

textlintから学んだこと

自己紹介

azu

@azu_re

Web scratch, JSer.info



アジェンダ

- What is textlint?
 - CLIツールの設計思想
- ASTのLintの仕組み
- モジュールに分けるべし
 - インターフェースとしての問題
 - ドキュメントとしての問題

textlint とは何か

- MarkdownやテキストをLintするツール
- ESLintのテキスト版！
- MarkdownやテキストをASTにしてチェックする
- チェックルールをJavaScriptで書いて簡単に追加出来る

textlintの設計

- LintはASTのKeyをEmitして行う
- そこそここにかい固まり
- 最初はESLintのフォークみたいな感じ
- 不要な機能を削ったり、モジュール化してる

textlintの構造

- CLI
 - I/Oに関係することのみ
- TextLint-Engine
 - textlintのラッパ
 - 複数のファイルを扱う抽象レイヤー
- textlint

Lintの仕組み

1. Markdown or TextをASTに変換
2. ASTは**TextNode**というインターフェースを持つ
 - 例えば、`node.type`が"Header"という種類
 - `node.raw`にテキストの中身、`node.loc`に行番号等の位置
 - `txtnode.d.ts`に定義してある

Lintの仕組み - 2

- ルールスクリプトは`node.type`ごとにイベントを受け取るような書き方をする

```
exports[context.Syntax.Link] = function (node) {  
  // Link nodeの時にこのチェック関数が呼ばれる  
  // 問題があったらcontext.report()で報告する  
};
```

- `textlint`は`node`の`type`毎にルールにチェックをお願いする
 - `api.emit(node.type, node)` という感じ

Markdown -> AST

Markdown

```
1 # Header
2
3 text string
```

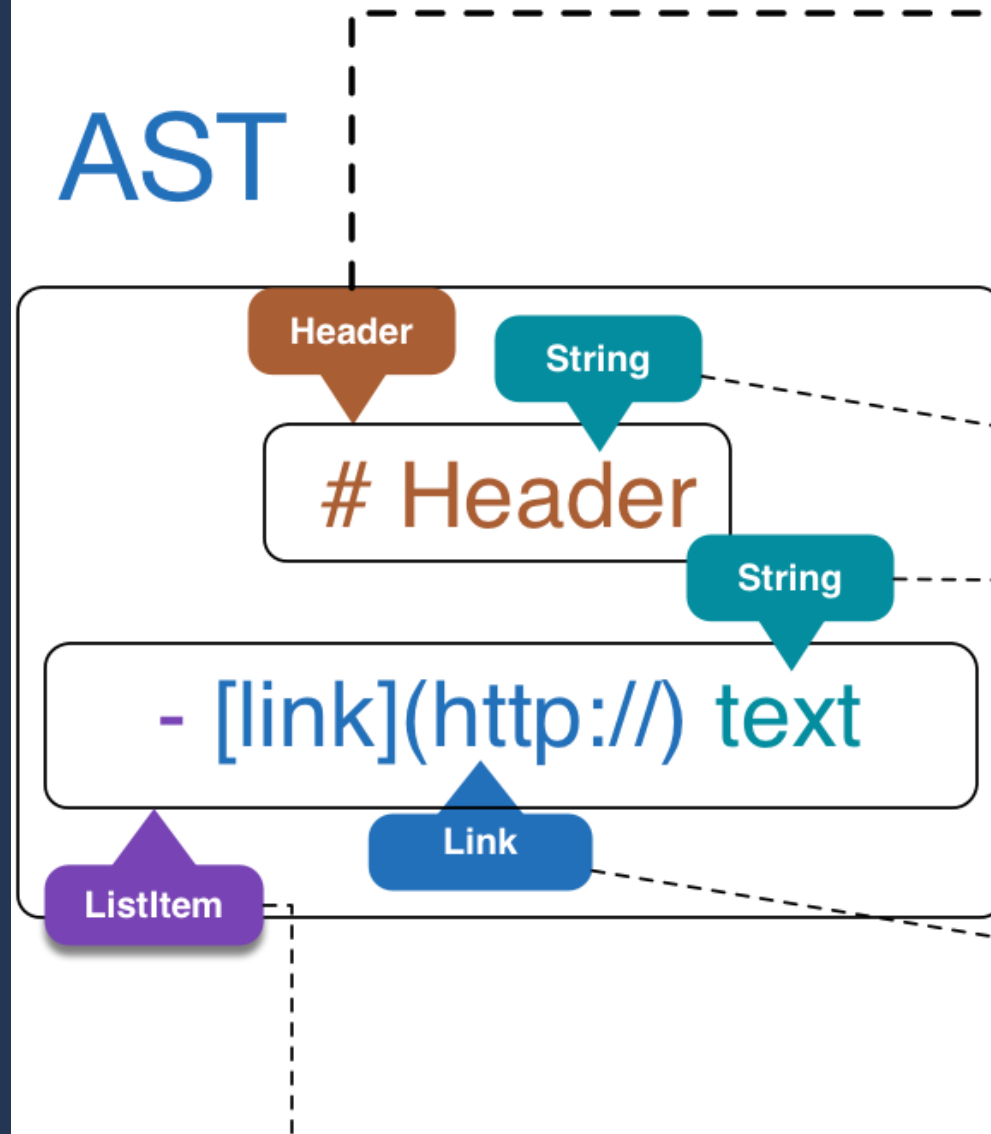
パース

Textlint AST

```
1 {
2   "type": "Document",
3   "raw": "# Header\n\ntext string",
4   "loc": {
5     "start": {
6       "line": 1,
7       "column": 0
8     },
9     "end": {
10      "line": 3,
11      "column": 11
12    }
13  },
14  "range": [
15    0,
16    21
17  ],
18  "children": [
19    {
20      "type": "Header"
21      "raw": "# Header",
22      "inline_content": [
23        {
24          "raw": "Header",
25          "loc": {
26            "start": {
27              "line": 1,
28              "column": 2
29            },
30            "end": {
31              "line": 1,
32              "column": 8
33            }
34          }
35        }
36      ]
37    }
38  ]
39 }
```

Rule script

AST



```
1 module.exports = function (context) {  
2   var exports = {};  
3   exports[context.Syntax.Header] = function (node) {  
4     // Header node come here  
5   };  
6   exports[context.Syntax.Str] = function (node) {  
7     // Str node come here  
8   };  
9   exports[context.Syntax.ListItem] = function (node) {  
10    // ListItem node come here  
11  };  
12  exports[context.Syntax.Link] = function (node) {  
13    // link node come here  
14  };  
15  return exports;  
16 };
```

Lintの仕組み - 3

- `context.report()` で報告されたエラーを textlint-formatter に
- 文字列を組み立ててくれるのでそれを標準出力へ流す or ファイルとして吐く

```
❌ azu@azu ~/Dropbox/workspace/node/lib/textlint master • node ./bin/textlint.js README.md -f pretty-error
found Todo: Todo: quick fix this. /Users/azu/Dropbox/workspace/node/lib/textlint/README.md:5:0
  ↓
4.
5. Todo: quick fix this.
6.
  ^

found Todo: - [ ] to /Users/azu/Dropbox/workspace/node/lib/textlint/README.md:8:0
  ↓
7. - list 1
8. - [ ] todo
9.
  ^
```

仕組みの仕組み

- textlintとルールスクリプトの関係は pub/sub
- ルールスクリプトはやってくるnodeだけを考えればLintを書ける
- やってくるnodeの流れは木構造を走査する形 - txt-ast-traverse
- ルールが疎結合なので、自由にルールを追加出来る！

```
1 # visualize-txt-traverse
2
3 This is a part of [azu/textlint](https://github.com/azu/textlint "azu/textlint").
4
```

Enter

Leave

Both(Enter/Leave)

[Permanent link](#)

Star

1

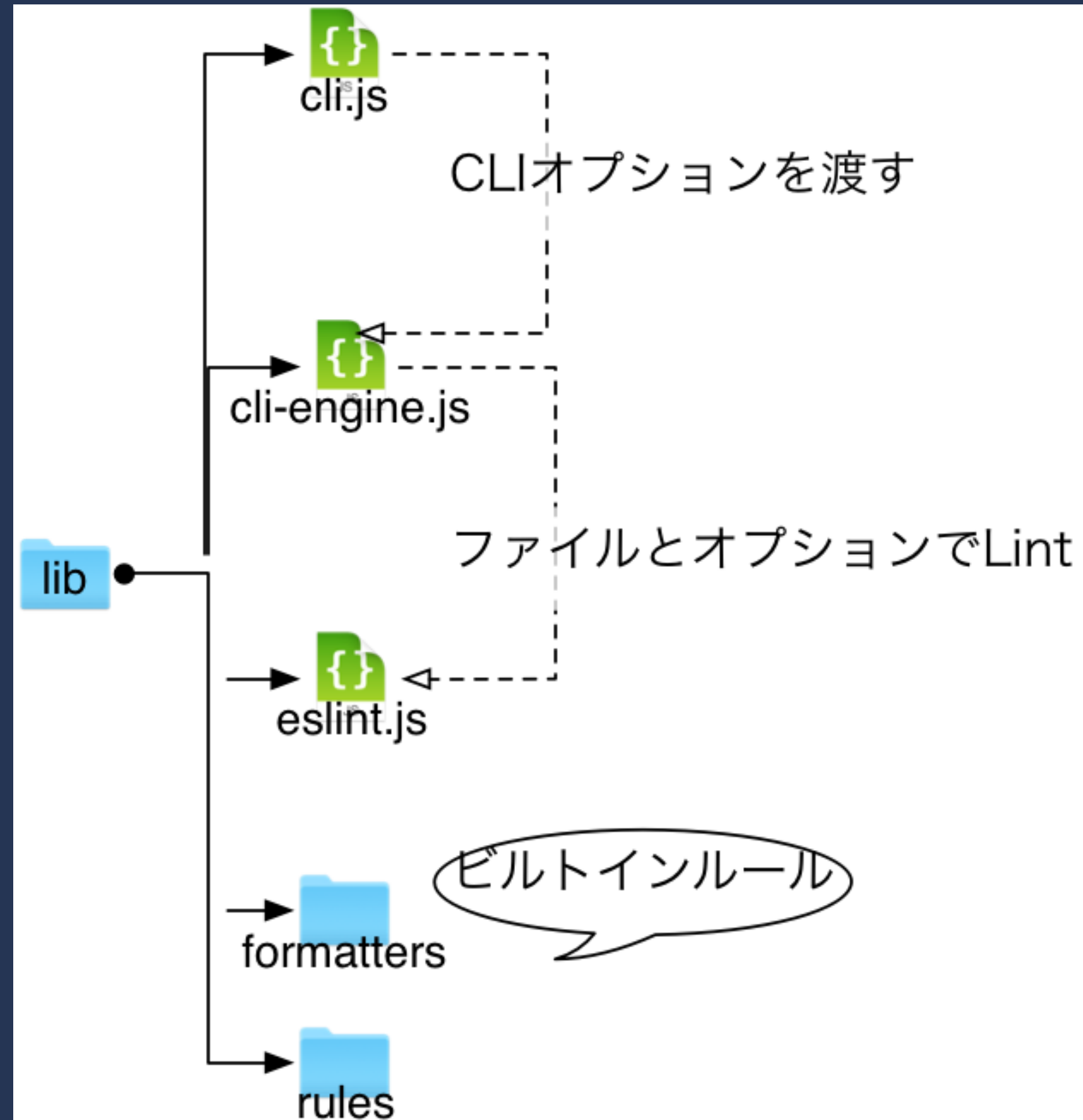
- This is a parts of [azu/textlint](#)
- [markdown-to-ast: online parsing demo](#)
- [txt-to-ast: online parsing demo](#)

textlintとESLint

- textlintはESLintのfork的な作りで始めた
- ESLintの構造をそのままもってきて分解していった

ESLintの構造

- cli.js
 - コマンドライン引数の処理
- cli-engine.js
 - 設定の読み込み、Core APIのラッパ、複数のファイルを扱ったり
- eslint.js(Core)
 - 実際にemitしたりLintするAPIを持つ



ESLintの公開API

- package.jsonの"main"でexportsされてるモジュールだけでは、微妙に届かないAPIが存在する

```
module.exports = {  
  linter: require("./eslint"),  
  cli: require("./cli"),  
  CLIEngine: require("./cli-engine")  
};
```

ESLintの公開APIの問題

- [adamestry/gulp-eslint](#)みたいなラッパを書く場合に、
formatterを正規な方法で取得できない
- `require('eslint/lib/config');` という感じで読んだり
- `node_modules` から自分で取り出す必要がある(ESLintの内部で
同じ事やってる)

どこまで公開APIか

- 何でも公開APIにする?
 - 面倒だし、すぐに壊しそう
- => それぞれの機能を別モジュールとして公開すれば自然と解決する
 - formatter、parser、traverse
- 同様のやり方: [twada/power-assert](https://github.com/twada/power-assert)

textlintの場合

- azu/textlint-formatter を別モジュールとして出してる
- azu/markdown-to-ast と azu/txt-to-ast パーサも分離してる
- azu/txt-ast-traverse Traverseも分離してる

```
module.exports = {  
  cli: require("./lib/cli"),  
  TextLintEngine: require("./lib/textlint-engine"),  
  textlint: require("./lib/textlint")  
};
```

公開APIはどこまで公開API?

- 目安としたもの:
- gulpプラグインを書く場合に、CLIと同じことをやるのにモジュールを使うだけでできるか?

モジュールとドキュメントの分離

ドキュメントも分離すべきか？

- ソフトウェアがでかくなる = ドキュメントがでかくなる
- 新規ユーザーはどこから見ればいいのか分からない
- ユーザーの種類を分けてドキュメントを分ける
 - コマンドラインアプリとして使うユーザー (READMEに入れる)
 - Nodeモジュールとして使う開発者 (docs/に入れる)

モジュールのドキュメント

- モジュールとして使うのは大体開発者
- 分離するかはライブラリの種類によってケースバイケース
- 類似APIなら一枚岩のドキュメントの方が検索できて便利?
- 単体として完全に分離できてるならそのモジュール毎にドキュメントがあったほうがいい
 - 索引だけはプロジェクトからリンクされてると良さそう

まとめ

- どのAPIを公開するか迷ったら、それより小さいモジュールに分けて別途公開できないかを考えよう
- コード共にドキュメントも肥大化する
 - モジュールとして分けるとドキュメントも分けやすい
 - ただ、利用者が迷子にならないように整理が必要
 - 小さいモジュールを大量に作った場合の問題も別にある
 - [Modularizing Underscore.js | &yet Blog](#)

課題。

READMEとAPI

- READMEにAPIの詳細をずらっと書くとあんまり読みやすい
ない
 - JSDocそのままREADMEに落とした感じのとか
- 適度に情報を間引く必要がありそう
- サンプルコードの方が視認性は高い

JSDocとd.tsの使い分け

- d.tsはオプションオブジェクトの定義が楽に出来る
 - 型定義的には使いやすい
 - 逆に全てのインターフェースの定義はだるい
- JSDocは関数の説明はしやすい
 - ドキュメントとしてはd.tsよりもやりやすい
 - [Javadoc ドキュメンテーションコメントの書き方 - Qiita](#)

JSDocとd.tsの使い分け

- [azu/textlint-formatter](https://github.com/azu/textlint-formatter)
- d.tsとJSDocが混在してる
- 書きやすいけど、解釈出来るものがWebStormぐらいしかなさそう...
- TypeScriptのASTが公開されたらd.ts周りのツールが充実するかもという希望的観測

不完全なd.ts配布の問題

- モジュールを細分化していくとインターフェースの共有をしなくなる
- d.tsを使いたい => けど何で配布する?
- d.tsがあるだけのモジュールをつくるべきなのか?
 - 全てのAPIについて定義してるわけじゃなくて完全にInternalの利用