Обо мне

- Артём Арутюнян @artalar
- DBeaver
- 7 лет в ИТ
- 3 года делаю веб
- ~15 веб-приложений
- Христианин, муж, программист, бушкрафтер

Обо мне

Reatom

deterministic state manager

@artalar

Тема

Контрактное программирование

- Парадигма программирования подразумевающая строгие соглашения между интерфейсами и их контроль

Внимание!

Предложенные в докладе подходы и библиотеки
не универсальны и приведены для примера,
серебряных пуль не существует,
каждый инструмент должен выбираться
в соответствии с задачей и командой

Документация

```
/** Class representing a point. */
class Point {
    /**
     * Create a point.
     * @param {number} x - The x value.
     * @param {number} y - The y value.
     */
    constructor(x, y) {
        // ...
    }

    /**
     * Get the x value.
     * @return {number} The x value.
     */
    getX() {
        // ...
    }

    /**
     * Get the y value.
     * @return {number} The y value.
     */
    getY() {
        // ...
    }

    /**
     * Convert a string containing two comma-separated numbers into a point.
     * @param {string} str - The string containing two comma-separated numbers.
     * @return {Point} A Point object.
     */
    static fromString(str) {
        // ...
    }
}
```

План

- Статическое описание типов
- Type Guards
- Assert генераторы
- Валидаторы
- Контракты

Статическое описание типов

```
type Data = {
  prop: boolean
}

export const fetchData = () =>
  fetch('/data')
    .then(resp => resp.json())
    .then((data: Data) => {
      return data
    })
```

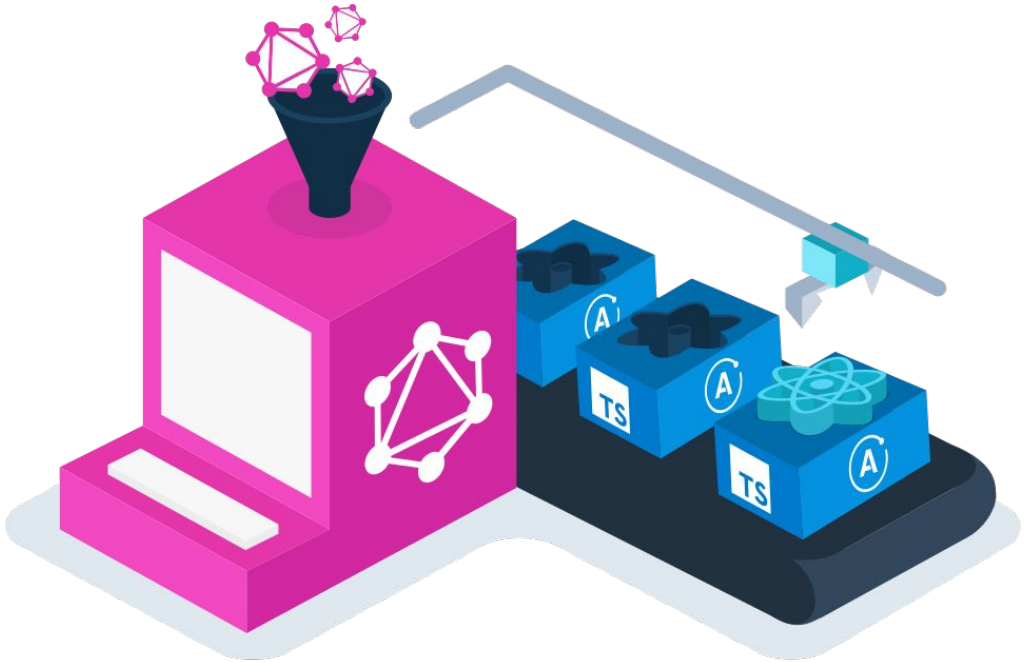
Статическое описание типов

Статическое описание типов

Кому верить?

Статическое описание типов

генерация



{ GraphQL }

code generator

Статическое описание типов

генерация

ts-generator supports:

- Primitive types, with or without explicit int.
- Kotlin and Java classes.
- Data classes.
- Enums.
- Any type.
- Generic classes, without type erasure.
- Generic constraints.
- Class inheritance.
- Abstract classes.
- Lists as JS arrays.
- Maps as JS objects.
- Null safety, even inside composite types.
- Java beans.
- Mapping types.
- Nullability annotations, when allowed by the retention policy.
- Customizing class definitions via transformers.
- Parenthesis optimization: They are placed only when they are needed to disambiguate.
- Emitting either `null` or `undefined` for JVM nullable types.

```
type Val = {
    kind: number,
    prop1?: number,
    prop2?: number,
}
const val: Val =
    { kind: 1, prop1: 1 } ||
    { kind: 2, prop2: 2 }
```

Статическое описание типов

Взять все в свои руки

Type Guards


```
if (typeof thing === 'string') {
                  let thing: string
  console.log(thing)
}
```

Type Guards

TypeScript

```
const fetchData: () => Promise<object>
export const fetchData = () =>
  fetch('/data')
    .then(resp => resp.json())
    .then((data: unknown) => {
      if (
        typeof data === 'object' &&
        data !== null &&
        'prop' in data &&
        typeof data['prop'] === 'boolean'
      ) {
        return data
      }
      throw new TypeError('Wrong response type')
    })
```

Type Guards

TypeScript

```ts
const isObj = (thing: unknown): thing is {} =>
  typeof thing === 'object' && thing !== null

const isBool = (thing: unknown): thing is boolean => typeof thing === 'boolean'
```

```ts
const fetchData: () => Promise<never>
export const fetchData = () =>
  fetch('/data')
    .then(resp => resp.json())
    .then((data: unknown) => {
      if (isObj(data) && 'prop' in data && isBool(data['prop'])) {
        return data
      }
      throw new TypeError('Unexpected response')
    })
```

Type Guards

TypeScript

(ノ ° □ °)ノ (┻━━┻)

Type Guards

TypeScript

```ts
const isData = (thing: unknown): thing is Data =>
  isObj(thing) && 'prop' in thing && isBool(thing['prop'])

export const fetchData = () =>
  fetch('/data')
    .then(resp => resp.json())
    .then((data: unknown) => {
      if (isData(data)) {
        return data
      }
      throw new TypeError('Unexpected response')
    })
```

```ts
const fetchData: () => Promise<Data>
```

Type Guards

TypeScript

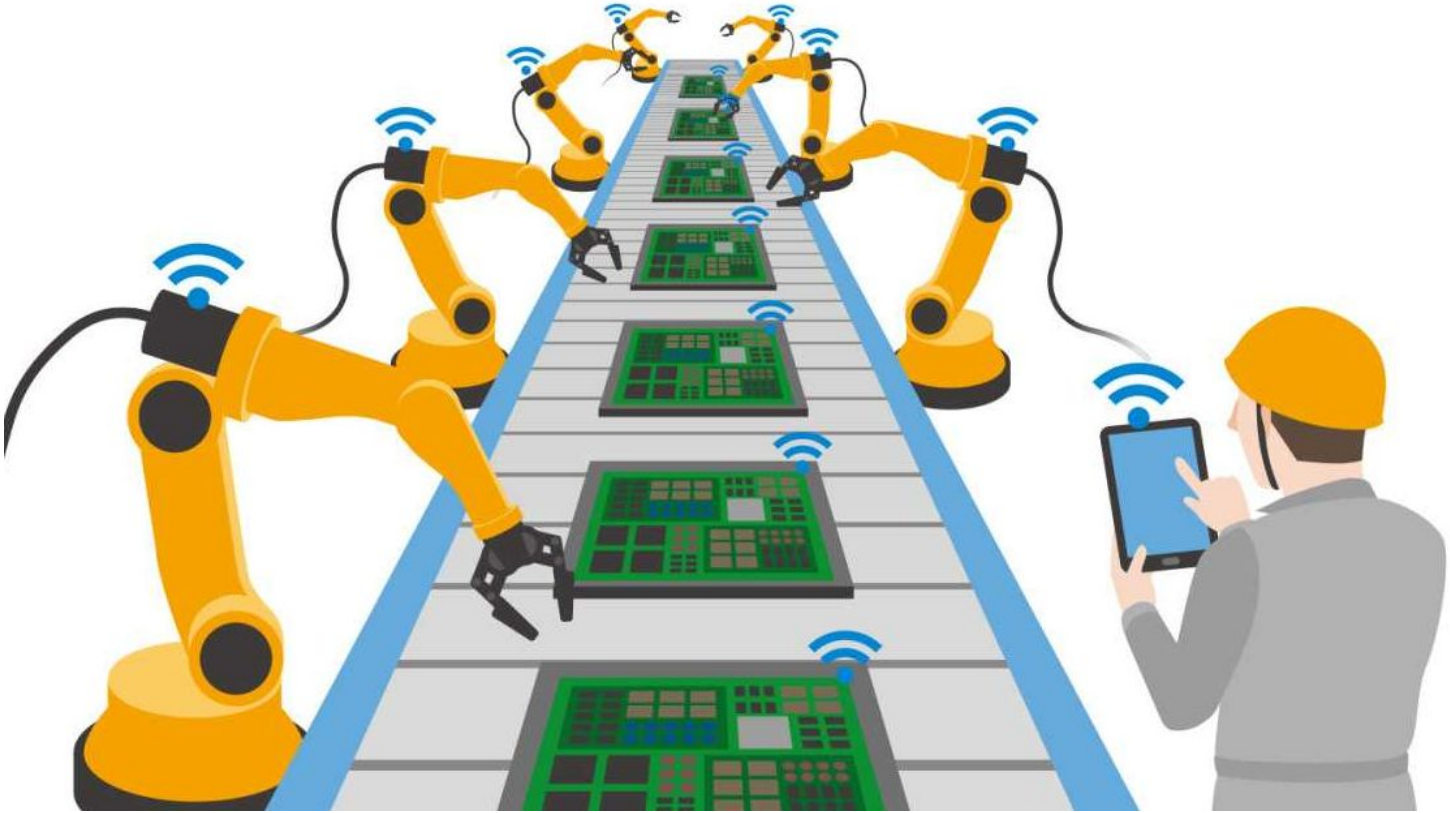
```ts
const isData = (thing: unknown): thing is Data =>
  thing === 'whatever'

export const fetchData = () =>
  fetch('/data')
    .then(resp => resp.json())
    .then((data: unknown) => {
      if (isData(data)) {
        return data
      }
      throw new TypeError('Unexpected response')
    })
```

```ts
const fetchData: () => Promise<Data>
```

@artalar

Type Guards

автоматизация



@artalar

Assert генераторы

- https://github.com/fabiandev/**ts-runtime**
- https://github.com/ts-type-makeup/**superstruct-ts-transformer**
- https://github.com/gajus/flow-runtime/tree/master/packages/**babel-plugin-flow-runtime**

Assert генераторы

ts-runtime

```
1  type Data = { prop: string }
2  fetch('/...')
3      .then(resp => resp.json())
4      .then(
5
6          (data: Data) => {
7              /*  ... */
8          }
9      )
10
11
```

```
1  import t from "ts-runtime/lib";
2  const Data = t.type("Data", t.object(t.property("prop
3  fetch('/...')
4      .then(t.annotate((resp) => {
5      return resp.json();
6  }, t.function(t.param("resp", t.any()), t.return(t.an
7      .then(t.annotate((data) => {
8      let _dataType = t.ref(Data);
9      t.param("data", _dataType).assert(data);
10 }, t.function(t.param("data", t.ref(Data)), t.return(
11
```

Run in new window (ctrl+r)

@artalar

Assert генераторы

ts-runtime

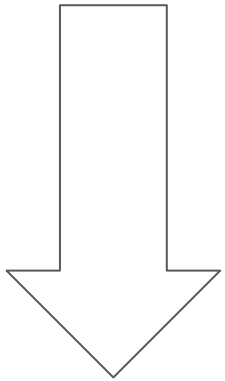
```
1 type AB = 'A' | 'B'
2
3 function fn(a: 'A') {
4     const result: AB = a
5 }
6
```

```
1 import t from "ts-runtime/lib";
2 const AB = t.type("AB", t.union(t.st
3 function fn(a) {
4     let _aType = t.string("A");
5     t.param("a", _aType).assert(a);
6     const result = t.ref(AB).assert(
7 }
8 t.annotate(fn, t.function(t.param("a
9
```

Run in new window (ctrl+r)

Assert генераторы

Superstruct Typescript transformer



```ts
import { validate } from "superstruct-ts-transformer";

type User = {
  name: string;
  alive: boolean;
};

const obj = validate<User>(JSON.parse('{ "name": "Me", "alive": true }'));
```

```ts
import superstruct from "superstruct";
var obj = validate_User(JSON.parse('{ "name": "Me", "alive": true }'));

function validate_User(jsonObj) {
  var validator = superstruct.struct({
    name: "string",
    alive: "boolean"
  });
  return validator(jsonObj);
}
```

@artalar

Формат

```
function getPhone(str: string) {
    const res = str.match(/(?:\+|\d)[\d\-\(\) ]{9,}\d/g)
    if (!res) throw new TypeError()
    else return str
}
```

Валидаторы

- https://github.com/hapijs/**joi**
- https://github.com/epoberezkin/**ajv**
- https://github.com/validatorjs/**validator.js**
- https://github.com/jquense/**yup**

Валидаторы

yup

```
import * as yup from 'yup';

const personSchema = yup.object({
  firstName: yup
    .string(),
  nickName: yup
    .string()
    .nullable(),
  gender: yup
    .mixed<'male' | 'female' | 'other'>()
    .oneOf(['male', 'female', 'other']),
  email: yup
    .string()
    .nullable()
    .notRequired()
    .email(),
  birthDate: yup
    .date()
    .nullable()
    .notRequired()
    .min(new Date(1900, 0, 1)),
});
```

You can derive the TypeScript type as follows:

```
type Person = yup.InferType<typeof personSchema>;
```

Which is equivalent to the following TypeScript type alias:

```
type Person = {
  firstName: string;
  nickName: string | null;
  gender: "male" | "female" | "other";
  email?: string | null | undefined;
  birthDate?: Date | null | undefined;
}
```

Валидаторы

yup

```
import * as yup from "yup";

const Data = yup.object({
  prop: yup.boolean()
});

export const fetchData = () =>
  fetch("/data")
    .then(resp => resp.json())
    .then((data: unknown) => {
      return Data.validateSync(data);
    });
```

Контракты

@artalar

Контракты

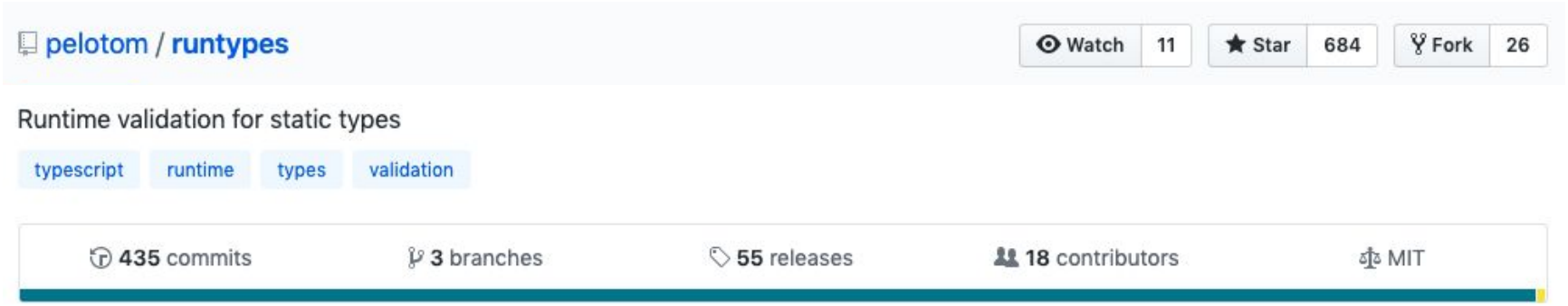
- https://github.com/ianstormtaylor/**superstruct**
- https://github.com/gajus/**flow-runtime**
- https://github.com/bigslycat/**typed-contracts**
- https://github.com/gcanti/**io-ts**
- https://github.com/pelotom/**runtypes**

Контракты

fp-ts

Контракты

runtypes



Контракты

runtypes

```
import * as t from "runtypes";

const Data = t.Record({
  prop: t.Boolean
});

export const fetchData = () =>
  fetch("/data")
    .then(resp => resp.json())
    .then((data: unknown) => {
      return Data.check(data);
    });
```

Контракты

runtypes

```ts
const Email = t.String.refine(
  'email',
  str => str.includes('@') || 'Is not email',
)
const Password = t.String.refine(
  'password',
  str => str.length ≥ 8 || 'To short password',
)

function send(
  email: Static<typeof Email>,
  password: Static<typeof Password>,
) {
  /* ... */
}
```

```ts
var email = Email.ensure('a@b.c')
var password = Password.ensure('12345678')

// static type ok
send(email, password)
```

```
(local var) password: UniqueType<string, "password">

Argument of type 'UniqueType<string, "password">' is not ass
'UniqueType<string, "email">'.
  Type 'UniqueType<string, "password">' is not assignable to
    Types of property '[UniqueSymbol]' are incompatible.
      Type '"password"' is not assignable to type '"email"'.

Peek Problem    No quick fixes available
```

```ts
// st
send(password, email)
```

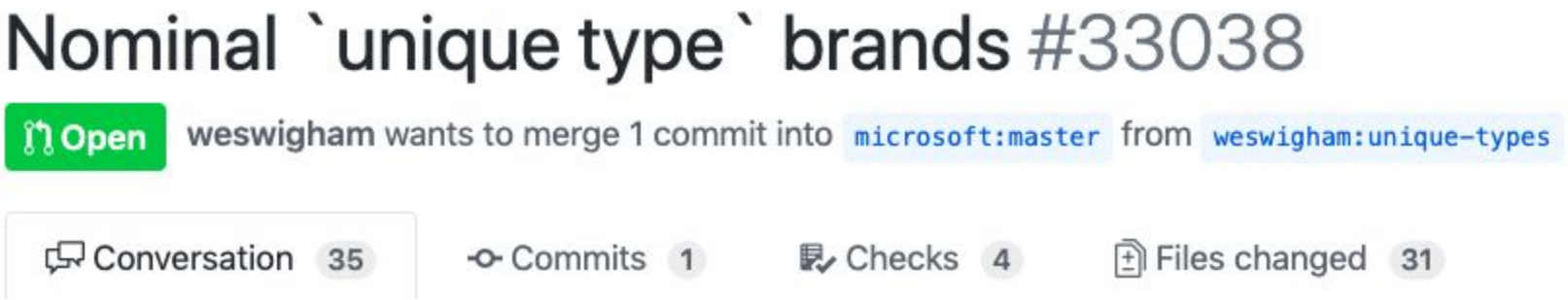
Контракты

runtypes

```ts
/**
 * @example 123e4567-e89b-12d3-a456-426655440000
 */
type UUID = unique string;

declare function isUUID(candidate: string): candidate is UUID;
declare function getUser(id: UUID): User | undefined;

declare const input: string;

/**
 * 🍎 Compile-time error. `getUser` won't accept just any string.
 */
getUser(input);

/**
 * 🍏 Works! We made sure our `input` is a special string.
 */
if (isUUID(input)) {
  getUser(input);
}
```

Контракты

runtypes



Контракты

runtypes

```
t.match(
  [AuthError, () => notify("Fail auth")],
  [SystemError, ({ error: { message } }) => notify(message)],
  [t.Unknown, () => notify("Something wrong")]
)(e);
```

Контракты

Ограничения

- Проверки занимают ресурсы ВМ (и бандлсайз)

Контракты

производительность

```
export const DATA = Object.freeze({
  number: 1,
  neg_number: -1,
  max_number: Number.MAX_VALUE,
  string: 'string',
  long_string: 'Lorem ipsum dolor sit amet,
  boolean: true,
  deeplyNested: {
    foo: 'bar',
    num: 1,
    bool: false
  }
})
```

Контракты

производительность

```
JSON Encode Decode:
  78 434 ops/s, ±4.34%      | 94.95% slower

runtypes:
  1 552 642 ops/s, ±4.46%   | fastest

io-ts:
  777 697 ops/s, ±3.03%     | 49.91% slower

class-validator sync:
  63 753 ops/s, ±4.15%      | 95.89% slower

class-validator async:
  41 651 ops/s, ±11.11%     | slowest, 97.32% slower
```

Контракты

Ограничения

- Проверки занимают ресурсы ВМ (и бандлсайз)
- Не получится не актуализировать (ну это же хорошо?)

Контракты

Ограничения

- Проверки занимают ресурсы ВМ (и бандлсайз)
- Не получится не актуализировать (ну это же хорошо?)
- Exception control flow

Контракты

Ограничения

- Проверки занимают ресурсы ВМ (и бандлсайз)
- Не получится не актуализировать (ну это же хорошо?)
- Exception control flow
- Неоднородный код (типов)

Контракты

Конец?

Контракты

Светлое будущее

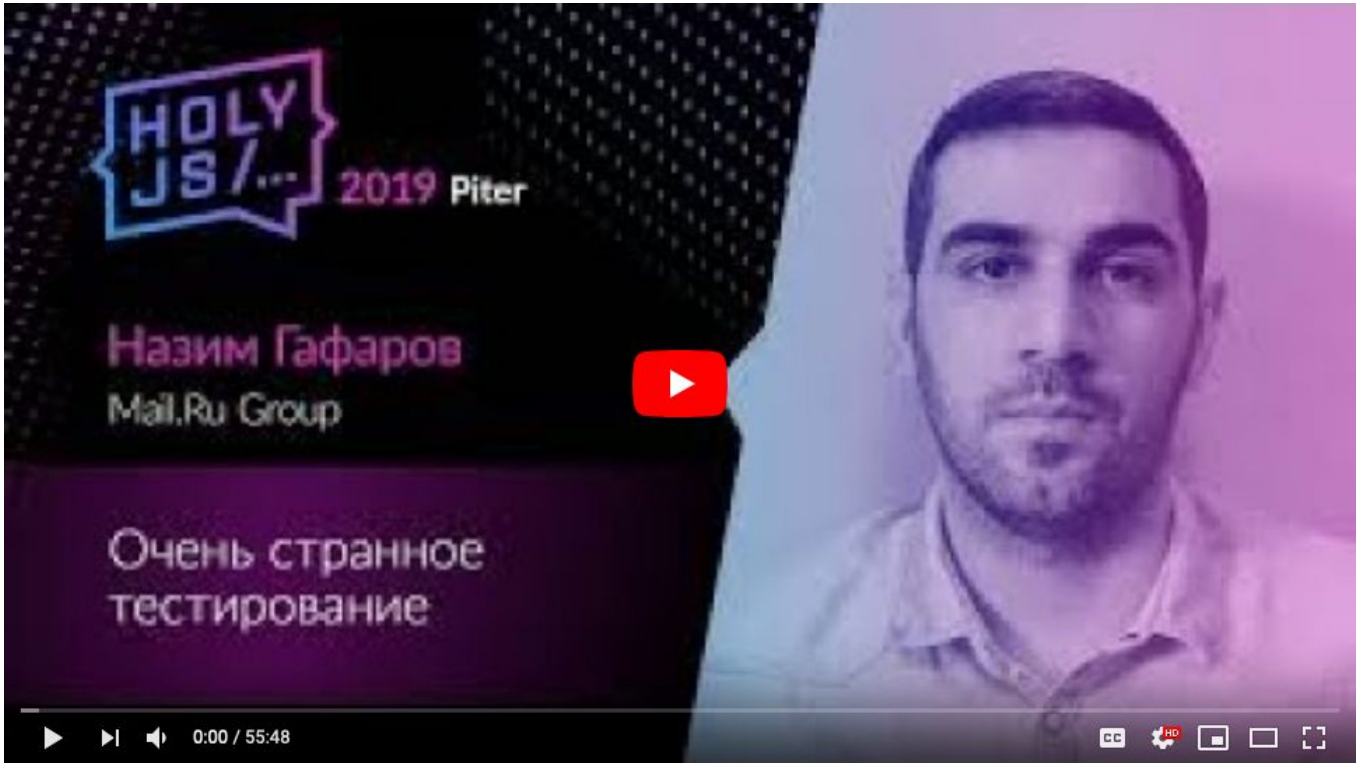
Контракты

Светлое будущее

- Автогенерация моков, property-base тестирование

Контракты

Property-base тестирование



Контракты

runtypes-generate

```
import { Number, Literal, Array, Tuple, Record } from 'runtypes'
const AsteroidType = Record({
    type: Literal('asteroid'),
    location: Tuple(Number, Number, Number),
    mass: Number,
})

const AsteroidArbitrary = jsc.record({
    type: jsc.constant('asteroid'),
    location: jsc.tuple(jsc.number, jsc.number, jsc.number),
    mass: jsc.number
})
```

But with `runtypes-generate` we can get `AsteroidArbitrary` from `AsteroidType`:

```
import { makeJsverifyArbitrary } from 'runtypes-generate'
const AsteroidType = Record({
    type: Literal('asteroid'),
    location: Tuple(Number, Number, Number),
    mass: Number,
})
const AsteroidArbitrary = makeJsverifyArbitrary(AsteroidType)
```

Контракты

Светлое будущее

- Автогенерация моков, property-base тестирование
- SSoT модели данных

Контракты

```
import * as t from "runtypes";

const Email = t.String`
  ## Email
  valid email
`(str => str.includes("@") || "Is not email");

const Password = t.String`
  ## Password
  secure password
`(str => str.length >= 8 || "To short password");

export const send = t.Contract`
  ## Send credentials
  > Can not authorize admins
`(Email, Password, t.Unknown).enforce(() => {
  /*...*/
});
```

Контракты

Светлое будущее

- Автогенерация моков, property-base тестирование
- SSoT модели данных
- Вырезать избыточные проверки

Спасибо

@artalar

Материалы

Дополнительные материалы

- https://github.com/microsoft/tsdoc
- https://typedoc.org/guides/doccomments/
  https://github.com/getify/TypL
- https://medium.com/@gcanti/refinements-with-flow-9c7eeae8478b
- https://github.com/ianstormtaylor/superstruct#why
- https://github.com/eigenmethod/mol/blob/master/data/readme.md#units
- https://docs.pact.io/