第 8 章

グリッドシステム

この章では、Bootstrap のグリッドシステムについて学習します。グリッドシステムを利用すると、ブラウザの表示幅に応じてレイアウトを切り替えるレスポンシブウェブデザインを簡単に実現できます。

8.1 レスポンシブウェブデザインの適用

本章では、PicoPlanner に**レスポンシブウェブデザイン**（responsive web design）を適用し、PC 用のブラウザで表示した時とスマートフォンのブラウザで表示した時で、適切にレイアウトが切り替わるようにしたいと思います。

`plan_items#index` アクション用のテンプレートを次のように書き換えてください。

app/views/plan_items/index.html.erb

```
1 - <div>
1 + <div class='container-fluid'>
2   <% @plan_items.each do |item| %>
3 -   <div>
3 +   <div class='row'>
4 -     <div>
4 +     <div class='col-12 col-md-4'>
5         <%= item.name %>
6       </div>
7 -     <div>
```

```
 7 +      <div class='col-12 col-md-8'>
 8          <%= item.description %>
 9        </div>
10      </div>
11    <% end %>
12    </div>
```

レスポンシブウェブデザインを適用するには、まず適用範囲全体を囲む要素の class 属性に container または container-fluid を指定する必要があります。違いについては後述します。また、3 行目、4 行目、7 行目で行っている変更の意味についても後で説明します。

PicoPlanner を起動し、ブラウザで http://localhost:3000/plan_items を開くと 図 8.1 のように件名と説明が横並びに表示されます。ただし、ブラウザの表示幅を十分に広く（768px 以上）する必要があります。

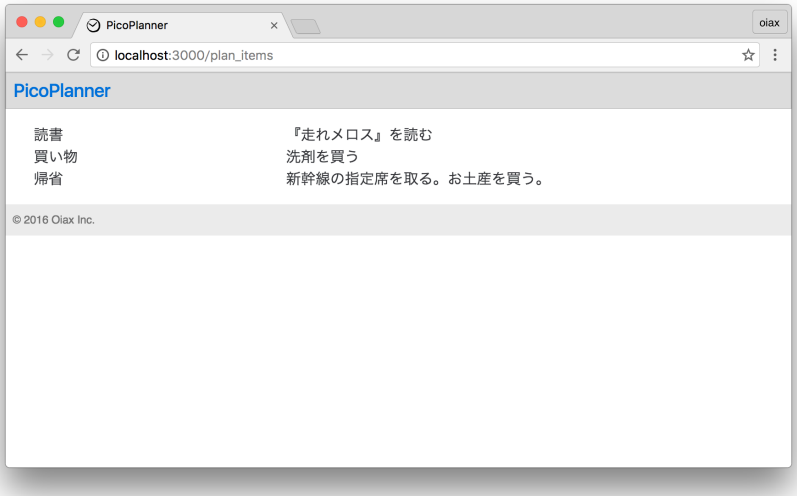

図 8.1　グリッドシステム導入後の予定表（PC 向けレイアウト）

しかし、Chrome のデベロッパーツールを用いてスマートフォン用の表示に切り替えると 図 8.2 のように件名と説明が縦並びに表示されます。

図 8.2　グリッドシステム導入後の予定表（スマートフォン向けレイアウト）

8.2　グリッドシステムとは

レスポンシブウェブデザインを実現する仕組みを、一般に**グリッドシステム**（grid system）と呼びます。英語の“grid”は「格子」を意味する名詞です。ページのある領域を碁盤目状の小区画に分割し、そこに要素を配置する仕組みです。

Bootstrap のグリッドシステムの特徴は、水平方向の小区画の分割数が 12 に固定されていることです。格子の中に配置される要素の横幅は、1 から 12 までの整数で表現されます。

さきほど書き換えたテンプレートの 4 行目に次のような記述があります。

```
    <div class='col-12 col-md-4'>
```

`class` 属性の中で使われている `12` や `4` という数字がこの `div` 要素の横幅を表しています。

さて、`class` 属性には `col-12` と `col-md-4` という二つの値が指定されています。先頭の `col` は “column” の略で「列」を表します。注目すべきはハイフン記号ではさまれた `md` という文字列です。これらは**レスポンシブブレイクポイン**

ト（responsive breakpoint）を表す略称です。

レスポンシブブレイクポイントは単に**ブレイクポイント**（breakpoint）とも呼ばれ、ブラウザの表示幅に名前を付けたものです。Bootstrap では、表 8.1 に示す 4 個のブレイクポイントが定義されています。

表 8.1　Bootstrap のレスポンシブブレイクポイント

正式名称	略称	ブラウザの表示幅
Small	sm	576px
Medium	md	768px
Large	lg	992px
Extra Large	xl	1,200px

Bootstrap 3 では xs、sm、md、lg という 4 個のブレイクポイントが定義されていましたが、Bootstrap 4 では xs が廃止され、新たに xl が新設されました。なお、Bootstrap 4.0.0-alpha.5 までは、xs という名前のブレイクポイントが存在していたので、ネット等で情報を調べるときには注意が必要です。

ある要素の class 属性に col-12 と col-md-4 という二つの値が指定されている場合、この要素の横幅はブラウザの表示幅により次のように変化します。

- ブラウザの表示幅が md（768px）未満であれば、小区画 12 個分の横幅
- ブラウザの表示幅が md（768px）以上であれば、小区画 4 個分の横幅

ブレイクポイント指定のないクラス col-12 により、すべての表示幅に対するデフォルトの横幅が 12 と決まります。そして、ブレイクポイント指定のある col-md-4 により、ブラウザの表示幅が md（768px）以上の場合は、横幅が 4 となります。なぜなら、ブレイクポイント lg と xl についてはクラス指定がないからです。

一般に、4 個のブレイクポイントのうちクラス指定のないものについては、下位ランクのブレイクポイントの指定が適用されることになります。もし、下位ランクのブレイクポイントについてもクラス指定がなければ、col-12 のようなブレイクポイント指定のないクラスが適用されます。

8.3　ブレイクポイントの使い方

さて、テンプレートの 7 行目に次のような記述があります。

```
    <div class='col-12 col-md-8'>
```

ここではブレイクポイント md（768px）以上の表示幅を持つブラウザにおいて、この div 要素が小区画 8 個分の横幅を持つように指定しています。なぜ 8 という値を使用しているかといえば、予定の件名の表示幅に 4 を指定したからです。合計が 12 になるようにすれば、ちょうど 1 行に収まります。

ややこしくなってきましたね。次の例をご覧ください。

```
<div class='container-fluid'>
  <div class='row'>
    <div class='col-12 col-md-4'>A</div>
    <div class='col-12 col-md-8'>B</div>
  </div>
</div>
```

これをブラウザで表示した時の表示を模式的に表したのが図 8.3 です。左側が md（768px）未満の幅（スマホモード）のブラウザでの表示、右側が md（768px）以上の幅（通常モード）のブラウザでの表示です。

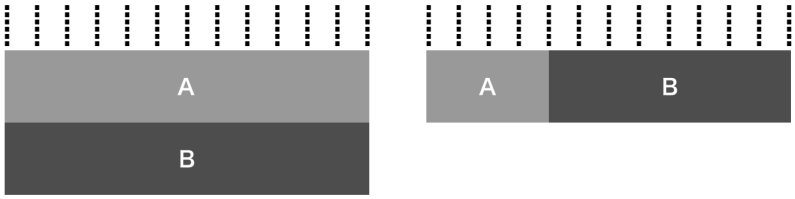

図 8.3　ブラウザの表示幅によるレイアウトの違い

ブレイクポイントで思考を切り替えて要素を配置していくのが、成功の秘訣です。

8.4 コンテナの表示幅

Bootstrap ではグリッドシステムが適用される範囲を**コンテナ**（container）と呼びます。コンテナは行（row）の集合であり、各行は列（column）の集合です。また、グリッドシステムの構成要素（コンテナ、行、列）すべてをまとめて**グリッド**（grid）と呼びます。

グリッドシステムのコンテナとなる要素の `class` 属性には `container` または `container-fluid` という値を指定します。英語の “fluid” は「流動性の」という意味の形容詞です。本書では、この属性に `container` が指定されたグリッドを**固定グリッド**、`container-fluid` が指定されたグリッドを**流動グリッド**と呼ぶことにします。

固定グリッドのコンテナには、ブレイクポイント別の `max-width` プロパティが設定されます。そのため、ブラウザの幅を徐々に広げていっても、あるブレイクポイントから次のブレイクポイントまでの範囲内であればコンテナの幅が一定に保たれます。そして、ブレイクポイントで非連続的にコンテナの幅が変化します。

ブラウザの表示幅別の具体的な `max-width` プロパティの値を表 8.2 にまとめます。

表 8.2　固定グリッドのコンテナの最大幅

ブラウザの表示幅	コンテナの最大幅
`sm`（576px）以上、`md`（768px）未満	540px
`md`（768px）以上、`lg`（992px）未満	720px
`lg`（992px）以上、`xl`（1,200px）未満	960px
`xl`（1,200px）以上	1,140px

流動グリッドのコンテナには `max-width` プロパティが設定されません。すなわち、コンテナは常にブラウザの表示幅いっぱいに広がります。

8.5 行に独自スタイルを設定する

コンテナのすぐ内側の HTML 要素を行（row）と呼びます。その `class` 属性には `row` という値を指定します。

行の左右のマージンには -15px という値が設定されています。つまり、左右に15px ずつ張り出す（飛び出る）わけですが、コンテナの左右のマージンに 15px という値が設定されているので、見かけ上は行の左右の端がコンテナの端と揃います。このようにグリッドシステムの背後には精密な CSS 設計が存在しますので、行や列に独自のスタイルを適用する場合は慎重に行う必要があります。

ちょっとやってみましょう。まず、コンテナの class 属性に plan-items という値を追加します。

app/views/plan_items/index.html.erb

```
1 - <div class='container-fluid'>
1 + <div class='container-fluid plan-items'>
:
```

そして、新規の SCSS ファイル plan_items.scss を次の内容で作成してください。

app/assets/stylesheets/plan_items.scss (New)

```
1  div.plan-items {
2    div.row {
3      margin-bottom: 0.5rem;
4      background-color: #ddd;
5    }
6    div.row:last-child {
7      margin-bottom: 0;
8    }
9  }
```

行と行の間に `0.5rem`（通常は `8px` に相当）のスペースを挿入し、行の背景色を薄い灰色（`#ddd`）に設定しています。

6 行目で使われている CSS セレクタ `div.row:last-child` に着目してください。コロン記号（`:`）の右側の `last-child` は**擬似クラス**（pseudo-class）の一種で、「自身の親要素の最後の子要素」にマッチします。ここでは「自身の親クラス」はグリッドシステムのコンテナを指すので、要するに「最後の行」にマッチすることになります。各行の下辺に `0.5rem` のマージンを指定するけれど、最後の行に関しては下辺のマージンを `0` とする、というわけです。

ブラウザを再読込すると、PC ブラウザでは 図 8.4 のようになります。

図 8.4　独自スタイル適用後の予定表（PC 向けレイアウト）

スマートフォンでの表示は 図 8.5 のようになります。

図 8.5　独自スタイル適用後の予定表（スマートフォン向けレイアウト）

8.6　上下のパディングを設定する

ブラウザでの表示を見ると、行の内側が少し詰まりすぎているようです。上下に `0.5rem` のパディングを設定しましょう。

app/assets/stylesheets/plan_items.scss

```
1    div.plan-items {
2      div.row {
3 +      padding-top: 0.5rem;
4 +      padding-bottom: 0.5rem;
5        margin-bottom: 0.5rem;
:
```

ブラウザを再読込すると、PC ブラウザでは 図 8.6 のようになります。

図 8.6　パディング設定後の予定表（PC 向けレイアウト）

スマートフォンでの表示は 図 8.7 のようになります。

図 8.7　パディング設定後の予定表（スマートフォン向けレイアウト）

8.7　フォントの太さとサイズの調整

続いて、フォントの太さとサイズを調整します。まず、予定の件名と説明が埋め込まれる部分を span タグで囲んで、class 属性を指定します。span は、スタイルを変更する範囲を限定するために主に用いられる HTML の要素です。

app/views/plan_items/index.html.erb

```
:
      <div class='col-12 col-md-4'>
-       <%= item.name %>
+       <span class='plan-item-name'><%= item.name %></span>
      </div>
      <div class='col-12 col-md-8'>
-       <%= item.description %>
+       <span class='plan-item-description'><%= item.description %></span>
      </div>
:
```

そして、plan_items.scss を次のように書き換えます。

app/assets/stylesheets/plan_items.scss

```
:
    margin-bottom: 0.5rem;
    background-color: #ddd;
+   span.plan-item-name {
+     font-weight: bold;
+   }
+   span.plan-item-description {
+     font-size: 0.75rem;
+   }
  }
:
```

font-weight は、フォントの太さを指定するための CSS プロパティです。値として bold を指定すれば「太字」で表示されるようになります。

実際にテキストが太字で表示されるかどうかは、フォントの種類によります。太字のデータを備えていないフォントの場合には、font-weight プロパティは効果がありません。

ブラウザを再読込すると、PC ブラウザでは 図 8.8 のようになります。

図 8.8　フォント調整後の予定表（PC 向けレイアウト）

スマートフォンでの表示は 図 8.9 のようになります。

図 8.9　フォント調整後の予定表（スマートフォン向けレイアウト）

8.8　改行への対応

ここまでビジュアルデザインを作りこんできたところで、筆者は PicoPlanner の仕様にちょっとした検討漏れのあることに気付きました。たぶん、ユーザーは予定の説明の途中で改行したいのではないでしょうか。しかし、現行の実装では改行されません。

そのことを確かめるため、シードデータを作りなおしましょう。

db/seeds.rb

```
  :
17  item = PlanItem.new
18  item.name = '帰省'
19 - item.description = '新幹線の指定席を取る。お土産を買う。'
19 + item.description = "新幹線の指定席を取る。\nお土産を買う。"
20  item.save!
```

バックスラッシュ（\）と n の組み合わせで「改行文字」を表します。ただし、この表記法を利用するためには、文字列をダブルクォートで囲む必要があります。

Ctrl-C を入力して、PicoPlanner を終了してください。そして、データベー

スを作りなおして、シードデータを再投入します。

```
$ rails db:reset
```

rails s コマンドで PicoPlanner を起動し、ブラウザで予定表ページを表示してください（図 8.10）。

図 8.10　改行対応前の予定表（PC 向けレイアウト）

「お土産を買う」の左に小さなスペースが現れましたが、改行されていませんね。HTML では改行文字はスペース文字と同じ扱いになります。表示上も改行したいなら、強制的に改行してくれる
 タグを使う必要があります。

そこで、plan_items#index アクションのテンプレートを次のように書き換えてください。

app/views/plan_items/index.html.erb

```
:
7       <div class='col-12 col-md-8'>
8 -       <span class='plan-item-description'><%= item.description %></span>
8 +       <span class='plan-item-description'>
9 +         <% item.description.split("\n").each do |line| %>
10 +          <%= line %><br>
```

```
11 +          <% end %>
12 +        </span>
13        </div>
 :
```

文字列オブジェクトの split メソッドは、引数に指定した値を区切り文字として文字列を分割し、配列を返します。次の例をご覧ください。

```
s = 'abc;def;ghi'
a = s.split(';')
```

このコードを評価すると変数 a には、['abc', 'def', 'ghi'] という配列がセットされます。テンプレートの中では、改行文字（"\n"）で分割して配列にしてから、各断片を
 タグ付きでページに埋め込んでいます。

ブラウザを再読込すると、図 8.11 のようになります。

図 8.11　改行対応後の予定表（PC 向けレイアウト）