AsciidoctorとGradleでつくる文書執筆環境

https://github.com/h1romas4/asciidoctor-gradle-template

2023-06-28

はじめに

本文書は `Asciidoc` とその Ruby による実装である `Asciidoctor` を用いて Asciidoc 文書を執筆する環境を構築する手順を示します。実行環境は Windows、Linux、macOS の各 OS に対応しています。

この文書の手順より以下のことができるようになります。

Asciidoc 形式で執筆した文書を HTML/PDF 形式に変換

Asciidoc 文書変換用 Gradle スクリプト

テキストエディターで変換結果をリアルタイムにプレビューしながら文書を編集

Visual Studio Code と AsciiDoc 拡張設定

Asciidoc は表現力の高い文書をテキストファイルベースで執筆できるテキストプロセッサーです。他の軽量テキストプロセッサーが持たない文書間のインクルードやソースコードの挿入などの機能も有し、かつ簡潔です。特に技術文書の執筆には大きな力を発揮することでしょう。

Asciidoc の表現力を示すひとつの例は、このような脚注表現です。

一般的にこのようなテキストプロセッサーを用いた執筆環境を構築するためには多くの準備が必要となりますが、本文書の手順は極力初期導入するプロダクトを少なく、簡単に快適な執筆環境を整えられるよう考えられました。

具体的には文書の変換に、実行を JVM 環境だけに依存する `AsciidoctorJ` と `Gradle` を活用し、執筆環境については `Visual Studio Code` を用いることでリアルタイムに文書をプレビューしながら、最後にコマンド一つで HTML/PDF 化できるように準備してあります。PDF 出力に関しては、フォントセットや禁則処理設定をプロジェクトにもたせることにより、実行環境に関わらず同一出力が得られるようになっています。

Asciidoc による文書の執筆は形式的で効率が良く、また更新差分が取りやすいため文書履歴管理や共同執筆環境としても有効です。このメリットをさらに活用するため、本手順ではテキストダイアグラム記法による図形や数式のベクター画像挿入もサポートしました。

本文書がみなさんの執筆活動のお手伝いになれば幸いです。

謝辞

本文書の手順の実装であるビルドスクリプトやテーマでは次のプロダクトと技術資料が使われています。

プロダクト名の隣にライセンスを併記しました。商用利用等で制限のあるプロダクトはありませんが、それぞれライセンスを確認してください。

Font

- 源真ゴシック - SIL Open Font License 1.1 - http://jikasei.me/font/genshin/
- 源様明朝 - SIL Open Font License 1.1 - https://github.com/ButTaiwan/genyo-font
- Ricty Diminished - SIL Open Font License 1.1 - https://github.com/edihbrandon/RictyDiminished
- Morisawa BIZ UDGothic - SIL Open Font License 1.1 - https://github.com/googlefonts/morisawa-biz-ud-gothic
- Morisawa BIZ UDMincho - SIL Open Font License 1.1 - https://github.com/googlefonts/morisawa-biz-ud-mincho
- Open Iconic - MIT License - https://github.com/iconic/open-iconic/

Asciidoc

- Asciidoctor - MIT License - https://asciidoctor.org/
- Asciidoctorj - Apache License 2.0 - https://github.com/asciidoctor/asciidoctorj
- Asciidoctor.js - MIT License - https://asciidoctor.org/docs/asciidoctor.js/
- Asciidoctor PDF - MIT License - https://asciidoctor.org/docs/asciidoctor-pdf/
- Asciidoctor Gradle Plugin Suite - Apache License 2.0 - https://github.com/asciidoctor/asciidoctor-gradle-plugin
- asciidoctor-pdf-linewrap-ja - MIT License - https://github.com/fuka/asciidoctor-pdf-linewrap-ja
- asciidoctor-nabetani - MIT License - https://github.com/nabetani/asciidoctor-nabetani

Diagram

- PlantUML - Multi(MIT/Apache/BSD/EPL/GPLv3/LGPL) - https://github.com/plantuml/plantuml
- ditaa - LGPL-3.0 license - https://github.com/stathissideris/ditaa

Build Tool

- SDKMAN - Apache License 2.0 - https://sdkman.io/
- Gradle - Apache License 2.0 - https://gradle.org/

Text Editor

- Visual Studio Code - Microsoft - https://code.visualstudio.com/
- asciidoctor-vscode - MIT License - https://github.com/asciidoctor/asciidoctor-vscode
- vscode-paste-image - MIT License - https://github.com/mushanshitiancai/vscode-paste-image
- vscode-plantuml - MIT License - https://github.com/qjebbs/vscode-plantuml

Guide

- asciidoctor-pdfでかっこいいPDFを作る - https://qiita.com/kuboaki/items/67774c5ebd41467b83e2

素晴らしい成果を公開されているみなさまに感謝します。

目次

1. Asciidoc 文書変換用スクリプトを使う準備

本手順で用いる Asciidoc 文書変換用スクリプトはビルドツールである Gradle を活用しており、実行するためには Java 実行環境が必要です。

Java 実行環境は、文書変換スクリプトを動作させる過程で唯一 OS 環境に手動で導入する必要があるプロダクトです。それ以外のものは Gradle によりプロジェクトとして独立した形で自動的に導入されます。

お使いのコンピューターのコマンドライン環境（macOS/Linux ではターミナル、Windows では cmd.exe か powershell.exe）で `java -version` コマンドを入力し、Java 8 以上のバージョンが表示されるようであれば既に準備は整っています。

macOS/Linux の場合

```
$ java -version
openjdk version "1.8.0_192"
OpenJDK Runtime Environment (Zulu 8.33.0.1-macosx) (build 1.8.0_192-b01)
OpenJDK 64-Bit Server VM (Zulu 8.33.0.1-macosx) (build 25.192-b01, mixed mode)
```

Windows の場合

```
C:¥> java -version
openjdk version "1.8.0_192"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_192-b12)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.192-b12, mixed mode)
```

本文書では Java 11 を用いて解説します。

1.1. Java 実行環境の導入（macOS / Linux の場合）

もし macOS / Linux 環境に Java 実行環境がなければ `SDKMAN` を利用することで、ターミナルから簡単に導入することができます。

> https://sdkman.io/
>
> SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix based systems.
>
> — SDKMAN

手順. SDKMAN を用いた Java の導入

```
$ curl -s "https://get.sdkman.io" | bash ❶
$ source "$HOME/.sdkman/bin/sdkman-init.sh" ❷
$ sdk list java ❸
================================================================================
Available Java Versions for Linux 64bit
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 Temurin       |     | 19           | tem     |            | 19-tem
               |     | 17.0.4       | tem     | local only | 17.0.4-tem
               |     | 17.0.4.1     | tem     |            | 17.0.4.1-tem
               |     | 11.0.16.1    | tem     |            | 11.0.16.1-tem
               |     | 8.0.345      | tem     |            | 8.0.345-tem
               |     | 8.0.342      | tem     |            | 8.0.342-tem
               |     | 8.0.332      | tem     |            | 8.0.332-tem
$ sdk install java 11.0.16.1-tem ❹
```

❶ SDKMAN を導入します。

❷ SDKMAN を環境に設定します。

❸ 導入できる Java のバージョンを一覧します。

❹ Java 11 系の最新バージョンを指定して Java を導入します。

また、Gradle は `JAVA_HOME` 環境変数に実行環境の Java のパスが設定されていることを期待していますので、`.bashrc` や `.zshrc` で次のように `JAVA_HOME` を設定します。

手順. JAVA_HOME の設定

```
$ vi ~/.bashrc ❶
export JAVA_HOME=~/.sdkman/candidates/java/current ❷
$ source ~/.bashrc ❸
```

❶ vi エディタで `.bashrc` を開きます。

❷ 本ラインをファイルの最下部に追加し vi を保存終了します。

❸ 設定を適用します。

これで準備完了です。

SDKMAN について

SDKMAN は主に Java エコシステムの開発環境をコマンドラインから簡単に導入・設定するためにつくたれた管理ソフトウェアです。

たとえば簡単に各種 Java のバージョンを導入し切り替えることができます。

手順. SDKMAN による Java のバージョン切り替え

```
$ sdk install java 8.0.345-tem ❶
$ sdk default java 8.0.345-tem ❷
$ sdk default java 11.0.16.1-tem ❸
```

❶ Java 8 を導入

❷ Java 8 をデフォルトに設定

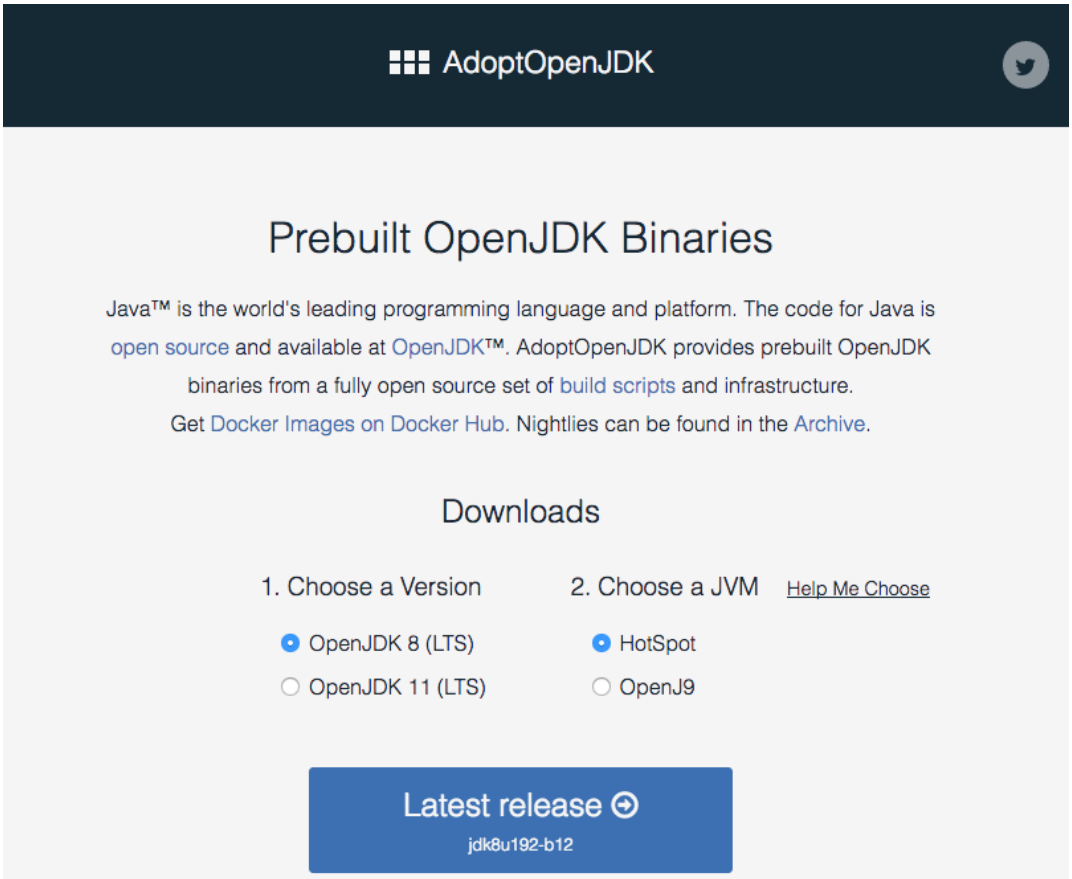
❸ Java 11 をデフォルトに戻す

1.2. Java 実行環境の導入（Windows の場合）

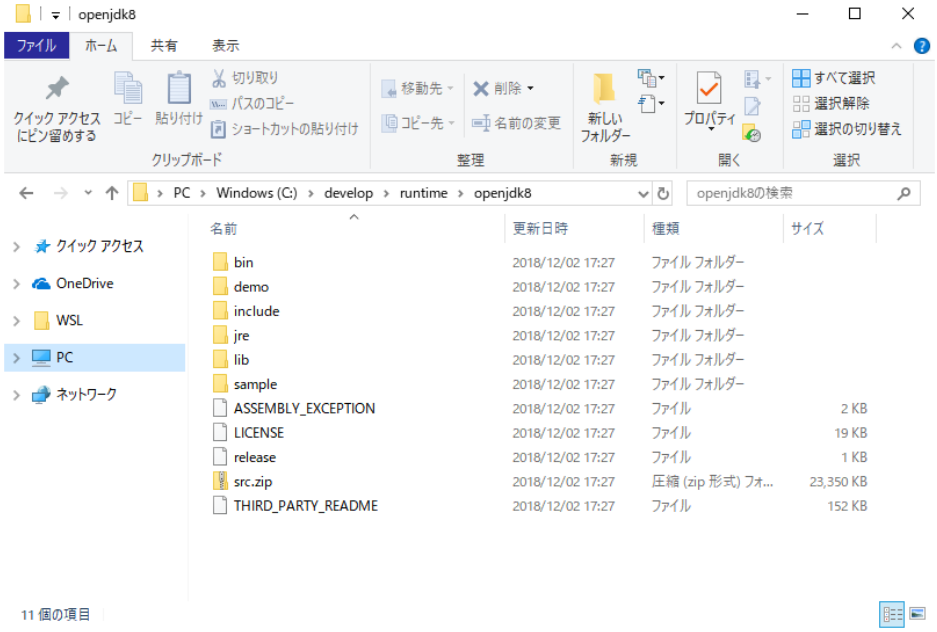
もし Windows 環境に Java 実行環境がなければ AdoptOpenJDK プロジェクトが提供する OpenJDK のバイナリを導入すると良いでしょう。

> https://adoptopenjdk.net
>
> Java™ is the world's leading programming language and platform. The code for Java is open source and available at OpenJDK™. AdoptOpenJDK provides prebuilt OpenJDK binaries from a fully open source set of build scripts and infrastructure.
>
> — AdoptOpenJDK

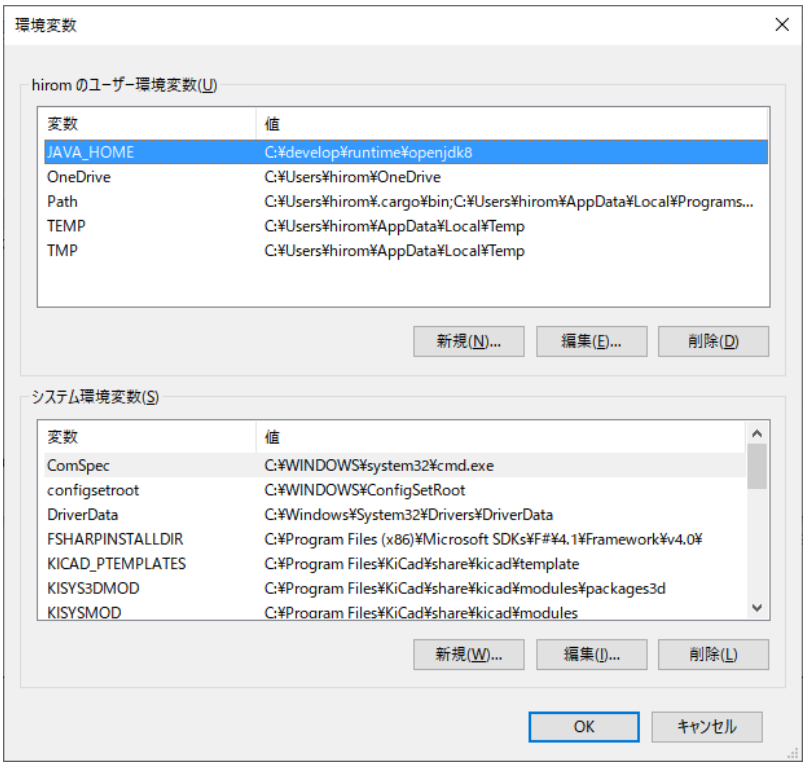
`https://adoptopenjdk.net` サイトにブラウザでアクセスし、OpenJDK 11 (LTS) - HotSpot を選択した後、zip ファイルをダウンロードしてください。



zip ファイルを任意の場所に展開します。ここでは `C:\develop\runtime\openjdk11` に展開したとします。



Gradle は `JAVA_HOME` 環境変数に実行環境の Java のパスが設定されていることを期待していますので、**エクスプローラー › PC（右クリック） › プロパティー › 詳細設定 › 環境設定** からユーザー環境変数に `JAVA_HOME` を追加し、先ほど .zip を展開したパス（`C:\develop\runtime\openjdk11` ）を設定します。



Gradle は JAVA_HOME 環境変数を元に Java の実行環境を探すため、`java` コマンドを使うための PATH 環境変数は設定しなくてもかまいません。

これで準備完了です。

2. Asciidoc から HTML/PDF 文書を作成する

2.1. サンプル文書の変換を試す

環境の準備ができましたので Asciidoc 文書を HTML/PDF に変換してみます。

変換に使うスクリプトは GitHub のリポジトリに公開されており、リポジトリには HTML/PDF 変換に使うファイル一式と、文書サンプルとして "この文書" の Asciidoc ファイルが置かれています。まずはサンプル文書が正しく変換できるかを試してみましょう。

macOS / Linux の場合は次のようにします。

手順. PDF 変換ビルドスクリプトの取得と実行

```
$ curl -L -O https://github.com/h1romas4/asciidoctor-gradle-template/archive/master.zip ❶
$ unzip master.zip ❷
$ cd asciidoctor-gradle-template-master ❸
$ ./gradlew docs ❹
BUILD SUCCESSFUL in 19s ❺
2 actionable tasks: 2 executed
```

❶ リポジトリのファイルをダウンロードします。

❷ ダウンロードした .zip ファイルを展開します。

❸ カレントディレクトリを展開したフォルダの中に移します。フォルダ名は任意のものに変更可能です。本手順ではプロジェクトフォルダと呼称します。

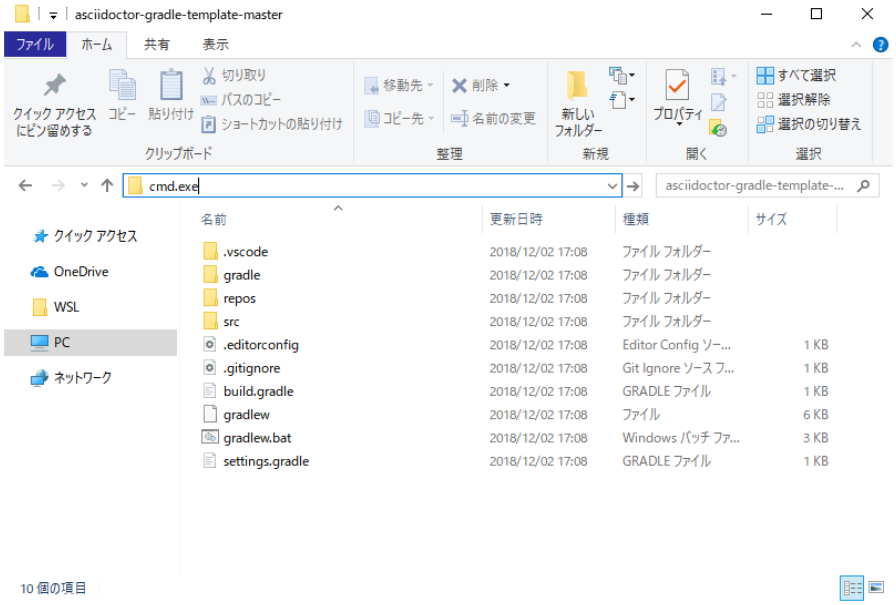
❹ Gradle のビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回以降は30秒程度で完了します。

❺ `BUILD SUCCESSFUL` が出力されればビルド成功です。

Windows をお使いの場合は同等の操作をブラウザとエクスプローラーを使って行います。

Windows の場合 .zip ファイルの展開先はマルチバイト文字を含まないパスにしてください。JRuby の制約により変換処理がエラーとなります。また、プロジェクト内部で使うフォルダやファイル名のマルチバイト名も同様です。

1. ブラウザを使って `https://github.com/h1romas4/asciidoctor-gradle-template/archive/master.zip` にアクセスしリポジトリのファイルを取得します。
2. ダウンロードした .zip ファイルを右クリックし展開します。フォルダ名は任意のものに変更可能です。本手順ではプロジェクトフォルダと呼称します。
3. 展開したフォルダ内をエクスプローラーで表示した上で、アドレスバーに `cmd.exe` と入力し、このフォルダをカレントディレクトリとしてコマンドプロンプトを起動します。



4. `.\gradlew.bat docs` と入力し Gradle ビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回以降は30秒程度で完了します。
5. `BUILD SUCCESSFUL` が出力されればビルド成功です。

プロキシーサーバーの設定

もしお使いのコンピューターがプロキシーサーバー経由のインターネットアクセスを行う場合は、次のコマンドを `./gradlew docs` をする前に入力してください。インターネットを使ったライブラリの取得が正しく行われるようになります。ホスト名（`example.com`） と ポート番号（`8080`）部分はそれぞれの環境に合わせてください。

手順. プロキシーサーバー設定（Windows）

```
set JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

手順. プロキシーサーバー設定（macOS / Linux）

```
export JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

2.2. 変換後の文書

`./gradlew docs` のビルド操作により Asciidoc から変換された文書は、`docs` フォルダに HTML版(`index.html`) と PDF版（`index.pdf`）として格納されます。

PDF 文書

HTML 文書

成果物の出力先となる `docs` フォルダの構成は次のとおりです。

```
docs
  Chapter00                    章ごとの HTML 版用のリソース
  Chapter01
    images                     HTML 版用の画像格納フォルダ
      AdoptOpenJDK.png         HTML 版用の PNG 画像リソース
      windows-01.png
      windows-02.png
  Chapter02
    images
      html.png
      pdf.png
      ..snip..
  Chapter03
    images
      diag-salt-sample1.svg    HTML 版向けに自動生成されたベクター画像
      diag-salt-sample2.svg    後述のダイアグラム記法による
      ..snip..
  index.html                   HTML 版（CSSを内包・画像はファイルに依存）
  index.pdf                    PDF 版（全ての画像・フォントリソースを内包）
```

成果物文書のソースとなるのは `src/docs/asciidoc` に配置された Asciidoc 文書と画像ファイル群です。

```
src
  docs
    asciidoc
      Chapter00
        index.adoc             章ごとの Asciidoc 文書
      Chapter01
        index.adoc
        images
          AdoptOpenJDK.png     文書で使う画像ファイル
          windows-01.png
          windows-02.png
      ..snip..
```

`docs` 出力フォルダの構成としては、章（`Chapter*/images`）フォルダには HTML 版からリンクされるリソースが `src/docs/asciidoc` からコピーされ配置されます。また、後述するダイアグラム記法を使った図表や数式も、ビルドのタイミングでベクター画像が生成され同フォルダに出力されます。

PDF 版のファイルについては `index.pdf` 単体で完結しており、フォントや画像リソースなど全てが PDF ファイル内に格納されます。

以上から成果物の配布は次のようになります。

HTML 文書	`docs`フォルダの `index.pdf`を除く全てのファイルを配布。
PDF 文書	`docs/index.pdf` のみを配布。

なおいずれの場合も `index.*` のファイル名は任意の名称に変更可能です。

`docs` フォルダは成果物の出力専用フォルダとなっており、ビルド時にいったん全てのファイルが削除されますので、**自らが作成したファイルは配置しない**ように注意してください。執筆で作成するファイルは必ず `src/docs/asciidoc` 配下に配置します。

2.3. テキストエディタで Asciidoc 文書を編集する

2.3.1. VS Code によるリアルタイムプレビュー

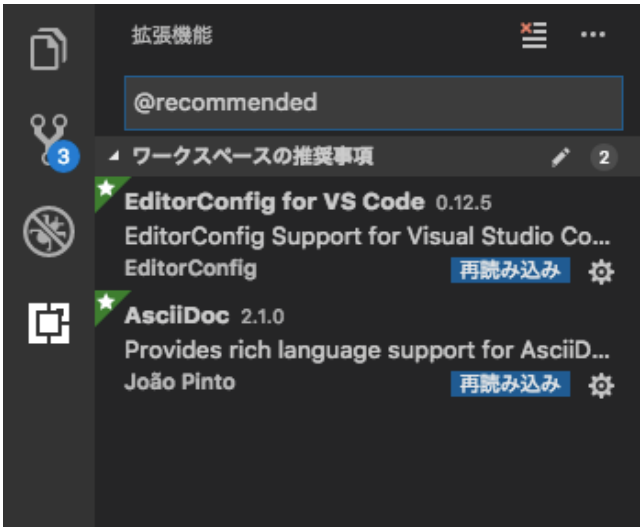
プロジェクトフォルダに配置された Asciidoc 文書は、Visual Studio Code を利用してリアルタイムに変換結果をプレビューしながら編集できるように準備されています。

プロジェクトフォルダを Visual Studio Code で開き、拡張機能の推奨事項から拡張を導入します。

拡張機能の推奨は `.vscode/extention.json` で設定されています。文書に応じて設定し、執筆メンバーの環境を揃えることができます。

1. **[すべてインストール]** ボタンをクリックします。

2. **[再読み込み]** ボタンをクリックします。



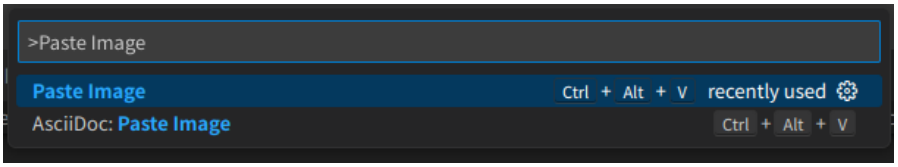
3. Asciidoc 文書（`src/docs/asciidoc/index.adoc`など）を Visual Studio Code のエクスプローラーから選択して開き、文書を開いたエディタ部右上に配置された **[Open Preview to the Side]**アイコンをクリックすると、画面右側に Asciidoc 文書の変換がリアルタイムに確認できるプレビューが表示されます。

2.3.2. クリップボードからの画像挿入

本手順で VS Code の推奨拡張として導入される Paste Image（`mushan.vscode-paste-image`） は、クリップボード上にある画像をファイルとして指定の位置に格納した上で、Asciidoc 文書にリンクを挿入する便利な拡張です。

特にコンピュータの操作手順文書をつくる場合に活用すると強力です。次のような簡単な操作でクリップボードの画像を Asciidoc 文書に挿入できます。

1. スクリーンキャプチャなどの機能でクリップボードに画像を保持する。
2. VS Code で Asciidoc 文書を開き、**VS Code › F1 キー押下 › Paste Image と入力 › エンターキーで選択**する。



Asciidoc 文書には画像リンクが挿入されるとともに、開いている Asciidoc 文書から相対で `images/` として見えるパスに画像ファイルが格納され、文書への画像挿入がひとつの操作で完了します。

```
image::2023-06-30-12-11-56.png[]
```

画像ファイル名の命名規約や出力先のフォルダ設定は `.vscode/setting.json` で行うことができます。

`vscode/setting.json`

```
{
    "pasteImage.path": "${currentFileDir}/images",
    "pasteImage.basePath": "${currentFileDir}/images",
    "pasteImage.defaultName": "Y-MM-DD-HH-mm-ss",
}
```

Asciidoc 拡張にも同様の機能（**VS Code › F1 キー押下 › AsciiDoc: Paste Image**）がありますが、現在のところ Paste Image 拡張のほうがファイル名などの設定が柔軟であるため本手順では Paste Image を紹介しています。

AsciiDoc: Paste Image は現在の開いている Asciidoc の `:imagesdir:` 属性を見て設定無しでファイル配置場所を決定してくれるなど優れた機能もありますので、それぞれの機能を確認し必要に応じて使い分けると良いでしょう。

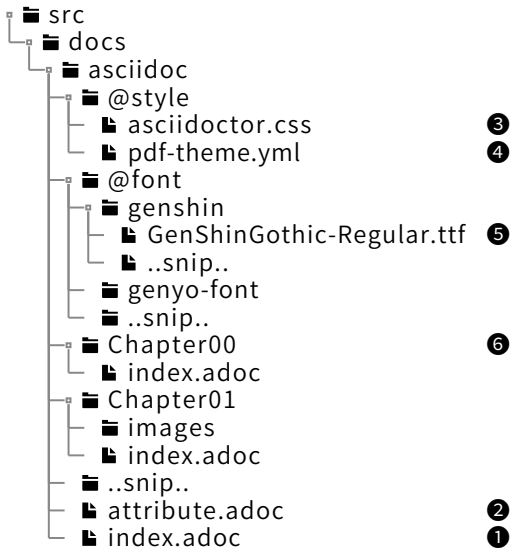
クリップボード連携のシステムインターフェース

Paste Image 拡張のクリップボード連携のシステムインターフェースは次のシェルスクリプトで実装されています。動作しない場合などに参考にしてください。

```
$ ls -laF ~/.vscode/extensions/mushan.vscode-paste-image-1.0.4/res
..snip..
-rw-r--r-- 1 hiromasa hiromasa   492  6月 26  2021 linux.sh
-rw-r--r-- 1 hiromasa hiromasa  1540  6月 26  2021 mac.applescript
-rw-r--r-- 1 hiromasa hiromasa   680  6月 26  2021 pc.ps1
..snip..
```

2.4. 文書のファイル構成

以下に文書のフォルダとファイル構成を示します。VS Code でプレビュー確認を行いながら、サンプル文書から執筆したい文書に合わせカスタマイズしていきます。




```
src/docs/asciidoc/index.adoc ❶
src/docs/asciidoc/attribute.adoc ❷
src/docs/asciidoc/@style/asciidoctor.css ❸
src/docs/asciidoc/@style/pdf-theme.yml ❹
src/docs/asciidoc/@font/**/*.ttf ❺
src/docs/asciidoc/Chapter{number}/index.adoc ❻
```

❶ 文書を作成する起点となる Asciidoc 文書です。表紙となる文書のタイトルなどを記載し、その後から章ごとの Asciidoc 文書を `include` して構成していきます。

❷ 文書設定をするためのファイルです。各文書から `incliude` します。

❸ HTML 出力とプレビュー用のスタイルシートです。文書に合わせて修正することができます。

❹ PDF 文書に変換する際に使われるスタイル定義です。文書に合わせて修正することができます。Asciidoctor PDF Theming Guide からドキュメントが参照できます。

❺ PDF 文書に埋め込みされるフォントファイルを配置します。`pdf-theme.yml` からファイル名で参照されています。TrueType フォント `.ttf` が指定できます。

❻ `src/docs/asciidoc/index.adoc` から参照される子文書です。後述の `build.gradle` による画像パス解決のためフォルダ名は `Chapter{number}` とします。

`src/docs/asciidoc/attribute.adoc` では文書のデフォルトキャプション名などの属性定義が行えます。必要に応じて修正してください。

src/docs/asciidoc/attribute.adoc

```
..snip..
:preface-title: まえがき
:toc-title: 目次
:appendix-caption: 付録
:caution-caption: 注意
:example-caption: 例
:figure-caption: 図
..snip..
```

Asciidoctor で利用可能な属性は次から参照できます。

> https://asciidoctor.org/docs/user-manual/#attributes
>
> Attributes are one of the features that sets Asciidoctor apart from other lightweight markup languages. Attributes can activate features (behaviors, styles, integrations, etc) or hold replacement (i.e., variable) content.
>
> — asciidoctor.org

2.5. 執筆の開始

一通りのサンプル文書の構成確認が終りましたら、次のような流れでご自身の執筆を開始してみてください。

1. サンプル文書の `src/docs/asciidoc/Cahpter02` 以降のフォルダの削除を行い見通しを良くする。
2. `src/docs/asciidoc/index.adoc` からの 1. で削除した `Chapter*` の`include` を削除する。
3. `src/docs/asciidoc/index.adoc` で文書のタイトルや版数などを設定する。
4. `src/docs/asciidoc/Cahpter00/index.adoc` を空にして目次前に挿入される「はじめに」や「謝辞」「本書の読み方」等の執筆。
5. `src/docs/asciidoc/Cahpter01/index.adoc` を空にして本編の執筆を開始。
6. 事前に目次案がある場合は、このタイミングで `Chapter*` フォルダと対応する `index.adoc` を新規作成し、目次案にあった章や節などの大項目のみを準備するのも良い方法です。目次が完成し全体の把握がしやすくなります。
7. ここで一度 `./gradlew docs` を行い、想定通りの HTML/PDF 文書が `docs/` 内に生成されていることを確認する。外観の修正点があれば、
    - 文書形式の場合は `src/docs/asciidoc/attribute.adoc` を調整。
    - PDF テーマの場合は `src/docs/asciidoc/@style/pdf-theme.yml` を調整。
    - HTML テーマの場合は `src/docs/asciidoc/@style/asciidoctor.css` を調整。

2.6. 執筆のイテレーション

ここまでの手順で執筆環境が整った後の、文書執筆の進め方のイメージは次のようになります。

1. プロジェクトフォルダを VS Code で開く。
2. `src/docs/asciidocs` 配下の Asciidoc 文書を VS Code でプレビューしながら執筆する。
    - Chapter が増えた場合は `src/docs/asciidocs/index.adoc` に Chapter を追記し、`src/docs/asciidocs/Chapter*/index.adoc` ファイルを新規追加する。
    - 画像ファイルが必要な場合は、`src/docs/asciidocs/Chapter*/images` フォルダに追加する。
3. 文書執筆の一定の区切りをもって `./gradlew docs` で文書のビルドを行い、HTML/PDF 文書を `docs/index.html` と `docs/index.pdf` で推敲する。
    - `./gradlew docs` の入力は VS Code の統合ターミナルが使えます。
4. 必要ならばプロジェクトディレクトリを Git などのバージョン管理下として、コミットを行う。
5. 本日の執筆を終え、明日はまた 1. に戻る。

推敲は、生成された PDF 文書を iPad などペンシルデバイスを持つ機械に転送し「赤入れ」すると便利かもしれません。

VS Code の Zen Mode

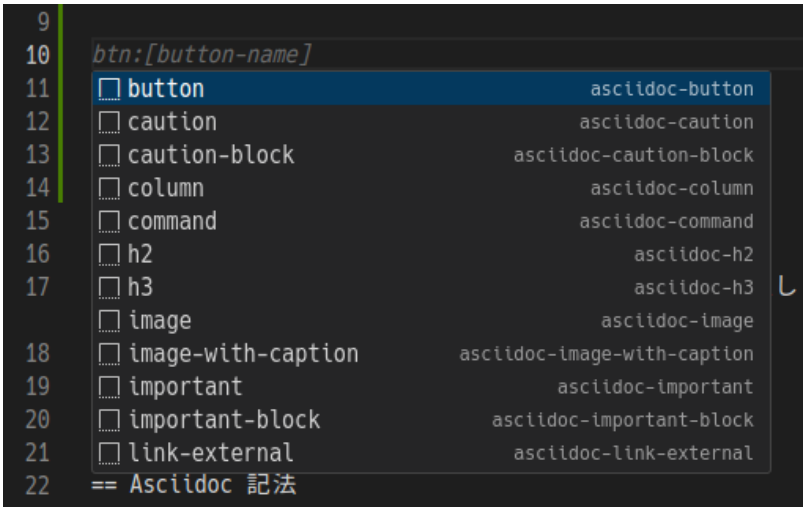
VS Code には執筆に集中したい場合に、不要な UI を隠しながらフルスクリーン表示をする Zen Mode があります。標準のキーアサインでは **[Ctrl + k, z]** でモードが遷移し、再度同じ操作で元の画面に戻ります。

Zen Mode は、VS Code や OS の不要なコンポーネントが見えなくなり画面が広く使えること、また思考を遮る要素がなくなることから快適に執筆の助けになります。

3. Asciidoc 記法

ここでは Asciidoc 文書で利用可能な記法の中で、特に技術文書で使いそうなものを取り上げて解説します。

なお、プロジェクトフォルダの `.vscode/asciidoc.code-snippets` 設定には VS Code の **[Ctrl + Spaec]** 操作で補完可能な形式で Asciidoc 記法のスニペットが格納されています。プロジェクトで定形の記法があれが追加・修正すると便利でしょう。



3.1. 文書中の相互参照

文書内の参照リンクは `.adoc` 文書中で次のように指定します。HTML/PDF でクリックリンクが生成されます。

```
[[project-structure]] ❶
== Asciidoc 記法

[[introduction]] ❷
== はじめに

<<project-structure,章の冒頭>> ❶
<<Chapter01/index.adoc#introduction,はじめに>> ❷
```

❶ 同一 `.adoc` 内での内部参照

❷ `.adoc` をまたいだドキュメント間参照

リンクの例

- 章の冒頭 を参照してください･･･
- はじめに で述べたように･･･

3.2. 外部リンク

インターネット上の URL は自動的にリンクに変換されます。

https://h1romas4.github.io/asciidoctor-gradle-template/index.html

次のように URL の末尾に [] を定義すると指定した文字列からのリンクとなります。

```
https://h1romas4.github.io/asciidoctor-gradle-template/index.html[AsciidoctorとGradleでつくる文書執筆環境]
```

AsciidoctorとGradleでつくる文書執筆環境

PDF の印刷を考慮する場合は、文字列リンクを用いると URL の情報が消えてしまうため、次項の引用機能を用いると良いでしょう。

3.3. 引用

外部の文献を参照する引用は次のように記述します。執筆している文書が紙面となり、引用先がウェブサイトの場合はいずれかの位置に URL を掲載します。

```
[quote, Asciidoctor Documentation Site]
____
https://docs.asciidoctor.org/

Welcome to the Asciidoctor documentation site! Here you can find the reference material,
guides, and examples to write content in AsciiDoc and publish it using Asciidoctor. This
documentation will help you start your journey with AsciiDoc or dive deeper if you're already
well on your way.
____
```

> https://docs.asciidoctor.org/
>
> Welcome to the Asciidoctor documentation site! Here you can find the reference material, guides, and examples to write content in AsciiDoc and publish it using Asciidoctor. This documentation will help you start your journey with AsciiDoc or dive deeper if you're already well on your way.
>
> — Asciidoctor Documentation Site

3.4. 画像

画像の挿入は、画像ファイルを編集中の `.adoc` 文書から相対パスでみた `images/` フォルダに格納した上で、次のように記述します。

```
image::2023-06-30-12-11-56.png[pdfwidth=70%, width=600px]
```

`pdfwidth` 属性では PDF 紙面に対する幅の比率を、`width` 属性では HTML 文書中での幅の比率もしくはピクセル値を指定します。

なお、本手順の Asciidoc 文書の構成では `images` に格納されている画像ファイルパスに `images/` を付与する必要はありません。

これは `src/docs/asciidoc/attribute.adoc` で `:imagesdir: ./images` が設定済みであるための動作です。

3.5. ソースコードシンタックスハイライト

`[source]` ブロックを用いて文書中にソースコードをシンタックスハイライト付きで埋め込みます。また、ソースコードに `// <1>` などのコメント形式でソースコード下部に対応する説明が記述できます。

```
[source, javascript]
[caption=""]
.JavaScript ソースコードのシンタックスハイライトの例
----
function hello() {
    console.log("Hello, World!"); // <1>
}
----

<1> コンソールに ``Hello, World`` を出力
```

JavaScript ソースコードのシンタックスハイライトの例

```
function hello() {
    console.log("Hello, World!"); ❶
}
```

❶ コンソールに `Hello, World` を出力

3.6. インラインコマンド・ファイル名

文中に現れるコマンドやファイル名などは `コマンド` で記述します。

```
``コマンド``
```

3.7. メニュー・ボタン

コンピュータの操作で使われる **メニュー** › **File** › **Open** や [OK] ボタンなどは次のように記述します。

```
menu:メニュー[File > Open]

btn:[OK] ボタン
```

3.8. ファイルインクルード

`.adoc` ファイルから別のファイルをインクルードできます。ソースコードや後述のダイアグラム形式などを別ファイルとして切り出すのに便利です。

Gradle のファイル更新監視は `.adoc` からインクルードされたファイルまでは及びません。インクルード先のファイルだけを書き換えた場合は `./gradlew docs` が処理なしで終了してしまいますので、この場合は `./gradlew clean` するなどしてからリビルドしてください。

```
[source, rust]
----
include::source/hello.rs[]
----
```

インクルードにより挿入されたソースコードの例

```rust
fn main() {
    println!("Hello, world!");
}
```

3.9. 表形式

表形式は次のようなコードで定義します。データ形式、罫線表現に関していくつかのオプションが設定できます。

属性の `cols="1,2"` ではカラムの比率を指定します。

表形式の例1（CSV 形式、ヘッダーなし、グリッドなし）

```
[format="csv",cols="1,2"]
[frame="topbot",grid="none"]
[caption=""]
.表形式の例1
|======
カラム1,カラム2
キー1,バリュー1
キー2,バリュー2
|======
```

キー1	バリュー1
キー2	バリュー2
キー3	バリュー3

表形式の例2（デフォルト形式、ヘッダーあり、グリッドあり）

```
[cols="1,2", options="header"]
[frame="topbot"]
[caption=""]
.表形式の例2
|======
|カラム名1
|カラム名2

|キー1
|バリュー1

|キー2
|バリュー2
|======
```

カラム名1	カラム名2
キー1	バリュー1
キー2	バリュー2

3.10. リスト

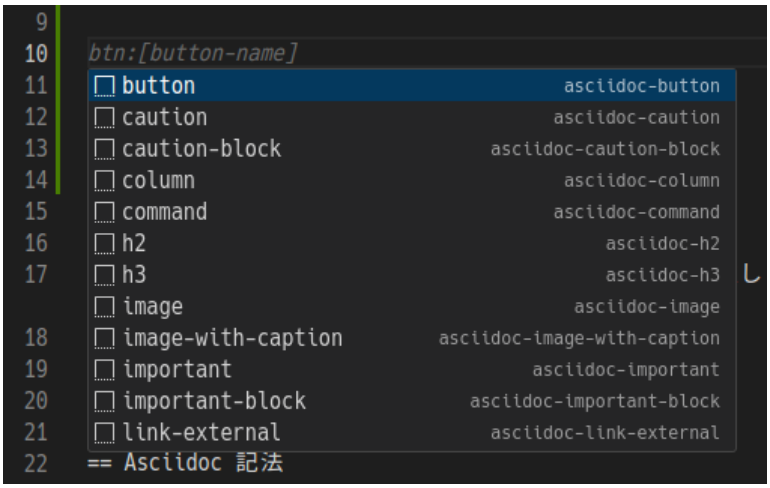
箇条書き表現は次のように記述します。

Asciidoc はリスト中にリスト継続マーカ（+）を使うことでブロックのネストが可能です。手順を順序ありリストで表現し、順序やインデントを維持しながら画像や表形式などの挿入をします。

```
. 手順1
. 手順2
+
リスト継続1
+
image::2023-06-30-15-39-26.png[pdfwidth=60%, width=60%]
. 手順3
.. 箇条書きのネスト1
... 箇条書きのネスト2
*** 順序なしリスト1
*** 順序なしリスト2
```

1. 手順1
2. 手順2

   リスト継続1



3. 手順3
   a. 箇条書きのネスト1
      i. 箇条書きのネスト2
         - 順序なしリスト1
         - 順序なしリスト2

3.11. 脚注

補足点を脚注するいくつかの表現があります。

```
NOTE: メモ

TIP: チップス

IMPORTANT: 重要

WARNING: 警告

CAUTION: 注意
```

メモ

チップス

重要

警告

注意

脚注が長文になる場合は次のようにブロックを用います。

```
[NOTE]
====
改行を含む脚注です。

吾輩は脚注である。名前はまだ無い。
====
```

改行を含む脚注です。

吾輩は脚注である。名前はまだ無い。

3.12. 改ページ・水平線

改ページは次の記述で挿入します。PDF 文書のみに効果があり HTML 文書には影響しません。

```
<<<<
```

水平線は HTML/PDF 文書のどちらでも有効です。

```
- - -
```

3.13. コラム表現

Asciidoc のサイドバーとブロック名称機能を組み合わせ、コラム表現を挿入します。

```
.コラム
****
ブロックを使ってコラム表現を挿入します。

吾輩はコラムである。名前はまだ無い。
****
```

コラム

ブロックを使ってコラム表現を挿入します。

吾輩はコラムである。名前はまだ無い。

3.14. コメント行

成果物に現れない Asciidoc 文書へのコメントは次のように文頭に `//` で記述します。

```
// TODO: さらに Asciidoc 記法を追加していくこと。
```

4. ダイアグラム記法

本変換スクリプトでは `asciidoctorj-diagram` が有効になっており、いくつかのダイアグラム記法を追加のアドイン無しで使うことができます。

ダイアグラムを SVG ベクター画像で出力する指定は次のようになります。

```
[ダイアグラム名，出力ファイル名，svg，pdfwidth=70%，width=480px]
```

`pdfwidth` は PDF 紙面に対しての比率指定、`width` は HTML 上の幅比率もしくはピクセル（`px`）で指定します。

本章では PlantUML と ditaa ダイアグラム記法を用いて、技術文書で使われやすい表現のいくつかの記述法を示します。詳しくはそれぞれのドキュメントを参考にしてください。

> https://plantuml.com/ja/
>
> PlantUML in a nutshell
>
> — PlantUML

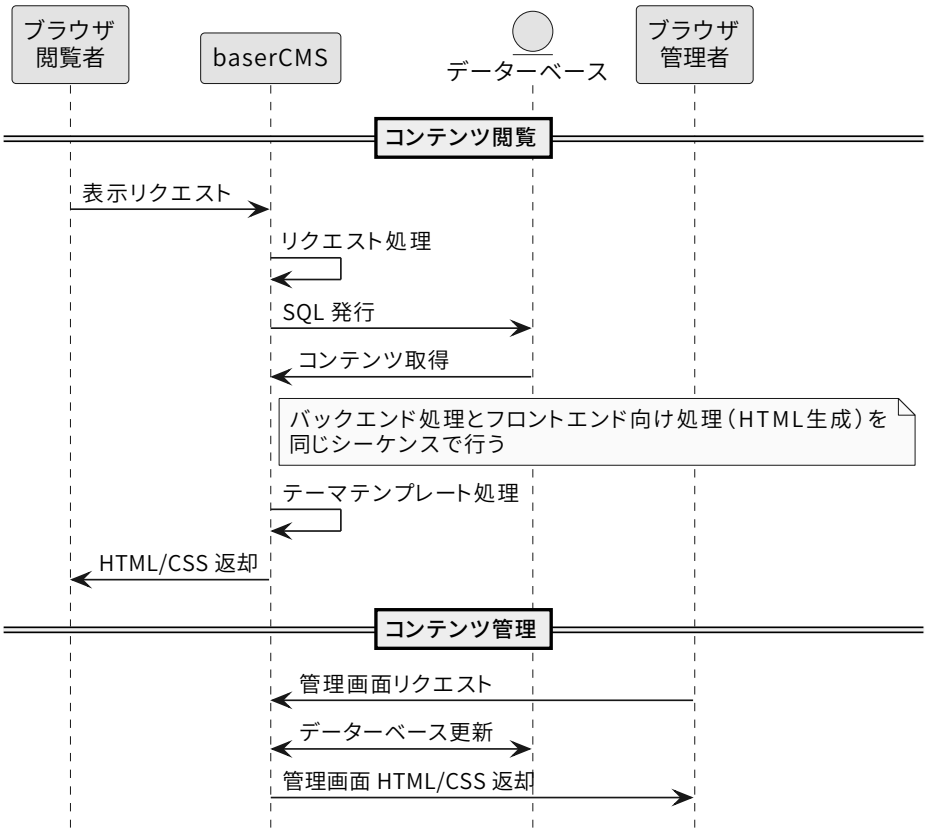
変換スクリプトは PlantUML ダイアグラムの PDF 出力時に日本語が化けないよう、自動的にフォントパッチが適用されるよう構成されています。

> https://ditaa.sourceforge.net/
>
> DIagrams Through Ascii Art by Stathis Sideris
>
> — ditaa

4.1. シーケンス図

シーケンス図を PlantUML ダイアグラム記法で記述します。

```
[plantuml, diag-sequence-sample2, svg, pdfwidth=70%, width=70%]
[caption=""]
----
@startuml diag-sequence-sample2
skinparam monochrome true
hide footbox
participant "ブラウザ\n閲覧者" as browser
participant "baserCMS" as basercms
entity データーベース as Entity
participant "ブラウザ\n管理者" as admin
== コンテンツ閲覧==
browser -> basercms: 表示リクエスト
basercms -> basercms: リクエスト処理
basercms -> Entity: SQL 発行
Entity -> basercms: コンテンツ取得
note right of basercms: バックエンド処理とフロントエンド向け処理（HTML生成）を\n同じシーケンスで行う
basercms -> basercms: テーマテンプレート処理
basercms -> browser: HTML/CSS 返却
== コンテンツ管理==
admin -> basercms: 管理画面リクエスト
basercms <-> Entity: データーベース更新
basercms -> admin: 管理画面 HTML/CSS 返却
@enduml
----
```



PlantUML ダイアグラムのリアルタイムプレビュー編集

本手順の VS Code の推奨プラグインとなっている PlantUML 拡張（`jebbs.plantuml`）を使うと、PlantUML のダイアグラム記法も VS Code 上でリアルタイムプレビューすることができます。

プレビュー処理をするために必要な PlantUML サーバは Docker コンテナで起動すると便利でしょう。なお、Gradle による文書の変換では内蔵の PlantUML ライブラリが使われますので本サーバーが起動している必要はありません。

```
docker run -d --rm -p 8080:8080 plantuml/plantuml-server:jetty
```

`.vscode/setting.json` は次のようにポート `8080` で PlantUML サーバが起動されることを期待して構成してあります。

`.vscode/setting.json`

```
{ "plantuml.server": "http://localhost:8080" }
```

文書で使う PlantUML ダイアグラムを `.puml` 拡張子で保存することで Asciidoc と同様にプレビューアイコンが表示され、プレビュー画面ではリアルタイムに `.puml` ファイルの修正が反映されます。

4.2. フォルダ・ファイルツリー

フォルダ・ファイルツリーを PlantUML の Salt 記法で記述します。

```
[plantuml, diag-salt-sample1, svg, pdfwidth=30%, width=280px]
[caption=""]
----
' https://github.com/iconic/open-iconic/
@startsalt diag-salt-sample1
{
{T
 + <&folder> wp-admin
 + <&folder> wp-content
 ++ <&folder> themes
 +++ <&folder> my-theme   | 新規作成
 ++++ <&file> style.css   | 新規作成
 ++++ <&file> index.php   | 新規作成
 ++ <&folder> uploads
 + <&folder> wp-includes
 + <&file> .htaccess      | 自動作成
 + <&file> index.php
}
}
@endsalt
----
```

4.3. 数式

数式を PlantUML の AsciiMath 記法で記述します。

```
[plantuml, math-sample1, svg, pdfwidth=70%, width=70%]
[caption=""]
----
@startmath math-sample1
f(t)=(a_0)/2 + sum_(n=1)^ooa_ncos((npit)/L)+sum_(n=1)^oo b_n\ sin((npit)/L)
@endmath
----
```

$$f\left(t\right)=\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos\left(\frac{n\pi t}{L}\right)+\sum_{n=1}^{\infty}b_n\ \sin\left(\frac{n\pi t}{L}\right)$$

PlantUML の Salt 記法のフォントサイズ

Salt は現在のところ残念ながらフォントサイズの指定をすることができません。このため任意のフォントサイズにするためには出力画像の `pdfwidth` と `width` を設定して算出する必要があります。

```
@startsalt
<style>
saltDiagram {
  FontSize 10 ❶
}
</style>
{
{T
 + <&folder> wp-admin
}
}
@endsalt
```

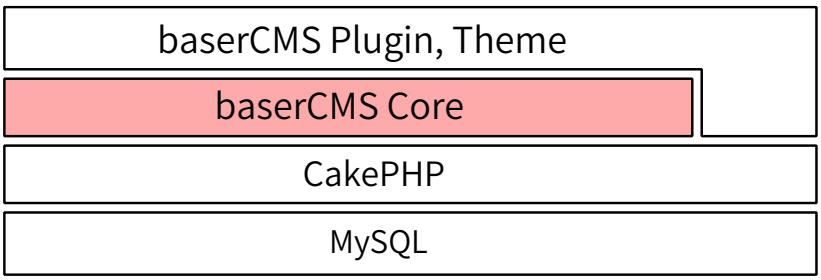
❶ 未実装で効果がない

4.4. ブロック図

ブロック図を ditaa 記法で記述します。

ditta 内で日本語を使う場合は、表示ずれを防ぐため半角スペースなどで文字数を調整する必要があります。また、ドロップシャドー `shadows` 設定は HTML 版に対してのみ有効で、現在のところ PDF 版に対しては効果がないようです。

```
[ditaa, diag-ditaa-block, svg, shadows=true, separation=true, pdfwidth=70%, width=70%]
[caption=""]
----
+---------------------------------+
|     baserCMS Plugin, Theme      |
+-----------------------------+   +
| cPNK   baserCMS Core        |   |
+-----------------------------+---+
|            CakePHP              |
+---------------------------------+
|             MySQL               |
+---------------------------------+

/-------------+-------------\
|cRED RED     |cBLU BLU     |
+-------------+-------------+
|cGRE GRE     |cPNK PNK     |
+-------------+-------------+
|cBLK BLK     |cYEL YEL     |
\-------------+-------------/
----
```

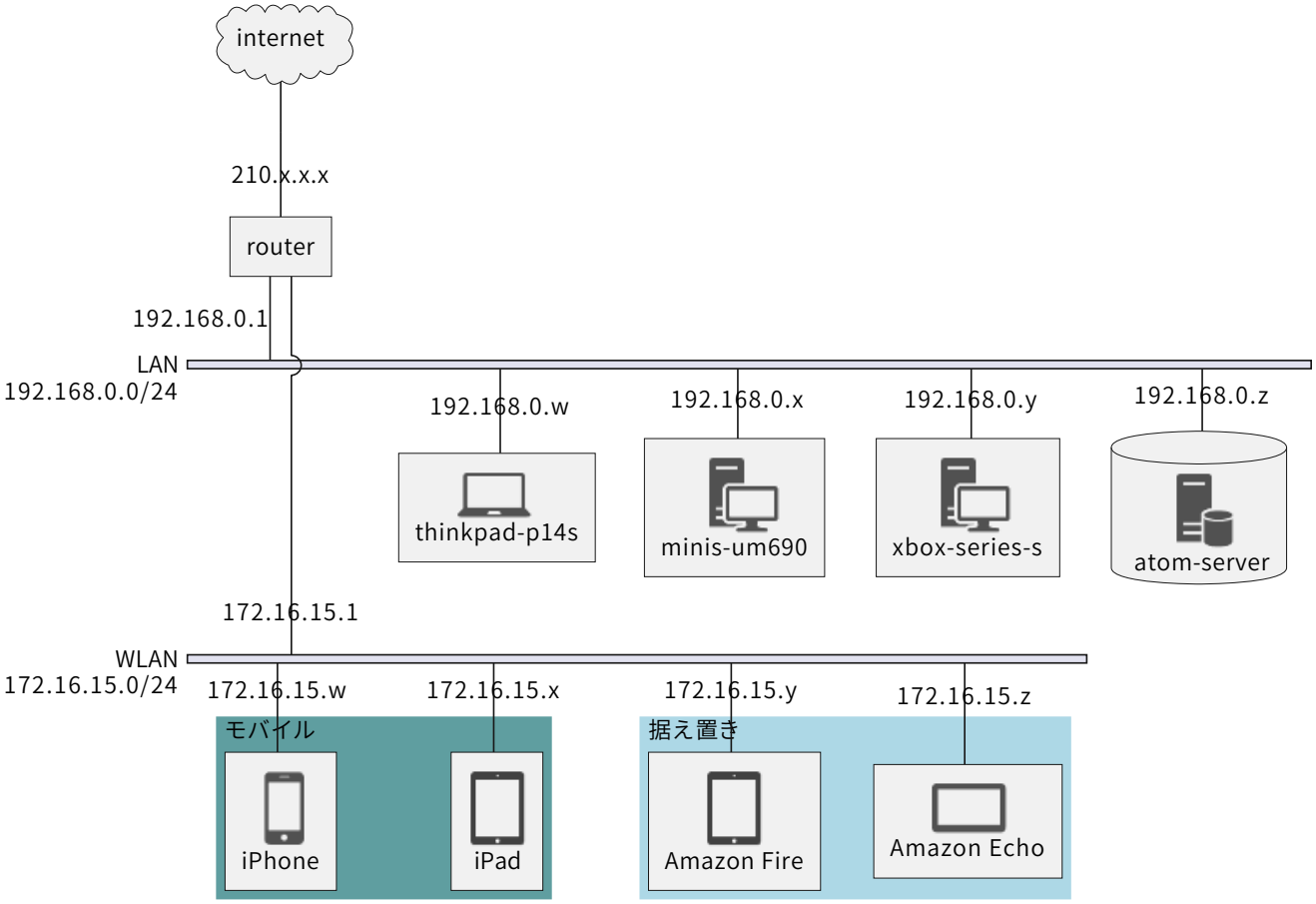


4.5. ネットワーク構成図

ネットワーク構成図を PlantUML の nwdiag で記述します。

`diag-nwdiag-network1.puml`（分かりやすいように実際のソースより簡略化）

```
[plantuml, diag-nwdiag-network1, svg]
----
@startuml diag-network1
<style>
nwdiagDiagram {
    FontSize 14
    group {
        BackGroundColor cadetblue
    }
}
</style>

nwdiag {
    internet [shape = cloud];
    internet -- router;
    router [address = "210.x.x.x"];
    network LAN {
        address = "192.168.0.0/24"
        router [address = "192.168.0.1"];
        thinkpad-p14s [address = "192.168.0.w"];
        minis-um690 [address = "192.168.0.x"];
        xbox-series-s [address = "192.168.0.y"];
        atom-server [address = "192.168.0.z", shape = database"]
    }
    network WLAN {
        address = "172.16.15.0/24"
        router [address = "172.16.15.1"];
        group {
            description = モバイル
            iphone01 [address = "172.16.15.w"];
            ipad01 [address = "172.16.15.x"];
        }
        group {
            color = "#add8e6"
            description = 据え置き
            fire01 [address = "172.16.15.y"];
            echo01 [address = "172.16.15.z"];
        }
    }
}
@enduml
----
```

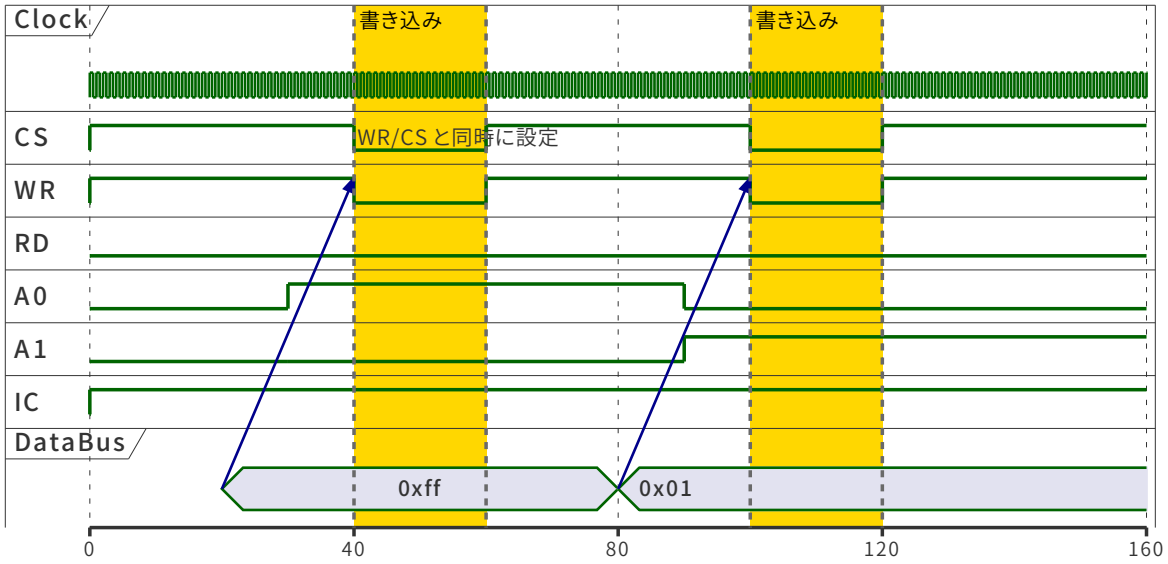

internet
210.x.x.x
router
192.168.0.1
LAN
192.168.0.0/24
192.168.0.w
192.168.0.x
192.168.0.y
192.168.0.z
thinkpad-p14s
minis-um690
xbox-series-s
atom-server
172.16.15.1
WLAN
172.16.15.0/24
172.16.15.w
172.16.15.x
172.16.15.y
172.16.15.z
モバイル
据え置き
iPhone
iPad
Amazon Fire
Amazon Echo

4.6. タイミングチャート

デジタル回路などのタイミングチャートを PlantUML で記述します。

`diag-timing-sample1.puml`（実際のソースより抜粋）

```
[plantuml, diag-timing-sample1, svg, pdfwidth=90%, width=80%]
----
@startuml diag-timing-sample1
scale 40 as 150 pixels
clock "Clock" as clk with period 1
binary "CS" as CS
binary "WR" as WR
binary "RD" as RD
binary "A0" as A0
binary "A1" as A1
binary "IC" as IC
concise "DataBus" as DB

@0 as :start
@20 as :set_data_bus_1
@30 as :set_addr_1
@40 as :write_start_1
@60 as :write_end_1

@:start
IC is high
CS is high
WR is high
RD is low
A0 is low
A1 is low

@:set_data_bus_1
DB is "0xff"
DB -> WR@+20

@:set_addr_1
A0 is high
A1 is low

@:write_start_1
CS is low : WR/CS と同時に設定
WR is low

@:write_end_1
CS is high
WR is high

highlight 40 to 60 #Gold;line:DimGrey : 書き込み
@enduml
----
```


Clock
書き込み
書き込み
CS
WR/CS と同時に設定
WR
RD
A0
A1
IC
DataBus
0xff
0x01
0
40
80
120
160

4.7. マインドマップ

PlantUML のマインドマップ記法を使い木構造を表現します。

```
[plantuml, diag-mindmap-sample1, svg]
[caption=""]
----
@startmindmap diag-mindmap-sample1
<style>
mindmapDiagram {
    FontSize 18
    .rose {
        BackgroundColor #ffbbcc
    }
}
</style>
* ウェブブラウザー
** HTML
** CSS
** JavaScript
** <&star> WebAssembly <<rose>>
@endjson
----
```