



textlint - Linterの作り方

自己紹介

- » Name : **azu**
- » Twitter : @azu_re
- » Website: Web scratch, JSer.info
- » Book: JavaScript Primer



アジェンダ

- » Linterを作る人向けの話
- » Linterの考え方とアーキテクチャを一致させる
- » 使うものを作るう、作るために使おう



textlint

textlint とは

- » 自然言語(日本語や英語など)に対するLinter
- » MarkdownやHTMLなどのマークアップ言語に対応している
- » ビルトインのルールは0
- » 利用できるルールは100以上ある
- » 多くのルールはオフラインで動くので、外部に入力してる内容を送信しない
- » CI/CDに組み込める自然言語のチェッカー(表記揺れ、スペルチェック、誤用、読みやすさのチェックなど)

README.md — js-primer

EXPLORESOR

OPEN EDITORS

JS-PRIMER

- comments
- condition
- data-type
- date
 - README.md
- ecmascript
 - README.md
- error-try-catch
- function-declaration
- function-scope
- function-this
- implicit-coercion
 - img
 - README.md 2, M
- introduction
- json
- loop
- map-and-set
- math
- module
 - src
 - test
 - README.md 9+
- object
- operator
- other-parts
- prototype-object
- read-eval-print
- statement-expression
- string
- string-unicode
- variables
- wrapper-object
 - README.md 9+
- cheatsheet

OUTLINE

TIMELINE

CELL TAGS

```
source > basic > implicit-coercion > README.md > # 暗黙的な型変換 {#implicit-coercion} > ## 明示的な型変換 {#explicit-coercion} > ### NaNはNot a NumberだけどNumber型 {#n
354 ↓
355 {{book.console}}↓
356 ```js↓
357 const x = 10;↓
358 const y = x + NaN;↓
359 const z = y + 20;↓
360 console.log(x); // => 10↓
361 console.log(y); // => NaN↓
362 console.log(z); // => NaN↓
363 ```↓
364 ⚠️
365 `NaN`はNumber型的一种であるという名前と矛盾したデータに見えます↓
366 ↓
367 {{book.console}}↓
368 ```js↓
369 // NaNはnumber型↓
370 console.log(typeof NaN); // => "number"↓
371 ```↓
372 ↓
373 NaNしか持っていない特殊な性質として、自分自身と一致しないというものがあります。↓
374 この特徴を利用することで、ある値が`NaN`であるかを判定できます。↓
375 ↓
376 {{book.console}}↓
377 ```
```

![] Aa ab .* No results ↑ ↓ ≡ ×

"に" が連続して2回使われています。 (ja-technical-writing/ja-no-successive-word) textlint(ja-technical-writing/ja-no-successive-word)

View Problem (F2) No quick fixes available

PROBLEMS 3K+ OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, !**/node_modules/**)

- README.md source/basic/implicit-coercion 2
 - × "に" が連続して2回使われています。 (ja-technical-writing/ja-no-successive-word) textlint(ja-technical-writing/ja-no-successive-word) [Ln 365, Col 34]
 - × 文末が"。"で終わっていません。 (ja-technical-writing/ja-no-mixed-period) textlint(ja-technical-writing/ja-no-mixed-period) [Ln 365, Col 38]
- OUTLINE.md source/basic/introduction 32
- README.md source/basic/introduction 1
- README.md source/basic/json 1
- README.md source/basic/loop 16
- README.md source/basic/module 18
- OUTLINE.md source/basic/object 10
- README.md source/basic/object 4
- README.md source/basic/operator 4
- OUTLINE.md source/basic/other-parts 32
- OUTLINE.md source/basic/prototype-object 1
- README.md source/basic/prototype-object 1

Ln 365, Col 35 (1 selected) Spaces: 4 UTF-8 LF Markdown textlint ESLint 4.8.4 Build Tree

preset-japanese/no-doubled-joshi: 一文に二回以上利用されている助詞 "が" がみつかりました。

/Users/azu/.ghq/github.com/azu/promises-book/Ch4_AdvancedPromises/race-delay-timeout.adoc:201:43

v

200. 一つの関数で全てやろうとしてるのがそもそも良くないので、

201. ひとつの関数で何でもやるのは止めて、以下のように関数に分離していくというのが妥当な気がします。

202.

^

preset-japanese/no-mix-dearu-desumasu: 本文: "である"調 と "ですます"調 が混在

=> "である XHR" がである調

Total:

である : 3

ですます: 7

/Users/azu/.ghq/github.com/azu/promises-book/Ch4_AdvancedPromises/race-delay-timeout.adoc:9:12

v

8. これを利用すると簡単に出来ますが、複数のXHRを束ねたタイムアウトや他の機能でも応用が効くため、

9. 分かりやすい非同期処理であるXHRにおけるタイムアウトによるキャンセルを例にしています。

10.

^

textlint users

» Translation:

Angular、React、Vue、Nuxt.js、Next.js、Gatsby

» Book:

JavaScript Primer、SurviveJS - Webpack、Pythonクローリング&スクレイピング

» Documentaion:

VuePress、Cypress、Microsoft Azure Identity(ja)、OWASP Cheat Sheet Series

» 小説:

「Visual Studio Code」で執筆するSF作家 藤井太洋氏が作る物書きのための拡張機能

» Company:

弁護士ドットコム、ソラコム、ヴェルク、SmartHR、Retty

» textlintを使っている企業の事例・ルールをまとめてみた

Editorの連携

» VScode

» [vscode-textlint](#)

» [テキスト校正くん - Visual Studio Marketplace](#)

» Vim

» 秀丸エディタ

» [秀まるおのホームページ\(サイト一企画\) - textlintを処理するマクロサンプル](#)

textlintをなぜ作ったか

- » JavaScript Promiseの本やJSer.infoなどを書いていて表記ゆれが気になることがあった
- » 汎用的な自然言語をチェックできるものが欲しくなった
- » CIで走らせることが前提の自然言語のLinterがなかった
- » 詳細:

[Maintainer Month: なぜtextlintを作ったか | Web Scratch](#)

自然言語は常に変化する言語

- » 言語の違い、文章のフォーマット、コンテキストで文章は変わる
- » 日本語と英語でそもそも文法のルールが異なる
- » 自然言語は、毎日新しい単語が増えるし¹、文法も変わることもあるし²、同じ文の意味がコンテキストや時間で変わることもある³
- » そのため、あらゆる自然言語を包含するAST(抽象構文木)を作ることは難しい

¹[mecab-ipadic-NEologd](#)、[SudachiDict](#)

²[変化する日本語 - 日本語ジャーナル](#)

³[日本語における差別語概念の変遷](#)、[ドメインにより意味が変化する単語の抽出](#)

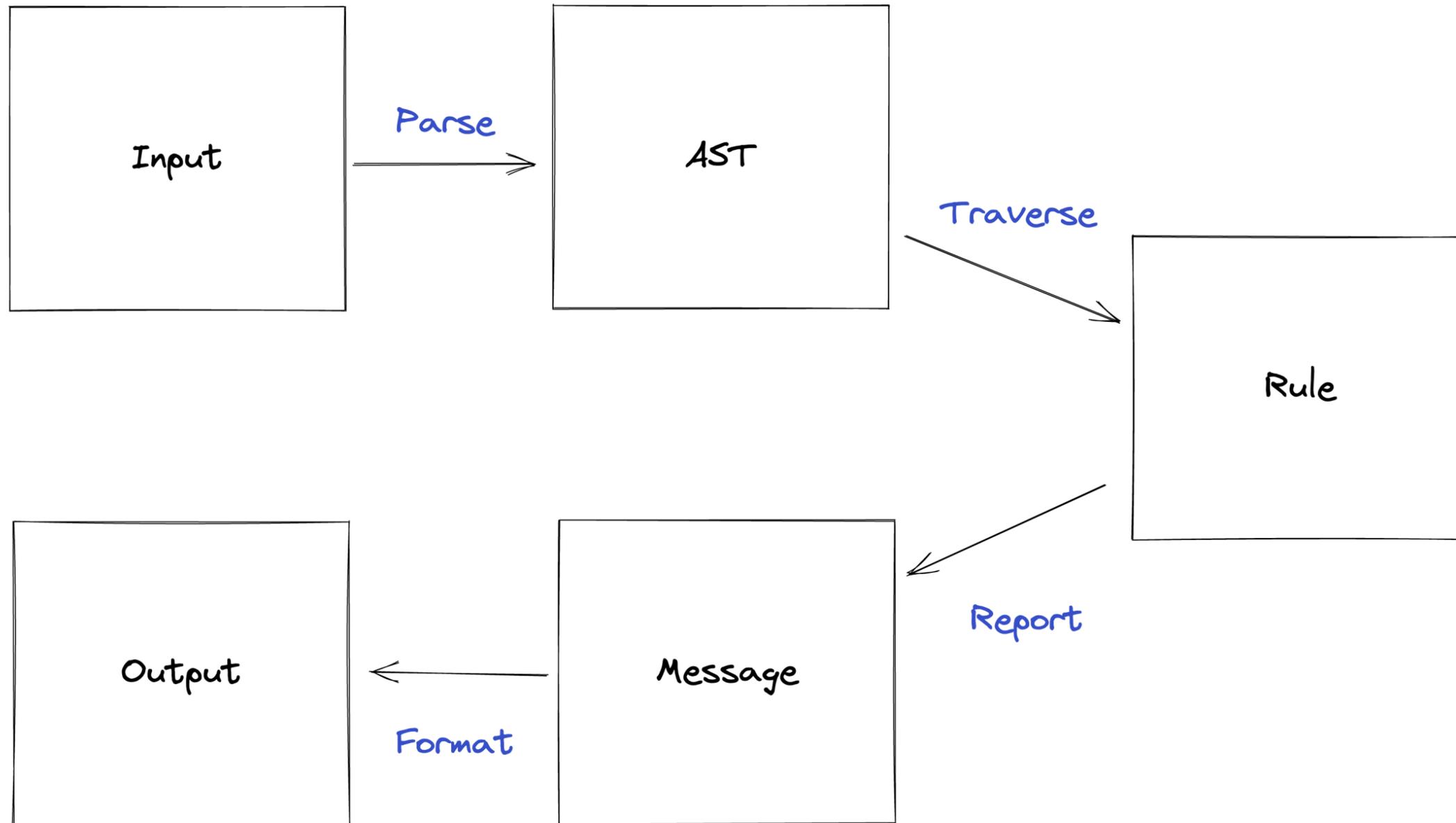
文字列のチェックでは、誤検知を避けられない

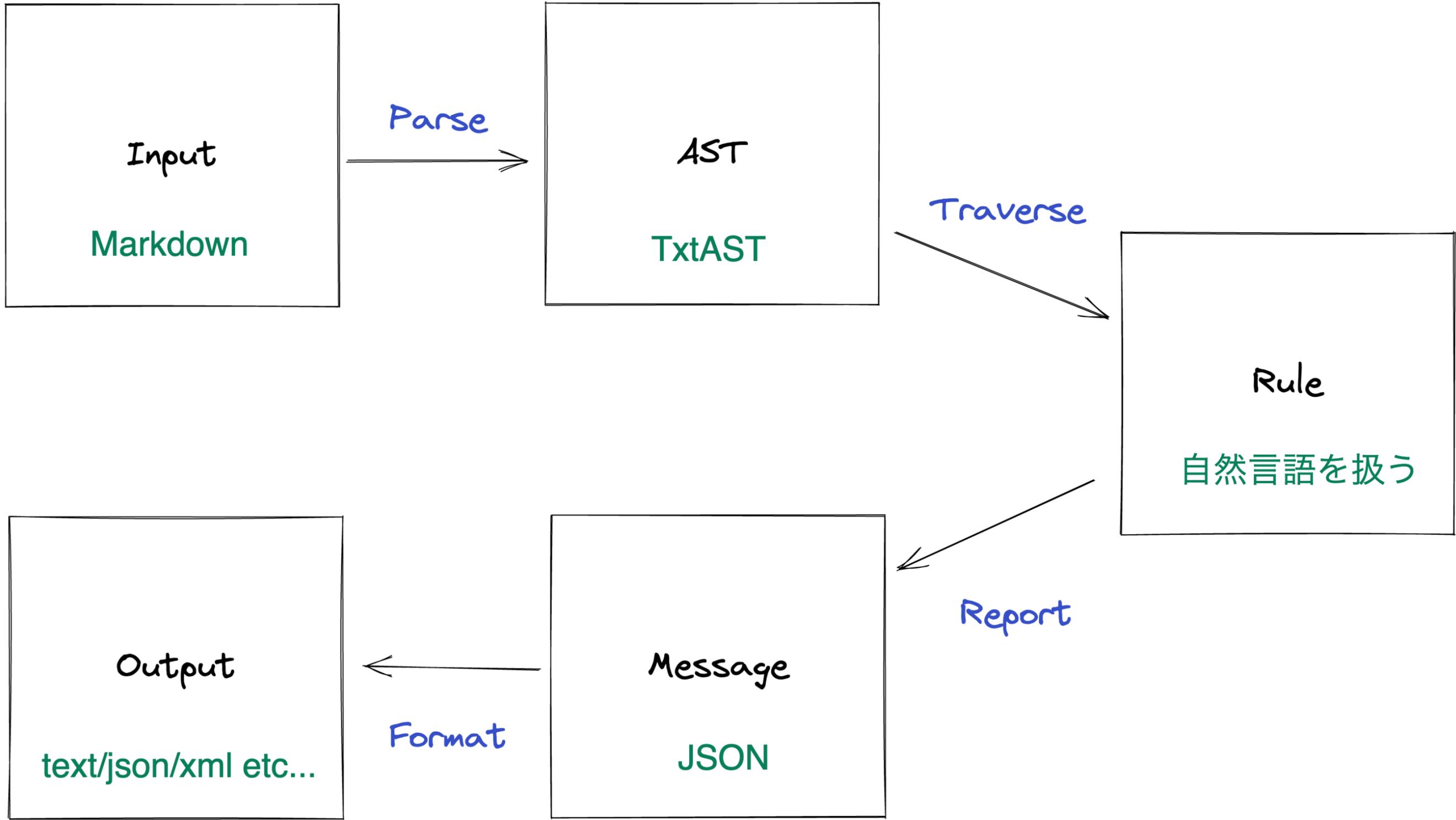
- » 多くの場合、MarkdownやLaTeXやWordといった装飾をしながら書いている
- » 何もパースせずにそのまま文字列として扱おうと誤検知が生まれる
 - » “文字数”をみるのもMarkdownとプレーンテキストでは異なる
 - » @とか特定の記号が特別な意味を持つことがある
 - » 文章の中でも、コードと引用文とは異なる扱いにしたい
- » マークアップはパースして扱わないと、それ自体が誤検知となる
 - » マークアップはパースするが、その中のテキスト(自然言語)はパースできない
- » Grep以上、人間未満のツールが必要だった

textlintの目標とアーキテクチャ

- » あらゆるマークアップ、あらゆる自然言語を扱えることを目標にした
- » そのために、次のようなアーキテクチャを選択した
 - » すべてがプラグブル → マークアップ言語が増えても対応できる
 - » ビルトインルールは持たない → ルールはオプトイン
- » コアはどのマークアップ、自然言語でも動く

textlintのアーキテクチャ





Input

Header

THIS IS TODO.

AST^{Txt}AST

```
{
  "type": "Document",
  "children": [
    {
      "type": "Header",
      "depth": 1,
      "children": [
        {
          "type": "Str",
          "value": "Header",
        }
      ],
      ...
    }
  ]
}
```

Rule

```
export default (context, options = {}) => {
  const { Syntax, RuleError, report, getSource, fixer } = context;
  return {
    [Syntax.Code](node) {
      const text = getSource(node);
      const match = text.match(/TODO/);
      if(match){
        report(node, new RuleError("Found TODO")); // ⇒ Message
      }
    }
  };
};
```

Message

```
[  
  {  
    "messages": [  
      {  
        "type": "lint",  
        "ruleId": "example",  
        "message": "Found TODO",  
        "range": [19, 24],  
        "severity": 2  
      }  
    ],  
    "filePath": "/path/to/A.md"  
  }  
]
```

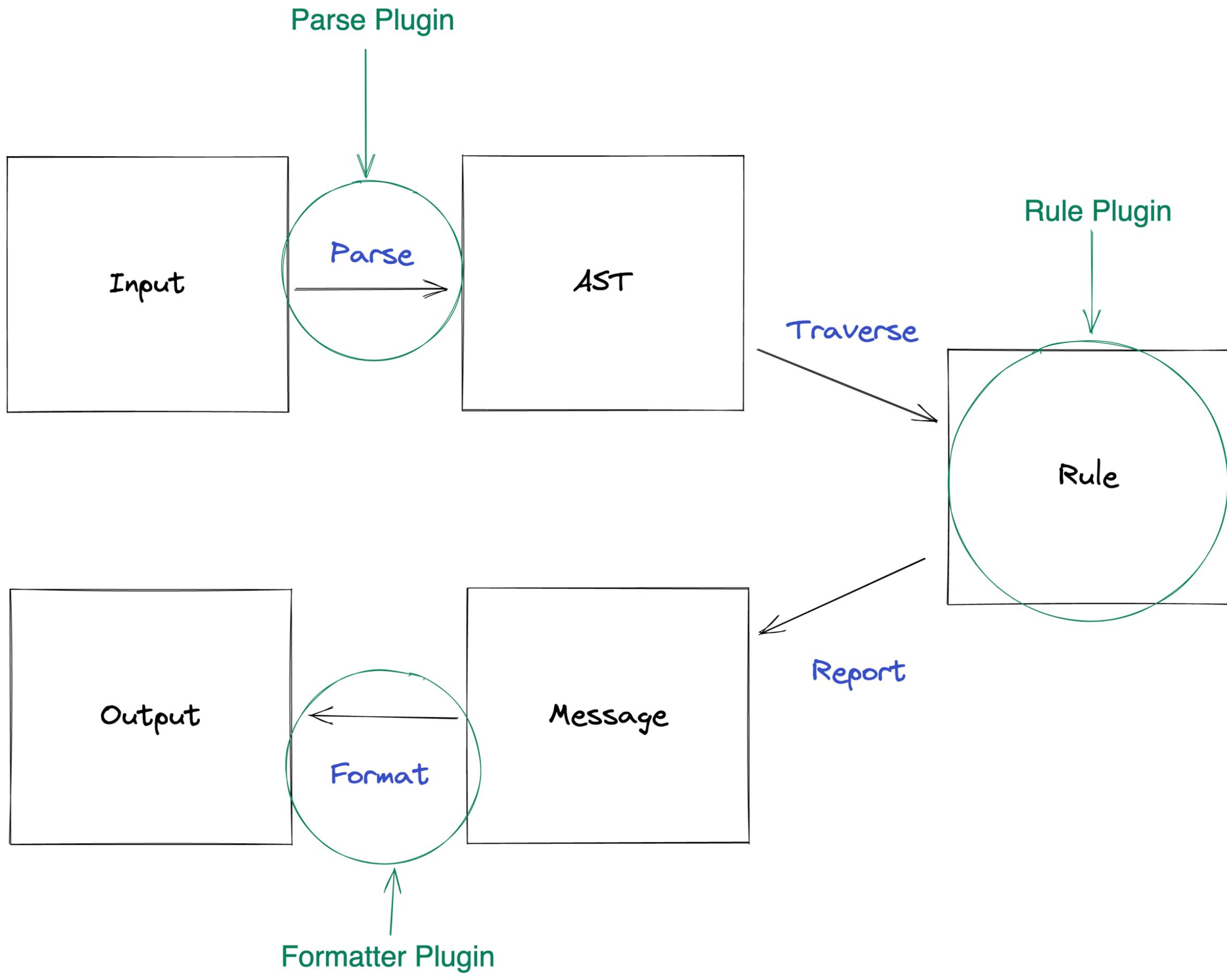
Output

Found TODO

(at 3:1)

THIS IS TODO

| ^^^^



プラガブルアーキテクチャのメリット

- » コアがマークアップ言語や自然言語の依存から切り離せる
 - » 新しいマークアップ言語に対応する → Parserプラグインを実装する
 - » 検出するルールを増やしたい → ルールを実装する
 - » 新しい自然言語を扱いたい → ルール内で扱えば問題ない
 - » 新しい出力に対応したい → Formatterを実装する
- » ParserとRuleとFormatterは全部プラグインとしてかける
 - » textlintが決めているのはASTとMessageのフォーマットだけ
- » これによって、マークアップ/自然言語が増えても対応できる
 - » 🚀🌟絵文字言語が登場しても対応できるように作るというのが目的

プラガブルアーキテクチャのデメリット

» アーキテクチャ視点

- » レイヤーが増える分、パフォーマンスは良くない
- » 実装が分散するため、まとめてアップデートはしにくい

» ユーザー視点

- » セットアップに手間がかかる
- » 何を使えばいいのかわからない問題
- » textlintもpresetは提供してる
- » がそのpresetもコンテキストが多いので、これ使えばいいというのは難しい

プラガブルなアーキテクチャで生まれる余白

- » 極めて自由度が高めで作ると想定していない使い方も増える
 - » プロダクトの表記揺れ統一
 - » [textlintを導入してみた / Usage status and effects of the textlint - Speaker Deck](#)
 - » UX ライティング
 - » [入力支援ツール・textlintを導入したら、テキストコミュニケーションを磨く時間をたくさんかけられるようになった話 | BASE DESIGNER | note](#)
 - » コミュニケーションの補助ツールとして
 - » [boardの顧客サポート返信時にtextlintでチェックする - ヴェルク - IT起業の記録](#)

ブラウザで動く textlint

- » textlintは、ブラウザでも動かせるように意識はしてる
- » これはクライアントサイドで動かして、サーバサイドにテキストデータを送らなくても動くようにするため
- » [textlint editor - ブラウザでも動く Privacy Firstの文章校正ツールを作る話](#)
- » 既に[@textlint/script-compiler](#)を使うと、ブラウザで動くようにtextlint+ルールをコンパイルできる
- » ESM + Import Mapsへネイティブ対応すれば、コンパイルなしでもルールが動く未来は見てきた
- » かなり大掛かりなことやらないと互換性が保てないし大変なので、Contributing Welcome !
- » <https://github.com/textlint/textlint/issues/902>

プラグインと責務の分離

責務の分離

- » あらゆる点をプラグインにすることは責務の分離にもつながる
- » textlintはコアにルールを持たないのがルール
- » そのため、ルールはコアとは別のOrganizationで公開してる
- » ルールの要望への対応は時間を使うので、コアに来てても単純にIssueを閉じられるというメリットがある

コアとルールの分離

- » ESLintなどはコアにpresetが紐づいているため、コアの時間がそこに使われる問題がある
- » RomeとかDenoとかもコアにLintを持っているので、それぞれ同じものを労力かけて開発
- » これはパフォーマンスを最適化するには必要だけど、時間は最適化できない
- » 自然言語のLintがbuilt-inされたものは前例があったけど、この辺をプラグインでやりきってるものはない。textlintはパフォーマンスより時間を最適化を意識してる
- » 自分も大量にオープンソースプロダクト持っているので、そこまで時間を使えない
- » コアではやらないことをはっきりさせることで、責務が分離できる

責務の分離とメンテナンスの分離

- » 責務が分散すると、メンテナンスも分散される
- » これはトレードオフ
 - » コアから切り離せると、ルールの追加とかはやりやすくなる
 - » 一方で、ルールが個人に紐づくともメンテナンスが止まる確率は高くなる
- » → ルールをorganizationに集めるなど、緩いまとまりを持ったコミュニティを作ることによってバランスをとっている

コミュニティと責務の統合

- » textlintのコミュニティと責務
- » プラグイン = パーサはコアに近い(新しいマークアップ言語の対応)なので、textlint organization 内でプラグインコミュニティを作ってメンテナンスをサポートしてる
 - » textlint plugin ownership · Discussion #1 · textlint
 - » プラグインはコアに近いものなので、コアの近くに置いている(権限は工夫すればなんとかなる)
 - » コアにあることによって、プラグインの開発メンバーもコアと同じ恩恵を受けられる
 - » textlint memberは1password🔑を無料で利用できるなど
 - » 1Password for Open Source Projectsの申請をした | Web Scratch

textlintルールのコミュニティ

» textlint-ruleとtextlint-ja

- » 自然言語を扱うルールはかなり言語に依存してるので、Universalなものと日本語で分けている
- » 日本語じゃないとIssueが書きにくい問題も多いというのもあるため
- » コミュニティを作って、メンテナンスできる人を増やす意図がある
- » ルールを開発する時間がなくなったら権限を委譲することで、アップデートは継続できるようにする

Lintor開発のコツ

開発を続けるコツ

- » 使うものを作るう
- » 作るために使おう
- » 使わないものは使われない
- » 自分が使っていないものはそこまで時間を使えない
 - » HTMLのパーサは最初コアにあったけど、自分は使うユースケースを持ってなかった切り離してプラグインにした
- » [Maintainer Month: オープンソースをメンテナンスするコツ | Web Scratch](#)

使うと開発は進む

- » ユースケースがないと、なかなか進まない
- » ユースケースを作ったり持ってきたりする
- » JavaScript Primerを書いている時はtextlintの開発も進む
- » JavaScript Primerの第2版を書いているが、textlintがなかったら文章のリファクタリングは難しかった

Secretlint の事例

- » Secretlintはファイルに含まれてる機密情報をチェックするLintルール
- » 構造はtextlintとかなり似ていて全部プラグインになってる
- » 機密情報は共通知識があるので、presetは別途用意してる
- » 開発するために、常に使う状況を作る
 - » global pre-commit hooksでどのリポジトリのコミットでも使ってる
 - » 常にLintを使うようにすることで、新しいルールを追加するモチベーションになる
- » すぐ使える状況を作る
 - » <https://secretlint.github.io/>
- » 使うものじゃないとクオリティが上がらない

リアルなデータでLintをチェックする

- » 新しいルールを書いたときに、想定していないケースに遭遇することはよくある
- » 実際のリアルデータを使って、誤検知していないかを確認する
- » textlintで技術文書向けのルールを書くときは、技術書を集めたコーパスを使ってチェックできる
 - » [textlint-ja/technological-book-corpus-ja](#): 日本語で書かれた技術書を収集した生コーパス/ツール
- » プログラミング言語なら、GitHubなどでコーパスを作って持っておくとチェックの負荷が下がる

まとめ

- » textlintは自然言語を扱うLint
 - » 自然言語の解釈はとても変化しやすいので、”公式のルール”は作るのが難しい
 - » textlintでは、この考えを反映してプラグブルなアーキテクチャを選んだ
- » プラグブルを基本とすることで、責務の分離や想定外の使い方も生まれる
 - » 責務を分離するとコードも分離し、メンテナンス性が悪くなることもある
 - » textlintでは、コードは分離するがコミュニティで集める方式をとっている
 - » pluginはコアに近いコミュニティとして管理、ruleは言語ごとにコミュニティを作って管理
- » 人間が作るものにはモチベーションの問題がある
 - » 長く続けるには、使うものを作ること/使うために作ること
 - » 積極的に使うパターンを増やすと、開発は継続しやすい

参考

- » [Reflections on ESLint's success - Human Who Codes](#)
- » [The inception of ESLint - ESLint - Pluggable JavaScript Linter](#)
- » [r2c blog — Semgrep: a static analysis journey](#)
- » [Maintainer Month: なぜtextlintを作ったか | Web Scratch](#)

余り

分散と集約はどちらが良いのか

- » どちらもいい時がある
- » ESLintは分散で、Deno/ESLint/Romeなどは集約してる
- » 集約することでパフォーマンスのメリットはあるけど、どちらかという機能じゃなくて組織的/マーケティング的な方向も入ってくる
- » 集約して勢いがあるうちは早くていいが、量が増えると問題ができてきて分散/Forkに向かっていく
- » また分散すると維持コストは高くなり、労力的な問題が出てくるため、集約していくというのを繰り返す
- » バランスの問題
- » 維持コストをみて、常に方向を決めている

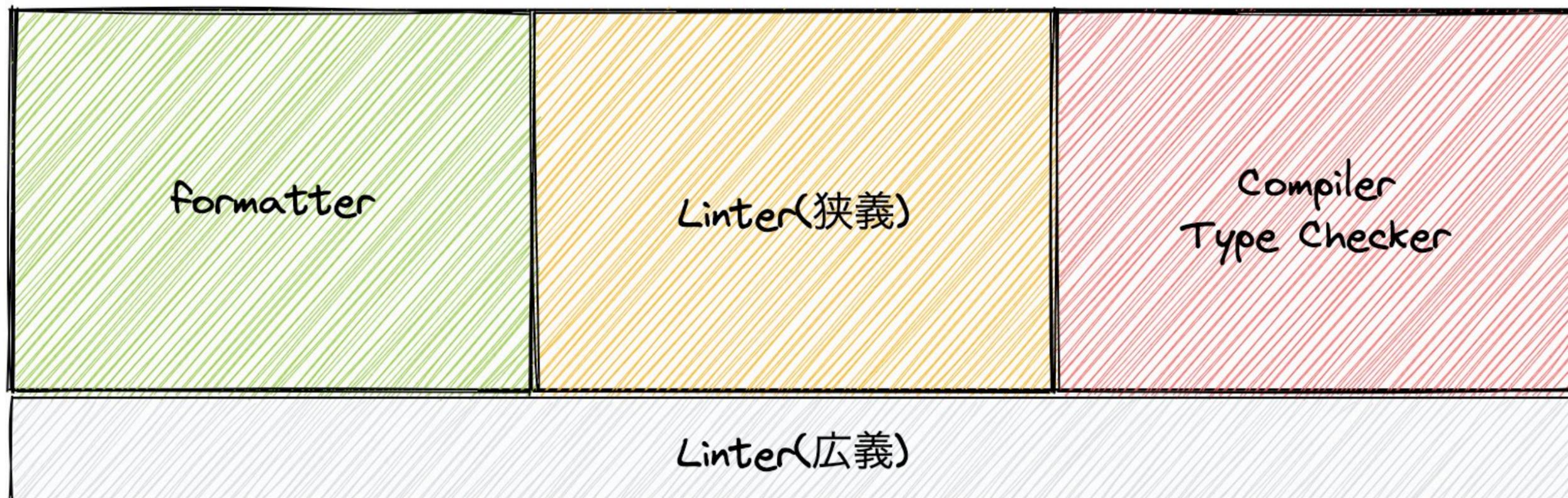
世の中のLint

<https://github.com/github/super-linter>

☰ README.md

Dart	dartanalyzer
Dockerfile	hadolint
EditorConfig	editorconfig-checker
ENV	dotenv-linter
Gherkin	gherkin-lint
GitHub Actions	actionlint
Golang	golangci-lint
Groovy	npm-groovy-lint
HTML	HTMLHint
Java	checkstyle / google-java-format
JavaScript	ESLint / standard js
JSON	eslint-plugin-json
JSONC	eslint-plugin-jsonc
Kubernetes	kubeval
Kotlin	ktlint
LaTeX	ChkTex
Lua	luacheck
Markdown	markdownlint
Natural language	textlint
OpenAPI	spectral
Perl	perlcritic
PHP	PHP built-in linter / PHP CodeSniffer / PHPStan / Psalm
PowerShell	PSScriptAnalyzer
Protocol Buffers	protolint
Python3	pylint / flake8 / black / isort
R	lintr

Lintの幅



- » 動的な言語などは、LintがType Checkなども扱っていることがある
- » Style(Formatter)とLinterの役割分担は曖昧になりやすい(実装的に仕組みが違うので本当は別)
- » [Technical Details · Prettier](#)

false positiveはopt-in

- » 過剰なエラーを出すツールは、過剰に終わって避けられる
 - » 怪しいものを全部Warnで表示すれば、デフォルトを増やせるという考え方もある
 - » これは、全てが無視される
 - » 回避方法として常にユーザーに選択させるという方法がある。スペルチェッカーやgrammarlyはこれを選んでいる
 - » ただし、CIで回すツールとは相性が悪い(インタラティブにして回避できないため)
 - » textlintやSecretlintがopt-inを選択したのはこれが理由
 - » 一方でopt-inはonionatedとは逆になる
 - » ルールはたくさんあるけど、あくまでユーザーが選択して入れる
 - » 使えるセキュリティツールがかなり少ないのもこれが理由
 - » CodeQLとかCode Scanningレベルのことやらないと担保できない