

# Web SyntaxHighlighter with TypeScript

## v2.0 系

2025 年 4 月 7 日

### 1 概要

v2.0 系では「独自マークダウンによる名前空間へのオブジェクト登録」を行う機能を作成した．本レポートではそこで，v1.0 系からの変更点をまとめていく．

### 2 オブジェクトの登録とは

まず，本システムにおけるオブジェクト登録の必要性について説明する．

Web コンテンツにおいて，ソースコードを分割して掲載することはよくあるパターンである．そこで，すでに定義済みとした変数や定数を，定義の記述を行うことなく適切にハイライトすることが必要である．

しかし，本プロジェクトでは (C 言語においては) 変数や定数の定義の際，名前空間に登録することでその後のハイライトに活用している．そのため，ソースコード中に定義の記述が無い場合にハイライトを施すことができない．

そこで，独自のマークダウンにより定義済みの変数や定数をマークダウンで記述する方針とした．

### 3 マークダウンの仕様

今回作成するマークダウンの仕様は以下の通りである．

1. マークダウンはソースコード冒頭に記述するものとする．
2. %==で囲まれた範囲はマークダウンとして判断される．
3. マークダウンは % に続いてオブジェクトのタイプを記述し，スペース区切りでオブジェクトを指定する．

例えば，C におけるマークダウンは以下のように記述できる．

```
%==  
%macro      LEN MAX MIN  
%alias      date boolean  
%enumMember SUN MON TUE  
%==
```

ここで，macro，alias，enumMember がオブジェクトのタイプであり，その後続く LEN 等がハイライト対象となる定数やユーザ定義型となる．

### 3.1 Cにおけるオブジェクトタイプ

Cにおける，指定可能なオブジェクトのタイプは表 1 のように定義した．

表 1 C におけるオブジェクトタイプ

オブジェクトタイプ	概要
macro	マクロ定数
alias	ユーザ定義型
enumMember	列挙型メンバ
function	関数
variable	変数

このように，5 種類のタイプを指定可能だが，変数と関数に関しては特にマークダウンで指定せずとも，トークン解析において適切なハイライトは可能である．ただし，関数の引数に関数を渡すといった特別な場合は，%function の指定が必須である．

ここで，マークダウンに使用するオブジェクトタイプは CHighlighter 実装時のオブジェクトタイプと揃えている点がポイントである．これにより，名前空間への登録を簡潔に実装可能である．

## 4 プログラムの実装

マークダウン記法を導入するために行った変更は主に 2 つである．

1 つ目は，基底クラス SyntaxHighlighter にマークダウンを解析し，オブジェクトを登録するメソッド `externDefine()` の実装である．当メソッドを図 1 に示す．

図 1 index.html(背景部分抜粋)

```
1 externDefine() {
2     const match = this.src.match(/^==\n([\s\S]*)\n%==/);
3     if (!match) return new Namespace();
4
5     const ns = new Namespace();
6     const rules = match[1].split('\n');
7     for (let i = 0; i < rules.length; i++) {
8         let rule = rules[i];
9         if (!rule.startsWith('%')) continue;
10
11         for (let j = i + 1; j < rules.length; j++) {
12             if (rules[j].startsWith('%')) break;
13             rule = rule.concat(' ', rules[j]);
14         }
15
16         const match = rule.match(/%(\S+)\s+([\s\S]*)/);
17         if (!match) continue;
18         ns.register(match[2], match[1]);
19     }
20
21     // If you make the markdown visible, you comment out these code below.
22     this.src = this.src.slice(match[0].length);
```

```

23     this.tokens.shift();
24     while (!this.tokens.shift()?.isTypeEqualTo('externSep'));
25
26     return ns;
27 }

```

externDefine() は NameSpace オブジェクトを受け取り, % から始まるマークダウンを解析して名前空間に登録する. また, 登録後にマークダウン部をトークン列とソースコードから削除することで, コンテンツとしてマークダウンが残らないようにしている.

2 つめは, CHighlighter の名前空間の初期化を externDefine() を用いて行った点である. ここでは, 37 行目を以下のように変更した.

```
this.nsStack = this.externDefine();
```

## 5 実行例

以下のソースコードを入力した結果を図 1 に, マークダウンを記述しなかった場合の結果を図 2 に示す.

```

%==
%macro A B
%alias date
%enumMember SUN
%==
int a = A, b = B;
date d = SUN;

```

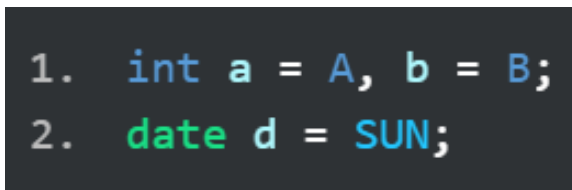


図 1 マークダウンあり

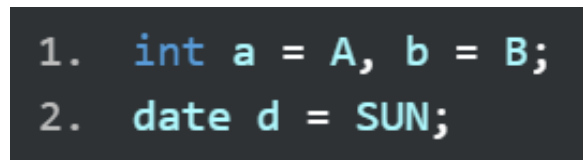


図 2 マークダウンなし

このように, マークダウンでオブジェクトタイプを指定することにより, 正しいハイライトが可能となっている.

## 6 おわりに

以上の流れで, 比較的少ない変更でマークダウン記法を導入することができた. 今後はリファクタリングや対応するソース言語の拡大に取り組んでいきたい.