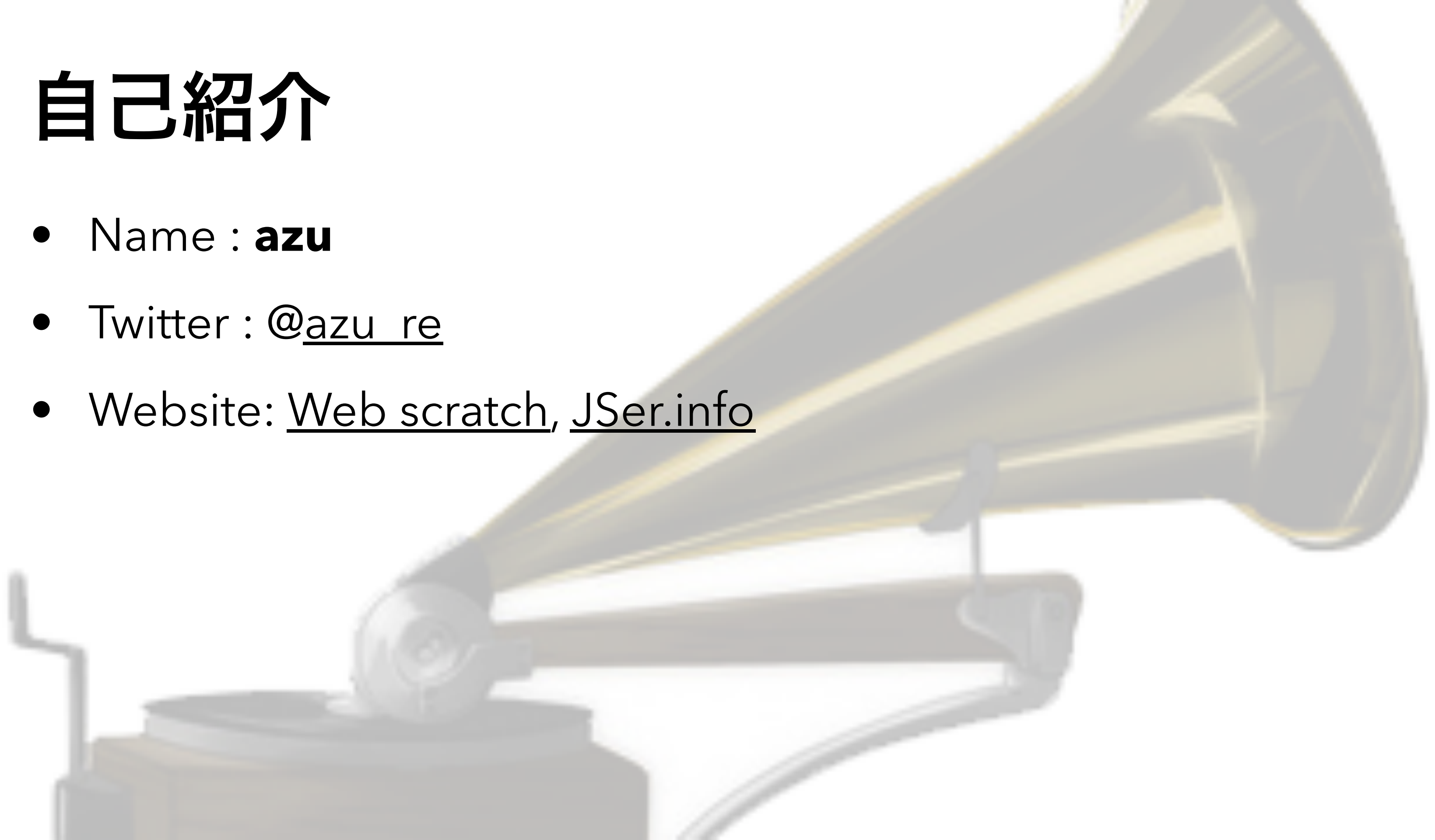


ロジック、E2E、描画、
音、動画、Example、文
章 - 色々なJSテスト

自己紹介

- Name : **azu**
- Twitter : @azu_re
- Website: Web scratch, JSer.info



ロジックテスト

解決したい問題

- コードのロジックを確かめたい

解決する方法

- Mocha、Jasmineなどでユニットテストを書く
- よくあることなので

E2Eテスト

解決したい問題

- 動いてるサイトのコードを変更したい
- だが手元にそのサイトを動かす手順がない
- 動いてる本番のサイトはある

解決する方法

- node-CocProxy + Karma
- KarmaにproxyとしてCocProxyを挟む
- ローカルのコードを本番サイト上のものとしり替える
- ProtractorでE2Eテストを書く

画像のテスト

解決したい問題

- 画像同士を差異をチェックしたい

解決する方法

- js-imagediffやBlink-Diffでdiffを取る

描画のテスト



解決したい問題

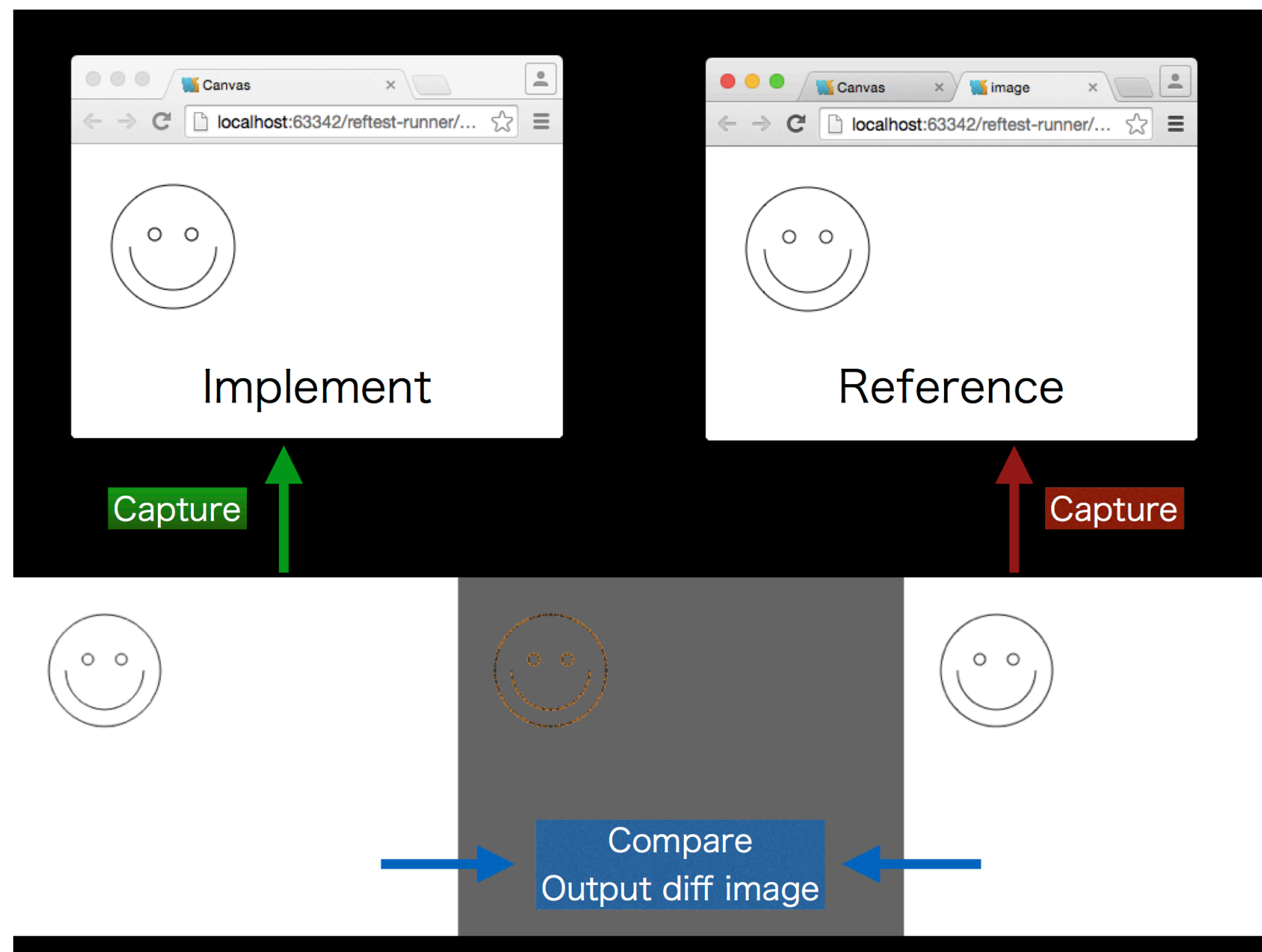
- 画像同士の比較はメンテコストが高い
 - 画像をイチイチ作るのが大変
- HTMLを表示して描画の結果から比較したい

解決する方法

- ReftestによりHTMLの描画結果を画像比較する


Reftest

- Reftest-runnerを作りReftestを行った
- ブラウザでビジュアルテストをする
reftest-runnerを作った | Web Scratch



Reftestからの学び

- Reftestで作成するHTMLはサンプルとしても動作する
 - デバッグに役立つ
- Canvasを使ったライブラリの互換テストとして役立った
- Runtime Errorの検知に実際に動かすテストは役立つ
- 実際に動かした時に色々なログを出すことが重要

音声のテスト 

解決したい問題

- Web Audio API、HTML Audio等の再生ができてるかをテストしたい

解決する方法

- 実際にテストで音声を再生して確認する
- テストに音が鳴り始める

動画のテスト



解決したい問題

- `<video>` 要素のライブラリのテストを書きたい

解決する方法

- 実際にブラウザで動かしてテストする

動画のテスト例 🎥

- <https://github.com/azu/video-prefetcher>
- <https://github.com/azu/video-transcript-tracker>

テストとFeature Detect

- 音声や動画はPhantomJSなどレガシーブラウザは動かない
- 動かない環境がない前提としてテストが必要になる
- テストにもFeature Detectを導入する

video-prefetcher-test.jsより

// 動画をサポートしてるか判定

```
function isSupportVideo() {  
    var video = document.createElement("video");  
    var playType = video.canPlayType('video/mp4; codecs="avc1.42E01E"');  
    if (!playType) {  
        return false;  
    }  
    return playType.length > 0;  
}  
  
var describe = isSupportVideo() ? window.describe : window.describe.skip;  
describe("Videoのテスト", () => { /* ... */ });
```

Feature DetectとDeferred test

- サポートしていない環境ではテストをスキップする
 - Mochaでは`.skip`、Jasmineでは`xdescribe`など
- Buster.jsのFeature detectionがこの機能を持っている
- 備考: PhantomJSで動くようにするより、実際のブラウザをCIで動かしたほうが良い

requestAnimationFrame
meのテスト

解決したい問題

- requestAnimationFrameでメインループを書いたけどテストできない
- requestAnimationFrameはモックすることできない
- setTimeoutなどはtickable-timerでモック出来るが、performance.nowのようなdelta値を制御が必要になる

解決する方法

- requestAnimationFrameでメインループを管理する機構を作る
- uupaa/Clock.js
 - Home · uupaa/Clock.js Wiki をベースに作成
- テストはtimeStamp, deltaTimeの値を元に行う
- Clock.jsはdeltaTimeを制御出来る仕組みを持ってる

解決までの道

- テストに何が必要なのかをモックをしながら調べる
- モック出来ない限界まできたら、そこを抽象化できる仕組みをまとめる
- `requestAnimationFrame(fn);` は `fn` に `timestamp` を渡してくれる
- 毎回呼ばれる `timestamp` 同士の差分 `delta` 値のみが大事だと分かる

- まず、コアは一つの機能に集中+外からConfigurableに設計
 - コアは、ひたすらrequestAnimationFrameでループを回す+外から渡された関数に(timestamp, delta)を渡す事だけに集中
- コアを実際のアプリに合わせた形に変換する層を作る
 - メインループ -> FPSにあわせたメインループ と調整して使う
 - コアでFPSの調整などを機能として入れてはいけない

なぜコア -> 変換層 -> アプリなのか

- 変換層はコアから貰ったdelta値を元に調整を行う
 - Dateなど他の時間軸には依存させない
- コアのdelta値を偽装する機能を追加する
- -> 変換層もdelta値を元に動くのでテストが書ける

通信のテスト

解決したい問題

- Socketを繋いで動くようなアプリで、実際につながってるのかをテストしたい

解決する方法

- 実際にテスト時にSocketにつながるようにしてテストを動かす
- テストサーバをbefore-testで立てて、after-testでサーバを落

Exampleテスト

解決したい問題

- テストが面倒なものをテストしたい
- gulp plugin等パターン化されたもの、テストが書きにくいものなど

解決する方法

- 実際に動くexampleを書いてexampleを実行する

Exampleテストのパターン

- 例) azu/kantan-ej-dictというnpmモジュール
- JSONをnpm install時にダウンロードして、
`require("kantan-ej-dict")`でJSONがとれるだけ
- テストコードは殆ど書く意味がないレベル
- `example/` ディレクトリにサンプルを作る
- `npm i -S ../` でローカルモジュールとして読み込む

Exampleテストのパターン

*example/example.js*という感じで動くサンプルコードを書く

```
var assert = require("assert");  
var dict = require("kantan-ej-dict");  
assert(typeof dict === "object");
```

- サンプルコードから相対パスがなくなる!

```
"script": { // npm testでサンプルコードを実行する  
  "test": "(cd example && npm i && node test.js)"  
}
```

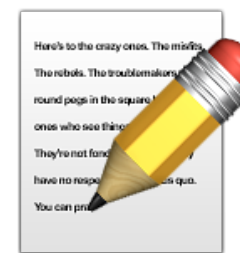
Exampleテストのいいところ

- package.jsonの設定ミスとかをチェック出来る
 - "main" の間違いとかがある場合、example.jsが失敗する
- exampleのコードがコピペフレンドリーになる
- ロジックテストが難しいものでも、動くサンプルコードを書くのは難易度が下がる

Example テストを使ったもの

- [azu/fly-textlint](#)
- [azu/reftest-runner](#)
- [azu/kantan-ej-dict](#)
- [azu/electron-zip-packager](#)
- [azu/video-transcript-tracker](#)

文章のテスト



解決したい問題

- typoを減らしたい
- 用語の統一性を持ちたい

解決する方法

- [azu/textlint](#)
- [JavaScriptでルールを書けるテキスト/Markdownの校正ツール textlint を作った | Web Scratch](#)

使ってる場所

- jser.info/blob/gh-pages/tests/lint-text-content.js

設定のテスト



解決したい問題

- Jekyll記事のカテゴリなどの設定ミスを減らしたい

解決する方法

- globで対象ファイルを取得、front-matterでメタ情報取得テストする
- jser.github.io/tree/develop/test

リリース(前)のテスト

解決したい問題

- npm publishのミスを減らしたい
- private moduleを間違ってpublicしたくない

解決する方法

- npm publishのラッパを使う
- [npm publishのパターン | Web Scratch](#)

まとめ

- exampleテストのようなテストも結構不安は解消される
- refestのような実際に動くサンプルが残るという副産物が便利
- selenium-webdriverなど結構気軽に使えるので、ブラウザを動かすテストも色々書ける

まとめ

- コードにもテストのしやすい設計が必要
 - ひたすらモックしていき、コアに必要なものを探る
 - ひたすらConfigurableに作り、コアに必要なものだけに絞る