

Document Authoring Environment

<http://github.com/kazurayam/DAE>

2020-04-13

目次

1. はじめに	1
1.1. 謝辞	1
2. AsciiDoc 文書変換用スクリプトを使う準備をする	2
2.1. Java 実行環境の導入（Windowsの場合）	5
3. AsciiDocからHTML/PD文書を作成する	6
3.1. サンプル文書の変換を試す	6
3.2. テキストエディタで AsciiDoc 文書を編集する	9
3.3. 文書のファイル構成	11
3.4. asciidoctorj-diagram	13

1. 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。

この文書の手順に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 形式に変換する。

AsciiDoc 文書変換用スクリプト

変換結果をリアルタイムにプレビューしながら、エディターで文書を編集する。

Visual Studio Code 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 でこのような脚注を表現することができます。AsciiDoc の高い表現力を示す一つの例です。

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ています。

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 PDF 化できるように準備してあります。

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。

1.1. 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/>

2. AsciiDoc

文書変換用スクリプトを使う準備をする

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あなたが自分のコンピュータの OS 環境に手動で導入する必要がある唯一のプロダクトです。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。

あなたがお使いのコンピュータ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_242"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_242-b08)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.242-b08, mixed mode)
```

Windowsの場合

```
C:\> java -version
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_242-b08)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.242-b08, mixed mode)
```



現在 Java 9 以降の環境ではビルド時にワーニングが出力されるため本手順では Java 8 を使って解説しています。筆者の Java 11 環境で変換の動作は正しいことが確認できていますので適宜読み替えて Java を導入してください。この問題は将来解消されるでしょう。

=== Java 実行環境の導入(macOS / Linuxの場合)

もしあなたの 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。

SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix bases systems.

<https://sdkman.io/>

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
=====
```

Vendor	Use	Version	Dist	Status	Identifier
AdoptOpenJDK		14.0.0.j9	adpt		14.0.0.j9-adpt
		14.0.0.hs	adpt		14.0.0.hs-adpt
		13.0.2.j9	adpt		13.0.2.j9-adpt
		13.0.2.hs	adpt		13.0.2.hs-adpt
		12.0.2.j9	adpt		12.0.2.j9-adpt
		12.0.2.hs	adpt		12.0.2.hs-adpt
		11.0.6.j9	adpt		11.0.6.j9-adpt
		11.0.6.hs	adpt		11.0.6.hs-adpt
Azul Zulu		8.0.242.j9	adpt		8.0.242.j9-adpt
		8.0.242.hs	adpt		8.0.242.hs-adpt
		14.0.0	zulu		14.0.0-zulu
		13.0.2	zulu		13.0.2-zulu
		12.0.2	zulu		12.0.2-zulu
		11.0.6	zulu		11.0.6-zulu
		11.0.6.fx	zulu		11.0.6.fx-zulu
		10.0.2	zulu		10.0.2-zulu
		9.0.7	zulu		9.0.7-zulu
		8.0.242	zulu		8.0.242-zulu
		8.0.232.fx	zulu		8.0.232.fx-zulu
		8.0.202	zulu		8.0.202-zulu
		7.0.181	zulu		7.0.181-zulu

```
=====
Use the Identifier for installation:

$ sdk install java 11.0.3.hs-adpt
=====
$ sdk install java 8.0.242.hs-adpt ❹
```

- ❶ SDKMANを導入します。
- ❷ SDKMANを起動するシェルスクリプト `sdkman-init.sh` を現在実行中のシェルにロードします。
- ❸ 今現在導入可能なJavaのバージョンを一覧します。

- ④ 8.0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、`.bash_profile` で次のように JAVA_HOME を設定します。

手順. JAVA_HOME の設定

```
$ vi ~/.bash_profile ①
export JAVA_HOME=~/.sdkman/candidates/java/current ②
$ source ~/.bash_profile ③
```

- ① vi エディタで `.bash_profile` を開きます。
- ② 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。
- ③ 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 11.0.1-open ①
$ sdk default java 11.0.1-open ②
$ sdk default java 8.0.192-zulu ③
```

- ① Java 11 を導入
- ② Java 11 をデフォルトに設定
- ③ Java 8 をデフォルトに設定

2.1. Java 実行環境の導入(Windowsの場合)

TODO

3. AsciiDocからHTML/PD文書を作成する

3.1. 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色と、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を取得して実行する

```
$ curl -L -O https://github.com/kazurayam/DAE/archive/master.zip ❶
$ unzip master.zip ❷
$ cd DAE 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 展開してできたディレクトリのなかにcd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群をダウンロードするため少し時間がかかります。次回からは数秒で完了します。
- ❺ BUILD SUCCESSFUL が出力されればビルド成功です。

Windowsをお使いの場合に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。

TODO

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を経由してインターネットにアクセスする場合は、次のコマンドを `./gradlew docs` する前に投入してください。インターネットからライブラリをダウンロードして取得するのに必要です。ホスト名(example.com)とポート番号(8080)はそれぞれの環境に合うように変更してください。

手順.プロキシサーバーの設定(Windows)

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

手順.プロキシサーバーの設定(macOS / Linux)

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

./gradlew docs タスクが実行されて出力された文書は次の場所に格納されます。

```
docs/index.html
docs/index.pdf
```

docs フォルダはもっぱらビルドからの出力を格納するために使われ、ビルド開始時にいったん全てのファイルが削除されますから、ユーザは自作したファイルをここに配置しないよう注意してください。

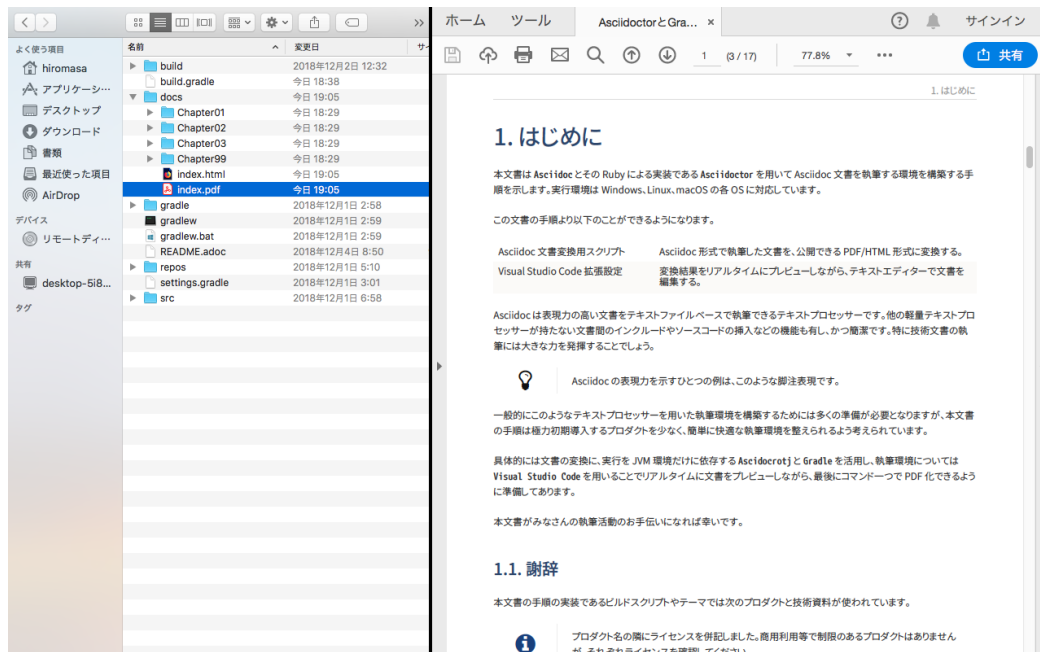


図 1. PDF文書

目次

- 1. はじめに
 - 1.1. 謝辞
- 2. AsciiDoc 文書変換用スクリプトを使う準備
 - 2.1. Java 実行環境の導入 (macOS / Linux の場合)
 - 2.2. Java 実行環境の導入 (Windows の場合)
- 3. AsciiDoc から PDF/HTML 文書を作成する
 - 3.1. サンプル文書の変換を試す
- 4. AsciiDoc テンプレート集
 - 4.1. 脚注
 - 4.2. 画像挿入
 - 4.3. キーボードショートカット表記
 - 4.4. 属性値の文書への挿入
 - 4.5. ソースコード (直接記述)
 - 4.6. ソースコード (外部ファイルの include)
 - 4.7. サイドバー
 - 4.8. 引用
 - 4.9. 表組み
 - 4.10. 改ページ
 - 4.11. 水平線
 - 4.12. リスト
 - 4.13. リスト (順番あり)

AsciiDoctorとGradleでつくる文書執筆環境

<https://github.com/h1romas4/asciidoctor-gradle-template> - 2018-12-01

1. 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 文書変換用スクリプト	AsciiDoc 形式で執筆した文書を、公開できる PDF/HTML 形式に変換する。
Visual Studio Code 拡張設定	変換結果をリアルタイムにプレビューしながら、テキストエディターで文書を編集する。

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ています。

図 2. HTML文書



文書を github 上に公開する場合は、プロジェクトのファイルを全てコミットし、GitHub Pages に docs フォルダを指定することで継続的に文書をパブリッシュすることができます。

3.2. テキストエディタで AsciiDoc 文書を編集する

Visual Studio Codeを使ってAsciiDoc文書（*.adocファイル）をリアルタイムにプレビューしながら編集してみましょう。

プロジェクトフォルダ(DAE)をVisual Studio Codeで開きましょう。index.adocファイルを開くとファイル拡張子`adoc`にたいしてこのプラグインを使うことを推奨しますというダイアログが表示されます。ダイアログにしたがってプラグインを導入しましょう。

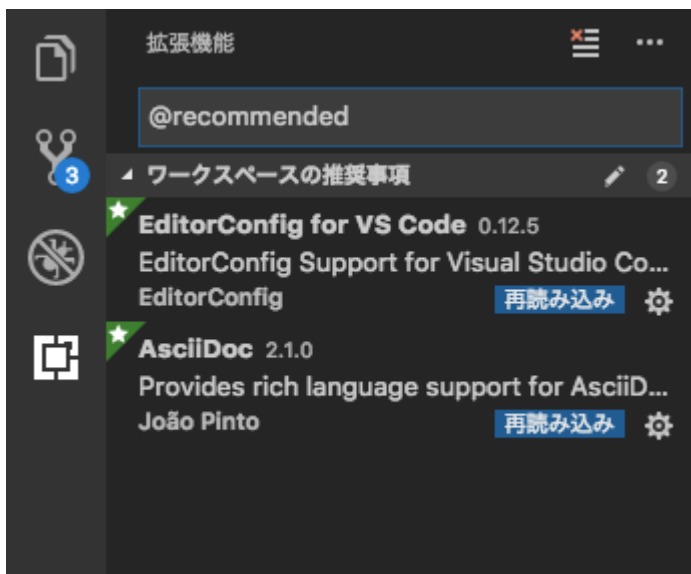


どのプラグインを推奨するかは .vscode/extensions.json で設定されています。

1. [すべてをインストール]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。

The image shows a side-by-side comparison of an AsciiDoc source file and its rendered HTML output. The left pane displays the raw text of the document, including instructions in Japanese and a terminal window showing the execution of a Gradle build script. The right pane shows the same content rendered as a web page, with the terminal output highlighted in a dark box and numbered steps corresponding to the commands.

Left Pane (Source AsciiDoc):

```
4 == AsciiDoc から HTML/PDF 文書を作成する
5
6 === サンプル文書の変換を試す
7
8 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。
9
10 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
11   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
12   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
13   を試してみましょう。
14
15 [source]
16 [caption="手順. "]
17 .PDF 変換ビルドスクリプトの取得と実行
18 ----
19 $ curl -L -O
20   https://github.com/hlromas4/asciidoctor-gradle-template/archive/master.
21   zip # <1>
22 $ unzip master.zip # <2>
23 $ cd asciidoctor-gradle-template-master # <3>
24 $ ./gradlew docs # <4>
25 BUILD SUCCESSFUL in 19s <5>
26 2 actionable tasks: 2 executed
27 ----
28
29 <1> リポジトリのファイルをダウンロードします。
30 <2> ダウンロードした .zip ファイルを展開します。
31 <3> 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。
32 <4> 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
33   ンロードするため少し時間がかかります。次回は数秒で完了します。
34 <5> ``BUILD SUCCESSFUL`` が出力されればビルド成功です。
```

Right Pane (Rendered HTML):

1. AsciiDoc から HTML/PDF 文書を作成する

1.1. 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。

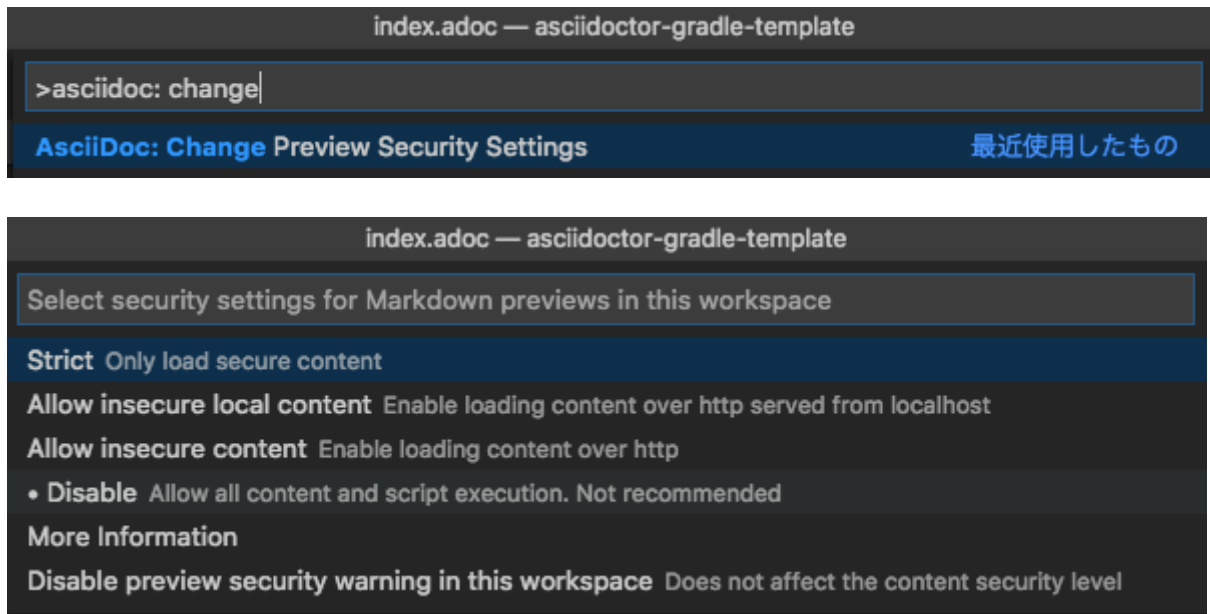
macOS / Linux の場合は次のようにします。

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/hlromas4/asciidoctor-gradle-template/archive/master.zip 1
$ unzip master.zip 2
$ cd asciidoctor-gradle-template-master 3
$ ./gradlew docs 4
BUILD SUCCESSFUL in 19s 5
2 actionable tasks: 2 executed
```

- 1 リポジトリのファイルをダウンロードします。
- 2 ダウンロードした .zip ファイルを展開します。
- 3 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。
- 4 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。次回は数秒で完了します。
- 5 BUILD SUCCESSFUL が出力されればビルド成功です。

なお、もしリアルタイムプレビューでソースコードのハイライトが動作しない場合は、AsciiDoc 文書を開いた状態で [F1] を押し、**Change Preview Security Setting > Disable** を選択してください。



本設定はプレビュー画面で動作するWebviewがインターネット上の外部リソースを評価できるようにセキュリティを下げる設定です。



プロジェクトつまりファイルが配置されたフォルダ毎の設定となりますが、この設定をしたあとはこのプロジェクトで第三者の作成した未知の AsciiDoc 文書を開かないようにしてください。外部から取り込んだ悪意のJavaScriptによセキュリティを侵害される可能性があります。この点、理解の上で設定してください。

3.3. 文書のファイル構成

文書は次のようなファイル群で構成します。サンプル文書を元に、自分が執筆しようとする文書に合わせてカスタマイズしていくとよいでしょう。

文書を構成するファイル

```
src/docs/asciidoc/index.adoc      ❶
src/docs/asciidoc/attribute.adoc  ❷
src/docs/asciidoc/@style/*.css    ❸
src/docs/asciidoc/@style/pdf-theme.yml  ❹
src/docs/asciidoc/@font/*         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。ここから他のAsciiDoc文書を `include` することで大きな文書を構成していきます。
- ❷ 文書全体に適用すべき設定を記述するためのファイルです。各文書はこれを `include` 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してかまいません。

- ④ PDF文書に変換する際に使われるスタイルシートです。文書に合わせて修正してかまいません。
- ⑤ PDF文書に埋め込みされるフォントファイルです。pdf-theme.ymlから参照されます。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。{number}の部分は具体的には01、02、03です。

build.gradleのなかでこれらのファイル構成を処理系に伝達するための設定をしています。

build.gradle

```
asciidoctor {
    asciidoctorj {
        attributes 'stylesdir': '@style',
                  'stylesheet': 'asciidoctor.css'
    }
}

asciidoctorPdf {
    asciidoctorj {
        attributes 'pdf-stylesdir': "@style",
                  'pdf-style': 'pdf'
    }
}
```

src/docs/asciidoc/attribute.adoc ではエディタによるリアルタイム・プレビューで適用すべき属性を定義しておき、HTML/PDFを出力する際にはbuild.gradleで属性を必要に応じて上書きするのが良いでしょう。

Asciidoctorで利用可能な属性は次から参照できます。

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.

<https://asciidoctor.org/docs/user-manual/#attributes>

また、文書に挿入する画像ファイルは images というフォルダの中に配置することを想定しています。画像を格納するフォルダは下記の部分を書き替えることで変更可能です。

src/docs/asciidoc/attribute.adoc

build.gradle

```
copy {
    from 'src/docs/asciidoc/'
    include 'Chapter*/images/*'    // 子文書のフォルダ
    into 'docs'
}
```

.adocファイルの中から画像ファイルを参照する記述においては、.adoc ファイルを基底とする相対パスで画像ファイルをリンクします。

3.3.1. 文書から文書への相互参照

文書のなか参照リンクは次の様に記述します。

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

- ❶ 同じ .adoc のなかでの内部参照
- ❷ .adoc を跨いだドキュメント間参照

リンクの例

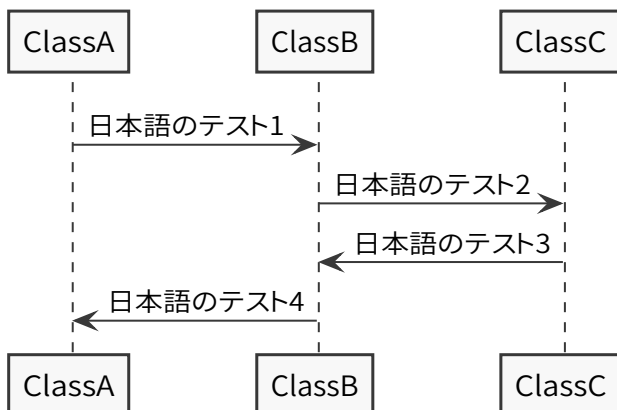
- [章の冒頭](#) を参照してください・・・
- [はじめに](#) で述べたように・・・

3.4. asciidoctorj-diagram



本変換ビルドスクリプトは `asciidoctorj-diagram` が有効になっており、いくつかのダイアグラム表記を使うことができます。

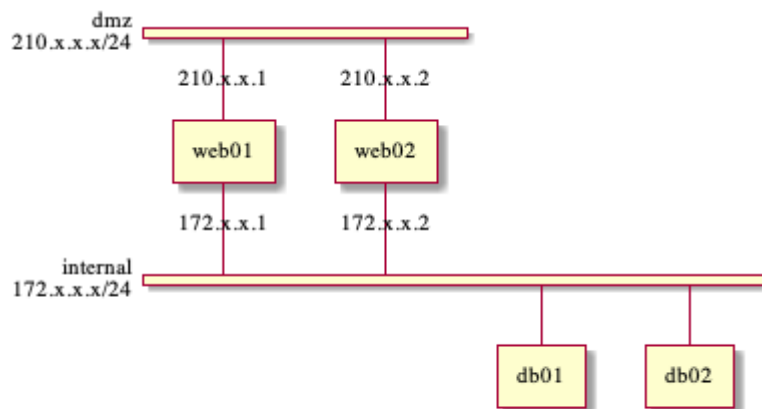
3.4.1. PlantUML

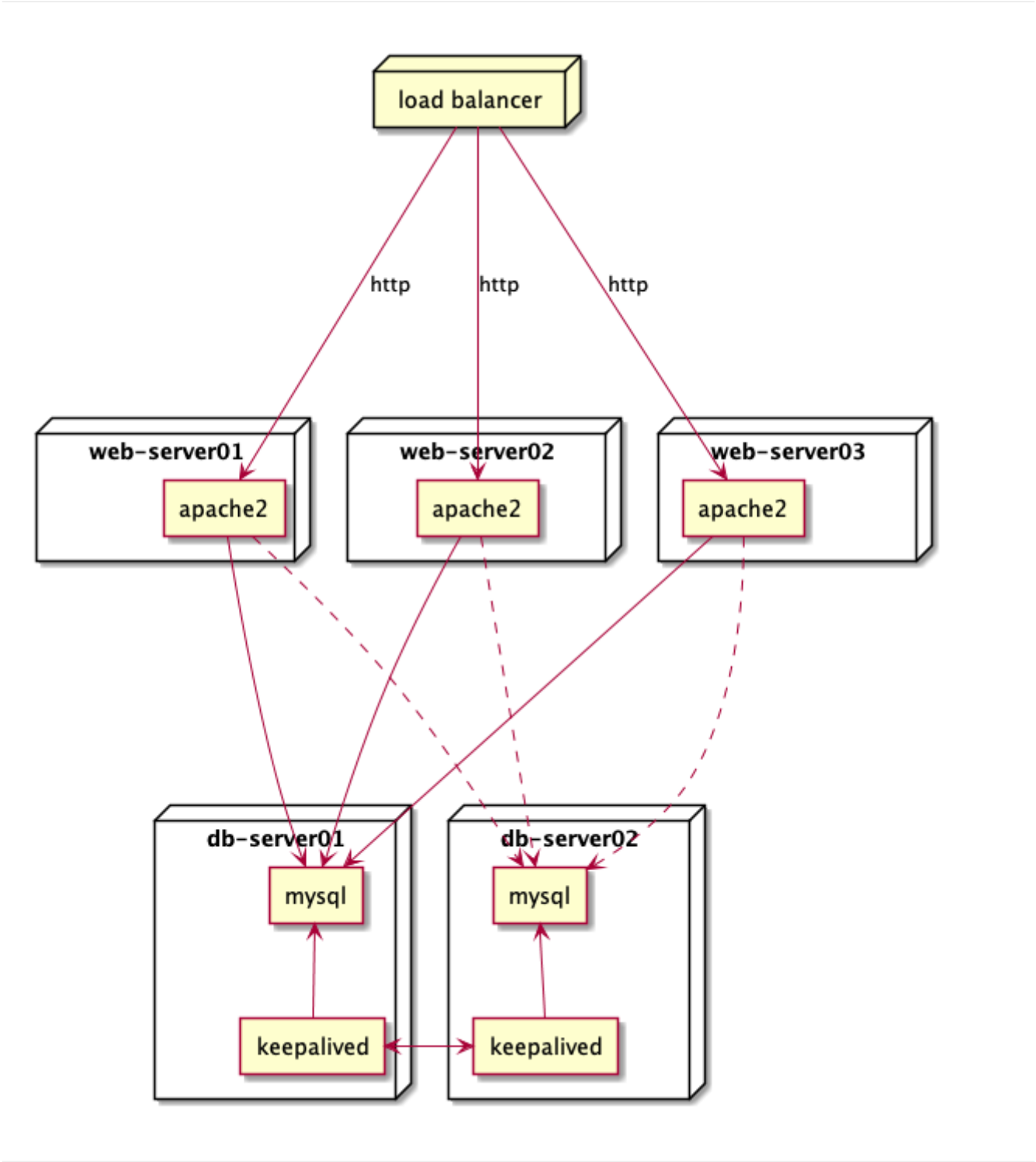
PlantUML形式のダイアグラムを `.svg` ベクトル画像に変換した上で文書に埋め込みます。



- アジア
 - 日本
 - └ 東京
 - └ 大阪
 - 中国
 - └ 北京
 - └ 上海
 - ベトナム
 - └ ホーチミン
- ヨーロッパ
 - イタリア
 - └ ローマ
 - └ ベネチア
 - ドイツ
 - └ ベルリン
 - └ フランクフルト
- アフリカ
 - └ エジプト

ログイン  MyName
パスワード  ****





That's all. Happy coding!