

AsciiDoctorと Gradleでつくる 文書執筆環境



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です。他の軽量テキストプロセッサ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ました。またさらに、クラウド統合環境である Gitpod もサポートし、ウェブブラウザだけでの執筆・変換も実現しています。

手間がかかることが多い PDF 出力に関しても、フォントセットや禁則処理設定をプロジェクトに持たせることにより、実行環境に関わらず同一出力が得られるように調整しました。

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をサポートしています。

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。

ライセンス

本文書に含まれる筆者が作成したビルドスクリプト、サンプル文書 `.adoc`、ダイアグラムのなどのソースは全て **MIT license** です。ライセンスに従って自由にご利用ください。

リポジトリのソースやライセンスファイルは次から参照できます。

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

AsciidoctorとGradleでつくる文書執筆環境

— <https://github.com/h1romas4>



本文書の PDF 版の表紙は @Fujix さん によって作成されました。

素敵なデザインをありがとうございました。

謝辞

本文書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でかつこいい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

はじめに	1
ライセンス	2
謝辞	2
1. AsciiDoc 文書変換用スクリプトを使う準備	5
1.1. Java 実行環境の導入 (macOS / Linux の場合)	6
1.2. Java 実行環境の導入 (Windows の場合)	8
2. AsciiDoc から HTML/PDF 文書を作成する	10
2.1. サンプル文書の変換を試す	10
2.2. 変換後の文書	13
2.3. テキストエディタで AsciiDoc 文書を編集する	15
2.4. 文書のファイル構成	18
2.5. 執筆の開始	20
2.6. 執筆のイテレーション	20
2.7. クラウドによる執筆	21
2.8. GitHub Actions によるビルド	22
3. AsciiDoc 記法	25
3.1. 文書中の相互参照	25
3.2. 外部リンク	26
3.3. 引用	26
3.4. 画像	27
3.5. ソースコードシンタックスハイライト	27
3.6. インラインコマンド・ファイル名	28
3.7. メニュー・ボタン	28
3.8. ファイルインクルード	28
3.9. 表形式	29
3.10. リスト	30
3.11. 脚注	31
3.12. 改ページ・水平線	32
3.13. コラム表現	32
3.14. コメント行	32
4. ダイアグラム記法	33
4.1. シーケンス図	34
4.2. フォルダ・ファイルツリー	36
4.3. 数式	37
4.4. ブロック図	38
4.5. メモリーマップ	40
4.6. ネットワーク構成図	41
4.7. システム構成図	43
4.8. タイミングチャート	45
4.9. マインドマップ	47

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 11 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 11.0.20.1 2023-08-24
OpenJDK Runtime Environment Temurin-11.0.20.1+1 (build 11.0.20.1+1)
OpenJDK 64-Bit Server VM Temurin-11.0.20.1+1 (build 11.0.20.1+1, mixed mode)
```

Windows の場合

```
C:¥> java -version
openjdk version "11.0.20.1" 2023-08-24
OpenJDK Runtime Environment Temurin-11.0.20.1+1 (build 11.0.20.1+1)
OpenJDK 64-Bit Server VM Temurin-11.0.20.1+1 (build 11.0.20.1+1, mixed mode)
```



本文書では Java 11 を用いて解説します。また、ビルド時に若干ワーニングが出力されますが Java 17 でも期待通り動作することを確認しています。

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		20.0.2	tem		20.0.2-tem
		20.0.1	tem		20.0.1-tem
		17.0.8	tem	installed	17.0.8-tem
		17.0.8.1	tem		17.0.8.1-tem
		17.0.7	tem	installed	17.0.7-tem
		17.0.4	tem	local only	17.0.4-tem
		11.0.20	tem		11.0.20-tem
	>>>	11.0.20.1	tem	installed	11.0.20.1-tem
		11.0.19	tem	installed	11.0.19-tem

```
$ sdk install java 11.0.20.1-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られた管理ソフトウェアです。

たとえば次のように簡単に各種 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 11.0.20.1-tem ❶  
$ sdk default java 11.0.20.1-tem ❷  
$ sdk default java 17.0.4.1-tem ❸
```

- ❶ Java 11 を導入
- ❷ Java 11 をデフォルトに設定
- ❸ Java 17 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 > 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/asciidoc-gradle-template/archive/master.zip ❶  
$ unzip master.zip ❷  
$ cd asciidoc-gradle-template-master ❸  
$ ./gradlew docs ❹  
BUILD SUCCESSFUL in 19s ❺  
2 actionable tasks: 2 executed
```

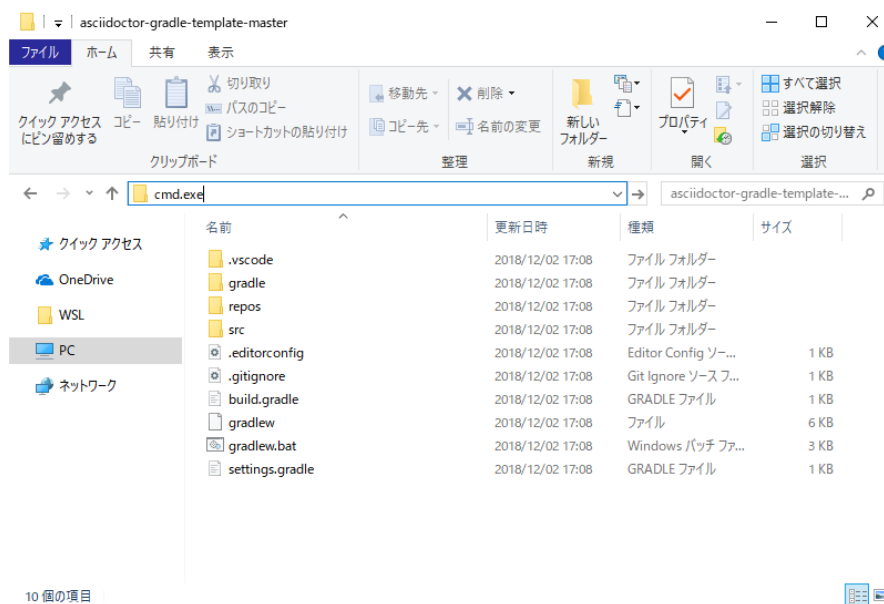
- ❶ リポジトリのファイルをダウンロードします。
- ❷ ダウンロードした .zip ファイルを展開します。
- ❸ カレントディレクトリを展開したフォルダの中に移します。フォルダ名は任意のものに変更可能です。本手順ではプロジェクトフォルダと呼称します。
- ❹ Gradle のビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回以降は30秒程度で完了します。
- ❺ BUILD SUCCESSFUL が出力されればビルド成功です。

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/hlromas4/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. プロキシサーバー設定 (Windows)

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

手順. プロキシサーバー設定 (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。

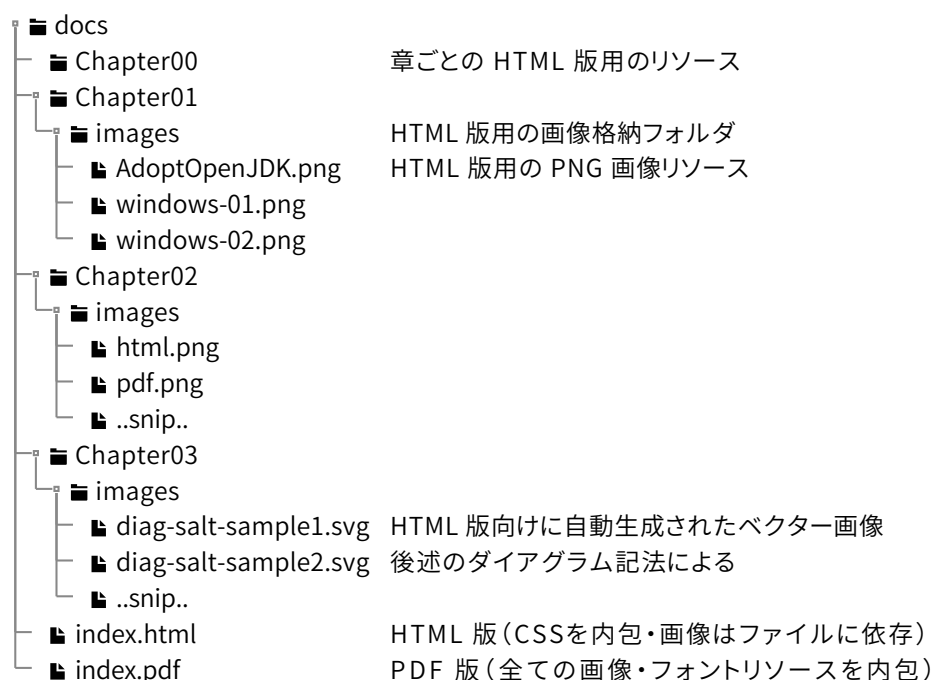


画像.HTML 文書

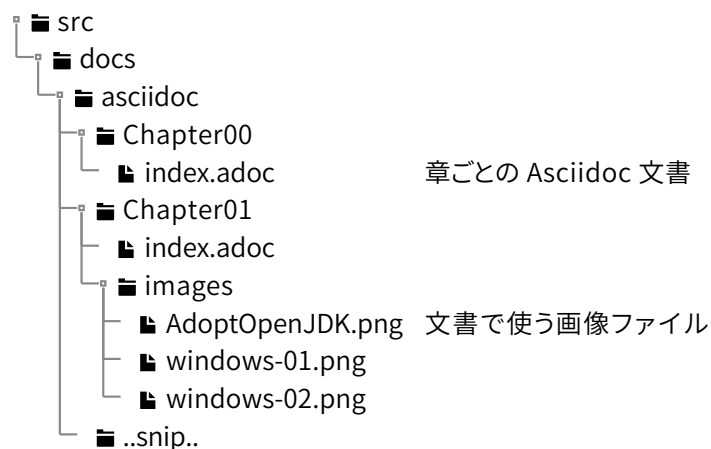


画像.PDF 文書

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。



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。



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 <code>index.pdf</code> 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。

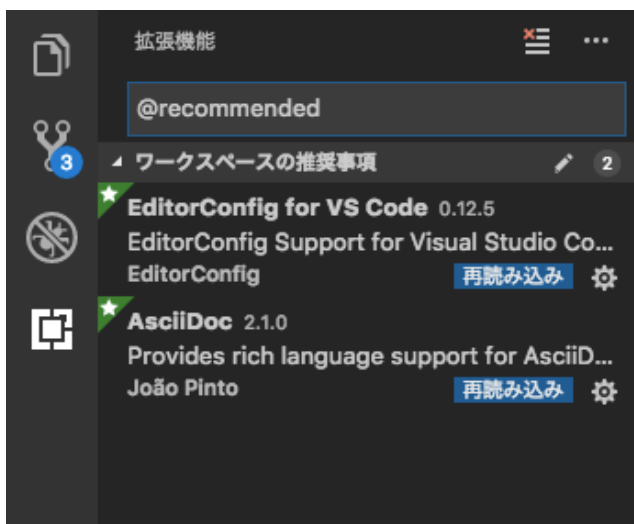


拡張機能の推奨は `.vscode/extension.json` で設定されています。文書に応じて設定し、執筆メンバーの環境を揃えることができます。

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。

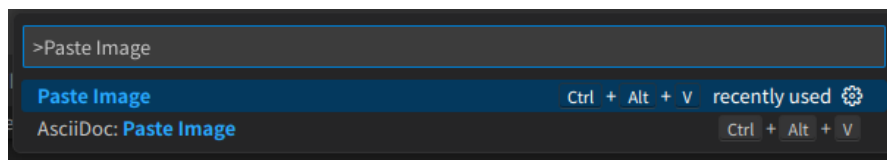


2.3.2. クリップボードからの画像挿入

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

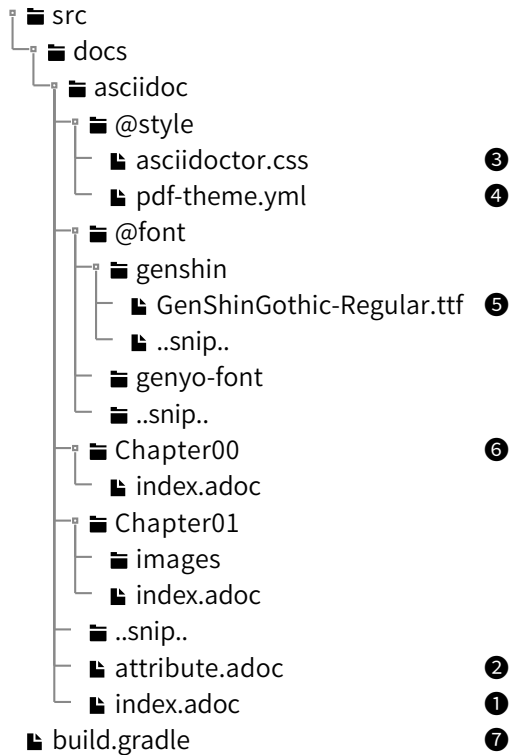
Paste Image 拡張のクリップボード連携のシステムインターフェースは次のシェルスクリプトで実装されています。AsciiDoc 拡張も `~/.vscode/extensions/asciidocctor.asciidocctor-vscode-2.9.8/res` 配下に同様のファイルがあります。

クリップボード連携は VS Code のセキュリティーポリシー外の世界で動作しますので、動作をご理解の上設定してください。

```
$ 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 ①
src/docs/asciidoc/attribute.adoc ②
src/docs/asciidoc/@style/asciidoctor.css ③
src/docs/asciidoc/@style/pdf-theme.yml ④
src/docs/asciidoc/@font/**/*.ttf ⑤
src/docs/asciidoc/Chapter{number}/index.adoc ⑥
build.gradle ⑦
  
```

- ① 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。
- ② 文書設定をするためのファイルです。各文書から `include` します。
- ③ 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。
- ④ 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。
- ⑤ 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。
- ⑥ `src/docs/asciidoc/index.adoc` から参照される子文書です。`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。
- ⑦ 文書を変換する Gradle ビルドスクリプトです。AsciiDoc 文書や画像パスの設定があります。

2.4.1. 文書属性定義

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

2.4.2. PDF テーマ定義

src/docs/asciidoc/@style/pdf-theme.yml では PDF 文書のスタイルを定義することができます。

属性の詳細は次から参照できます。

<https://docs.asciidoctor.org/pdf-converter/latest/>

AsciiDoctor PDF Theming Guide

— asciidoctor.org

pdf-theme.yml の修正は .adoc 修正時のビルドで PDF 文書に反映します。pdf-theme.yml の修正だけを反映したい場合は、いずれかの .adoc 文書に影響のない修正を加えてファイルを更新するか、./gradlew clean してからビルドを実行してください。



本手順で採用している AsciiDoctor PDF のバージョンは 1.6.2 です。日本語禁則処理を正しく動作させるために、現在のところ最新の 2.3 にはなっていません。新しい Theming Guide ドキュメントの属性や動作とは乖離している場合があることに注意してください。

2.5. 執筆の開始

一通りのサンプル文書の構成確認が заверши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/Chapter02` 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/Chapter00/index.adoc` を空にして目次前に挿入される「はじめに」や「本書の読み方」等の執筆。
5. `src/docs/asciidoc/Chapter01/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` の入力には 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。

2.7. クラウドによる執筆

クラウド統合環境である [Gitpod](https://www.gitpod.io/) を活用すると、ローカル環境の準備なしにウェブブラウザ上での執筆と HTML/PDF 文書の変換が可能です。サブマシンや iPad などタブレット端末でも執筆が行えます。

<https://www.gitpod.io/>

The developer platform for on-demand cloud development environments. Create software faster and more securely.

— Gitpod

実行にあたっての条件は次のとおりです。

- 文書ファイル一式を GitHub、GitLab、Bitbucket のいずれかに配置する必要があります。（プライベートリポジトリも利用可能です）
- 2023-07 現在 Gitpod の無料枠は 50時間/月です。それ以上は利用料が必要です。

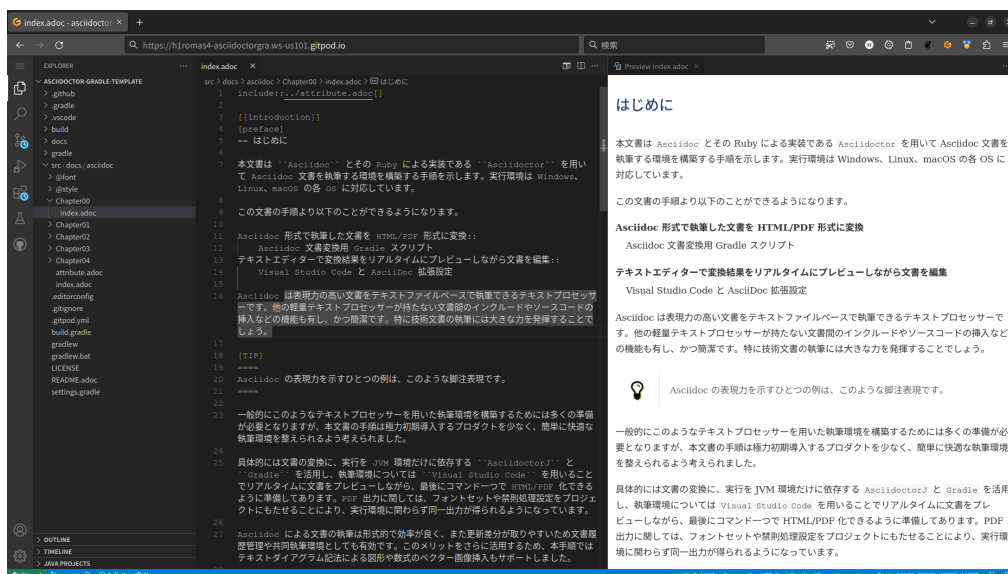
準備ができれば、次の要領で URL を作成しウェブブラウザでアクセスすると環境が起動します。まずは本手順のリポジトリで試してみるのも良いでしょう。



Gitpod 向けの設定は `.gitpod.yml` ファイルで定義されています。

<https://gitpod.io/new/#https://github.com/h1romas4/asciidoctor-gradle-template> ①

- ① <https://github.com/h1romas4/asciidoctor-gradle-template> 部分を自身の GitHub リポジトリにしてウェブブラウザの URL 欄に貼り付けます。本手順の試用の場合はこのままでかまいません。



画像.ウェブブラウザ上で起動した VS Code と AsciiDoc 拡張

起動後に VS Code 推奨拡張導入のダイアログが出力されますので、確認してインストールしてください。

2.8. GitHub Actions によるビルド

本手順のリポジトリに含まれるファイル `.github/workflows/gradle.yml` 及び `.github/workflows/release.yml` は GitHub サーバ上で文書の変換テスト、及び `.tar.gz` 形式での HTML/PDF 文書のリリースを行うための定義ファイルです。

これらのファイルを文書ファイルとともにお使いの GitHub にコミットすることで、変換テストとリリースが git の操作を契機として GitHub サーバ上で自動的に動作します。

2.8.1. `.github/workflows/gradle.yml`

git による main ブランチへの push 操作から自動的に動作する文書変換テストです。Linux、Windows、macOS のそれぞれの環境で、Java 11 と 17 においてビルドが試行され、一定の検査が行われます。

環境はマトリクスになっており、計 6 環境でのテストでビルドが正しいかが確認され GitHub Actions 上のログやメール通知で結果を知ることができます。自分の執筆環境以外のさまざまな環境で、変換処理が実行できることへの確信を得ることができるはずです。

検査に関しては、文書が生成されたかを基本として確認を行い、文書の内容自体に対しては行われません。

- HTML、PDF 文書ファイルが存在するか。
- 後述の `asciidoctor-diagram` による `.svg` 画像生成が動作したか。
- GitHub Actions ログへのファイル数、一覧の出力。

検査は以下の部分で行われます。文書に合わせたさらなる検査が必要な場合は Linux、macOS ではシェルスクリプトで、Windows では PowerShell の記述でカスタマイズしてください。

.github/workflows/gradle.yml

```

- name: Check (UNIX)
  if: startsWith(matrix.os, 'ubuntu') || startsWith(matrix.os, 'macos')
  run: |
    find docs/ -type f -ls
    find docs/ -type f -ls | wc
    # test asciidoctor-diagram (If you are not using diagram, comment out the next two
lines)
    find docs/ -type f -ls | egrep 'svg$' ❶
    # test asciidoctor
    file docs/index.pdf | grep PDF
    file docs/index.html | grep HTML

- name: Check (Windows)
  if: startsWith(matrix.os, 'windows')
  shell: pwsh
  run: |
    Get-ChildItem -Recurse -File docs
    $(Get-ChildItem -Recurse -File docs).Length
    # test asciidoctor-diagram (If you are not using diagram, comment out the next two
lines)
    Get-ChildItem -Recurse -File -Name -Filter "*.svg" docs ❶
    (Get-ChildItem -Recurse -File -Name -Filter "*.svg" docs).Length -ne 0 ❶
    # test asciidoctor
    Get-ChildItem docs/index.pdf
    Get-ChildItem docs/index.html

```

❶ asciidoctor-diagram を使った .svg 図表が生成されているかの検査。文書中にない場合はチェックを削除するか、コメントアウトしてください。

なお、文書の変換ができたことの証となる「バッジ」は以下のようにすることで画像として取得可能です。正常の場合はグリーンに、失敗の場合は赤で表示されます。

[https://github.com/\[ユーザ名\]/\[リポジトリ名\]/workflows/Build/badge.svg](https://github.com/[ユーザ名]/[リポジトリ名]/workflows/Build/badge.svg)

具体的には、本手順のリポジトリの GitHub 向けの README.adoc は次のようになっています。

image::https://github.com/h1romas4/asciidoctor-gradle-template/workflows/Build/badge.svg[]



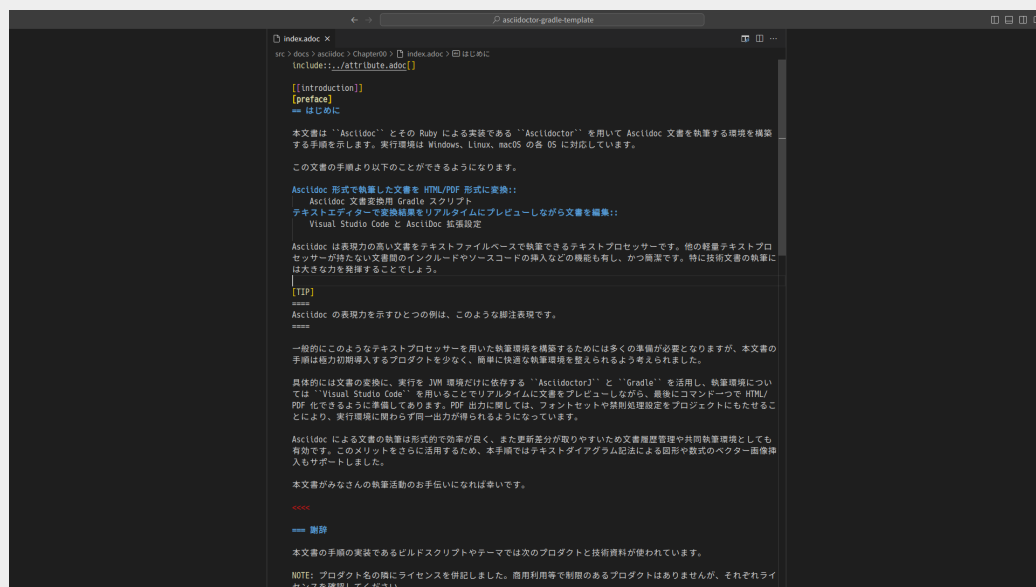
2.8.2. .github/workflows/release.yml

git のバージョンタグを push する操作で、GitHub のリリースページにソースコード一式と GitHub Actions 上で変換処理された docs 配下のファイル docs.tar.gz を掲載する GitHub Actions 定義です。

頒布や出版時にタグを打つことで、その時点のファイル断面がダウンロード可能な形で固定化できます。文書の版管理に活用してみてください。

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。

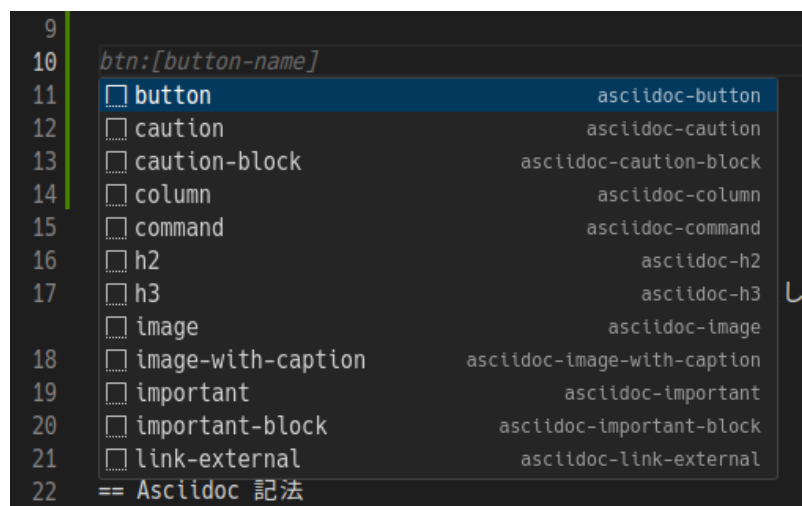


Zen Mode は、VS Code や OS の不要なコンポーネントが見えなくなり画面が広く使えること、また思考を遮る要素がなくなることから快適な執筆のユーティリティとなります。

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。

入力の助けになるよう、プロジェクトフォルダの `.vscode/asciidoc.code-snippets` 設定には VS Code の [Ctrl + Space] 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ば追加・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() { // <1>
    console.log("Hello, World!"); // <2>
}
----

<1> 関数定義
<2> 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() { ❶
    console.log("Hello, World!"); ❷
}
```

❶ 関数定義

❷ 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

```
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。



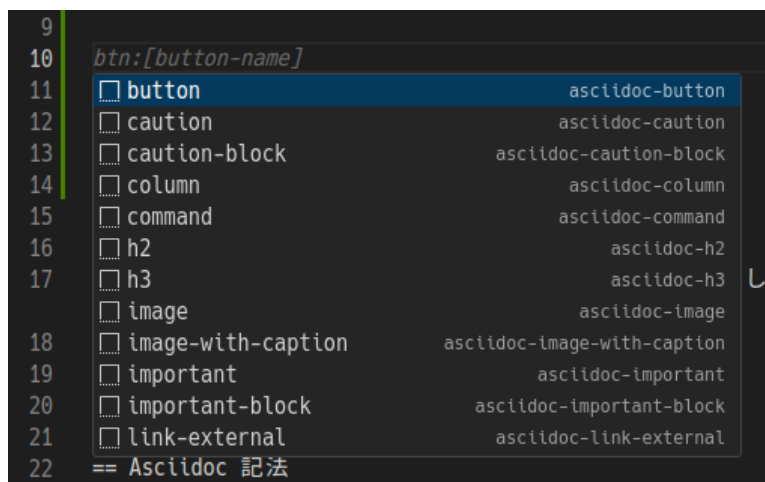
AsciiDoc はリスト中にリスト継続マーカ (+) を使うことでブロックのネストが可能です。手順を順序ありリストで表現し、順序やインデントを維持しながら画像や表形式などの挿入をします。

```
. 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 文書のみには効果があり 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。

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



出力ファイル名は `images` フォルダ内で一意になるように設定してください。特にダイアログ記法を別の場所からコピーアンドペーストした場合は要チェックです。同名ファイル名が指定された場合は上書きされる動作になります。

`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に、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/>

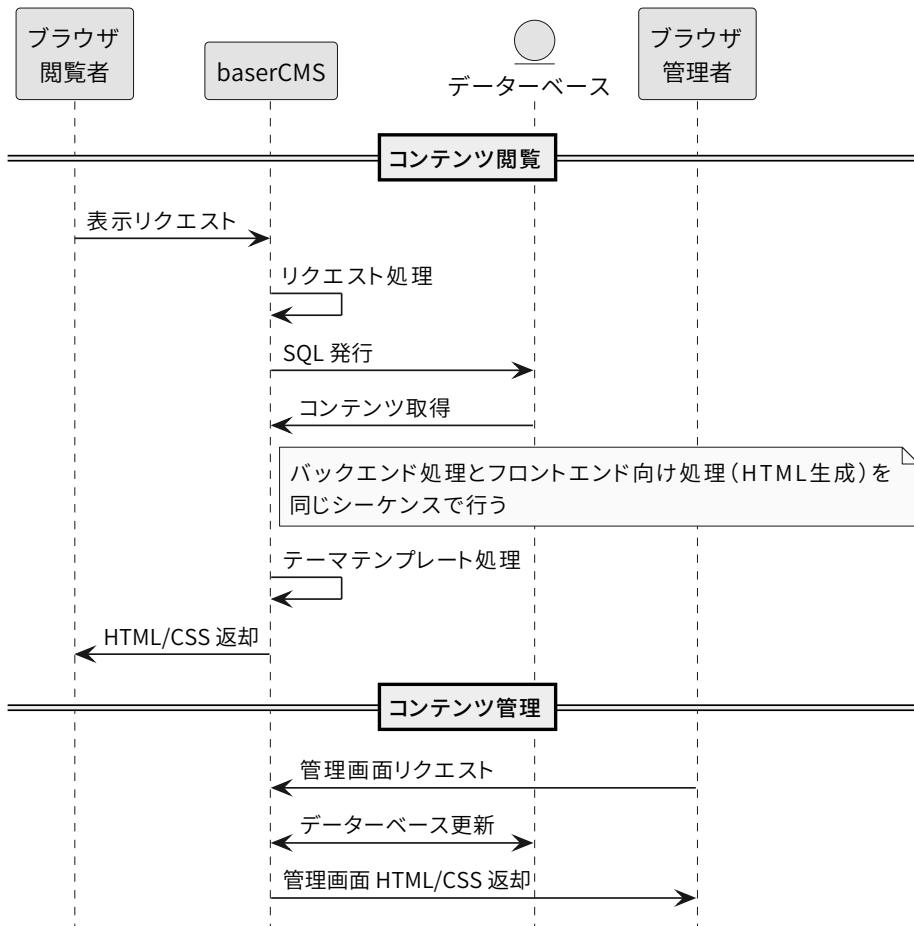
DIagrams Through Ascii Art by Stathis Sideris

— 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)^oo a_n cos((n*pi*t)/L)+sum_(n=1)^oo b_n sin((n*pi*t)/L)
@endmath
----
```

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

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

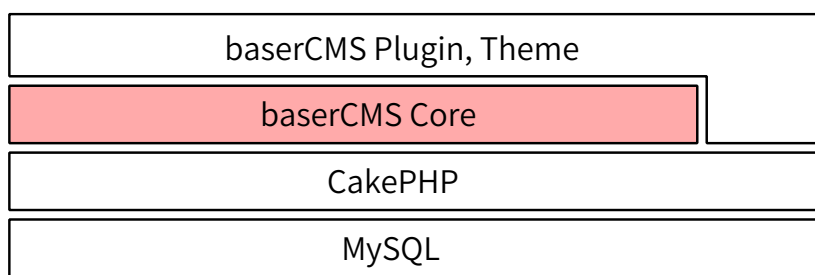
ブロック図を ditaa 記法で記述します。



ditaa 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、ドロップシャドウ 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    |  
\-----+-----/  
-----
```



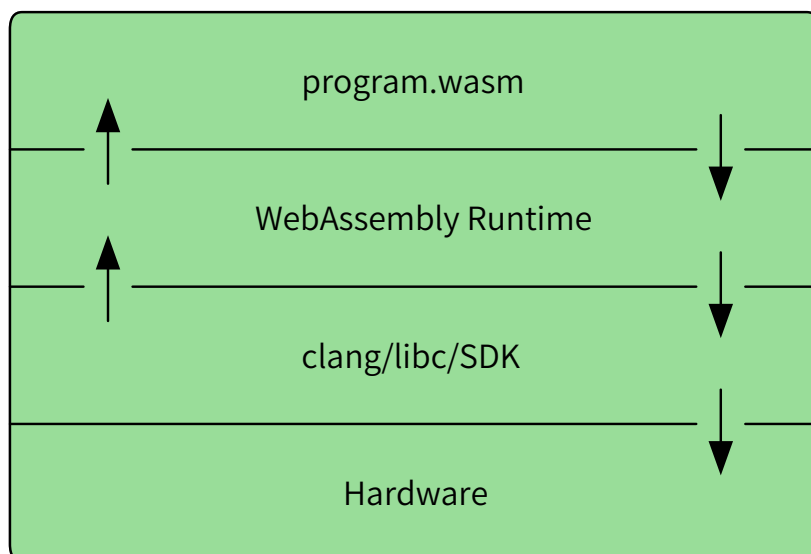
```
[ditaa, diag-ditaa-block-2, svg, shadows=true, separation=true, pdfwidth=70%, width=70%]
```

```
[caption=""]
```

```

-----
/-----\
|               |
|   program.wasm   |
|   ^             |
+---+-----+---+
|   |             |   v
|   | WebAssembly Runtime |
|   ^             |
+---+-----+---+
|   |             |   v
|   | clang/libc/SDK      |
|   |             |
+---+-----+---+
|   |             |   v
| cGRE   Hardware      |
|   |             |
\-----/
-----

```

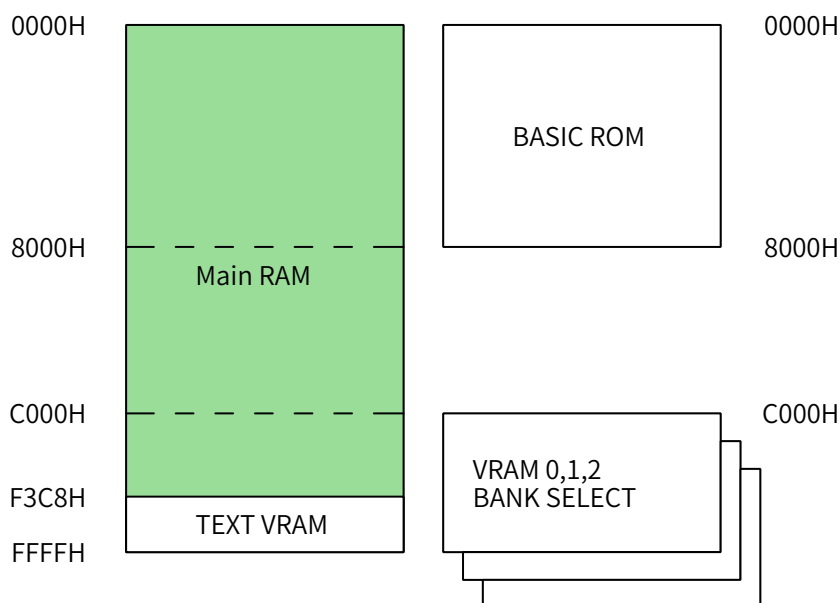


4.5. メモリーマップ

メモリーマップを ditaa 記法で記述します。

```
[ditaa, diag-ditaa-memory-map, svg, shadows=true, separation=false, pdfwidth=70%, width=70%]  
[caption=""]
```

```
-----  
0000H +-----+ +-----+ 0000H  
      | cGRE      | |      |  
      |          | |      |  
      |          | | BASIC ROM |  
      |          | |      |  
      |          | |      |  
8000H +-----+ +-----+ 8000H  
      | Main RAM  | |      |  
      |          | |      |  
      |          | |      |  
C000H +-----+ +-----+ C000H  
      |          | |      | ++  
      |          | | VRAM 0,1,2 | ++  
F3C8H +-----+ | BANK SELECT | |  
      | TEXT VRAM | |      | |  
FFFFH +-----+ +-----+ |  
      |          | ++-----+ |  
      |          | +-----+  
-----
```



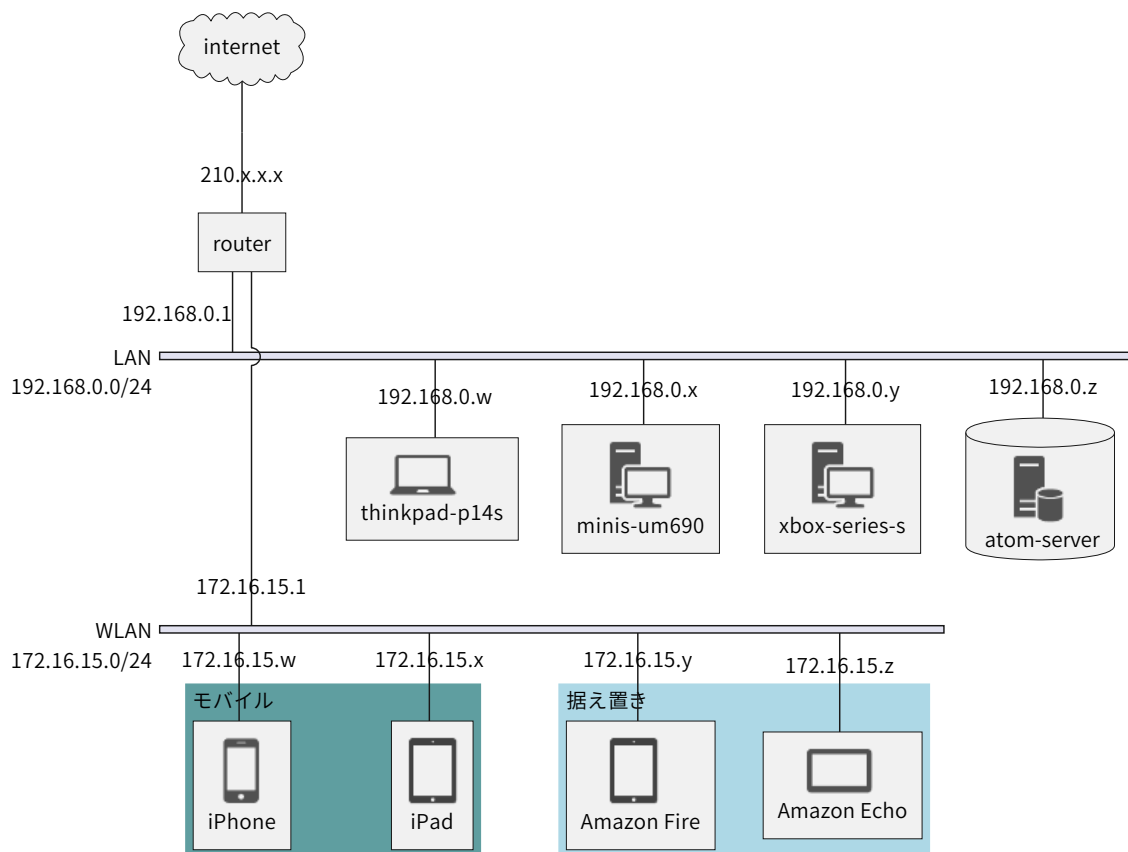
4.6. ネットワーク構成図

ネットワーク構成図を PlantUML の nwdiag で記述します。

diag-nwdiag-network1.puml - [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
----
```



WSL2 上の VS Code で Paste Image 拡張を使う

本手順は Windows WSL2 上の Linux と VS Code の WSL Remote 拡張でも構成できますが、VS Code Paste Image 拡張はそのままでは OS のクリップボードを跨げないためうまく動作しません。

これを解決するために、ワークアラウンドですが次のように WSL2 側の拡張のファイルを書き換えると動作するようになります。AsciiDoc 拡張も同じ方法で修正できます。なお、拡張にアップデートがあるとファイルが戻りますので注意してください。

~/vscode-server/extensions/mushan.vscode-paste-image-1.0.4/res/linux.sh

```
#!/bin/sh
# change current directory
cd `dirname $0`
# set dist path
DISTPATH=//wsl.localhost/Ubuntu-22.04 ❶
# call Windows PowerShell
powershell.exe -ExecutionPolicy Bypass -File pc.ps1 ${DISTPATH}$1
```

❶ 利用しているディストリビューションのパスを設定する。

4.7. システム構成図

簡単なシステム構成図を [PlantUML/C4 コンポーネント](#) で記述します。

```
[plantuml, diag-c4-component-sample1, svg]
[caption=""]
----
@startuml diag-c4-component-sample1
!include <C4/C4_Container>
!include <office/users/users.puml>
!include <office/Servers/database_server>
!include <office/Servers/file_server>
!include <office/Servers/application_server>
!include <office/Concepts/service_application>
!include <office/Concepts/firewall>
!include <office/Devices/workstation>

skinparam linetype ortho
top to bottom direction
'left to right direction
HIDE_STEREOYPE()

AddContainerTag("user", $bgColor="#5f9ea0", $shadowing="true")
AddContainerTag("linux", $sprite="application_server", $bgColor="#dcdcdc",
$fontColor="#000000", $borderColor="#696969", $sprite="file_server", $shadowing="true")

Person(user_person, "ウェブサイト\n閲覧者", $sprite="users", $tags="user")

System_Boundary(system_1, "Application") {
    Container_Boundary(bound_backend_1, "Backend") {
        ContainerDb(rel_db, "Database", "MySQL 8.0", $sprite="database_server", $tags="linux")
        Container(filesystem, "File System", "ext4", $tags="linux")
        Container(batch, "Bacth Server", "Java, Spring Boot", $tags="linux")
    }
    Container_Boundary(bound_frontend_1, "Frontend") {
        Container(load_balancer, "Load balancer", "nginx", $sprite="application_server",
$tags="linux")
        Container_Boundary(bound_cluster_1, "Cluster") {
            Container(web_app_1, "Web Application", "Java, Spring Boot", $tags="linux")
            Container(web_app_2, "Web Application", "Java, Spring Boot", $tags="linux")
        }
    }
    Person(user_admnin, "アプリケーション\n管理者", $sprite="workstation", $tags="user")
}

Lay_R(user_person, system_1)
Lay_D(user_admnin, batch)
Lay_R(bound_backend_1, bound_frontend_1)

Lay_R(load_balancer, web_app_1)
Lay_D(web_app_1, web_app_2)
Lay_D(rel_db, filesystem)
Lay_R(rel_db, batch)

AddRelTag("rel_line", $textColor="black", $lineColor="#696969", $lineStyle=BoldLine())
```

```

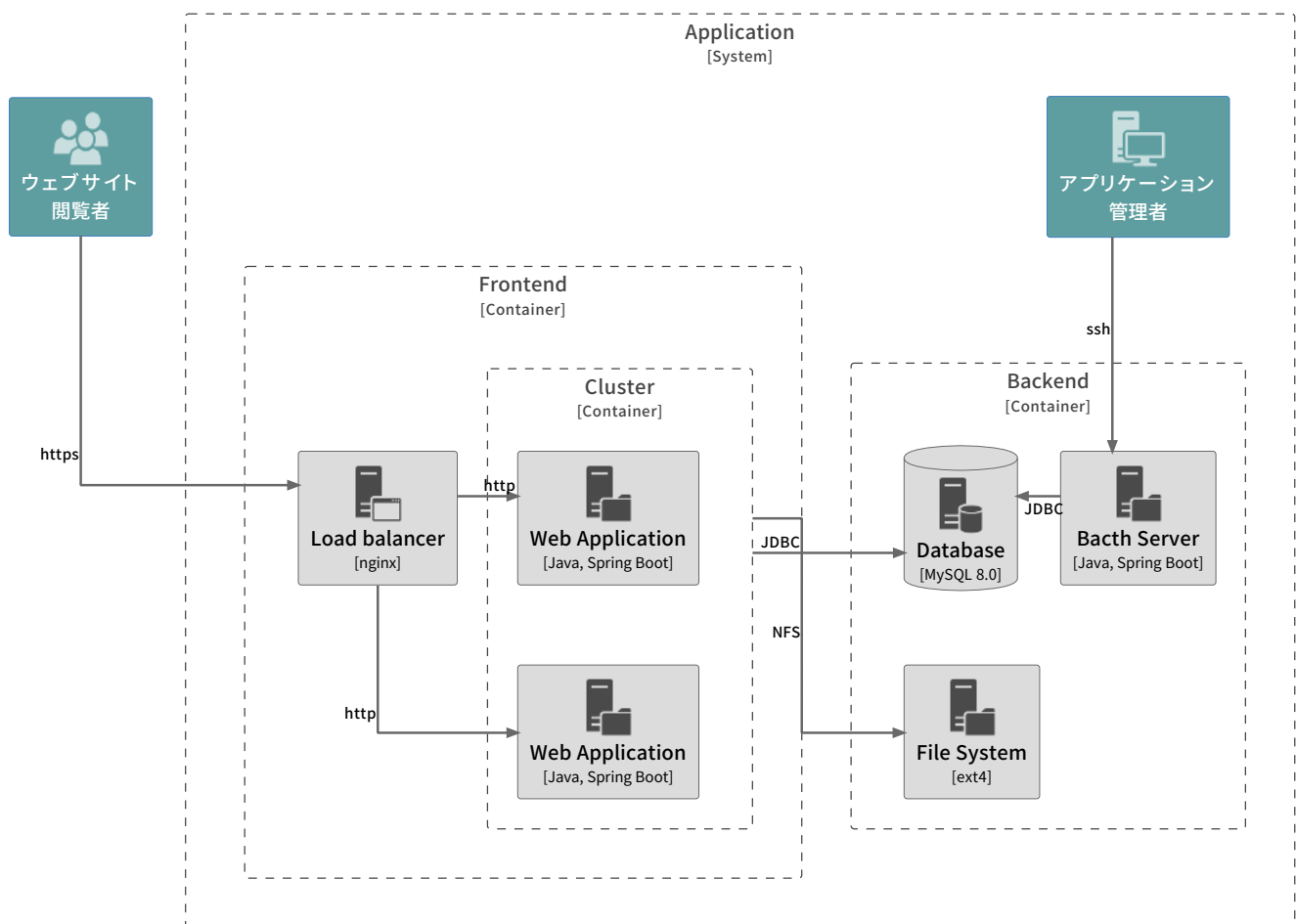
Rel_R(user_person, load_balancer, "https", $tags="rel_line")
Rel_R(load_balancer, web_app_1, "http", $tags="rel_line")
Rel_R(load_balancer, web_app_2, "http", $tags="rel_line")

Rel_R(bound_cluster_1, rel_db, "JDBC", $tags="rel_line")
Rel_R(bound_cluster_1, filesystem, "NFS", $tags="rel_line")

Rel_R(batch, rel_db, "JDBC", $tags="rel_line")
Rel_R(user_adminin, batch, "ssh", $tags="rel_line")

@enduml
-----

```



現在のところ `$shadowing="true"` 指定のドロップシャドウは HTML 版のみ反映します。

4.8. 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 - 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 high
A0 is low
A1 is low

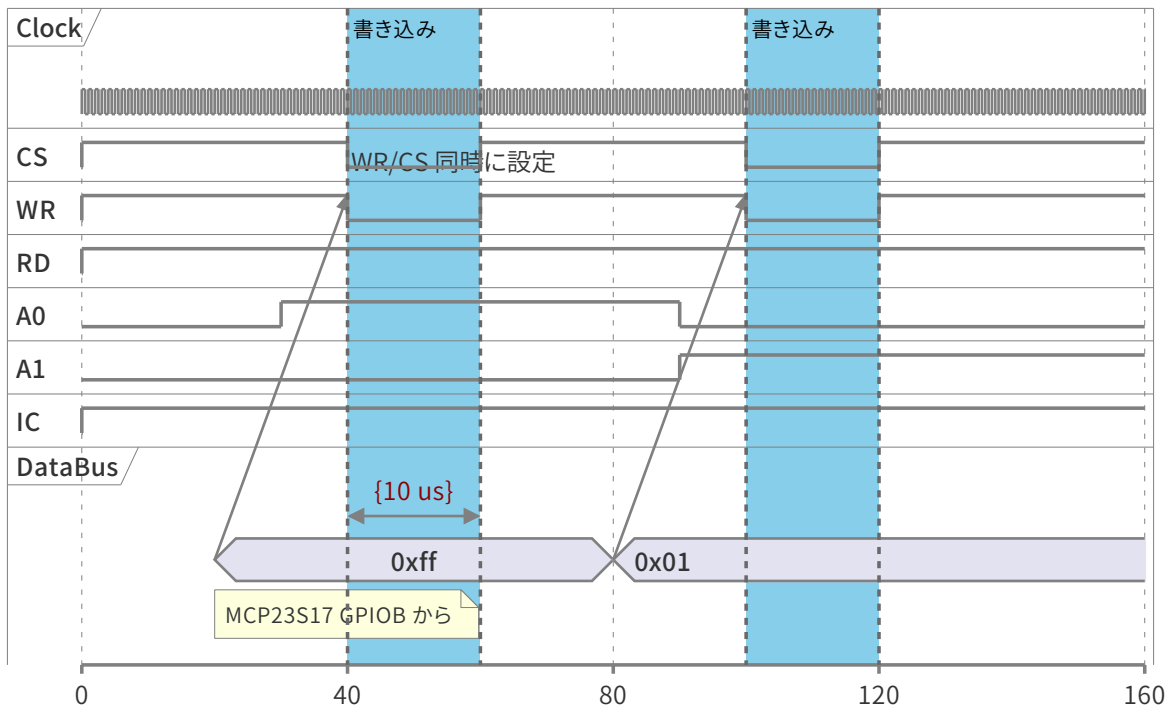
@:set_data_bus_1
DB is "0xff"
DB -> WR@+20
note bottom of DB : MCP23S17 GPIOB から

@:set_addr_1
A0 is high
A1 is low

@:write_start_1
CS is low : WR/CS 同時に設定
WR is low
DB@40 <-> @60 : {10 us}

@:write_end_1
CS is high
WR is high

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



Open Iconic アイコン

PlantUML 内で `<アイコン名>` 形式で使える Open Iconic アイコンは次の通りです。アイコン名は HTML 版の場合 [ウェブブラウザ](#) > [右クリック](#) > [画像を開く](#) するとコピーアンドペーストで取得できます。

List Open Iconic						
Credit to						
https://useiconic.com/open						

4.9. 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
----
```

