**JavaScript開発ガイド**

2015年7月23日版

ガイド目次

[JavaScript開発ガイド 1](#_Toc427567563)

[1 JavaScript開発ガイド 4](#_Toc427567564)

[2 レスポンシブデザイン 4](#_Toc427567565)

[3 多言語化対応 4](#_Toc427567566)

[4 入力と連動した即時処理 5](#_Toc427567567)

[4.1. 概要 5](#_Toc427567568)

[4.2. 仕組 5](#_Toc427567569)

[4.3. 実装例 5](#_Toc427567570)

[4.4. SPAでの実装例 6](#_Toc427567571)

[5 データ取得時のLoading表示 8](#_Toc427567572)

[5.1. 概要 8](#_Toc427567573)

[5.2. 仕組 8](#_Toc427567574)

[5.3. 実装例 8](#_Toc427567575)

[5.4. ユーティリティ化 10](#_Toc427567576)

[6 データ送信(登録/更新/削除)時のProgress表示 12](#_Toc427567577)

[6.1. 概要 12](#_Toc427567578)

[6.2. 仕組 12](#_Toc427567579)

[6.3. 実装例 12](#_Toc427567580)

[7 モーダル表示 15](#_Toc427567581)

[8 タブ表示 15](#_Toc427567582)

[9 マウスオーバーによる吹き出し表示 15](#_Toc427567583)

[10 グラフ表示 15](#_Toc427567584)

[11 Bootstrapに合わせたプルダウン表示 15](#_Toc427567585)

[12 カレンダー表示 15](#_Toc427567586)

[13 画像を組み合わせたリスト表示 15](#_Toc427567587)

[14 カメラ起動 16](#_Toc427567588)

[14.1. 概要 16](#_Toc427567589)

[14.2. 仕組 16](#_Toc427567590)

[14.3. 実装例 16](#_Toc427567591)

[15 カルーセル表示 20](#_Toc427567592)

[16 要素のドラッグによるソート 21](#_Toc427567593)

[16.1. 概要 21](#_Toc427567594)

[16.2. 仕組 21](#_Toc427567595)

[16.3. 実装例 21](#_Toc427567596)

[16.4. モバイル対応策 22](#_Toc427567597)

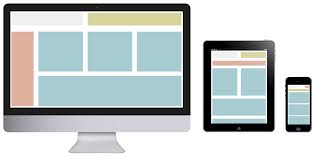
# はじめに

この章ではJavaScript開発ガイドについて説明する。

# レスポンシブWebデザイン

概要

### レスポンシブWebデザイン / Responsive Web Design / RWD 概念



図表 2‑1　Responsive Web Design

レスポンシブデザインとは、Webデザインの手法の一つで、様々な種類の機器や画面サイズに単一のファイルで対応すること。

WebサイトやWebページをパソコンやスマートフォン、タブレット端末など複数の機器や画面サイズに対応させる際、内容が同じでデザインや動作をそれぞれに最適化した複数のHTMLファイルやCSSファイルを用意し、HTTPリダイレクトなどでアクセスを振り分ける手法がよく用いられる。

レスポンシブデザインではこのような対応はせず、表示された機器の種類やサイズに応じて表示内容が最適な状態に変化するよう設定された単一のファイルを制作し、すべての機器に同じように送信する。

### レスポンシブWEBデザインのメリット

1. サイト制作・運営が楽

複数デバイスへ向けて、個々にサイトを構築する場合に比べ、制作工数が少なくなりや更新がかんたんになる。また、デバイスが増えた際も動作確認以外の特別な対応が必要ない。

1. ブランドイメージを伝えやすい

どの端末でも同一デザインを元に表示することができるからである。異なる端末でユーザーがアクセスした場合でも、ページ遷移や表示内容がほぼ同一なため、ユーザビリティの向上も見込める。

1. WebサイトやSNSでシェアされた際、複数端末からスムーズに閲覧できる

サイトがfacebookやtwitterでシェアされることも増え、リンクをPCで開くこともあればスマホからアクセスすることもあるよね？つまり、どの端末からでも見やすいサイトが求められている。

1. サーバーの特別な設定がいらない

デバイスごとにサイトを作る場合、ユーザーエージェントを判別してリダイレクト処理を行うのが一般的だが、レスポンシブWEBデザインではその処理が必要ないため自由度の低いレンタルサーバ等でも使うことができる。

### レスポンシブWEBデザインのデメリット

1. レスポンシブWEBデザイン特有の知識が必要

2010年にアメリカで生まれた比較的新しい技術なので、デザインにおける制約や注意する点など、特有のノウハウがなくてはいけない。

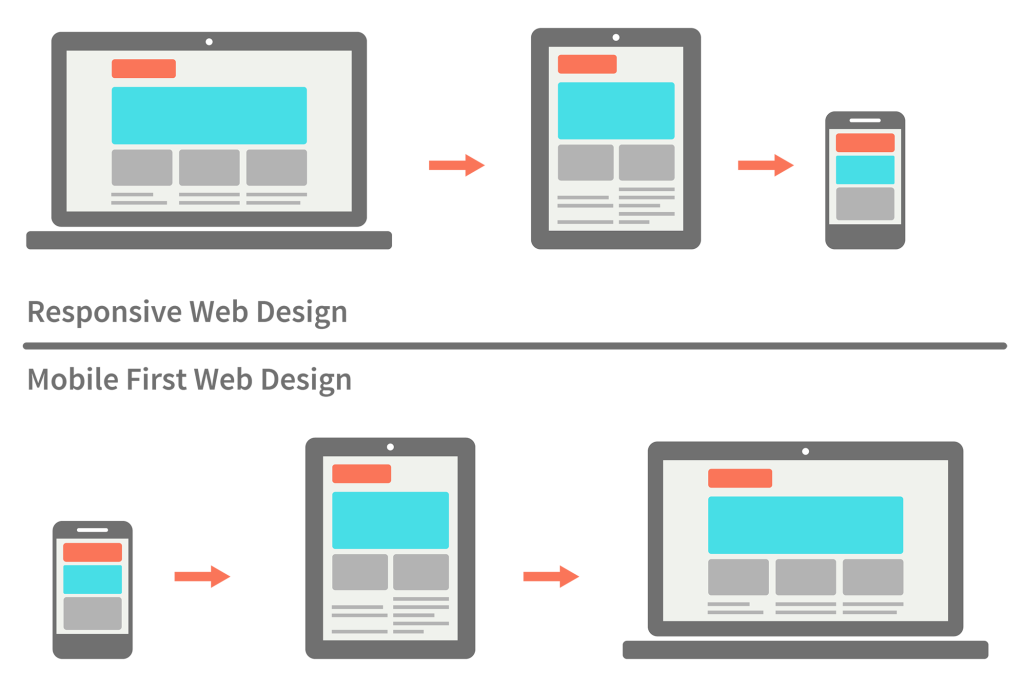
1. デバイス単位でコンテンツを出し分けることができない

あらゆるデバイスで同一のソースコードを使うため、CSSによる表示制御はできるがソースコードを出し分けることができない。そのため、タブレットやスマートフォンなどの回線速度が比較的遅い端末では表示に時間がかかることがある。

1. レイアウトの自由度が低い

上記の理由により、レイアウトの自由度は低くなる。

### モバイルファースト（Mobile First）



図表 2‑2　Mobile First Web Design

モバイルファーストとは、WebサイトやWebサービス、ソフトウェア開発において、**パソコン版よりも先にスマートフォンやタブレットなどのモバイル端末に対応した開発を行う手法、**および概念（コンセプト）を意味する。通常ならパソコン版を先に開発し、スマートフォンなどのモバイル端末に最適化を行うが、スマートフォンとタブレットの普及に伴い、モバイル端末への最適化が重視されるようになっている。

モバイルファーストはアップル、マイクロソフト、日本マクドナルドなどが導入しており、モバイル市場をターゲットにした戦略につなげている。スマートフォンやタブレットでショッピングをしたりチケットを購入しやすくなっており、ユーザーがモバイル端末で快適に操作できるようにUIを工夫するなど、モバイルファーストを導入するWebサイトやWebサービスが増えている。

スマートフォンはデスクトップパソコンと比べて画面が小さく掲載できるコンテンツ量や情報量が制限されるため、ユーザーが本当に必要な情報を精査したり、ユーザビリティを向上する必要がある。また、通信速度もパソコンと比べて落ちるため、パフォーマンスも考慮するなど、モバイル端末に対応した開発が求められる。

仕組

レスポンシブWebデザインは、「フルードグリッド（Fluid Grid）」「フルードイメージ（Fluid Image）」「メディアクエリー（Media Query）」の3つの技術的な要素で構成される。

●フルードグリッド

　フルードグリッドは、Webページの要素を罫線や升目に沿って配置する「グリッドデザイン（Grid Design）」と、ブラウザーの横幅が変わってもレイアウトを維持したまま要素のサイズを調整する「フルードデザイン（Fluid Design）」を合わせたものである。レスポンシブWebデザインでは、最初にグリッドデザインによって部品や表示領域をpx単位で配置していき、レイアウトが整った後に、値を％に変換してフルードデザインに変更する。

●フルードイメージ

　フルードイメージは、レイアウトの大きさに追随して画像のサイズを拡大・縮小する手法で、CSSのみで実装できる。イギリスのコンサルティングファームClearleftのリチャード・ルター（Richard Rutter）氏によって提唱される。

●メディアクエリー

　メディアクエリーは、画像解像度、ウィンドウの幅、デバイスの向きなどの指定条件にあわせて別々のCSSを適用する技術である。レスポンシブWebデザインではメディアクエリーを使ってスクリーンサイズに応じたCSSに切り替える。

　このほかにも、文字をレイアウトの大きさに追随して伸縮する「レスポンシブタイプセッティング（Responsive Typesetting）」や、テーブル（表組）のサイズを調整する「レスポンシブテーブル（Responsive Table）」などの技術があり、必要に応じて利用していく。

簡単な実装例を作りながら、基本となる3つの技術を使ったレスポンシブWebデザインの基礎を説明する。

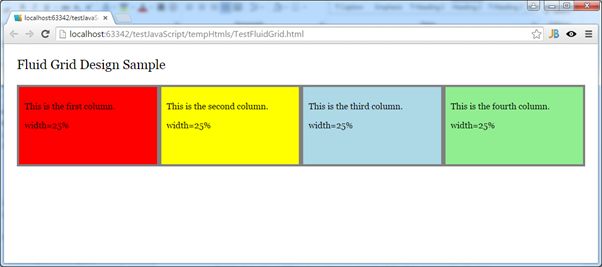
実装例

### フルードグリッド

■サンプルとしては、ソースは下記のようになります。

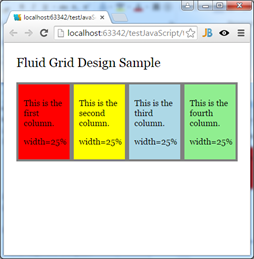
|  |
| --- |
| <!DOCTYPE> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**style type="text/css"**>  **\***{  **font-family**: **georgia**, **serif**;  }  .**title**{  **font-size**: 24**px**;  **color**: **black**;  }  .**container**{  **margin**:10**px** 15**px**;  }  ***/\*グリッドでもコラムでも％のように設定することは必須です。\*/*** .**col**{  **box-sizing**: **border-box**;  **float**: **left**;  **width**: 25%;　  **height**: 150**px**;  **border**: 4**px solid grey**;  **padding**: 10**px**;  }  **#col\_1**{  **background-color**: **red**;  }  **#col\_2**{  **background-color**: **yellow**;  }  **#col\_3**{  **background-color**: **lightblue**;  }  **#col\_4**{  **background-color**: **lightgreen**;  }  </**style**> </**head**> <**body**> <**div class="container"**>  <**p class="title"**>Fluid Grid Design Sample</**p**>  <**div id="col\_1" class="col"**>  <**p**>This is the first column.</**p**>  <**p**>width=25%</**p**>  </**div**>  <**div id="col\_2" class="col"**>  <**p**>This is the second column.</**p**>  <**p**>width=25%</**p**>  </**div**>  <**div id="col\_3" class="col"**>  <**p**>This is the third column.</**p**>  <**p**>width=25%</**p**>  </**div**>  <**div id="col\_4" class="col"**>  <**p**>This is the fourth column.</**p**>  <**p**>width=25%</**p**>  </**div**> </**div**> </**body**> </**html**> |

■実行結果



図表 2‑3　Fluid Gridの表示結果

ウィンドウの幅はみじかくなるように調整しているうちに、結果は下記のようになります。



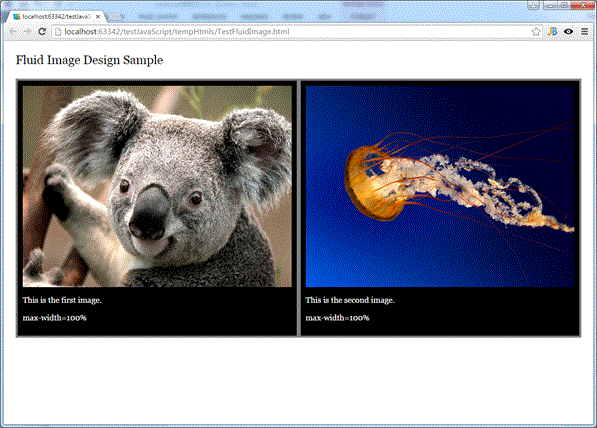
図表 2‑4　短くなる場合、Fluid Gridの表示結果

### フルードイメージ

■サンプルとしては、ソースは下記のようになります。

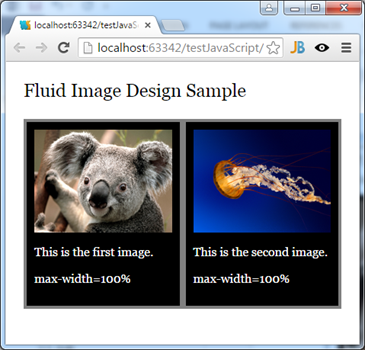
|  |
| --- |
| <!DOCTYPE> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**style type="text/css"**>  **\***{  **font-family**: **georgia**, **serif**;  }  .**title**{  **font-size**: 24**px**;  **color**: **black**;  }  .**container**{  **margin**:10**px** 15**px**;  }  .**col**{  **background-color**: **black**;  **box-sizing**: **border-box**;  **float**: **left**;  **width**: 50%;  **border**: 4**px solid grey**;  **padding**: 10**px**;  }  .**col p**{  **color**: **#fff**;  }  ***/\*imgではmax-width:100%を設定することは必須です。 \*/***  **img**{  **max-width**: 100%;  }  </**style**> </**head**> <**body**> <**div class="container"**>  <**p class="title"**>Fluid Image Design Sample</**p**>  <**div id="col\_1" class="col"**>  <**img src="../lib/image/Koala.jpg"**>  <**p**>This is the first image.</**p**>  <**p**>max-width=100%</**p**>  </**div**>  <**div id="col\_2" class="col"**>  <**img src="../lib/image/Jellyfish.jpg"**>  <**p**>This is the second image.</**p**>  <**p**>max-width=100%</**p**>  </**div**> </**div**> </**body**> </**html**> |

■実行結果



図表 2‑5　Fluid Imageの表示結果

ブラウザーのが大小はみじかくなるように調整しているうちに、結果は下記のようになります。



図表 2‑6　短くなる場合、Fluid Gridの表示結果

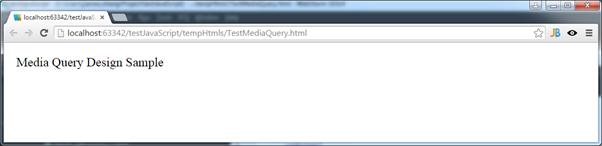
### メディアクエリー

■サンプルとしては、ソースは下記のようになります。

|  |
| --- |
| <!DOCTYPE> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**style type="text/css"**>  .**container**{  **margin**:10**px** 15**px**;  **font-size**: 24**px**;  }  ***/\*ウィンドウの幅は701px以上の場合、黒字で表示\*/***  .**title**{  **color**: **black**;  }  ***/\*ウィンドウの幅は(401px～700px)となる場合、赤字で表示\*/*****@media screen and** (**min-width**: 401**px**) **and** (**max-width**: 700**px**){  .**title**{  **color**: **red**;  }  }  ***/\*ウィンドウの幅は(0px～400px)となる場合、青字で表示\*/*****@media screen and** (**max-width**: 400**px**){  .**title**{  **color**: **blue**;  }  }  </**style**> </**head**> <**body**> <**div class="container"**>  <**p class="title"**>Media Query Design Sample</**p**> </**div**> </**body**> </**html**> |

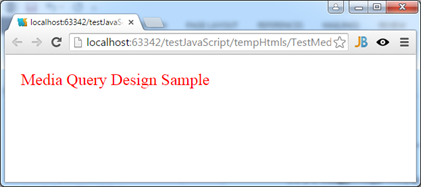
■実行結果

ウィンドウの幅は(701px～∞)となる場合、



図表 2‑7　大ウィンドウの場合

ウィンドウの幅は(401px～700px)となる場合、



図表 2‑8　中ウィンドウの場合

ウィンドウの幅は(0～400px)となる場合、



図表 2‑9　小ウィンドウの場合

### レスポンシブWebデザイン

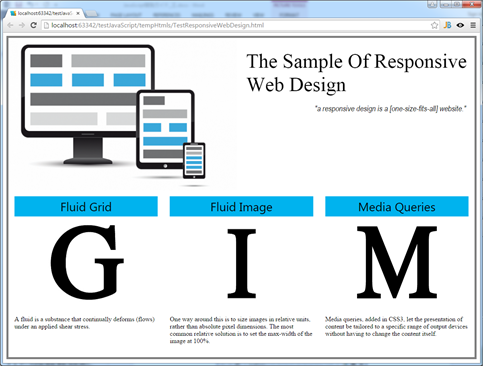
上記2.3.1～2.3.3で紹介された技術的な要素を纏めてレスポンシブWebデザインのサンプルを作成する。

■サンプルとしては、ソースは下記のようになります。

|  |
| --- |
| <!DOCTYPE> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  　　<**meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"**>  <**style type="text/css"**>  **\***{  **box-sizing**: **border-box**;  }  .**container**{  **border**: 5**px solid grey**;  }  **img**{  **max-width**: 100%;  }  .**clearFixed**{  **clear**: **both**;  }  .**col\_lg\_1\_in\_2**, .**col\_lg\_1\_in\_3**, .**col\_sm\_1\_in\_1**{  **float**: **left**;  **position**: **relative**;  }  .**col\_lg\_1\_in\_2**{  **width**: 49%;  }  .**col\_lg\_1\_in\_3**{  **width**: 33.333%;  }   .**headImg** {  **float**: **left**;  }  .**headWords** {  **float**: **right**;  }  .**headTitle**{  **font-size**: 50**px**;  **margin-top**: 5%;  }  .**headContent**{  **width**: 70%;  **font-size**: 20**px**;  **float**: **right**;  **font-style**: **italic** ;  **font-family**:**"Arial Narrow"**;  }  .**elementsTitle**{  **background-color**: **#00b3ee**;  **font-family**: **"Segoe UI"**;  **font-size**: 30**px**;  **padding**: 5**px**;  }  .**elements** .**grid**{  **text-align**: **center**;  **vertical-align**: **top**;  **padding**: 15**px**;  }  .**elements p**{  **text-align**: **left**;  }   */\*PCで中ウィンドウの場合\*/* **@media screen and** (**max-width**: 1024**px**){  .**headTitle**{  **font-size**: 30**px**;  }  }  */\*モバイルの場合\*/* **@media screen and** (**max-width**: 767**px**){  .**headTitle**{  **font-size**: 40**px**;  **padding**: 5**px** 10**px**;  }  .**headContent**{  **width**: 70%;  **font-size**: 20**px**;  **float**: **right**;  }  .**col\_sm\_1\_in\_1**{  **width**: 100%;  }  .**headImg** {  **float**: **right**;  }  .**headWords** {  **float**: **left**;  }  }  </**style**> </**head**> <**body**> <**div class="container"**>  <**div class="heads"**>  <**div class="headWords col\_lg\_1\_in\_2 col\_sm\_1\_in\_1 "**>  <**div class="headTitle"**>The Sample Of Responsive Web Design</**div**>  <**div class="headContent"**><**p**>"a responsive design is a [one-size-fits-all] website."</**p**></**div**>  </**div**>  <**div class="headImg col\_lg\_1\_in\_2 col\_sm\_1\_in\_1"**>  <**img src="../lib/image/responsive-web-design.png"**>  </**div**>  <**div class="clearFixed"**></**div**>  </**div**>   <**div class="elements"**>  <**div class="grid col\_lg\_1\_in\_3 col\_sm\_1\_in\_1"**>  <**div class="elementsTitle"**>Fluid Grid</**div**>  <**img src="../lib/image/G.png"**>  <**p**>A fluid is a substance that continually deforms (flows) under an applied shear stress.</**p**>  </**div**>  <**div class="grid col\_lg\_1\_in\_3 col\_sm\_1\_in\_1"**>  <**div class="elementsTitle"**>Fluid Image</**div**>  <**img src="../lib/image/I.png"**>  <**p**>One way around this is to size images in relative units, rather than absolute pixel dimensions. The most common relative solution is to set the max-width of the image at 100%.</**p**>  </**div**>  <**div class="grid col\_lg\_1\_in\_3 col\_sm\_1\_in\_1"**>  <**div class="elementsTitle"**>Media Queries</**div**>  <**img src="../lib/image/M.png"**>  <**p**>Media queries, added in CSS3, let the presentation of content be tailored to a specific range of output devices without having to change the content itself.</**p**>  </**div**>  <**div class="clearFixed"**></**div**>  </**div**>  <**div class="clearFixed"**></**div**> </**div**> </**body**> </**html**> |

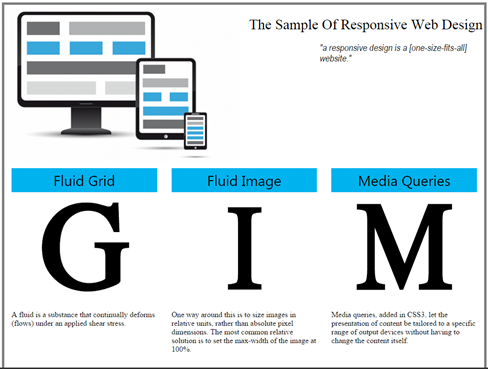
■実行結果

パソコンの場合、



図表 2‑8　PCの場合

Ipadの場合、



図表 2‑9　 Ipadの場合

※Iphone6の場合、



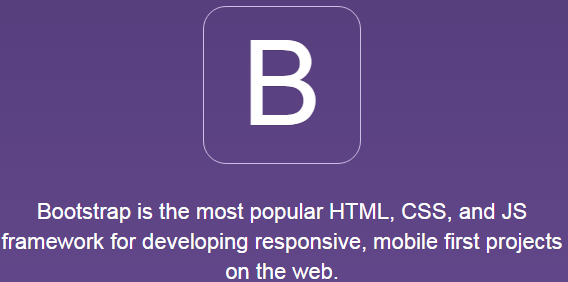
図表 2‑10　モバイルの場合

※モバイルの場合、下記どおり、device-widthはWidthとしてHTMLに設定する必要がある。

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

レスポンシブWebデザイン対応のフレームワーク

### Bootstrap3



図表 2‑3　Bootstrap3

■特徴

1. モバイルファーストのアプローチ
2. 12カラムグリッドシステム
3. 豊富なプラグイン・拡張ライブラリ

■参考サイト

Bootstrap英語版版（<http://getbootstrap.com/>）

Bootstrap 非公式日本語版（<http://www.wivern.com/bootstrap/index.html>）

### Responsive Grid System



図表 2‑4　 Responsive Grid System

■特徴

1. 100行以下の軽量サイズ
2. モバイルファースト志向の設計
3. 1～12のグリッドに対応

■参考サイト

<http://www.responsivegridsystem.com/>

### Foundation 5



図表 2‑5　Foundation 5

■特徴

1. モバイルファースト
2. レスポンシブサイト対応のグリッドシステム
3. 豊富に用意されたCSSやコンポーネントやJavaScript
4. 標準で用意されたオフキャンバスメニュー

■参考サイト

<http://foundation.zurb.com/>

# 多言語化対応

概要

SPAの多言語化対応はi18nextというライブラリを使って国際化を行うことです。これはブラウザ側でも、node.jsのサーバー側でも使える素敵なライブラリです。

仕組

i18nextライブラリを利用することで、言語ごとにプログラムの外部にJsonファイルを分けることができる。各Jsonファイルにデータを保存することです。

データの取得では、i18next自体に提供された関数でJsonファイルからKeyより表示内容のValue取得することです。

データの表示では、JQueryやHandlebarsなどを用いて様々な形式で表現することができることです。

* 初期化のオプションについて

i18nextの初期化に関する常用オプションについて、下表で説明する。

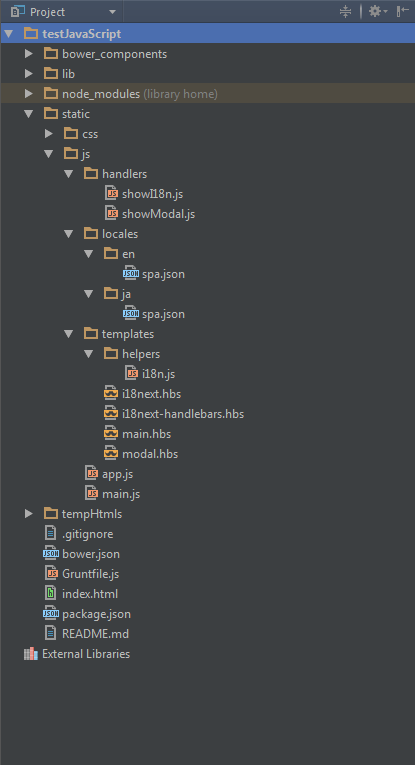
|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **型** | **デフォルト** | **説明** |
| resGetPath | String | “locales/\_\_lng\_\_/\_\_ns\_\_.json” | 国際化用ファイルパス |
| lowerCaseLng | Boolean | false | システムから取込された言語種類は小文字にする |
| ns | Object | ns: { namespaces: ['translation'], defaultNs: 'translation' } | 指定した場合、resGetPathに\_\_ns\_\_は「PageA」になる、デフォルト値は「translation」です。 |
| getAsync | boolean | true | Jsonファイルの取得は同期/非同期にする。　falseの場合、同期、trueの場合、非同期。 |
| debug | boolean | false | エラーが発生した場合、エラーログを出力する。 |
| fallbackLng | Array | ['dev'] | システムから言語種類が取得できない場合など、デフォルト言語を設定する。 |

参考資料：<http://i18next.com/>

実装例

### プロジェクトの構造

サンポルとして、RequireJS +Bootstrap + Jquery+ Handlerbarでのプロジェクトプを作成する。プロジェクトの構造は以下にのようになります。

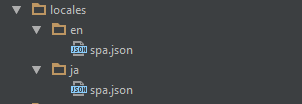


図表 3‑1　プロジェクトの構造

### ローカライズファイルの用意。

* フォルダ体系

今回は日本語/英語の二種類に対応します。ローカライズ用のデータは、localesという名前のフォルダの下に言語ごとに分けて入れている。



図表 3‑2　ローカライズ様データのフォルダー体系

* spa.json

en/jaそれぞれのフォルダに、[json](http://d.hatena.ne.jp/keyword/json)ファイルで[ローカライズ](http://d.hatena.ne.jp/keyword/%A5%ED%A1%BC%A5%AB%A5%E9%A5%A4%A5%BA)に使用するテキストを配置します。

enのフォルダには、以下のようなファイルを置きます。

|  |
| --- |
| {  **"label"** : {  **"title"** : **"Attention!"**,  **"content"** : **"Hello World!"** } } |

jaフォルダのspa.[json](http://d.hatena.ne.jp/keyword/json)では以下のように、日本語で文字列を書いておきます。

|  |
| --- |
| {  **"label"**: {  **"title"**: **"ご注意"**,  **"content"**: **"こんにちは！"** } } |

### ソースコード(Template and Helpersを利用する場合)

* index.js

RequireJS + Handlebarsを利用したことで、index画面に<div id="mainRegion"></div>しか存在しないことをご注意。

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="UTF-8"**>  <**meta http-equiv="X-UA-Compatible" content="IE=edge"**>  <**meta name="viewport" content="width=device-width, initial-scale=1"**>  <**meta name="author" content="OptimusZhang"**>  <**title**>testJavaScript</**title**>  <**link rel="stylesheet" href="lib/css/bootstrap/bootstrap.css"**/>  <**link rel="stylesheet" href="static/css/testStyle.css"**/>  <**script src="lib/js/requirejs/require.js"  data-main="static/js/main"**></**script**> </**head**> <**body**> <**div id="mainRegion"**></**div**>  </**body**> </**html**> |

* main.js

RequireJSに基づいて、必要なJSファイルを配置した上で、i18nextの初期化処理を入れた。

初期化処理に必要なオプションを設定した。

|  |
| --- |
| ***requirejs***.config({  **baseUrl**: **'./static/js/'**,  **paths**: {  **'jquery'**: **'../../lib/js/jquery/jquery'**,  **'bootstrap'**: **'../../lib/js/bootstrap/bootstrap'**,  **'hbs'** : **'../../lib/js/require-handlebars-plugin/hbs'**,  **'i18next'**: **'../../lib/js/i18next/i18next'** },   **shim**:{  ***// i18nextは「AMD」ではないので、Jqueryと依頼関係を追加する。*****'i18next'**:{  **deps**: [**'jquery'**],  **exports**: **'i18n'** },  **'bootstrap'**:{  **deps**: [**'jquery'**]  }  },   ***// RequireJSでHandlebars.jsを使うプラグイン「require-handlebars-plugin」を配置する。*****hbs**:{  helperPathCallback: **function**(name) {**return 'templates/helpers/'** + name;}  } });  ***requirejs***([**'jquery'**, **'app'**,**'bootstrap'**, **'i18next'**], **function** ($, app) {   ***// i18nextを初期化する。***$.i18n.init(  ***// i18nのoptionsを設定する。*** {  ***// 国際化用ファイルパスを指定する。*****resGetPath**: **'static/js/locales/\_\_lng\_\_/\_\_ns\_\_.json'**,  ***// システムから取込された言語種類は小文字にする。*****lowerCaseLng**: **true**,  */****/ 指定した場合、resGetPathに\_\_ns\_\_は「PageA」になる、デフォルト値は「translation」です。*** **ns**: **'spa'**,  ***// Jsonファイルの取得は同期/非同期にする。　falseの場合、同期、trueの場合、非同期。*****getAsync**: **false**,  ***// エラーが発生した場合、エラーログを出力する。*****debug**: **true**,  ***// システムから言語種類が取得できない場合など、デフォルト言語を設定する。*****fallbackLng**: **'ja'** }, **function** () {  ***//ファイルのロードが終わった後、したいものを入れて。  // 今回は使わないので、何もいれない。***});   app.start(); }); |

* app.js

appにて、画面を表示する処理を入れた。

また、「言語切替」ボタンより画面の再表示処理があるため、それもapp.jsに入れた。

|  |
| --- |
| define([**'hbs!templates/main'**,  **'handlers/showModal'**,  **'handlers/showI18n'**],  **function** (mainTmpl, showModal, showI18n) {   **return** {  start : **function**() { ***$***(**'#mainRegion'**).html(mainTmpl());   ***// モーダルエリアを表示する。***showModal.show();  ***// 多言語化エリアを表示する。***showI18n.show();   ***// bind the app to the window*** **window**.**app** = **this**;  },   ***// 英語へボタンをクリックした場合、システム言語をenに設定する。***i18n\_ChangeLng\_En: **function** () {  ***//システム言語をenに設定する。******$***.i18n.setLng(**'en'**, **function** () {  });  ***// 多言語化エリアを再表示する。***showI18n.show();  },   ***// 日本語へボタンをクリックした場合、システム言語をjaに設定する。***i18n\_ChangeLng\_Ja: **function** () {  ***//システム言語をjaに設定する。******$***.i18n.setLng(**'ja'**, **function** () {  });  ***// 多言語化エリアを再表示する。***showI18n.show();  }  };  }); |

* showI18n.js

|  |
| --- |
| define([**'hbs!templates/i18next'**], **function**(i18nTmpl){   **return** {  ***//TODO i18nの使い例  // init the i18n part.***show : **function**(){   ***// htmlをロードする。******$***(**'#i18nextRegion'**).html(i18nTmpl());   ***// システム言語を設定する。******$***(**'#systemLang'**).html(***$***.i18n.lng());   ***// 英語へボタンにクリックイベントをバインドする。******$***(**"#btnChangeToEng"**)[0].onclick = **function** (event) {  app.i18n\_ChangeLng\_En();  };   ***// 日本語へボタンにクリックイベントをバインドする。******$***(**"#btnChangeToJp"**)[0].onclick = **function** (event) {  app.i18n\_ChangeLng\_Ja();  };  }  } }); |

* main.hbs

App.jsでindex.htmlの<div id="mainRegion"></div>に入れたものです。

|  |
| --- |
| <**nav class="navbar navbar-inverse navbar-fixed-top"**>  <**div class="container"**>  <**div class="navbar-header"**>  <**a class="navbar-brand" href="#"**>  <**span class="glyphicon glyphicon-hand-right" aria-hidden="true"**></**span**>  For Bootstrap Test!  </**a**>  </**div**>  </**div**> </**nav**>  <**div class="container-fluid"**>  <**div class="row"**>  <**div class="col-md-3 zjw\_nav"**>  <**ul class="nav nav-sidebar"**>  <**li role="presentation" class="active"**><**a href="#modalRegion" class="list-group-item text-capitalize"**>モーダル(Modal)</**a**>  </**li**>  <**li role="presentation"**><**a href="#i18nextRegion" class="list-group-item"**>国際化(i18n)</**a**></**li**>  <**li role="presentation"**><**a href="#333" class="list-group-item"**>333</**a**></**li**>  <**li role="presentation"**><**a href="#444" class="list-group-item"**>444</**a**></**li**>  <**li role="presentation"**><**a href="#555" class="list-group-item"**>555</**a**></**li**>  </**ul**>  </**div**>  <**div class="col-md-9 main sidebar"**>  ***<!-- モーダルエリアを表示 -->***  <**div id="modalRegion" class="zjw\_module"**></**div**>  ***<!--多言語エリアを表示 -->***  <**div id="i18nextRegion" class="zjw\_module"**></**div**>  <**div id="333" class="zjw\_module"**>  ***…略して表示***  　　</**div**>  </**div**>  </**div**> </**div**> |

* i18next.hbs

多言語エリアに対するソースコード。

{{i18n “label.title”}}のようなものを利用したことで、Jsonファイルで記載されているものを画面に転記できる。

|  |
| --- |
| <**div class="panel panel-primary"**>  <**div class="panel-heading"**>  <**h3 class="panel-title zjw\_fontBold"**><**span class="glyphicon glyphicon-screenshot"**></**span**>  国際化(i18n)</**h3**>  </**div**>  <**div class="panel-body"**>  <**div class="row"**>  <**div class="col-md-6"**>  <**div class="panel panel-default"**>  <**div class="panel-heading"**>  <**h3 class="panel-title"**>英語(en/spa.json)</**h3**>  </**div**>  <**div class="panel-body"**>  <**pre class="bg-info" style="margin**:0 0**"**>{  "label" : {  "title" : "Attention!",  "content" : "Hello World!"  } }</**pre**>  </**div**>  </**div**>  </**div**>  <**div class="col-md-6"**>  <**div class="panel panel-default"**>  <**div class="panel-heading"**>  <**h3 class="panel-title"**>日本語(jp/spa.json)</**h3**>  </**div**>  <**div class="panel-body"**>  <**pre class="bg-info" style="margin**:0 0**"**>{  "label" : {  "title" : "ご注意",  "content" : "こんにちは！"  } }</**pre**>  </**div**>  </**div**>  </**div**>  </**div**>  <**div class="row"**>  <**div class="col-md-12"**>  <**table class="table table-bordered"**>  <**tr**>  <**th class="col-md-2 bg-info"**>Title</**th**>  ***<!-- handlebarsのHelperで作ったi18n.jsで多言語を実現する-->***  <**td class="col-md-4" id="i18nTitle"**>**{{i18n "label.title"}}**</**td**>  <**th class="col-md-2 bg-info"**>現在言語</**th**>  <**td class="col-md-4" id="systemLang"**></**td**>  </**tr**>  <**tr**>  <**th class="col-md-2 bg-info"**>Content</**th**>  ***<!-- handlebarsのHelperで作ったi18n.jsで多言語を実現する-->***  <**td class="col-md-4" id="i18nContent"**>**{{i18n "label.content"}}**</**td**>  <**th class="col-md-2 bg-info"**>言語切替</**th**>  <**td class="col-md-4" style="padding**: 2**px** 5**px**;**"**>  <**button type="button" class="btn btn-warning" id="btnChangeToEng"**>英語へ（en）</**button**>  <**button type="button" class="btn btn-warning" id="btnChangeToJp"**>日本語へ（ja）</**button**>  </**td**>  </**tr**>  </**table**>  </**div**>  </**div**>  </**div**> </**div**> |

* i18n.js

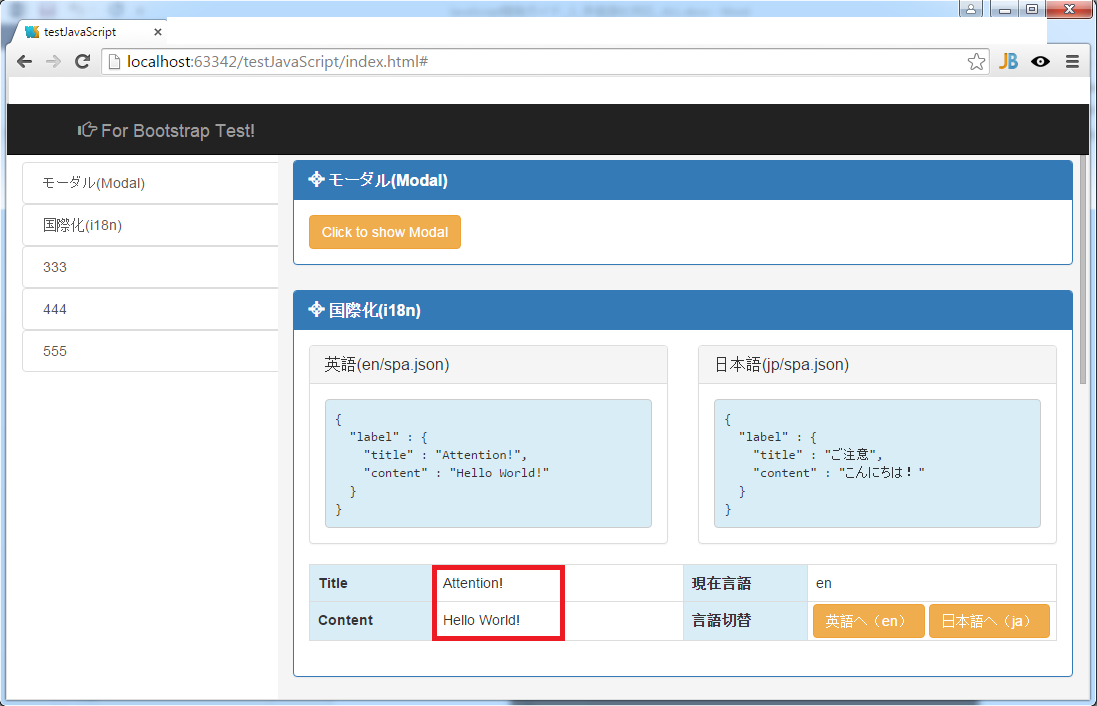
Handlebars.jsのhelperでi18nextの「$.i18n.t(key)」を利用した上で、Jsonファイルから表現したいものを取得する。

※i18n.jsはmain.jsで配置された「require-handlebars-plugin」によって呼ばれることです。

|  |
| --- |
| ***define***([**'hbs/handlebars'**],**function**(Handlebars) {  ***// \*.hbsに直接に利用できる関数「i18n」を定義する。***  Handlebars.registerHelper(**'i18n'**, **function**(key) {  ***// JsonファイルからKey,valueの形で、表現したいものを取得する。***  **return new** Handlebars.**SafeString**(***$***.i18n.t(key));  }); }); |

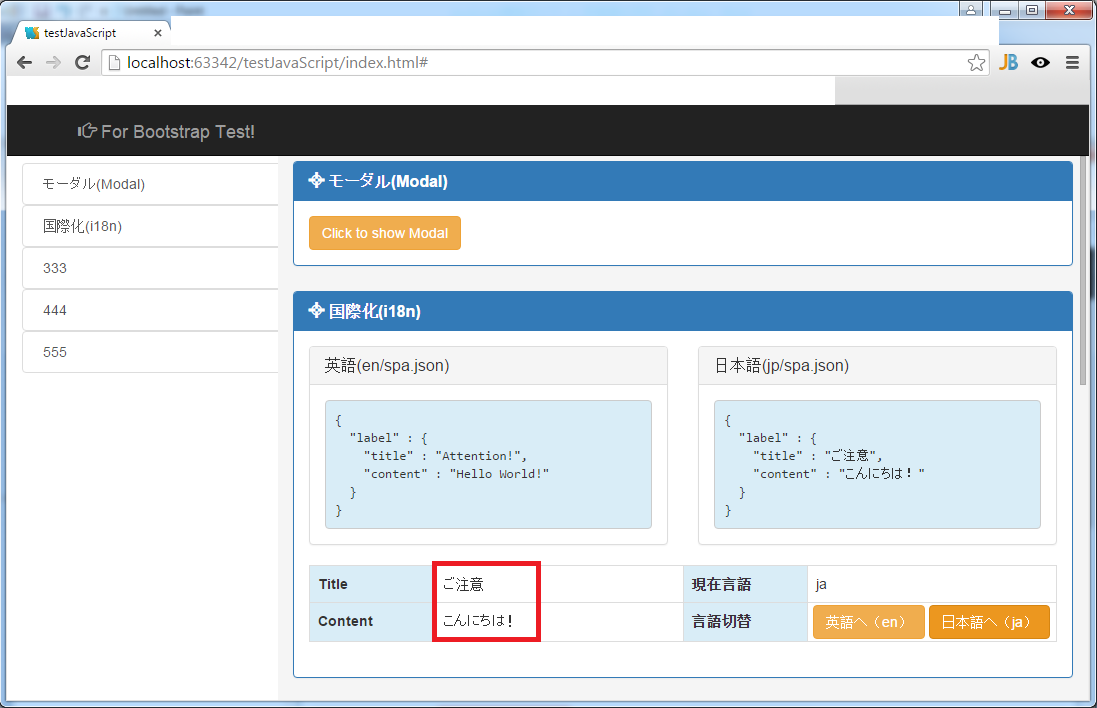
* 実行結果

初期表示の場合、



図表 3-3　システム言語が英語の場合、画面の初期化

「日本語へ」ボタンを押下して、



図表 3-4　システム言語を日本語へ切り替えた場合

### ソースコード(Jquery Integrationを利用する場合)

※Template&Helpersと比べて、差異点だけを記載している。

* i18next.hbs

{{i18n　”label.title”}}を利用せずに、data-i18n=”label.title”のように書き直す。

|  |
| --- |
| ***・・・略して表示*** <**table class="table table-bordered"**>  <**tr**>  <**th class="col-md-2 bg-info"**>Title</**th**>  <**td class="col-md-4" id="i18nTitle" data-i18n="label.title"**></**td**>  <**th class="col-md-2 bg-info"**>現在言語</**th**>  <**td class="col-md-4" id="systemLang"**></**td**>  </**tr**>  <**tr**>  <**th class="col-md-2 bg-info"**>Content</**th**>  <**td class="col-md-4" id="i18nContent" data-i18n="label.content"**></**td**>  <**th class="col-md-2 bg-info"**>言語切替</**th**>  <**td class="col-md-4" style="padding**: 2**px** 5**px**;**"**>  <**button type="button" class="btn btn-warning" id="btnChangeToEng"**>英語へ（en）</**button**>  <**button type="button" class="btn btn-warning" id="btnChangeToJp"**>日本語へ（ja）</**button**>  </**td**>  </**tr**> </**table**>  ***・・・略して表示*** |

* showI18n.js

$('hoge’).i18n();のように処理を追加する必要。

|  |
| --- |
| ***define***([**'hbs!templates/i18next'**], **function**(i18nTmpl){   **return** {  *//TODO i18nの使い例**// init the i18n part.* show : **function**(){   *// htmlをロードする。* ***$***(**'#i18nextRegion'**).html(i18nTmpl());  ***// これが差異点です。***  ***$***(**'#i18nextRegion'**).i18n();   *// システム言語を設定する。* ***$***(**'#systemLang'**).html(***$***.i18n.lng());   *// 英語へボタンにクリックイベントをバインドする。* ***$***(**"#btnChangeToEng"**)[0].onclick = **function** (event) {  app.i18n\_ChangeLng\_En();  };   *// 日本語へボタンにクリックイベントをバインドする。* ***$***(**"#btnChangeToJp"**)[0].onclick = **function** (event) {  app.i18n\_ChangeLng\_Ja();  };  }  } }); |

* 実行結果

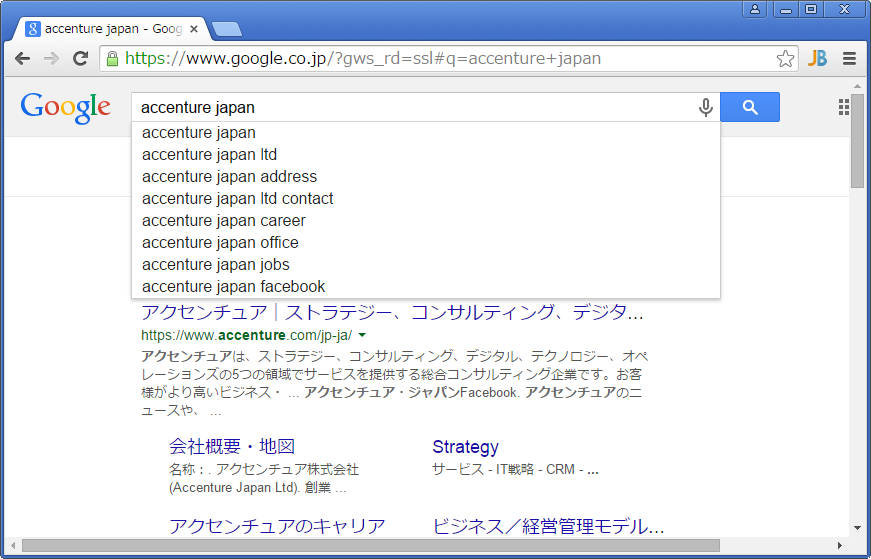
実行結果は3.3.3の実行結果と同じです。

# 入力と連動した即時処理

概要

通常WEBアプリケーションの処理は、ボタン/リンククリックの操作で処理が実行される。

しかし近年、ボタン/リンククリックの操作ではなく、テキストボックスの入力、マウス操作などと連動して、処理が実行されるアプリケーションが増えてきた。



検索条件を入力しながら、自動検索してくれる

図表 4‑1　Google検索で見られる入力と連動した即時処理

ここでは、検索などでよく使われるテキストボックスへの入力と連動した即時処理を説明する。

なお、入力を検知する部分をマウス操作やフォーカスイン/アウトなどの操作の検知に変更することで応用が可能となる。

仕組

jQueryのkeyupイベントで入力操作を監視して、定義したJavaScript関数をを呼出す。setTimeout/clearTimeoutを利用して検索の連続通信を防いでいる。

実装例

・jQueryを利用した実装例を説明する。（入力間隔は0.5秒以上の場合、検索が行われる例）

|  |
| --- |
| **// 検索タイマー定義**  **var** searchTimer;  **// input[type=text]のセレクタで取得したオブジェクトに対して、keyupイベントを監視する。**  **$**(**"input[type=text]"**).keyup(**function** () {  **// 検索タイマーを解除する。**  clearTimeout(searchTimer);  **// 検索タイマーを設定する。（500の単位はミリ秒であり、適宜直すのは可能である。）**  searchTimer = setTimeout(*search*, 500); });  **// サーバーにアクセスして、検索結果を取得するなど。** **function** *search*() {  } |

keyupについて <https://api.jquery.com/keyup/>をご参考ください。

・keyupの代わりに下記のイベントも利用できる。

|  |  |
| --- | --- |
| イベント名 | イベント説明 |
| blur | <https://api.jquery.com/blur/> |
| change | <https://api.jquery.com/change/> |
| focusin | <http://api.jquery.com/focusin/> |
| focusout | <https://api.jquery.com/focusout/> |
| mouseenter | <http://api.jquery.com/mouseenter/> |
| mouseleave | <http://api.jquery.com/mouseleave/> |

※ほかのイベントについては<https://api.jquery.com/category/events/>をご覧ください。

上記のサイトは英語で、日本語の説明は<http://js.studio-kingdom.com/jquery/events>をご覧ください。

SPAでの実装例

* 入力イベントを定義する。

|  |
| --- |
| **return** Marionette.**ItemView**.extend({  **events**: {  **// input[type=text]のセレクタで取得したオブジェクトに対して、keyupイベントを監視する。**  **'keyup input[type=text]'**: **'setIntervalForSearch'** },  setIntervalForSearch: **function** () {  **// 一定間隔で検索関数を実行するように、ViewUtilを利用する。**  ViewUtil.setIntervalForFunctionCall(**this**, **this**.search.bind(**this**));  },  **// サーバーにアクセスして、検索結果を取得するなど。**  search: **function** () {  }  }); |

* ViewUtil.jsの「一定間隔で指定関数を実行する」ファンクション  
  (<https://innersource.accenture.com/afpjshowcase/afpjshowcase/source/afpjshowcase-front/static/js/aicj/util/ViewUtil.js>)

|  |
| --- |
| **return** {  **// 一定間隔で指定関数を実行する。**  **// @param {view} viewObject – 任意Marionette View**  **// @param {function} func - 実行関数**  setIntervalForFunctionCall: **function** (viewObject, func) {  **// 関数を呼出すタイマーを解除する。**  **if** (viewObject.**\_searchTimer**) {  clearTimeout(viewObject.**\_searchTimer**);  }  **// 検索タイマーを設定する。（500の単位はミリ秒であり、適宜直すのは可能である。）**  viewObject.**\_searchTimer** = setTimeout(func, 500);  }  }; |

# データ取得時のLoading表示

概要

Ajaxでサーバーと通信中にクルクル画像が表示されるのは多くて、そのクルクル画像はスピナーといいます。

今回はspin.jsライブラリを利用して、スピナーの実装方法を説明する。



図表 5‑1　スピナーイメージ

spin.jsを選ぶ理由は以下である。

1. 画像なし、外部CSSファイルは不要、spin.jsファイルだけ使用する。
2. jQuery等の外部ライブラリは不要である。
3. スピナー表示位置/色/サイズなどは設定できる。

<http://fgnass.github.io/spin.js/>にて、spin.jsライブラリはダウンロードできる。

仕組

spin.jsを用いて、SpinnerをNewして、指定DOMエレメントにスピナーの表示・非表示を制御できる。

なお、表示位置/色などの設定も提供されている。

実装例

■スピナー表示

|  |
| --- |
| **// スピナー定義**  **var** spinner = **new** *Spinner*();  **// スピナーを渡したDOMエレメントに表示** spinner.spin(**document**.getElementById(**"foo"**)); **// 普通の書き方**  **OR**  spinner.spin(**$**(**"#foo"**).get(0)); **// JQueryの書き方** |

・生成されるスピナーは実際にclass=”spinner”のdivタグである。DOMは以下の形になる。

|  |
| --- |
| **// スピナー表示前**    **// スピナー表示後** |

■スピナー非表示

|  |
| --- |
| **// スピナーを消す**  spinner.stop(); |

■スピナー表示のデフォルトオプションは下記となり、スピナー定義する時、パラメータとして渡して、上書きできる。

|  |
| --- |
| **var *defaults*** = {  **lines**: 12, *// The number of lines to draw* **length**: 7, *// The length of each line* **width**: 5, *// The line thickness* **radius**: 10, *// The radius of the inner circle* **scale**: 1.0, *// Scales overall size of the spinner* **rotate**: 0, *// Rotation offset* **corners**: 1, *// Roundness (0..1)* **color**: **'#000'**, *// #rgb or #rrggbb* **direction**: 1, *// 1: clockwise, -1: counterclockwise* **speed**: 1, *// Rounds per second* **trail**: 100, *// Afterglow percentage* **opacity**: 1/4, *// Opacity of the lines* **fps**: 20, *// Frames per second when using setTimeout()* **zIndex**: 2e9, *// Use a high z-index by default* **className**: **'spinner'**, *// CSS class to assign to the element* **top**: **'50%'**, *// center vertically* **left**: **'50%'**, *// center horizontally* **position**: **'absolute'** *// element position* }; |

たとえば、スピナー位置を変えたい場合、

|  |
| --- |
| ・デフォルトはDOMエレメントの中心に表示される。    ・上中央に表示したい場合、**new** *Spinner*()のパラメータに{**top**: **"0%"**}を渡す。    ・下中央に表示したい場合、**new** *Spinner*()のパラメータに{**top**: **"100%"**}を渡す。 |

{**color**: **"#337ab7"**}のように渡して、スピナー色を変える。ほかのデフォルトオプションも同じように変えられる。

ユーティリティ化

■spin.jsのインポートは省略されている。

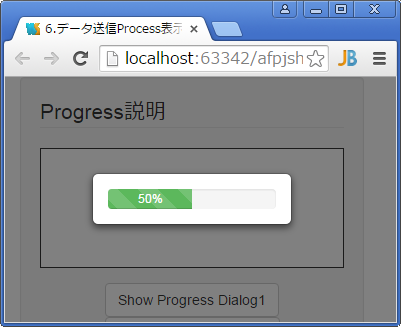
|  |
| --- |
| **// スピナー表示関数と非表示関数は用意されている。  // 表示関数は何回も呼ばれる場合、一つスピナーだけ表示される。 // 非表示関数はスピナー表示された回数までに呼ばれた場合、非表示になる。 // つまり、もし表示関数は２回呼ばれたら、非表示関数は１回だけ呼ばれて、 // スピナーは表示のままで、２回呼ばれたら、非表示になる。**  **// @param {HTMLElement} targetElement – 任意DOMエレメント（div spanなど）**  **// @param {object} opts – オプション（スピナー位置/サイズ/色など）**  **var** *AjaxSpinner* = **function** (targetElement, opts) {   **var** cnt = 0;  **var** spinner = **new** *Spinner*(opts);   **// スピナー表示関数**  **this**.showSpinner = **function** () {  **// 表示対象がHtmlElement以外の場合はエラー****if** (!(targetElement **instanceof HTMLElement**)) {  **console**.error(**"spin target is not a HTMLElement."**);  **return**;  }   cnt = cnt + 1;  **if** (cnt === 1) {  spinner.spin(targetElement);  }  };   **// スピナー非表示関数**  **this**.stopSpinner = **function** () {  **if** (cnt > 0) {  cnt = cnt - 1;  **if** (cnt === 0) {  spinner.stop();  }  }  }; }; |

# データ送信(登録/更新/削除)時のProgress表示

概要

Ajaxでデータ送信(登録/更新/削除)時に、ユーザーの操作を防ぐために、画面は半透明のレイヤーでブロックされて、処理中などのメッセージが表示されることはよく見える。

今回はbootstrapのモーダルとプログレスバーを利用して、おしゃれな通信進捗の表示方法を説明する。



図表 6‑1　通信プログレス表示イメージ

■参照ライブラリ

・bootstrap-dialog

<https://github.com/nakupanda/bootstrap3-dialog/releases>よりダウンロードできる。

Bootstrapのモーダルに依存しているので、Bootstrapライブラリも必要になる。

・Bootstrap

<http://getbootstrap.com/getting-started/#download>よりダウンロードできる。

仕組

bootstrap-dialogライブラリを用いて、モーダルダイアログを表示させ、ダイアログの中にプログレスバーを表示させる。

プログレスのパーセント表示について、モーダルが開く際に、プログレスが50%で表示され、閉じる際に、100%で表示される。つまり、Ajax通信開始に50%で表示され、Ajax通信完了に100%で表示されることを想定している。

実装例

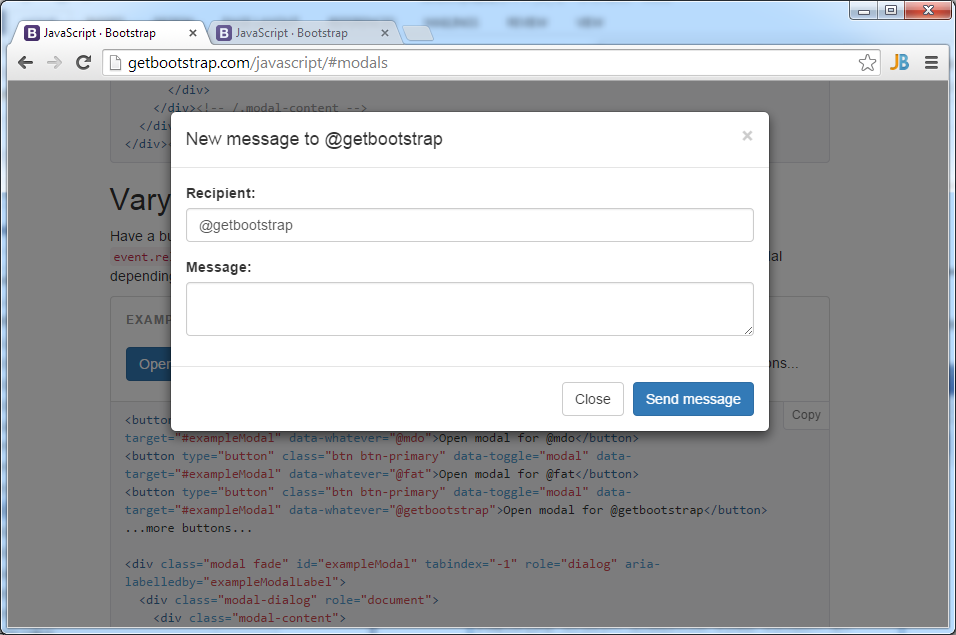
■プログレスダイアログの実装例（bootstrap-dialogとbootstrapのインポートは省略されている）

|  |
| --- |
| **$**(**function** () {  **var** dialog;   **// プログレスダイアログを開く  function** *openProgressDialog*() {  **// BootstrapDialogインスタンス生成** dialog = **new BootstrapDialog**({  **id**: **'saveProgressDialog'** });   dialog.realize(); **// モーダル初期化** dialog.getModalHeader().**hide**(); **// ヘッダーを非表示にする** dialog.getModalFooter().**hide**(); **// フッターを非表示にする** dialog.setClosable(**false**); **// ユーザーは手動でクローズできないようにする   // プログレスダイアログを画面中心に表示する** dialog.getModal().css(**"top"**, **" 35%"**);  dialog.getModalDialog().css(**"width"**, **"200px"**);  dialog.getModalDialog().css(**"margin"**, **"10px auto"**);   **// プログレスバーDiv定義  var** progressBar =  **$**(**'<div class="progress" style="margin-bottom: 0px;">'** +  **'<div id="dialogProgressBar" class="progress-bar progress-bar-success progress-bar-striped active" role="progressbar" aria-valuenow="60" aria-valuemin="0" aria-valuemax="100" style="width: 0%;">'** +  **'0%'** +  **'</div>'** +  **'</div>'**);   **// プログレスバーをモーダルボディに追加する** dialog.getModalBody().append(progressBar);   **// モーダルが可視状態にされた時に、呼び出される** dialog.**options**.onshown = **function** (dialogRef) {  **// プログレスバー  var** $progressBar = dialogRef.getModalBody().find(**"#dialogProgressBar"**);   **// モーダルのClose関数は先に進めた場合に、100%のままで表示する  if** ($progressBar.text() !== **'100%'**) {  **// プログレスを50%に設定する** $progressBar.**width**(**"50%"**);  $progressBar.text(**"50%"**);  }  };   dialog.open();  }   **// プログレスダイアログを閉じる  function** *closeProgressDialog*() {  **// プログレスバー  var** $progressBar = dialog.getModalBody().find(**"#dialogProgressBar"**);  **// プログレスを100%に設定する** $progressBar.**width**(**"100%"**);  $progressBar.text(**"100%"**);  **// プログレスバーのアニメーション効果を無効にする** $progressBar.removeClass(**'active'**);   **// 100%の表示はユーザーに見せるため、0.5秒後にクローズする** setTimeout(**function** () {  dialog.close();  }, 500);  }   **// プログレスダイアログのテスト** (**function** () {  **// ダイアログを開く（Ajax通信開始イベントの時）** *openProgressDialog*();   **// Ajaxの通信時間の模擬** setTimeout(**function** () {  **// ダイアログを閉じる（Ajax通信完了イベントの時）** *closeProgressDialog*();  }, 2000);  })(); }); |

# モーダル表示

概要

モーダルは、最小限必要な機能とスマートさをデフォルトに備えた、効率化されてはいても柔軟なダイアログ・プロンプトです。

以下に図示する。

図表 7‑1　getbootstrap.comから提供されたモーダルです。

仕組

　　モーダル・プラグインはデータ属性や JavaScript 経由で必要に応じて隠れたコンテンツを切り替えます。これはまた、<body> に .modal-open を追加することで、デフォルトのスクロールの振る舞いを上書きして、モーダルの外側がクリックされたときに見せていたモーダル閉じるためのクリック領域を提供するために、 .modal-backdrop を生成します。

* オプションについて

　　データ属性か JavaScript 経由でオプションを指定します。データ属性には、data-backdrop="" のように、data- にオプション名をつけます。

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **型** | **デフォルト** | **説明** |
| backdrop | ブール値か 'static' 文字列 | true | モーダルのバックドロップをインクルードします。あるいは、クリックでモーダルを閉じないバックドロップには、static を明記します。 |
| keyboard | ブール値 | true | エスケープ・キーが押されたときにモーダルを閉じます。 |
| show | ブール値 | true | 初期化時にモーダルを表示します。 |
| remote | パス | false | このオプションは v3.3.0 以来不評で、v4 ではなくなります。その代わりにクライアント・サイド・テンプレートかデータ・バインディング・フレームワークを使うか、自分で jQuery.load を呼び出すことを推奨します。  リモート URL がある場合は、jQuery の load メソッドによってコンテンツは一度にロードされ、.modal-content div に送り込まれます。データ API を使っているなら、代わりに href 属性を使ってリモート・ソースを指定してもかまいません。この例を下にお見せします：  <a data-toggle="modal" href="remote.html" data-target="#modal">Click me</a> |

* メソッドについて

|  |  |  |
| --- | --- | --- |
| **名称** | **説明** | **使い方** |
| .modal(options) | コンテンツをモーダルとして動作させます。オプションとして object を受け付けます。 | $('#myModal').modal({  keyboard: false,  ……  }) |
| .modal('toggle') | 手動でモーダルを切り替えます。モーダルが実際に開いたり閉じたりする前に呼び出し元に帰ります。(例： shown.bs.modal や hidden.bs.modal イベントが起きる前) | $('#myModal').modal('toggle') |
| .modal('show') | 手動でモーダルを開きます。モーダルが実際に開く前に呼び出し元に帰ります。(例： shown.bs.modal イベントが起きる前) | $('#myModal').modal('show') |
| .modal('hide') | 手動でモーダルを閉じます。モーダルが実際に閉じる前に呼び出し元に帰ります。(例： hidden.bs.modal イベントが起きる前) | $('#myModal').modal('hide') |
| .modal('handleUpdate') | モーダルが開くときに、モーダルが左に行くようにスクロール・バーとの位置関係を再調整します。  モーダルが開いているときにモーダルの高さを変える場合にだけ必要になります。 | $('#myModal').modal('handleUpdate') |

* イベントについて

Bootstrap のモーダル・クラスは、モーダルの機能にフックする少数のイベントを公開しています。

全てのモーダル・イベントは、モーダル自身に起動されます(例： <div class="modal"> )。

|  |  |
| --- | --- |
| **イベント型** | **説明** |
| show.bs.modal | このイベントは、show インスタンス・メソッドが呼ばれたときすぐに起動されます。クリックによって起動された場合は、クリックされた要素はイベントの relatedTarget 属性として利用できます。 |
| shown.bs.modal | このイベントは、モーダルが可視状態にされたときに(CSS の切り替えが完了するのを待って)起動されます。クリックによって起動された場合は、クリックされた要素はイベントのrelatedTarget 属性として利用できます。 |
| hide.bs.modal | このイベントは、hide インスタンス・メソッドが呼ばれたときすぐに起動されます。 |
| hidden.bs.modal | このイベントは、モーダルが閉じ終わったときに(CSS の切り替えが完了するのを待って)起動されます。 |
| loaded.bs.modal | このイベントは、モーダルが remote オプションを使っているコンテンツをロードしたときに起動されます。 |

　　イベントの使い方：

|  |
| --- |
| $('#myModal').on('hidden.bs.modal', function (e) {  // do something...  }) |

実装例

### ボタンをクリクするとHTMLのデータ属性でオプションを指定してモーダルを切り替える例。

* ボタンのソースコード（html）

|  |
| --- |
| **// 呼出元のボタンに、data-toggleとdata-targetを指定する**  <**button type="button" class="btn btn-primary" data-toggle="modal" data-target="#hogeModal"**>Click to show  Modal </**button**> |

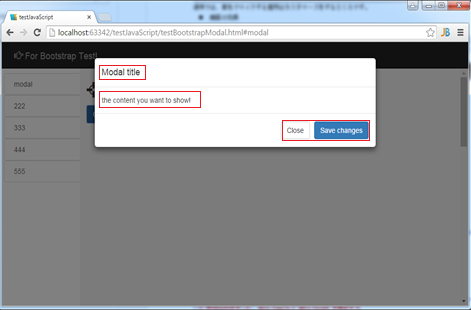
* モーダルのソースコード（html）

|  |
| --- |
| **// 呼出先のモーダルはhtmlのbodyの直下に置く。**  *<!-- Modal -->*  **// idは上記ボタンのdata-targetと同値（hogeModal）を指定する。**<**div class="modal fade" id="hogeModal"**>  <**div class="modal-dialog" role="document"**>  <**div class="modal-content"**>   <**div class="modal-header"**>  <**h4 class="modal-title" id="hogeModelLabel"**>Modal title</**h4**>  </**div**>   <**div class="modal-body"**>  the content you want to show!  </**div**>   <**div class="modal-footer"**>  <**button type="button" class="btn btn-default" data-dismiss="modal"**>Close</**button**>  <**button type="button" class="btn btn-primary"**>Save changes</**button**>  </**div**>  </**div**>  </**div**> </**div**> |

　　※通常では、黄色でマイクする箇所はカスタマーズをするところです。

* 画面の効果

この際、ボタンをクリックすると、ボーダルを呼び出すことをを確認できる。



図表 7‑2　htmlのdata属性でモーダルを呼出す

### ボタンをクリクするとJavaScriptでオプションを指定してモーダルを切り替える例。

* ボタンのソースコード（html）

|  |
| --- |
| **// 呼出元のボタン**  <**button type="button" class="btn btn-primary" id="hogeButton"**>Click to show Modal</**button**> |

* モーダルのソースコード（html）

|  |
| --- |
| **// 呼出先のModalはhtmlのbodyの直下に置く。**  *<!-- Modal -->* <**div class="modal fade" id="hogeModal"**>  <**div class="modal-dialog" role="document"**>  <**div class="modal-content"**>   <**div class="modal-header"**>  <**h4 class="modal-title" id="hogeModelLabel"**>Modal title</**h4**>  </**div**>   <**div class="modal-body"**>  the content you want to show!  </**div**>   <**div class="modal-footer"**>  <**button type="button" class="btn btn-default" data-dismiss="modal"**>Close</**button**>  <**button type="button" class="btn btn-primary"**>Save changes</**button**>  </**div**>  </**div**>  </**div**> </**div**> |

　　※通常では、黄色でマイクする箇所はカスタマーズをするところです。

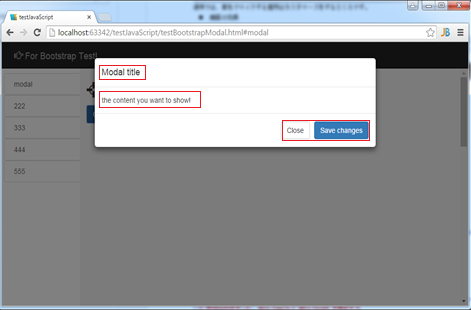
* JavaScriptのソースコード（js）

　　 JavaScriptでボタンのクリックイベントなどで、.modal('show')を利用してモーダルを呼出す。

|  |
| --- |
| **// 呼出元のボタンに、特に何もしない。**  ***$***(**function**(){  **// jQueryで呼出元のボタンを選択する。**  **var** $button = ***$***(**'#hogeButton'**);  **if** ($button.length > 0) {  **// 呼出元にクリックイベントをバインドする。**  $button[0].addEventListener(**'click'**, **function**(){  **// .modal('show')を利用してモーダルを呼出す。**  ***$***(**'#hogeModal'**).**modal**(**'show'**);  },**false**);  } }); |

* 画面の効果

この際、ボタンをクリックすると、ボーダルを呼び出すことをを確認できる。



図表 7‑3　Javascriptでモーダルを呼出す

### カスタマイズ内容を追加する例

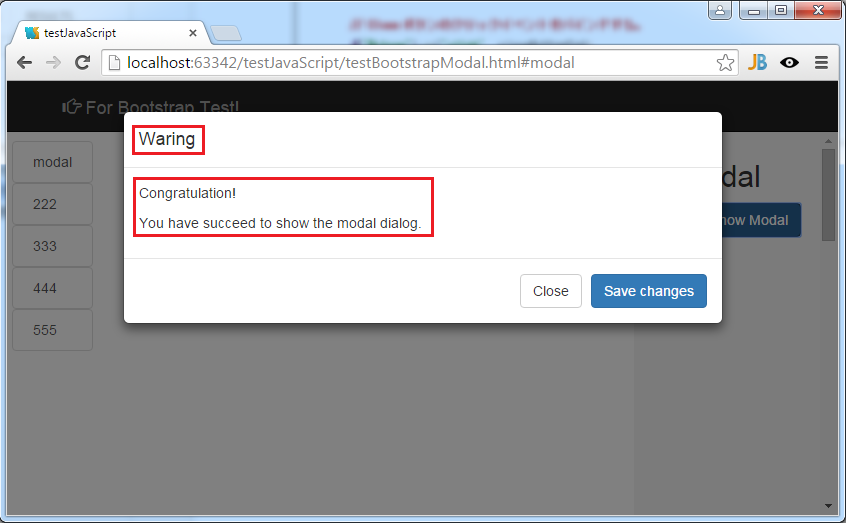
* JavaScriptのソースコード（js）

　　 JavaScriptでボタンのクリックイベントなどで、.modal('show')を利用してモーダルを呼出す。

|  |
| --- |
| **// 呼出元のボタンに、特に何もしない。**  ***$***(**function** () {  **// jQueryで呼出元のボタンとモーダルを選択する。**  **var** $button = ***$***(**'#hogeButton'**),  $modal = ***$***(**'#hogeModal'**);  **if** ($button.length > 0) {  **// 呼出元ボタンのクリックイベントをバインドする。**  $button.on(**'click'**, **function** () {  $modal.**modal**(**'show'**);  $modal.**modal**({**show**: **true**});  });  **// モーダルのCloseボタンをクリックすると実行するFunctionを定義する。**  **var** *closeButtonFun* = **function** () {  **// Closeボタンを押すと、モーダルダイアログを閉じる。**  $modal.**modal**(**'hide'**);  **// 「'Close' button is Clicked.」というメッセージをAlertする**  alert(**"'Close' button is Clicked."**);  }; **// モーダルのSave Changesボタンをクリックすると実行するFunctionを定義する。**  **var** *saveChangeButtonFun* = **function** () {  **// SaveChangesボタンを押すと、モーダルダイアログを閉じる。**  $modal.**modal**(**'hide'**);  **// 「'Save Changes' button is Clicked.」というメッセージをAlertする**  alert(**"'Save Changes' button is Clicked."**);  };  **// モーダルを開いた後で、実行したい動きはこちらに入れます。**  $modal.on(**'shown.bs.modal'**, **function** () {   **// モーダルのタイトルをカスタマイズする。**  **// タイトルは「Modal title」から「Waring」に変更する。*****$***(**'#hogeModalLabel'**).html(**"Waring"**);   **// モーダルの中身をカスタマイズする。**  **// 中身は「the content you want to show!」から次のように変更する。*****$***(**'.modal-body'**).html(**""**).append(**"<p>Congratulation!</p>"**).  append(**"<p>You have succeed to show the modal dialog.</p>"**)  });   **// モーダルを閉じた後で、実行したい動きはこちらに入れます。**$modal.on(**'hidden.bs.modal'**, **function** () { **// 「We have returned to home page.」というメッセージをAlertする**  alert(**"We have returned to home page."**);  });  **// Closeボタンのクリックイベントをバインドする。**  ***$***(**'#close'**).on(**'click'**, *closeButtonFun*);  **// SaveChangesボタンのクリックイベントをバインドする。**  ***$***(**'#saveChanges'**).on(**'click'**, *saveChangeButtonFun*);  } }); |

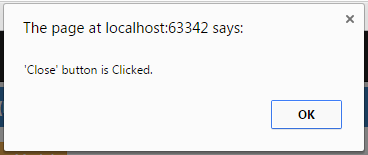
* 実行結果

この際、ボタンをクリックすると、ボーダルを呼び出すことをを確認できる。

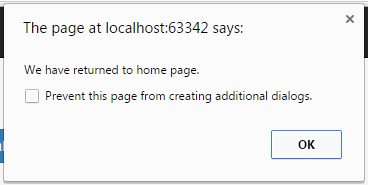


図表 7‑4　モーダルを呼び出す、カスタマイズ内容も入れた。

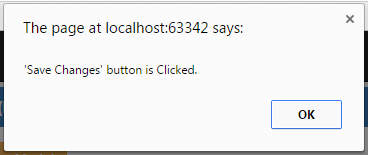
　　　　Closeボタンをクリックする場合。



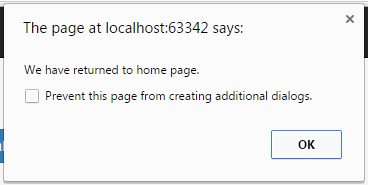
OKボタンを押下し、



　　　　Save Changesボタンを押下する場合。



　　　　OKボタンを押下し、



# タブ表示

概要

Bootstrap3のTab機能を使うことで画面遷移を行わずに、簡単にコンテンツを切り替えることができます。



**図表8－1** **Bootstrap3のTab機能表示のサンプル**

仕組

タブ形式での表示は、ulとliを使用したリストを使用する。ulのクラスに「nav nav-tabs」を指定すればOK。liのクラスに「active」を指定すると、そこが選択状態として表示される。

実装例



**図表8－2　実装例サンプル図**

HTMLを利用した実装例を説明する。

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="ja"**> <**head**>  <**meta charset="utf-8"**>  <**title**>初期表示時</**title**>  **// 必要なライブラリを読み込み**  <**link href="css/bootstrap.css" rel="stylesheet"**>  <**script src="js/bootstrap.js"**></**script**> </**head**> <**body**> <**div class="container"**>  **// タブ表示部**  <**div class="row"**>  <**div class="col-md-12"**>  <**ul class="nav nav-tabs"**>  <**li role="presentation" class="active"**><**a href="#"**>追加要求日順</**a**></**li**>  <**li role="presentation"**><**a href="#"**>数量順</**a**></**li**>  <**li role="presentation"**><**a href="#"**>在庫日数順</**a**></**li**>  </**ul**>  </**div**>  <**p**>**&nbsp;**</**p**>  </**div**> 　　**// 下に表示するデータ**  <**div class="row"**>  <**div class="col-md-12"**>  <**div**>  <**div class="media"**>  <**div class="media-left"**>  <**a data-toggle="modal" data-target="#myModal"**>  <**img class="media-object" src="img/example2.png" alt="2014年7月広報誌" width="64" height="64"** />  </**a**>  </**div**>  <**div class="media-body"**>  <**h4 class="media-heading"**>2014年7月広報誌/ABCD0017</**h4**>  <**table class="table"**>  <**thead**>  <**tr**>  <**th style="width**: 300**px**;**"**>申請店舗</**th**>  <**th style="width**: 150**px**;**"**>追加要求日</**th**>  <**th style="width**: 100**px**;**"**>要求数量</**th**>  <**th style="width**: 100**px**;**"**>在庫日数(昨日時点)</**th**>  </**tr**>  </**thead**>  <**tbody**>  <**tr**>  <**td**>関東地区：赤坂支店</**td**>  <**td**>2014/10/10</**td**>  <**td**>300枚</**td**>  <**td**>8日</**td**>  </**tr**>  </**tbody**>  </**table**>  </**div**>  </**div**>  <**div class="media"**>  <**div class="media-left"**>  <**a data-toggle="modal" data-target="#myModal"**>  <**img class="media-object" src="img/example.png" alt="2014年6月広報誌" width="64" height="64"** />  </**a**>  </**div**>  <**div class="media-body"**>  <**h4 class="media-heading"**>2014年6月広報誌/ABCD0016</**h4**>  <**table class="table"**>  <**thead**>  <**tr**>  <**th style="width**: 300**px**;**"**>申請店舗</**th**>  <**th style="width**: 150**px**;**"**>追加要求日</**th**>  <**th style="width**: 100**px**;**"**>要求数量</**th**>  <**th style="width**: 100**px**;**"**>在庫日数(昨日時点)</**th**>  </**tr**>  </**thead**>  <**tbody**>  <**tr**>  <**td**>関東地区：みなとみらい支店</**td**>  <**td**>2014/10/10</**td**>  <**td**>300枚</**td**>  <**td**>8日</**td**>  </**tr**>  </**tbody**>  </**table**>  </**div**>  </**div**>  <**div class="media"**>  <**div class="media-left"**>  <**a data-toggle="modal" data-target="#myModal"**>  <**img class="media-object" src="img/example2.png" alt="2014年5月広報誌" width="64" height="64"** />  </**a**>  </**div**>  <**div class="media-body"**>  <**h4 class="media-heading"**>2014年5月広報誌/ABCD0015</**h4**>  <**table class="table"**>  <**thead**>  <**tr**>  <**th style="width**: 300**px**;**"**>申請店舗</**th**>  <**th style="width**: 150**px**;**"**>追加要求日</**th**>  <**th style="width**: 100**px**;**"**>要求数量</**th**>  <**th style="width**: 100**px**;**"**>在庫日数(昨日時点)</**th**>  </**tr**>  </**thead**>  <**tbody**>  <**tr**>  <**td**>関東地区：勝どき支店</**td**>  <**td**>2014/10/10</**td**>  <**td**>300枚</**td**>  <**td**>8日</**td**>  </**tr**>  </**tbody**>  </**table**>  </**div**>  </**div**>  <**div class="media"**>  <**div class="media-left"**>  <**a data-toggle="modal" data-target="#myModal"**>  <**img class="media-object" src="img/example.png" alt="2014年4月広報誌" width="64" height="64"** />  </**a**>  </**div**>  <**div class="media-body"**>  <**h4 class="media-heading"**>2014年4月広報誌/ABCD0014</**h4**>  <**table class="table"**>  <**thead**>  <**tr**>  <**th style="width**: 300**px**;**"**>申請店舗</**th**>  <**th style="width**: 150**px**;**"**>追加要求日</**th**>  <**th style="width**: 100**px**;**"**>要求数量</**th**>  <**th style="width**: 100**px**;**"**>在庫日数(昨日時点)</**th**>  </**tr**>  </**thead**>  <**tbody**>  <**tr**>  <**td**>関東地区：八王子支店</**td**>  <**td**>2014/10/10</**td**>  <**td**>300枚</**td**>  <**td**>8日</**td**>  </**tr**>  </**tbody**>  </**table**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**> </**div**> </**body**> </**html**> |

SPAでの実装例



**図表8－3　SPAでの実装図**

タブを表示するため、タグulのClassに**nav nav-tabs** を指定する。

HbsファイルAdditionalDeliveryConfirmListSortItem.hbsを定義する。

|  |
| --- |
| <**ul class="nav nav-tabs"**>  **// 三つのタブを定義**  <**li role="presentation"**>  <**a id="preferredDateSort" class="text-capitalize" href="#listShowToggleTab" data-toggle="tab"**>{{**i18n "label.preferredDeliveryDateOrder"**}}</**a**>  </**li**>  <**li role="presentation"**>  <**a id="amountSort" class="text-capitalize" href="#listShowToggleTab" data-toggle="tab"**>{{**i18n "label.amountOrder"**}}</**a**>  </**li**>  <**li role="presentation"**>  <**a id="stockDaysSort" class="text-capitalize" href="#listShowToggleTab" data-toggle="tab"**>{{**i18n "label.inventoryDayOrder"**}}</**a**>  </**li**> </**ul**> **// タブを切り替える時、コンテンツをここに表示する** <**div class="tab-content"**>  <**div class="tab-pane active" id="listShowToggleTab"**></**div**> </**div**> <**p**>**&nbsp;**</**p**> |

ItemViewファイルAdditionalDeliveryConfirmListSortItemView.jsを定義する。

|  |
| --- |
| **return** Marionette.ItemView.extend({  initialize: **function**(){  },  template: additionalDeliveryConfirmListSortItemTmpl,  onShow: **function**() {  },  ui: {  preferredDateSort: **'#preferredDateSort'** },  events: {  **// タブをクリックされた時、呼ぶメソッドを指定**  **'click a[data-toggle="tab"]'**: **'confirmItemListSortTabClick'** },  confirmItemListSortTabClick: **function** (e) {  **this**.model.set({clickedTabName: e.target.id});  **// トリガー定義、LayoutViewのメソッドを呼ぶ**  app.currentModule.moduleVent.trigger(**"ConfirmListSortItemView:confirmItemListSortTabClick"**, **true**);  } }); |

LayoutViewファイルAdditionalDeliveryConfirmListLayout.jsを定義する。

|  |
| --- |
| **return** Marionette.LayoutView.extend({  initialize: **function** () {  **// 承認一覧画面の承認一覧ソートタブのイベント監視、ItemViewのトリガーを監視**app.currentModule.moduleVent.on(**"ConfirmListSortItemView:confirmItemListSortTabClick"**, **this**.confirmItemListSortTabClick, **this**);   *// 承認一覧画面の承認一覧結果のイベント監視* app.currentModule.moduleVent.on(**"ConfirmListShowItemView:confirmListItemImgLinkClick"**, **this**.confirmListItemImgLinkClick, **this**);  },  template: additionalDeliveryConfirmListTmpl,  regions: {  confirmListSortRegion: **"#additionalDeliveryConfirmListSort-region"**,  confirmListShowRegion: **"#additionalDeliveryConfirmListShow-region"**,  confirmListDetailModalRegion: **"#additionalDeliveryConfirmDetailModal-region"** },  **// 初期表示時**  onShow: **function** () {  **// タブ表示部**  **var** confirmListSortItemView = **new** AdditionalDeliveryConfirmListSortItemView({model: **new** AdditionalDeliveryConfirmListSortModel()});  **this**.confirmListSortRegion.show(confirmListSortItemView); 　　　　**// タブの下のデータ表示部**  **var** confirmListShowCollectionView = **new** AdditionalDeliveryConfirmListShowCollectionView({collection: **new** AdditionalDeliveryConfirmListShowCollection()});  **this**.confirmListShowRegion.show(confirmListShowCollectionView);   **// デフォルト追加要求日順で表示する**confirmListSortItemView.ui.preferredDateSort.click();  },  *// 承認一覧画面の承認一覧ソートタブクリック* confirmItemListSortTabClick: **function** (sortOrderSetFlg) {  **var** approveStatus = **'0'**;  **// Modelを取得**  **var** confirmListSortModel = **this**.confirmListSortRegion.currentView.model;  **// クリックされたタブの名前を取得**  **var** sortKey = confirmListSortModel.get(**'clickedTabName'**);  **// ソートオーダーを取得**  **var** sortOrder = confirmListSortModel.get(**'sortOrder'**);  **// 前回と同じタブがクリックされた時、表示順が逆になる**  **if** (sortKey === confirmListSortModel.previousAttributes().clickedTabName) {  **if** (sortOrderSetFlg === **true**) {  sortOrder = sortOrder === **'1'** ? **'2'** : **'1'**;  }  } **else** {  sortOrder = **'1'**;  }  confirmListSortModel.set(**'sortOrder'**, sortOrder);   **var** confirmListShowCollection = **this**.confirmListShowRegion.currentView.collection;  **// サーバと通信して、データを取得**  confirmListShowCollection.fetch({  data:{approveStatus: approveStatus, sortKey: sortKey, sortOrder: sortOrder},  success: **function** () {  LoggerUtil.consoleLogger.debug(**'[Layout]AdditionalDeliveryConfirmListLayout::fetch::success()'** + JSON.stringify(confirmListShowCollection));  }  });  } }); |

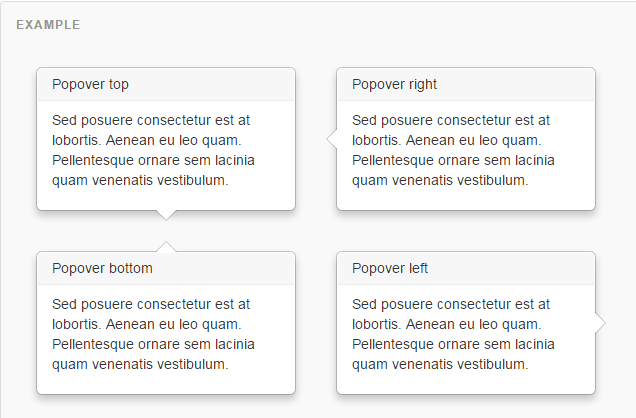
# マウスオーバーによる吹き出し表示

概要

フォームの項目説明や画像にマウスオーバーした時などに便利なポップオーバーである。

iPad のように、コンテンツの小さなオーバーレイを追加して、要素に２次的な情報を提供する。

長さが０のタイトルやコンテンツのあるポップオーバーは決して表示されない。

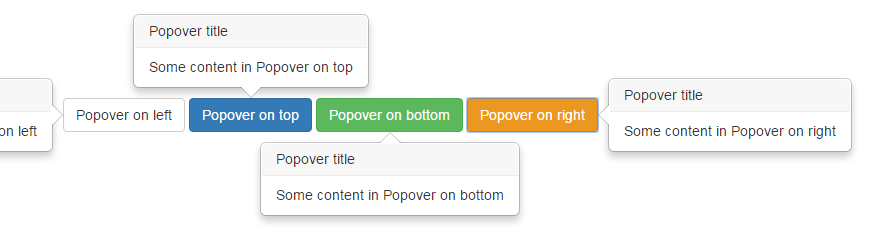


**図表9－1　ポップオーバーのサンプル**

仕組

マウスオーバーで説明のメッセージが出てくる。４つのオプションが利用できます：上、右、下、左に配置されます。

実装例



**図表9－2　四つの方向のポップオーバー**

jQueryを利用した実装例を説明する。

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Bootstrap Popover</**title**>  <**link rel="stylesheet" href="bootstrap.min.css"**>  <**script src="jquery.js"**></**script**>  <**script src="bootstrap.min.js"**></**script**> </**head**> <**body**>  <**div class="container" style="padding**: 100**px** 50**px** 10**px**;**"** >  <**button type="button" class="btn btn-default" title="Popover title"  data-container="body" data-toggle="popover" data-placement="left"  data-content="Some content in Popover on left"**>  Popover on left  </**button**>  <**button type="button" class="btn btn-primary" title="Popover title"  data-container="body" data-toggle="popover" data-placement="top"  data-content="Some content in Popover on top"**>  Popover on top  </**button**>  <**button type="button" class="btn btn-success" title="Popover title"  data-container="body" data-toggle="popover" data-placement="bottom"  data-content="Some content in Popover on bottom"**>  Popover on bottom  </**button**>  <**button type="button" class="btn btn-warning" title="Popover title"  data-container="body" data-toggle="popover" data-placement="right"  data-content="Some content in Popover on right"**>  Popover on right  </**button**> </**div**> // HTMLコードの場合の呼び方 <**script**>***$***(**function** () {  ***$***(**"[data-toggle='popover']"**).**popover**(); });  </**script**> </**div**> </**body**> </**html**> |

SPAでの実装例



**図表9－3　SPAでの実装後の効果図**

* Hbsファイルを定義する。

タグ<a>にidを付与する。

|  |
| --- |
| <**div class="media-left"**>  <**a id="stockInHistoryPopover**{{**itemCode**}}**" name="stockInHistoryPopover"** >  <**img class="media-object" src="**{{**imgPath**}}**" width="64" height="64"**/>  {{#**if lastStorageDate**}}  {{**else**}}  <**div style="position**:**relative**; **margin**:-1**em** -0.4**em"**><**span class="label label-success pull-right text-capitalize"**>{{**i18n "label.noStocked"**}}</**span**></**div**>  {{/**if**}}  </**a**> </**div**> <**div class="media-body"**>  <**h6 class="media-heading text-capitalize"**>{{**i18n "label.theFinalStockedDay"**}}：{{#**if lastStorageDate**}}{{**lastStorageDate**}}{{**else**}}{{**i18n "label.noStocked"**}}{{/**if**}}</**h6**>  <**h4 class="media-heading"**>{{**itemName**}}/{{**itemCode**}}</**h4**>  {{**description**}} </**div**><**br**/> |

* ItemViewファイルを定義する。

|  |
| --- |
| **return** Marionette.**ItemView**.extend({  initialize: **function**(options){  },  **template**: stockInspectionSearchResultItemTmpl,  **events**: {  **'click img'**: **'searchResultClick'**,  **// マウスオーバーした時のイベントを指定する**  **'mouseover img'**: **'setIntervalForShowStockInHistory'**,  **// マウスアウトした時のイベントを指定する**  **'mouseout img'**: **'hideStockInHistoryPopover'** },   *// 初期表示を行う。* onShow: **function**() {  },   *// 無駄な通信を回避するため、延期0.5秒Popoverを呼び出す。* setIntervalForShowStockInHistory: **function** () {  **this**.**mouseTargetFlg** = **true**;  **// 一定間隔で吹き出しを実行するように、ViewUtilを利用する。**  ViewUtil.setIntervalForFunctionCall(**app**.**currentModule**, **this**.showStockInHistory.bind(**this**));  },   *// マウスに当たった商品の入荷実績を検索し、Popoverで表示する。* showStockInHistory: **function** () {   *// 未入庫またはマウスアウトの場合、何もしない。* **if** (!**this**.**model**.get(**'lastStorageDate'**) || **this**.**mouseTargetFlg** === **false**) {  **return**;  }   *// popoverのoptionsを初期化する。* **var** popoverOptions = {  **html**:**true**,  **title**: $.i18n.*t*(**"label.stockedInformation"**),  **content**: **'<div style="width:40px; height:10px"></div>'** };   **var** $collectionDom = **this**.**\_parent**.**$el**;  **var** $allPopover = $collectionDom.find(**'a[name=stockInHistoryPopover]'**);   *// 既に呼ばれたpopoverを閉じる。* $allPopover.popover(**'hide'**);  $allPopover.popover(**'destroy'**);   *// 入庫実績データを取得する前に、初期化したpopoverを表示する。* **var** $itemDom = **this**.**$el**;  **var** $activePopover = $itemDom.find(**'#stockInHistoryPopover'** + **this**.**model**.get(**'itemCode'**));  $activePopover.popover(popoverOptions);  $activePopover.popover(**'show'**);  **// 表示位置を調整する**  $itemDom.find(**'.popover-title'**).attr(**'align'**, **'center'**);  $itemDom.find(**'.popover-title'**).addClass(**'text-capitalize'**);   *// 入庫実績リストを取得する。* **var** spinner = ViewUtil.getSpinner($itemDom.find(**'.popover-content'**)[0], ViewUtil.**positions**.**top**, 0.5);  **var** stockInHistoryList = **new** StockInspectionStockInHistoryCollection();  **var** self = **this**;  stockInHistoryList.fetch({  **data**: **this**.**model**.toJSON(),   *// Ajax通信開始時、Spinnerを表示する* beforeSend: **function** () {  spinner.addSpinner();  },   success: **function** (rCollection) {  LoggerUtil.**consoleLogger**.debug(**"save success: "** + *JSON*.*stringify*(rCollection));   *// 入庫データが取得できない、またはマウスアウトの場合、何もしない。* **if** (!self.**mouseTargetFlg**) {  $activePopover.popover(**'hide'**);  $activePopover.popover(**'destroy'**);  **return**;  }  *// Contentを編集する。* **var** content = **''**;  **if** (!rCollection || rCollection.length === 0) {  content = **'入庫データなし'**;  }  rCollection.each(  **function**(model){  content += model.get(**'stockInDate'**) + **' : '** + model.get(**'amount'**) + **'<br />'**;  }.bind(**this**)  );  *// データが取得したので、Spinnerを非表示にする。* spinner.stopSpinner();  *// 取得した入庫実績データをPopoverで表示する。* $itemDom.find(**'.popover-content'**).html(content);  },  error: **function** () {  DialogUtil.showServerErrorMessage();  }  });  },  *// マウスアウトで入荷実績Popoverを閉じる。* hideStockInHistoryPopover: **function** () {  **this**.**mouseTargetFlg** = **false**;  **var** $allPopover = **this**.**\_parent**.**$el**.find(**'a[name=stockInHistoryPopover]'**);  $allPopover.popover(**'hide'**);  $allPopover.popover(**'destroy'**);  } }); |

# グラフ表示

概要

Chart.jsを使って、折れ線グラフ、棒グラフ、円グラフ、レーダーチャートなど、6種類のグラフが簡単に描ける。

仕組

まずはライブラリ読み込み。

|  |
| --- |
| <**script src="Chart.js"**></**script**> |

HTML上にチャートの描画エリアを作成する

|  |
| --- |
| <canvas id="lineChartCanvas" height="450" width="600"></canvas> |

表示するデータを指定

|  |
| --- |
| **var *lineChartData*** = {  *//x軸の情報* **labels** : [**'1月'**, **'2月'**, **'3月'**, **'4月'**, **'5月'**, **'6月'**, **'7月'**, **'8月'**, **'9月'**, **'10月'**, **'11月'**, **'12月'**],  *//各グラフの情報。複数渡すことができる。* **datasets** : [  {  *//面の色・透明度*  **fillColor** : **"rgba(220, 220, 220, 0.5)"**,  *//線の色・透明度*  **strokeColor** : **"rgba(220, 220, 220, 1)"**,  *//点の色・透明度*  **pointColor** : **"rgba(220, 220, 220, 1)"**,  *//点の周りの色*  **pointStrokeColor** : **"#fff"**,  *//実際のデータ* **data** : [1, 59, 90, 81, 56, 55, 40, 30, 10, 40, 48, 58]  },  {  **fillColor** : **"rgba(151, 187, 205, 0.5)"**,  **strokeColor** : **"rgba(151, 187, 205, 1)"**,  **pointColor** : **"rgba(151, 187, 205, 1)"**,  **pointStrokeColor** : **"#fff"**,  **data** : [28, 48, 40, 19, 96, 27, 100, 33, 63, 31, 64, 51]  }  ]  } |

グラフ全体のオプション（option）

|  |
| --- |
| **var *option*** = {  *//縦軸の目盛りの上書き許可。これ設定しないとscale関連の設定が有効にならないので注意。* **scaleOverride** : **true**,  *//以下設定で、縦軸のレンジは、最小値0から20区切りで140(0+20\*7)までになる。  //縦軸の区切りの数* **scaleSteps** : 7,  *//縦軸の目盛り区切りの間隔* **scaleStepWidth** : 20,  *//縦軸の目盛りの最小値* **scaleStartValue** : 0,  *//アニメーション設定* **animation** : **false**,  *//Y軸の表記（単位など）* **scaleLabel** : **"<%=value%>"**,  *//ツールチップ表示設定* **showTooltips**: **false**,  *//ドットの表示設定* **pointDot** : **false**,  *//線を曲線にするかどうか。falseで折れ線になる。* **bezierCurve** : **false** } |

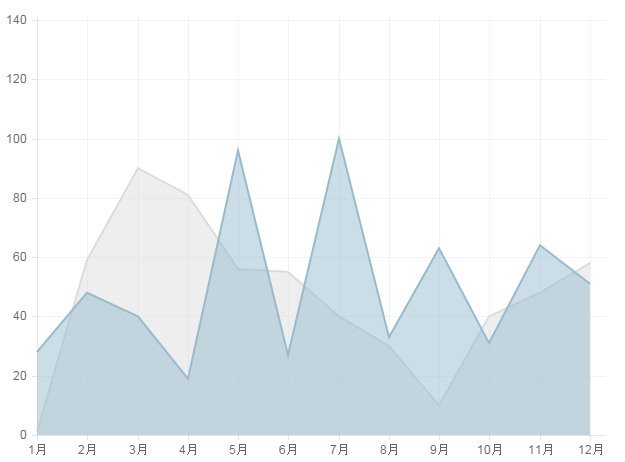
描画する

|  |
| --- |
| **var** ctx = document.getElementById(**"myChart"**).getContext(**"2d"**); **var** myNewChart = **new** Chart(ctx).Line(data, option); |

実装例

折れ線チャートとレーダーチャートを例として説明する。

* 折れ線チャート

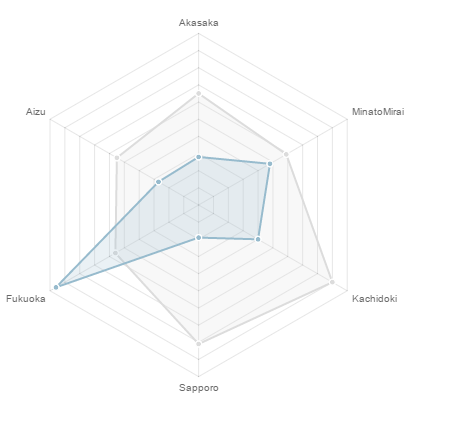


**図表10－1　折れ線グラフ**

Chart.jsを利用した実装例を説明する。

|  |
| --- |
| <!DOCTYPE **html**> <**head**>  *<!--ライブラリ読み込み -->* <**script src="Chart.js"**></**script**> </**head**> <**body**> *<!--描画箇所 -->* <**canvas id="lineChartCanvas" height="450" width="600"**></**canvas**> <**script**>  **var *lineChartData*** = {  *//x軸の情報* **labels** : [**'1月'**, **'2月'**, **'3月'**, **'4月'**, **'5月'**, **'6月'**, **'7月'**, **'8月'**, **'9月'**, **'10月'**, **'11月'**, **'12月'**],  *//各グラフの情報。複数渡すことができる。* **datasets** : [  {  **fillColor** : **"rgba(220, 220, 220, 0.5)"**,  **strokeColor** : **"rgba(220, 220, 220, 1)"**,  **pointColor** : **"rgba(220, 220, 220, 1)"**,  **pointStrokeColor** : **"#fff"**,  *//実際のデータ* **data** : [1, 59, 90, 81, 56, 55, 40, 30, 10, 40, 48, 58]  },  {  **fillColor** : **"rgba(151, 187, 205, 0.5)"**,  **strokeColor** : **"rgba(151, 187, 205, 1)"**,  **pointColor** : **"rgba(151, 187, 205, 1)"**,  **pointStrokeColor** : **"#fff"**,  **data** : [28, 48, 40, 19, 96, 27, 100, 33, 63, 31, 64, 51]  }  ]  }   **var *option*** = {  *//縦軸の目盛りの上書き許可。これ設定しないとscale関連の設定が有効にならないので注意。* **scaleOverride** : **true**,  *//以下設定で、縦軸のレンジは、最小値0から20区切りで140(0+20\*7)までになる。  //縦軸の区切りの数* **scaleSteps** : 7,  *//縦軸の目盛り区切りの間隔* **scaleStepWidth** : 20,  *//縦軸の目盛りの最小値* **scaleStartValue** : 0,  *//アニメーション設定* **animation** : **false**,  *//Y軸の表記（単位など）* **scaleLabel** : **"<%=value%>"**,  *//ツールチップ表示設定* **showTooltips**: **false**,  *//ドットの表示設定* **pointDot** : **false**,  *//線を曲線にするかどうか。falseで折れ線になる。* **bezierCurve** : **false** }   **var *canvas*** = ***document***.getElementById(**"lineChartCanvas"**);  **var *ctx*** = ***canvas***.getContext(**"2d"**);  *//Line()メソッドで折れ線が作れる* **new** *Chart*(***ctx***).Line(***lineChartData***,***option***); </**script**> </**body**> </**html**> |

* レーダーチャート



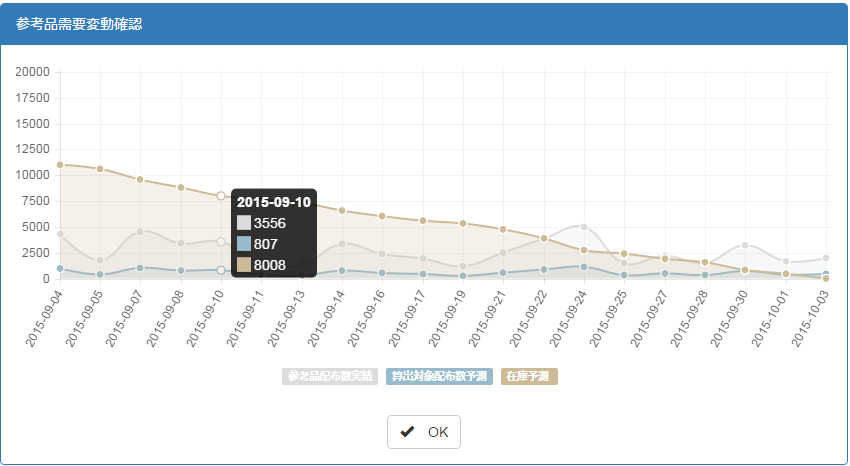
**図表10－2　レーダーチャート**

Chart.jsを利用した実装例を説明する。

|  |
| --- |
| <!doctype **html**> <**html lang="en"**>  <**head**>  <**meta charset="UTF-8"**/>  <**title**>Radar Sample</**title**>  <**script src="Chart.js"**></**script**>  </**head**>  <**body**>  <**canvas id="myChart" width="400" height="400"**></**canvas**>  <**script**>  **var *data*** = {  **labels**: [**"Akasaka"**, **"MinatoMirai"**, **"Kachidoki"**, **"Sapporo"**, **"Fukuoka"**, **"Aizu"**],  **datasets**: [  {  **label**: **"My First dataset"**,  **fillColor**: **"rgba(220,220,220,0.2)"**,  **strokeColor**: **"rgba(220,220,220,1)"**,  **pointColor**: **"rgba(220,220,220,1)"**,  **pointStrokeColor**: **"#fff"**,  **pointHighlightFill**: **"#fff"**,  **pointHighlightStroke**: **"rgba(220,220,220,1)"**,  *//実際のデータ*  **data**: [65, 59, 90, 81, 56, 55]  },  {  **label**: **"My Second dataset"**,  **fillColor**: **"rgba(151,187,205,0.2)"**,  **strokeColor**: **"rgba(151,187,205,1)"**,  **pointColor**: **"rgba(151,187,205,1)"**,  **pointStrokeColor**: **"#fff"**,  **pointHighlightFill**: **"#fff"**,  **pointHighlightStroke**: **"rgba(151,187,205,1)"**,  *//実際のデータ*  **data**: [28, 48, 40, 19, 96, 27]  }  ]  };  **//ID使ってcanvasを取得して、canvas領域に表示する**  **var *canvas*** = ***document***.getElementById(**"myChart"**);  **var *ctx*** = ***canvas***.getContext(**"2d"**);  **// Radar()メソッドでレーダーチャートが作れる**  **var *myRadarChart*** = **new** *Chart*(***ctx***).Radar(***data***);   *// 変数定義 ==============================================* **var *zero***;  **var *unitValue***;  **var *max*** = [0, 0, 0];  *// イベント定義 ===========================================   // クリック開始* **var *clickedLabel*** = **""**;  ***canvas***.*addEventListener*(**"mousedown"**, **function**(evt){ *// 中心座標(x=0,y=0)の座標データを取得する* ***zero*** = [  0,  ***document***.getElementById(**"myChart"**).**width** / 2,  ***document***.getElementById(**"myChart"**).height / 2 + 8 *// TODO この8別途0,0のデータを裏に作って算出する事* ]; *// value, x, y    // Max値&当該データの座標を取得する  //var max = [0, 0, 0]; // value, x, y* ***myRadarChart***.**datasets**.forEach(**function**(dataset){  dataset.**points**.forEach(**function**(data){  **if**(***max***[0] < data.value){  ***max*** = [data.value, data.x, data.y];  }  });  });   *// 中心座標とMaxの距離を計算する* **var** maxDist = ***Math***.*sqrt*(***Math***.*pow*(***zero***[1] - ***max***[1], 2) +  ***Math***.*pow*(***zero***[2] - ***max***[2], 2)  , 2); *// 2点間の距離を求める公式    // maxDistの距離->maxの実際の値。つまり、値を割れば、距離1当たりの実際の値が取得できる* ***unitValue*** = ***max***[0] / maxDist;  **var** activePoints = ***myRadarChart***.getPointsAtEvent(evt);  **if**(activePoints.*length*){ *// データが取得できない場合->データ外でクリックされた場合 は無視する* ***clickedLabel*** = activePoints[0].**label**; *// 当該データ種の特定* } **else** {  ***clickedLabel*** = **""**;  }  }, **false**);   *// クリック中* ***canvas***.*addEventListener*(**"mousemove"**, **function**(evt){  *// クリックされ、データを変更する対象が決まっている時のみデータを動かす* **if**(***clickedLabel***){  *// 中心との距離を算出* **var** dist = ***Math***.*sqrt*(  ***Math***.*pow*(***zero***[1] - evt.**pageX** - ***canvas***.**offsetLeft**, 2) +  ***Math***.*pow*(***zero***[2] - evt.**pageY** - ***canvas***.**offsetTop**, 2)  , 2);  *// 距離から設定すべき値を算出* **var** newVal = ***Math***.*floor*(dist \* ***unitValue***);  *// 新しい値でデータを入れ替える* ***myRadarChart***.**datasets**[0].**points**.forEach(**function**(data){  **if**(data.**label** === ***clickedLabel***){  **var** activePointsCollection = ***myRadarChart***.getPointsAtEvent(evt);  data.**value** = newVal;  **if**(activePointsCollection.*length* > 0){  ***myRadarChart***.**activeElements** = [];  }  }  });  }  }, **false**);   *// クリック終了(マウスアップはどこでも良いのでdocumentから引っ張っている)* ***document***.*addEventListener*(**"mouseup"**, **function**(){  ***myRadarChart***.update();  ***clickedLabel*** = **""**;  }, **false**);  </**script**>  </**body**> </**html**> |

SPAでの実装例

* 折れ線チャート



**図表10－3　折れ線チャート**

LayoutViewの中にItemViewの表示位置を定義する。

|  |
| --- |
| **return** Marionette.LayoutView.extend({  *// 利用条件入力ボタンを押下する* useConditionInput\_confirmButton: **function**(model) {  **var** sampleGoodsChangeConfirmItemView = **new** FirstDeliverySampleGoodsChangeConfirmItemView({**paramModel**: model});  **// searchResultRegionリージョンにItemView を表示するthis**.**mainRegion**.**currentView**.**searchResultRegion**.show(sampleGoodsChangeConfirmItemView.render());  },  }) |

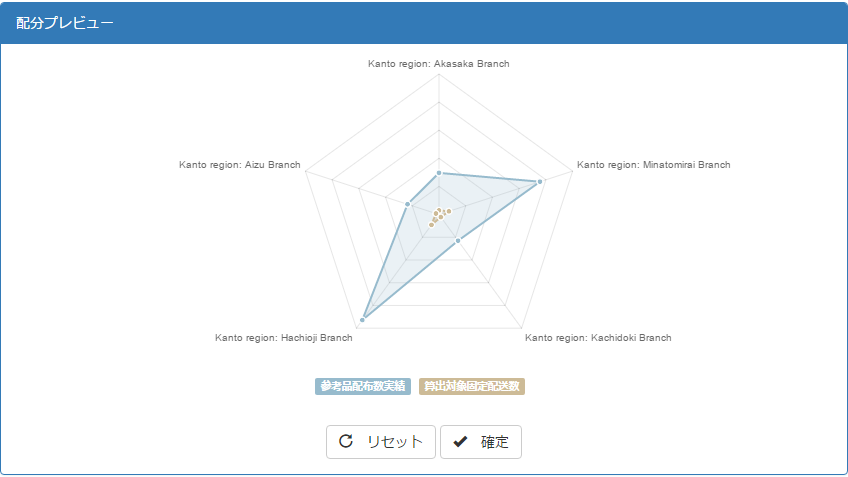
テンプレートであるHbsファイル(FirstDeliverySampleGoodsChangeConfirmItem.hbs)を定義する。

|  |
| --- |
| <**div class="row"**> <**div class="col-md-12"**>  <**div id="changeConfirm" class="panel panel-primary"**>  <**div id="title" class="panel-heading text-capitalize"**>{{**i18n "label.referenceProductDemandFluctuationConfirm"**}}</**div**>  <**div class="panel-body"**>  **// チャートがHTML5のCanvasタグ上で表示される**  <**canvas id="myChart"**></**canvas**>  **// 凡例がdiv上で表示される**  <**div id="myLegend"**></**div**>  **// OKボタン**  <**div id="okButton" class="text-center" style="display**:**block"**>  <**p**><**br** /></**p**>  <**button id="confirmButton" type="button" class="btn btn-default"**>  <**span class="glyphicon glyphicon-ok" aria-hidden="true"**></**span**>　OK  </**button**>  </**div**>  </**div**>  </**div**> </**div**> </**div**> |

ItemView(FirstDeliverySampleGoodsChangeConfirmItemView.js)を定義する。

|  |
| --- |
| **// 折れ線チャートで表示するデータを用意**  **var** data = {  **labels**: [],  **datasets**: [  {  **label**: $.i18n.*t*(**"label.theAmountOfReferenceProductDistribution"**),  **fillColor** : **"rgba(220,220,220,0.2)"**,  **strokeColor** : **"rgba(220,220,220,1)"**,  **pointColor**: **"rgba(220,220,220,1)"**,  **pointStrokeColor** : **"#fff"**,  **pointHighlightFill**: **"#fff"**,  **pointHighlightStroke**: **"rgba(220,220,220,1)"**,  **data**: []  },  {  **label**: $.i18n.*t*(**"label.consumptionForecast"**),  **fillColor**: **"rgba(151,187,205,0.2)"**,  **strokeColor**: **"rgba(151,187,205,1)"**,  **pointColor**: **"rgba(151,187,205,1)"**,  **pointStrokeColor**: **"#fff"**,  **pointHighlightFill**: **"#fff"**,  **pointHighlightStroke**: **"rgba(151,187,205,1)"**,  **data**: []  },  {  **label**: $.i18n.*t*(**"label.stockForecast"**),  **fillColor**: **"rgba(205,187,151,0.2)"**,  **strokeColor**: **"rgba(205,187,151,1)"**,  **pointColor**: **"rgba(205,187,151,1)"**,  **pointStrokeColor**: **"#fff"**,  **pointHighlightFill**: **"#fff"**,  **pointHighlightStroke**: **"rgba(205,187,151,1)"**,  **data**: []  }  ] }; **var** options = {  **scaleBeginAtZero**: **true**,  **legend** : **true**,  **legendTemplate** : **"<div id=\"lengendTemplate\" style=\"text-align:center\">"** +  **"<% for (var i=0; i<datasets.length; i++){%>"** +  **"<span class=\"label\" style=\"background-color: <%=datasets[i].strokeColor%>\">"** +  **"<%=datasets[i].label%></span>&nbsp;&nbsp;<%}%>"** +  **"</div>"** };  **return** Marionette.**ItemView**.extend({  **// 初期化**  initialize: **function** (args) {  **// 外からパラメータを取得**  **this**.**pModel** = args.**paramModel**;  **// modelを新規作成**  **this**.**model** = **new** GraphModel();  **// spinner表示と位置調整**  ViewUtil.setModelCollectionEventsAjax(**this**, ViewUtil.**positions**.**middle**);  }, 　　**// ItemViewが使うテンプレートを定義**  **template**: ItemsTmpl,  **// グラフを表示するメソッド**  onShow: **function** () {  **var** self = **this**;  **// modelで定義したURLでデータをサーバーから取得**  **this**.**model**.fetch({  **data**: **this**.**pModel**.toJSON(),  success: **function** (graphModel) {  LoggerUtil.**consoleLogger**.debug(**"fetch success: "** + *JSON*.*stringify*(graphModel));  **// サーバ通信が成功した後、グラフ表示メソッドを呼ぶ**  self.modalShown();  },  error: **function** () {  DialogUtil.showServerErrorMessage();  }  });  },  **ui**: {  **confirmButton**: **'#confirmButton'**,  **myChart**: **'#myChart'**,  **myLegend**: **'#myLegend'**,  **changeConfirm**: **'#changeConfirm'**,  **title**: **'#title'**,  **okButton**: **'#okButton'** },  **events**: {  **// OKボタン押されたときのメソッドを定義**  **'click @ui.confirmButton'**: **'confirmButtonClick'** },  confirmButtonClick: **function** () {  **app**.**currentModule**.**moduleVent**.trigger(**"sampleGoodsChangeConfirmItemView:confirmButton"**, **this**.**model**);  },  modalShown: **function** () {  **// Canvasをuiから取得**  **var** $canvas = **this**.**ui**.**myChart**;  **// 高さと幅を定義**  $canvas.height(300);  $canvas.**width**($canvas.closest(**"div"**).**width**()); *// 左右のpadding値が15なので、その分引く* **// グラフのラベル表示用データをmodelから取得**  data.**labels** = **this**.**model**.get(**"dateList"**);  **// 実際のデータをmodelから取得、modelのデータはサーバーから受け取り**  data.**datasets**[0].**data** = **this**.**model**.get(**"referSample"**);  data.**datasets**[1].**data** = **this**.**model**.get(**"countObjectForecast"**);  data.**datasets**[2].**data** = **this**.**model**.get(**"stockInForecast"**);   **var** ctx = $canvas.get(0).getContext(**"2d"**);  **var** myChart = **new** Chart(ctx);  **// 折れ線グラフ作成**  **var** legend = myChart.Line(data, options);  **// 凡例の表示内容を生成する**  **this**.**ui**.**myLegend**.get(0).**innerHTML** = legend.generateLegend();  },  **// 使いまわしのため、ほかのページで使う時、このメソッドを呼び出し画面属性を変更する**  attributeModify: **function** () {  **// CSSを削除**  **this**.**ui**.**changeConfirm**.removeClass(**"panel panel-primary"**);  **// CSSを追加**  **this**.**ui**.**changeConfirm**.addClass(**"panel panel-default"**);  **// タイトルを変更**  **this**.**ui**.**title**.text($.i18n.*t*(**"label.referenceProductDemandFluctuationForecast"**));  **// OKボタンを非表示にする**  **this**.**ui**.**okButton**.css(**"display"**, **"none"**);  }  });  }) |

* レーダーチャート



**図表10－4　レーダーチャート**

LayoutViewの中にItemViewの表示位置を定義する。

|  |
| --- |
| *// 参考品需要変動確認OKボタンを押下する* sampleGoodsChangeConfirm\_confirmButton: **function**() {  **var** sampleGoodsChangeforecastItemView = **new** FirstDeliverySampleGoodsChangeforecastItemView({**paramModel**: **this**.**useConditionModel**});  **this**.**mainRegion**.**currentView**.**previewRegion**.show(sampleGoodsChangeforecastItemView);  }, |

テンプレートであるHbsファイル(firstDeliverySampleGoodsChangeforecastItem.hbs)を定義する。

|  |
| --- |
| <**div class="row"**>  <**div class="col-md-12"**>  <**div class="panel panel-primary"**>  <**div class="panel-heading text-capitalize"**>{{**i18n "label.allocationPreview"**}}</**div**>  <**div class="panel-body"**>  **// チャートがHTML5のCanvasタグ上で表示される**  <**canvas id="deliveryPreviewChart"**></**canvas**>  **// 凡例を表示するところ**  <**div id="deliveryPreviewChartLegend"**></**div**>  <**div class="text-center"**>  <**p**><**br**/></**p**>  <**button id="resetButton" type="button" class="btn btn-default"**>  <**span class="glyphicon glyphicon-repeat" aria-hidden="true"**></**span**>　{{**i18n "label.reset"**}}  </**button**>  <**button id="confirmButton" type="button" class="btn btn-default"**>  <**span class="glyphicon glyphicon-ok" aria-hidden="true"**></**span**>　{{**i18n "label.confirm"**}}  </**button**>  </**div**>  </**div**>  </**div**>  </**div**> </**div**> |

ItemView(FirstDeliverySampleGoodsChangeConfirmItemView.js)を定義する。

|  |
| --- |
| **// レーダーチャートで表示するデータを用意**  **var** data = {  **labels**: [],  **datasets**: [  {  **// ラベルの多言語対応**  **label**: $.i18n.*t*(**"label.theAmountOfReferenceProductDistribution"**),  **fillColor**: **"rgba(151,187,205,0.2)"**,  **strokeColor**: **"rgba(151,187,205,1)"**,  **pointColor**: **"rgba(151,187,205,1)"**,  **pointStrokeColor**: **"#fff"**,  **data**: []  },  {  **label**: $.i18n.*t*(**"label.theAmountOfCalcTargetFixedDelivery"**),  **fillColor**: **"rgba(205,187,151,0.2)"**,  **strokeColor**: **"rgba(205,187,151,1)"**,  **pointColor**: **"rgba(205,187,151,1)"**,  **pointStrokeColor**: **"#fff"**,  **data**: []  }  ] };  **// オプションを設定して、チャートをカスタマイズすることが可能、無くても描画可能** **var** options = {  **responsive** : **true**,  **scaleBeginAtZero**: **true**,  **scaleShowGridLines**: **false**,  **scaleGridLineWidth**: 1,  **scaleShowHorizontalLines**: **true**,  **scaleShowVerticalLines**: **true**,  **barShowStroke**: **true**,  **barStrokeWidth**: 2,  **barValueSpacing**: 5,  **barDatasetSpacing**: 1, **// 凡例を表示するかどうか。Trueで表示する**  **legend** : **true**,  **// 凡例に表示する内容**  **tooltipFontColor** : **"rgba(0,255,255,0.8)"**,  **// 凡例に表示する内容**  **legendTemplate** : **"<div id=\"lengendTemplate\" style=\"text-align:center\">"** +  **"<% for (var i=0; i<datasets.length; i++){%>"** +  **"<span class=\"label\" style=\"background-color: <%=datasets[i].strokeColor%>\">"** +  **"<%=datasets[i].label%></span>&nbsp;&nbsp;<%}%>"** +  **"</div>"** }; **var** deliveryPreview; **var** zero; **var** unitValue; **var** clickedLabel = **""**; **var** modelData = []; **return** Marionette.**ItemView**.extend({  initialize: **function** (args) {  **this**.**model** = **new** RadarChartModel();  **this**.**pModel** = args.**paramModel**;  **// Spinner表示位置**  ViewUtil.setModelCollectionEventsAjax(**this**, ViewUtil.**positions**.**middle**);  },  **template**: ItemsTmpl,  onShow: **function** () {  **var** self = **this**;  **this**.**model**.fetch({  **data**: **this**.**pModel**.toJSON(),  success: **function** (radarChartModel) {  **// サーバーとの通信が成功したら、レーダーチャートを表示する**  self.modalShown();  },  error: **function** () {  DialogUtil.showServerErrorMessage();  }  });  },  **// 画面要素のIDとUIを紐づく**  **ui**: {  **confirmButton**: **'#confirmButton'**,  **resetButton**: **'#resetButton'**,  **deliveryPreviewChart**: **'#deliveryPreviewChart'**,  **deliveryPreviewChartLegend**: **'#deliveryPreviewChartLegend'**,  **forecastChart**: **'#forecastChart'**,  **forecastChartLegend**: **'#forecastChartLegend'** },  **// イベント定義**  **events**: {  **'click @ui.confirmButton'**: **'confirmButtonClick'**,  **'click @ui.resetButton'**: **'resetButtonClick'** },  confirmButtonClick: **function** () {  **app**.**currentModule**.**moduleVent**.trigger(**"sampleGoodsChangeforecastItemView:confirmButton"**, **this**.**model**);  },  **// リセットボタンをクリックする**  resetButtonClick: **function** () {  **// データの長さを取得**  **var** length = modelData.**length**;  **for** (**var** i = 0; i < length; i++) {  **// 初期値をレーダーチャートに設定する**  deliveryPreview.**datasets**[1].**points**[i].**value** = modelData[i];  }  **// アップデートして元の値に戻す**  deliveryPreview.update();  },  modalShown: **function** () {  **// Canvasオブジェクトを取得**  **var** $canvas = **this**.**$el**.find(**'#deliveryPreviewChart'**);  **// 高さと幅を設定**  $canvas.height(300);  $canvas.**width**($canvas.closest(**"div"**).**width**());  **// チャートで表示するラベルを設定**  data.**labels** = **this**.**model**.get(**"storeName"**);  **// データをmodelから取得し、チャートに設定**  data.**datasets**[0].**data** = **this**.**model**.get(**"referSample"**);  data.**datasets**[1].**data** = **this**.**model**.get(**"countObjectForecast"**);  **// 編集前の値を保持しておく**modelData = **this**.**model**.get(**"countObjectForecast"**).slice();  **var** ctx = $canvas.get(0).getContext(**"2d"**);  **this**.**myRadarChart** = **new** Chart(ctx);  **// レーダーチャートを描画する**  deliveryPreview = **this**.**myRadarChart**.Radar(data, options);  **// 凡例で表示する内容を指定**  **this**.**ui**.**deliveryPreviewChartLegend**.get(0).**innerHTML** = deliveryPreview.generateLegend(); **// マウスダウン**  $canvas.mousedown(**function** (evt) {  zero = [  0,  **this**.**width** / 2,  **this**.**height** / 2  ];   **// Max値&当該データの座標を取得する****var** max = [0, 0, 0];  deliveryPreview.**datasets**.forEach(**function** (dataset) {  dataset.**points**.forEach(**function** (data) {  **if** (max[0] < data.value) {  max = [data.value, data.x, data.y];  }  });  });  **// 中心座標とMaxの距離を計算する****var** maxDist = **Math**.*sqrt*(**Math**.*pow*(zero[1] - max[1], 2) +  **Math**.*pow*(zero[2] - max[2], 2), 2);  **// 値を割れば、距離1当たりの実際の値が取得できる**unitValue = max[0] / maxDist;   **var** activePoints = deliveryPreview.getPointsAtEvent(evt);  **if** (activePoints.length) {  clickedLabel = activePoints[0].**label**;  } **else** {  clickedLabel = **""**;  }  });   **var** self = **this**;  **// マウスオーバー**  $canvas.*mousemove*(**function** (evt) {  **if** (clickedLabel) {  **// 中心との距離を算出****var** dist = **Math**.*sqrt*(  **Math**.*pow*(zero[1] - evt.**offsetX**, 2) +  **Math**.*pow*(zero[2] - evt.**offsetY**, 2), 2);  **// 距離から設定すべき値を算出****var** newVal = **Math**.*floor*(dist \* unitValue);  **var** cnt = 0;  **// 新しい値でデータを入れ替える**deliveryPreview.**datasets**[1].**points**.forEach(**function** (data) {  **if** (data.**label** === clickedLabel) {  **var** activePointsCollection = deliveryPreview.getPointsAtEvent(evt);  **if**(activePointsCollection.length > 0){  deliveryPreview.**activeElements** = [];  }  data.**value** = newVal;  self.model.get(**"countObjectForecast"**)[cnt] = newVal;  }  cnt = cnt + 1;  });  }  });  **// マウスアップ**  $(**document**).mouseup(**function** () {  deliveryPreview.update();  clickedLabel = **""**;  });  }  }); |

# Bootstrapに合わせたプルダウン表示

概要

bootstrap-selectを使うことでセレクトボックスを格好よくすることができる。

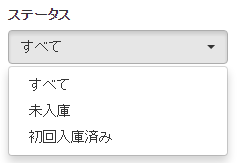


**図表11－1　セレクトボックスの変更前と変更後**

仕組

<select class="selectpicker">タグselectのclass属性にselectpickerを指定すると、使用可能。

実装例



**図表11－2　セレクトボックスの効果図**

前提として、[jQuery](http://d.hatena.ne.jp/keyword/jQuery)が必要。

[公式サイト](http://silviomoreto.github.io/bootstrap-select/)にて、ライブラリをダウンロード。

その中から

・bootstrap-select.[css](http://d.hatena.ne.jp/keyword/css) or bootstrap-select.min.[css](http://d.hatena.ne.jp/keyword/css)  
・bootstrap-select.js or bootstrap-select.min.js

コピーして使用したいHTMLページに読む込むようにする。

|  |
| --- |
| <script **src**=**"jquery.js"**></**script**> <**script src="bootstrap.js"**></**script**> <**script src="bootstrap-select.js"**></**script**> <**link rel="stylesheet" href="bootstrap.css"**/> <**link rel="stylesheet" type="text/css" href="bootstrap-select.css"**> |

セレクトボックスのclassにselectpickerとする

|  |
| --- |
| <div> <label **for**=**"search-condition-status" class**=**"text-capitalize"**>ステータス</label><br/> <select **class**=**"selectpicker"**> <option **value**=**"1"**>すべて</option> <option **value**=**"2"**>未入庫</option> <option **value**=**"3"**>初回入庫済み</option> </select> </div> |

SPAでの実装例



**図表11－3　SPAでのセレクトボックスの効果図**

HbsファイルstockInspectionSearchConditionItem.hbsを定義する

|  |
| --- |
| <**div class="panel panel-primary"**>  <**div class="panel-heading text-capitalize"**>{{**i18n "label.promotionKitSearch"**}}</**div**>  <**div class="panel-body"**>  <**div class="col-md-12"**>  <**div class="col-md-5"**>  <**div class="form-group"**>  <**label for="search-condition-name" class="text-capitalize"**>{{**i18n "label.promotionKitName"**}}</**label**>  <**input type="text" id="search-condition-name" class="form-control" placeholder="**{{**i18n "label.promotionKitName"**}}**（**{{**i18n "label.partialMatch"**}}**）"** />  </**div**>  </**div**>  <**div class="col-md-4"**>  <**div class="form-group"**>  <**label for="search-condition-code" class="text-capitalize"**>{{**i18n "label.promotionKitCode"**}}</**label**>  <**input type="text" id="search-condition-code" class="form-control" placeholder="**{{**i18n "label.promotionKitCode"**}}**（**{{**i18n "label.partialMatch"**}}**）"** />  </**div**>  </**div**>  <**div class="col-md-3"**>  <**div class="form-group"**>  <**label for="search-condition-status" class="text-capitalize"**>{{**i18n "label.status"**}}</**label**><**br**/>  **// プルダウンを定義**  <**select id="search-condition-status" name="status" class="selectpicker text-capitalize"**>  <**option value= "0"**>{{**i18n "label.all"**}}</**option**>  <**option value= "1"**>{{**i18n "label.noStocked"**}}</**option**>  <**option value= "2"**>{{**i18n "label.stocked"**}}</**option**>  </**select**>  </**div**>  </**div**>  </**div**>  </**div**> </**div**> |

stockInspectionSearchConditionItemView.js

|  |
| --- |
| **return** Marionette.**ItemView**.extend({  initialize: **function**(){  },  **// テンプレートファイル（stockInspectionSearchConditionItem.hbs）を指定**  **template**: stockInspectionSearchConditionItemTmpl,  **events**: {  **'keyup input[type=text]'**: **'setIntervalForItemSearch'**,  **// プルダウンの値が変わった時、呼ぶメソッド、検索処理実行**  **'change @ui.Status'**: **'ItemSearch'** },  **ui**:{  *// プロモキット名称* **ItemName**:**'#search-condition-name'**,  *// プロモキット物品コード* **ItemCode**:**'#search-condition-code'**,  *// ステータス* **Status**:**'#search-condition-status'** },   *// 初期表示を行う。* onShow: **function**() {  **this**.**$el**.find(**'.selectpicker'**).*selectpicker*();  *// 言語が英語の場合、先頭のアルファベットを大文字に。* **this**.**$el**.find(**'.selectpicker'**).*selectpicker*(**'setStyle'**, **'text-capitalize'**, **'add'**);  },   *// 入力条件を基づいて商品検索を行う。* setIntervalForItemSearch: **function** () {  ViewUtil.setIntervalForFunctionCall(**this**, **this**.ItemSearch.bind(**this**));  },  ItemSearch : **function** () {  **this**.**model** = **new** StockInspectionItemModel({  **itemName**: **this**.ui.**ItemName**.val(),  **itemCode**: **this**.ui.**ItemCode**.val(),  **status**: **this**.ui.**Status**.val()  });  **// triggerでlayoutView中のメソッドを呼び出す**  **app**.**currentModule**.**moduleVent**.trigger(**"StockInspectionSearchConditionItemView:ItemSearch"**, **this**.**model**);  } }); |

LayoutViewファイルStockInspectionSearchLayout.jsを定義する

|  |
| --- |
| **return** Marionette.**LayoutView**.extend({  initialize: **function** () {  **app**.**currentModule**.**moduleVent**.on(**"StockInspectionSearchConditionItemView:ItemSearch"**, **this**.ItemSearch, **this**);  },   **template**: stockInspectionSearchLayoutTmpl,  **regions**: {  *// 検索条件エリア* **searchConditionRegion**: **"#stockInspection-searchCondition-region"**,  *// 検索結果エリア* **searchResultRegion**: **"#stockInspection-searchResult-region"** },   *// 初期表示を行う。* onShow: **function** () {  **this**.**SearchConditionItemView** = **new** StockInspectionSearchConditionItemView();  **this**.**searchConditionRegion**.show(**this**.**SearchConditionItemView**.render());  **this**.**SearchResultCollectionView** = **new** StockInspectionSearchResultCollectionView(  {  **collection**: **new** StockInspectionSearchResultCollection(),  **model**: **new** StockInspectionItemModel()  });  **this**.**searchResultRegion**.show(**this**.**SearchResultCollectionView**.render());  },   ***// 入力条件により検索を行う。***ItemSearch: **function** (searchConditionModel) {  **// collectionViewの中のメソッドを呼ぶ**  **this**.**SearchResultCollectionView**.searchItemByCondition(searchConditionModel);  } }); |

**CollectionView**ファイルStockInspectionSearchResultCollectionView.jsを定義する

|  |
| --- |
| **return** Marionette.**CollectionView**.extend({  initialize: **function**(args){  **this**.**collection** = args.**collection**;  **this**.**model** = args.**model**;  *// 通信時のspinnerを表示する。* ViewUtil.setModelCollectionEventsAjax(**this**);  },  **childView**: StockInspectionSearchResultItemView,  **tagName**: **'div'**,  **className**: **'list-group'**,    **childViewOptions**: {  **imgClickEventEffectiveFlg**: **true** },   *// 初期表示を行う。* onShow: **function** () {  *// 検索条件より商品情報を検索する。* **this**.searchItemByCondition(**this**.**model**);  },   *// 検索条件より商品情報を検索する。* searchItemByCondition: **function**(conditionModel) {  **// fetchでサーバーからデータを取得**  **this**.**collection**.fetch({  **data**: conditionModel.toJSON(),   error: **function** () {  DialogUtil.showServerErrorMessage();  }  });  } }); |

# カレンダー表示

概要

bootstrap-datepickerを使って、カレンダー表示ができる。



**図表12－1　カレンダー表示画面**

仕組



**図表12－2　シンプルなカレンダー画面**

必要なライブラリ読み込み。

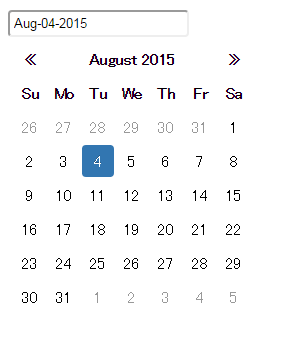
|  |
| --- |
| <script src=**"jquery.js"**></script> <script src=**"bootstrap-datepicker.js"**></script> <link rel=**"stylesheet"** href=**"bootstrap-datepicker3.css"** /> |

HTML上Input要素を追加すれば、表示できる。

|  |
| --- |
| <input data-provide=**"datepicker"**> <script> *$*(**'.datepicker'**).datepicker() </script> |

フォーマット指定も可能。

|  |
| --- |
| <input type=**"text" class**=**"form-control datepicker"**/>  <script> *$*(**'.datepicker'**).datepicker({  **format**:’M-dd-yy’ }) </script> |



**図表12－3　フォーマットM-dd-yyyyのカレンダー画面**

日本語化

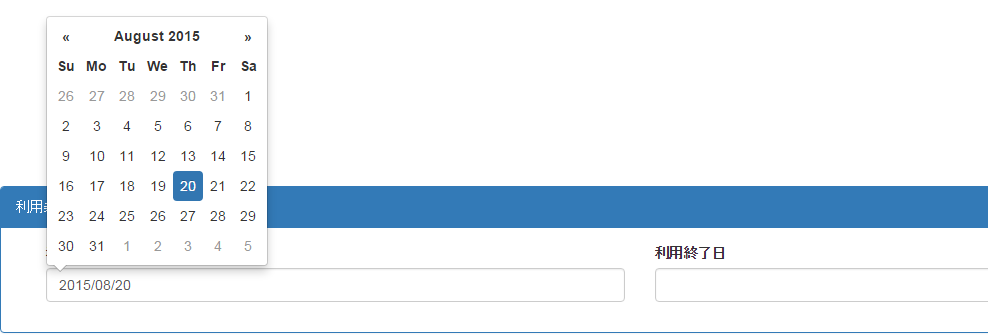
locales/bootstrap-datepicker.ja.jsを追加

|  |
| --- |
| <script src=**"bootstrap-datepicker.ja.js"**></script> |

languageを指定

|  |
| --- |
| $(**'.datepicker'**).datepicker({  **language**: **'ja'** }) |

実装例



**図表12－4　HTMLを使って作成したカレンダー表示画面**

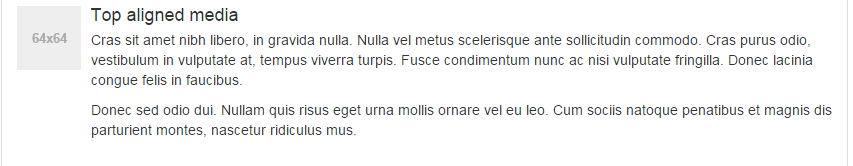
htmlを利用した実装例。

|  |
| --- |
| <!DOCTYPE **html**> <**head**>  <**script src="jquery.js"**></**script**>  <**script src="bootstrap.js"**></**script**>  <**script src="bootstrap-datepicker.js"**></**script**>  <**link rel="stylesheet" href="bootstrap.css"**/>  <**link rel="stylesheet" href="bootstrap-datepicker3.css"** />  <**link rel="stylesheet" href="bootstrap-select.css"** /> </**head**> <**body**>  <**div class="row"**>  <**div class="col-md-12"**>  <**div class="panel panel-primary"**>  <**div class="panel-heading text-capitalize"**>利用条件入力</**div**>  <**div class="panel-body"**>  <**div class="col-md-12"**>  <**div class="col-md-6"**>  <**div class="form-group"**>  <**label for="start-date" class="text-capitalize"**>利用開始日</**label**>  <**input id="start-date" type="text" class="form-control" data-provide="datepicker" data-date-format="yyyy/mm/dd" value=""**>  </**div**>  </**div**>  <**div class="col-md-6"**>  <**div class="form-group"**>  <**label for="end-date" class="text-capitalize"**>利用終了日</**label**>  <**input id="end-date" type="text" class="form-control" data-provide="datepicker" data-date-format="yyyy/mm/dd" value=""**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**> </**div**> </**body**> </**html**> |

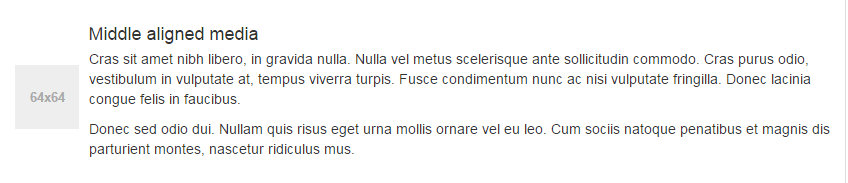
# 画像を組み合わせたリスト表示

概要

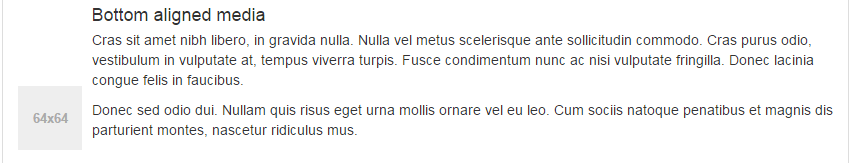
Bootstrap3のMediaオブジェクトを使うことで画像を組み合わせたリストが表示できる。class属性を指定すれば簡単に位置や装飾が適用される。



**図表13－1　画像64\*64の表示位置が上部**

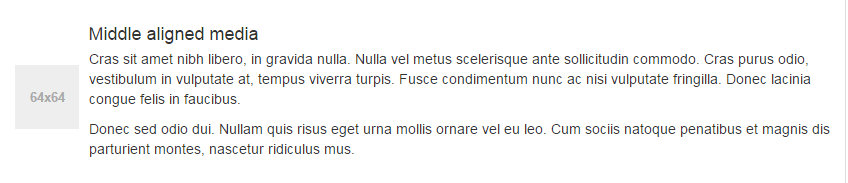


**図表13－2　画像64\*64の表示位置が中部**



**図表13－3　画像64\*64の表示位置が下部**

仕組



**図表13－4　画像64\*64の表示位置が左中**

div要素を使うスタイルのポイントは３つ。

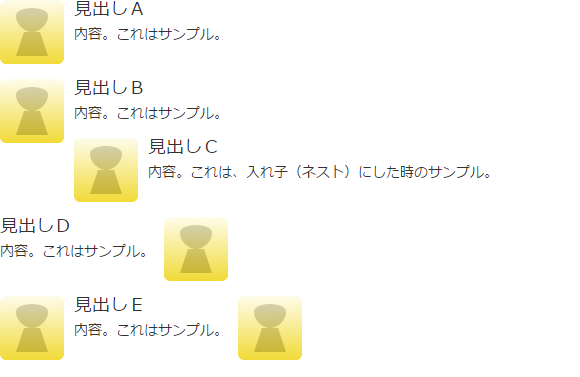
1. class属性にmediaを指定したdiv要素を用意する。
2. １のdiv要素内に、media-left、media-body、media-rightの何れかをclass属性に指定した要素を配置する。
3. ２の要素内で、見出しタグ（h1～h6）を使う場合は、class属性にmedia-headingを指定する。

|  |
| --- |
| <**div class="media"**>  <**div class="media-left media-middle"**>  <**a href="#"**>  <**img class="media-object" src="..." alt="..."**>  </**a**>  </**div**>  <**div class="media-body"**>  <**h4 class="media-heading"**>Middle aligned media</**h4**>  ...  </**div**> </**div**> |

実装例

### div要素を使うスタイル

class属性によって、画像の表示位置が違う。



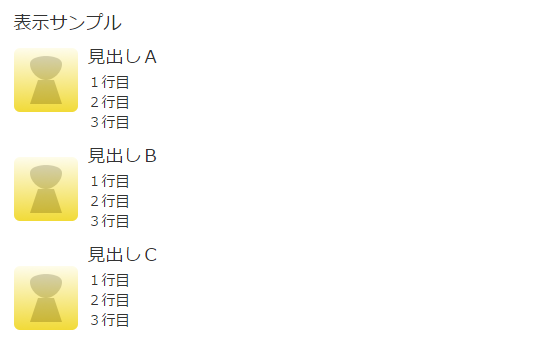
**図表13－5　div要素を使うスタイルの画面**

htmlを利用した実装例。

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="ja"**> <**head**>  <**meta http-equiv="X-UA-Compatible" content="IE=edge"**>  <**meta name="viewport" content="width=device-width, initial-scale=1"**>   *<!-- Bootstrap core CSS -->* <**link rel="stylesheet" href="http://bootstrap3.cyberlab.info/bootstrap/dist/css/bootstrap.css"**>  *<!-- Bootstrap theme -->* <**link rel="stylesheet" href="http://bootstrap3.cyberlab.info/bootstrap/dist/css/bootstrap-theme.css"**>  <**title**>test</**title**> </**head**> <**body**> <**div class="media"**>  <**a class="media-left" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＡ</**h4**>  内容。これはサンプル。  </**div**> </**div**> <**div class="media"**>  <**a class="media-left" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＢ</**h4**>  内容。これはサンプル。  <**div class="media"**>  <**a class="media-left" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＣ</**h4**>  内容。これは、入れ子（ネスト）にした時のサンプル。  </**div**>  </**div**>  </**div**> </**div**> <**div class="media"**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＤ</**h4**>  内容。これはサンプル。  </**div**>  <**a class="media-right" href="#"**>  <**img src="sample-64x64.png"**>  </**a**> </**div**> <**div class="media"**>  <**a class="media-left" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＥ</**h4**>  内容。これはサンプル。  </**div**>  <**a class="media-right" href="#"**>  <**img src="sample-64x64.png"**>  </**a**> </**div**> </**body**> </**html**> |

### 配置

画像の位置がtop,middle,bottom三つの種類がある。



**図表13－6　画像配置**

ソースコード

|  |
| --- |
| <**div class="media"**>  <**a class="media-left media-top" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＡ</**h4**>  １行目<**br**>２行目<**br**>３行目  </**div**> </**div**> <**div class="media"**>  <**a class="media-left media-middle" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＢ</**h4**>  １行目<**br**>２行目<**br**>３行目  </**div**> </**div**> <**div class="media"**>  <**a class="media-left media-bottom" href="#"**>  <**img src="sample-64x64.png"**>  </**a**>  <**div class="media-body"**>  <**h4 class="media-heading"**>見出しＣ</**h4**>  １行目<**br**>２行目<**br**>３行目  </**div**> </**div**> |

# カメラ起動

概要

長い間、Audio/Video capture はウェブ開発にとって難しいことであった。HTML5の登場によって、デバイスハードウェアへのアクセスは可能になった。

今回はHTML5 のファイル入力タグを利用して、モバイルのカメラ起動機能を説明する。



図表 14‑1　カメラ起動タグイメージ

仕組

<input type="file">タグにcapture=”camera”属性を追加して、該当ファイル選択ボタンを押下時に、モバイルのカメラを起動できる。撮影した写真は画面に表示したり、サーバーに送ったりすることはできる。

実装例

■HTML5のInputタグで実装する

|  |
| --- |
| <**input type="file" id="qr-code-input" accept="image/\*" capture="camera"**/>  　　　　　　　　　　　　　↓    　　　　　　　　　　　　　↓  「ファイルを選択」ボタンを押下して、カメラは起動する。 |

■<input type="file">はそのままだとかっこう悪くて、以下の方法でボタンに変更できる。

（bootstrapのインポートは省略されている）

|  |
| --- |
| *<!-- カメラを起動するタグ：非表示 -->* <**input type="file" id="qr-code-input" accept="image/\*" capture="camera" style="display**:**none"**/>  *<!-- カメラ起動ボタン：押下時にカメラ起動 -->* <**button type="button" id="camera-start-button" class="btn btn-primary btn-block btn-lg"**>  <**span class="glyphicon glyphicon-camera" aria-hidden="true"**></**span**> カメラ起動 </**button**>  <**script**>  **$**(**function** () {  **$**(**"#camera-start-button"**).click(**function** () {  **$**(**"#qr-code-input"**).click();  });  **$**(**"#qr-code-input"**).change(**function** () {  **// ～ファイルの情報を取得して、後続処理を行う～   // inputタグの値をクリアする。  $**(**"#qr-code-input"**).val(**""**);   **return false**;  });  }); </**script**> |

・上記ソースの表示結果は以下となる。



■カメラ起動後に撮影した写真を画面に表示したい場合

|  |
| --- |
| <**div class="thumbnail"**>  <**img id="uploadImage1" style="width**:100%**" alt="..."**> </**div**>  *<!-- カメラを起動するタグ：非表示 -->* <**input type="file" id="qr-code-input" accept="image/\*" capture="camera" style="display**:**none"**/>  *<!-- カメラ起動ボタン：押下時にカメラ起動 -->* <**button type="button" id="camera-start-button" class="btn btn-primary btn-block btn-lg"**>  <**span class="glyphicon glyphicon-camera" aria-hidden="true"**></**span**> カメラ起動 </**button**>  <**script**>  **$**(**function** () {  **$**(**"#camera-start-button"**).click(**function** () {  **$**(**"#qr-code-input"**).click();  });  **$**(**"#qr-code-input"**).change(**function** () {   **// ファイルの情報を取得  var** uploadFile = **$**(**"#qr-code-input"**)[0].**files**[0];   **var** reader = **new FileReader**();  reader.onload = **function** (event) {  **$**(**"#uploadImage1"**)[0].**src** = reader.**result**;  };  reader.readAsDataURL(uploadFile);   **// inputタグの値をクリアする。  $**(**"#qr-code-input"**).val(**""**);   **return false**;  });  }); </**script**> |

■カメラ起動後に撮影した写真をサーバーに送信したい場合

|  |
| --- |
| *<!-- カメラを起動するタグ：非表示 -->* <**input type="file" id="qr-code-input" accept="image/\*" capture="camera" style="display**:**none"**/>  *<!-- カメラ起動ボタン：押下時にカメラ起動 -->* <**button type="button" id="camera-start-button" class="btn btn-primary btn-block btn-lg"**>  <**span class="glyphicon glyphicon-camera" aria-hidden="true"**></**span**> カメラ起動 </**button**>  <**script**>  **$**(**function** () {  **$**(**"#camera-start-button"**).click(**function** () {  **$**(**"#qr-code-input"**).click();  });  **$**(**"#qr-code-input"**).change(**function** () {  **// ファイルの情報を取得  var** uploadFile = **$**(**"#qr-code-input"**)[0].**files**[0];   **var** formData = **new FormData**();  formData.append(**"file"**, uploadFile);   **// ファイルをサーバーに送信する。  // サーバー側はSpringの場合、org.springframework.web.multipart.MultipartHttpServletRequestで  // 受取ることはできる。  $**.ajax({  **type**: **'post'**,  **// ファイル送信のURLを指定する必要がある。  // url: /xxx',  data**: formData,  **processData**: **false**,  **contentType**: **false**, **// 送信するデータをFormDataにする場合、こうしないといけない。  cache**: **false**,  **dataType**: **'json'**,  success: **function** () {  **// 成功処理** },  error: **function** () {  **// エラー処理** }  });   **// inputタグの値をクリアする。  $**(**"#qr-code-input"**).val(**""**);   **return false**;  });  }); </**script**> |

# カルーセル表示

概要

Bootstrapで用意されている、JavaScriptを使ったコンポーネント「Carousel（カルーセル）」を利用する。

「Carousel（カルーセル）」は複数の画像をスライド表示してくれるものである。

仕組

まずはBootstrapのサイトで用意されているカルーセルをみてみよう。  
 [Bootstrap](http://getbootstrap.com/" \t "_blank" \o "Bootstrap)の上部メニューより「JavaScript」をクリックしてページ移動後、右サイドメニューから「Carousel」をクリック。  
 すると「Carousel」のExamplesが表示されている。

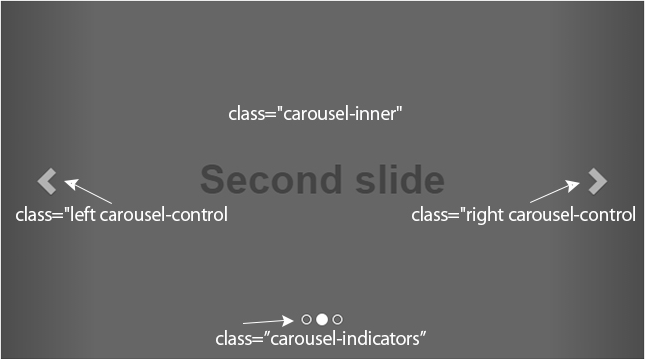
|  |
| --- |
| <**div id="carousel-example-generic" class="carousel slide" data-ride="carousel"**>  *<!-- Indicators -->* <**ol class="carousel-indicators"**>  <**li data-target="#carousel-example-generic" data-slide-to="0" class="active"**></**li**>  <**li data-target="#carousel-example-generic" data-slide-to="1"**></**li**>  <**li data-target="#carousel-example-generic" data-slide-to="2"**></**li**>  </**ol**>  *<!-- Wrapper for slides -->* <**div class="carousel-inner" role="listbox"**>  <**div class="item active"**>  <**img src="..." alt="..."**>  <**div class="carousel-caption"**>  ...  </**div**>  </**div**>  <**div class="item"**>  <**img src="..." alt="..."**>  <**div class="carousel-caption"**>  ...  </**div**>  </**div**>  ...  </**div**>  *<!-- Controls -->* <**a class="left carousel-control" href="#carousel-example-generic" role="button" data-slide="prev"**>  <**span class="glyphicon glyphicon-chevron-left" aria-hidden="true"**></**span**>  <**span class="sr-only"**>Previous</**span**>  </**a**>  <**a class="right carousel-control" href="#carousel-example-generic" role="button" data-slide="next"**>  <**span class="glyphicon glyphicon-chevron-right" aria-hidden="true"**></**span**>  <**span class="sr-only"**>Next</**span**>  </**a**> </**div**> |

大まかに３つのクラス属性で作られている。

1. [class=”carousel-indicators”]=中央に配置されている3つの丸い目印。
2. [class=”carousel-inner”]=画像を配置する場所。div class=”item acitve“を付けたところが最初に表示される。
3. [class=”carousel-control”]=左右に付いている送りボタン。

その他

1. デフォルトのスライド時間は5秒。
2. 複数のCarouselを使うときはid変更を忘れずに。
3. 画像ホバー時に一時停止。



**図表15－1　カルーセル表示例**

画像を配置するためには[class=”carousel-inner”]の[img]タグに画像までのリンクを記述する。  
 captionもつけられるのでタイトルと説明をつける。

|  |
| --- |
| <**div class="carousel-inner" role="listbox"**>  <**div class="item active"**>  <**img src="../img/img1.png" alt="写真1"**>  <**div class="carousel-caption"**>  <**h1**>写真1</**h1**>  <**p**>おいしいコーヒーと自分だけの時間</**p**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="../img/img2.png" alt="写真2"**>  <**div class="carousel-caption"**>  <**h1**>写真2</**h1**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="../img/img3.png" alt="写真3"**>  <**div class="carousel-caption"**>  <**h1**>写真3</**h1**>  </**div**>  </**div**> </**div**> |

作業はこれだけ。サイトに合わせたサイズ変更などの微調整は必要かと思う。

実装例



**図表15－2　カルーセル表示例**

htmlを利用した実装例。

|  |
| --- |
| <!DOCTYPE **html**> <**head**>  <**script src="jquery.js"**></**script**>  <**script src="bootstrap.js"**></**script**>  <**link rel="stylesheet" href="bootstrap.css"**/>  <**link rel="stylesheet" href="carousel.css"**/> </**head**> <**body**> <**div id="top-carousel" class="carousel slide" data-ride="carousel"**>  *<!-- Indicators -->* <**ol class="carousel-indicators"**>  <**li data-target="#top-carousel" data-slide-to="0" class="active"**></**li**>  <**li data-target="#top-carousel" data-slide-to="1"**></**li**>  <**li data-target="#top-carousel" data-slide-to="2"**></**li**>  </**ol**>  <**div class="carousel-inner" role="listbox"**>  <**div class="item active"**>  <**img src="Carousel1.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h5**>Accenture Foundation Platform for Java - Sample Solution -</**h5**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="Carousel2.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h3**>Check Code!</**h3**>  <**span class="glyphicon glyphicon-hand-right" aria-hidden="true"**></**span**>**&nbsp;** <**a href="https://innersource.accenture.com/afpjshowcase/afpjshowcase/" target="\_blank"**>  https://innersource.accenture.com/afpjshowcase/afpjshowcase/  </**a**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="Carousel3.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h3**>Join Us!</**h3**>  <**span class="glyphicon glyphicon-hand-right" aria-hidden="true"**></**span**>**&nbsp;** <**a href="https://www.yammer.com/accenture.com/#/groups/5321071/info" target="\_blank"**>  https://www.yammer.com/accenture.com/#/groups/5321071/info  </**a**>  </**div**>  </**div**>  </**div**> </**div**> </**body**> </**html**> |

SPAでの実装例



**図表15－3　SPA画面でトップページのカルーセル表示例**

LayoutViewを定義する。

|  |
| --- |
| **return** Marionette.LayoutView.extend({  template: topLayoutTmpl,  regions: {  portletsRegion: **'#top-portlets-region'**,  bottomMenuRegion: **'#top-portlet-bottom-menu-region'** },  initialize: **function** () {  },  onShow: **function** () {  **this**.portletsRegion.show((**new** PortletsCollectionView()).render());  *// carouselの自動ループ* **this**.$el.find(**'#top-carousel'**).carousel({interval: 5000});  } }); |

テンプレートであるHbsファイルを定義する。

|  |
| --- |
| <**div class="container"**>  <**div class="row"**>  <**div class="col-md-2"**></**div**>  <**div class="col-md-8"**>  <**div id="top-carousel" class="carousel slide" data-ride="carousel"**>  *<!-- Indicators -->* <**ol class="carousel-indicators"**>  <**li data-target="#top-carousel" data-slide-to="0" class="active"**></**li**>  <**li data-target="#top-carousel" data-slide-to="1"**></**li**>  <**li data-target="#top-carousel" data-slide-to="2"**></**li**>  </**ol**>  *<!-- Wrapper for slides -->* <**div class="carousel-inner" role="listbox"**>  <**div class="item active"**>  <**img src="/static/img/carousel/Carousel1.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h5**>Accenture Foundation Platform for Java - Sample Solution -</**h5**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="/static/img/carousel/Carousel2.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h3**>Check Code!</**h3**>  <**span class="glyphicon glyphicon-hand-right" aria-hidden="true"**></**span**>**&nbsp;** <**a href="https://innersource.accenture.com/afpjshowcase/afpjshowcase/" target="\_blank"**>  https://innersource.accenture.com/afpjshowcase/afpjshowcase/  </**a**>  </**div**>  </**div**>  <**div class="item"**>  <**img src="/static/img/carousel/Carousel3.png" alt="Accenture Foundation Platform for Java - Sample Solution -"**>  <**div class="carousel-caption" style="color**: **#bf800c**; **text-shadow**: **none"**>  <**h3**>Join Us!</**h3**>  <**span class="glyphicon glyphicon-hand-right" aria-hidden="true"**></**span**>**&nbsp;** <**a href="https://www.yammer.com/accenture.com/#/groups/5321071/info" target="\_blank"**>  https://www.yammer.com/accenture.com/#/groups/5321071/info  </**a**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**> </**div**> |

# 要素のドラッグによるソート

概要

リストの要素をドラッグして、リストを並べ替える機能は便利である。

今回はjQuery UI の Sortable プラグインを使い、ドラッグ＆ドロップで並べ替えを行える方法を説明する。



図表 16‑1　ドラッグ＆ドロップによる並べ替えが行えるイメージ

■参照ライブラリ

・jQuery UI

<http://jqueryui.com/download/>よりダウンロードできる。

JQueryライブラリに依存しているので、JQueryライブラリも必要になる。

仕組

jQuery UIプラグインを用いて、リスト要素にsortableメソッドをかけて、リストの並び順はドラッグによるソースできる。ソート順変更などのイベントで、ソート後の処理は実装できる。

実装例

■ドラッグによるソートの実装例（jQuery UIとbootstrapのインポートは省略されている）

|  |
| --- |
| <**ul id="sortable-ul" class="list-group" style="cursor**: **move**;**"**>  <**li id="sortable-item1" class="list-group-item"**>Item1</**li**>  <**li id="sortable-item2" class="list-group-item"**>Item2</**li**>  <**li id="sortable-item3" class="list-group-item"**>Item3</**li**>  <**li id="sortable-item4" class="list-group-item"**>Item4</**li**>  <**li id="sortable-item5" class="list-group-item"**>Item5</**li**>  <**li id="sortable-item6" class="list-group-item"**>Item6</**li**> </**ul**>  <**script**>  **$**(**function** () {  **// リスト順を変更できる要素にする。  $**(**"#sortable-ul"**).sortable({  **// DOMのpositionが変更された際にトリガされる。** update: **function** (event, ui) {  **// 要素の並び順をサーバーに保存しないと意味がないので、  // toArrayメソッドで取得した要素id配列を識別子として、  // サーバーに送信して、並び順をDBに保存できる。  console**.log(**$**(**"#sortable-ul"**).sortable(**"toArray"**));  }  });  **// テキスト選択を無効にする。  $**(**"#sortable-ul"**).**disableSelection**();  }); </**script**> |

　　詳細なイベント、メソッドとオプションのAPIは<http://api.jqueryui.com/sortable/>をご参照ください。  
日本語のサイトは<http://js.studio-kingdom.com/jqueryui/interactions/sortable>である。

モバイル対応策

実装例のソースはそのままだとiPhoneなどのタッチデバイスで動かない。jQuery UIはマウスイベントを監視していて、タッチイベントを監視していない原因である。

jQuery UIをタッチデバイスに対応させるため、jQuery UI Touch Punchライブラリを導入することで解決している。モバイルでドラッグによるソートは可能になるわけ。

■参照ライブラリ

・jQuery UI Touch Punch

<http://touchpunch.furf.com/>よりダウンロードできる。

注意：jQuery UIプラグインの後ろにjQuery UI Touch Punchをインポートする。

■使い方

・jQuery UIを参照

・jQuery UI Touch Punchを参照

　　・上記実装例と同じように実装する