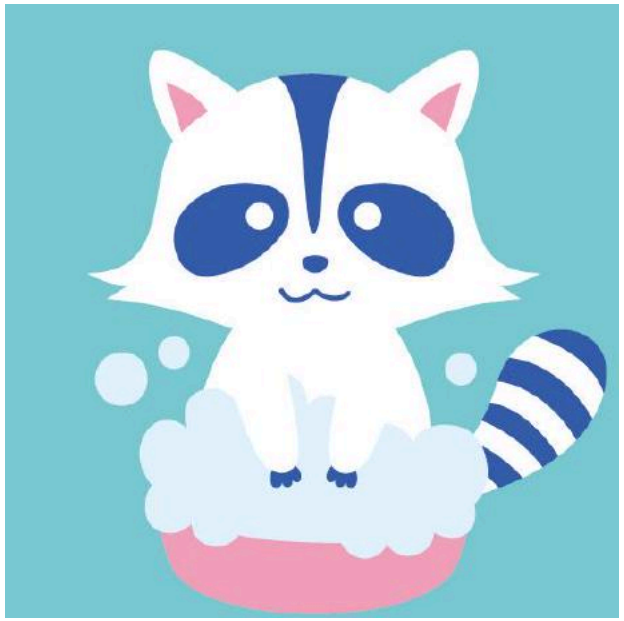
VitestのIn-Source Testingが便利

2025-04-23 Mita.ts #5

taro(@taroro_tarotaro)

自己紹介

- taro(@taroro_tarotaro)
- ベースマキナのソフトウェアエンジニア
- TypeScript, Go, GraphQL, etc.



目次

- VitestのIn-Source Testingとは
- 嬉しいポイント
  - テストのためのexportが不要になる
  - プライベート関数にテストが書ける
  - AIとの相性が良い
- まとめ

VitestのIn-Source Testingとは

その名の通り、実装と同じファイルにテストが書けるやつ

```
export function add(...args: number[]) {
  return args.reduce((a, b) => a + b, 0);
}

// ソースコード内にテストが書ける
if (import.meta.vitest) {
  it("add", () => {
    expect(add()).toBe(0);
    expect(add(1)).toBe(1);
    expect(add(1, 2, 3)).toBe(6);
  });
}
```

https://vitest.dev/guide/in-source

VitestのIn-Source Testingとは

わりと普通に動く

- 各種vitestの関数
- Snapshotテスト
- モック

嬉しいポイント

テストのためのexportが不要になる

テストのために関数をexportしてコメントを書く…🙃

```
// export for test
export const swap = <T>(data: T[], i: number, j: number) => {
  // dataのi番目とj番目を入れ替える処理
  // ...
};
```

```
import { swap } from "./swap";

describe("swap", () => {
  it("basic", () => {
    expect(swap(["a", "b", "c"], 0, 1)).toEqual(["b", "a", "c"]);
  });

  it("out of range", () => {
    // ...
```

テストのためのexportが不要になる

テストのためのexportが不要！🥰

```
const swap = <T>(data: T[], i: number, j: number) => {
  // dataのi番目とj番目を入れ替える処理
  // ...
};

if (import.meta.vitest) {
  it("swap: basic", () => {
    expect(swap(["a", "b", "c"], 0, 1)).toEqual(["b", "a", "c"]);
  });

  it("swap: out of range", () => {
    // ...
}

const useSwap = (initialData: Props) => {
```

テストのためのexportが不要になる

テストのために定数をexport...🙃

```
// export for test
export const userStatus = {
  // ...
} as const;

export const checkUserStatus = (user: User): UserStatus => {
  if (user.lastLoggedIn === null) {
    return userStatus.NEW;
  }
  if (user.lastLoggedIn > 30) {
    return userStatus.ACTIVE;
  }
  return userStatus.INACTIVE;
};
```

テストのためのexportが不要になる

テストのための定数や型などのexportが不要になる！☺️

```ts
const userStatus = {
  // ...
} as const;

export const checkUserStatus = (user: User): UserStatus => {
  // ...
};

if (import.meta.vitest) {
  describe("checkUserStatus", () => {
    test.each([
      // テスト内で定数が使える！
      { lastLoggedIn: null, expected: userStatus.NEW },
      // ...
```

プライベートな関数にテストが書ける

```
const swap = <T>(data: T[], i: number, j: number) => {
  // dataのi番目とj番目を入れ替える処理
};

const useSwap = (initialData: Props) => {
  // swap関数を使って、データを入れ替える
};

export const Swapper: FC<Props> = (initialData) => {
  const { data, swapData } = useSwap(initialData);
  // ...
};
```

exportしている関数やコンポーネントに、まとめて全てのケースのテストを書く…🙃

```
describe("Swapper", () => {
  // ...
```

プライベートな関数にテストが書ける

プライベートな関数に細かくテストを書ける！🥰

```
const swap = <T>(data: T[], i: number, j: number) => {
  // dataのi番目とj番目を入れ替える処理
  // ...
};

if (import.meta.vitest) {
  it("swap: basic", () => {
    expect(swap(["a", "b", "c"], 0, 1)).toEqual(["b", "a", "c"]);
  });

  it("swap: out of range", () => {
    // ...
}

const useSwap = (initialData: Props) => {
```

プライベートな関数にテストが書ける

条件分岐で色んな関数を呼ぶ関数に全てのケースのテストを書く…🙃

```
export const dataConverter = (data: Data) => {
  switch (data.type) {
    case "type1":
      return convertType1(data);
    case "type2":
      return convertType2(data);
    // ...
  }
};
```

```
describe("dataConverter", () => {
  // 全てのtypeのテストをdataConverterに対して書く…
  // ...
});
```

プライベートな関数にテストが書ける

プライベートな関数に細かくテストを書ける！🥰

```
const convertType1 = (data: Data) => {
// ...
if (import.meta.vitest) {
  it("convertType1", () => {
    // ...
  });
}
// ...

// この関数のテストは薄くできる！
export const dataConverter = (data: Data) => {
  switch (data.type) {
    case "type1":
      return convertType1(data);
    // ...
```

AIとの相性が良い（気がする）

AI Agentを使っていると、特に指示しなくてもテストに気づくことが多い（気がする）

- テストの内容も情報源として、実装をしてくれる（気がする）
- テストの修正を指示しなくていい（気がする）

AIとの相性が良い（気がする）

AI Agentを使っていると、特に指示しなくてもテストに気づくことが多い（気がする）

- テストの内容も情報源として、実装をしてくれる（気がする）
- テストの修正を指示しなくていい（気がする）

AIが理解しやすいってことは人間も理解しやすいはず…！

まとめ

- テストのためのexportが不要になる
- プライベートな関数にテストが書ける
- AIとの相性が良い（気がする）

ぜひIn-Source Testing使ってみてください！