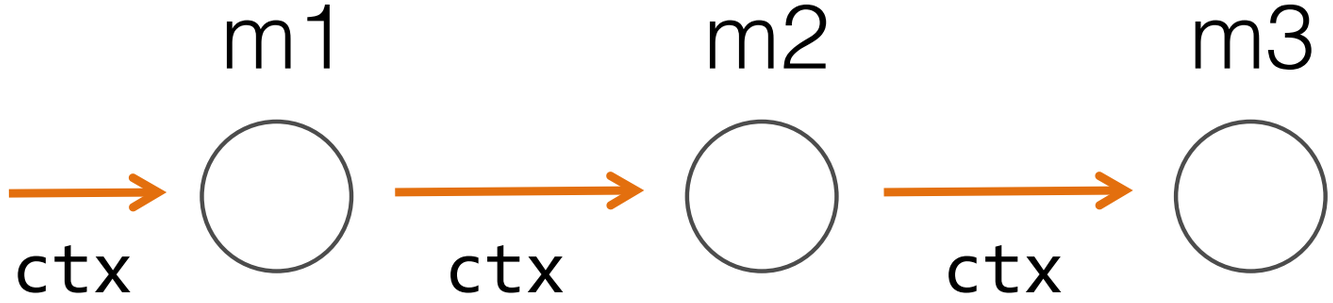
Async/await

http сервер

Plain Node

```
var http = require('http')

var server = http.createServer(function (request, response) {
  response.writeHead(200)
  response.write('whatever')
  response.end()
})

server.listen(3000)
```

Идея middleware




```
ctx = {request, response}
```

Каскад middleware

Каскад middleware – последовательность функций, связанных вместе. Каждая функция имеет доступ к объекту ctx и имеет возможность передать управление следующей по порядку функции.

Callbacks

Каскад middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function (ctx, next) {
  // do some logic
  next()
})
```

```
middleware.push(function (ctx, next) {
  // do some another logic
  next()
})
```

Комбинация sync и async

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function (ctx, next){
  somethingSync(ctx)
  next()
})
```

```
middleware.push(function (ctx, next){
  somethingAsync(function(err, asyncResult) {
    if (err) return next(err)
    ctx.response.someAsync = asyncResult
    next()
  })
})
```

Run middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function (ctx, next){
  somethingSync(ctx)
  next()
})
```

```
middleware.push(function (ctx, next){
  somethingAsync(function(err, asyncResult) {
    if (err) return next(err)
    ctx.response.someAsync = asyncResult
    next()
  })
})

run(ctx, middleware, done)
```

Run middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function (ctx, next){
  somethingSync(ctx)
  next()
})
```

```
middleware.push(function (ctx, next){
  somethingAsync(function(err, asyncResult) {
    if (err) return next(err)
    ctx.response.someAsync = asyncResult
    next()
  })
})

run(ctx, middleware, done)
```

Функция run

```
function run(ctx, middleware, done) {
  var index = 0

  function next() {
    var nextMiddleware = middleware[index++]
    if (!nextMiddleware) {
      done(ctx)
    }

    nextMiddleware(ctx, next)
  }

  next()
}
```

Плюсы

- Нет оверхеда
- Легко начать

Минусы

- Callback hell
- Обработка ошибок
- Сложность чтения

Generators

Koa v1 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function* (next) {
  this.output.asyncResult = yield funcThatReturnsPromise()
  yield next
})
```

Koa v1 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function* (next) {
  this.output.asyncResult = yield funcThatReturnsPromise()
  yield next
})
```

Koa v1 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function* (next) {
  this.output.asyncResult = yield funcThatReturnsPromise()
  yield next
})
```

ctx -> this

Koa v1 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function* (next) {
  this.output.asyncResult = yield funcThatReturnsPromise()
  yield next
})
```

Koa v1 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function* (next) {
  this.output.asyncResult = yield funcThatReturnsPromise()
  yield next
})
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}

let iter = gen()

iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
iter.next() // -> {done: false, value: 3}
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
iter.next() // -> {done: false, value: 3}
iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
iter.next() // -> {done: false, value: 3}
iter.next() // -> {done: true, value: 4}
```

Как работают генераторы

```
function* gen() {
  yield 1
  yield 2
  yield 3
  return 4
}
```

```
let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 2}
iter.next() // -> {done: false, value: 3}
iter.next() // -> {done: true, value: 4}
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next()
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53)
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53)
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53)
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53)
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53) // -> {done: false, value: 3}
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53) // -> {done: false, value: 3}
iter.next()
```

Как работают генераторы

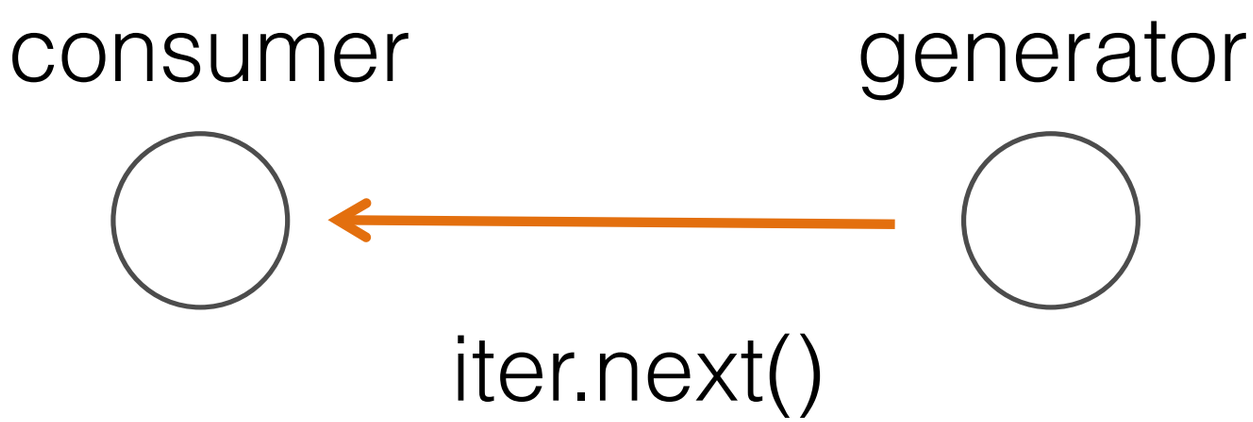
```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53) // -> {done: false, value: 3}
iter.next() // -> {done: true, value: 4}
```

Как работают генераторы

```
function* gen() {
  yield 1
  let a = yield 42
  console.log(a)
  yield 3
  return 4
}

let iter = gen()

iter.next() // -> {done: false, value: 1}
iter.next() // -> {done: false, value: 42}
iter.next(53) // -> {done: false, value: 3}
iter.next() // -> {done: true, value: 4}
```

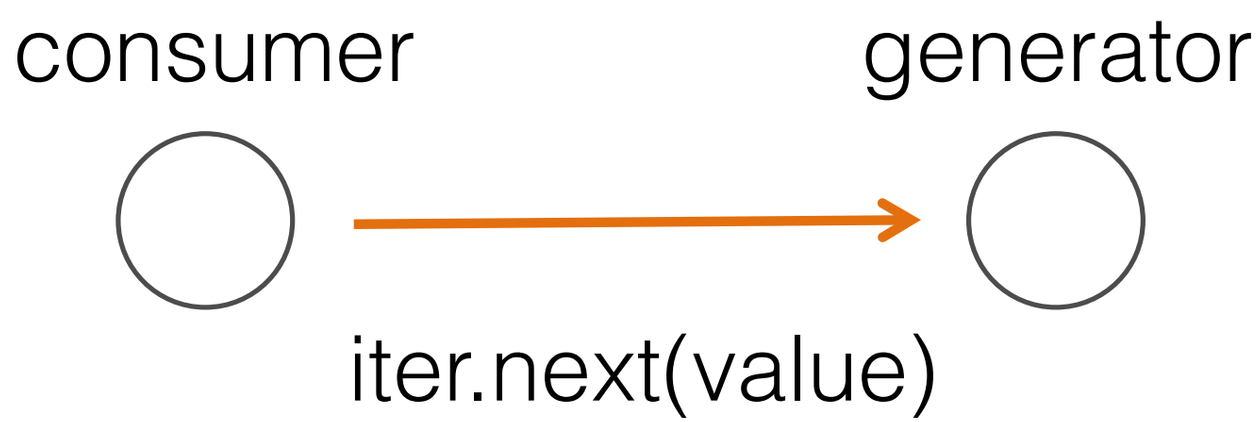
Генераторы

Генератор – функция, которая может быть приостановлена в середине и возобновлена позже. Пока функция генератор стоит на паузе, может быть исполнен другой код.



Генераторы

Генератор – функция, которая может быть приостановлена в середине и возобновлена позже. Пока функция генератор стоит на паузе, может быть исполнен другой код.

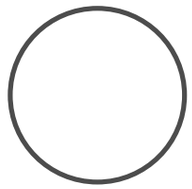


Генераторы

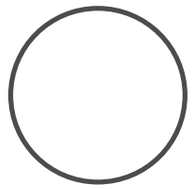
```
function* () {
  let val = yield Promise.resolve(42)
}
```

Генераторы

consumer



generator



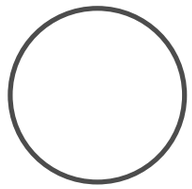
```
let iter = generator()
```

Генераторы


```
let iter = generator()

let state = iter.next() // -> {done: false, value: Promise.resove(42)}
```

Генераторы

consumer



generator

```
let iter = generator()

let state = iter.next() // -> {done: false, value: Promise.resove(42)}

state.value.then(function(result) {
   // нужно теперь передать этот результат внутрь
})
```

Генераторы


```
let iter = generator()

let state = iter.next() // -> {done: false, value: Promise.resove(42)}

state.value.then(function(result) {
  iter.next(result)
})
```

co

co – runner для генераторов, который автоматизирует процесс получения и передачи значений из генератора

`github.com/tj/co`

`co(generator) -> Promise`

co принимает генератор,
исполняет его до конца
и тогда **Promise** резолвится

Использование co

```
function* gen() {
  let a = yield someAsyncJob()
  let b = yield anotherAsyncJob(a)
  return b
}
```

```
co(gen).then(function (value) {
  // value это то b, которое вернет генератор
}, function (err) {
  // обрабатываем ошибки
})
```

Обработка ошибок с **co**

```
co(function *(){
  try {
    yield Promise.reject(new Error('boom'))
  } catch (err) {
    console.error(err.message)
  }
}).then(function (value) {
  // обрабатываем value
}, function (err) {
  // обрабатываем ошибку
})
```

Генераторы

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function *(next) {
  // какая-то логика
  yield next
})

middleware.push(function *(next) {
  // доступ к ctx через this
  this.response = ...
  yield next
})

run(ctx, middleware, done)
```

Генераторы

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function *(next) {
  // какая-то логика
  yield next
})

middleware.push(function *(next) {
  // доступ к ctx через this
  this.response = ...
  yield next
})

run(ctx, middleware, done)
```

Генераторы

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(function *(next) {
  // какая-то логика
  yield next
})

middleware.push(function *(next) {
  // доступ к ctx через this
  this.response = ...
  yield next
})

run(ctx, middleware, done)
```

Run

```
function run(ctx, middleware, done) {
  co(compose(middleware).call(ctx)).then(function (value) {
    done(value)
  }, function (err) {
    // если вы забыли обработать исключение, то co перехватит
    console.error(err.stack)
  })
}
```

Run

```
function run(ctx, middleware, done) {
  co(compose(middleware).call(ctx)).then(function (value) {
    done(value)
  }, function (err) {
    // если вы забыли обработать исключение, то co перехватит
    console.error(err.stack)
  })
}
```

Run

```
function run(ctx, middleware, done) {
  co(compose(middleware).call(ctx)).then(function (value) {
    done(value)
  }, function (err) {
    // если вы забыли обработать исключение, то co перехватит
    console.error(err.stack)
  })
}
```

Compose

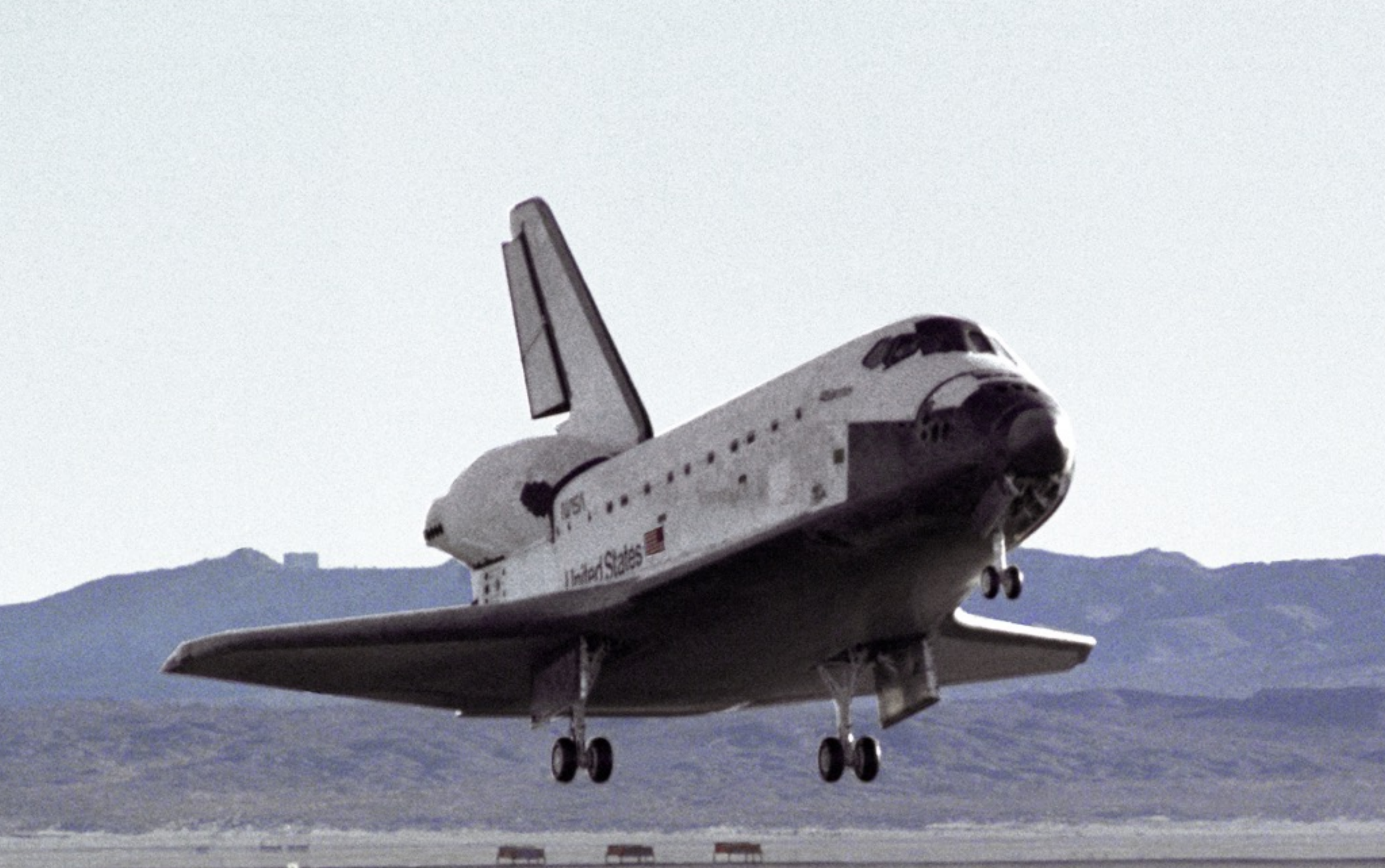
```
function compose(middleware) {
  return function *() {
    var i = middleware.length
    var prev = noop()
    var curr

    while (i--) {
      curr = middleware[i]
      // передаем ctx через this
      prev = curr.call(this, prev)
    }

    yield prev
  }
}

function* noop() {}
```

Плюсы

- Линейный код
- Ни одного не обработанного исключения

Минусы

- Сложно понять, как это работает



async/await

Koa v2 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  ctx.response.asyncResult = await somethingAsync()
  await next()
})
```

Koa v2 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  ctx.response.asyncResult = await somethingAsync()
  await next()
})
```

Koa v2 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  ctx.response.asyncResult = await somethingAsync()
  await next()
})
```

Koa v2 middleware

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  ctx.response.asyncResult = await somethingAsync()
  await next()
})
```

Async/await

```
async function func() {
  let result = await Promise.resolve(42)
}

func()
```

Async/await

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  // some logic
  await next()
})

middleware.push(async (ctx, next) => {
  // some another logic
  await next()
})

run(ctx, middleware, done)
```

Async/await

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  // some logic
  await next()
})

middleware.push(async (ctx, next) => {
  // some another logic
  await next()
})

run(ctx, middleware, done)
```

Async/await

```
let middleware = []
let ctx = {request: {…}, response: {…}}

middleware.push(async (ctx, next) => {
  // some logic
  await next()
})

middleware.push(async (ctx, next) => {
  // some another logic
  await next()
})

run(ctx, middleware, done)
```

Run

```
function run(ctx, middleware, done) {
  compose(middleware)(ctx).then(function (value) {
    done(value)
  }, function (err) {
    // обрабатываем ошибку
  })
}
```

Run

```
function run(ctx, middleware, done) {
  compose(middleware)(ctx).then(function (value) {
    done(value)
  }, function (err) {
    // обрабатываем ошибку
  })
}
```

Run

```
function run(ctx, middleware, done) {
  compose(middleware)(ctx).then(function (value) {
    done(value)
  }, function (err) {
    // обрабатываем ошибку
  })
}
```

Compose

```
function compose(middleware) {
  return function (ctx, next) {

    return dispatch(0)

    function dispatch(i) {
      const fn = middleware[i]
      try {
        return Promise.resolve(fn(ctx, function next() {
          return dispatch(i + 1)
        }))
      } catch(err) {
        return Promise.reject(err)
      }
    }
  }
}
```

Плюсы

- Линейный код
- Семантичный код
- Без генераторов!

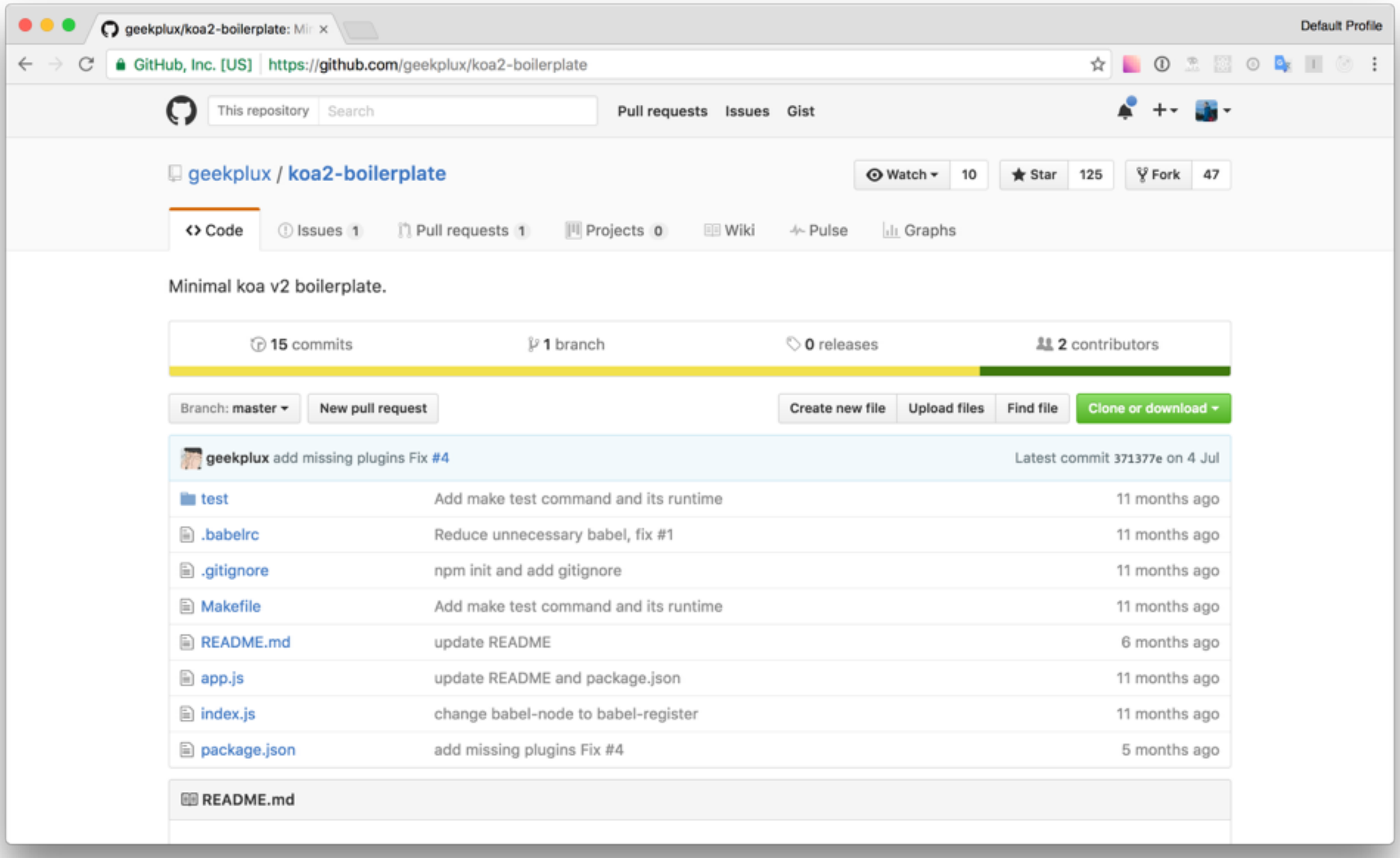
Минусы

- Promise overhead
- ~~Babel~~

Koa v2 almost ready

Нативные async/await появятся в nodejs вместе с апгрейдом на *v8 5.5*

github.com/geekplux/koa2-boilerplate



Koa vs Express

Feature	Koa	Express	Connect
Middleware Kernel	✓	✓	✓
Routing		✓	
Templating		✓	
Sending Files		✓	
JSONP		✓	

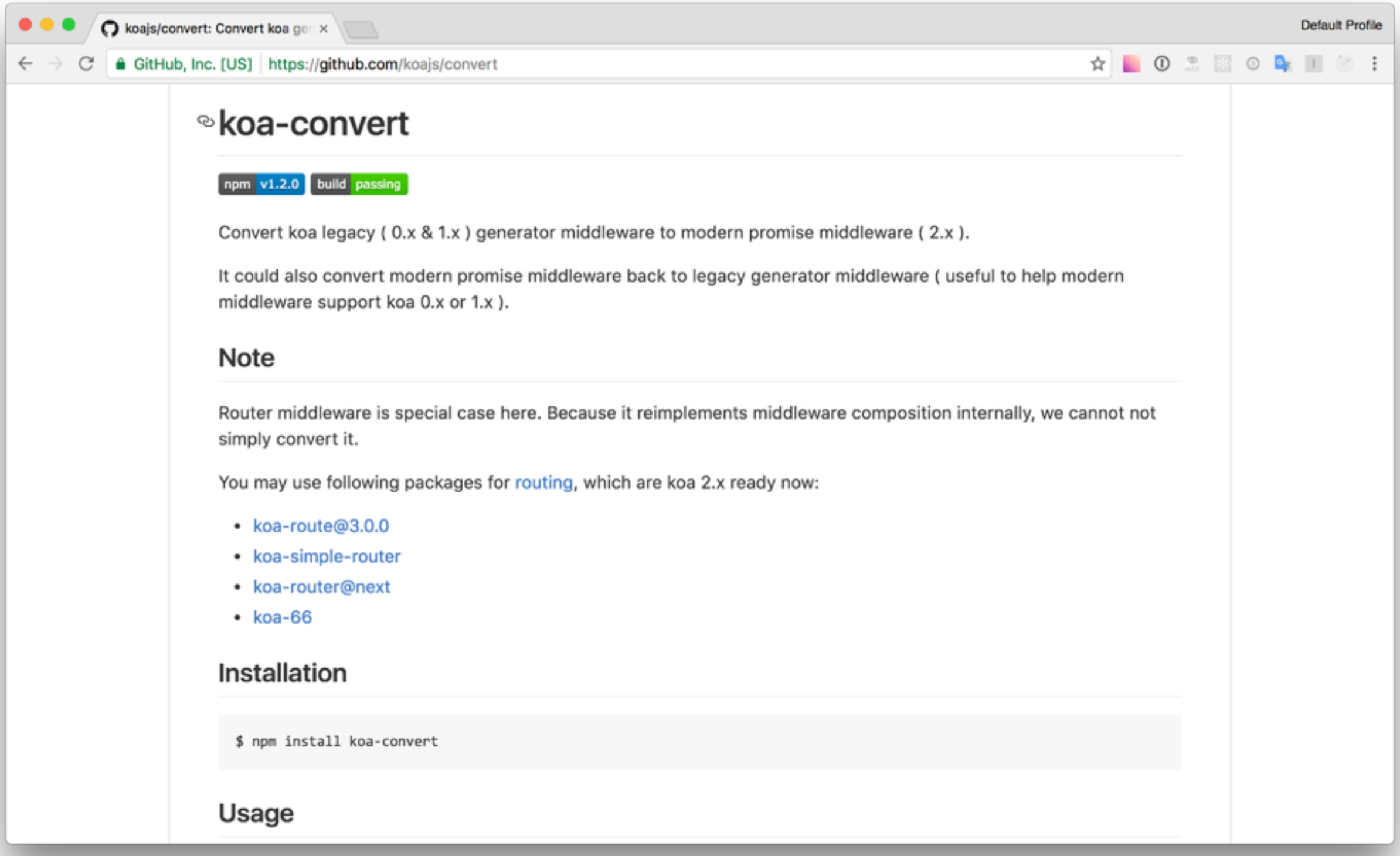
Middleware

Known middleware for Koa, you may want to search npm with "koa" to find more.

Koa v2 will try to convert legacy middleware for you, thus many in the lists below might just work out of the box. See Readme.md for more information about legacy middleware usage in Koa v2.

Name and description	Supports v2	Downloads
koa-exception - Exception Handler Middleware Base on Koa@1.x		downloads 59/month
koa-webpack - Development and Hot Module Reload Middleware for Koa2. Simple setup and use. Composes webpack-dev-middleware and webpack-hot-middleware.	✓	downloads 664/month
koa-webpack-middleware - webpack-dev-middleware for koa2 with HMR (Hot Module Replacement).	✓	downloads 2k/month
koa-hello-world - Koa 'Hello World' middleware, useful for testing		downloads 22/month
grant - OAuth middleware		downloads 3k/month
koa-cors - CORS middleware		downloads 34k/month
koa-slow - delay answering requests by URL RegExp, useful for debugging.	✓	downloads 101/month
koa-force-ssl - Middleware for force SSL		downloads 257/month

github.com/koajs/convert



Koa-convert

```
import Koa from 'koa'
import convert from 'koa-convert'
const app = new Koa()

app.use(convert(function* (next) {
  this.set('Access-Control-Allow-Origin', '*')
  yield next
}))

app.use(async (ctx, next) => {
  let result = await makeSomething(ctx)
  ctx.body = result
  await next()
})
```

generators vs. async

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}
```

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}

let iter = saga()
```

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}

let iter = saga()
iter.next()
```

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}

let iter = saga()
iter.next() // -> {CALL: {fn: findMeaningOfLife, args: []}}
```

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}

let iter = saga()
iter.next() // -> {CALL: {fn: findMeaningOfLife, args: []}}
iter.next(53)
```

Redux-saga

```
function findMeaningOfLife() {
  return Promise.resolve(42)
}

function* saga() {
  let result = yield call(findMeaningOfLife)
  yield put({type: 'RESULT', result: result})
}

let iter = saga()
iter.next() // -> {CALL: {fn: findMeaningOfLife, args: []}}
iter.next(53) // -> {type: 'RESULT', result: 53}
```

Cancellable

```
async function start() {
  await doSomething()
  await doSomethingElse()
  await doOtherThing()
}
```

```
async function start(getIsCancelled) {
  await doSomething()

  if (getIsCancelled) {
    return
  }

  await doSomethingElse()

  if (getIsCancelled) {
    return
  }

  await doOtherThing()
}
```

```
function* processPDF(pdf, directory) {
  const imagePaths = yield extractImagesFromPDF(pdf, directory)
  for (let imagePath of imagePaths) {
    yield generateThumbnail(imagePath)
  }
}

function extractImagesFromPDF(pdf, dir) {
  return {bin: '/usr/bin/images-extractor', args: [pdf, dir]}
}

function generateThumbnail(imagePath) {
  return {bin: '/usr/bin/thumbnail-generator', args: [imagePath]}
}

const {cancel, promise} = run(processPDF,
  '/path/to/file.pdf', '/path/to/out/dir')
```

```
function* processPDF(pdf, directory) {
  const imagePaths = yield extractImagesFromPDF(pdf, directory)
  for (let imagePath of imagePaths) {
    yield generateThumbnail(imagePath)
  }
}

function extractImagesFromPDF(pdf, dir) {
  return {bin: '/usr/bin/images-extractor', args: [pdf, dir]}
}

function generateThumbnail(imagePath) {
  return {bin: '/usr/bin/thumbnail-generator', args: [imagePath]}
}

const {cancel, promise} = run(processPDF,
  '/path/to/file.pdf', '/path/to/out/dir')
```

```
function* processPDF(pdf, directory) {
  const imagePaths = yield extractImagesFromPDF(pdf, directory)
  for (let imagePath of imagePaths) {
    yield generateThumbnail(imagePath)
  }
}

function extractImagesFromPDF(pdf, dir) {
  return {bin: '/usr/bin/images-extractor', args: [pdf, dir]}
}

function generateThumbnail(imagePath) {
  return {bin: '/usr/bin/thumbnail-generator', args: [imagePath]}
}

const {cancel, promise} = run(processPDF,
  '/path/to/file.pdf', '/path/to/out/dir')

cancel()
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function run(gen, ...args) {
  let cancel

  const promise = new Promise((resolve, reject) => {
    const iter = gen(...args)
    let activePid = null

    function step(value) {
      const state = iter.next(value)
      if (state.done) {
        return resolve(state.value)
      }
      const cmd = state.value // { bin: '...', args: [...] }
      const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
      promise.then(step)
      activePid = pid
    }

    cancel = callOnce(() => {
      iter.return()
      killProcess(activePid)
    })

    step(undefined)
  })

  return {cancel, promise}
}
```

```
function step(value, err) {
  let state
  if (err) {
    try {
      state = iter.throw(err)
    } catch (err) {
      return reject(err)
    }
  } else {
    state = iter.next(value)
  }
  if (state.done) {
    return resolve(state.value)
  }
  const cmd = state.value // { bin: '...', args: [...] }
  const {pid, promise} = spawnProcess(cmd.bin, cmd.args)
  promise.then(step).catch(err => step(null, err))
  activePid = pid
}
```

Мы живем в мире,
где есть выбор!

Евгений Пшеничный

codehipsters.com