

Visual Studio Code Ninja Guide

Atsushi Morimoto (@74th) 著

目次

はじめに『秘伝の VSCode 操作術』	5
1. エクスプローラービューとエディター	6
1.1 ファイルをすばやく開く方法	
1.2 ジャンプした後に戻ってくる	
1.3 対応する括弧にジャンプする	
1.4 選択領域を広げる	
1.5 指定した行番号に移動する	
1.6 次のエラーの行に飛び、すぐにクイックフィックスを適用する	
1.7 エディターを分割表示したり、プリセットのレイアウトを使う	
1.8 ファイルツリーから特定のファイルを非表示にする	
1.9 ファイルコピー時に数値をインクリメントする	
1.10 同じディレクトリのファイルを開く	
2. 検索	18
2.1 検索から特定のファイルを除外する	
2.2 特定のディレクトリ内で検索する	
3. Git	20
3.1 変更を部分的にステージに追加する、またはもとに戻す	
3.2 次の変更行に飛ぶ	
4. 自動系	22
4.1 保存時にフォーマットする	
4.2 デバッグ実行前にビルドを自動で実行する	
5. 環境、設定系	24
5.1 異なるマシン間で設定や拡張機能を同期する	
5.2 Setting Sync で設定を同期しつつ、ホストや OS ごとに異なる設定を行う	
6. 見たい系	27
6.1 使っているカラーテーマを一部分だけ変更する	
6.2 ワークスペースごとに色を変える	
7. キーボードショートカットとスニペット	29
7.1 サイドバー（左タブ）の切り替え	
7.2 左右のタブグループ、サイドバーにフォーカスを移す	

7.3 パネルのタブを切り替える	
8. その他	33
8.1 TODO コメントを管理する	
8.2 スペルミスをチェックする	
9. 筆者が活用している設定	35
9.1 見た目、カーソル周りの設定	
9.2 エディター内の表示の設定	
著者 VSCode 関連著作紹介	37
商業誌: Visual Studio Code 実践ガイド	
商業誌: Visual Studio Code デバッグ技術	
同人誌: Visual Studio Code Remote Dev & Cloud Code Guide	
筆者について	41

はじめに『秘伝の VSCode 操作術』




今や、拡張機能や LSP によってソフトウェア開発のほとんどが VSCode 上で可能となりました。筆者は、ほとんど*1の業務を VSCode を使って開発しています。

VSCode の機能はマニュアルなしに直感的に使えるように工夫されていますが、VSCode の開発が進んだ現在では意外と知られていない機能も多いのではないのでしょうか。また、VSCode を自分好みにカスタマイズして、手に馴染むように強化している人も多いと思います。本書では、VSCode の入門書では扱われていないような、細かすぎるテクニックを解説します。これらによって、読者の VSCode の操作が 1 秒でも速くなることを目指しています。

また、筆者は他のユーザがどのように VSCode をカスタマイズし、日常の開発をブーストされているのかに非常に興味があります。そのために、まず筆者自身のテクニックを公開してみようというのが本書執筆の動機です。本書で紹介していない VSCode のテクニックのブログ記事やスライド、同人誌等があればぜひお知らせください。

日本での VSCode コミュニティーを活性化するべく 2019 年末から、VSCode Meetup が開催されています。VSCode の知見を得たい、もしくは共有したい方はぜひ参加してみてください。これまでのイベントではリモートでも参加できるようストリーミング配信を行っており、イベント後にスライドや動画がアップロードされています。イベントページの URL は <https://vscode.connpass.com/> です。

本書は VSCode の 2020 年 2 月時点のバージョン 1.42.0 でのテクニックとなっています。

本書では、OS ごとに異なるキーボードショートカットを、macOS , Windows , Linux  のアイコンを使って示します。

Visual Studio Code は Microsoft 社の製品です。また、本書で紹介する拡張機能のライセンスについては、マーケットプレースの各拡張機能のページを参照下さい。本書は Microsoft 社、および拡張機能の開発者とは無関係のファンブックになります。本書の内容について Microsoft 社、および各拡張機能の開発者に問い合わせることはご遠慮いただき、筆者に問い合わせ下さい。

*1 Python、Go、Kubernetes など。Android 開発だけは Android Studio を使っています。

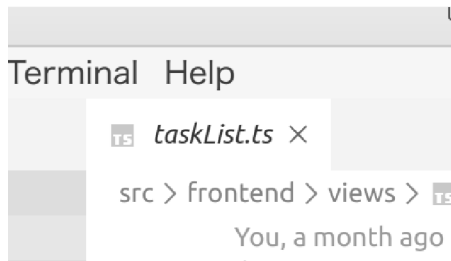
1. エクスプローラービューとエディター

1.1 ファイルをすばやく開く方法




VSCode では目的のファイルを開く方法として、複数の方法が用意されています。状況に応じてすばやくファイルを開くことができると開発に集中できます。

方法 1 : エクスプローラービューから

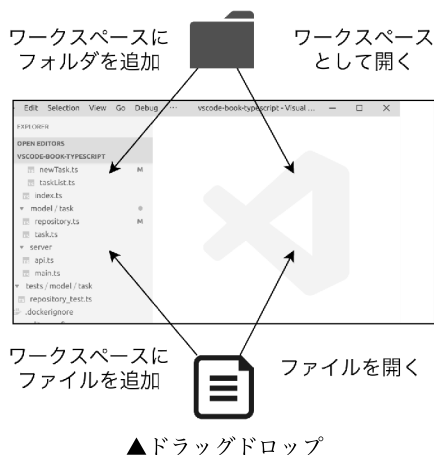
エクスプローラービューのファイルツリーでファイルをクリックすると、そのファイルがエディターで開かれます。 ファイルをクリックすると、そのエディターのタブのファイル名は斜体で表示されます。 これは、一時的に開いているファイルで、別のファイルを開くとこのファイルは閉じられます。



▲一時的に開いているファイルのタブ（ファイル名が斜体で表示）

一時的に開かれているファイルを閉じないようにするには、 :Cmd+S、 /  :Ctrl+S で保存するか、ファイルツリーのファイル名をダブルクリック、もしくはエディターのタブのファイル名をダブルクリックします。

方法 2 : ファイラーからドラッグドロップ



macOS の Finder、Windows のエクスプローラー、Linux の Gnome3 Files から VSCode にファイルをドラッグドロップすることができます。そして、ドロップ先によって機能が異なります。

エディターの中にファイルをドロップしたときには、そのファイルをエディターで開くことができます。また、エディターの中にフォルダーをドロップした場合は、そのフォルダーをワークスペースとして開くことができます。この時、現在開いているワークスペースは一度閉じられることに注意してください。

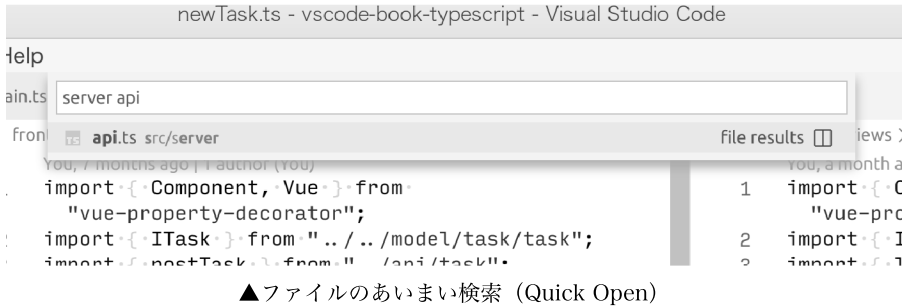
一方、ファイルやフォルダーをエクスプローラービューにドロップした時には、そのファイルやフォルダーがファイルツリーに追加されます。

また、ファイルやフォルダーをパネルのターミナルタブにドロップすると、ターミナルにフルパスが記入されます。

方法 3 : Go To File のあいまい検索を使う (Quick Open)




ワークスペースからあいまい検索でファイルを検索することができます。

1. エクスプローラービューとエディター



▲ファイルのあいまい検索 (Quick Open)

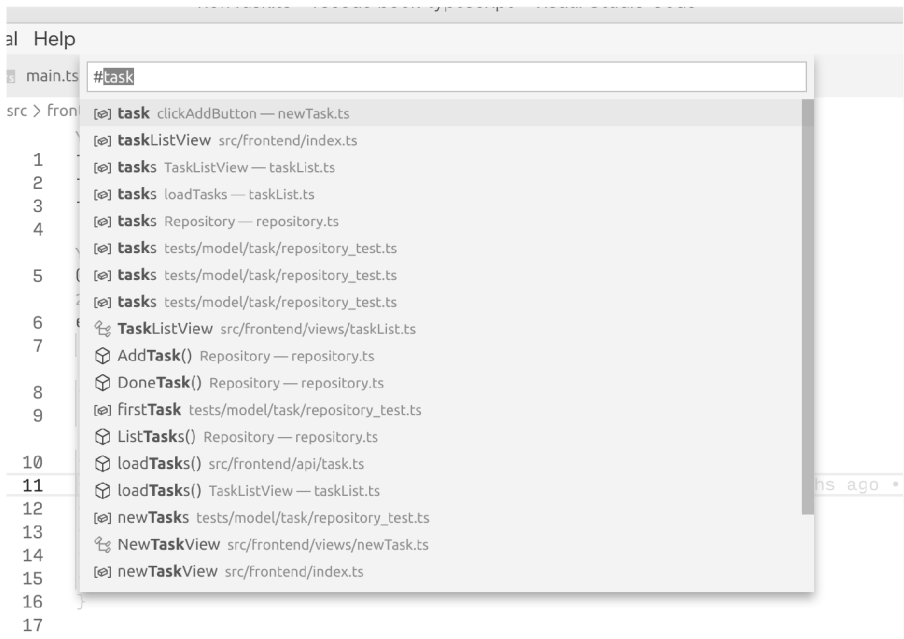
リストに表示されるファイルの右側にフォルダーのパスが表示されます。ここにはディレクトリの名前を入れて検索することもできます。ただし、ファイル名の前にディレクトリ名を入れる必要があることに注意してください。

- ファイルをあいまい検索する
 - コマンド: Go to file... (workbench.action.quickOpen)
 - ショートカット:  Cmd+P,  Ctrl+P,  Ctrl+P

このあいまい検索ができる機能には「Quick Open」という名称です。はじめに文字を入れることで機能を変えることができます。文字と機能の対応は「?」をいれると確認できます。

方法 4: シンボルであいまい検索

TypeScript でのクラスや名前空間は VSCode 内ではシンボルとして扱われます。これらのシンボルはキーボードショートカットかコマンド「Go To Symbol in Workspace...」を押すと、ワークスペース内のシンボルをあいまい検索し、シンボルが記述されたファイルを開くことができます。



▲シンボルであいまい検索

- シンボルを検索する
 - コマンド: Go to Symbol in Workspace... (workbench.action.showAllSymbols)
 - ショートカット: Cmd+T, Ctrl+T, Ctrl+T
- ファイル内のシンボルを検索する
 - コマンド: Go to Symbol in File... (workbench.action.gotoSymbol)
 - ショートカット: Cmd+Shift+O, Ctrl+Shift+O, Ctrl+Shift+O

また、Quick Open からは、ワークスペース内のシンボル検索は「#」、ファイル内のシンボル検索は「@」で呼び出すことができます。

方法 5 : code コマンド

VSCode をインストールすると code コマンドが使えますが、このコマンドでファイルを開くこともできます。code コマンドの引数にファイルを指定すると、VSCode が起動していない場合、新しく起動してそのファイルを開きます。既に VSCode が起動している場合は、その起動しているウィンドウの中に表示されます。フォルダーを指定すると、新しいウィンドウでフォルダーが開くため注意が必要です。

ターミナルタブ内でも code コマンドを使うことができるため、ターミナルタブを使っているならばそのまま code コマンドを使ってファイルを開くことができます。

方法 6 : 定義や参照で開く

TypeScript では他のソースから引用したクラス名に対して、F12 キーやコマンド「Go To Definition」、もしくはクラス名の単語を Ctrl キーを押しながらクリックすると、定義先のファイルを開くことができます。また、そのクラス名を右クリックして、「Go To Reference」を選ぶと、クリックした場所に定義を参照しているファイルが開かれます。



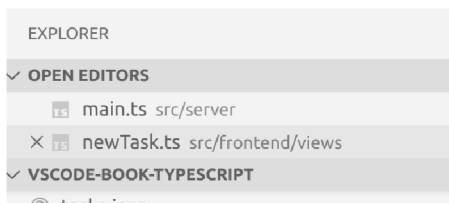
▲参照の検索

このエディター内ウィンドウをダブルクリックするとそのファイルが開きます。

- 定義にジャンプ
 - コマンド: Go to Definition (editor.action.revealDefinition)
 - ショートカット: F12, F12, F12
- 参照を表示する
 - コマンド: Go to References (editor.action.findReferences)
 - ショートカット: 割当なし

方法 6 : エディタータブで開いているファイルを開く

エディタータブで既に開いているファイルをアクティブにしたい場合は、そのファイルのタブをクリックします。このタブで開いているファイルの一覧はエクスプローラービューにもあるため、ここから見つけてクリックすることもできます。



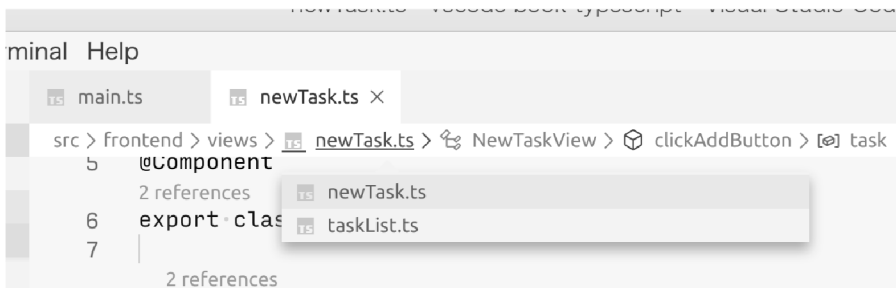
▲開いているファイル

この一覧についてもあいまい検索を行うことができます。 Cmd+P, / Ctrl+P を押し `edt` と入力するとエディターで開いているファイルの一覧が検索できます。 コマンド「View: Show All Editors」でも行うことができるので、よく使われる方はこれをキーボードショートカットに登録すると良いでしょう。

- エディターで開いているファイルをあいまい検索する
 - コマンド: View: Show All Editors By Appearance (`workbench.action.showAllEditors`)
 - ショートカット: Alt+Cmd+Tab, Ctrl+K Ctrl+P, Ctrl+K Ctrl+P
- 現在のエディターグループで開いているファイルをあいまい検索する
 - コマンド: View: Show Editors in Active Editor By Most Recently Used (`workbench.action.showEditorsInActiveGroup`)
 - ショートカット: 割当なし

方法 7 : 階層リンク

エディターの上部には階層リンク（パンくずリスト）が表示されます。階層リンクには、ディレクトリやファイルが表示されています。ここをクリックすると、ディレクトリやこのファイルと兄弟関係にあるファイルを開くことができます。



▲パンくずリスト

階層リンクが表示されない場合は以下の設定を追加して下さい。

1. エクスプローラービューとエディター

```
// settings.json
{
  "breadcrumbs.enabled": true,
}
```

方法 8 : 問題パネル

問題パネルには、リントツールのエラーが表示されます。多くの拡張機能では現在開いているファイルのエラーが表示されますが、参照しているファイルのエラーも表示することがあります。その場合、エラーをクリックすることでファイルが開きます。



▲問題パネル

1.2 ジャンプした後に戻ってくる

いくつか別のファイルにジャンプする方法を説明しましたが、再び元のファイルに戻ってくるコマンドとして「Go Back」が用意されています。

- 戻る/進む
 - コマンド: Go Back/Forward (workbench.action.navigateBack/navigateForward)
 - ショートカット:  Ctrl+-/Ctrl+Shift+-,  Alt+Right,  Ctrl+Alt+:/Ctrl+Shift+-




筆者は、F12 で定義に飛んだ後に戻ることが多いため、以下のように Shift+F12 にこのコマンドを割り当てています。

```
// keybindings.json
{
  {
    "key": "shift+f12",
```

```
"command": "workbench.action.navigateBack"
  },
}
```

1.3 対応する括弧にジャンプする



階層構造のあるプログラムを記述する時、対応する括弧にジャンプしたいことがあります。カーソルの括弧と対応する括弧にジャンプするショートカットがあります。

- 対応する括弧にジャンプする
 - コマンド: Go to Bracket (editor.action.jumpToBracket)
 - ショートカット:  Cmd+Shift+\\,  Ctrl+Shift+\\,  Ctrl+Shift+\\

なお、拡張機能「Bracket Pair Colorizer 2」を使うと、対応する括弧が色分けされるため、見やすくなります。




1.4 選択領域を広げる

構文をみて、選択領域を広げることができます。例えば、名前空間 → クラス → メソッド → if 文が括弧で入れ子になっている場合、if 文 → メソッド → クラス → 名前空間に選択範囲を広げたり、逆に狭くすることができます。

- 選択範囲を広げる/狭くする
 - コマンド: Expand/Shrink Selection (editor.action.smartSelect.expand/shrink)
 - ショートカット:  Ctrl+Shift+→/←,  Alt+Shift+→/←,  Ctrl+Alt+→/←

1.5 指定した行番号に移動する

トレースバックなどでは、エラーになった箇所が行番号で表示されることがあります。コマンド「Go to Line...」を使うと、行番号を入力してその行に飛ぶことができます。

- 指定した行に飛ぶ
 - コマンド: Go to Line... (workbench.action.gotoLine)
 - ショートカット:  Ctrl+G,  Ctrl+G,  Ctrl+G

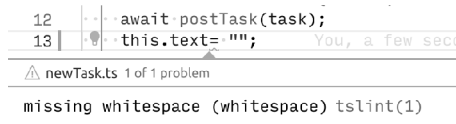
Quick Open からは「:123」と入力すると 123 行目にジャンプします。

1.6 次のエラーの行に飛び、すぐにクイックフィックスを適用する

リントの結果の修正など、ファイルの中に複数のエラーがある場合、1 つを修正した後、次に次のエラーに移動したいことがあります。以下のコマンドやキーで次のエラーにジャンプします。

- 次の/前のエラーを表示する
 - コマンド: Go to Next/Previous Problem (editor.action.marker.nextInFiles/prevInFiles)
 - ショートカット:  F8/Shift+F8,  F8/Shift+F8,  F8/Shift+F8

そのエラーに対してクイックフィックス（コードアクション）がある場合、左側に電球マークが表示されます。



▲クイックフィックス（コードアクション）

このアイコンをクリックするか、キーボードショートカットを押すと、エラーを修正する候補が表示されます。

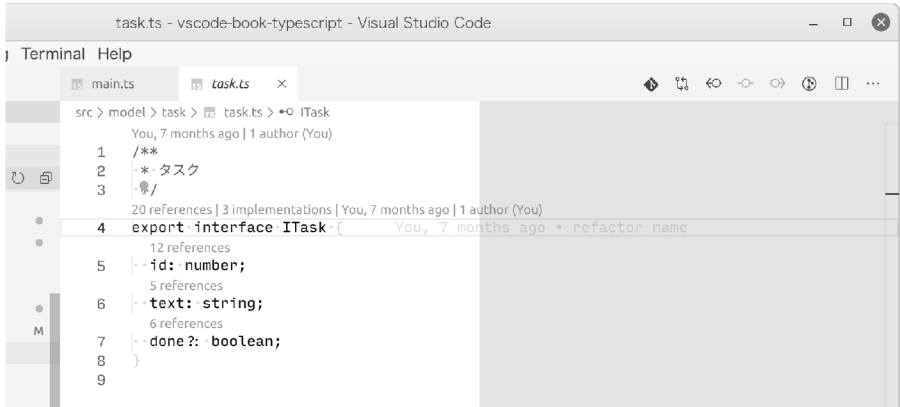
- クイックフィックス（コードアクション）を適用する
 - コマンド: Quick fix... (editor.action.quickFix)
 - ショートカット:  Cmd+.,  Ctrl+.,  Ctrl+.

筆者は F8 キーを押して次のエラーに飛ぶことが多いため、クイックフィックスのキーを F8 キーに近い F9 キーに設定しています。

```
// keybindings.json
[
  {
    "key": "f9",
    "command": "editor.action.quickFix",
    "when": "!inDebugMode && editorTextFocus"
  },
]
```

1.7 エディターを分割表示したり、プリセットのレイアウトを使う

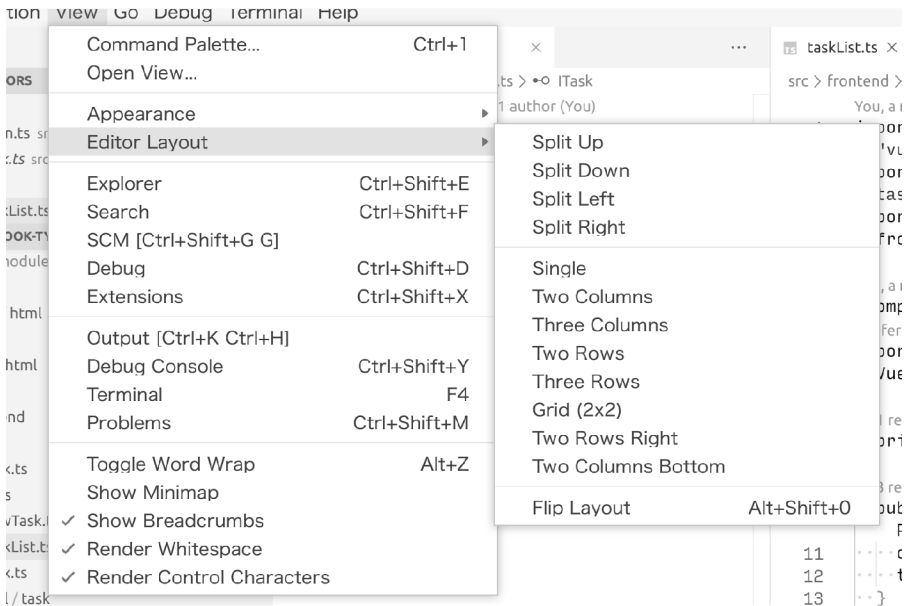
エディター内にファイルなどをドラッグするとファイルを開けると前述しましたが、この時エディターの端でドラッグドロップすると、左右上下に分割して表示することができます。タブをドラッグした際も同様です。



▲ドラッグでエディター分割

また、ウィンドウメニューの View→Editor Layout には、多くのプリセットが用意されています。メニューと同じコマンドも用意されています。

1. エクスプローラービューとエディター



▲エディターレイアウトのプリセット

田の形に 4 分割する 2x2 も用意されています。逆に、分割を解除するコマンド「View: Single Column Editor Layout」もあります。

1.8 ファイルツリーから特定のファイルを非表示にする

コンパイルされたファイルや、npm の `node_modules` ディレクトリなど、編集時には表示したくないファイルがあると思います。設定「`files.exclude`」を使うと、ファイルツリーに表示したくない除外ファイルを指定することができます。`.git` ディレクトリなどは VSCode のデフォルトで表示されなくなっています。

設定は以下のように、ファイルパターンをキーに、値を `true` と入力します。逆に、ユーザー設定では除外にするが、このプロジェクトでは表示したいファイルがある場合には、`false` を設定します。




```
// .vscode/settings.json
{
  "files.exclude": {
    "**/node_modules": true,
  }
}
```


1.9 ファイルコピー時に数値をインクリメントする

VSCode のエクスプローラービュー上でファイルをコピーし、貼り付けを行うと、末尾に copy のついたファイル名になります。「8.article.md」のようにファイル名に数値がついていた場合、copy を付けずに「9.article.md」と自動でインクリメントしてくれる機能があります。以下の設定を追加すると有効になります。

```
// settings.json
{
  "explorer.incrementalNaming": "smart",
}
```

1.10 同じディレクトリのファイルを開く

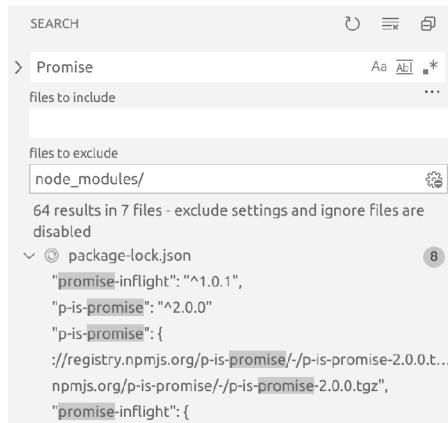
エディターから  :Cmd+Shift+E、 /  :Ctrl+Shift+E でエクスプローラービューを開くと、エディターで開いていたファイルがファイルツリーで選択された状態になります。同じディレクトリのファイルが一覧された状態のため、ここからエクスプローラービューでテキストの入力、もしくは矢印キーで選択をして、Enter を押します。

2. 検索

2.1 検索から特定のファイルを除外する

node_modules ディレクトリなどを Search View から除外したい場合があると思います。

検索ビューには、「file to exclude」という除外するファイルパスを指定するエリアがあります。ここにファイルパスやディレクトリを入力すると、該当するファイルを検索から除外します。



▲file to exclude

「file to exclude」のエリアの右端のアイコンは、除外するファイルパスを設定「search.exclude」から取得します。ワークスペースの設定にこの設定を追加しておくと、他の開発者と共有できて便利です。

```
// .vscode/settings.json
{
  "search.exclude": {
    "**/node_modules": true,
    "out/js": true,
  }
}
```

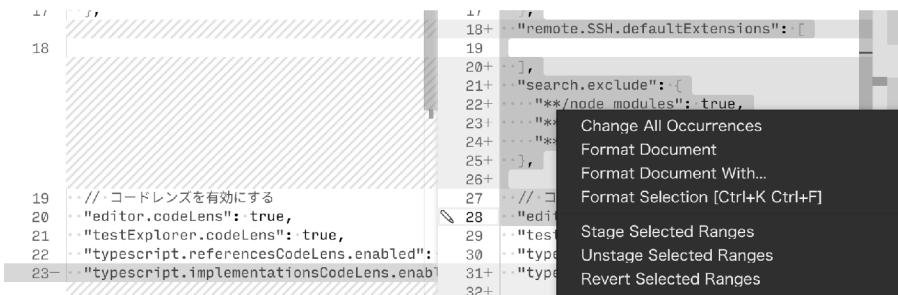
2.2 特定のディレクトリ内で検索する

検索ビューは「file to exclude」にファイルパスを追加すると、そのパスの範囲で検索を行います。ファイルツリー中のディレクトリの右クリックメニューから「Find in Folder」を選ぶと、そのディレクトリを「file to exclude」に追加して、そのディレクトリを対象に検索を行うことができます。

3. Git

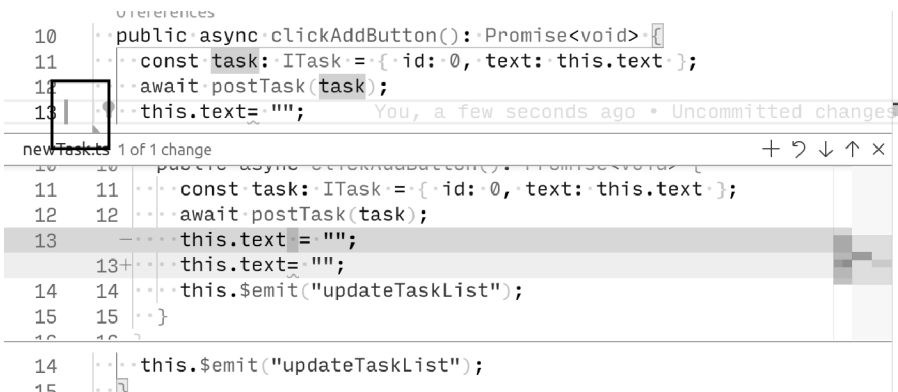
3.1 変更を部分的にステージに追加する、またはもとに戻す

Git では行単位で変更をステージングし、コミットすることができます。VSCode では、ソースコントロールビューでファイルの差分を表示している時に部分的に追加したい行を選択し、右クリックメニューで「Stage Selected Ranges」を選ぶと部分的にステージに追加することができます。また、変更の取消もここから行うことができます。



▲選択した行の右クリックメニュー

また、ステージングされていない変更がある場合、エディターの行番号とテキストの間にマークが付きます。これをクリックすると、変更をエディター内で確認することができます。このウィンドウの右上のボタンで、この範囲をステージに追加する、またはもとに戻すことができます。



▲マークの位置と、エディター内で開いた変更

3.2 次の変更行に飛ぶ

ステージングされていない変更ジャンプするコマンドがあります。このコマンドを使うと次の変更ジャンプすることができ、コミット前に変更を効率的に確認することに使えます。

- 次の/前の変更へジャンプ
 - Show Next/Previous Change (editor.action.dirtydiff.next/previous)
 - ショートカット:  Alt+F3/Alt+Shift+F3,  Alt+F3/Alt+Shift+F3,  Alt+F3/Alt+Shift+F3

4. 自動系

4.1 保存時にフォーマットする

ファイルの保存時にフォーマットを有効にするには、以下の設定を追加します。ワークスペースの設定に入れておくのも良いでしょう。

```
// .vscode/settings.json
{
  "editor.formatOnSave": true
}
```

また、行の末尾や、文末に入っている余分な改行を自動で削除したり、改行が不足する場合に追加してくれる設定があります。多くは、EditorConfig やフォーマットツールで制御と思いますが、フォーマットツールがないファイルの種類に対しても VSCode で適用できるようになります。

```
// settings.json
{
  // 末尾の空白は自動でトリムする
  "files.trimTrailingWhitespace": true,
  // ファイル最後の空の行はトリムする
  "files.trimFinalNewlines": true,
  // ファイルの最後には改行を入れる
  "files.insertFinalNewline": true,
}
```

4.2 デバッグ実行前にビルドを自動で実行する

デバッグの実行時には、事前に実行するタスクを設定できます。

デバッグ実行前に webpack を自動で行われるようにする例を示します。まず webpack を実行するタスクを以下のように作ります。

```
// .vscode/tasks.json
{
  "version": "2.0.0",
  "tasks": [
```

```

{
  "type": "shell",
  "label": "webpack build",
  "command": [
    "${workspaceFolder}/node_modules/.bin/webpack"
  ],
  "problemMatcher": [
    "$tsc"
  ]
},
]
}

```

タスクの名前は"webpack build"になります。デバッグの設定の"preLaunchTask"に、そのタスクの名前を設定します。

```

{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "Launch Chrome",
      "url": "http://localhost:8080",
      "webRoot": "${workspaceFolder}/public/html",
      "preLaunchTask": "webpack build",
      "sourceMapPathOverrides": {
        "webpack:///./src/*": "${workspaceRoot}/src/*"
      },
      "sourceMaps": true
    }
  ]
}

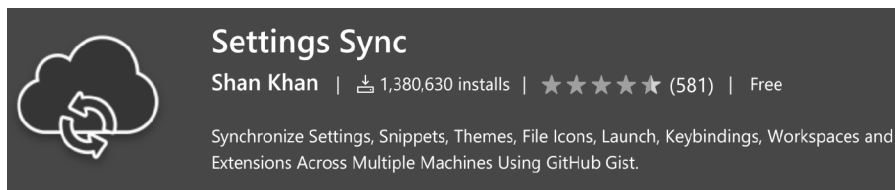
```

これでデバッグ開始前にタスクが実行されるようになります。

5. 環境、設定系

5.1 異なるマシン間で設定や拡張機能を同期する

複数のマシンで設定やインストールした拡張機能を同期させる拡張機能に「Setting Sync *1」があります。



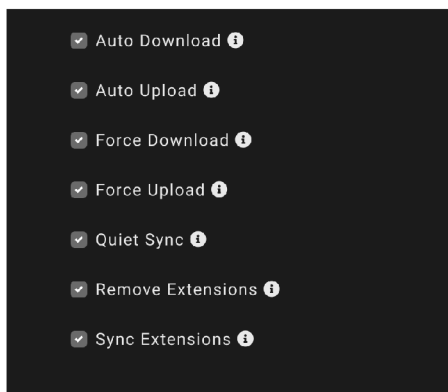
▲Setting Sync

Setting Sync は、gist に設定ファイルをコミットし、その設定ファイルを各マシンで同期させるものです。

gist の設定を済ませた後は、コマンド「Sync: Advanced Options」から、「Sync: Toggle Auto-Upload On Setting Change」と「Sync: Toggle Auto-Download On Startup」を実行すると、自動アップロード、ダウンロードを有効にできます。

また、設定だけではなく拡張機能も同期することができます。 コマンド「Sync: Advanced Options」から「Sync: Open Settings」を選択して設定画面を開き、「Sync Extensions」をクリックして有効化します。

*1 <https://marketplace.visualstudio.com/items?itemName=Shan.code-settings-sync>



▲Settings Syncの設定

5.2 Setting Sync で設定を同期しつつ、ホストや OS ごとに異なる設定を行う

Setting Sync にはホスト名、OS ごとに設定を使い分ける機能があります。以下のように `// @sync os=linux` というコメントを付けると、その下の行の設定は特定の OS の場合のみ有効になります。

```
// settings.json
{
  // @sync os=linux
  "python.pythonPath": "/home/linuxbrew/.linuxbrew/bin/python3",
  // @sync os=mac
  // "python.pythonPath": "/usr/local/bin/python3",
  // @sync os=windows
  // "python.pythonPath":
  "C:\\Users\\nnyn\\AppData\\Local\\Programs\\Python37\\python.exe",
}
```

その環境で有効でない設定はコメントアウトしておきます。OS によってコメントアウトが切り替えられ、それぞれの環境で有効と無効が切り替わります。

また、同一 OS であるが 2 台以上の PC で異なる設定をしたい場合、ホスト名を設定して設定を切り替えることができます。

コマンド「Sync: Advanced Options」から「Sync: Edit Extension Local Settings」を実行すると、ローカルの Setting Sync の設定ファイルを開くことができます。この設定に「hostName」という設定があり、これが環境名として使われます。

5. 環境、設定系

```
// syncLocalSettings.json
{
  // ...
  "hostName": "home",
}
```

先ほどの OS の分別と同じように、`// @sync host=home`というコメントを設定の前の行に指定します。OS と環境名の設定を同時に記述することも可能です。

```
// settings.json
{
  "cSpell.dictionaryDefinitions": [
    {
      "name": "my_work",
      // @sync host=work os=mac
      // "path": "/Users/atsushi.morimoto/dotfiles/CodeSpellChecker/work.txt"
      // @sync host=work os=linux
      // "path": "/home/atsushi.morimoto/dotfiles/CodeSpellChecker/work.txt"
      // @sync host=home os=mac
      // "path": "/Users/nnyn/dotfiles/CodeSpellChecker/work.txt"
      // @sync host=home os=linux
      "path": "/home/nnyn/dotfiles/CodeSpellChecker/work.txt"
    },
  ]
}
```

6. 見た目系

6.1 使っているカラーテーマを一部分だけ変更する

マーケットプレイスにあるテーマを使っていて、一部の色が見にくいことがあるでしょう。そのときは設定を使ってテーマに上書き設定をすることができます。

設定には、エディターの色を設定する「`editor.tokenColorCustomizations`」と、エディター以外のフレームの色を設定する「`workbench.colorCustomizations`」があります。

例えば「Monokai Pro」のコメントの色を変えたい場合、以下のように設定します。


```
// settings.json
{
  "editor.tokenColorCustomizations": {
    "[Monokai Pro]": {
      "comments": "#B6B6B6"
    }
  },
}
```

6.2 ワークスペースごとに色を変える

複数の VSCode のウィンドウを開いて作業する時、どのワークスペースの VSCode を使っているか迷うことがあります。ひと目でどのワークスペースかわかるようにするのは良い方法です。

ワークスペースの設定に、前の項目の設定を行うと色を変えることができます。

おしゃれな色にさっと変えたい場合、拡張機能「Peacock *1」を使うと便利です。



Peacock

John Papa | 487,536 installs | ★★★★★ (81) | Free

Subtly change the workspace color of your workspace. Ideal when you have multiple VS Code instances and you want to quickly identify which is which.

▲Peacock

*1 <https://marketplace.visualstudio.com/items?itemName=johnpapa.vscode-peacock>




6. 見た目系

コマンド「Peacock: Change to a Favorite Color」を実行すると色が選択でき、ワークスペースの設定にこの色の設定が反映されます。

7. キーボードショートカットとスニペット

7.1 サイドバー（左タブ）の切り替え

サイドバーを切り替えるキーボードショートカットには、デフォルトで簡単なキーが設定されています。

- 組み合わせ
 -  Cmd+Shift+?
 -  Ctrl+Shift+?
 -  Ctrl+Shift+?
- タブのキー
 - エクスプローラビュー: "E"xplorer
 - 検索ビュー: "S"earch
 - ソースコントロールビュー: "G"it
 - デバッグビュー: "D"ebug
 - 拡張機能ビュー: e"X"tensions

アクティビティーバーをマウスでクリックするのもいいですが、このショートカットを覚えておくとすばやく操作できます。

7.2 左右のタブグループ、サイドバーにフォーカスを移す

左右に複数のエディターを表示している時、素早く左右のエディターや、サイドバーに移動できるキーボードショートカットを用意しておくと便利です。

筆者の場合は、Ctrl+W と H/J/K/L キーの組み合わせで左右のエディターグループに移動できるようにしています。さらに、when 節で現在左端のエディターであるかを判別し、左端のエディターからさらに左に移動しようとするときサイドバーに移動できるようにしています。

```
// keybindings.json
{
  {
    // サイドバーの場合、すぐ右にある左端のエディターに移る
    "key": "ctrl+w l",
    "command": "workbench.action.focusFirstEditorGroup",
    "when": "sideBarFocus",
```

7. キーボードショートカットとスニペット

```
},
{
  // エディターの場合、右側のエディターに移る
  "key": "ctrl+w l",
  "command": "workbench.action.focusRightGroup",
  "when": "editorFocus",
},
{
  // 左側にまだエディターがある場合、左のエディターに映る
  "key": "ctrl+w h",
  "command": "workbench.action.focusLeftGroup",
  "when": "activeEditorGroupIndex != 1"
},
{
  // 左側のエディターの場合、サイドバーに移る
  "key": "ctrl+w b",
  "command": "workbench.action.focusSideBar",
  "when": "activeEditorGroupIndex == 1"
},
{
  // 縦に分割している場合、下に移る
  "key": "ctrl+w j",
  "command": "workbench.action.focusBelowGroup"
},
{
  // 縦に分割している場合、上に移る
  "key": "ctrl+w k",
  "command": "workbench.action.focusAboveGroup"
},
}
```

7.3 パネルのタブを切り替える

同様に、Ctrl+W H/L のキーボードショートカットを使ってパネルのタブを切り替えられると便利です。 パネルのタブの種類は `activePanel` で取得可能ですが、各タブをアクティブにするコマンドはそれぞれ異なっています。 また、ターミナルにおいては、`"terminalFocus"` を使ってターミナルタブにカーソルがあることを検知します。

```
// keybindings.json
{
  {
    // デバッグコンソール←ターミナル
    "key": "ctrl+w h",
    "command": "workbench.panel.repl.view.focus",
```

```

    "when": "terminalFocus",
  },
  {
    // 出力←デバッグコンソール
    "key": "ctrl+w h",
    "command": "workbench.action.output.toggleOutput",
    "when": "panelFocus && activePanel == 'workbench.panel.repl'",
  },
  {
    // 問題←出力
    "key": "ctrl+w h",
    "command": "workbench.panel.markers.view.focus",
    "when": "panelFocus && activePanel == 'workbench.panel.output'",
  },
  {
    // 問題→出力
    "key": "ctrl+w l",
    "command": "workbench.action.output.toggleOutput",
    "when": "panelFocus && activePanel == 'workbench.panel.markers'",
  },
  {
    // 出力→デバッグコンソール
    "key": "ctrl+w l",
    "command": "workbench.panel.repl.view.focus",
    "when": "panelFocus && activePanel == 'workbench.panel.output'",
  },
  {
    // デバッグコンソール→ターミナル
    "key": "ctrl+w l",
    "command": "workbench.action.terminal.focus",
    "when": "panelFocus && activePanel == 'workbench.panel.repl'",
  },
}

```

加えて、F4 キーなどの簡単なキーでパネルとエディターを行き来できるようにしておく
と便利です。

```

// keybindings.json
{
  {
    // エディター→パネル
    "key": "f4",
    "command": "workbench.action.focusPanel",
    "when": "!panelFocus"
  },
  {
    // エディター←パネル

```

7. キーボードショートカットとスニペット

```
    "key": "f4",  
    "command": "workbench.action.focusActiveEditorGroup",  
    "when": "panelFocus"  
  },  
}
```


8. その他

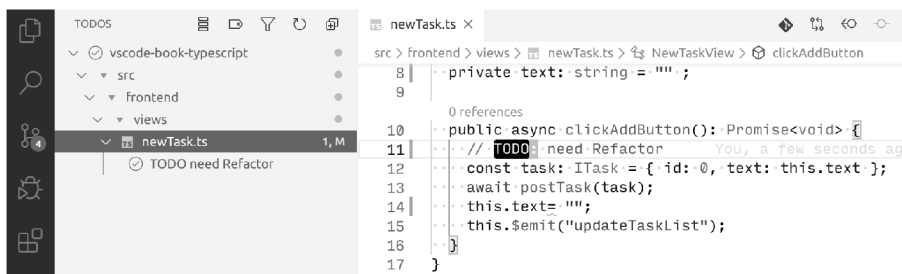
8.1 TODO コメントを管理する



▲Todo Tree

拡張機能「Todo Tree *1」をインストールすると、Todo Tree のタブがアクティビティバーに追加されます。この Todo Tree ビューでは、ソースコードの `// TODO: hoge hoge` とコメント中に `TODO:` とつけると、それを拾ってツリー状の UI で表示します。

また、その部分をハイライトし、見やすくすることもできます。



▲Todo Treeビュー

*1 <https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree>

8.2 スペルミスをチェックする



Code Spell Checker

Street Side Software | 📦 1,073,381 installs | ★★★★★ (114) | Free

Spelling checker for source code

▲Code Spell Checker

拡張機能「Code Spell Checker *2」を使うと、一般的な英単語のスペルミスをチェックしてくれます。

また、行区切りの辞書ファイルを作成し、単語を登録することもできます。

```
// settings.json
{
  "cSpell.dictionaryDefinitions": [
    {
      "name": "my_work",
      "path": "/home/nnyn/dotfiles/CodeSpellChecker/work.txt"
    },
  ],
}
```

*2 <https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker>

9. 筆者が活用している設定

VSCode には多数の設定がありますが、初期状態では有効になっていない、隠れている機能が多々あります。ここでは筆者が実際に活用している設定を紹介します。読者のみなさんが使っている設定と比較して、参考にしてもらえればと思います。

9.1 見た目、カーソル周りの設定

```
// settings.json
{
  // フォントの設定
  // 英語と日本語で別のフォントを設定するため、英語のフォントを先に書く
  "editor.fontFamily": "Liga InputMono,Han Code JP",
  // リガチャを有効にする
  "editor.fontLigatures": true,
  // フォントウェイトと、サイズ
  "editor.fontWeight": "400",
  "editor.fontSize": 14,
  // デバッグコンソールのフォントサイズ
  "debug.console.fontSize": 14,
  // ターミナルのフォントサイズ
  "terminal.integrated.fontSize": 14,

  // ハイコントラストのオリジナルカラーテーマ
  "workbench.colorTheme": "monokai-chacoral(HC)",
  // 有償の Monokai Pro のアイコンが上のテーマに合います
  "workbench.iconTheme": "Monokai Pro Icons",

  // カーソル
  "editor.cursorStyle": "line",
  // カーソルの点滅は複数種類定義があります
  "editor.cursorBlinking": "smooth",
  // カーソルの太さ
  "editor.cursorWidth": 3,
  // カーソルラインの表示を行番号まで囲う表示にする
  "editor.renderLineHighlight": "all",

  // メニューバーをタイトルバーにドッキングしてシンプルにする
  "window.titleBarStyle": "custom",
  // メニューバーは使わないけど、バージョン番号を表示するためだけに有効化している
  "window.menuBarVisibility": "default",
}
```

9. 筆者が活用している設定

筆者は「Monokai Charcoal high contrast *1」という、Monokai をベースに少し色の変更を加えたカラーテーマを作成して配布しています。ハイコントラストのウィンドウフレームを使っていることが特徴です。



▲Monokai Charcoal high contrast

9.2 エディター内の表示の設定

```
// settings.json
{
  // 長い行の折返しを有効にする
  "editor.wordWrap": "on",
  // 折返した行はインデントしてわかるようにする
  "editor.wrappingIndent": "indent",

  // 行番号を表示する
  "editor.lineNumbers": "on",
  // 折りたたみは表示しない
  "editor.folding": false,
  // スペース、タブを表示する
  "editor.renderWhitespace": "all",
  // 制御文字を表示する
  "editor.renderControlCharacters": true,
  // ミニマップは表示しない
  "editor.minimap.enabled": false,
  // パンくずリストは表示する
  "breadcrumbs.enabled": true,
  // コードレンズは有効化する
  "editor.codeLens": true,
  // カーソル位置の問題をステータスバーに表示する
  "problems.showCurrentInStatus": true,
}
```

*1 <https://marketplace.visualstudio.com/items?itemName=74th.monokai-charcoal-high-contrast>

著者 VSCode 関連著作紹介

商業誌: Visual Studio Code 実践ガイド



▲Visual Studio Code 実践ガイド

VSCode の解説書の決定版、『Visual Studio Code 実践ガイド』*1 を技術評論社より、2020 年 2 月に出版しました！ 基本的な VSCode の使い方もさることながら、プロダクトで Go、TypeScript、Python で開発してきたノウハウを詰めました。さらに拡張機能開発のガイドとして使えるようにし、最後は LSP の解説まで踏み込みます。以下に目次を掲載します。

- 第 1 部 Visual Studio Code の基本
 - 第 1 章 インストールと初期設定 —— Visual Studio Code を使いはじめる
 - 第 2 章 画面構成と基本機能 —— 直感的な画面に隠された多くの機能たち
 - 第 3 章 ビューとコマンドパレット —— 多彩な情報を整理し、簡単に呼び出す
 - 第 4 章 Git との連携 —— 基本操作から便利な拡張機能まで
 - 第 5 章 デバッグ機能 —— さまざまな言語のデバッグを直感的な UI で行う
 - 第 6 章 そのほかの機能 —— タスク、リント、スニペット、ターミナル
 - 第 7 章 リモート開発機能 —— 開発環境と実行環境の差分を抑える新機能
 - 第 8 章 カスタマイズ —— 柔軟な設定項目、ショートカットでより使いやすく
 - 第 9 章 拡張機能 —— 導入、管理、おすすめの拡張機能

*1 技術評論社 <https://gihyo.jp/book/2020/978-4-297-11201-1>

- 第 2 部 実際の開発で Visual Studio Code を使う
 - 第 10 章 TypeScript での開発 —— デフォルトで使えるフロントエンド／Web API アプリ開発機能たち
 - 第 11 章 Go での開発 —— 各種の開発支援ツールと連携した拡張機能
 - 第 12 章 Python での開発 —— Web API 開発にも、Jupyter 機械学習にも活用できる
- 第 3 部 拡張機能の開発と Language Server Protocol
 - 第 13 章 拡張機能開発の基本 —— Visual Studio Code の拡張ポリシーとひな形の作成
 - 第 14 章 実践・拡張機能開発
 - テキストを編集する拡張機能の開発
 - スニペットの拡張機能の開発
 - リントの拡張機能開発
 - カラーテーマの拡張機能の開発
 - 新しい UI を提供する拡張機能の開発
 - 第 15 章 自作の拡張機能を公開する —— 広く使ってもらうため必要なさまざまな事項
 - 第 16 章 Language Server Protocol —— エディター拡張のための次世代プロトコル
 - LSP とは
 - LSP の仕様と Visual Studio Code の動作
 - 言語サーバーとしてリントの拡張機能を作成する

こちらは全国の書店、Amazon などで販売中です。また、電子版は技術評論社のサイトから購入できます。

なお本同人誌（VSCode Ninja Guide）とこの商業誌は無関係の著作になります。本同人誌について、技術評論社に問い合わせることはおやめいただき、直接筆者*2 までお願いします。

商業誌: Visual Studio Code デバッグ技術

Visual Studio Code の 14 の言語と環境それぞれでのユニットテストや実行プログラムのデバッグ方法をまとめた『Visual Studio Code デバッグ技術』*3 を、インプレス R&D より 2018 年 12 月に出版しました。もとは技術書典 3 で頒布した本ですが、Ruby など

*2 Twitter: @74th, mail: site@74th.tech

*3 <https://nextpublishing.jp/book/10255.html>

の言語を追加した他、出版時の最新の内容へ更新しました。書店では注文が必要になりますが、Amazon で紙の書籍 *4 と Kindle 版 *5 を、達人出版会 *6、Booth *7 で PDF、ePub 版を販売しています。よろしければこちらもお手にとっていただければと思います。



▲Visual Studio Codeデバッグ技術

またこの本の内容である、本書でも紹介した VSCode のデバッグ設定に関する情報は、オープンソースとして github *8 でも公開しています(英語情報のみ)。

なお本同人誌とこの商業誌は無関係の著作になります。本同人誌について、インプレス R&D に問い合わせることはおやめいただき、直接筆者までお願いします。

*4 <https://www.amazon.co.jp/dp/4844398628/>

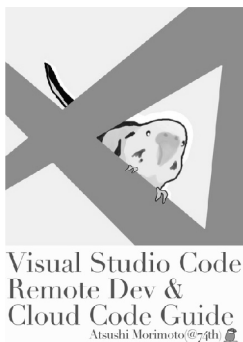
*5 <https://www.amazon.co.jp/dp/B07KXGNVS2/>

*6 <https://tatsu-zine.com/books/vscode-debug-tech>

*7 <https://booth.pm/ja/items/1314180>

*8 <https://github.com/74th/vscode-debug-specs>

同人誌: Visual Studio Code Remote Dev & Cloud Code Guide



▲Visual Studio Code Remote Dev & Cloud Code Guide

リモート開発機能と Cloud Code について書いた同人誌『Visual Studio Code Remote Dev & Cloud Code Guide』*9 を頒布しています。なぜリモート開発機能が開発環境を一変させるほど優れているといえるのか、その魅力に迫ります。また、リモートコンテナ機能の実践例、Cloud Code の使い方についても記載しています。

*9 <https://booth.pm/ja/items/1575583>

筆者について



森下 篤 (Atsushi Morimoto)

twitter,github : @74th

Kubernetes、Python、Go で、オートモーティブサービスのサーバアプリを開発に従事。いつも鳥のイラストの T シャツを着ている。

<https://74th.tech>

著作

- 『Visual Studio Code 実践ガイド』 技術評論社 (2020)
- 『Visual Studio Code デバッグ技術』 インプレス R&D (2018)

同人誌

- 『Kubernetes わいわい会の本』 技術書典 7 (2019)
- 『VSCode Remote Dev & Cloud Code Guide』 技術書典 7 (2019)
- 『Customizing Python Shell xonsh』 技術書典 6 (2019)
- 『Shell Script の代わりに Python タスクランナー Fabric&Invoke を活用する技術』 技術書典 5 (2018)
- 『構造化と性能の間を Golang で攻める技術+WebWorker 活用技術』 こばたく共著 技術書典 4 (2018)
- 『Visual Studio Code デバッグ技術』 技術書典 3 (2017)

Booth でも同人誌を販売しています。 <https://74th.booth.pm/>

Visual Studio Code Ninja Guide

2020年3月1日 初版発行 (@技術書典8)

筆者 Atsushi Morimoto (@74th)

イラスト もりもり
