Thymleaf入門

<https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page>

★本家

<https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf_ja.html>

# Thymeleaf基本編、Web画面を作ってみよう！

**このチャプターの目次**

1. [■SpringBootがサポートするテンプレートエンジン](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%96%A0springboot%E3%81%8C%E3%82%B5%E3%83%9D%E3%83%BC%E3%83%88%E3%81%99%E3%82%8B%E3%83%86%E3%83%B3%E3%83%97%E3%83%AC%E3%83%BC%E3%83%88%E3%82%A8%E3%83%B3%E3%82%B8%E3%83%B3)
2. [■とにもかくにも画面だしてみよっか](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%96%A0%E3%81%A8%E3%81%AB%E3%82%82%E3%81%8B%E3%81%8F%E3%81%AB%E3%82%82%E7%94%BB%E9%9D%A2%E3%81%A0%E3%81%97%E3%81%A6%E3%81%BF%E3%82%88%E3%81%A3%E3%81%8B)
3. [■取り敢えず日付時刻を出すだけの画面をつくる](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%96%A0%E5%8F%96%E3%82%8A%E6%95%A2%E3%81%88%E3%81%9A%E6%97%A5%E4%BB%98%E6%99%82%E5%88%BB%E3%82%92%E5%87%BA%E3%81%99%E3%81%A0%E3%81%91%E3%81%AE%E7%94%BB%E9%9D%A2%E3%82%92%E3%81%A4%E3%81%8F%E3%82%8B)
   1. [＞ What time is it コントローラ の作成](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-what-time-is-it-%E3%82%B3%E3%83%B3%E3%83%88%E3%83%AD%E3%83%BC%E3%83%A9-%E3%81%AE%E4%BD%9C%E6%88%90)
   2. [＞ What time is it 画面HTML の作成](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-what-time-is-it-%E7%94%BB%E9%9D%A2html-%E3%81%AE%E4%BD%9C%E6%88%90)
   3. [＞ What time is it コントローラ の実装](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-what-time-is-it-%E3%82%B3%E3%83%B3%E3%83%88%E3%83%AD%E3%83%BC%E3%83%A9-%E3%81%AE%E5%AE%9F%E8%A3%85)
   4. [＞ What time is it 画面HTML の実装](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-what-time-is-it-%E7%94%BB%E9%9D%A2html-%E3%81%AE%E5%AE%9F%E8%A3%85)
4. [■式言語：SpEL - Spring Expression Language](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%96%A0%E5%BC%8F%E8%A8%80%E8%AA%9E%EF%BC%9Aspel---spring-expression-language)
   1. [＞ EL式（SpEL式）の概要](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-el%E5%BC%8F%EF%BC%88spel%E5%BC%8F%EF%BC%89%E3%81%AE%E6%A6%82%E8%A6%81)
   2. [＞ EL式のいろいろな機能](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-el%E5%BC%8F%E3%81%AE%E3%81%84%E3%82%8D%E3%81%84%E3%82%8D%E3%81%AA%E6%A9%9F%E8%83%BD)
   3. [＞ ４つの式 [変数式／メッセージ式／リンク式／選択変数式]](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-%EF%BC%94%E3%81%A4%E3%81%AE%E5%BC%8F-%5B%E5%A4%89%E6%95%B0%E5%BC%8F%EF%BC%8F%E3%83%A1%E3%83%83%E3%82%BB%E3%83%BC%E3%82%B8%E5%BC%8F%EF%BC%8F%E3%83%AA%E3%83%B3%E3%82%AF%E5%BC%8F%EF%BC%8F%E9%81%B8%E6%8A%9E%E5%A4%89%E6%95%B0%E5%BC%8F%5D)
5. [■画面をイケメンにしてみよう](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%96%A0%E7%94%BB%E9%9D%A2%E3%82%92%E3%82%A4%E3%82%B1%E3%83%A1%E3%83%B3%E3%81%AB%E3%81%97%E3%81%A6%E3%81%BF%E3%82%88%E3%81%86)
   1. [＞ pomにBootstrapの依存関係を追加](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-pom%E3%81%ABbootstrap%E3%81%AE%E4%BE%9D%E5%AD%98%E9%96%A2%E4%BF%82%E3%82%92%E8%BF%BD%E5%8A%A0)
   2. [＞ ViewテンプレートにBootstrapのCSSを適用](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%EF%BC%9E-view%E3%83%86%E3%83%B3%E3%83%97%E3%83%AC%E3%83%BC%E3%83%88%E3%81%ABbootstrap%E3%81%AEcss%E3%82%92%E9%81%A9%E7%94%A8)
6. [★Thymeleaf画面作成の参考ページ★](https://zenn.dev/sugaryo/books/spring-boot-run-up/viewer/thymeleaf_web_page#%E2%98%85thymeleaf%E7%94%BB%E9%9D%A2%E4%BD%9C%E6%88%90%E3%81%AE%E5%8F%82%E8%80%83%E3%83%9A%E3%83%BC%E3%82%B8%E2%98%85)

以下のようにSpringBootで画面を作る場合は Thymeleaf を使用します。

* バックエンドとフロントエンドを分離して RestAPI + React とかの構成にせず、SpringBoot単体で画面まで作っちゃう場合。
* クライアント機能は Vue やら React やらもあるけど、どうせadminしか使わない管理画面はSpringBootの方で済ませる場合。

とか。

# ■SpringBootがサポートするテンプレートエンジン

* SpringBoot のサポートする [テンプレートエンジン](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-spring-mvc-template-engines)
  + Groovy
  + **Thymeleaf ★**
  + FreeMaker
  + Mustache

今回は公式の推奨している Thymeleaf（タイムリーフ：タイムの葉っぱ） を素直に使用します。

* タイムリーフの利点
  + **独自タグを使用しない ( メタ属性を仕込む ) ため、HTMLを汚さない。★**
  + HTMLを汚さないので、そのままブラウザで表示できる。  
    ( サーバ処理なしでも画面表示できる )
  + そのままブラウザで表示できるので、デザイナと実装の親和性が高い。  
    ( と、いわれている )
  + **HTML (view-template) 用 だけでなく、プレーンテキスト用にも使える。 ★**  
    ( いわゆる mail-template のような使い方も出来る )
  + 値をバインドするEL式が比較的わかりやすく、学習コストが低い。  
    ( たぶん。JSFとかやってた人なら「ふーん、なるほどね」って感じで見れると思う。 )
  + [日本語のドキュメント](https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf_ja.html) が充実している。  
    ( 英語が苦手なフレンズでもあんしん！！ )

# ■とにもかくにも画面だしてみよっか

取り敢えずバックエンドのモデル（どころかコントローラも）なしで出来る、まっさらなハリボテ画面だけ出してみましょう。

まずは resource/template/index.html を作成し、内容をこんな感じにします。

index.html

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>SpringBoot - テスト用画面</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta charset="UTF-8" />

</head>

<body>

<h1 th:text="'これはThymeleafですか？'">ｈｔｍｌ－見出し１</h1>

はい、SpringBootで推奨されています。

</body>

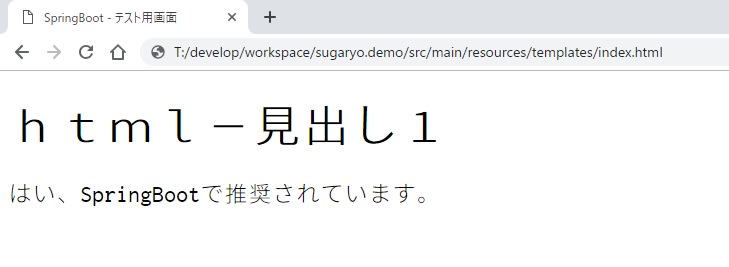
</html>

じゃあ早速これを表示させてみましょう。

デプロイ ( っていうか Run as SpringBootApp ) して http://localhost:8080 にアクセスします。  
すると、こんな画面が出た筈。



次に、 **この resource/template/index.html を直接ブラウザで** 開いてみましょう。  
今度は、こんな画面が出た筈。



<h1 th:text="'これはThymeleafですか？'">ｈｔｍｌ－見出し１</h1>

この部分に違いが出ていますね。

直接ブラウザで **htmlファイル** を表示した場合、th:text という属性は標準HTMLに存在しないなので華麗にスルーされ、普通にタグボディに設定されている ｈｔｍｌ－見出し１ という見出しが表示されます。  
対して、localhost で（つまりサーバ側のテンプレートエンジンを通して）表示した **HTMLレスポンス** では、th:text 属性値が処理されてタグボディに埋め込まれて これはThymeleafですか？ という見出しが表示された。  
という流れです。

**と言ってもこれだけやっても別に何も嬉しくない。**

サーバ側で動的な値を埋め込んだりしてこそ意味があるので、なんとなーくその辺を追加していきましょう。

# ■取り敢えず日付時刻を出すだけの画面をつくる

すげーシンプルに、静的な値ではなく動的な値で、且つインプットを必要としない、最小構成的な物はなにか無いかな。  
と思って考えてみた結果 **「取り敢えず日付時刻でいいんじゃネ？」** と思ったので **日付時刻** を出すだけの画面 what time is it? を作ってみます。

## ＞ What time is it コントローラ の作成

まず、コントローラのガワだけ作っちゃいましょう。

お好きなパッケージ（ぼくの場合は [ProjectRoot].app.controller.view に作りました）にWhatTimeIsItControllerクラス（クラス名は何でも良い）を作り、[@Controller](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Controller.html) をアノテートします。

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@RequestMapping("view/what-time-is-it")

@Controller

public class WhatTimeIsItController {

// まだじっそうしてないよ。

}

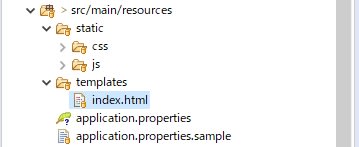
実装内容はまだありません、これから作り込んで行きますが、その前に **View (html)** の方を用意しちゃいましょう。

コントローラの分割レベルとかも議論になるかと思いますが、ここでは（助走本のレベルを超えてるので）華麗にスルーします。

## ＞ What time is it 画面HTML の作成

次に、これから作る画面のHTMLファイルを作成しましょう。

source/main/resources 配下に templates フォルダがあって、現状 index.html が置いてあるかと思います。



実際の業務開発ではこの辺は良い感じにフォルダ構成とか切っていくと思いますが、面倒なのでここでは直置きしちゃいましょう。

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>sugaryo.spring-boot-tips</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta charset="UTF-8" />

<link th:href="@{/css/common.css}" rel="stylesheet"></link>

<script th:src="@{/js/common.js}"></script>

</head>

<body>

<h1 th:text="'■What time is it?■'"></h1>

<div class="border">

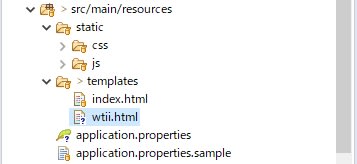
まだなにもないよ。

</div>

</body>

</html>

ハイ、こんな感じ。



それではこれから **コントローラ** で動的な値を生成し、それを埋め込んで **Viewテンプレート** に渡し、 Thymeleaf の機能を使って HTML に埋め込んで **ブラウザの画面に表示** する所までやってみましょう。

このあとやる事はだいたいこんな感じ。

1. コントローラに画面のパス（ @RequestMapping ）を設定。
2. コントローラ（サーバ処理）で **動的な値** を生成。  
   （今回は単純に日付時刻の文字列を作る）
3. SpringMVCのお作法に則って、生成した **動的な値** を設定。
4. 設定した **動的な値** を、ThymeleafのViewテンプレートで参照。  
   （詳細は後述）
5. ブラウザで画面にアクセスすると、サーバ処理で生成した **動的な値** が表示される、、、 **はず！**

## ＞ What time is it コントローラ の実装

まずコントローラのクラスにこんなメソッドを作ります。

@GetMapping()

public String view( Model model ) {

return "wtii"; // ここで返すのはView名、つまり `wtii.html` の "wtii" を返す。

}

コメントに書いた通り、基本的にViewController（ @Controller でアノテートした画面のコントローラ）は、**String（ビュー名）を返すメソッド** を実装します。  
厳密には **ビュー名 ＋ メタ情報** で :forward を指定したりするんですが、取り敢えずここでは **「遷移したい画面のView名を返すと、SpringBoot先生がよしなに画面遷移してくれるんだな」** とだけ思ってください。

ほんで、これだけだとまだ固定の画面を出すだけで、最初に作った index.html と大差ありません。

ので、これから上記メソッドを改造してサーバ側の日付時刻文字列を生成して埋め込む所まで作り込みます。

importの追加

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import org.springframework.ui.Model;

viewメソッドの変更

@GetMapping

public String view( Model model ) { // ①

String now = LocalDateTime.now().format( DateTimeFormatter.ISO\_DATE\_TIME ); // ②

model.addAttribute( "datetime", now ); // ③

return "wtii";

}

**コントローラでやる事はこれだけです。**

以下、変更点に関してなんとなく解説。

* **変更点 ①**

まず、メソッドに org.springframework.ui.Model を受け取るようにパラメータを追加しています。

SpringBoot先生は大変頭が良いので、我々が **「コレを受け取りたいよー」** ってメソッドパラメータに持たせておけば、渡せるものを渡してくれるのです。  
この場合、 Controller が View までデータを渡すために使用する [ui.Model インタフェース](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/ui/Model.html) を受け取るように宣言しています。

* **変更点 ②**

で、サーバ処理として LocalDateTime.now().format(); でサーバ側の現在時刻をもとに日付時刻の文字列を生成しています。

フォーマットはめんどくさいので、定義済みの ISO\_DATE\_TIME にしてますが、好きなフォーマットにして構わないです。  
実際のシステムでは「ログインユーザの設定で変更できる」なんて要件もありそうですよね。

まぁ、ここは本題とはあまり関係ないので値が動的なら何でも良いです、何なら適当に UUID.randomUUID().toString(); とかでも良いです。

* **変更点 ③**

そして model.addAttribute( "datetime", now ); で値を設定しています。

今回、ここでは詳しく説明しませんが [Model](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/ui/Model.html) のほかに [ModelAndView](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/ModelAndView.html) というものがあります。  
上記サンプル実装では **View名の指定** と **Modelへのデータ設定** が別れていますが、これを単一オブジェクトへの操作に出来るのが ModelAndView です。

では、これで **Controller** の改造は終わったので、次に **View** の方に手を入れましょう。

## ＞ What time is it 画面HTML の実装

**View** の方はもっと簡単です。

コントローラで生成したデータの表示

<body>

<h1 th:text="'■What time is it?■'"></h1>

<div class="border">

<p th:text="${datetime}"><p>

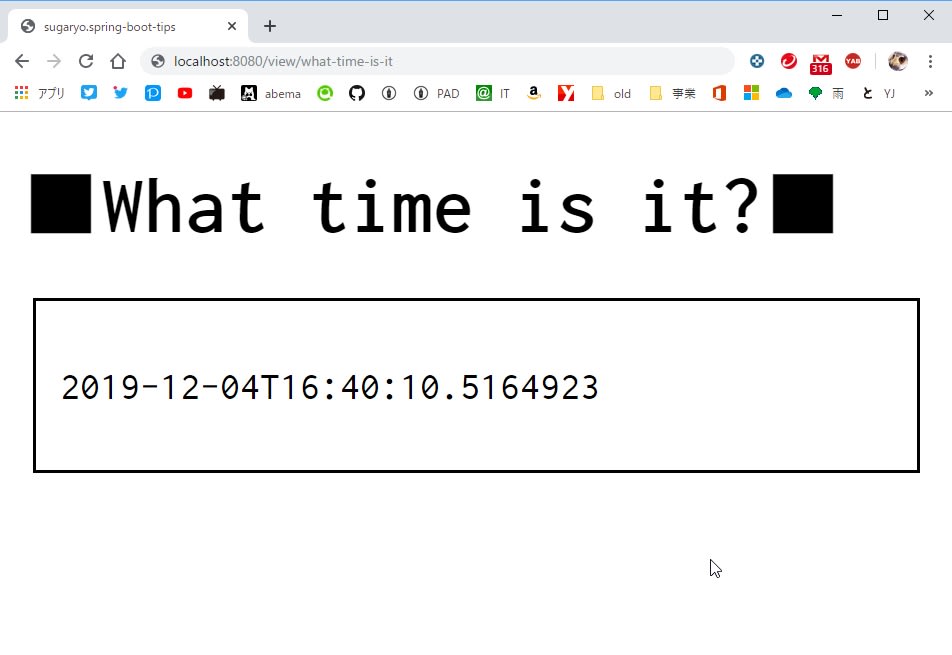
</div>

</body>

これだけ。

追加したのは <p th:text="${datetime}"><p> の部分。  
タグは <p> にしたけど、別に今は表示出来れば何でも良いです。

説明の前に、動きを見てみた方がテンション上がると思うので、前回の index.html の時同様、SpringBootApplicationを起動したら http://localhost:8080/view/what-time-is-it にアクセスして実際にブラウザで見てみましょう。



ハイ、Viewテンプレートの <p th:text="${datetime}"><p> 部分に、見事サーバ側の日付時刻 (つってもlocalhostだけどｗ) が表示されましたね。

* **軽く解説**

ここで指定してる datetime というのは、さっきコントローラで model.addAttribute( "datetime", now ); した時の "datetime" をキーに指定して、中身の値を引っ張り出してる訳です。  
普通のJavaのコードで Map.get( "datetime" ); って書いたようなもんです。

要するに **Contoroller** で Map.put( "datetime", now ); したものを **View** が Map.get("datetime"); して取り出すようなイメージ。  
コードの字面が model.addAttribute( "datetime", now ) と ${datetime} に変わっただけです。

* **EL式**

さて、この Map.get みたいなことをしている ${} コイツは一体何者なんでしょうか？

コイツは **式言語** 、別名 **EL式** と呼ばれるものです。

# ■式言語：SpEL - Spring Expression Language

## ＞ EL式（SpEL式）の概要

* **概要**

SpringBoot + Thymeleaf での開発では、データモデル model からビュー view にデータを渡す（データバインドする）には **SpEL式という仕組み** を利用します。

JavaEE(JSF) 時代に ManagedBean から xhtml にデータバインドするのに **EL式** を触ってた人なら馴染み深いかと思います、 **あんな感じのアレ** です。

EL式自体はかなり昔からある一般的な技術で、Spring固有のものではないです。

**Sp**ringの**EL式** として拡張実装されたものなので、正式名称としては **「SpEL式」** というようですが、名前なんてここでは重要じゃないので **「EL式」** で良いでしょう。

SpEL式というのかい！？

* **EL式 is 何？**

EL式を使った経験のない人向けにざっくり説明すると、さっきの [model.addAttribute](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/ui/Model.html" \l "addAttribute-java.lang.String-java.lang.Object-) で仕込んでおいたデータを、ビューであるHTMLに埋め込む時に使ったのが **EL式** です。

この **EL式のルールに則って記述した部分** は、HTMLのビューテンプレート上で **特別な意味を持つ** ようになり、テンプレートエンジン ( 今回の場合は Thymeleafさん ) によって処理されます。

* **さっきの What time is it での具体例**

具体的には <p th:text="${datetime}"><p> の部分ですね。

${datetime} この部分がEL式になっており、datetime という識別子（キー）のデータに置き換えて貰う、という記述になっています。  
つまり、model.addAttribute( "datetime", now ) で与えたデータ（この場合は 2019-12-04T16:40:10.5164923 という文字列ですね）が使用されます。

そして、その置き換えたデータが th:text というThmeleafの独自属性に与えられるので、テンプレートエンジンがHTMLをレンダリングする時にこのデータを使用して <p>タグ のテキスト要素として埋め込まれた訳です。

要するに **EL式** とは **Thymeleaf先生とお話するための言語** ですね。

## ＞ EL式のいろいろな機能

**EL式** の最もシンプルで基本的な使い方が、先程の例のような **「addAttributeで埋め込んでおいたデータを取り出す」** でしょう。

勿論EL式には他にも様々な機能があります。  
**条件判定** ができたり、 Math クラスなどの **staticメソッドを利用** できたり、他にもリクエストパラメータparamやセッションsessionや下記ユーティリティなど **よく使うオブジェクトがデフォルトで提供** されていたりします。

* [Thymeleafチートシート#ユーティリティオブジェクト - Qiita](https://qiita.com/NagaokaKenichi/items/c6d1b76090ef5ef39482#%E3%83%A6%E3%83%BC%E3%83%86%E3%82%A3%E3%83%AA%E3%83%86%E3%82%A3%E3%82%AA%E3%83%96%E3%82%B8%E3%82%A7%E3%82%AF%E3%83%88)
  + #strings
  + #numbers
  + #bools
  + #dates
  + #calendars
  + #objects
  + #arrays
  + #lists
  + #maps
  + #sets

これだけ色々と揃っていれば、バックエンドのロジック無しでEL式だけでも、多少の機能なら作れちゃいますね。

## ＞ ４つの式 [変数式／メッセージ式／リンク式／選択変数式]

ちなみに、先程説明した ${datetime} というEL式ですが、これはThymeleafに於ける式四天王 **「標準式構文」** のひとり 変数式 というものです。

* 標準式構文
  + ${変数式}
  + \*{選択変数式}
  + #{メッセージ式}
  + @{リンク式}

それぞれざっくり説明すると、こんなかんじ。

* **変数式**

既に説明したとおり、埋め込んでおいた変数を参照する式。

この辺を更に詳しく知りたい人は OGNL とかのワードでググると良いでしょう、ここでは省略します。

* **選択変数式**

th:object や th:field と合わせて使用する、オブジェクトのショートカットポインタみたいな式。

例えば th:object="$(hoge)" として hoge をオブジェクトに仕込んでいるタグ内で、 \*{id} とすると ${hoge.id} 相当、\*{name} とすると ${hoge.name} 相当の記述になるようです。

なんかあれですね、VBの WITHステートメント みたいなアレですね。

* **メッセージ式**

これは割と解かりやすい、プロパティファイルの外部定義値をバインドするのに使用するものです。

これが「メッセージ式」という名前になってるのは、一般的に多言語対応とかで message.properties がよくいるから、ですかね？

* **リンク式**

超便利なのがこのリンク式。

これはコンテキストパスを補完して良い感じにURLを記述できるスグレモノです。

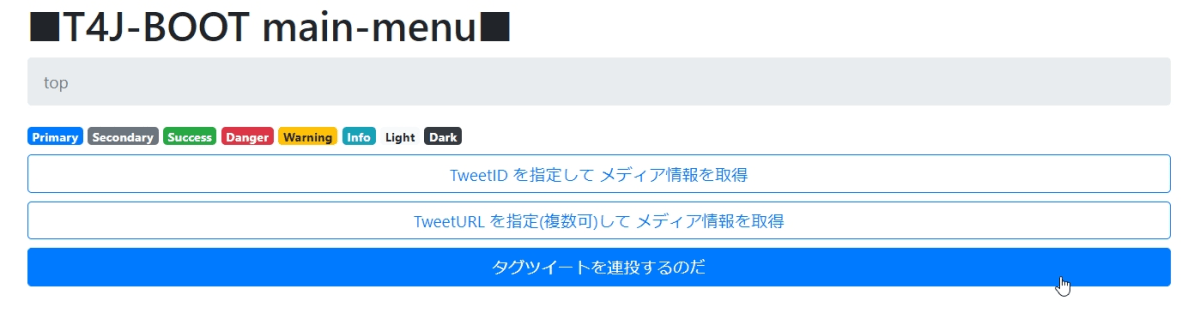
何より便利なのが、パス変数を使って **動的なURLを表現する事も出来る** ということ。

例えば @{/resource/{id}/detail(id=${resourceId})} みたいな事ができます。

この記述を細かく見ていくと /resource/{id}/detail という相対パスがあり、このパスの中には {id} という動的パス要素が含まれており、そのidには ${resourceId} の変数式で参照した値を使う、、、と言う感じです。

# ■画面をイケメンにしてみよう

これは趣味ツールの画面なんですけど。





What time is it? 画面はお化粧なしのHTMLすっぴん画面でしたが、Thymeleaf に [Bootstrap](https://getbootstrap.jp/) を組み合わせれば簡単にイケメンは作れます。

## ＞ pomにBootstrapの依存関係を追加

pom.xml

<dependency>

<groupId>org.webjars</groupId>

<artifactId>bootstrap</artifactId>

<version>4.5.2</version>

</dependency>

## ＞ ViewテンプレートにBootstrapのCSSを適用

<link rel="stylesheet" th:href="@{/webjars/bootstrap/4.5.2/css/bootstrap.min.css}">

後は普通にBootstrapを利用すれば良き。

ちょっとした管理画面とかなら Thymeleaf + Bootstrap で十分だと思うんだよなぁ。

# ★Thymeleaf画面作成の参考ページ★

* **EL/SpEL式 関連**
  + [SpringFramework公式（英語）](https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/html/index.html)
    - [EL式](https://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/html/expressions.html)
  + [Spring Boot で Thymeleaf 使い方メモ - Qiita](https://qiita.com/opengl-8080/items/eb3bf3b5301bae398cc2)
    - [SpEL 式](https://qiita.com/opengl-8080/items/eb3bf3b5301bae398cc2#spel-%E5%BC%8F)
  + [Thymeleafチートシート - Qiita](https://qiita.com/NagaokaKenichi/items/c6d1b76090ef5ef39482)
  + [必要最小限のサンプルでThymeleafを完全マスター - Instructor's memo](http://masatoshitada.hatenadiary.jp/entry/2016/12/11/100528)
    - ４つの記法
  + [Thymeleafの標準式構文 - Java好き](https://javazuki.com/articles/thymeleaf-standard-expression-syntax.html)
  + [Spring BootでThymeleafを使ってみよう！（応用編）#記述方法 - Marinroad](https://www.marineroad.com/staff-blog/17761.html#i-2)
* **Thymeleaf＋Bootstrap 関連**
  + [Bootstrap](https://getbootstrap.jp/)
  + [SpringBootアプリにBootstrap4を追加（WebJars使用） - One IT Thing](https://one-it-thing.com/2074/)
  + [SpringBoot入門 vol.13：Bootstrapでスタイルを整えよう - プログラミング逆引き辞典](https://learning-collection.com/springboot%E5%85%A5%E9%96%80-vol-13%EF%BC%9Abootstrap%E3%81%A7%E3%82%B9%E3%82%BF%E3%82%A4%E3%83%AB%E3%82%92%E6%95%B4%E3%81%88%E3%82%88%E3%81%86/)
  + [Spring Boot ThymeleafにBootstrapを追加 - ITSakura](https://itsakura.com/springboot-bootstrap)

目標：

1. HTML範囲の基本タグのThymleaf化できる
2. Server側から渡されたデータを画面に表示できる（メッセージも含む）