TypeScript による Web シンタックスハイライト

2025 年 1 月 1 日

1 概要

本プロジェクトでは，TypeScript で Web ページ用のシンタックスハイライト (コードの色付け) を作成する．また，Node.js やその他ツールについての理解を深めることも本プロジェクトの目的である．

本プロジェクトではブログにおけるソースコードの自動スタイリングを目的とし，プログラムを作成した．これは React や Next.js への移行を想定したものであり，これを利用すれば効率的な自作ハイライティングを実装できる．

今回はシンタックスハイライトの機能実装に加え，リアルタイムにハイライトの様子を確認できるようリッチテキスト風のテキストエディタをフロントエンドとして実装していく．

本プロジェクトにおける実行環境を以下に示す．

- Ubuntu 20.04.4 LTS
- TypeScript 5.7.2
- Node.js 20.11.1
- npm 10.4.0
- webpack 5.97.0

その他パッケージについては，使用する際に別途記述していく．

2 実行環境及びツールの概要

2.1 Node.js とは

Node.js とは，JavaScript の実行環境である (Python をインストールすると Python を開発・実行できることと同じ)．従来 Web ブラウザ上でしか実行できなかった JS を，PC やサーバ上でも実行するために開発された．

サーバサイドの JS 実行環境と呼ばれることが多いが，あくまでも「JS の実行環境」なので，クライアントサイドの開発にも利用できる．(HTML にスクリプトを読み込んでブラウザで実行せずとも，クライアントサイドの開発が可能になる．)

Node.js の利点の 1 つとしては，オープンソースのパッケージを `npm` でインストールして効率的な開発が可能になる点である (Python における `pip` と同じ)．これにより，`<script>`タグでライブラリのロードを行わずとも，コマンド 1 つで利用できるようになる．

Node.js のインストール方法は以下の通りである．

```
$ sudo apt install nodejs
```

利用の目的としては，主に以下のものが挙げられる．

(1) 新仕様の JS/TS での開発

JS や TS の最新仕様で開発した際，ブラウザがその仕様に対応していないことがある．この問題を解決するため，旧仕様へのコンパイル (トランスコンパイル) を行う必要があるのだが，主要なトランスコンパイラである Babel の実行環境として Node.js を用いることが多い．

(2) Web アプリケーションの作成

Node.js は Web サーバの役割も果たすことができるため，Rails のように Web アプリを開発することができる．この場合，実行環境は Node.js，言語は JS/TS，フレームワークは Express や Next.js がよく用いられる．

その他にも，webpack や vite 等のバンドラを利用する際や，Sass のコンパイルのために用いることもある．

2.2 npm とは

npm とは，Node.js におけるパッケージ管理ツールである．コマンド 1 つでパッケージをインストールしたり，バージョンの管理を行うことができる．

2.2.1 主要なコマンド

(1) プロジェクトの開始 (package.json の生成)

```
$ npm init
```

-y オプションを付けると，プロジェクトの初期設定情報を記した package.json が作成される．

(2) パッケージのインストール

```
$ npm install [package name]
```

-g オプションを付けるとグローバルにインストールでき，どのディレクトリからもパッケージを利用できる．

また，-D or --save-dev オプションを付けると，開発環境でのローカルインストールになる．この場合，package.json の dependencies ではなく devDependencies に追記される．例えば Git からクローンした後に必要なパッケージをインストールする場合，npm i --production とすると，そのパッケージ群はインストールされない．そのため，開発環境依存のパッケージ (webpack 等) は-D オプションを付けるべきである．

(3) パッケージのアンインストール

```
$ npm uninstall [package name]
```

グローバルにインストールしたパッケージの削除には-g オプションが必要になる．

(4) npm **及びパッケージのアップデート**

```
$ npm install -g npm@latest  # Update npm to latest version.
$ npm update [package name]
```

グローバルにインストールしたパッケージの更新には-g オプションが必要になる．

(5) **パッケージの一覧表示**

```
$ npm list
```

グローバルにインストールしたパッケージの表示には-g オプションが必要になる．

2.2.2 package.json

Node.js では，package.json を用いてプロジェクトにインストールされているすべてのパッケージを効率的に管理している．

例えばインストールしたパッケージ及びその依存パッケージは node_modules 以下に格納されるが，Git にプッシュする際はこのディレクトリを除外することが多い．しかし，インストールしたパッケージや依存関係を記した package.json さえダウンロードすれば，同ディレクトリ上で npm install を実行することですべてのパッケージを一括インストールできる．

図 1 は本プロジェクトにおける package.json である．

図 1 package.json

```
{
  "name": "web-syntaxhighlighter",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "webpack serve --mode development",
    "build": "webpack build --mode production"
  },
  "keywords": [],
  "author": "Yuto Matsuda",
  "license": "ISC",
  "devDependencies": {
    "@fortawesome/fontawesome-free": "^6.7.1",
    "css-loader": "^7.1.2",
    "html-webpack-plugin": "^5.6.3",
    "mini-css-extract-plugin": "^2.9.2",
    "sass": "^1.82.0",
    "sass-loader": "^16.0.4",
    "ts-loader": "^9.5.1",
    "typescript": "^5.7.2",
    "webpack": "^5.97.0",
    "webpack-cli": "^5.1.4",
    "webpack-dev-server": "^5.1.0"
  }
}
```

package.json は，そのプロジェクトを npm パッケージとして公開するという目線で読むと分かりやすい．例えば name はパッケージ名となる．

インストールしたパッケージは dependencies に記述されている．なぜ「依存関係」として記述するかと

いうと，当プロジェクトをパッケージとして公開する場合に，一緒にインストールしてもらう必要があるものだからである．

`script` は npm-script と呼ばれ, スクリプト名とシェルスクリプトの組で定義する．`npm [script name]` としてコマンドを実行すると，定義したシェルスクリプトを実行できる．

2.3 webpack とは

webpack とは，複数の JS ファイルや CSS，画像等を 1 つの JS ファイルにまとめるモジュールバンドラである．モジュール分割による開発効率向上だけでなく，1 つのファイルにまとめることで HTTP リクエストの数を減らすことにも繋がる．また，JS ファイルの圧縮やローカルサーバの起動等，フロントエンドにおける開発環境が webpack 一つで整う点も特徴である．

インストール方法は以下の通りである．

```
$ npm i -D webpack webpack-cli webpack-dev-server
```

`webpack-cli` は webpack を CUI 操作するためのツールで，ver4.0 以降から必要なものである．`webpack-dev-server` はローカルサーバの起動や，ソースの変更を検知 (watch) しビルドの自動実行とリロードを行うツールである．

エントリポイントと呼ばれるメインの JS ファイル (TS や JSX ファイルも可) を基準に複数ファイルがバンドルされ，1 つの JS ファイルが作成される．それを HTML で読み込むことで，クライアントサイドでの動作を確認できる．

webpack の利用には `webpack.config.js` の利用が一般的であり，詳しくは第 3 章で解説する．

3 環境構築

本プロジェクトにおける環境構築の手順を示す [7].

3.1 プロジェクトの開始

まず，以下のコマンドでプロジェクトの初期化及び開発環境の構築に必要なパッケージのインストールを行う．

```
$ npm init -y
$ npm i -D webpack webpack-cli webpack-dev-server
$ npm i -D typescript ts-loader
$ npm i -D mini-css-extract-plugin css-loader sass sass-loader
$ npm i -D @fortawesome/fontawesome-free
$ npm i -D html-webpack-plugin
```

3.2 スクリプトの作成

次に，以下の npm-script を用意する．

```
"dev":   "webpack serve --mode development",
"build": "webpack build --mode production"
```

なお，`--mode` オプションの概要は以下の通りである．

- development: ソースマップが作成され，ビルド結果に付加される．
- production: ビルド結果を圧縮して生成する．

3.3 webpack の設定

次に，webpack.config.js を図 2 のように作成する．

図 2 webpack.config.js

```
1  const MiniCssExtractPlugin = require('mini-css-extract-plugin');
2  const HtmlWebpackPlugin = require("html-webpack-plugin");
3
4  module.exports = {
5      // entry point
6      entry: {
7          main: `${__dirname}/src/ts/main.ts` // __dirname: current directory
8      },
9
10     // configuration of output files.
11     output: {
12         path: `${__dirname}/dst`,
13         filename: '[name].js' // In this time, [name] = main.
14     },
15
16     resolve: {
17         extensions:['.ts','.js'], // TS and JS are treated as module.
18         alias: {
19             '@': `${__dirname}/src/`,
20         },
21     },
22
23     devServer: {
24         // Open 'dst/index.html' automatically.
25         static: {
26             directory: `${__dirname}/dst`,
27         },
28         open: true,
29     },
30
31     module: {
32         rules: [
33             {
34                 // Apply TS compiler to files ending with '.ts'.
35                 test: /\.ts$/,
36                 loader:'ts-loader'
37             },
38             {
39                 test: /\.(scss|sass|css)$/,
40                 use: [
41                     MiniCssExtractPlugin.loader,
42                     'css-loader',
43                     'sass-loader',
44                 ]
45             }
46         ]
47     },
48
49     plugins: [
```

```
50        new MiniCssExtractPlugin(),
51        new HtmlWebpackPlugin({
52            template: `${__dirname}/src/index.html`,
53            inject: 'body', // insert <script> to just before <body>
54            scriptLoading: 'defer'
55        }),
56    ]
57 }
```

設定のポイントをいくつかまとめる．

まず，resolve の extensions の指定により，JS ファイルと TS ファイルをモジュールとして扱うことを明示している．例えば，module1.js をインポートする際，import func from 'module1' のように，拡張子を省略することができる．これは，module1 という名前の JS ファイルもしくは TS ファイルをモジュールとして探すことを意味する．

また，alias はインポート時のパスの指定を簡単化するために設定している．これにより，絶対パスによるインポートを import foo from '@/ts/molude1' のように記述できる．

モジュールに対するルール設定においては，test に指定したファイルに対して loader もしくは use 配列にしていしたモジュールを適用するよう指定している．use 配列に指定したモジュールは配列の末尾から順に適用される点がポイントである．例えば今回なら，sass-loader で CSS へのコンパイルを，css-loader で JS へのバンドルを行い，mini-css-extract-plugin で style.css として出力する流れとなる．

html-webpack-plugin は HTML を webpack から出力するためのモジュールで，バンドルされた JS ファイルや CSS ファイルの読み込みを意識することなく HTML を記述することができる．

3.4 tsconfig.json の設定

まず，以下のコマンドで tsconfig.json を作成する．

```
$ tsc --init
```

コメントを削除し，パスエイリアスの設定を施した tsconfig.json を図 3 に示す．

図 3 tsconfig.json

```
1  {
2    "compilerOptions": {
3      "target": "es2016",
4      "module": "commonjs",
5      "esModuleInterop": true,
6      "forceConsistentCasingInFileNames": true,
7      "strict": true,
8      "skipLibCheck": true,
9      "paths": {
10       "@/*": ["./src/*"]
11     }
12   }
13 }
```

4 フロントエンドのスタイリング

まず，シンタックスハイライトの動作確認のためにテキストエディタを作成，スタイリングする．

4.1 FontAwesome の設定

第 3.1 節にて Web アイコンの配信サービスである FontAwesome をインストールしたが，エントリポイントで必要なモジュールをインポートする必要がある．そこで，エントリポイント `main.ts` の冒頭に以下の記述を追加する．

```
import '@fortawesome/fontawesome-free/js/fontawesome';
import '@fortawesome/fontawesome-free/js/solid';
import '@fortawesome/fontawesome-free/js/regular';
import '@fortawesome/fontawesome-free/css/all.css';
```

次に，CSS 側から簡単に利用できるよう，以下の `mixin` を作成する．

```
@mixin fontawesome($style: 'solid', $unicode) {
    @if $style == 'solid' {
        font: var(--fa-font-solid);
    }
    @if $style == 'regular' {
        font: var(--fa-font-regular);
    }
    @if $style == 'brands' {
        font: var(--fa-font-brands);
    }
    content: $unicode;
}
```

4.2 背景テーマの作成

背景は宇宙をイメージし，星が散らばるアニメーションを作成した [9]．これは TS による CSS アニメーションの動的な制御によって実装した．

背景デザインに関する HTML，SCSS，TS を抜粋して図 4–図 6 に示す．

図 4 `index.html`(**背景部分抜粋**)

```
<body>
  <div class="bg"></div>
</body>
```

図 5 `bg.scss`

```
.bg {
  background: #000;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  perspective: 500px;
  -webkit-perspective: 500px;
  -moz-perspective: 500px;

  .stars {
      position: absolute;
      top: 50%;
      left: 50%;
      width: 1px;
```

```
        height: 1px;

        .star {
            display: block;
            position: absolute;
            top: 50%;
            left: 50%;
            width: 5px;
            height: 5px;
            border-radius: 100%;
            transform:
              translate(-50%,-50%) rotate(var(--angle))
              translateY(-100px) translateZ(0)
            ;
            background: #fff;
            animation: ScatteringStars 4s var(--delay) linear infinite;
        }
    }
}

@keyframes ScatteringStars {
    from {
        transform:
            translate(-50%, -50%) rotate(var(--angle))
            translateY(-100px) translateZ(var(--z))
        ;
    }
    to {
        transform:
            translate(-50%, -50%) rotate(var(--angle))
            translateY(-75vw) translateZ(var(--z))
        ;
    }
}
```

図 6 bgAnime.ts

```
type StarConfig = {
    angle: number,
    z: number,
    delay: string,
};

export default function bgAnime() {
    const bg = document.getElementsByClassName('bg')[0];
    const stars = document.createElement('div');
    const starsProps: StarConfig[] = [
        { angle: 0, z: -100, delay: '-2.0s' },
        { angle: 30, z: -200, delay: '-1.3s' },
        { angle: 60, z: -10, delay: '-4.2s' },
        { angle: 90, z: -90, delay: '-3.3s' },
        { angle: 120, z: -180, delay: '-2.1s' },
        { angle: 150, z: -300, delay: '-5.3s' },
        { angle: 180, z: -150, delay: '-6.7s' },
        { angle: 210, z: -220, delay: '-1.5s' },
        { angle: 240, z: -250, delay: '-2.4s' },
        { angle: 270, z: -30, delay: '-3.1s' },
        { angle: 300, z: -80, delay: '-5.0s' },
        { angle: 330, z: -120, delay: '-7.1s' },
```

```
    ];

    stars.classList.add('stars');
    bg.appendChild(stars);
    starsProps.forEach(({ angle, z, delay }) => {
        const star = document.createElement('span');
        star.classList.add('star');
        star.style.setProperty('--angle', `${angle}deg`);
        star.style.setProperty('--z', `${z}px`);
        star.style.setProperty('--delay', delay);
        stars.appendChild(star);
    });
}

document.addEventListener('DOMContentLoaded', () => {
    bgAnime();
});
```

TS 側で動的に星を表現する要素 (star) を追加しているのは，HTML を煩雑にさせないためである．星の数や設定値をランダムにすると，星の動きにランダムさを持たせることができる．

4.3 エディタの作成

次に，実際にコードを打ち込むエディタを作成する．ヘッダ部とコンテンツ部に分けて実装し，ヘッダ部では言語の選択が行えるようにする．

エディタの枠組み部分を図 7 及び図 8 に示す．

図 7 index.html(**エディタ枠組み抜粋**)

```
<div class="container">
    <div class="header">
        <!-- header -->
    </div>
    <div class="content">
        <!-- content -->
    </div>
</div>
```

図 8 style.scss(**エディタ枠組み抜粋**)

```
.container {
    display: flex;
    flex-direction: column;
    z-index: 100;
    width: 80%;
    max-width: 800px;
    height: 80%;
    max-height: 500px;
    box-shadow: 0 0 15px 7px rgba(255, 255, 255, 0.5);

    .header {
        // header style
    }

    .content {
        // content style
    }
```

```
}
```

4.3.1 ヘッダ部の作成

エディタのヘッダ部では，ハイライト言語の表示部を用意し，これをクリックすることで言語選択のメニューが表示できるように実装していく．メニューの表示や言語の設定等の制御は TS で行う．

図 9–図 11 にコンテンツ部のデザイン及び制御プログラムを示す．

図 9　index.html(**ヘッダ部抜粋**)

```
<div class="header">
    <button id="lang-btn"></button>
    <div id="config-popup">
        <label class="item">
            <input type="radio" id="html" name="lang">
            <span>HTML</span>
        </label>
        <label class="item">
            <input type="radio" id="css" name="lang">
            <span>CSS</span>
        </label>
        <label class="item">
            <input type="radio" id="scss" name="lang">
            <span>SCSS</span>
        </label>
        <label class="item">
            <input type="radio" id="c" name="lang">
            <span>C</span>
        </label>
    </div>
</div>
```

図 10　style.scss(**ヘッダ部抜粋**)

```
@keyframes openPopup {
    0% {
        opacity: 0;
    } 100% {
        opacity: 1;
    }
}

@keyframes closePopup {
    0% {
        opacity: 1;
    } 100% {
        opacity: 0;
    }
}

.header {
    position: relative;
    height: $code_header-hgt;
    background: #000;
    box-shadow: 0px 5px 10px -5px rgba(0, 0, 0, 0.5);

    #lang-btn {
        margin-left: .5em;
```

```
25          color: #fff;
26
27          &.html::before {
28              content: "< >";
29              display: inline-block;
30              font-weight: 900;
31              font-size: 1.2em;
32              color: #ec9751;
33              transform:
34                  translateY(2px)
35                  scaleX(.6)
36              ;
37          }
38
39          &.css::before {
40              content: '#';
41              font-weight: 900;
42              color: #72c1ff;
43              padding: 0 .5em 0 .75em;
44          }
45
46          &.scss::before {
47              @include fontawesome('brands', '\f41e');
48              font-size: .9em;
49              padding: 0 .25em 0 0.5em;
50              color: #ed5262;
51          }
52
53          &.c::before {
54              content: 'C';
55              font-weight: 900;
56              color: #72c1ff;
57              padding: 0 .5em 0 .75em;
58          }
59      }
60
61      #config-popup {
62          position: absolute;
63          left: 1.5em;
64          z-index: 600;
65          opacity: 0;
66          pointer-events: none;
67          text-align: left;
68          color: #fff;
69          background: #2e3235;
70          box-shadow: 0 2px 3px rgb(0,0,0,.9);
71          border: 1.5px solid #bdbdbd;
72
73          &.open {
74              animation: openPopup .2s forwards;
75              pointer-events: auto;
76          }
77
78          &.close {
79              animation: closePopup .2s forwards;
80              pointer-events: none;
81          }
82
83
```

```
        .item {
            position: relative;
            cursor: pointer;
            display: block;
            width: 100%;
            font-size: .8em;
            padding: .25em .5em 0 1.75em;

            &:last-child {
                padding-bottom: .25em;
            }

            &:hover {
                background: rgba(255, 255, 255, .25);
            }

            &.checked {
                &::before {
                    @include fontawesome('solid', '\f00c');
                    position: absolute;
                    top: 40%;
                    left: 10%;
                }
            }
        }
    }
}
```

図 11　main.ts(**ヘッダ制御部抜粋**)

```
const langBtn = <HTMLElement>document.getElementById('lang-btn');
const langList = <HTMLCollectionOf<Element>>document.getElementsByClassName('item');
const langRadios = <NodeListOf<HTMLInputElement>>document.getElementsByName('lang');
const langConfig = <HTMLElement>document.getElementById('config-popup');

let isOpenLangConfig = false;
let currentLang = 'html';

const toggleLangConfig = () => {
    langConfig.classList.remove(isOpenLangConfig ? 'open' : 'close');
    langConfig.classList.add(isOpenLangConfig ? 'close' : 'open');
    isOpenLangConfig = !isOpenLangConfig;
}

const initRadio = () => {
    const defaultRadio = <HTMLInputElement>document.getElementById(currentLang);
    const defaultLabel = defaultRadio.parentElement;
    if (defaultRadio && defaultLabel) {
        defaultRadio.checked = true;
        defaultLabel.classList.add('checked');
    }
}

const modifyHeaderLangName = (lang: string) => {
    currentLang = lang;
    langBtn.classList.remove(...langBtn.classList);
    langBtn.classList.add(lang);
    switch (lang) {
        case 'html': langBtn.innerText = 'HTML'; break;
```

```
        case 'css': langBtn.innerText = 'CSS'; break;
        case 'scss': langBtn.innerText = 'SCSS'; break;
        case 'c': langBtn.innerText = 'C'; break;
    }
}

document.addEventListener('DOMContentLoaded', () => {
    initRadio();
    modifyHeaderLangName(currentLang);
});

langBtn.addEventListener('click', toggleLangConfig);

[...langList].forEach((elm, i) => {
    elm.addEventListener('click', () => {
        const radio = langRadios[i];
        [...langList].forEach(elm => {
            elm.classList.remove('checked');
        });
        if (radio.checked) {
            langList[i].classList.add('checked');
            modifyHeaderLangName(radio.id);
        }
        codeHighlight();
    });
});
```

このように，言語の初期化に関する処理を TS 側に一任することで，HTML を直接書き換えなくても良い設計になっている．今後ハイライト対象の言語を増やす場合には，HTML，SCSS，TS にその旨の記述を追加し，スタイリングを行えば良い．

4.3.2 コンテンツ部の作成

シンタックスハイライトは入力トークンを適切なクラスを施した span タグで囲むことで実現する．しかし，textarea の入力文字そのものにスタイルを当てることはできないという問題点がある．そこで，**テキストエリア上にコード要素をオーバレイ表示する**というアプローチを取った．しかしこのままでは 2 要素を同時にスクロールできないため，TS で同時にスクロールするよう制御を行う．

図 12–図 14 にコンテンツ部のデザイン及び制御プログラムを示す．

図 12　index.html(**コンテンツ部抜粋**)(オーバレイ)

```
<div class="content">
    <div id="line-num"></div>
    <code id="code" class="overlay"></code>
    <textarea id="text" class="text"></textarea>
</div>
```

図 13　style.scss(**コンテンツ部抜粋**)(オーバレイ)

```
@mixin codeContent($type: 'editor') {
    position: absolute;
    top: $code_content-padding;
    left: calc($code_line-num-width + 1em);
    font-family: $code-font;
    font-size: 1.4rem;
    line-height: 1.4;
    letter-spacing: 0;
```

```
    white-space: pre;
    overflow: auto;
    width: calc(100% - $code_content-padding - $code_line-num-width);
    height: calc(100% - $code_content-padding * 2);

    &::-webkit-scrollbar {
        width: 10px;
        height: 10px;
    }

    &::-webkit-scrollbar-thumb {
        background: #5e6163;
        border-radius: 7px;
        border: 2px solid transparent;
        background-clip: padding-box;
    }

    &::-webkit-scrollbar-corner {
        background: #2e3235;
        border-radius: 7px;
    }

    &::-webkit-scrollbar-track {
        background: #2e3235;
        border-radius: 7px;
        margin: 4px;
    }

    @if $type == 'line-num' {
        left: 0;
        width: $code_line-num-width;

        &::-webkit-scrollbar {
            display: none;
        }
    }
}

.content {
    flex: 1;
    position: relative;
    display: flex;
    background: #2e3235;

    #line-num {
        display: flex;
        flex-direction: column;
        user-select: none;
        width: 3rem;
        color: #bdbdbd;
    }

    .overlay {
        @include codeContent;
        pointer-events: none;
        color: #fff;
    }

    .text {
```

```
68          @include codeContent;
69          resize: none;
70          background: transparent;
71          color: transparent;
72          caret-color: #bdbdbd;
73
74          &::selection {
75              background: rgba(55, 165, 255, 0.3);
76          }
77      }
78  }
```

図 14　main.ts(**コンテンツ制御部抜粋**)(オーバレイ)

```
1  const lineNum = <HTMLElement>document.getElementById('line-num');
2  const code = <HTMLInputElement>document.getElementById('code');
3  const text = <HTMLInputElement>document.getElementById('text');
4
5  text.addEventListener("scroll", synchronizeScroll);
6  text.addEventListener('input', () => {
7      codeHighlight();
8  });
9  text.addEventListener('keydown', (e: KeyboardEvent) => {
10     if (e.key === 'Tab') {
11         e.preventDefault();
12         const startPos = text.selectionStart ?? 0;
13         const endPos = text.selectionEnd ?? 0;
14         const newPos = startPos + 1;
15         const val = text.value;
16         const head = val.slice(0, startPos);
17         const foot = val.slice(endPos);
18         text.value = `${head}\t${foot}`;
19         text.setSelectionRange(newPos, newPos);
20     }
21     codeHighlight();
22 });
```

main.ts では，テキストエリアとオーバレイ要素の同時スクロール，及びタブ入力の実装を行っている．

本方針における課題を以下に述べる．まず，overflow: auto; とした場合，padding-right が適用されない現象が起こった．疑似要素による力技も試したが改善されなかったため，テキストエリア及びオーバレイ要素を親コンテナよりも小さく設定し，中央配置することで応急処置を図った．また，スクロールは始まるがテキストエリアの高さも伸びてしまうという現象が起きたが，これは親要素の height を % 指定していたのが原因であり，max-height を指定することで解決できた．更に，テキストエリアのスクロールが始まった状態で改行を行うと，1 行分の高さが追加されず，表示が崩れる現象が起こった．これはおそらくテキストエリアとオーバレイのスクロールが同期されていない点が問題であり，様々な解決策を試したが改善には至らなかった．

5　ハイライタの実装

5.1　実装方針

ハイライタの実装方針は次の通りである．

まず，与えられたソースコードを字句単位に分割し，必要に応じて種別と HTML のクラス名を付加した

トークン列を生成する．最後にトークン列を順番に走査し，クラス名がある場合は字句要素を span タグで囲いながら HTML を生成する．この時，元のソースコードのホワイトスペースを維持しながら HTML を構築していく点に注意する．

前提としてハイライトを行うソースコードの文法は正しいものと仮定する．これにより，ハイライタは簡易的なコンパイラのように構成を考えることができる．(例えば，細かい文法チェックはハイライタにおいては不要である．) コンパイラが「字句解析」,「構文解析」,「意味解析」,「コード生成」の 4 フェーズで構成されているのに倣い，ハイライタは「字句解析」,「トークン解析」,「コード生成」の 3 フェーズで構成することとする．ここで,「トークン解析」は「字句解析で生成したトークン列を，ソース言語の仕様に従い属性を調整する処理」と定義する．

TS で実装するに当たり，字句解析とコード生成はソース言語によらず処理は共通であるため，これを基底クラスのメソッドとして定義する．そして，この基底クラスを継承した派生クラスをソース言語ごとに定義し，内部にトークン解析の処理をメソッドとして定義する．

5.2 インタフェースの定義

ハイライタ実装に必要なインタフェースを定義した type.ts を図 15 に示す．

図 15　インタフェースの定義 (type.ts)

```
1  export type ClassName = string | null;
2
3  export interface Token {
4      lexeme: string
5      type: string
6      className: ClassName
7      tag?: string[]
8  };
9
10 export interface PatternList {
11     pattern: RegExp
12     className: ClassName
13 };
```

Token は字句解析において生成するデータで，次の要素からなる．

- 字句要素 (lexeme)
- トークン種別 (type)
- クラス名 (className)
- タグ (tag)

なお，タグはトークン解析の際に付加情報として用いるプロパティである．

PatternList は字句解析器に渡すデータで，次の要素からなる．

- 正規表現 (pattern)
- クラス名 (className)

この PatternList を渡された字句解析器は，正規表現 pattern にマッチする字句要素を切り出し，クラス名を className としてトークン列を生成する．

5.3 基底クラス SyntaxHighlight の実装

図 16　ハイライタ基底クラス (base.ts)

JS の split メソッドは Python 等とは異なり，最大分割数を超えた文字列は破棄されてしまう．例えば以下のような挙動となる．

```
'color: getColor($var: red);'.split(':', 2);  // ['color', ' getColor($var']
// I want to get an array ['color', 'getColor($var: red);'].
```

そこで，Python と同等の split メソッドの挙動を得られるよう，String クラスに拡張メソッドを定義することを考える．

(1) interface の拡張

組み込みオブジェクトである String に，拡張メソッド splitWithRest() を定義したい．そこで，以下のようにインタフェースを拡張する．

```
interface String {
    splitWithRest(separator: string | RegExp, limit?: number): string[];
}
```

これにより String オブジェクトに splitWithRest() を定義することができる．

(2) メソッド本体の実装

まず前提として，JS(TS) はプロトタイプベースのオブジェクト指向言語である．JS では各オブジェクトは内部に prototype プロパティを保持し，そこにメソッド本体の定義等が行われている．

そのため，interface を拡張した後は以下のようにメソッド本体の定義が可能である．

```
String.prototype.splitWithRest = function (separator: string | RegExp, limit?: number) {
    if (limit === undefined) return this.split(separator);
    if (limit < 0)           return this.split(separator);
    if (limit === 0)         return [String(this)];

    let   rest = String(this);
    const ary = [];

    const parts = String(this).split(separator);

    while (limit--) {
        const part = parts.shift();
        if (part === undefined) break;
        ary.push(part);
        rest = rest.slice(part.length);
        const match = rest.match(separator);
        if (match) rest = rest.slice(match[0].length);
    }
    if (parts.length > 0) ary.push(rest);

    return ary;
}
```

このように，prototype は自由に書き換え可能なのだが，その自由度ゆえに名前空間の衝突等が起こり得る (これを**プロトタイプ汚染**という)．また，この性質はあらゆるオブジェクトを改変できることを意味するた

め，XSS 等の攻撃にも通ずる重大な脆弱性である．

そこでプロトタイプ汚染を抑制するため，Object.defineProperties() を利用する．適切な属性を付加しつつ prototype を変更することで，安全性を高めることが目的である．

```
Object.defineProperties(String.prototype, {
    splitWithRest: {
        configurable: true,
        enumerable: false,
        writable: true,
        value: function (separator: string | RegExp, limit?: number) {
            if (limit === undefined) return this.split(separator);
            if (limit < 0)           return this.split(separator);
            if (limit === 0)         return [String(this)];

            let   rest = String(this);
            const ary = [];

            const parts = String(this).split(separator);

            while (limit--) {
                const part = parts.shift();
                if (part === undefined) break;
                ary.push(part);
                rest = rest.slice(part.length);
                const match = rest.match(separator);
                if (match) rest = rest.slice(match[0].length);
            }
            if (parts.length > 0) ary.push(rest);

            return ary;
        },
    }
});
```

※ここで，メソッド定義の際にアロー関数ではなく function を使っているのは，this の値の変更を防ぐためである．

グローバルな宣言ではないため，このメソッドの利用には定義ファイルをインポートする必要があることを付記しておく．

参考文献

[1] Node.js とはなにか？，https://qiita.com/non_cal/items/a8fee0b7ad96e67713eb，2024/12/5

[2] nodejs とは，https://kinsta.com/jp/knowledgebase/what-is-node-js/，2024/12/5.

[3] npm とは，https://kinsta.com/jp/knowledgebase/what-is-npm/，2024/12/5.

[4] package.json の中身を理解する，
https://qiita.com/dondoko-susumu/items/cf252bd6494412ed7847，2024/12/5.

[5] 最新版で学ぶ webpack 5 入門，https://ics.media/entry/12140/，2024/12/5.

[6] npm install の –save-dev って何？，
https://qiita.com/kohecchi/items/092fcbc490a249a2d05c，2024/12/6.

[7] TypeScript チュートリアル -環境構築編-，
https://qiita.com/ochiochi/items/efdaa0ae7d8c972c8103，2024/12/6.

[8] webpack の苦手意識を無くす，https://zenn.dev/msy/articles/c1f00c55e88358，2024/12/8.

[9] CSS で宇宙空間を表現する。, https://qiita.com/junya/items/a2f8984841dc0d559a68, 2024/12/9.
[10]【Font Awesome】バージョン 6 で変わった CSS 擬似要素の設定でアイコン表示する時の備忘録いろいろ,
https://www.appleach.co.jp/note/webdesigner/6960/, 2024/12/13.
[11] Webpack5 で FontAwesome-free を利用する,
https://zenn.dev/manappe/articles/da22d23f73de3b, 2024/12/13.
[12] Documentation, https://sass-lang.com/documentation/, 2024/12/18.
[13] プロトタイプ汚染とは,
https://www.sompocybersecurity.com/column/glossary/prototype-pollution, 2024/12/19.
[14]「Typescript」で prototype の沼にハマったので、色々調べてみた。,
https://note.alhinc.jp/n/n2ee7f772e020, 2024/12/19.
[15] React に TypeScript で拡張メソッドを作る,
https://qiita.com/s-ueno/items/90030bab006c79173dc5, 2024/12/19.
[16] ,
, 2024/12/19.
[17] ,
, 2024/12/18.
[18] ,
, 2024/12/18.