第 17 章

画像

本章では、Web アプリ上に写真やイラストなどの画像を表示する方法と、Web アプリにファビコン（ブックマークやショートカット用のアイコン）を設定する方法について解説します。

17.1 `img` 要素

HTML 文書に**画像**（image）を挿入するには `img` 要素を使用します。この要素の `src` 属性には画像ファイルの URL を、`alt` 属性には**代替文字列**（alternative text）を指定します。代替文字列は、何らかの理由で画像ファイルにアクセスできない場合に画像の代わりにブラウザの画面に表示されます。音声読み上げブラウザも代替文字列を利用します。

HTML 文書では **PNG**、**JPEG**、**GIF**、**SVG** という 4 種類のファイル形式の画像がよく使われます。通常、ファイルの拡張子には、`png`、`jpeg`、`gif`、`svg` 等ファイル形式の名前を小文字にしたものを使います。ただし、JPEG 形式のファイルの拡張子には `jpg` もよく使われます。

各ファイル形式の特徴を箇条書きでまとめます：

PNG フルカラー（約 1,677 万色）の画像を扱える。透過処理が可能。

JPEG フルカラー（約 1,677 万色）の画像を扱える。写真向き。

GIF 最大 256 色の画像しか扱えないが、透過処理とアニメーション処理が可能。

SVG "Scalable Vector Graphics" の略。拡大・縮小しても品質が劣化しない。

次に示すのは、アメリカ航空宇宙局（NASA）が配布している地球の画像を表示する img 要素の記述例です。

```
<img src="https://www.oiax.jp/books/files/nasa_earth.jpg" alt="地球の写真">
```

img 要素に width 属性と height 属性に値を指定することにより、画像がブラウザ上に表示される幅と高さを調節できます。これらの属性には幅と高さのピクセル数を「100」のような整数値で指定します。

```
<img src="https://www.oiax.jp/books/files/nasa_earth.jpg" alt="地球の写真"
  width="240" height="240">
```

17.2 tag 関数と static_path 関数

Phoenix アプリ上に画像を表示する場合、img 要素を直接記述する代わりに tag 関数と static_path 関数を使用して img 要素を生成するのが筆者のお勧めです。

画像ファイルは web/static/assets/images ディレクトリに置きます。ターミナルで次のコマンドを実行してください。

```
$ cd ~/projects/modest_greeter
$ wget https://www.oiax.jp/books/files/modest_greeter.svg
$ mv modest_greeter.svg web/static/assets/images
```

初期状態の OS X には *wget* コマンドが備わっていません。あらかじめ brew install wget コマンドでインストールしてから利用してください。

そして、HelloController.show アクション用のテンプレートを次のように書き換えます。

web/templates/hello/show.html.eex

```
1   <div class="card m-3">
2 +   <%= tag :img, src: static_path(@conn, "/images/modest_greeter.svg"),
3 +     alt: "Robot", width: "320", height: "320",
4 +     class: "card-image-top mx-auto" %>
5 +   </div>
6     <div class="card-body text-white bg-success">
:
```

tag 関数の第 1 引数にはタグ名をアトムで指定します。第 2 引数にはキーワードリストでオプションを指定します。このオプションには HTML 属性とその値を指定します。src オプションには画像の URL、alt オプションに代替文字列、width オプションと height オプションに画像の幅と高さをピクセル単位で指定します。

class オプションには、card-image-top と mx-auto を指定しました。前者は、Card コンポーネントの本体（body）の上に載せる画像に対して指定するクラスです。後者の mx-auto は、要素を中央寄せにするクラスです。

> mx-auto クラスの名前に含まれる m はマージン（margin）の頭文字で、x は X 軸方向、すなわち左右の辺が対象であることを示しています。左右の辺のマージンに auto という特別な値が設定されると、要素の左右に「（ブラウザによって）自動的に調節される空間」が追加されます。その結果、要素が中央寄せで表示せれることになります。

static_path 関数には、「Plug.Conn 構造体」の @conn と、画像ファイルのパスを指定します。このパスは、web/static/assets ディレクトリからの相対パスですが、先頭にスラッシュ（/）を付け加えてください。

ブラウザで ModestGreeter のトップページで「Alice」リンクをクリックすると、図 17.1 のような画面が表示されます。

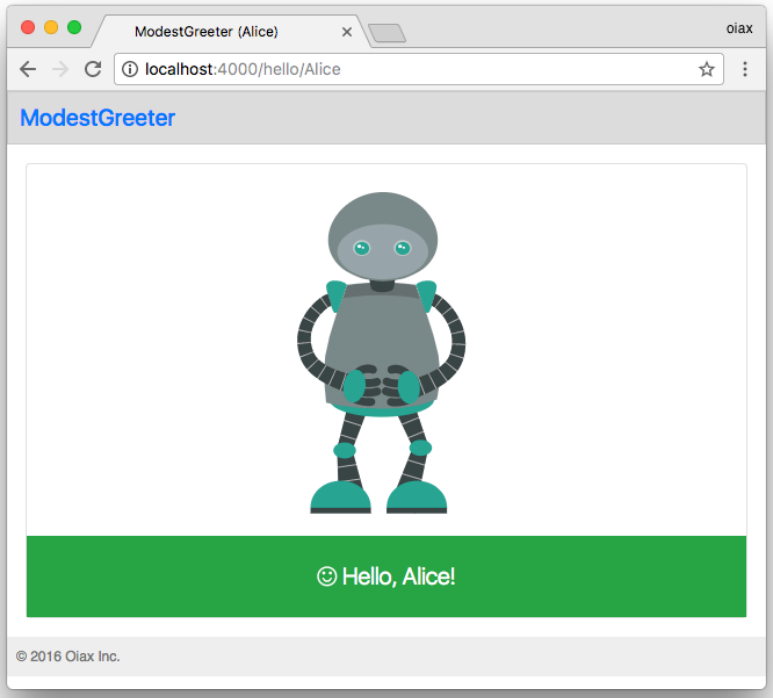

図 17.1　メッセージの上にロボットの画像を表示

17.3　画像表示幅の調節

実は、さきほど見たページを iPhone 5 で表示すると 図 17.2 左側のようにロボットのイラストが少し右寄りになります。

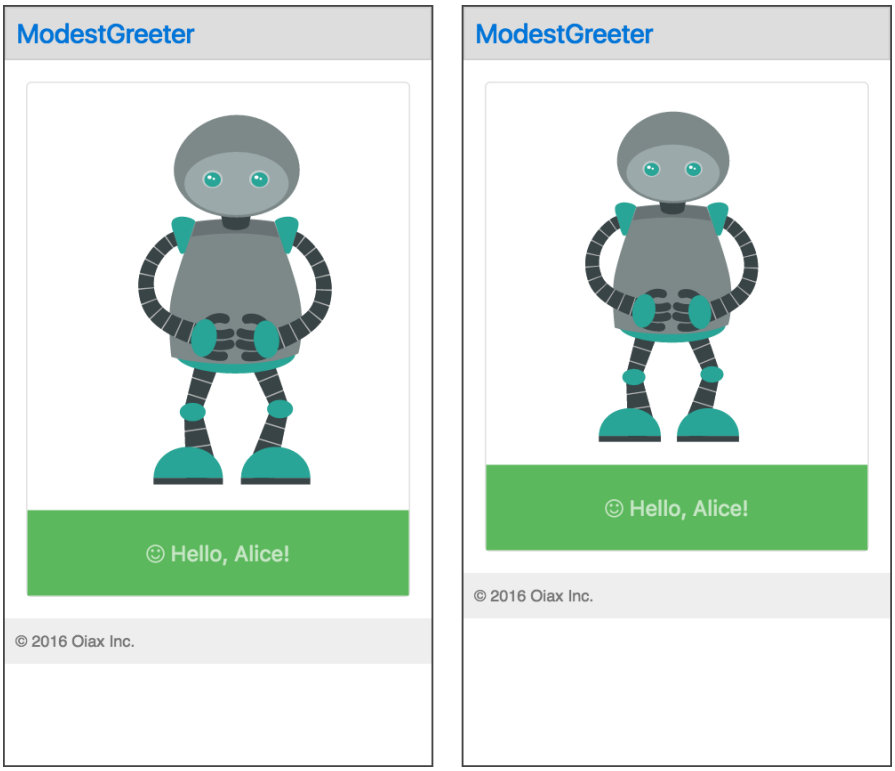

図 17.2　スマートフォン向けに画像の表示幅を調節

iPhone 5 の画面は 320px 幅の画像がちょうど収まる大きさなので（縦長表示の場合）、Card コンポーネントの内側に収まらないのです。そこで、show.html.erb を次のように修正します。

web/templates/hello/show.html.eex

```
:
<div class="card m-3">
  <%= tag :img, src: static_path(@conn, "/images/modest_greeter.svg"),
-   alt: "Robot", width: "320", height: "320",
+   alt: "Robot", style: "width: 100%; max-width: 320px",
    class: "card-image-top mx-auto" %>
  <div class="card-body text-white bg-success">
:
```

tag 関数から width オプションと height オプションを取り去り、代わりに style オプションに CSS でスタイルを指定しました。まず、width プロパティに 100% という値を指定することで、画像が親要素の幅いっぱいに表示されるようにします。これで iPhone 5 のような表示幅の狭いスマートフォンで画像が中央に揃います（図 17.2 右側）。

ただし、この指定だけでは幅の広いブラウザで表示した時にイラストが大きくなりすぎます。そこで、max-width プロパティに 320px という値を指定しました。このプロパティは要素の表示幅の最大値を示します。

■コラム: 画像ファイルのダイジェスト

図 17.1 を開いている状態でブラウザの「ページのソースを表示」機能を使って調べると、画像を表示する部分のコードが次のようになっています。

```
<img alt="Robot" src="/images/modest_greeter.svg"
  style="width: 100%; max-width: 320px">
```

しかし、『Ruby on Rails 初級①』で作った SimpleGreeter で対応する箇所を調べると次のようになります。ただし、src 属性の値は長すぎるので一部省略しています（... の部分）。

```
<img alt="Robot" src="/assets/robot-e31c3ae...f5fa5a1.svg"
  style="width: 100%; max-width: 320px"/>
```

Rails はキャッシュ制御のために画像ファイルからフィンガープリントという数値を計算し、自動的にファイル名に加えます。Phoenix にも同様の仕組みが用意されているのですが、開発モード（dev 環境）では無効であるため、画像ファイルの名前は変化しません。

付録 E の説明に沿って、ModestGreeter を本番モード（prod 環境）で起動してみてください。画像ファイルの名前にフィンガープリントが加えられていることが確認できるはずです。なお、Phoenix ではフィンガープリントのことを**ダイジェスト**（digest）と呼びます。

17.4 ファビコンと Web クリップアイコンの指定

本巻の締めくくりとして、ModestGreeter にファビコンと Web クリップアイコンを設定しましょう。

ファビコン（favicon）とは、ブラウザで特定の Web サイトを開いた時にタブに表示される小さなアイコンです。また、ブラウザのブックマークやお気に入りをリスト表示する際にもこのアイコンが使われます。他方、**Web クリップアイコン**は、スマートフォンのブラウザで Web サイトをホーム画面に追加した時に表示されるアイコンです。

ターミナルで次のコマンドを実行してください。

```
$ wget https://www.oiax.jp/books/files/modest_greeter.ico
$ wget https://www.oiax.jp/books/files/modest_greeter256.png
$ mv modest_greeter.ico web/static/assets/images
$ mv modest_greeter256.png web/static/assets/images
```

ファビコンは **ICO 形式**の画像ファイルです。拡張子は ico とします。Web クリップアイコンには、256 ピクセルの幅と高さを持つ JPEG 形式または PNG 形式の画像ファイルを用意してください。

次に、レイアウトテンプレートを次のように書き換えます。

web/templates/layout/app.html.eex

```
  :
10      <title><%= @view_module.document_title assigns %></title>
11      <link rel="stylesheet" href="<%= static_path(@conn, "/css/app.css") %>">
12 +    <link rel="shortcut icon" type="image/x-icon"
13 +      href="<%= static_path(@conn, "/images/modest_greeter.ico") %>">
14 +    <link rel="apple-touch-icon" type="image/x-icon"
15 +      href="<%= static_path(@conn, "/images/modest_greeter256.png") %>">
16    </head>
  :
```

HTML の link 要素の役割は、HTML 文書に関連するファイルを指定することです。必ず head 要素の中に置かれます。CSS ファイルを読み込むために用い

られるほか、上記の例のようにファビコンや Web クリップアイコンを指定するのにも用いられます。なお、link 要素は空要素の一種です。

ブラウザのタブにファビコンが表示されます（図 17.3）。

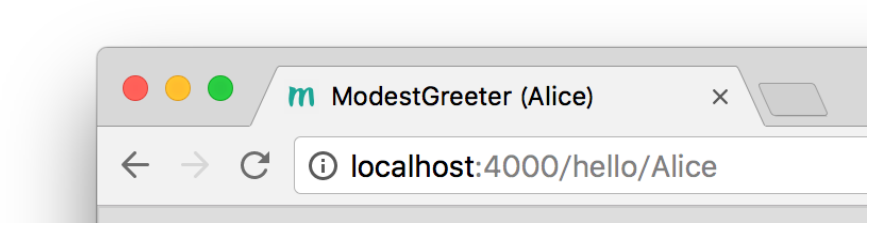

図 17.3　タブにファビコンを表示

ブラウザはファビコンのデータをキャッシュに保存するため、ファビコンファイルを置き換えても表示が変わらないことがあります。その場合は、ブラウザ自体を完全に終了して起動しなおしてください。

■コラム: ファビコンファイルの置き場所

『Ruby on Rails 初級①』では、ファビコンファイルを app/assets ディレクトリの直下に配置しました。Rails ではこのディレクトリ全体がアセットパイプライン処理の対象になるのでこれでよかったのですが、Phoenix では、 app/assets ディレクトリにある特定のサブディレクトリ（css, fonts, images, js）にアセットファイルを置く必要があります。ただし、favicon.ico と robots.txt というふたつのファイルに関しては app/assets ディレクトリ直下に置けます。

この設定を変更するには、lib/modest_greeter ディレクトリにある endpoint.ex ファイルを書き換えます。

lib/modest_greeter/endpoint.ex

```
 :
10    plug Plug.Static,
11      at: "/", from: :modest_greeter, gzip: false,
12      only: ~w(css fonts images js favicon.ico robots.txt)
 :
```

12 行目の括弧の中にアセットファイルを設置できるディレクトリとファイルの名前が列挙されています。

17.5 content_tag 関数

現在、レイアウトテンプレートの 11 行目は、次のように記述されています。

```
<link rel="stylesheet" href="<%= static_path(@conn, "/css/app.css") %>">
```

12 ～ 15 行にも同様の記述があります。筆者は、HTML タグの属性値の内側に <%= と %> で値が埋め込まれるのが好きではありません。ちょっと読みにくいように思えるからです。そこで、tag 関数を使って書き換えることにします。

web/templates/layout/app.html.eex

```
 :
10      <title><%= @view_module.document_title assigns %></title>
11 -    <link rel="stylesheet" href="<%= static_path(@conn, "/css/app.css") %>">
11 +    <%= tag :link, rel: "stylesheet",
12 +      href: static_path(@conn, "/css/app.css") %>
12 -    <link rel="shortcut icon" type="image/x-icon"
13 -      href="<%= static_path(@conn, "/images/modest_greeter.ico") %>">
13 +    <%= tag :link, rel: "shortcut icon", type: "image/x-icon",
14 +      href: static_path(@conn, "/images/modest_greeter.ico") %>
14 -    <link rel="apple-touch-icon" type="image/x-icon"
15 -      href="<%= static_path(@conn, "/images/modest_greeter256.png") %>">
15 +    <%= tag :link, rel: "apple-touch-icon", type: "image/x-icon",
16 +      href: static_path(@conn, "/images/modest_greeter256.png") %>
17    </head>
 :
```

また、レイアウトテンプレートの 29 行目も同様に書き換えます。ただし、script 要素は空要素ではないので、content_tag 関数を用いてタグを生成します。

web/templates/layout/app.html.eex

```
 :
29 -     <script src="<%= static_path(@conn, "/js/app.js") %>"></script>
29 +     <%= content_tag :script, "", src: static_path(@conn, "/js/app.js") %>
30     </body>
31   </html>
```

content_tag 関数の第 2 引数に空の文字列（""）を指定する点に注意してください。

以上で、本書『Elixir/Phoenix 初級①』はおしまいです。