Введение в Effect



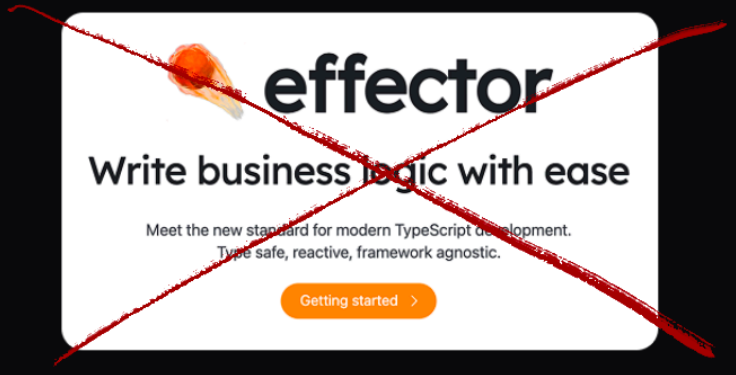
Как писать на TypeScript и не страдать

24 октября 2026 • HolyJS • Владислав Дерябкин

Что такое Effect?

Что такое Effect?

Не effector



Что такое Effect?

Не effector

Экосистема модулей

~/my-app

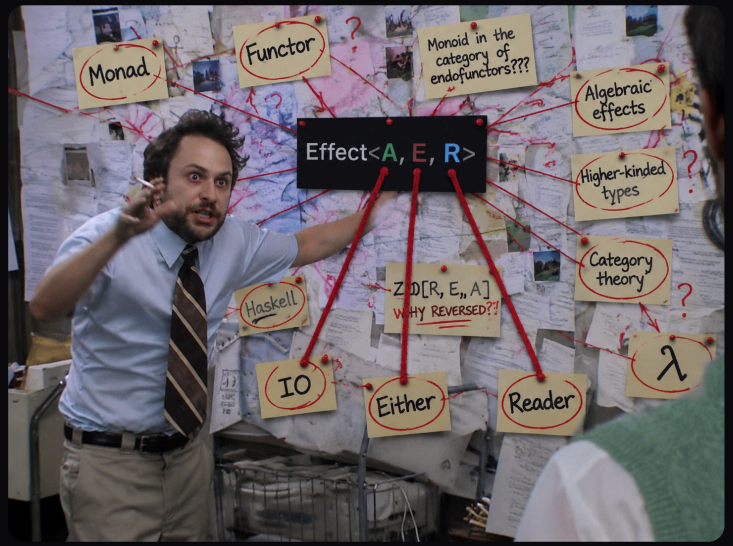
```
$ npm install effect
$ npm uninstall zod yup valibot arktype @sinclair/typebox  # effect/Schema
$ npm uninstall pino winston bunyan loglevel consola  # effect/Logger
$ npm uninstall commander yargs @stricli/core @oclif/core  # effect/cli
$ npm uninstall rxjs  # effect/Stream
$ # Ai, Atom, Config, FileSystem, Schedule, Redacted, Duration, …
```

Что такое Effect?

Не effector

Экосистема модулей

Система эффектов

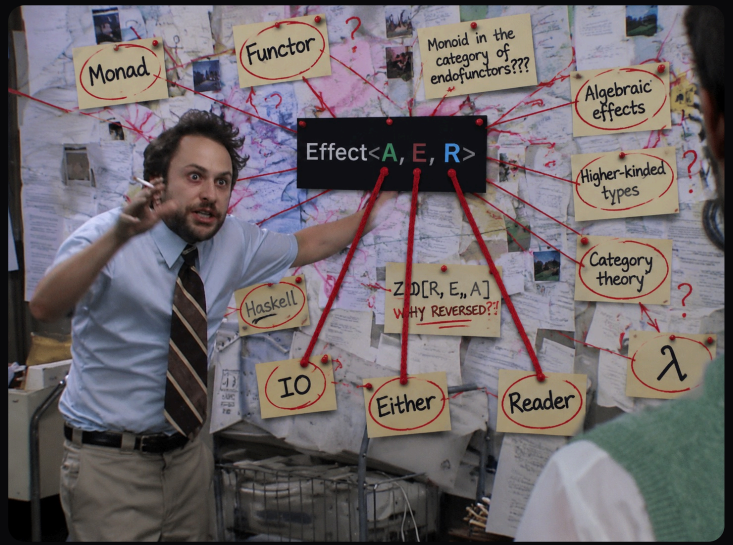


Что такое Effect?

Не effector

Экосистема модулей

Система эффектов



О себе

О себе

- 2016–2021 Робототехника и фриланс

О себе

- 2016–2021 Робототехника и фриланс
- 2021–2025 Бакалавриат (Computer Science)

О себе

- `2016–2021` Робототехника и фриланс
- `2021–2025` Бакалавриат (Computer Science)
  - Type-checker на Haskell

О себе

- 2016–2021 Робототехника и фриланс
- 2021–2025 Бакалавриат (Computer Science)
  - Type-checker на Haskell
- 2025–н.в. React-разработчик в Т-Банке

О себе

- 2016–2021 Робототехника и фриланс
- 2021–2025 Бакалавриат (Computer Science)
  - Type-checker на Haskell
- 2025–н.в. React-разработчик в Т-Банке
- 2026–н.в. Пробую Effect и адопчусь к ИИ

О себе

- `2016–2021` Робототехника и фриланс
- `2021–2025` Бакалавриат (Computer Science)
  - Type-checker на Haskell
- `2025–н.в.` React-разработчик в Т-Банке
- `2026–н.в.` Пробую Effect и адопчусь к ИИ
  - effect-tg

О себе

- `2016–2021` Робототехника и фриланс
- `2021–2025` Бакалавриат (Computer Science)
  - Type-checker на Haskell
- `2025–н.в.` React-разработчик в Т-Банке
- `2026–н.в.` Пробую Effect и адопчусь к ИИ
  - effect-tg
  - Harness (CLI, MCP, ...)

О себе

- 2016–2021 Робототехника и фриланс
- 2021–2025 Бакалавриат (Computer Science)
  - Type-checker на Haskell
- 2025–н.в. React-разработчик в Т-Банке
- 2026–н.в. Пробую Effect и адопчусь к ИИ
  - effect-tg
  - Harness (CLI, MCP, ...)
- Дисклеймер: Не продаю Effect

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

Что не так с TypeScript?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция?
Что, если изменится реализация?

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                    ▲
  //                    └── (id: string) => Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) ⇒ Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) => Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                    ▲
  //                    └── (id: string) => Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) ⇒ Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) => Promise<User>
}
```

```ts
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) => Promise<User>
}
```

```ts
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
async function doStuff() {
  // ...
  const user = await fetchUser('admin')
  //                 ▲
  //                 └── (id: string) ⇒ Promise<User>
}
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

Может скоро TypeScript поможет?

Suggestion: `throws` clause and typed catch clause #13219

Closed as not planned · microsoft/work... #86

Nitzan Tomer (nitzantomer) opened on Dec 29, 2016

The typescript type system is helpful in most cases, but it can't be utilized when handling exceptions.

Ryan Cavanaugh (RyanCavanaugh) on Apr 19, 2023 — Last edited by RyanCavanaugh — Member

After reviewing all the comments here over the years and much discussion internally, we don't think that the JavaScript runtime or overall ecosystem provide a platform on which to build this feature in a way that would meet user expectations. Per popular request to either add the feature or close this issue for clarity, we're opting for the latter. As with Minification (#8), we're implementing a two-week cool-down period on further comments (ends 5/3).

TL;DR: нет (microsoft/TypeScript#13219)

Решение

Save The Flow

- ✗ predictability
- ✗ readability
- ✗ tight coupling
- ✗ hard to debug/test/optimize
- ✗ ...

```ts
/**
 * @throws {MyError}
 */
const foo = () => {
  a();
  if (somethingBadHappened) {
    throw new MyError()
  }
  b();
}

const bar = () => {
  c();
  try {
    foo();
  } catch (error: unknown) {
    if (error instanceof MyError) {
      handleMyError(error)
    }
  }
  d();
}
```

14



Алексей Тушинский

tushinski

HOLY JS/... × JUG.RU group

ГРИНАТОМ РОСАТОМ

OpenIDE

HolyJS 2025 Autumn, Безупречная обработка ошибок в TypeScript

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```
type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { Result } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { Result, err } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return err('NotOk')

  const json = await resp.json()
  return json as User
}
```

```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return err('NotOk')

  const json = await resp.json()
  return ok(json as User)
}
```

```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetchResult(`${base}/users/${id}`)
  if (resp.isErr())
    return err('Network')

  if (!resp.value.ok)
    return err('NotOk')

  const json = await jsonResult(resp.value)
  if (json.isErr())
    return err('Decode')

  return ok(json.value as User)
}
```

```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

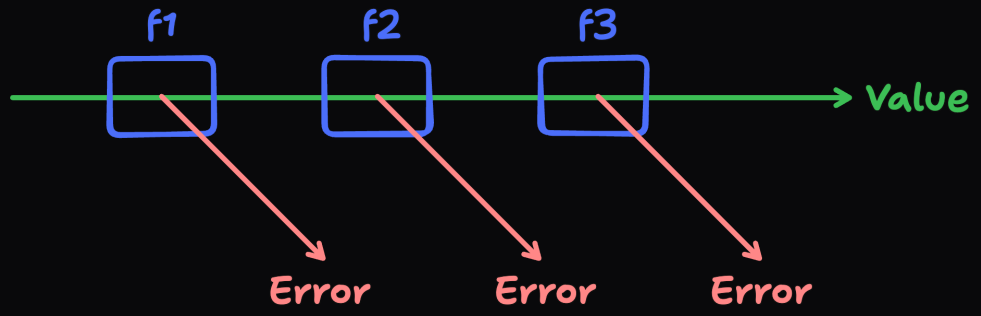
  const resp = await fetchResult(`${base}/users/${id}`)
  if (resp.isErr())
    return err('Network')

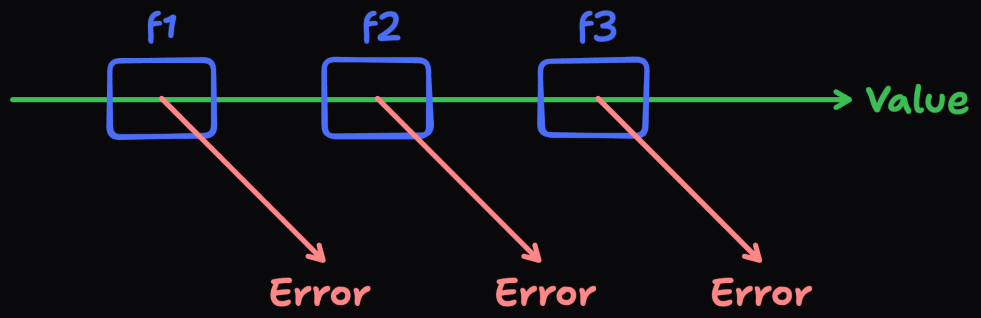
  if (!resp.value.ok)
    return err('NotOk')

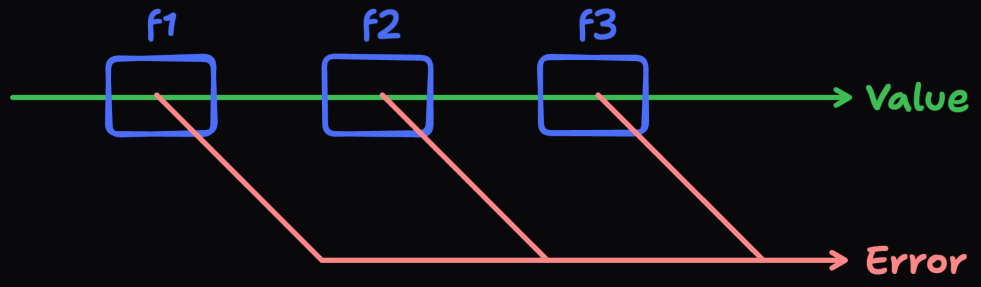
  const json = await jsonResult(resp.value)
  if (json.isErr())
    return err('Decode')

  return ok(json.value as User)
}
```




f1
f2
f3
Value
Error
Error
Error


f1
f2
f3
Value
Error
Error
Error


f1
f2
f3
Value
Error

```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetchResult(`${base}/users/${id}`)
  if (resp.isErr())
    return err('Network')

  if (!resp.value.ok)
    return err('NotOk')

  const json = await jsonResult(resp.value)
  if (json.isErr())
    return err('Decode')

  return ok(json.value as User)
}
```

```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetchResult(`${base}/users/${id}`)
  if (resp.isErr())
    return err('Network')

  if (!resp.value.ok)
    return err('NotOk')

  const json = await jsonResult(resp.value)
  if (json.isErr())
    return err('Decode')

  return ok(json.value as User)
}
```



```ts
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

async function fetchUser(id: string): Promise<Result<User, UserError>> {
  const base = process.env.API_URL
  if (!base)
    return err('NoBase')

  const resp = await fetchResult(`${base}/users/${id}`)
  if (resp.isErr())
    return err('Network')

  if (!resp.value.ok)
    return err('NotOk')

  const json = await jsonResult(resp.value)
  if (json.isErr())
    return err('Decode')

  return ok(json.value as User)
}
```

```ts
import { ResultAsync, err, ok, safeTry } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

function fetchUser(id: string): ResultAsync<User, UserError> {
  return safeTry(async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)

    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  })
}
```

```ts
import { ResultAsync, err, ok, safeTry } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

function fetchUser(id: string): ResultAsync<User, UserError> {
  return safeTry(async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)

    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  })
}
```

Генераторы

function* и yield
Синтаксис ES2015 (ES6)

```
function* greet() {
  const name = yield 'Как тебя зовут?'
  return `Привет, ${name}!`
}
```

Генераторы

function* и yield
Синтаксис ES2015 (ES6)

```
function* greet() {
  const name = yield 'Как тебя зовут?'
  return `Привет, ${name}!`
}

const gen = greet() // пока ничего не выполнилось
```

Генераторы

function* и yield
Синтаксис ES2015 (ES6)

Пауза
Функция выполняется до `yield` и ждёт следующего `next()`

```
function* greet() {
  const name = yield 'Как тебя зовут?'
  return `Привет, ${name}!`
}

const gen = greet() // пока ничего не выполнилось

gen.next()
// { value: 'Как тебя зовут?', done: false }
```

Генераторы

function* и yield
Синтаксис ES2015 (ES6)

Пауза
Функция выполняется до `yield` и ждёт следующего `next()`

Inversion of Control
Функция спрашивает, а отвечает тот, кто её запускает

```
function* greet() {
  const name = yield 'Как тебя зовут?'
  return `Привет, ${name}!`
}

const gen = greet() // пока ничего не выполнилось

gen.next()
// { value: 'Как тебя зовут?', done: false }

gen.next('Влад')
// { value: 'Привет, Влад!', done: true }
```

Генераторы

function* и yield
Синтаксис ES2015 (ES6)

Пауза
Функция выполняется до `yield` и ждёт следующего `next()`

Inversion of Control
Функция спрашивает, а отвечает тот, кто её запускает

Но без типов
Что придёт в ответ на `yield`, TypeScript не знает

```
function* greet() {
  const name = yield 'Как тебя зовут?'
  //     └── any
  return `Привет, ${name}!`
}

const gen = greet() // пока ничего не выполнилось

gen.next()
// { value: 'Как тебя зовут?', done: false }

gen.next('Влад')
// { value: 'Привет, Влад!', done: true }
```

Inversion of Control — с типами

```
function* greet() {
  const name = yield 'name'
  const age = yield 'age'
  return `${name}, ${age}`
}
```

Inversion of Control — с типами

```
function* greet() {
  const name = yield* ask<string>('name')
  const age = yield* ask<number>('age')
  return `${name}, ${age}`
}
```

Inversion of Control — с типами

```
function* greet() {
  const name = yield* ask<string>('name')
  const age = yield* ask<number>('age')
  return `${name}, ${age}`
}
```

```
class Ask<T> {
  constructor(readonly key: string) {}
  *[Symbol.iterator](): Generator<Ask<T>, T> {
    return yield this
  }
}
const ask = <T>(key: string) ⇒ new Ask<T>(key)
```

Inversion of Control — с типами

```
function* greet() {
  const name = yield* ask<string>('name')
  //      └── string
  const age = yield* ask<number>('age')
  //      └── number
  return `${name}, ${age}`
}
// Generator<Ask<string> | Ask<number>, string>
```

```
class Ask<T> {
  constructor(readonly key: string) {}
  *[Symbol.iterator](): Generator<Ask<T>, T> {
    return yield this
  }
}
const ask = <T>(key: string) ⇒ new Ask<T>(key)
```

Inversion of Control — с типами

```ts
function* greet() {
  const name = yield* ask<string>('name')
  //      └── string
  const age = yield* ask<number>('age')
  //      └── number
  return `${name}, ${age}`
}
// Generator<Ask<string> | Ask<number>, string>
```

```ts
class Ask<T> {
  constructor(readonly key: string) {}
  *[Symbol.iterator](): Generator<Ask<T>, T> {
    return yield this
  }
}
const ask = <T>(key: string) => new Ask<T>(key)
```

```ts
function run<T>(
  gen: Generator<Ask<any>, T>,
  answers: Record<string, any>,
): T {
  let step = gen.next()
  while (!step.done)
    step = gen.next(answers[step.value.key])
  return step.value
}

run(greet(), { name: 'Влад', age: 23 })
// 'Влад, 23'
```

```ts
import { ResultAsync, err, ok, safeTry } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

function fetchUser(id: string): ResultAsync<User, UserError> {
  return safeTry(async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)

    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  })
}
```

```ts
import { ResultAsync, err, ok, safeTry } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'NotOk' | 'Decode'

function fetchUser(id: string): ResultAsync<User, UserError> {
  return safeTry(async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)

    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  })
}
```

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция?
Что, если изменится реализация?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция?
Что, если изменится реализация?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Dependency Injection

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  const resp = await fetch(`${base}/users/${id}`)
  return await resp.json()
}
```

Dependency Injection

Аргументы

+ просто, type-safe, без магии

```
type Context = { http: HttpClient, config: Config }

async function fetchUser(ctx: Context, id: string) {
  const base = ctx.config.apiUrl
  const resp = await ctx.http.get(`${base}/users/${id}`)
  return await resp.json()
}
```

Dependency Injection

Аргументы

+ просто, type-safe, без магии
– arg-drilling



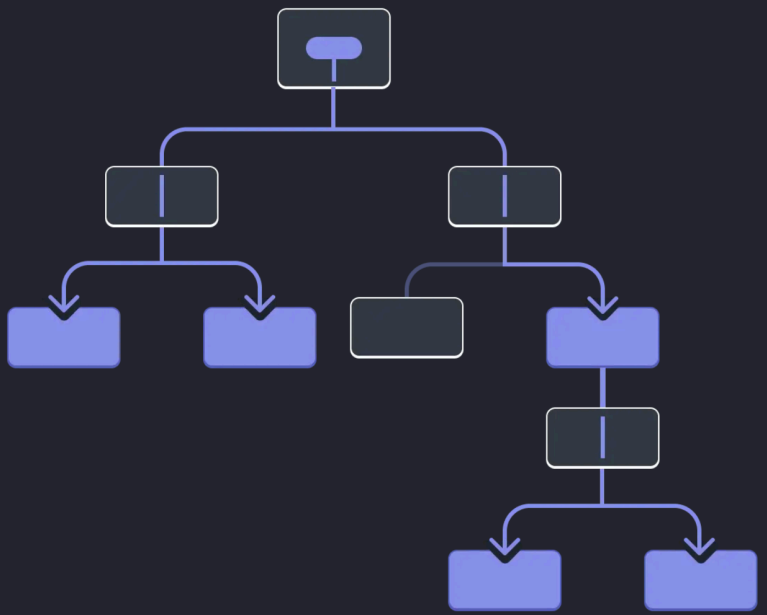
```
type Context = { http: HttpClient, config: Config }

async function fetchUser(ctx: Context, id: string) {
  const base = ctx.config.apiUrl
  const resp = await ctx.http.get(`${base}/users/${id}`)
  return await resp.json()
}

async function getProfile(ctx: Context, id: string) {
  const user = await fetchUser(ctx, id)
  // ...
}

async function handleRequest(ctx: Context, req: Request) {
  const profile = await getProfile(ctx, req.params.id)
  // ...
}
```

Prop drilling



react.dev: Passing Data Deeply with Context

Dependency Injection

Аргументы

+ просто, type-safe, без магии
− arg-drilling

```
type Context = { http: HttpClient, config: Config }

async function fetchUser(ctx: Context, id: string) {
  const base = ctx.config.apiUrl
  const resp = await ctx.http.get(`${base}/users/${id}`)
  return await resp.json()
}

async function getProfile(ctx: Context, id: string) {
  const user = await fetchUser(ctx, id)
  // ...
}

async function handleRequest(ctx: Context, req: Request) {
  const profile = await getProfile(ctx, req.params.id)
  // ...
}
```

Dependency Injection

Аргументы

+ просто, type-safe, без магии
− arg-drilling

Глобальные переменные

+ ничего не надо передавать
− неявные зависимости, ...

```ts
import { config } from './config'
import { http } from './http'

async function fetchUser(id: string) {
  const base = config.apiUrl
  const resp = await http.get(`${base}/users/${id}`)
  return await resp.json()
}

// fetchUser.test.ts
vi.mock('./http', () => ({
  http: { get: vi.fn() },
}))
```

Dependency Injection

Аргументы
+ просто, type-safe, без магии
– arg-drilling

Глобальные переменные
+ ничего не надо передавать
– неявные зависимости, ...

Service locator
+ легко подменить
– ошибки — только в рантайме

```
async function fetchUser(container: Di, id: string) {
  const config = container.get<Config>('config')
  const http = container.get<HttpClient>('http')

  const base = config.apiUrl
  const resp = await http.get(`${base}/users/${id}`)
  return await resp.json()
}

// Error: No provider for 'http'
```

Dependency Injection

Аргументы
+ просто, type-safe, без магии
– arg-drilling

Глобальные переменные
+ ничего не надо передавать
– неявные зависимости, ...

Service locator
+ легко подменить
– ошибки — только в рантайме

Декораторы
+ всё связывает контейнер
– магия, ошибки в рантайме

```
@Injectable()
class UserService {
  constructor(
    @Inject(CONFIG) private config: Config,
    private http: HttpClient,
  ) {}

  async fetchUser(id: string) {
    const base = this.config.apiUrl
    const resp = await this.http.get(`${base}/users/${id}`)
    return await resp.json()
  }
}

// Nest can't resolve dependencies of the UserService (CONFIG, ?)
```

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Interruptions

```
async function fetchUser(id: string): Promise<User> {
  const resp = await fetch(`/users/${id}`)
  return await resp.json()
}
```

Interruptions

Promise нельзя отменить
проигравшие в `Promise.race` сами не остановятся

```
async function fetchUser(id: string): Promise<User> {
  const resp = await fetch(`/users/${id}`)
  return await resp.json()
}

const user = await Promise.race([fetchUser('admin'), timeout(2000)])
```

Interruptions

Promise нельзя отменить
проигравшие в `Promise.race` сами не остановятся

AbortSignal — вручную
arg-drilling; легко забыть

```ts
async function fetchUser(id: string, signal?: AbortSignal) {
  const resp = await fetch(`/users/${id}`, { signal })
  return await resp.json()
}

const user = await fetchUser('admin', AbortSignal.timeout(2000))
```

Interruptions

Promise нельзя отменить
проигравшие в `Promise.race` сами не остановятся

AbortSignal — вручную
arg-drilling; легко забыть

```
async function fetchUser(id: string, signal?: AbortSignal) {
  const resp = await fetch(`/users/${id}`, { signal })
  return await resp.json()
}

async function getProfile(id: string, signal?: AbortSignal) {
  const user = await fetchUser(id, signal)
  const posts = await fetchPosts(id) // ← забыли signal
  return { user, posts }
}
```

Interruptions

Promise нельзя отменить
проигравшие в `Promise.race` сами не остановятся

AbortSignal — вручную
arg-drilling; легко забыть

Отмена соседей — вручную
упал один — остальные работают дальше

```
async function fetchUser(id: string, signal?: AbortSignal) {
  const resp = await fetch(`/users/${id}`, { signal })
  return await resp.json()
}

const controller = new AbortController()

const [user, posts] = await Promise.all([
  fetchUser('admin', controller.signal),
  fetchPosts('admin', controller.signal),
]).catch((error) => {
  controller.abort() // остановить остальных
  throw error
})
```

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Composition: «универсальный» retry

```ts
const user = await retry(() => fetchUser('admin'), 3)

async function retry<T>(
  fn: () => Promise<T>,
  times: number,
): Promise<T> {
  for (let attempt = 1; ; attempt++) {
    try {
      return await fn()
    }
    catch (error) {
      if (attempt === times)
        throw error
    }
  }
}
```

Composition: «универсальный» retry

Error Handling

А если `fn` не бросает исключение, а возвращает `Result` ?

```
const user = await retry(() => fetchUser('admin'), 3)

async function retry<T>(
  fn: () => Promise<T>,
  times: number,
): Promise<T> {
  for (let attempt = 1; ; attempt++) {
    try {
      return await fn()
    }
    catch (error) {
      if (attempt === times)
        throw error
    }
  }
}
```

Composition: «универсальный» retry

Error Handling

А если `fn` не бросает исключение, а возвращает `Result` ?

Interruptions

Как отменить попытку, которая уже запустилась?

```ts
const user = await retry(() => fetchUser('admin'), 3)

async function retry<T>(
  fn: () => Promise<T>,
  times: number,
): Promise<T> {
  for (let attempt = 1; ; attempt++) {
    try {
      return await fn()
    }
    catch (error) {
      if (attempt === times)
        throw error
    }
  }
}
```

Composition: «универсальный» retry

Error Handling
А если `fn` не бросает исключение, а возвращает `Result` ?

Interruptions
Как отменить попытку, которая уже запустилась?

Dependency Injection
Хочу логировать попытки. Откуда взять логгер?

```
const user = await retry(() => fetchUser('admin'), 3)

async function retry<T>(
  fn: () => Promise<T>,
  times: number,
): Promise<T> {
  for (let attempt = 1; ; attempt++) {
    try {
      return await fn()
    }
    catch (error) {
      if (attempt === times)
        throw error
    }
  }
}
```

Composition: «универсальный» retry

Error Handling
А если `fn` не бросает исключение, а возвращает `Result` ?

Interruptions
Как отменить попытку, которая уже запустилась?

Dependency Injection
Хочу логировать попытки. Откуда взять логгер?

Composition
Общего ответа нет — поэтому retry у каждого свой

```
import pRetry from 'p-retry'
import asyncRetry from 'async-retry'
import promiseRetry from 'promise-retry'
import axiosRetry from 'axios-retry'
import fetchRetry from 'fetch-retry'
import { backOff } from 'exponential-backoff'
import { retry } from 'ts-retry-promise'
import { retry } from 'cockatiel'
import { retry } from 'rxjs'

ky(url, { retry: 3 })
got(url, { retry: { limit: 3 } })
useQuery({ queryKey, queryFn, retry: 3 })
```

Что не так с TypeScript?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { setTimeout as sleep } from 'node:timers/promises'
import { SpanStatusCode, trace } from '@opentelemetry/api'
import { Result, err, ok } from 'neverthrow'

type UserError = 'NoBase' | 'Network' | 'Timeout' | 'NotOk' | 'Decode'

const tracer = trace.getTracer('users-client')

export async function fetchUser(
  deps: { config: { apiUrl?: string }, logger: Logger },
  id: string,
  { signal, timeoutMs = 5_000, retries = 3 }: {
    signal?: AbortSignal
    timeoutMs?: number
    retries?: number
  } = {},
): Promise<Result<User, UserError>> {
  return tracer.startActiveSpan(
    'fetchUser',
    { attributes: { 'user.id': id } },
    async (span): Promise<Result<User, UserError>> => {
      try {
        const base = deps.config.apiUrl
        if (!base) {
          span.setStatus({ code: SpanStatusCode.ERROR, message: 'API_URL is not set' })
          return err('NoBase')
        }

        for (let attempt = 1; ; attempt++) {
          span.setAttribute('http.request.resend_count', attempt - 1)
          const timeout = AbortSignal.timeout(timeoutMs)
          const attemptSignal = signal ? AbortSignal.any([signal, timeout]) : timeout

          let error: UserError
          try {
            const resp = await fetch(`${base}/users/${encodeURIComponent(id)}`, {
              signal: attemptSignal,
            })
            span.setAttribute('http.response.status_code', resp.status)

            if (resp.ok) {
              const json = await resp.json().catch(() => undefined)
              const parsed = UserSchema.safeParse(json)
              if (!parsed.success) {
                span.recordException(parsed.error)
                span.setStatus({ code: SpanStatusCode.ERROR, message: 'Invalid user' })
                return err('Decode')
              }
              return ok(parsed.data)
            }

            if (resp.status < 500 && resp.status !== 429) {
              span.setStatus({ code: SpanStatusCode.ERROR, message: `HTTP ${resp.status}` })
              return err('NotOk') // 4xx — повторять бессмысленно
            }
            error = 'NotOk'
          }
          catch (e) {
            if (signal?.aborted) {
              span.setStatus({ code: SpanStatusCode.ERROR, message: 'Aborted' })
              throw signal.reason // отменили снаружи — это не наша ошибка
            }
            span.recordException(e as Error)
            error = timeout.aborted ? 'Timeout' : 'Network'
          }

          if (attempt > retries) {
            span.setStatus({ code: SpanStatusCode.ERROR, message: `${error} after ${attempt} attempts` })
            return err(error)
          }

          const delay = 2 ** attempt * 100 + Math.random() * 100
          span.addEvent('retry', { attempt, delay, error })
          deps.logger.warn(`fetchUser(${id}): attempt ${attempt} failed (${error}), retry in ${delay}ms`)
          await sleep(delay, undefined, { signal })
        }
      }
      finally {
        span.end()
      }
    },
  )
}
```

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Effect<A>

|

Success

результат

Создаём эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
```

Создаём эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string>
```

Создаём эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string>

const maybe = Effect.sync(() => Math.random() > 0.5)
//      └── Effect<boolean>
```

Создаём эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string>

const maybe = Effect.sync(() => Math.random() > 0.5)
//      └── Effect<boolean>

const response = Effect.promise(() => fetch('https://example.com'))
//      └── Effect<Response>
```

Запускаем эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)
const response = Effect.promise(() => fetch('https://example.com'))
```

Запускаем эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)
const response = Effect.promise(() => fetch('https://example.com'))

Effect.runSync(event) // 'HolyJS'
```

Запускаем эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)
const response = Effect.promise(() => fetch('https://example.com'))

Effect.runSync(event) // 'HolyJS'
Effect.runSync(maybe) // true
```

Запускаем эффекты

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)
const response = Effect.promise(() => fetch('https://example.com'))

Effect.runSync(event) // 'HolyJS'
Effect.runSync(maybe) // true
Effect.runPromise(response) // Promise<Response>
```

Promise vs. Effect

```
const promise = fetch('https://example.com')
//      └── запрос уже улетел

const effect = Effect.promise(() => fetch('https://example.com'))
//      └── это просто значение
```

Promise vs. Effect

	Promise	Effect
Что это?	результат вычисления	описание вычисления

Promise vs. Effect

	Promise	Effect
Что это?	результат вычисления	описание вычисления
Когда запускается?	в момент создания	когда запустит рантайм

Promise vs. Effect

	Promise	Effect
Что это?	результат вычисления	описание вычисления
Когда запускается?	в момент создания	когда запустит рантайм
Как повторить при ошибке?	никак	Effect.retry

Promise vs. Effect

	Promise	Effect
Что это?	результат вычисления	описание вычисления
Когда запускается?	в момент создания	когда запустит рантайм
Как повторить при ошибке?	никак	Effect.retry
Как прервать по таймауту?	никак	Effect.timeout

Effect.gen

```
const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() ⇒ Math.random() > 0.5)

const program = () ⇒ {
  const name = event() // ???

  for (let year = 2016; year ⩽ 2026; year++) {
    const accepted = maybe() // ???
    if (accepted)
      return `Выступаю на ${name} ${year}!`
  }

  return 'Попробую в следующем году'
}
```

Effect.gen

```ts
const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)

const program = Effect.gen(function* () {
  const name = yield* event

  for (let year = 2016; year <= 2026; year++) {
    const accepted = yield* maybe
    if (accepted)
      return `Выступаю на ${name} ${year}!`
  }

  return 'Попробую в следующем году'
})
```

Effect.gen

```
const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() ⇒ Math.random() > 0.5)

//      ┌── Effect<string>
const program = Effect.gen(function* () {
  const name = yield* event
  //     └── string     └── Effect<string>

  for (let year = 2016; year ⩽ 2026; year++) {
    const accepted = yield* maybe
    //     └── boolean        └── Effect<boolean>
    if (accepted)
      return `Выступаю на ${name} ${year}!`
  }

  return 'Попробую в следующем году'
})
```

Effect.gen

```ts
const event = Effect.succeed('HolyJS')
const maybe = Effect.sync(() => Math.random() > 0.5)

//      ┌── Effect<string>
const program = Effect.gen(function* () {
  const name = yield* event
  //     └── string     └── Effect<string>

  for (let year = 2016; year <= 2026; year++) {
    const accepted = yield* maybe
    //     └── boolean        └── Effect<boolean>
    if (accepted)
      return `Выступаю на ${name} ${year}!`
  }

  return 'Попробую в следующем году'
})

Effect.runSync(program) // 'Выступаю на HolyJS 2026!'
```

```
async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { Effect } from 'effect'

async function fetchUser(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { Effect } from 'effect'

async function fetchUserEffect(id: string): Promise<User> {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = await fetch(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = await resp.json()
  return json as User
}
```

```ts
import { Effect } from 'effect'

const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* fetch(`${base}/users/${id}`) // ???
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* resp.json() // ???
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* Effect.promise(() => fetch(`${base}/users/${id}`))
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* Effect.promise(() => resp.json())
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

//     ┌── (id: string) => Effect<User>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* jsonEffect(resp)
  return json as User
})
```

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

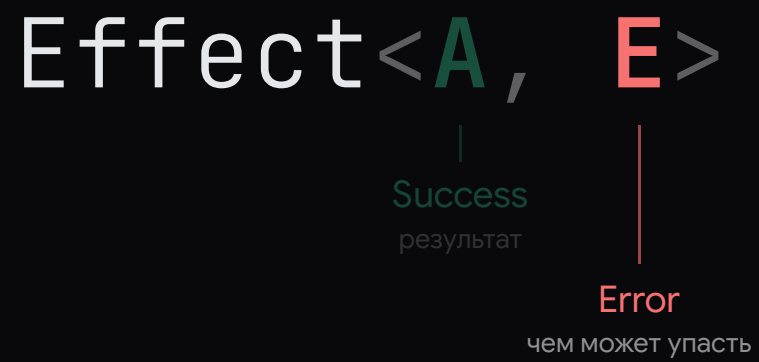
Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...


Effect<A, E>
Success
результат
Error
чем может упасть

```ts
// Plain TS
function fetchUser(id: string): Promise<User>

// neverthrow
function fetchUserResult(id: string): ResultAsync<User, UserError>

// Effect
function fetchUserEffect(id: string): Effect<User, UserError>
```

```ts
// Plain TS
function fetchUser(id: string): Promise<User>

// neverthrow
function fetchUserResult(id: string): ResultAsync<User, UserError>

// Effect
function fetchUserEffect(id: string): Effect<User, UserError>
```

```ts
// Plain TS
function fetchUser(id: string): Promise<User>

// neverthrow
function fetchUserResult(id: string): ResultAsync<User, UserError>

// Effect
function fetchUserEffect(id: string): Effect<User, UserError>
```

```ts
// Plain TS
function fetchUser(id: string): Promise<User>

// neverthrow
function fetchUserResult(id: string): ResultAsync<User, UserError>

// Effect
function fetchUserEffect(id: string): Effect<User, UserError>
```

Создаём ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string>

const response = Effect.promise(() => fetch('https://example.com'))
//      └── Effect<Response>
```

Создаём ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string, never>

const response = Effect.promise(() => fetch('https://example.com'))
//      └── Effect<Response, never>
```

Создаём ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string, never>

const boom = Effect.fail('BOOM')
//      └── Effect<never, 'BOOM'>

const response = Effect.promise(() => fetch('https://example.com'))
//      └── Effect<Response, never>
```

Создаём ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
//      └── Effect<string, never>

const boom = Effect.fail('BOOM')
//      └── Effect<never, 'BOOM'>

const response = Effect.tryPromise({
  try: () => fetch('https://example.com'),
  catch: () => 'Network',
})
//      └── Effect<Response, 'Network'>
```

Получаем ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const boom = Effect.fail('BOOM')
const response = Effect.tryPromise({
  try: () => fetch('https://example.com'),
  catch: () => 'Network',
})

Effect.runSync(boom) // ❌ Uncaught: 'BOOM'
```

Получаем ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const boom = Effect.fail('BOOM')
const response = Effect.tryPromise({
  try: () => fetch('https://example.com'),
  catch: () => 'Network',
})

Effect.runSync(boom)        // ❌ Uncaught: 'BOOM'
Effect.runPromise(response) // ❌ Rejected: 'Network'
```

Обрабатываем ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const boom = Effect.fail('BOOM')
const response = Effect.tryPromise({
  try: () => fetch('https://example.com'),
  catch: () => 'Network',
})

const safe = Effect.catch(boom, e => Effect.succeed(`Поймали ${e}`))
//     └── Effect<string, never>
```

Обрабатываем ошибки

```ts
import { Effect } from 'effect'

const event = Effect.succeed('HolyJS')
const boom = Effect.fail('BOOM')
const response = Effect.tryPromise({
  try: () => fetch('https://example.com'),
  catch: () => 'Network',
})

const safe = Effect.catch(boom, e => Effect.succeed(`Поймали ${e}`))
//     └── Effect<string, never>

Effect.runPromise(safe) // 'Поймали BOOM'
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

//     ┌── (id: string) => Effect<User>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

//     ┌── (id: string) => Effect<User, never>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    throw 'NoBase'

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

//     ┌── (id: string) => Effect<User, 'NoBase'>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    throw 'NotOk'

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.promise(() => fetch(url))
const jsonEffect = (resp: Response) => Effect.promise(() => resp.json())

//     ┌── (id: string) => Effect<User, 'NoBase' | 'NotOk'>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.tryPromise({
  try: () => fetch(url),
  catch: () => 'Network',
})
const jsonEffect = (resp: Response) => Effect.tryPromise({
  try: () => resp.json(),
  catch: () => 'Decode',
})

//     ┌── (id: string) => Effect<User, 'NoBase' | 'NotOk' | 'Network' | 'Decode'>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => Effect.tryPromise({
  try: () => fetch(url),
  catch: () => 'Network',
})
const jsonEffect = (resp: Response) => Effect.tryPromise({
  try: () => resp.json(),
  catch: () => 'Decode',
})

//     ┌── (id: string) => Effect<User, 'NoBase' | 'NotOk' | 'Network' | 'Decode'>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
import { err, ok, safeTry } from 'neverthrow'

const fetchResult = (url: string) => /* … */
const jsonResult = (resp: Response) => /* … */

//     ┌── (id: string) => ResultAsync<User, UserError>
//     ▼
const fetchUserResult = (id: string) => safeTry(
  async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)
    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  }
)
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => /* … */
const jsonEffect = (resp: Response) => /* … */

//     ┌── (id: string) => Effect<User, UserError>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(
  function* () {
    const base = process.env.API_URL
    if (!base)
      return yield* Effect.fail('NoBase')

    const resp = yield* fetchEffect(`${base}/users/${id}`)
    if (!resp.ok)
      return yield* Effect.fail('NotOk')

    const json = yield* jsonEffect(resp)

    return json as User
  }
)
```

```ts
import { err, ok, safeTry } from 'neverthrow'

const fetchResult = (url: string) => /* … */
const jsonResult = (resp: Response) => /* … */

//     ┌── (id: string) => ResultAsync<User, UserError>
//     ▼
const fetchUserResult = (id: string) => safeTry(
  async function* () {
    const base = process.env.API_URL
    if (!base)
      return err('NoBase')

    const resp = yield* fetchResult(`${base}/users/${id}`)
    if (!resp.ok)
      return err('NotOk')

    const json = yield* jsonResult(resp)

    return ok(json as User)
  }
)
```

```ts
import { Effect } from 'effect'

const fetchEffect = (url: string) => /* … */
const jsonEffect = (resp: Response) => /* … */

//     ┌── (id: string) => Effect<User, UserError>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(
  function* () {
    const base = process.env.API_URL
    if (!base)
      return yield* Effect.fail('NoBase')

    const resp = yield* fetchEffect(`${base}/users/${id}`)
    if (!resp.ok)
      return yield* Effect.fail('NotOk')

    const json = yield* jsonEffect(resp)

    return json as User
  }
)
```

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

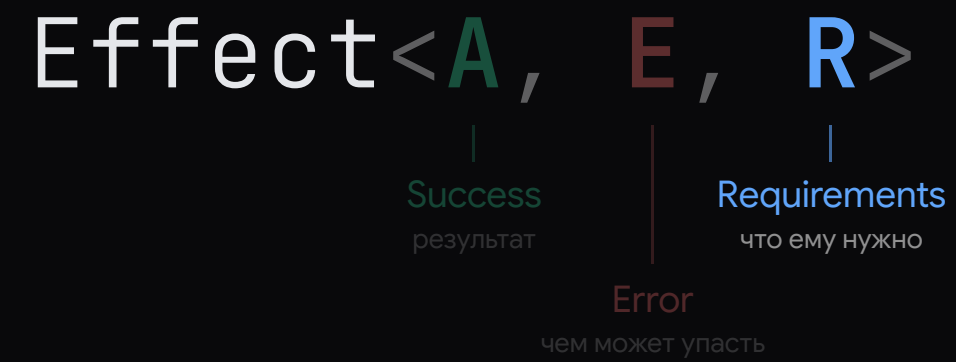
Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...


Effect<A, E, R>
Success
результат
Error
чем может упасть
Requirements
что ему нужно

```ts
const fetchEffect = (url: string) => Effect.tryPromise({
  try: () => fetch(url),
  catch: () => 'Network',
})

//     ┌── (id: string) => Effect<User, UserError>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
const fetchEffect = (url: string) => Effect.tryPromise({
  try: () => fetch(url),
  catch: () => 'Network',
})

//     ┌── (id: string) => Effect<User, UserError, never>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
const fetchEffect = (url: string) => Effect.tryPromise({
  try: () => fetch(url),
  catch: () => 'Network',
})

//     ┌── (id: string) => Effect<User, UserError, never>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

Сервисы

```
interface Network {
  fetch: (url: string) ⇒ Effect<Response, 'Network'>
}
```

Сервисы

```ts
import { Context } from 'effect'

interface Network {
  fetch: (url: string) => Effect<Response, 'Network'>
}

const Network = Context.Service<Network>('app/Network')
```

Сервисы

```ts
import { Context } from 'effect'

interface Network {
  fetch: (url: string) => Effect<Response, 'Network'>
}

const Network = Context.Service<Network>('app/Network')

interface Env {
  get: (name: string) => Effect<string | undefined>
}

const Env = Context.Service<Env>('app/Env')
```

Сервисы

```ts
import { Context } from 'effect'

interface Network {
  fetch: (url: string) => Effect<Response, 'Network'>
}

//     ┌── Service<Network>
//     ▼
const Network = Context.Service<Network>('app/Network')

interface Env {
  get: (name: string) => Effect<string | undefined>
}

//     ┌── Service<Env>
//     ▼
const Env = Context.Service<Env>('app/Env')
```

```ts
import { Effect } from 'effect'

//     ┌── (id: string) ⇒ Effect<User, UserError, never>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env

  const base = process.env.API_URL
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* fetchEffect(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* network.fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) => Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

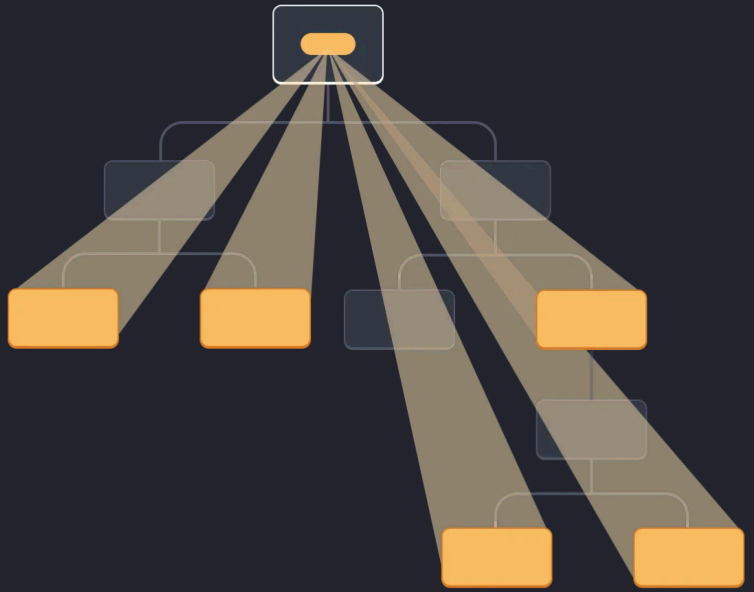
  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* network.fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

Using context in distant children



react.dev: Passing Data Deeply with Context

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* network.fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)

  return json as User
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) ⇒ Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  // ...
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) => Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  // ...
})

const userInfo = (id: string) => Effect.gen(function* () {
  const user = yield* fetchUserEffect(id)
  return `${user.name}, ${user.age}`
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) => Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  // ...
})

//     ┌── (id: string) => Effect<string, UserError, Env | Network>
//     ▼
const userInfo = (id: string) => Effect.gen(function* () {
  const user = yield* fetchUserEffect(id)
  return `${user.name}, ${user.age}`
})
```

```ts
import { Effect } from 'effect'
import { Env, Network } from './services'

//     ┌── (id: string) => Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  // ...
})

//     ┌── (id: string) => Effect<string, UserError, Env | Network>
//     ▼
const userInfo = (id: string) => Effect.gen(function* () {
  const user = yield* fetchUserEffect(id)
  return `${user.name}, ${user.age}`
})

Effect.runPromise(userInfo('42'))
//                ~~~~~~~~~~~~~~
// ❌ Type 'Env | Network' is not assignable to type 'never'
```

```
Effect<User, UserError, Env | Network>
```

- Env
- Network

`Effect.runPromise(…)` ❌ не компилируется

`Effect<User, UserError, Network>`

- **Env** ← `realEnv`
- **Network**

`Effect.runPromise(…)` ❌ не компилируется

`Effect<User, UserError, never>`

- Env ← `realEnv`
- Network ← `realNetwork`

`Effect.runPromise(…)` ✅ компилируется

```ts
const realEnv: Env = {
  get: name ⇒ Effect.succeed(process.env[name]),
}

const realNetwork: Network = {
  fetch: url ⇒ Effect.tryPromise({
    try: () ⇒ fetch(url),
    catch: () ⇒ 'Network',
  }),
}

const program = fetchUserEffect('42')
//    └── Effect<User, UserError, Env | Network>

Effect.runPromise(program) // ❌ Type 'Env | Network' is not assignable to type 'never'
```

```ts
const realEnv: Env = {
  get: name ⇒ Effect.succeed(process.env[name]),
}

const realNetwork: Network = {
  fetch: url ⇒ Effect.tryPromise({
    try: () ⇒ fetch(url),
    catch: () ⇒ 'Network',
  }),
}

const program = fetchUserEffect('42')
//    └── Effect<User, UserError, Env | Network>

const program2 = Effect.provideService(program, Env, realEnv)
//    └── Effect<User, UserError, Network>

Effect.runPromise(program2) // ❌ Type 'Network' is not assignable to type 'never'
```

```ts
const realEnv: Env = {
  get: name => Effect.succeed(process.env[name]),
}

const realNetwork: Network = {
  fetch: url => Effect.tryPromise({
    try: () => fetch(url),
    catch: () => 'Network',
  }),
}

const program = fetchUserEffect('42')
//    └── Effect<User, UserError, Env | Network>

const program2 = Effect.provideService(program, Env, realEnv)
//    └── Effect<User, UserError, Network>

const program3 = Effect.provideService(program2, Network, realNetwork)
//    └── Effect<User, UserError, never>

Effect.runPromise(program3) // ✅
```

```ts
const testEnv: Env = {
  get: () => Effect.succeed('http://test'),
}

const testNetwork: Network = {
  fetch: () => Effect.succeed(Response.json({ name: 'Claude', age: 1 })),
}

const program = fetchUserEffect('42')
//    └── Effect<User, UserError, Env | Network>

const program2 = Effect.provideService(program, Env, testEnv)
//    └── Effect<User, UserError, Network>

const program3 = Effect.provideService(program2, Network, testNetwork)
//    └── Effect<User, UserError, never>

Effect.runPromise(program3) // ✅
```

Dependency Injection (Effect)

Dependency Injection (Effect)

- ✓ Зависимости отражены в типах `(type-safe)`

Dependency Injection (Effect)

- ✓ Зависимости отражены в типах `(type-safe)`
- ✓ Зависимости не нужно протаскивать `(arg-drilling)`

Dependency Injection (Effect)

- ✓ Зависимости отражены в типах `(type-safe)`
- ✓ Зависимости не нужно протаскивать `(arg-drilling)`
- ✓ Зависимости легко подменить

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

```ts
//     ┌── (id: string) => Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) => Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* network.fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
const realNetwork: Network = {
  fetch: url => Effect.tryPromise({
    try: () => fetch(url),
    catch: () => 'Network',
  }),
}
```

```ts
//     ┌── (id: string) ⇒ Effect<User, UserError, Env | Network>
//     ▼
const fetchUserEffect = (id: string) ⇒ Effect.gen(function* () {
  const env = yield* Env
  const network = yield* Network

  const base = yield* env.get('API_URL')
  if (!base)
    return yield* Effect.fail('NoBase')

  const resp = yield* network.fetch(`${base}/users/${id}`)
  if (!resp.ok)
    return yield* Effect.fail('NotOk')

  const json = yield* jsonEffect(resp)
  return json as User
})
```

```ts
const realNetwork: Network = {
  fetch: url ⇒ Effect.tryPromise({
    try: signal ⇒ fetch(url, { signal }),
    //   └── рантайм сам вызовет abort()
    catch: () ⇒ 'Network',
  }),
}
```

Прерываем по таймауту

```
import { Effect } from 'effect'

const withTimeout = Effect.timeout(
  fetchUserEffect('admin'),
  '2 seconds',
)
```

Прерываем проигравших

```
import { Effect } from 'effect'

const withTimeout = Effect.timeout(
  fetchUserEffect('admin'),
  '2 seconds',
)

const fastest = Effect.race(
  fetchUserEffect('bob'),
  fetchUserEffect('alice'),
)
```

Прерываем соседей

```
import { Effect } from 'effect'

const withTimeout = Effect.timeout(
  fetchUserEffect('admin'),
  '2 seconds',
)

const fastest = Effect.race(
  fetchUserEffect('bob'),
  fetchUserEffect('alice'),
)

const users = Effect.all(
  [
    fetchUserEffect('bob'),
    fetchUserEffect('alice'),
    fetchUserEffect('vlad'),
  ],
  { concurrency: 'unbounded' },
)
```

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Композиция

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
```

Композиция

```ts
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user ⇒ user.name)
```

Композиция

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user ⇒ user.name)
const effect3 = Effect.retry(effect2, { times: 3 })
```

Композиция

```ts
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user => user.name)
const effect3 = Effect.retry(effect2, { times: 3 })
const effect4 = Effect.withSpan(effect3, 'fetchUser')
```

Композиция

```
import { Effect } from 'effect'

const effect = Effect.withSpan(
  Effect.retry(
    Effect.map(
      fetchUserEffect('admin'),
      user ⇒ user.name,
    ),
    { times: 3 },
  ),
  'fetchUser',
)
```

А может, как в jQuery?

jQuery

```
$('#user')
  .addClass('admin')
  .text('Привет!')
  .fadeIn(200)
```

Effect (на методах)

```
const effect = fetchUserEffect('admin')
  .map(user ⇒ user.name)
  .retry({ times: 3 })
  .withSpan('fetchUser')
```

А может, как в jQuery?

jQuery

```
$('#user')
  .addClass('admin')
  .text('Привет!')
  .fadeIn(200)
```

Effect (на методах)

```
const effect = fetchUserEffect('admin')
  .map(user ⇒ user.name)
  .retry({ times: 3 })
  .withSpan('fetchUser')
```

Не расширить снаружи

Свой метод — только патчем чужого прототипа:

`$.fn.myPlugin = ...`

А может, как в jQuery?

jQuery

```
$('#user')
  .addClass('admin')
  .text('Привет!')
  .fadeIn(200)
```

Не расширить снаружи

Свой метод — только патчем чужого прототипа:

```
$.fn.myPlugin = ...
```

Effect (на методах)

```
const effect = fetchUserEffect('admin')
  .map(user ⇒ user.name)
  .retry({ times: 3 })
  .withSpan('fetchUser')
```

Нет tree-shaking

Методы нельзя импортировать по одному — в бандл едет вся библиотека

Композиция

```ts
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user ⇒ user.name)
const effect3 = Effect.retry(effect2, { times: 3 })
const effect4 = Effect.withSpan(effect3, 'fetchUser')
```

Композиция

Data-first

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user ⇒ user.name)
const effect3 = Effect.retry(effect2, { times: 3 })
const effect4 = Effect.withSpan(effect3, 'fetchUser')
```

Data-last

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(user ⇒ user.name)(effect)
const effect3 = Effect.retry({ times: 3 })(effect2)
const effect4 = Effect.withSpan('fetchUser')(effect3)
```

Композиция

Data-first

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(effect, user ⇒ user.name)
//                         ^^^^^^
const effect3 = Effect.retry(effect2, { times: 3 })
//                           ^^^^^^^
const effect4 = Effect.withSpan(effect3, 'fetchUser')
//                              ^^^^^^^
```

Data-last

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(user ⇒ user.name)(effect)
//                                           ^^^^^^
const effect3 = Effect.retry({ times: 3 })(effect2)
//                                        ^^^^^^^
const effect4 = Effect.withSpan('fetchUser')(effect3)
//                                           ^^^^^^^
```

```ts
import { Effect } from 'effect'

const effect = fetchUserEffect('admin')
const effect2 = Effect.map(user => user.name)(effect)
const effect3 = Effect.retry({ times: 3 })(effect2)
const effect4 = Effect.withSpan('fetchUser')(effect3)
```

```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect2 = f(effect)
const effect3 = g(effect2)
const effect4 = h(effect3)
```

```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect4 = h(g(f(effect)))
```

h(g(f(effect)))

```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect4 = h(g(f(effect)))
```



```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect4 = h(g(f(effect)))
```



|> Pipeline Operator

```
const effect = fetchUserEffect('admin')
  |> Effect.map(%, user => user.name)
  |> Effect.retry(%, { times: 3 })
  |> Effect.withSpan(%, 'fetchUser')
```

tc39/proposal-pipeline-operator

|> Pipeline Operator

```
const effect = fetchUserEffect('admin')
  |> Effect.map(%, user => user.name)
  |> Effect.retry(%, { times: 3 })
  |> Effect.withSpan(%, 'fetchUser')
```

2015	2017	2021	2026
proposal	Stage 1	Stage 2	Stage 2

tc39/proposal-pipeline-operator

```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect4 = h(g(f(effect)))
```



```ts
import { Effect } from 'effect'

const f = Effect.map(user ⇒ user.name)
const g = Effect.retry({ times: 3 })
const h = Effect.withSpan('fetchUser')

const effect = fetchUserEffect('admin')
const effect4 = h(g(f(effect)))
```



Композиция

```ts
import { Effect } from 'effect'

const effect = Effect.withSpan(
  Effect.retry(
    Effect.map(
      fetchUserEffect('admin'),
      user => user.name,
    ),
    { times: 3 },
  ),
  'fetchUser',
)
```

Композиция

```
import { Effect } from 'effect'

const effect = fetchUserEffect('admin').pipe(
  Effect.map(user ⇒ user.name),
  Effect.retry({ times: 3 }),
  Effect.withSpan('fetchUser'),
)
```

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно ли лучше?

Error Handling

Какие исключения может выбросить эта функция? Что, если изменится реализация?

Dependency Injection

Как снизить coupling? Как это тестировать? Какие зависимости я забыл передать?

Interruptions

Что насчёт graceful shutdown? Как отменить запрос, если он уже никому не нужен?

Composition

Хорошо бы retry добавить. Для этой функции тоже. И телеметрию...

Можно в прод

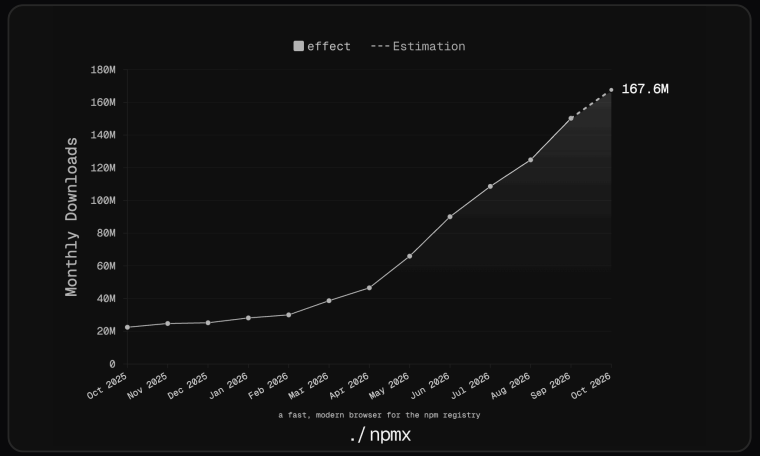


effect.website/blog

Можно в прод



effect.website/blog

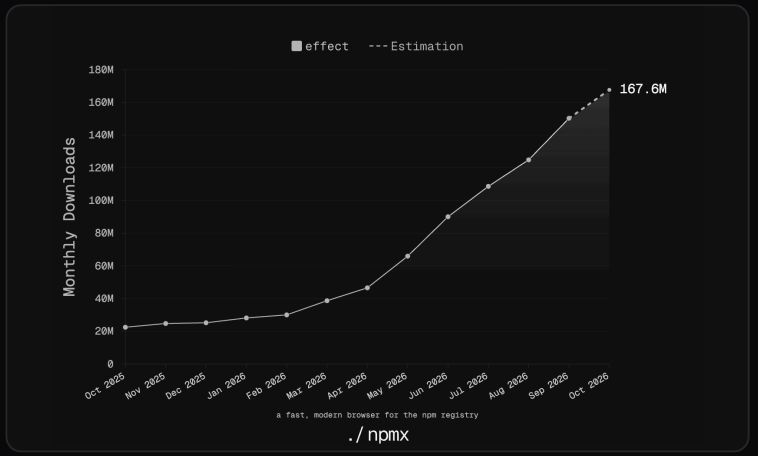


npmx.dev/package/effect

Можно в прод



effect.website/blog

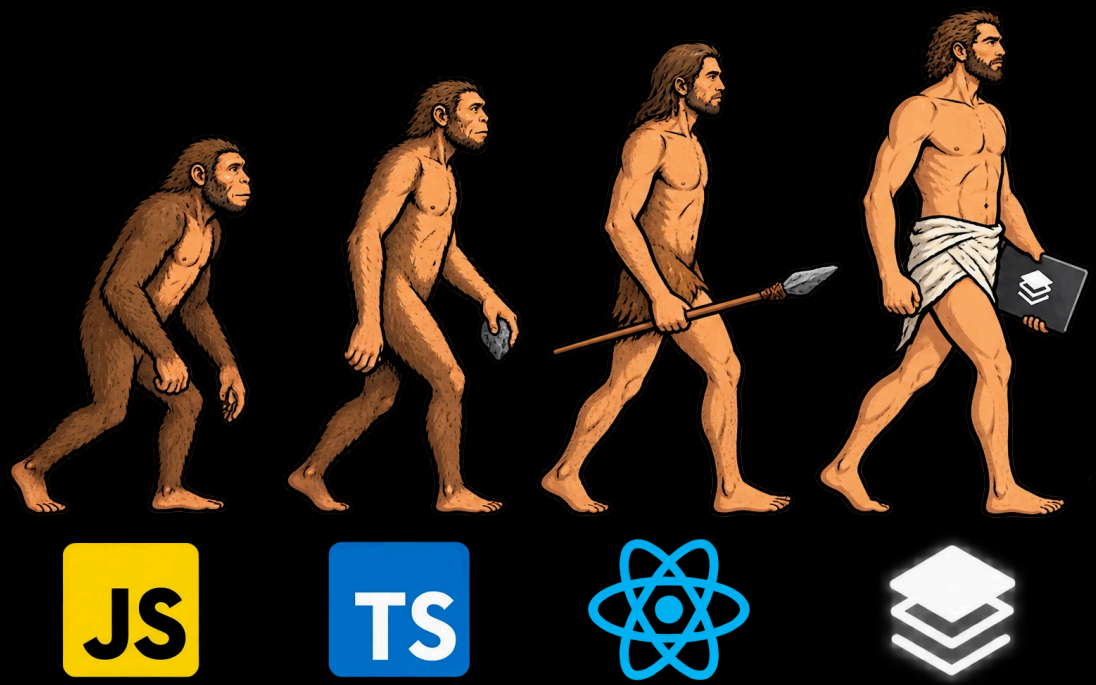


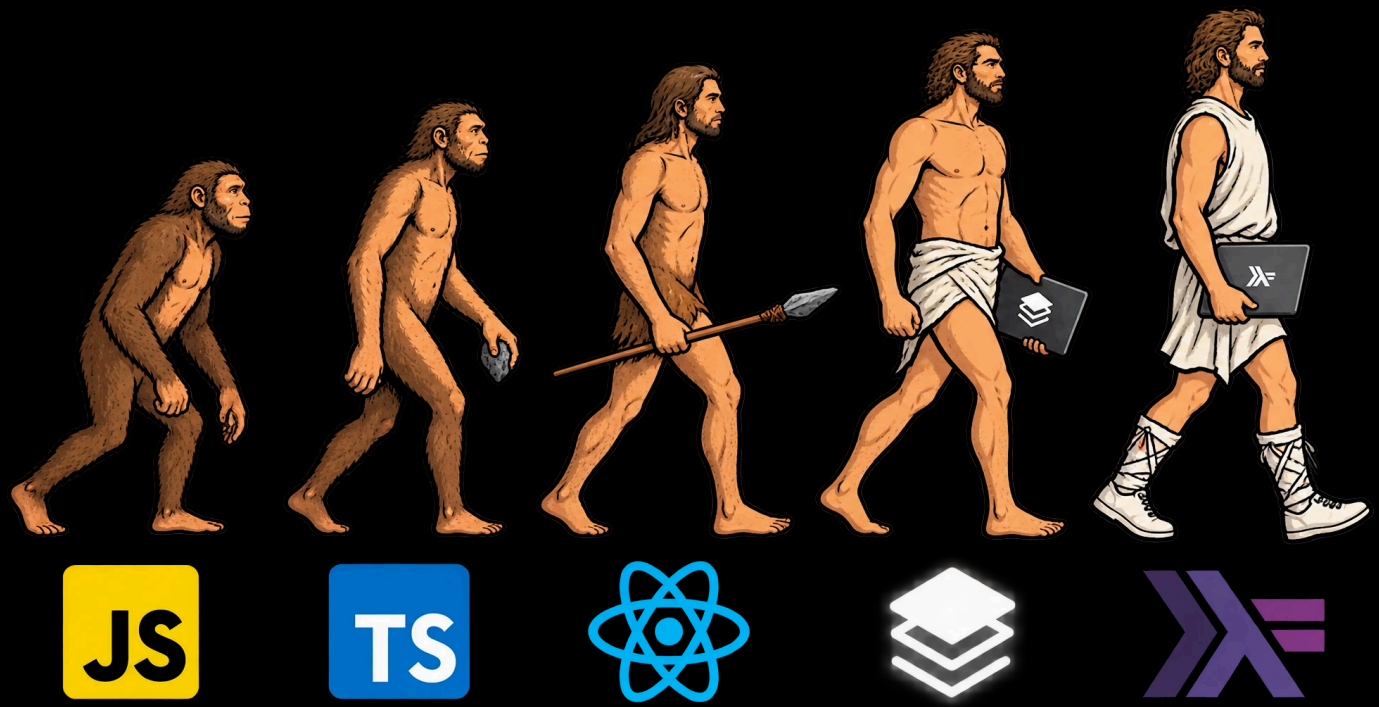
npmx.dev/package/effect

// IN PRODUCTION AT

CLOUDFLARE opencode MasterClass T3.chat

effect.website


JS
TS


JS
TS

```
Effect.forEach(questions).pipe(
  Vlad.answer,
  Effect.orDie,
)
```

24.10.2026 • HolyJS • @evermake