React/Rails アプリケーション構築ガイド

リリース ***alpha***

2018 年 **05** 月 **23** 日

目次

React と **Ruby on Rails** を利用する人のための、**Web** アプリケーション構築ガイド

最終更新日: 2018 年 05 月 19 日

現在執筆中です

React + Rails での Web アプリケーション構築方法について解説します。

本ガイドは以下読者を想定しています。

- React + Rails で Web アプリケーションを構築したいと考えているエンジニア
- React 初学者
- Rails 初学者

第 1 章

React/Rails 入門編

1.1 はじめに

1.1.1 本ガイドの想定読者と前提

本ガイドは React と Ruby on Rails を用いて Web アプリケーションを構築したいと考えている以下読者を想定しています。

- React + Rails で Web アプリケーションを構築したいと考えているエンジニア
- React 初学者
- Rails 初学者

本ガイドを読み進めるにあたり、以下ソフトウェア・フレームワークが利用可能な状態になっていることを前提としています。

- Ruby 2.5.0 以上
    - bundler が利用可能なこと
- Nodejs v9.0 以上
    - npm / yarn が利用可能なこと
- Mysql 5.7 以上

本ガイドで利用する技術スタックは以下のとおりです。

- 言語
    - Ruby v2.5
    - Javascript ES6
- フレームワーク
    - React v16
    - Redux v4.0

    - Ruby on Rails v5.2
- DB
    - Mysql 5.7
- その他ツール
    - Webpacker (Webpack)
    - yarn

Ruby と Javascript、および DB まわりの経験が全くない場合は、本ガイドを読む前に別の書籍・リソースで学習をすすめることをおすすめします。

1.1.2 Rails とは何か

1.1.3 React とは何か

1.1.4 ES6 とは何か

1.1.5 Webpack / Webpacker とは何か

1.1.6 Yarn とは何か

1.1.7 Redux とは何か

1.1.8 Single Page Application (SPA) とは何か

1.1.9 Server Side Rendering (SSR) とは何か

1.2 Ruby on Rails 入門

1.2.1 Rails で使われる概念

Rails でプログラムを作り始める前に、Rails に持ち込まれているいくつかの概念を説明します。

Rails の MVC

MVC とは Model/View/Controller の略です。MVC モデルでは、プログラムをモデル、ビュー、コントローラの 3 つに分けておくことで、処理の見通しがよくなります。Rails で Web アプリケーションを作る場合、基本的に MVC に従ってプログラムを作成する必要があります。

- モデル (Model)
    - モデルはビジネスロジック (=システム固有の処理) を表現するために利用します

    - データの保持、データストアへのアクセスにも利用します
- ビュー (View)
    - モデルやコントローラから受け取ったデータを元に、ユーザに見える形のデータを生成します
- コントローラ (Controller)
    - 入力イベントを受け付けてモデル、ビューの操作をし、結果をブラウザに返します

注釈: Rails の MVC は、View から直接モデルのビジネスロジックやデータストアにアクセスすることが可能です。この点が一般的な MVC アーキテクチャとは異なります。このアーキテクチャを MVC と呼んでもよいかは議論の余地がありますが、Rails の MVC はこういうものだと理解してください。

ブログサイトを例として、Rails の MVC を具体的に説明します。このサイトでは http://0.0.0.0:3000/articles にアクセスすると記事一覧を返します。

Rails では以下 4 つのコードを作成する必要があります。

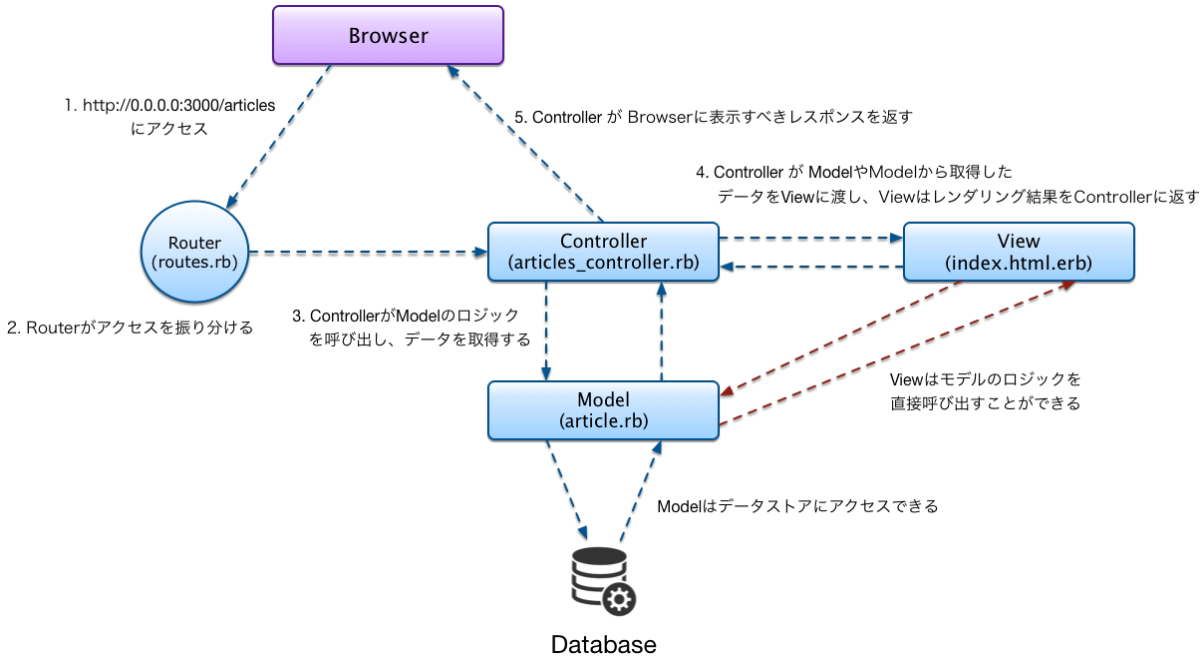
- ルーティング
    - ルーティングはブラウザからアクセスが来た際、URL からどのコントローラに処理を引き渡すべきかを判定します
    - /articles にアクセスされたら 記事一覧を取得するコントローラのアクションに処理を渡します
- モデル
    - ブログの記事一覧を DB から取得します
- ビュー
    - ブログの記事一覧の HTML を生成します
- コントローラ
    - モデルから記事一覧を取得し、ビューに結果を引き渡して HTML を取得し結果をブラウザに返します

上記 4 つを Rails で記述すると次のようになります。

- config/routes.rb

```
# config/routes.rb

Rails.application.routes.draw do
  # /articles にアクセスすると
  # articles_controller の index アクションに処理を渡す
  resources :articles, only: [:index]
end
```

- app/models/article.rb




```
# app/models/article.rb
#
# == Schema Information
#
# Table name: articles
#
#  id          :bigint(8)        not null, primary key
#  title       :string(255)
#  description :text(65535)
#  created_at  :datetime         not null
#  updated_at  :datetime         not null
#

class Article < ApplicationRecord
end
```

- app/controllers/articles_controller.rb

```
# app/controllers/articles_controller.rb
class ArticlesController < ApplicationController
  def index
    # Article モデルは、はじめから all という DB の articles テーブルから全てのデータを取得するメソッドをもっています
    @articles = Article.all
  end
end
```

- app/views/articles/index.html.erb

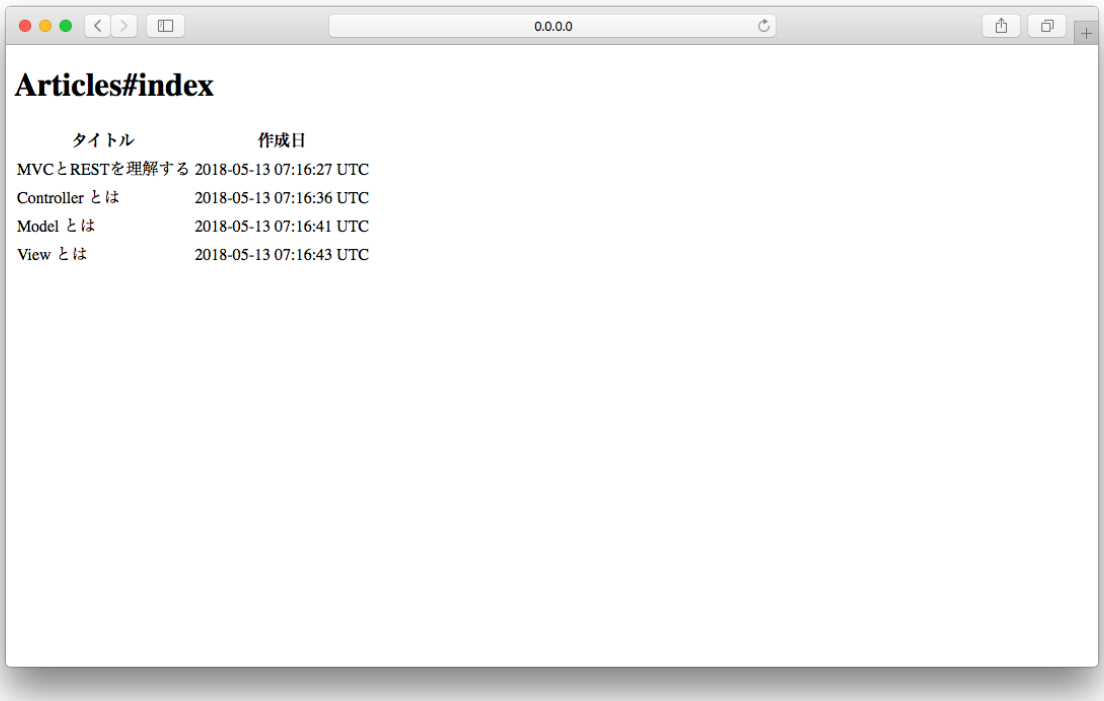
```
<h1>Articles#index</h1>
```

(次のページに続く)

(前のページからの続き)

```
<table>
  <thead>
    <th>タイトル</th>
    <th>作成日</th>
  </thead>
  <tbody>
    <% @articles.each do |article| %>
    <tr>
      <td>
        <%= article.title %>
      </td>
      <td>
        <%= article.created_at %>
      </td>
    </tr>
    <% end %>
  </tbody>
</table>
```

ブラウザから http://0.0.0.0:3000/articles にアクセスした結果は次のようになります。



REST

Rails でルーティング、コントローラを定義する前に、REST について理解しておく必要があります。なぜなら、Rails は基本的に REST の原則にしたがってコードを簡単に定義できるようにできているからです。

REST の特徴は以下のとおりです。

- すべてをリソースで表す
- リソースの参照、操作は URI により参照・更新することができる
- ステートレスである

RESTful なルーティングでは、あるリソースをどのように操作するかを、URI や HTTP のメソッドで表現します。HTTP メソッドは以下を利用します。

- GET
    - データの取得
- POST
    - データの作成
- PATCH (PUT)
    - データの更新
- DELETE
    - データの削除

具体例をあげます。

1.2.2 ブログづくりで学ぶ CRUD アプリケーション

基本概念を学んだところで、次は簡単なブログアプリケーションを作ってみましょう。

CRUD とは

CRUD とは Create/Read/Update/Delete の略称です。多くの Web アプリケーション (特に管理系のアプリケーション) は CRUD で構成されています。

ブログアプリケーションの場合、記事は以下 CRUD を持ちます。

- Create
    - ブログ記事の新規作成
- Read
    - ブログ記事一覧の表示・ブログ記事の表示
- Update
    - 作成したブログ記事の更新
- Destroy
    - 作成したブログ記事の削除

Rails アプリケーションの新規作成

Rails で新しいブログアプリケーションを作成しましょう。データベースには mysql を利用しますので、あらかじめ mysql が使える状態にしておいてください。

まずは rails コマンドがコンソールで使える状態にします。次の gem install コマンドで rails をインストールします。

```
gem install rails
```

rails 導入後に次のコマンドで新規 rails アプリケーションを作成します。

```
rails new hello_blog --database=mysql
```

hello_blog はアプリケーションの名前です。アプリケーションには好きな名前をつけましょう。

アプリケーションサーバを起動する

新しい Rails アプリケーションを作成したので、アプリケーションサーバを起動してみましょう。作成した hello_blog ディレクトリ内に移動し、ターミナルから次のコマンドを入力します。

```
cd hello_blog  # 新しく作成した hello_blog プロジェクト内に移動する
bundle exec rails server
```

コマンドを実行するときは、先頭に bundle exec コマンドをつけましょう。bundle exec をつけると、プロジェクト内の Gemfile に指定した環境でコマンドを実行することができます。

注釈: rails コマンドの場合は bundle exec をつけてもつけなくても挙動は変わりません。すなわち、bundle exec rails server と bin/rails server 、rails server は全く同じ意味になります。これは rails コマンドの場合、Gemfile に指定されたバージョンを判別してコマンドを実行してくれるためです。

Rails5.2 では、Puma というアプリケーションサーバが起動します。サーバを起動したら、ブラウザから http://0.0.0.0:3000 にアクセスします。次の画面が表示されたら、正常にアプリケーションサーバが起動できています。

アプリケーションサーバを停止するには、 Ctrl + C を押します。

記事一覧表示用コントローラとビューを作成する

http://0.0.0.0:3000/articles/index にアクセスしたら、ブログ記事一覧をブラウザに表示させたいところですが、まずはブラウザ上に Hello World を表示するプログラムを作りましょう。この機能を実現するためには、ルーティングとコントローラとビューを作成する必要があります。

まずはコントローラとビューを作成しましょう。コントローラとビューを 1 から全て記述して作成することもできますが、Rails では次のコマンドでコントローラとビューの初期テンプレートを自動生成することができます。


```
bundle exec rails generate controller articles index
```

rails generate controller コマンドでコントローラを自動生成します。articles は コントローラの名前です。Rails では基本的にコントローラ名は複数形で定義します。

index は生成したいアクション名です。アクションは基本的に index/show/new/create/edit/update/destroy の 7 つを使います。リソースの一覧を表現するときには、index を利用します。

コマンドを実行すると、次のようにファイルが自動生成されます。

```
$ bundle exec rails generate controller articles index

      create  app/controllers/articles_controller.rb
       route  get 'articles/index'
      invoke  erb
      create    app/views/articles
      create    app/views/articles/index.html.erb
      invoke  test_unit
      create    test/controllers/articles_controller_test.rb
      invoke  helper
      create    app/helpers/articles_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/articles.coffee
      invoke    scss
      create      app/assets/stylesheets/articles.scss
```

多くのファイルが自動的に生成されましたが、今回見るべきなのはルーティングとコントローラ、そしてビューの部分です。

```
create app/controllers/articles_controller.rb
```

と出力されていますが、これは articles controller が自動的に作成されたことを表しています。

```
route get 'articles/index'
```

という出力がありますが、これは config/routes.rb にルーティングの定義が自動追加されたということを表しています。ルーティングについては、次節で詳しく学びます。

```
create    app/views/articles/index.html.erb
```

という出力は、app/views/articles/index.html.erb が自動生成されたことを表しています。

次にコントローラとビューの定義を詳しくみていきましょう。

app/controllers/articles_controller.rb は次のように定義されています。

```
class ArticlesController < ApplicationController
  def index
  end
end
```

Rails のコントローラは必ず ApplicationController を継承する必要があります。

次に index ですが、特に何も定義はありません。今回はブラウザに Hello World と表示させたいので、index アクションでビューを呼び出す必要があります。よって次のようなコードでも動かすことができます。

```
class ArticlesController < ApplicationController
  def index
    # app/views/articles/index.html.erb にあるビューを呼び出す
    render "articles/index.html.erb"
  end
end
```

しかし、**render "articles/index.html.erb" の行は記述しなくても動作します。**

これは、アクションの最後に明確なビュー指定がない場合は、Rails が自動的に app/views/articles/index.html.erb を探して出力してくれるためです。ただし、コントローラ名/アクションとビューのファイルパスは合わせておく必要があります。

ArticlesController#index (= ArticlesController の index アクション) ならば、render 省略時は app/views/articles/index.html.erb を自動的に探して利用してくれます。例えば ProductsController#new というコントローラ/アクションを定義した場合は、render 省略時は app/views/products/new.html.erb を利用します。

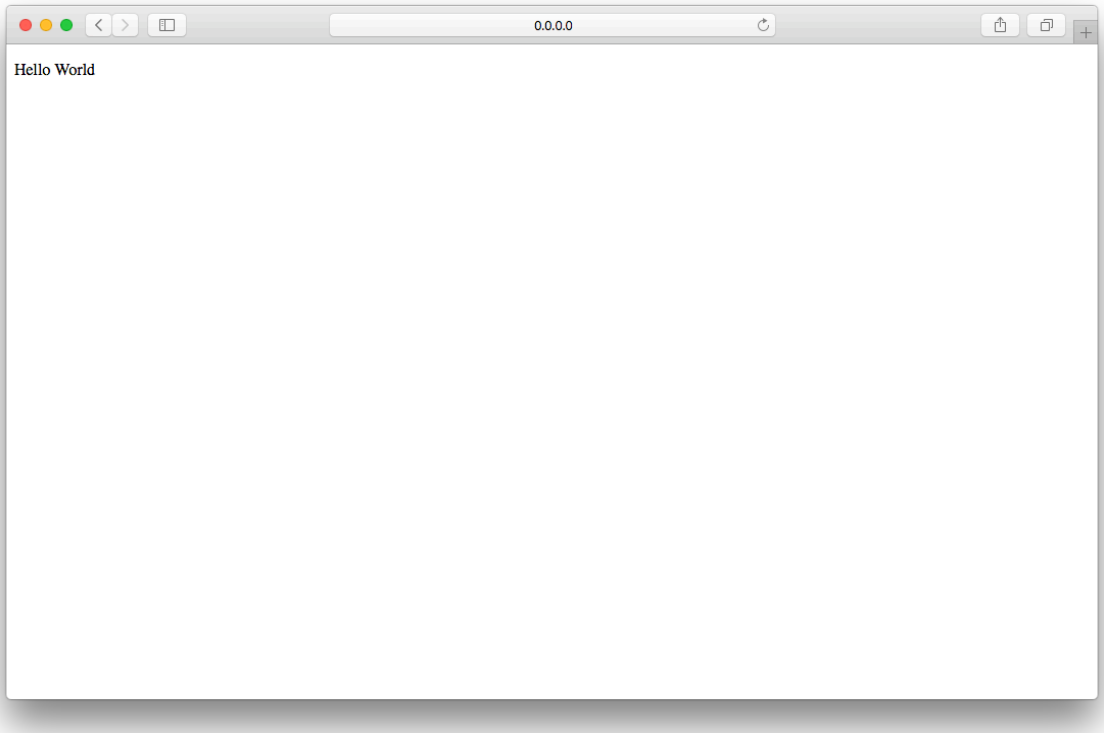
Rails では **設定より規約 (CoC)** というアプローチを採用しています。前述したように、ファイルの配置場所やコントローラ名/ビューの名前の付け方に一定の規則をもたせることで、コードの記述量を減らすことができます。

次に app/views/articles/index.html.erb を見てみます。ビューは今回は ERB(eRuby) と呼ばれる、HTML に Ruby スクリプトを埋め込むことができるテンプレートライブラリを使って記述します。

Hello World と表示したいだけなら、特に Ruby スクリプトを入れ込む必要はないので、以下のように index.html.erb を編集します。

```
<p>Hello World</p>
```

ブラウザで http://0.0.0.0:3000/articles/index にアクセスしてみましょう。



ルーティングを理解する

Rails のルーティングは RESTful なインターフェースを簡単に定義できるように作られています。

データベースから記事を取得する

記事一覧を表示する

<%= %> と <% %> の違い

記事詳細を表示する

画面から記事新規作成をできるようにする

画面から記事の更新ができるようにする

画面から記事を削除する

デザインを整える

Layout の説明

まとめ

足りない機能

- Validation
- 執筆 Note:
    - rails console
    - binding.pry
    - annotate
    - デバッグのコツ
    - ActiveRecord
    - ActiveModel
    - View
    - Controller
    - Router
    - デプロイ

1.3 React 入門

1.3.1 React の環境構築

React を利用するために、フロントエンドビルドツールである webpack を利用して開発をすすめます。

React をビルドするにあたり、webpack をそのまま利用する方法と、Rails から webpack を簡単に利用できるようにした webpacker という Gem を使う方法があります。

1. webpack をそのまま利用する
    - 利点
        - Rails に詳しくなくても扱える (ビルドは Rails とは独立して考えればよい)
    - 欠点

    - ボイラープレート（アプリの雛形）や Create React App などのツールを使わないと初期の環境構築が難しい
    - Rails と React を同時にデプロイする仕組みを考える必要がある

2. webpacker を利用する

    - 利点
        - 複雑な webpack の設定ファイルが隠蔽されているので、webpack に慣れていない場合でも容易に環境構築ができる
        - Rails のアセットと一緒に webpack build したアセットも容易にデプロイできる
        - ヘルパーが用意されており、Rails の View から webpacker でビルドしたファイルを簡単に呼び出せる
    - 欠点
        - webpack の最新版に Webpacker が追いついていかない (=最新バージョンをすぐに利用できない）おそれがある

webpack をそのまま利用するか webpacker を利用するかは、システムの設計方針によっても変わります。

Rails とフロントエンド (Javascript/View) を完全に分離して開発したい場合は webpack をそのまま利用すべきです。逆に Rails のエコシステムにのりつつ React を導入したい場合は webpacker を利用するのをおすすめします。

webpack を利用してビルドする

webpack を利用して、Hello World を作成しましょう。

React アプリを新規作成する

Create React App を利用すると、新規の設定済み React + Webpack プロジェクトを容易に作成でき、複雑な webpack の初期設定をしなくて済みます。

Javascript のパッケージマネージャには、以降 yarn を利用します。次のコマンドでインストールしてください。

```
npm install -g yarn
```

Create React App はグローバルなパッケージとしてインストールする必要があります。

```
yarn global add create-react-app
```

Create React App をインストール後、次のコマンドで React アプリを作成します。

```
create-react-app hello-react
```

コマンド実行後に次のファイルが生成されます。

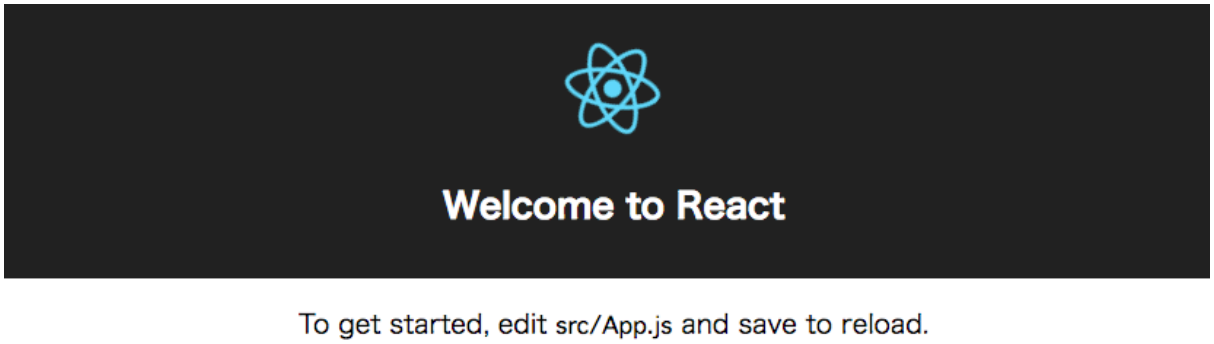
```
$ tree -I 'node_modules'

.
├── README.md
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── registerServiceWorker.js
└── yarn.lock
```

React アプリを起動する

作成した react プロジェクトディレクトリ内で、次のコマンドを使い development 環境で開発用サーバを起動します。

```
yarn start
```

起動後に表示されているアドレスにアクセスすると、React アプリが表示されることを確認してください。



src/App.js には画面表示用のコンポーネントがあります。

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
```

(次のページに続く)

(前のページからの続き)

```
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

h1 タグ、Welcome to React の文字列を適当なものに変更すると、自動でビルドが走り画面に表示された文字列が変化していることを確認してください。

React アプリをビルドする

yarn build コマンドを利用することで、作成した React アプリの Production 用 js/css ファイルを作成することができます。

```
$ yarn build

yarn run v1.1.0
$ react-scripts build
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  38.06 KB  build/static/js/main.a3b22bcc.js
  299 B     build/static/css/main.c17080f1.css
```

build ディレクトリ以下に作成されたファイルは、そのままサーバに配置することができます。

```
$ tree build/

build/
├── asset-manifest.json
├── favicon.ico
├── index.html
├── manifest.json
├── service-worker.js
└── static
    ├── css
    |   ├── main.c17080f1.css
```

(次のページに続く)

(前のページからの続き)

```
    |   └── main.c17080f1.css.map
    ├── js
    |   ├── main.a3b22bcc.js
    |   └── main.a3b22bcc.js.map
    └── media
        └── logo.5d5d9eef.svg

4 directories, 10 files
```

webpacker を利用してビルドする

webpacker を利用して React をビルドしてみましょう。webpacker は Ruby の gem であり webpack のラッパーでもあります。webpacker は Rails と組み合わせて使うことを想定しています。

以下手順で Rails アプリから React を呼び出します。

1. Rails を新規作成する
2. Rails の Controller と View、Routing を定義する
3. React コンポーネントを作成する
4. View から React を呼び出す

Rails アプリを新規作成する

gem コマンドを使って Rails を導入します。

```
gem install rails
```

rails new コマンドで新規 Rails + React アプリケーションを作成します。

```
rails new hello-react-rails --webpack=react --database=mysql
```

作成した Rails アプリディレクトリの中で次のコマンドを実行し、データベースの初期化をします。

```
bundle exec rake db:create db:migrate
```

この状態で Rails を起動してみましょう。次のコマンドでサーバを起動することができます。

```
bundle exec rails server
```

更に、Webpack でのビルドをするために webpack-dev-server も別途起動する必要があります。

```
bin/webpack-dev-server
```

ブラウザで http://0.0.0.0:3000 にアクセスすると、次のような Rails の初期画面が表示されることを確認してください。

Controller/View と Routing を追加する

Hello World 表示用の Controller/View を作成します。次のコマンドで index アクションを持った DashboardsController を作成します。

```
bundle exec rails generate controller dashboards index
```

app/controllers/dashboards_controller.rb 、app/views/dashboards/index.html.erb が新規作成されます。

```
class DashboardsController < ApplicationController
  def index
  end
end
```

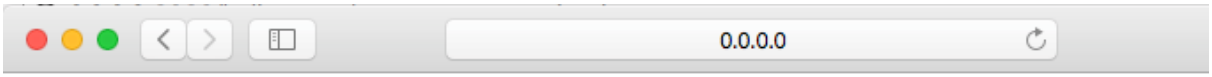
```
<h1>Dashboards#index</h1>
<p>Find me in app/views/dashboards/index.html.erb</p>
```

config/routes.rb には routing が追加されています。

```
Rails.application.routes.draw do
  get 'dashboards/index'
  # For details on the DSL available within this file, see http://guides.
↪rubyonrails.org/routing.html
end
```

Controller、View、および Routing を追加することで、画面にアクセスできるようになります。

http://0.0.0.0:3000/dashboards/index にアクセスすると、次の画面が表示されます。



Dashboards#index

Find me in app/views/dashboards/index.html.erb

HTML は app/views/dashboards/index.html.erb に記述します。中身を適当な文字列に書き換えて画面を更新してみてください。

React コンポーネントを作成する

webpack を使ってビルドする javascript ファイルは、app/javascript 以下に配置します。app/assets/javascript ではないので注意してください。

app/javascript 以下は webpacker (webpack) でビルドされるファイルを配置し、app/assets/javascript は Sprockets でコンパイルするファイルを配置します。

Sprockets とは Rails が利用しているアセット管理用のライブラリで、js/css 等のアセットのコンパイル等を担っています。Sprockets では ES6 のコンパイルはできませんので、ES6 形式の JS は app/javascript に配置し webpack にビルドさせましょう。

app/javascript/packs/hello_react.jsx に以下ファイルを配置します。Rails の View から直接呼び出すエントリーポイントとなる javascript は、app/javascript/packs 以下に配置する必要があります。

```
// app/javascript/packs/hello_react.jsx

import React from 'react'
import ReactDOM from 'react-dom'
import PropTypes from 'prop-types'

const Hello = props => (
  <div>Hello {props.name}!</div>
)

Hello.defaultProps = {
  name: 'David'
}

Hello.propTypes = {
  name: PropTypes.string
}
```

(次のページに続く)

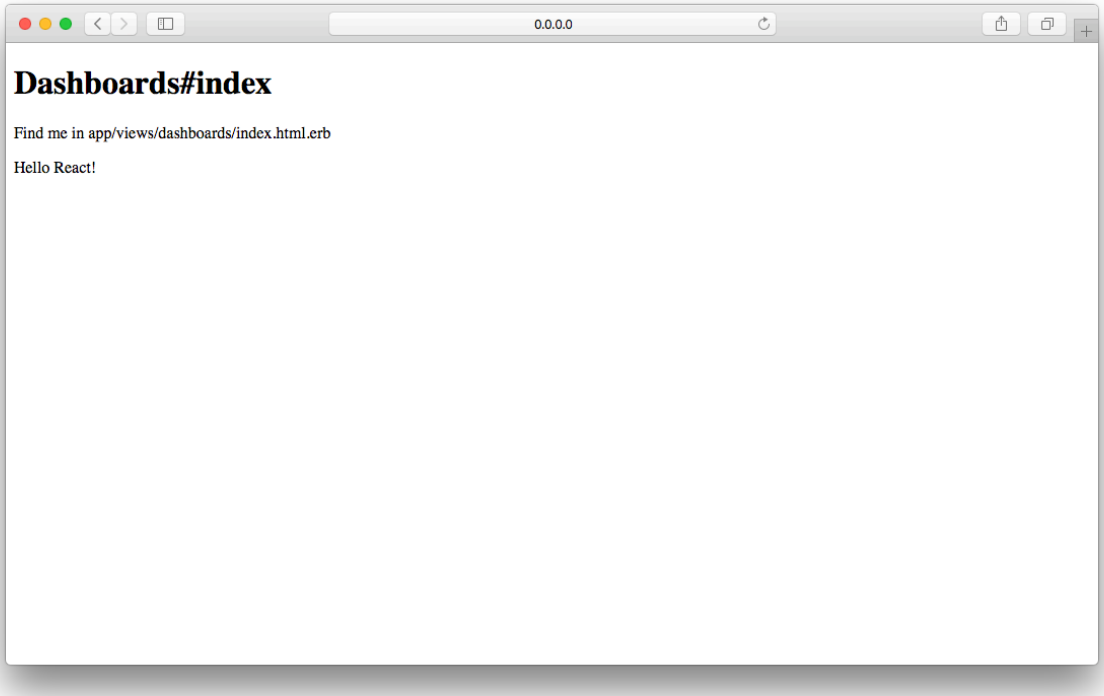
(前のページからの続き)

```
document.addEventListener('DOMContentLoaded', () => {
  ReactDOM.render(
    <Hello name="React" />,
    document.body.appendChild(document.createElement('div')),
  )
})
```

このファイルを View から呼び出してみましょう。app/views/dashboards/index.html を以下のように書き換えます。

```
<h1>Dashboards#index</h1>
<p>Find me in app/views/dashboards/index.html.erb</p>

<%= javascript_pack_tag 'hello_react' %>
```

hello_react の部分は app/javascript/packs 以下のファイル名を指定します。この状態で dashboards/index 画面を更新してみましょう。



Hello React! という部分は React を使ってレンダリングしています。

app/javascript/packs/hello_react.jsx の Hello コンポーネントを次のように書き換えてみましょう。

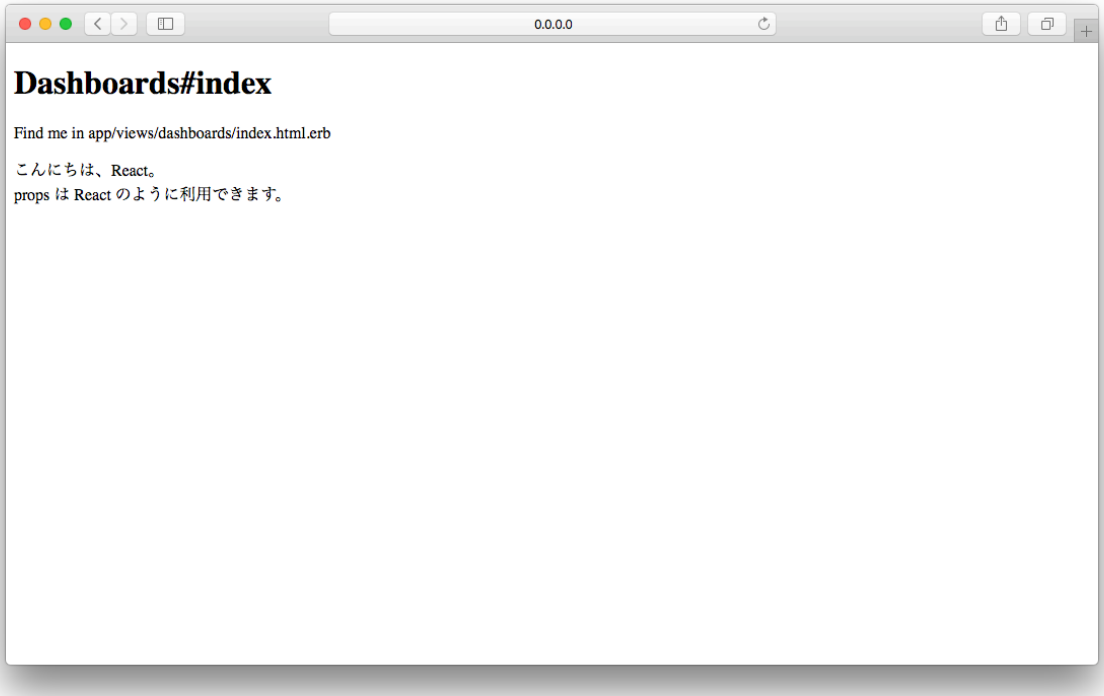
```
import React from 'react'
import ReactDOM from 'react-dom'
import PropTypes from 'prop-types'

class Hello extends React.Component {
```

(次のページに続く)

(前のページからの続き)

```
  render() {
    return (
      <div>
        こんにちは、React。<br />
        props は {this.props.name} のように利用できます。
      </div>
    );
  }
}

Hello.defaultProps = {
  name: 'David'
}

Hello.propTypes = {
  name: PropTypes.string
}

document.addEventListener('DOMContentLoaded', () => {
  ReactDOM.render(
    <Hello name="React" />,
    document.body.appendChild(document.createElement('div')),
  )
})
```

画面をリロードすると次のように表示されます。



1.3.2 JSX の基本

1.3.3 React コンポーネント

1.3.4 Props と State

1.3.5 React のライフサイクル

1.4 React+Rails CRUD アプリケーション

1.5 シングルページ CRUD アプリケーション

第 2 章

React/Rails 実践編