Web SyntaxHighlighter with TypeScript
v1.0

2025 年 2 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タを作成する．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　シンタックスハイライタの実装

5.1　シンタックスハイライタとは

シンタックスハイライタとは，入力されたソースコードに適切な色付けを行うプログラムである．本プロジェクトでは，VS Code と同様のハイライトを施せるようシンタックスハイライタを実装する．

今回作成するのは Web 用のシンタックスハイライタであるため，何らかの言語で書かれたソースコードを入力とし，ハイライトができるよう適切な字句単位でタグで囲まれた HTML を出力とすることになる．例えば図 15 の C ソースを入力とした場合，図 16 の HTML を出力することを目的とする．

図 15　シンタックスハイライタへの入力 (C 言語)

```
#include <stdio.h>

int main(void) {
    int a = 0;
    printf("%d\n", a);
    return 0;
}
```

図 16　シンタックスハイライタの出力

```
<span class='hl-pp'>#include</span> <span class='hl-hf'>&lt;stdio.h&gt;</span>

<span class='hl-vt'>int</span> <span class='hl-f'>main</span><span class='hl-b-1'>(</span><span class='hl-vt'>void</span>
>\\<span class='hl-b-1'>)</span> <span class='hl-b-1'>{</span>
    <span class='hl-vt'>int</span> <span class='hl-v'>a</span> = <span class='hl-n'>0</span>;
    <span class='hl-f'>printf</span><span class='hl-b-2'>(</span><span class='hl-str'>&quot;</span><span class='hl-cs'>%d</span><span class='hl-es'>\n</span><span class='hl-str'>&quot;</span>, <span class='hl-v'>a</span><span class='hl-b-2'>)</span>;
    <span class='hl-k1'>return</span> <span class='hl-n'>0</span>;
<span class='hl-b-1'>}</span>
```

このように，字句要素を適切に色付けできるようタグで囲むことがシンタックスハイライタの大きな役割である．後は，CSS でクラスごとに色を指定すればハイライトが施されることになる．

このように，何らかのソースコードを HTML に変換するという側面を見れば，シンタックスハイライタは簡易的なコンパイラとも言える．

なお，本レポート執筆時点ではコマンドプロンプト，HTML，CSS/SCSS，及び C のハイライタの完成を目的としている．

5.2　実装方針

シンタックスハイライタの実装方針は次の通りである．

まず前提として，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．細かい文法チェックを省いた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．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，シンタックスハイライタは「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．

まず「字句解析」で，与えられたソースコードを字句単位に分割し，必要に応じてトークン種別と HTML のクラス名を付加したトークン列を生成する．次に「トークン解析」で，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．最後に「コード生成」で，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．

5.3 ユーザ定義型，インタフェースの定義

シンタックスハイライタ実装に必要な型エイリアス，及びインタフェースを定義した type.ts を図 17 に示す．

図 17　インタフェースの定義 (type.ts)

```
1 export type ClassName = string | null;
2
3 interface Pattern {
4     regexp: RegExp
5     className: string | null
6 };
7
8 export type PatternList = Record<string, Pattern>;
```

PatternList は字句解析器に渡すオブジェクトで，正規表現 (regexp) とクラス名 (className) からなる．この PatternList を渡された字句解析器は，正規表現 regexp 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．

5.3.1 Record 型とは

Record とは，key と value の型を指定してオブジェクト (dict，連想配列) の型を決定する utility type である．そのため，Record を用いずとも同様の型構造を定義できるが，例えば以下のようになり，複雑な型定義となってしまう．

```
1: interface Pattern {
2:     pattern:   RegExp
3:     className: ClassName
4: };
5:
6: export type PatternList = { [key: string]: Pattern };
```

5.4 クラス設計

図 1 に本プロジェクトのクラス図を示す．

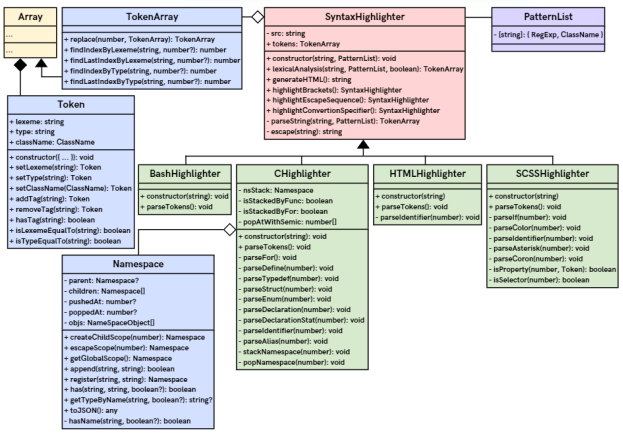

図 1　本プロジェクトのクラス図

5.4.1 SyntaxHighlighter クラス

ソース言語によらず共通的な処理を抽象基底クラス SyntaxHighlighter として定義する．各ソース言語のハイライタはこの基底クラスを継承し，トークン解析を行うメソッド parseTokens() を個別に定義する．詳しくは第 5.5 節以降で解説する．

5.4.2 Token クラス，及び TokenArray クラス

Token クラス，及び TokenArray クラスの実装を一部抜粋し，図 18 に示す．

図 18　Token, TokenArray

```
1  import { ClassName } from './type';
2
3  export class Token {
4      lexeme: string;
5      type: string;
6      className: ClassName;
7      tags: string[];
8
9      constructor({
10         lexeme = '',
11         type = '',
12         className = null,
13         tags = []
14     }: {
15         lexeme?: string
16         type?: string
17         className?: ClassName
18         tags?: string[]
19     } = {}) {
20         this.lexeme = lexeme;
21         this.type = type;
22         this.className = className;
23         this.tags = tags;
24     }
25
26     setLexeme(lexeme: string) {
27         this.lexeme = lexeme;
28         return this;
29     }
30
31     // ...
32
33     isLexemeEqualTo(lex: string | string[]) {
34         if (Array.isArray(lex)) {
35             return lex.includes(this.lexeme);
36         }
37         return lex === this.lexeme;
38     }
39
40     // ...
41 }
42
43 export class TokenArray extends Array<Token> {
44     replace(pos: number, tokens: TokenArray) {
45         this.splice(pos, 1, ...tokens);
46         return this;
```

```
47      }
48
49      findIndexByLexeme(lex: string, startPos = 0) {
50          if (startPos >= this.length) return -1;
51
52          for (let i = Math.max(0, startPos); i < this.length; i++) {
53              if (this[i].lexeme === lex) return i;
54          }
55          return -1;
56      }
57
58      findLastIndexByLexeme(lex: string, startPos = this.length - 1) {
59          for (let i = Math.min(startPos, this.length - 1); i >= 0; i--) {
60              if (this[i].lexeme === lex) return i;
61          }
62          return -1;
63      }
64
65      // ...
66  }
```

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．

Token クラス実装のポイントは，メソッドチェーンを可能とするために this(インスタンス自身) を返している点である (28 行目)．これにより，例えば tokens.setType('var').setClassName('hl-var'); といった記述が可能となる．

TokenArray クラスは Token オブジェクトの配列を管理するためのものだが，これはわざわざ実装しなくとも Token[] 型で表現できる．しかしわざわざクラスを定義している理由は，トークン検索のメソッド (findIndexByLexeme() 等) を定義するためである．Array オブジェクトの indexOf() では Token そのものを渡さなければならないため，lexeme のみでの検索をかけることができない．そこで，Array のサブクラスとしてこれを実装することで，従来の配列としてのアクセスやメソッドを保ったまま任意のメソッドを定義している．

5.4.3 NameSpace クラス

NameSpace クラスの実装を一部抜粋し，図 19 に示す．

図 19 NameSpace

```
1   interface NameSpaceObject {
2       name: string
3       type: string
4   };
5
6   export default class Namespace {
7       private parent: Namespace | null;
8       private children: Namespace[];
9       private name: string;
```

```
10      private objs: NameSpaceObject[];
11
12      constructor(name: string, parent: Namespace | null = null) {
13          this.parent = parent;
14          this.children = [];
15          this.name = name;
16          this.objs = [];
17      }
18
19      createChildScope(scopeName: string) {
20          const child = new Namespace(scopeName, this);
21          this.children.push(child);
22          return child;
23      }
24
25      escapeScope() {
26          if (!this.parent) return this;
27          return this.parent;
28      }
29
30      getGlobalScope() {
31          let ns: Namespace = this;
32          for (; ns.parent; ns = this.escapeScope());
33          return ns;
34      }
35
36      append(objName: string, type: string) {
37          if (this.hasName(objName, true)) return;
38          this.objs.push({ name: objName, type });
39      }
40
41      register(str: string, type: string) {
42          const objNames = str.split(' ');
43          objNames.forEach(objName => this.append(objName, type));
44
45          return this;
46      }
47
48      has(objName: string, type: string, currScopeOnly = false): boolean {
49          const foundInCurrScope = this.objs.some((obj) =>
50              obj.name === objName && obj.type === type
51          );
52
53          if (foundInCurrScope) return true;
54
55          return !currScopeOnly && this.parent?.has(objName, type) || false;
56      }
57
58      getTypeByName(name: string, currScopeOnly = false): string | null {
59          const foundInCurrScope = this.objs.find((obj) => obj.name === name);
60
61          if (foundInCurrScope) return foundInCurrScope.type;
62
63          return !currScopeOnly && this.parent ? this.parent.getTypeByName(name) : null;
64      }
65
66      // For test
67      toJSON(): any {
68          return {
```

```
69              name: this.name,
70              objs: this.objs,
71              children: this.children.map((child) => child.toJSON()),
72          };
73      }
74
75      private hasName(objName: string, currScopeOnly = false): boolean {
76          const foundInCurrScope = this.objs.some((obj) =>
77              obj.name === objName
78          );
79
80          if (foundInCurrScope) return true;
81
82          return !currScopeOnly && this.parent?.hasName(objName) || false;
83      }
84  }
```

NameSpace は名前空間を管理するためのクラスで，スタックに似たデータ構造を取っている．なお，名前空間に登録するオブジェクトはトークン解析において必要なもののみを登録すれば十分である．(出現した変数，型エイリアス等のオブジェクトを全て登録しても良いが，厳密な構文解析は必要ないためいささか冗長である．) push に当たる操作は createChildScope() であり，スコープ形成の際に呼び出す．pop に当たる操作は escapeScope() であり，スコープの終わりを検知した際に呼び出す．トークン解析において既に無効なスコープの情報も保持しておきたいため，pop のような破壊的操作ではなく，親スコープに戻る形で実装している．

詳しい使い方については，各言語のトークン解析の解説にて触れる．

5.5 基底クラス SyntaxHighlighter の実装

図 20 に，ハイライタ基底クラス SyntaxHighlighter を示す．

図 20　ハイライタ基底クラス (base.ts)

```
1  import { PatternList } from './type';
2  import { Token, TokenArray } from './token';
3  import { mod } from '../helper';
4
5
6  export default abstract class Syntaxhighlighter {
7      src: string;
8      tokens: TokenArray;
9
10     constructor(src: string, patternList: PatternList ) {
11         this.src = src;
12         this.tokens = this.lexicalAnalysis(src, patternList);
13     }
14
15     abstract parseTokens(): void;
16
17     lexicalAnalysis(src: string, patternList: PatternList, includeWhiteSpace = false) {
18         const tokens = new TokenArray();
19         patternList = {
20             ...patternList,
21             other: { regexp: /^\S/, className: null },
22             whiteSpace: { regexp: /^\s+/, className: null }
23         };
```

```
24
25          while (src) {
26              for (const type in patternList) {
27                  const match = src.match(patternList[type].regexp);
28                  if (match) {
29                      const lexeme = match[0];
30                      src = src.slice(lexeme.length);
31                      if (type === 'whiteSpace' && !includeWhiteSpace) continue;
32                      tokens.push(new Token({
33                          lexeme: lexeme,
34                          type: type,
35                          className: patternList[type].className
36                      }));
37                      break;
38                  }
39              }
40          }
41          return tokens;
42      }
43
44      generateHTML() {
45          this.parseTokens();
46
47          let highlightedSrc = '';
48          let i = 0;
49          this.tokens.forEach(({ lexeme, className }) => {
50              const lexFrom = this.src.indexOf(lexeme, i);
51
52              if (i < lexFrom) highlightedSrc += this.src.slice(i, lexFrom);
53              highlightedSrc += className
54                  ? `<span class='${className}'>${this.escape(lexeme)}</span>`
55                  : this.escape(lexeme);
56
57              i = lexFrom + lexeme.length;
58          });
59          if (i < this.src.length) {
60              highlightedSrc += this.src.slice(i);
61          }
62
63          return highlightedSrc;
64      }
65
66      highlightBrackets() {
67          const includesOpenBracket = (lex: string) => {
68              return lex.includes('(') || lex.includes('{') || lex.includes('[')
69                  ? true
70                  : false
71          }
72          const includesCloseBracket = (lex: string) => {
73              return lex.includes(')') || lex.includes('}') || lex.includes(']')
74                  ? true
75                  : false
76          }
77
78          let bracketNest = 0;
79          this.tokens.forEach((token, i) => {
80              if (token.hasTag('ignoreHighlightBrackets')) return;
81              if (includesOpenBracket(token.lexeme)) {
82                  bracketNest = mod(bracketNest, 3) + 1;
```

```
83              this.tokens[i].setClassName(`hl-b-${bracketNest}`);
84          }
85          if (includesCloseBracket(token.lexeme)) {
86              this.tokens[i].setClassName(`hl-b-${bracketNest}`);
87              bracketNest = bracketNest - 1 || 3;
88          }
89      });
90
91      return this;
92  }
93
94  highlightEscapeSequence() {
95      const patternList: PatternList = {
96          escapeSequence: { regexp: /^\\(?:[0-9a-fA-F]{4}|\S)/, className: 'hl-es' },
97      }
98
99      for (let i = 0; i < this.tokens.length; i++) {
100         const token = this.tokens[i];
101         if (token.isTypeEqualTo('string')) this.tokens.replace(i, this.parseString(token
                .lexeme, patternList));
102     }
103
104     return this;
105 }
106
107 highlightConvertionSpecifier() {
108     const patternList: PatternList = {
109         conversionSpecifier: { regexp: /^(?:%[-+ #0]*(?:0|[1-9][0-9]*)
                ?(?:\.(?:0|[1-9][0-9]*)?)?(?:h{1,2}|l{1,2}|[Ljzt])?[diuoxXfFgGeEpcs%])/g,
                className: 'hl-cs' },
110     }
111
112     for (let i = 0; i < this.tokens.length; i++) {
113         const token = this.tokens[i];
114         if (token.isTypeEqualTo('string')) this.tokens.replace(i, this.parseString(token
                .lexeme, patternList));
115     }
116
117     return this;
118 }
119
120 private parseString(lex: string, patternList: PatternList) {
121     const margeTokens = (tokens: Token[]) => {
122         const margedTokens = new TokenArray();
123         let lexeme = tokens[0].lexeme;
124         let type = tokens[0].type;
125         let className = tokens[0].className;
126
127         for (let i = 1; i < tokens.length; i++) {
128             const token = tokens[i];
129             if (type === token.type) {
130                 lexeme += token.lexeme;
131             } else {
132                 margedTokens.push(new Token({ lexeme, type, className }));
133                 lexeme = token.lexeme;
134                 type = token.type;
135                 className = token.className
136             }
137         }
```

```
138             if (type === tokens[tokens.length - 1].type) margedTokens.push(new Token({
                    lexeme, type, className }));
139
140             return margedTokens;
141         }
142
143         const tokens = this.lexicalAnalysis(lex, patternList, true).map(token => {
144             if (token.type === 'other') token.setType('string').setClassName('hl-str');
145             else if (token.type !== 'whiteSpace') token.setType(token.type).setClassName(
                    token.className);
146
147             return token;
148         });
149
150         return margeTokens(tokens);
151     }
152
153     private escape(str: string) {
154         return str.replace(/</g, '&lt;')
155                   .replace(/>/g, '&gt;')
156                   .replace(/\$/g, '&#036;')
157                   .replace(/"/g, '&quot;')
158                   .replace(/'/g, '&#39;')
159                   .replace(/`/g, '&#096;');
160     }
161 }
```

本クラスでは，いずれのソース言語でも共通の字句解析 lexicalAnalysis() とコード生成 generateHTML() を定義している．また，必要に応じてサブクラスから呼び出せるよう，括弧類やエスケープシーケンス，変換指定子のハイライトメソッドも定義している．

lexicalAnalysis() はトークン列を生成するが，後のトークン解析に備えてホワイトスペースはトークンに含めるべきではない．しかし, highlightEscapeSequence() と highlightConvertionSpecifier() にて利用する文字列解析メソッド parseString() は，トークン分割後も文字列内のホワイトスペースをそのまま保持しなくてはならない．そこで，オプションでトークン列にホワイトスペースを含めるか否かを選択できるような設計とした．

先述した通り，トークン解析後のトークン列はホワイトスペースを含まない．しかし，generateHTML() は元のホワイトスペースを保ったまま HTML を生成する必要があるため，元のソースコード src と字句長を同期させている点がポイントである．

トークン解析 parseTokens については，ソース言語ごとに異なるため抽象メソッドとして定義している．これにより，サブクラスに対して parseTokens() のオーバライドを強制することができる．

5.6 プロンプトハイライタの実装 BashHighlighter

図 21 に，プロンプトハイライト用クラス BashHighlighter を示す．

図 21　プロンプトハイライタ (bash.ts)

```
1 import Syntaxhighlighter from '../base';
2
3
4 export default class BashHighlighter extends Syntaxhighlighter {
5     constructor(src: string) {
6         super(src, {});
```

```
7     }
8
9     parseTokens() {}
10 }
```

プロンプトは，軸切り出しのハイライトの必要は無い．そのため，パターンリストの登録やトークン解析の必要は無いため，最小構成のシンタックスハイライタとも言える．

今後の展望としては，任意の文字列をハイライトできるようなマークアップ記法を解釈できるように拡張することが考えられる．

5.7 HTML ハイライタの実装 HTMLHighlighter

図 22 に，HTML ハイライト用クラス HTMLHighlighter を示す．

図 22　HTML ハイライタ (html.ts)

```
1  import SyntaxHighlighter from '../base';
2  import { Token } from '../token';
3  import { PatternList } from '../type';
4
5  // TODO: To highlight JS within script tags and CSS within style tags.
6
7  const patternList: PatternList = {
8      comment: { regexp: /^(?:<!--[\s\S]*?-->|<!--[\s\S]*)/, className: 'hl-cm' },
9      tagOpen: { regexp: /^<[!\/]?/, className: 'hl-tag' },
10     tagClose: { regexp: /^\/?>/, className: 'hl-tag' },
11     identifier: { regexp: /^[a-zA-Z_][\w:.-]*/, className: null },
12     equal: { regexp: /^=/, className: null },
13     value: { regexp: /^(["']).*?\1/, className: 'hl-str' },
14 };
15
16 export default class HTMLHighlighter extends SyntaxHighlighter {
17     constructor(src: string) {
18         super(src, patternList);
19     }
20
21     parseTokens() {
22         for (let i = 0; i < this.tokens.length; i++) {
23             const token = this.tokens[i];
24             if (token.type === 'identifier') this.parseIdentifier(i);
25         }
26     }
27
28     private parseIdentifier(pos: number) {
29         const prevToken = (pos > 0) ? this.tokens[pos - 1] : new Token();
30
31         if (prevToken.isTypeEqualTo('tagOpen')) this.tokens[pos].setType('tagName').
               setClassName('hl-tn');
32         if (prevToken.isTypeEqualTo(['tagName', 'value'])) this.tokens[pos].setType('attr').
               setClassName('hl-attr');
33     }
34 }
```

HTML のハイライトは非常に単純で，トークンの大半が字句解析終了時点においてクラス名が確定する．

しかし，identifier トークンは「タグ名」か「属性名」か否かが字句解析では判別できない．そこで，この分類をトークン解析にてトークンの並び順を元に行っている．

今後の展望としては，<script>タグで囲まれた範囲に JS 用のハイライトを施すといった拡張が考えられる．

5.8 CSS/SCSS ハイライタの実装 SCSSHighlighter

CSS/SCSS のトークン解析を行うに当たっては，identifier の属性決定が重大な課題であった．これは，プロパティやセレクタ，値，属性，属性値と候補が多い上に，単なるトークン走査では確定し切れないからである．(例えば':' は，セレクタにもプロパティと値の区切り文字にも用いられる．)

CSS/SCSS のトークン解析の実装は複雑で，改善を繰り返していくうちに，2 つの方針が定まった．以下，各方針について説明していく．

5.8.1 方針 1: 段階的に字句を分割

一つ目の実装方針は，字句要素を段階的に分割する方法である．この方針で実装した CSS ハイライタを図 23 に示す．

図 23 CSS/SCSS ハイライタ (scss.ts)(1)

```
1  import SyntaxHighlighter from '../base';
2  import { Token, TokenArray } from '../token';
3  import { PatternList } from '../type';
4
5
6  const patternList: PatternList = {
7      comment: { regexp: /^(?:\/\/.*|\/\*[\s\S]*?\*\/|\/\*[\s\S]*)/, className: 'hl-cm' },
8      propVal: { regexp: /^\S+\s*:\s*[^{](?:[^{]|\s)*?;/, className: null },
9      keyword1: { regexp: /^(?:as)(?!\w)/, className: 'hl-k1' },
10     keyword2: { regexp: /^(?:\*)(?!\w)/, className: 'hl-k2' },
11     string: { regexp: /^(["']).*?\1/, className: 'hl-str' },
12     atRules: { regexp: /^@(?:use|else)(?!\w)/, className: 'hl-at' },
13     include: { regexp: /^@include\s+\S[\s\S]*?;/, className: null },
14     extend: { regexp: /^@extend\s+\S+\s*?;/, className: null },
15     if: { regexp: /^@(?:else\s+)?if\s+\S[\s\S]*?{/, className: null },
16     userDef: { regexp: /^@(?:keyframes|mixin|function)\s+\S[\s\S]*?{/, className: null },
17     selectors: { regexp: /^(?:[&.#\[>+~\w][\s\S]*?|:root\s*){/, className: null },
18 };
19
20 export default class SCSSHighlight extends SyntaxHighlighter {
21     constructor(src: string) {
22         super(src, patternList);
23     }
24
25     parseTokens() {
26         const newTokens = new TokenArray();
27
28         // TODO: for, each, etc...
29         this.tokens.forEach(token => {
30             switch (token.type) {
31                 case 'comment': newTokens.push(new Token({...token, tags: ['
                       ignoreHighlightBrackets']})); break;
32                 case 'propVal': newTokens.push(...this.parsePropVal(token.lexeme)); break;
33                 case 'include': newTokens.push(...this.parseInclude(token.lexeme)); break;
34                 case 'extend': newTokens.push(...this.parseExtend(token.lexeme)); break;
35                 case 'if': newTokens.push(...this.parseIf(token.lexeme)); break;
36                 case 'userDef': newTokens.push(...this.parseUserDef(token.lexeme)); break;
37                 case 'selectors': newTokens.push(...this.parseSelectors(token.lexeme)); break
```

```
                ;
38              default: newTokens.push(token);
39          }
40      });
41
42      this.tokens = newTokens;
43      this.highlightBrackets()
44          .highlightEscapeSequence();
45  }
46
47  private parsePropVal(lex: string) {
48      const tokens = new TokenArray();
49
50      const lexemes = lex.splitWithRest(':', 1); // split with first ':'. (e.g.) 'a:b:c'
            => ['a', 'b:c']
51      const prop = lexemes[0].trim();
52      const values = lexemes[1].slice(0, -1).trim();
53
54      tokens.push(new Token({ lexeme: prop, type: 'property', className: 'hl-prop' }));
55      tokens.push(new Token({ lexeme: ':', type: 'other', className: null }));
56      tokens.push(...this.parseValues(values));
57      tokens.push(new Token({ lexeme: ';', type: 'other', className: null }));
58
59      return tokens;
60  }
61
62  private parseValues(lex: string) {
63      const patternList: PatternList = {
64          number: { regexp: /^(?:-?\.?[0-9]+(?:px|rem|em|%|vw|vh|vmin|vmax|s)?)/,
                className: 'hl-n' },
65          color: { regexp: /^#[0-9a-fA-F]*/, className: 'hl-val' },
66          string: { regexp: /^(["']).*?\1/, className: 'hl-str' },
67          important: { regexp: /^!important/, className: 'hl-k2' },
68          operator: { regexp: /^(?:and|or|not)/, className: null }, // For expression
69          boolean: { regexp: /^true|false/, className: null }, // For expression
70          funcCall: { regexp: /^[A-Za-z_$][\w$-]*\s*\(\s*(?:[^()]*|\([^()]*\))*\s*\)/,
                className: null },
71          identifier: { regexp: /^(?:--)?[A-Za-z_$][\w$-]*/, className: null },
72      }
73
74      const tokens = this.lexicalAnalysis(lex, patternList);
75      const newTokens = new TokenArray();
76
77      tokens.forEach(token => {
78          switch (token.type) {
79              case 'funcCall': newTokens.push(...this.parseFuncHead(token.lexeme)); break;
80              case 'identifier': newTokens.push(this.parseIdentifier(token.lexeme)); break;
81              default: newTokens.push(token);
82          }
83      });
84
85      return newTokens;
86  }
87
88  private parseIdentifier(lex: string){
89      return lex.startsWith('$') || lex.startsWith('--')
90          ? new Token({ lexeme: lex, type: 'variable', className: 'hl-v' })
91          : new Token({ lexeme: lex, type: 'value', className: 'hl-val' })
92  }
```

```
93
94      private parseFuncHead(lex: string) {
95          const patternList: PatternList = {
96              identifier: { regexp: /^[A-Za-z_$-][\w$-]*/, className: null },
97              argList: { regexp: /^\([\s\S]*\)/, className: null },
98          }
99          const tokens = this.lexicalAnalysis(lex, patternList);
100         const newTokens = new TokenArray();
101
102         const funcNameToken = tokens.filter(({ type }) => type === 'identifier');
103         const argListToken = tokens.filter(({ type }) => type === 'argList');
104
105         if (!argListToken.length) return [new Token({ lexeme: lex, type: 'function',
                classname: 'hl-f' })];
106
107         const funcName = funcNameToken[0].lexeme;
108         const args = argListToken[0].lexeme.slice(1, -1).trim();
109
110         newTokens.push(new Token({ lexeme: funcName, type: 'function', className: 'hl-f' }))
                ;
111         newTokens.push(new Token({ lexeme: '(', type: 'other', className: null }));
112         newTokens.push(...this.parseArgs(args));
113         newTokens.push(new Token({ lexeme: ')', type: 'other', className: null }));
114
115         return newTokens;
116     }
117
118     private parseArgs(lex: string) {
119         const tokens = new TokenArray();
120
121         const args = lex.split(',');
122
123         args.forEach(arg => {
124             arg = arg.trim();
125             if (arg.includes(':')) tokens.push(...this.parseOptArg(arg));
126             else tokens.push(...this.parseValues(arg)); // normal argument => treat as '
                    values' and parse it.
127             tokens.push(new Token({ lexeme: ',', type: 'other', className: null }));
128         });
129         tokens.pop();
130
131         return tokens;
132     }
133
134     private parseOptArg(lex: string) {
135         const tokens = new TokenArray();
136
137         const lexemes = lex.split(':');
138         const variable = lexemes[0].trim();
139         const value = lexemes[1];
140
141         tokens.push(new Token({ lexeme: variable, type: 'variable', className: 'hl-prop' }))
                ;
142         tokens.push(new Token({ lexeme: ':', type: 'other', className: null }));
143         tokens.push(...this.parseValues(value));
144
145         return tokens;
146     }
147
```

```
148     private parseInclude(lex: string) {
149         const tokens = new TokenArray();
150 
151         const mixin = lex.split('@include')[1].trim().slice(0, -1).trim();
152 
153         tokens.push(new Token({ lexeme: '@include', type: 'atRules', className: 'hl-at' }));
154         tokens.push(...this.parseFuncHead(mixin));
155         tokens.push(new Token({ lexeme: ';', type: 'other', className: null }));
156 
157         return tokens;
158     }
159 
160     private parseExtend(lex: string) {
161         const tokens = new TokenArray();
162 
163         const lexemes = lex.split(/\s+/);
164         const keyword = lexemes[0];
165         const selector = lexemes[1].split(';')[0].trim();
166 
167         tokens.push(new Token({ lexeme: keyword, type: 'atRules', className: 'hl-at' }));
168         tokens.push(new Token({ lexeme: selector, type: 'selector', className: 'hl-sel' }));
169         tokens.push(new Token({ lexeme: ';', type: 'other', className: null }));
170 
171         return tokens;
172     }
173 
174     private parseIf(lex: string) {
175         const tokens = new TokenArray();
176 
177         const atRules = lex.startsWith('@if') ? ['@if'] : ['@else', 'if'];
178         const expr = lex.split(/if/)[1].slice(0, -1).trim();
179 
180         atRules.forEach(lex => {
181             tokens.push(new Token({ lexeme: lex, type: 'atRules', className: 'hl-at' }));
182         });
183         tokens.push(...this.parseValues(expr));
184         tokens.push(new Token({ lexeme: '{', type: 'other', className: null }));
185 
186         return tokens;
187     }
188 
189     private parseUserDef(lex: string) {
190         const tokens = new TokenArray();
191 
192         const lexemes = lex.splitWithRest(/\s+/, 1);
193         const atRule = lexemes[0];
194         const funcHead = lexemes[1].slice(0, -1).trim();
195 
196         tokens.push(new Token({ lexeme: atRule, type: 'atRules', className: 'hl-at' }));
197         tokens.push(...this.parseFuncHead(funcHead));
198         tokens.push(new Token({ lexeme: '{', type: 'other', className: null }));
199 
200         return tokens;
201     }
202 
203     private parseSelectors(lex: string) {
204         const patternList: PatternList = {
205             symbol: { regexp: /^(?:&|\*)/, className: 'hl-k2' },
206             selector: { regexp: /^[.#]?[A-Za-z_:][\w:-]*/, className: 'hl-sel' },
```

```
207             attr: { regexp: /^\[.*?\]/, className: null },
208         };
209         const tokens = this.lexicalAnalysis(lex, patternList);
210         const newTokens = new TokenArray();
211 
212         tokens.forEach(token => {
213             if (token.type === 'attr') newTokens.push(...this.parseAttr(token.lexeme));
214             else newTokens.push(token);
215         });
216 
217         return newTokens;
218     }
219 
220     private parseAttr(lex: string) {
221         const tokens = new TokenArray();
222 
223         const lexemes = lex.slice(1, -1).split('=');
224         const attr = lexemes[0].trim();
225         const value = lexemes.length > 1 ? lexemes[1].trim() : null;
226 
227         tokens.push(new Token({ lexeme: '[', type: 'other', className: null }));
228         tokens.push(new Token({ lexeme: attr, type: 'attr', className: 'hl-attr' }));
229         if (value) {
230             tokens.push(new Token({ lexeme: '=', type: 'other', className: null }));
231             tokens.push(new Token({ lexeme: value, type: 'value', className: 'hl-val' }));
232         }
233         tokens.push(new Token({ lexeme: ']', type: 'other', className: null }));
234 
235         return tokens;
236     }
237 }
```

本方針の利点は，トークン解析を簡単化できる点である．一つのまとまった構文を字句として切り出し，そこから徐々に最小単位になるまで字句解析を繰り返すことで，トークン列を確定していく．

ただし，考えられる欠点としては次の三つがある．一つ目は，字句解析を何度も行うため計算量やメモリ効率においてパフォーマンスが悪い点である．二つ目は，コード全体の見通しが悪く，保守性に優れない点．三つめは，構文ごとに解析関数を作成しなければならず，言語仕様の変更に弱い点である．実際，本方針では未だに `for` 文や `each` 文に対応していない．

5.8.2 組み込みオブジェクトの拡張

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．

これは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．

5.8.3 方針 2:トークンの並びを元に解析

二つ目の実装方針は，HTML ハイライタと同様にトークンの並び順を元に解析する方法である．この方針で実装した CSS ハイライタを図 24 に示す．

図 24　CSS/SCSS ハイライタ (scss.ts)(2)

```
1  import SyntaxHighlighter from '../base';
2  import { Token } from '../token';
3  import { PatternList } from '../type';
4
5
6  const patternList: PatternList = {
7      comment: { regexp: /^(?:\/\/.*|\/\*[\s\S]*?\*\/|\/\*[\s\S]*)/, className: 'hl-cm' },
8      keyword1: { regexp: /^(?:as)(?!\w)/, className: 'hl-k1' },
9      keyword2: { regexp: /^(?:&|!important|true|false)/, className: 'hl-k2' },
10     if: { regexp: /^@(?:else[ ]+)?if/, className: 'hl-at' },
11     atRules: { regexp: /^@(?:use|keyframes|function|mixin|include|extend|else|for|each)/,
           className: 'hl-at' },
12     logicalOp: { regexp: /^(?:and|or|not)/, className: null },
13     color: { regexp: /^#(?:[0-9a-fA-F]{6}|[0-9a-fA-F]{3})/, className: 'hl-val' },
14     variable: { regexp: /^(?:--|\$)[A-Za-z_$][\w$-]*/, className: 'hl-v' },
15     identifier: { regexp: /^[.#]?-?[A-Za-z_][\w-]*/, className: null },
16     number: { regexp: /^(?:-?\.?[0-9]+(?:px|rem|em|%|vw|vh|vmin|vmax|s)?)/, className: 'hl-n
           ' },
17     string: { regexp: /^(["']).*?\1/, className: 'hl-str' },
18     equivalent: { regexp: /^==/, className: null },
19     asterisk: { regexp: /^\*/, className: null },
20     coron: { regexp: /^:/, className: null },
21     semic: { regexp: /^;/, className: null },
22     connector: { regexp: /^[,>+~]/, className: null },
23 };
24
25 export default class SCSSHighlighter extends SyntaxHighlighter {
26     constructor(src: string) {
27         super(src, patternList);
28     }
29
30     parseTokens() {
31         for (let i = 0; i < this.tokens.length; i++) {
```

```
32          const token = this.tokens[i];
33          switch (token.type) {
34              case 'comment': this.tokens[i].addTag('ignoreHighlightBrackets'); break;
35              case 'if': this.parseIf(i); break;
36              case 'color': this.parseColor(i); break;
37              case 'identifier': this.parseIdentifier(i); break;
38              case 'coron': this.parseCoron(i); break;
39          }
40      }
41      for (let i = 0; i < this.tokens.length; i++) {
42          const token = this.tokens[i];
43          switch (token.type) {
44              case 'identifier': this.tokens[i].setType('value').setClassName('hl-val');
                    break;
45              case 'asterisk': this.parseAsterisk(i); break;
46              case 'coron': this.parseCoron(i); break;
47          }
48      }
49
50      this.highlightBrackets()
51          .highlightEscapeSequence();
52  }
53
54  private parseIf(pos: number) {
55      const from = pos + 1;
56      const to = this.tokens.findIndexByLexeme('{', pos) - 1;
57      for (let i = from; i <= to; i++) {
58          const token = this.tokens[i];
59          if (token.isTypeEqualTo('identifier')) this.tokens[i].setType('value').
                setClassName('hl-val');
60          else if (token.isTypeEqualTo('color')) this.tokens[i].addTag('ignoreParseColor')
                ;
61      }
62  }
63
64  private parseColor(pos: number) {
65      if (!this.tokens[pos].hasTag('ignoreParseColor') && this.isSelector(pos)) this.
            tokens[pos].setType('selector').setClassName('hl-sel');
66  }
67
68  private parseIdentifier(pos: number) {
69      const prevToken = (pos > 0) ? this.tokens[pos - 1] : new Token();
70      const nextToken = (pos + 1 < this.tokens.length) ? this.tokens[pos + 1] : new Token
            ();
71
72      if (prevToken.isLexemeEqualTo('@extend')) {
73          this.tokens[pos].setType('selector').setClassName('hl-sel');
74      } else if (prevToken.isLexemeEqualTo(['@keyframes', '@function', '@mixin', '@include
            '])) {
75          this.tokens[pos].setType('function').setClassName('hl-f');
76      } else if (nextToken.isLexemeEqualTo('(') && !this.isSelector(pos)) {
77          this.tokens[pos].setType('function').setClassName('hl-f');
78      } else if (nextToken.isLexemeEqualTo('=')) {
79          this.tokens[pos].setType('attr').setClassName('hl-attr');
80      } else if (prevToken.isLexemeEqualTo('=')) {
81          this.tokens[pos].setType('value').setClassName('hl-val');
82      } else if (this.isProperty(pos, nextToken)) {
83          this.tokens[pos].setType('property').setClassName('hl-prop');
84      } else if (this.isSelector(pos)) {
```

```
85              this.tokens[pos].setType('selector').setClassName('hl-sel');
86          }
87      }
88
89      private parseAsterisk(pos: number) {
90          const prevToken = (pos > 0) ? this.tokens[pos - 1] : new Token();
91          const nextToken = (pos + 1 < this.tokens.length) ? this.tokens[pos + 1] : new Token
                ();
92
93          if ((prevToken.isLexemeEqualTo(')') || prevToken.isTypeEqualTo(['variable', 'number
                ']])) &&
94              (nextToken.isTypeEqualTo(['variable', 'function', 'number']))) {
95              return;
96          } else {
97              this.tokens[pos].setType('keyword2').setClassName('hl-k2');
98          }
99      }
100
101     private parseCoron(pos: number) {
102         const nextToken = (pos + 1 < this.tokens.length) ? this.tokens[pos + 1] : new Token
                ();
103         if (nextToken.isTypeEqualTo('coron')) this.parseCoron(pos + 1);
104         if (nextToken.isTypeEqualTo('selector')) this.tokens[pos].setType('selector').
                setClassName('hl-sel');
105     }
106
107     private isProperty(pos: number, nextToken: Token) {
108         if (!nextToken.isTypeEqualTo('coron')) return false;
109
110         const semicPos = this.tokens.findIndexByType('semic', pos);
111         const lcurPos = this.tokens.findIndexByLexeme('{', pos);
112
113         return semicPos > 0 && semicPos < lcurPos || lcurPos < 0;
114     }
115
116     private isSelector(pos: number) {
117         const lcurPos = this.tokens.findIndexByLexeme('{', pos);
118         const semicPos = this.tokens.findIndexByType('semic', pos);
119         return lcurPos > 0 && lcurPos < semicPos || semicPos < 0;
120     }
121 }
```

本方針の利点は，あらかじめ最小単位で字句解析が終了するため，本来の実装方針にマッチしたシンタックスハイライタの構成を取れる点である．トークン解析において字句解析を行うことがないため，コード全体の見通しも方針 1 と比べて良い．

ただし，先述した通りトークン解析は複雑であるため，一回のトークン列走査では全てのトークンを確定させることができない．そこで，トークン列の走査を段階的に二回行うことで，最終的に解析を完了させている．

5.9 C ハイライタの実装 CHighlighter

C ハイライタでは，識別子 identifier を「変数」,「関数」,「マクロ定数」,「構造体，共用体，typedef 名」,「enum メンバ」の 5 つに分類する必要がある．そこでまず，C における変数スコープ，及び名前空間について整理する．

- 変数のスコープはブロック単位であり，typedef 名にもこれは適用される．
- bf 同一スコープにおいては typedef 名を識別子 (変数名，関数名) として使用してはならない．

以下は検証用プログラムである．

```
 1: #include <stdio.h>
 2:
 3: typedef char alias;
 4:
 5: // void alias(void) {}  // Error
 6:
 7: struct alias {
 8:     int a;
 9: };
10:
11: int main(void) {
12:     int alias[100];
13:     printf("%ld\n", sizeof(alias));  // Output: 400
14:
15:     typedef char type;
16:
17:     type a = 'A';
18:     printf("%c\n", a);  // Output: A
19:
20:     // int type;  // Error
21:
22:     {
23:         typedef int type;   // Shadowing the outer 'type'
24:         type b = 42;
25:         printf("%d\n", b);  // Output: 42
26:     }
27:
28:     // printf("%d\n", b);  // Error
29:
30:     type c = 'C';
31:     printf("%c\n", c);  // Output: C
32:
33:     return 0;
34: }
```

図 25 に，C ハイライト用クラス CHighlighter を示す．

図 25　C ハイライタ (c.ts)

```
1  import Syntaxhighlighter from '../base';
2  import Namespace from '../namespase';
3  import { Token } from '../token';
4  import { PatternList } from '../type';
5
6
7  const macroConstant = "EOF NULL sizeof stderr stdin stdout";
8  const alias = "FILE size_t";
9
10 const patternList: PatternList = {
11     comment: { regexp: /^(?:\/\/.*|\/\*[\s\S]*?\*\/|\/\*[\s\S]*)/, className: 'hl-cm' },
12     for: { regexp: /^for(?!\w)/, className: 'hl-k1' },
13     keyword: { regexp: /^(?:return|while|if|else|break|continue)(?!\w)/, className: 'hl-k1'
           },
14     define: { regexp: /^#define/, className: 'hl-pp' },
15     preprocessor: { regexp: /^#(?:include|if|endif|ifdef|ifndef)/, className: 'hl-pp' },
16     headerFile: { regexp: /^(?:<.+\.h>)/, className: 'hl-hf' },
```

```
17      typedef: { regexp: /^typedef(?!\w)/, className: 'hl-k2' },
18      struct: { regexp: /^(?:struct|union)(?!\w)/, className: 'hl-k2' },
19      enum: { regexp: /^enum(?!\w)/, className: 'hl-k2' },
20      variableType: { regexp: /^(?:int|float|double|char|void)(?!\w)/, className: 'hl-vt' },
21      identifier: { regexp: /^[A-Za-z_]\w*/, className: null },
22      string: { regexp: /^(["']).*?\1/, className: 'hl-str' },
23      number: { regexp: /^(?:0x[0-9a-fA-F]+|[0-9]+(?:\.[0-9]+)?)\b/, className: 'hl-n' },
24      memberRef: { regexp: /^(?:\.|->)/, className: null },
25      lcur: { regexp: /^{/, className: null },
26      rcur: { regexp: /^}/, className: null },
27      comma: { regexp: /^,/, className: null },
28      semic: { regexp: /^;/, className: null },
29  };
30
31  export default class CHighlighter extends Syntaxhighlighter {
32      nsStack: Namespace;
33      isStackedByFunc: boolean;
34
35      constructor(src: string) {
36          super(src, patternList);
37          this.nsStack = new Namespace();
38          this.isStackedByFunc = false;
39
40          this.nsStack.register(macroConstant, 'macroConstant')
41                      .register(alias, 'alias');
42      }
43
44      parseTokens() {
45          for (let i = 0; i < this.tokens.length; i++) {
46              const token = this.tokens[i];
47              switch (token.type) {
48                  case 'comment': this.tokens[i].addTag('ignoreHighlightBrackets'); break;
49                  case 'define': this.parseDefine(i); break;
50                  case 'typedef': this.parseTypedef(i); break;
51                  case 'struct': this.parseStruct(i); break;
52                  case 'enum': this.parseEnum(i); break;
53                  case 'variableType': this.parseDeclaration(i); break;
54                  case 'identifier': this.parseIdentifier(i); break;
55                  case 'lcur': this.stackNamespace(i); break;
56                  case 'rcur': this.popNamespace(i); break;
57              }
58          }
59
60          this.tokens.forEach(({ lexeme, type }, i) => {
61              if (type === 'identifier') {
62                  if (this.nsStack.has(lexeme, 'function')) {
63                      this.tokens[i].setType('function').setClassName('hl-f');
64                  } else {
65                      this.tokens[i].setType('variable').setClassName('hl-v');
66                  }
67              }
68          });
69
70          this.highlightBrackets()
71              .highlightEscapeSequence()
72              .highlightConvertionSpecifier();
73      }
74
75      private parseDefine(kwPos: number) {
```

```
76          if (this.tokens.length <= kwPos + 1) return;
77
78          const ident = this.tokens[kwPos + 1];
79          this.tokens[kwPos + 1].setType('macroConstant').setClassName('hl-mc');
80          this.nsStack.append(ident.lexeme, 'macroConstant');
81      }
82
83      private parseTypedef(kwPos: number) {
84          if (this.tokens.length <= kwPos + 1) return;
85
86          const nextToken = this.tokens[kwPos + 1];
87          const identPos = (nextToken.isTypeEqualTo(['struct', 'enum']))
88              ? this.tokens.findIndexByType('rcur', kwPos) + 1
89              : kwPos + 2
90          if (this.tokens.length <= identPos) return;
91
92          this.tokens[identPos].setType('alias').setClassName('hl-al');
93          this.nsStack.append(this.tokens[identPos].lexeme, 'alias')
94      }
95
96      private parseStruct(kwPos: number) {
97          const nextToken = (kwPos + 1 < this.tokens.length) ? this.tokens[kwPos + 1] : new
                Token();
98          const followedByIdent = nextToken.isTypeEqualTo('identifier');
99          if (followedByIdent) this.tokens[kwPos + 1].setType('alias').setClassName('hl-al');
100
101         if (followedByIdent) {
102             const nextnextToken = (kwPos + 2 < this.tokens.length) ? this.tokens[kwPos + 2]
                    : new Token();
103             if (nextnextToken.isTypeEqualTo('identifier') || nextnextToken.isLexemeEqualTo
                    ('*')) this.parseDeclaration(kwPos + 1);
104         }
105     }
106
107     private parseEnum(kwPos: number) {
108         const nextToken = (kwPos + 1 < this.tokens.length) ? this.tokens[kwPos + 1] : new
                Token();
109         const followedByIdent = nextToken.isTypeEqualTo('identifier');
110         if (followedByIdent) this.tokens[kwPos + 1].setType('alias').setClassName('hl-al');
111
112         if (followedByIdent) {
113             const nextnextToken = (kwPos + 2 < this.tokens.length) ? this.tokens[kwPos + 2]
                    : new Token();
114             if (nextnextToken.isTypeEqualTo('identifier') || nextnextToken.isLexemeEqualTo
                    ('*')) this.parseDeclaration(kwPos + 1);
115         }
116
117         const lcurPos = this.tokens.findIndexByType('lcur', kwPos);
118         const isDeclaration = lcurPos !== -1 && lcurPos - kwPos <= 2;
119         if (!isDeclaration) return;
120
121         const from = lcurPos + 1;
122         const to = this.tokens.findIndexByType('rcur', kwPos) - 1;
123         for (let i = from; i <= to; i++) {
124             const token = this.tokens[i];
125             if (token.isTypeEqualTo('identifier')) {
126                 this.tokens[i].setType('enumMember').setClassName('hl-en');
127                 this.nsStack.append(token.lexeme, 'enumMember');
128             }
```

```
129         }
130     }
131 
132     private parseDeclaration(varTypePos: number) {
133         let i: number;
134         for (i = varTypePos + 1; i < this.tokens.length && this.tokens[i].isLexemeEqualTo
                ('*'); i++);
135         const isDeclaration = i < this.tokens.length && this.tokens[i].isTypeEqualTo('
                identifier');
136 
137         if (!isDeclaration) return; // (e.g.) cast operation
138 
139         const isFunctionDeclaration = i + 1 < this.tokens.length && this.tokens[i + 1].
                isLexemeEqualTo('(');
140         if (isFunctionDeclaration) {
141             this.tokens[i].setType('function').setClassName('hl-f');
142             this.nsStack.append(this.tokens[i].lexeme, 'function');
143             const semicPos = this.tokens.findIndexByType('semic', i);
144             const lcurPos = this.tokens.findIndexByType('lcur', i);
145             if (lcurPos > 0 && (lcurPos < semicPos || lcurPos < 0)) {
146                 this.nsStack = this.nsStack.createChildScope();
147                 this.isStackedByFunc = true;
148             }
149             const rbraPos = this.tokens.findIndexByLexeme(')', i);
150             for (let j = i + 2; j < rbraPos; j++) {
151                 const token = this.tokens[j];
152                 const prevToken = this.tokens[j - 1];
153                 if (this.nsStack.has(token.lexeme, 'alias') && prevToken.isLexemeEqualTo
                        (['(', ',',])) {
154                     this.tokens[j].setType('variableType').setClassName('hl-al');
155                 }
156             }
157         } else {
158             this.parseDeclarationStat(varTypePos);
159         }
160     }
161 
162     private parseDeclarationStat(varTypePos: number) {
163         const semicPos = this.tokens.findIndexByType('semic', varTypePos);
164         const rbraPos = this.tokens.findIndexByLexeme(')', varTypePos); // For function
                definition
165 
166         const from = varTypePos + 1;
167         const to = Math.min(semicPos, rbraPos) - 1;
168         for (let i = from; i <= to; i++) {
169             const token = this.tokens[i];
170             const prevToken = this.tokens[i - 1];
171             if (token.isTypeEqualTo('identifier') && (prevToken.isTypeEqualTo('comma') ||
                    prevToken.isLexemeEqualTo('*') || i === from)) {
172                 this.nsStack.append(token.lexeme, 'variable');
173                 this.tokens[i].setType('variable').setClassName('hl-v');
174             }
175         }
176     }
177 
178     private parseIdentifier(pos: number) {
179         const token = this.tokens[pos];
180         const prevToken = (pos > 0) ? this.tokens[pos - 1] : new Token();
181         const nextToken = (pos + 1 < this.tokens.length) ? this.tokens[pos + 1] : new Token
```

```
            ();
182
183         if (this.nsStack.has(token.lexeme, 'macroConstant')) {
184             this.tokens[pos].setType('macroConstant').setClassName('hl-mc');
185         } else if (this.nsStack.has(token.lexeme, 'enumMember')) {
186             this.tokens[pos].setType('enumMember').setClassName('hl-en');
187         } else if (prevToken.isTypeEqualTo('memberRef')) {
188             this.tokens[pos].setType('structMember').setClassName('hl-v');
189         } else if (this.nsStack.has(token.lexeme, 'alias')) {
190             this.parseAlias(pos); // determine if it is variable or alias
191         } else if (nextToken.isLexemeEqualTo('(')) {
192             this.tokens[pos].setType('function').setClassName('hl-f');
193         }
194     }
195
196     private parseAlias(aliasPos: number) {
197         const prevprevToken = (aliasPos - 1 > 0) ? this.tokens[aliasPos - 2] : new Token();
198         const prevToken = (aliasPos > 0) ? this.tokens[aliasPos - 1] : new Token();
199         const nextToken = (aliasPos + 1 < this.tokens.length) ? this.tokens[aliasPos + 1] :
                new Token();
200
201         if ((!prevToken.isTypeEqualTo('other') || aliasPos === 0) && // NOT operator, left-
                bracket, etc...
202             (nextToken.isTypeEqualTo('identifier') || nextToken.isLexemeEqualTo('*'))) { //
                    variable declaration
203             this.tokens[aliasPos].setType('alias').setClassName('hl-al');
204             this.parseDeclaration(aliasPos);
205         } else if (prevprevToken.isLexemeEqualTo('sizeof') && this.nsStack.getTypeByName(
                this.tokens[aliasPos].lexeme) === 'alias') { // operand of sizeof()
206             this.tokens[aliasPos].setType('alias').setClassName('hl-al');
207         } else if (prevToken.isLexemeEqualTo('(')) { // cast operation
208             let i: number;
209             for (i = aliasPos + 1; i < this.tokens.length && this.tokens[i].isLexemeEqualTo
                    ('*'); i++);
210             if (i + 1 >= this.tokens.length) return;
211             if (this.tokens[i].isLexemeEqualTo(')') && this.tokens[i + 1].isTypeEqualTo(['
                    identifier']))
212                 this.tokens[aliasPos].setType('alias').setClassName('hl-al');
213         }
214     }
215
216     private stackNamespace() {
217         if (this.isStackedByFunc) {
218             this.isStackedByFunc = false;
219             return;
220         }
221         this.nsStack = this.nsStack.createChildScope();
222     }
223
224     private popNamespace() {
225         this.nsStack = this.nsStack.escapeScope();
226     }
227 }
```

実装のポイントは，同一名称の typedef 名と変数を名前空間の管理により適切にハイライトしている点である．例えば，キャスト演算や変数宣言，sizeof() の際にトークンの並びや名前空間 nsStack を元に属性を振り分けている．

また，ヘッダファイルに登録されているマクロ定数や型エイリアスは constructor で文字列を元にグ

ローバルスコープに登録するようにしている．

実装において苦労した点は，型エイリアスによるポインタ変数の宣言である．トークン解析の際には，トークンの並びが"identifier * identifier ..."となるため，同一名称の変数が存在する場合に算術式との区別が付かない．そこで，直前のトークンが')' や'=' 等の演算記号でない場合は変数宣言であると解釈したのだが，無理矢理な実装となってしまった (201 行目)．

関数定義の場合は，{によるブロック形成が行われる前に変数宣言が行われる可能性がある (仮引数宣言)．そこで，関数定義の開始時にはその時点でスコープを形成し，{によるブロック形成をスキップする方針とした．

次に，改善点を二つ述べる．一つ目は，for 文初期化式での変数宣言への対応である．for 文終了時での適切なスコープの脱退が課題であり，現在は実装ができていない．これは，C においては現状のままでも特に問題なくハイライトを施すことができる．しかし，JS 等の言語においては適切なスコープの形成・脱退を行わなければ，同一名称の場合に定数か変数を判断できないため，改善が必要である．二つ目は，リファクタリングである．現コードでは，エイリアスによる変数宣言に関するハイライト等，見通しの悪い実装が多々ある．そこで，Token のタグの活用等を含めて実装方針を再検討することも必要であろう．

今後の展望としては，別ファイルで定義した定数等もハイライトできるよう，独自の形式言語で名前空間に登録する機能が考えられる．

最後に，図 2 の C プログラムを与えた結果得られる nsStack を図 3 に示す．

```c
#define SIZE

typedef int integer;

typedef enum {
    SUN,
    MON,
} Date;

integer sum(integer, int);

int main(void) {
    integer integer;
    integer = sum(1, 2);
    return 0;
}

integer sum(integer a, int b) {
    return a + b;
}
```

図 2　**ハイライトの例**

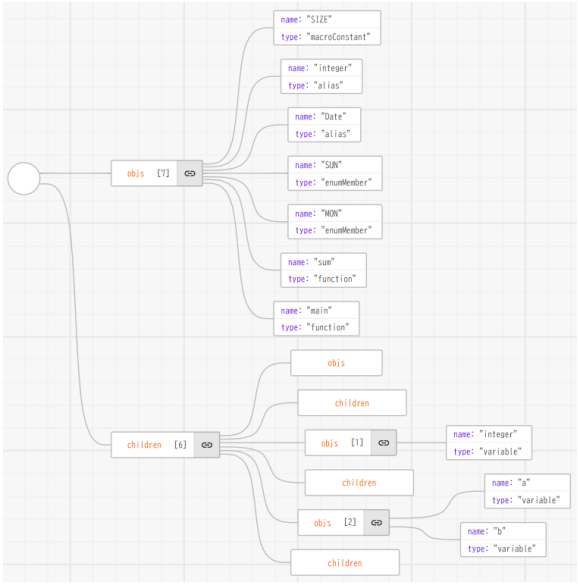

図 3　**得られる名前空間**

6　おわりに

以上の流れで Web シンタックスハイライタを作成したが実行環境の構築から作成方針の検討・実装に至るまで，非常に多くの学びがあった．しかし，更なるコードの改善やリファクタリング等，課題も多く残っている．今後はこれらの課題を改善しつつ，対応するソースコードを増やす等のアップデートを行っていきたい．

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] 名前空間,
https://learn.microsoft.com/ja-jp/cpp/c-language/name-spaces?view=msvc-170,
2025/1/26.

[17] JSON CRACK, https://jsoncrack.com/editor, 2024/1/26.