Webエンジニアコース フルタイム

– Webアプリケーション開発 アセット管理と実装 –

DIVE INTO CODE

はじめに

この講義の模様は、今後の運営改善に役立てるため、録画をいたします。

映像は、弊社YouTubeアカウントに**限定公開**で保存され、一般に公開されることはありません。

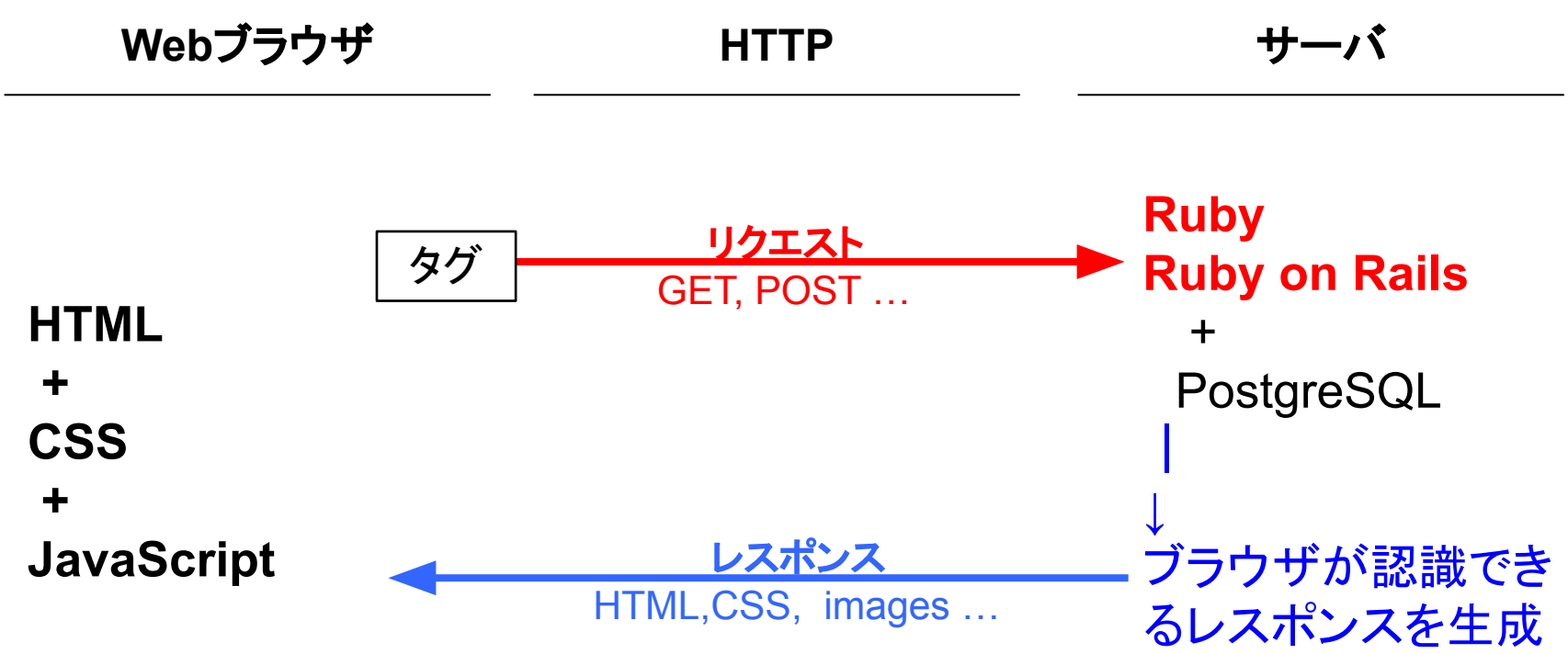
ご理解ご協力のほど、よろしくお願いいたします。

構成

1. Railsとアセットパイプライン
2. Bootstrap
3. グループワーク
4. まとめ

Railsとアセットパイプライン

ブラウザは、HTMLとCSS、JavaScriptしか認識ができない。そのため、サーバ内でそれらを用意する必要がある。



Railsとアセットパイプライン

ブラウザが認識できる各ファイル形式に対応でき、かつ開発しやすい様々な拡張言語を Railsでは扱うことができる。

ブラウザが認識できる	Ruby on Rails 内で扱える	
HTML	ERB(イーアールビー)	HTMLに Rubyコードを埋め込むだけで記述ができ、理解しやすい。
	Haml(ハムル)	学習コストがかかるが、コード量を 30%削減できる。
	Slim(スリム)	（Hamlと同様）
CSS	SASS, SCSS	Rubyで開発されている。変数や CSSのライブラリの組み込みが可能。
	LESS	JavaScriptで開発されている。変数や CSSのライブラリの組み込みが可能。
JavaScript	CoffeeScript	コード量を大幅に削減ができ、Rubyのような記法で書ける。

Railsとアセットパイプライン

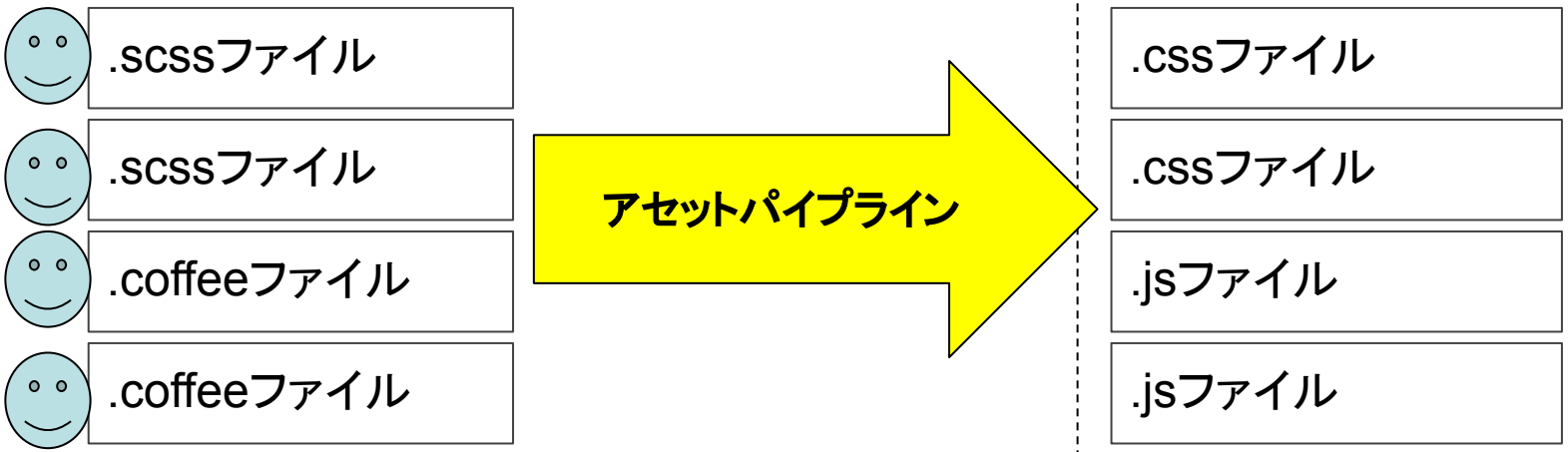
Railsのアセットパイプラインは、エンジニアの開発スピードとWebブラウザの表示スピードを双方共に向上させる。

エンジニアが開発 → **Railsが自動化** → **Webブラウザで表示**

機能や画面別にフォルダやファイルを作成して分業できる。

各フォルダを集約してまとめ、HTMLの linkとして自動記述

軽量化されたファイルは素早く読み込め、更新分の反映もされる。

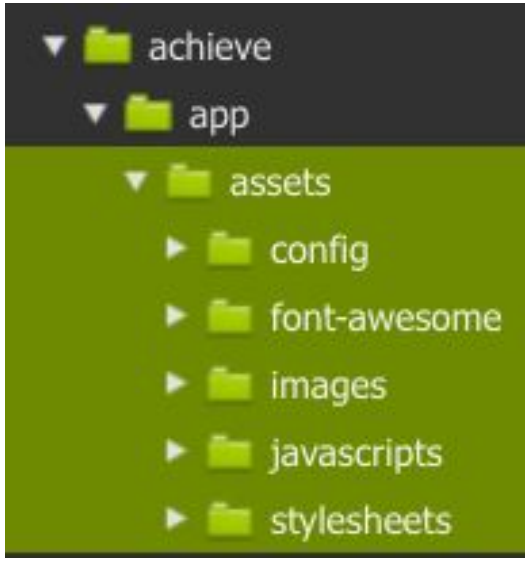


Railsとアセットパイプライン

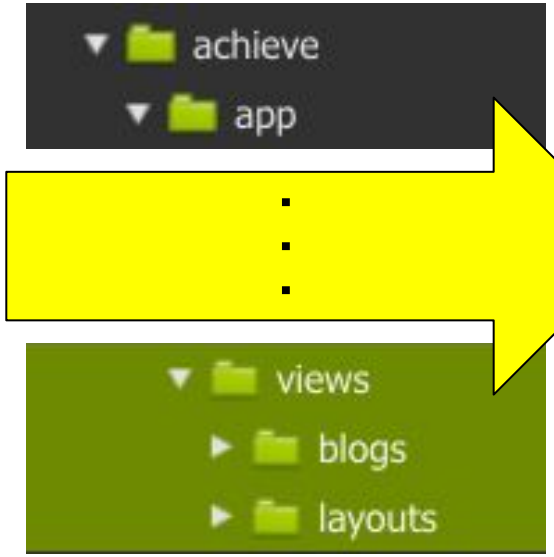
assets内のSCSSやCoffeeScriptが、View内のERBと結合して、HTMLやCSS、JavaScriptのレスポンスを返す。

アセット → ERB → Webブラウザで表示

app/assets



app/views

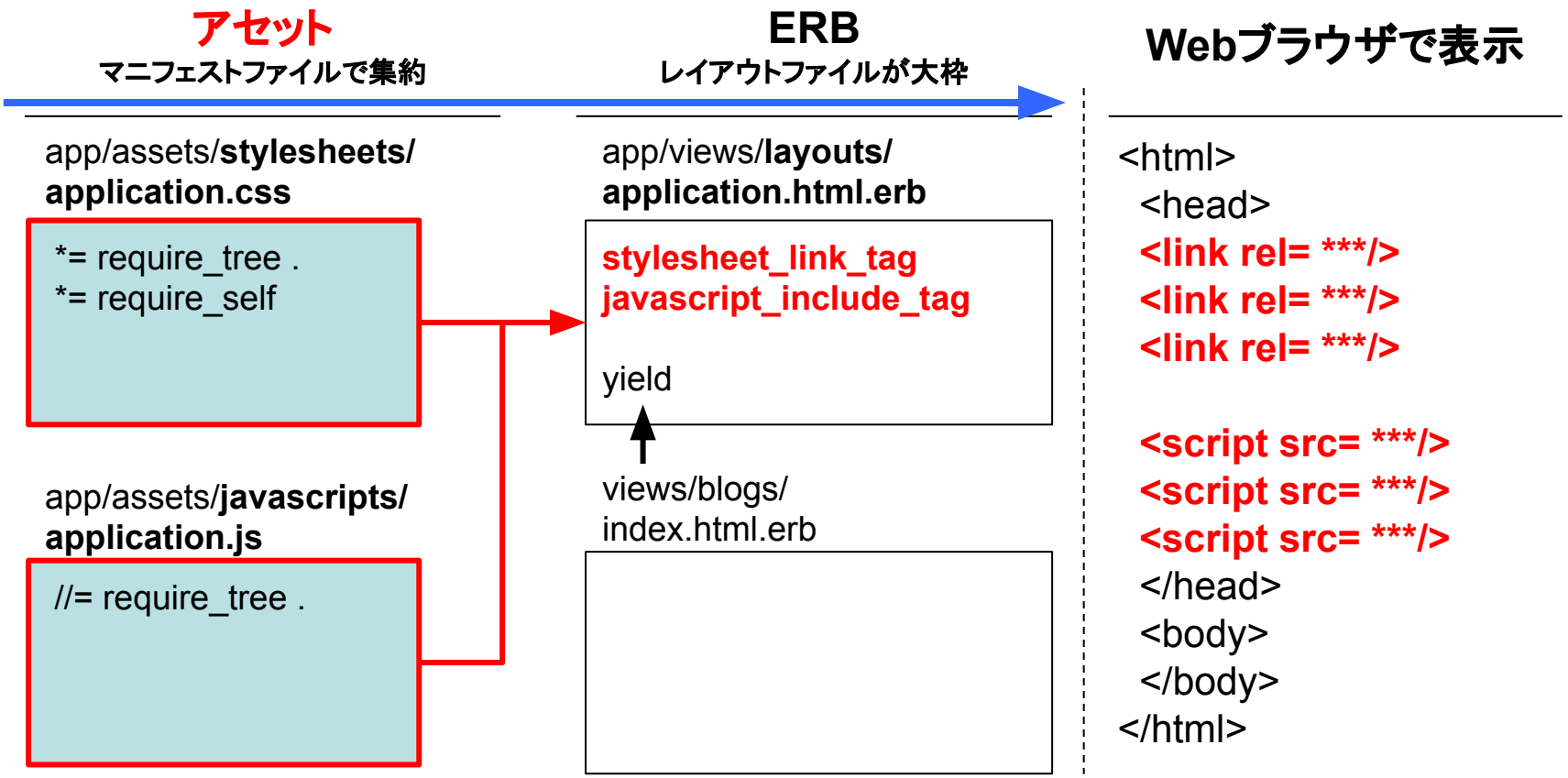



```
<html>
  <head>
  <link rel= ***/>
  <link rel= ***/>
  <link rel= ***/>

  <script src= ***/>
  <script src= ***/>
  <script src= ***/>
  </head>
  <body>
  </body>
</html>
```

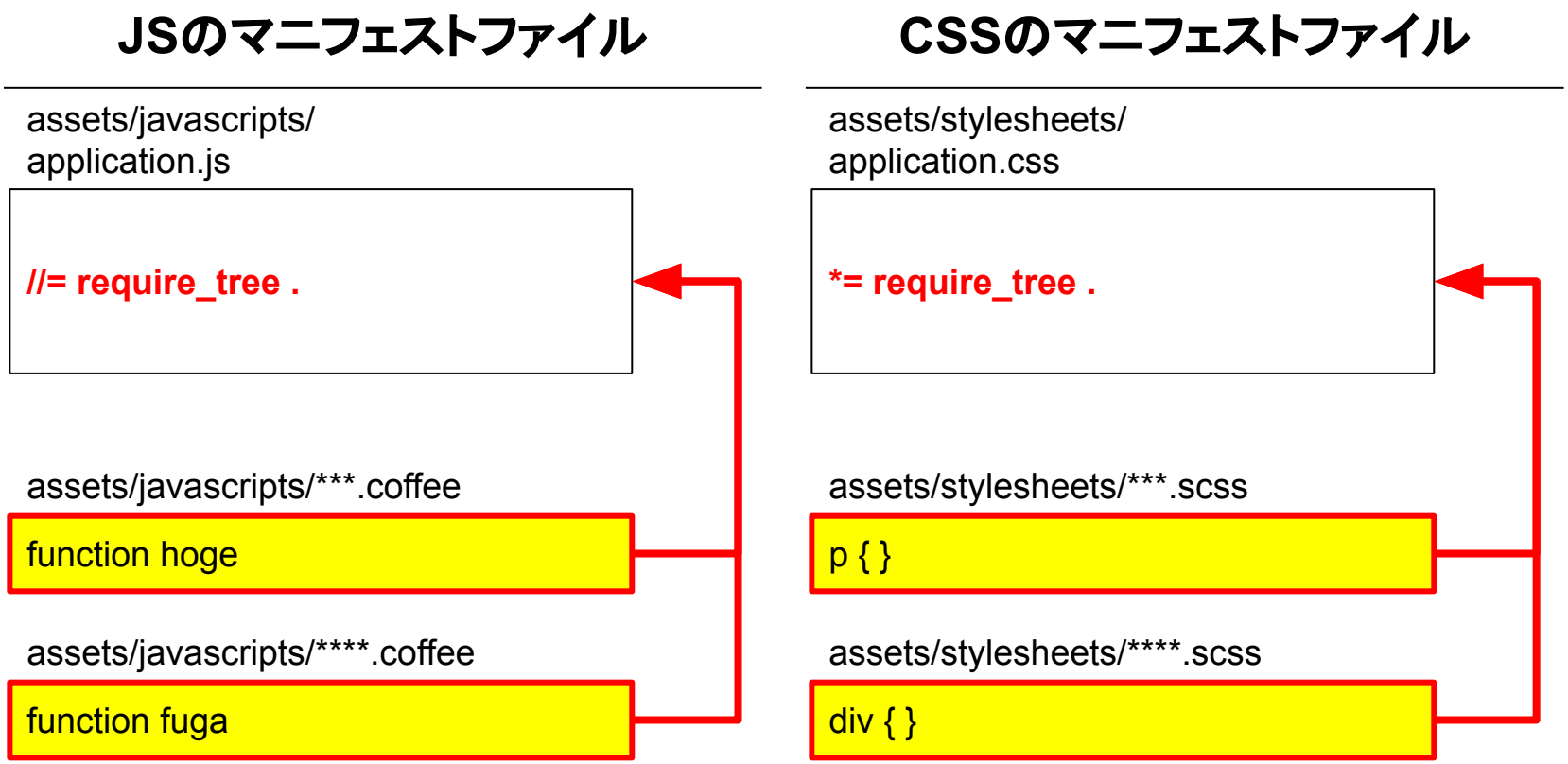
Railsとアセットパイプライン

SCSSやCoffeeScriptは、マニフェストファイルで集約され、レイアウトファイル内のヘルパーメソッドでリンクが生成される。



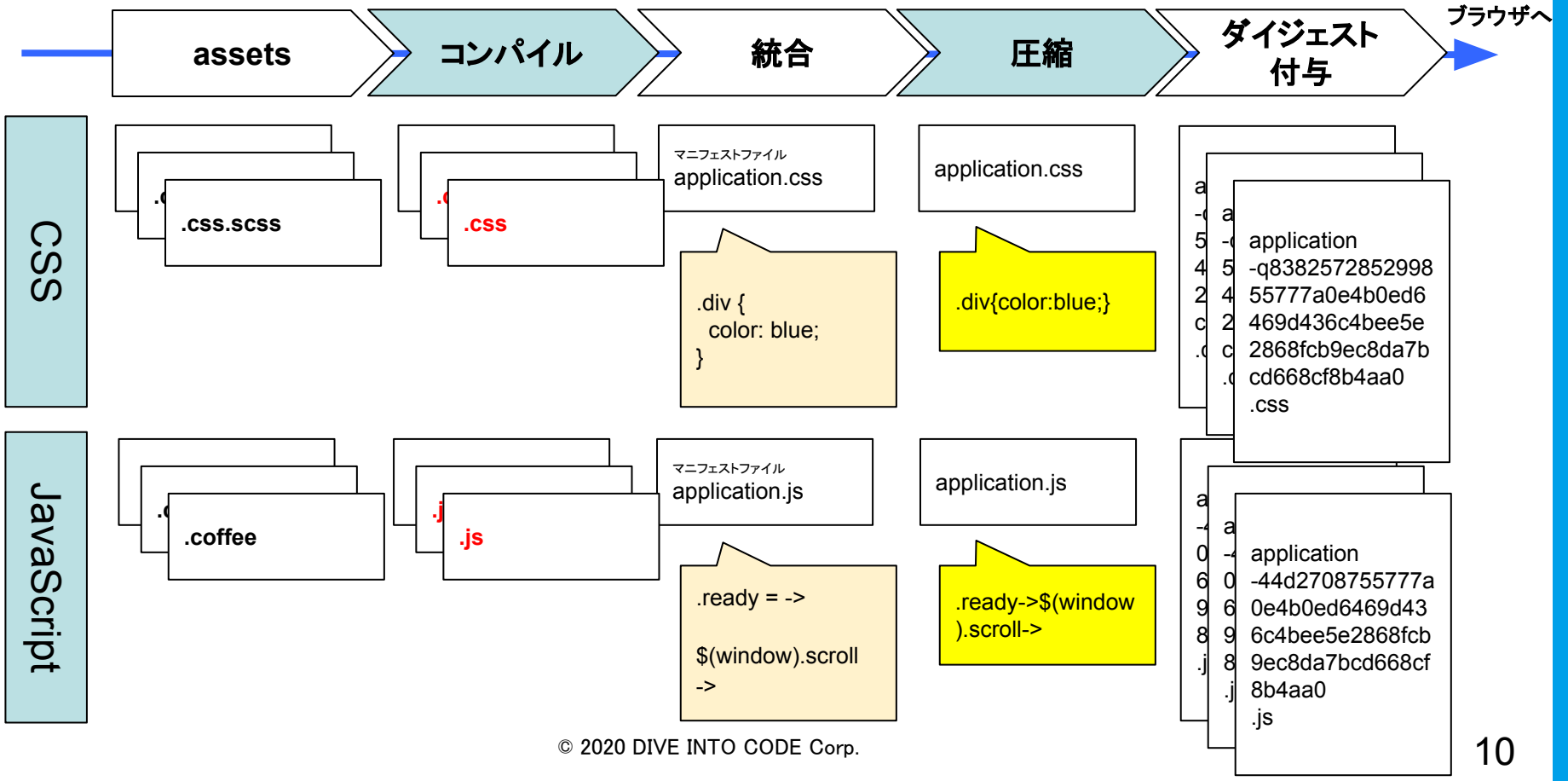
Railsとアセットパイプライン

マニフェストファイル内に記述されているコメントのようなコードが、他のファイルを読み込む役割を担う。



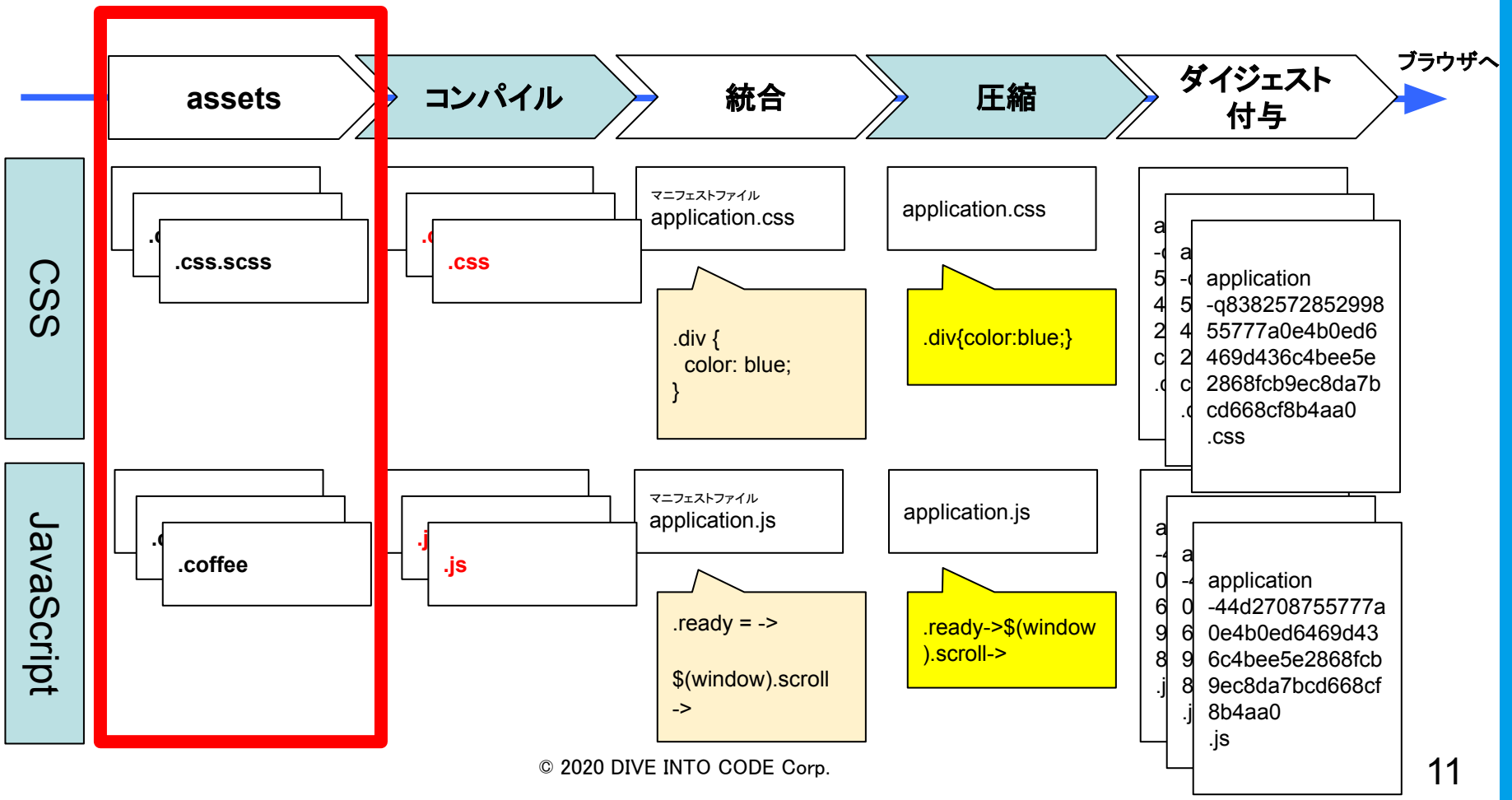
Railsとアセットパイプライン

アセットパイプラインは、assets内ファイルを集約し、ブラウザから参照できるCSS、JavaScript を生成する。



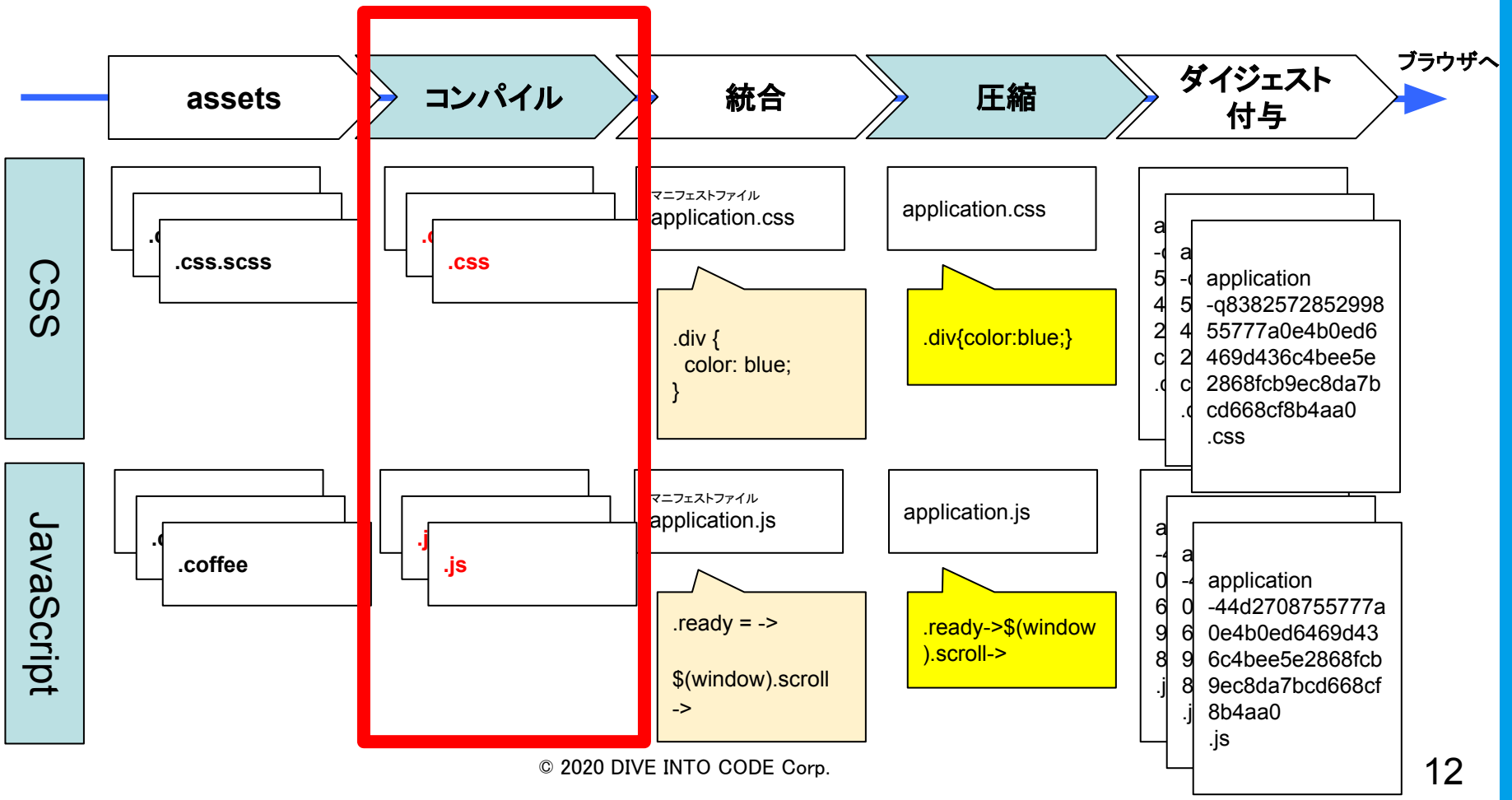
Railsとアセットパイプライン

質問:「なぜ、必要なのか？」



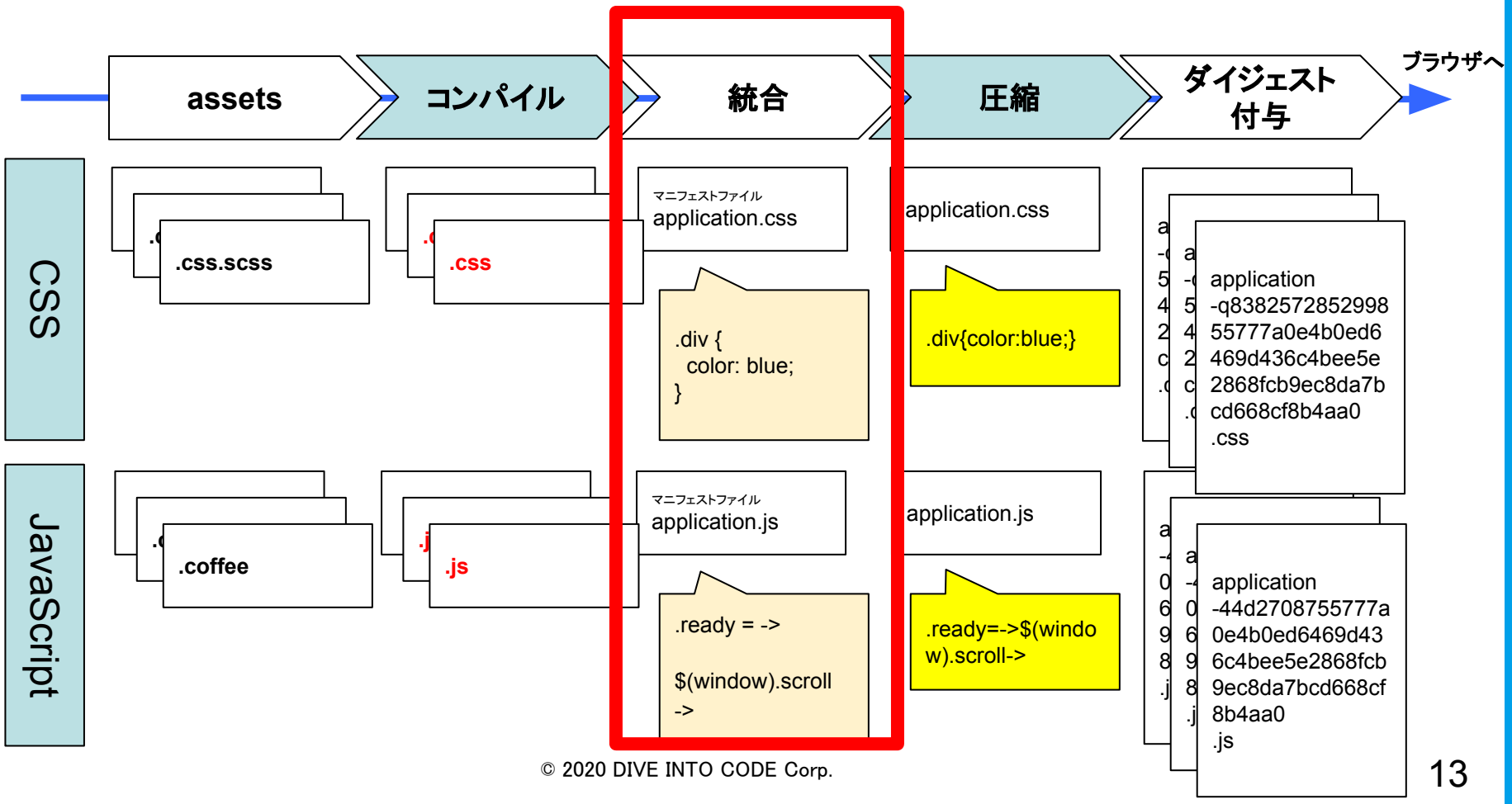
Railsとアセットパイプライン

質問:「なぜ、必要なのか？」



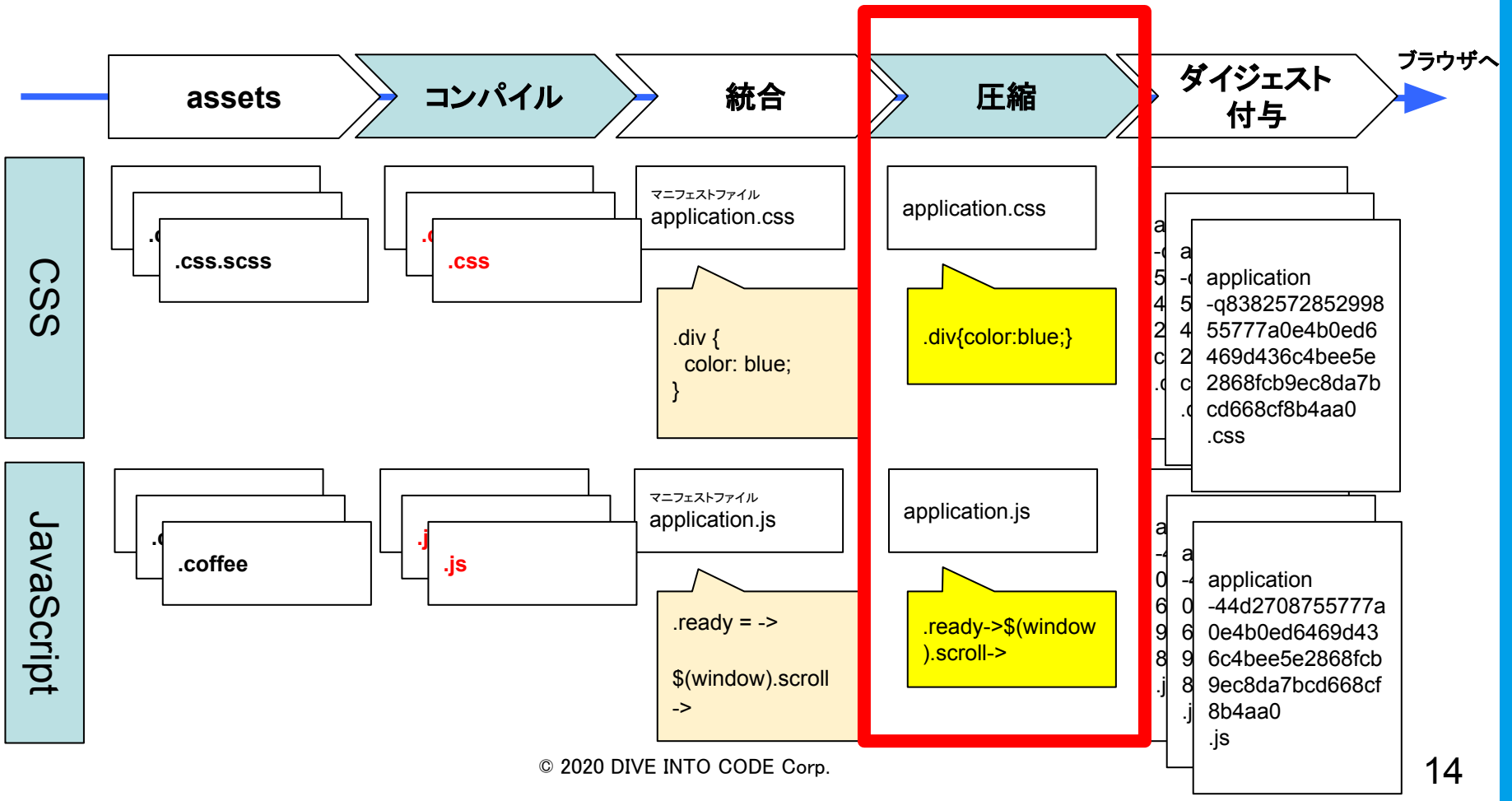
Railsとアセットパイプライン

質問:「なぜ、必要なのか？」



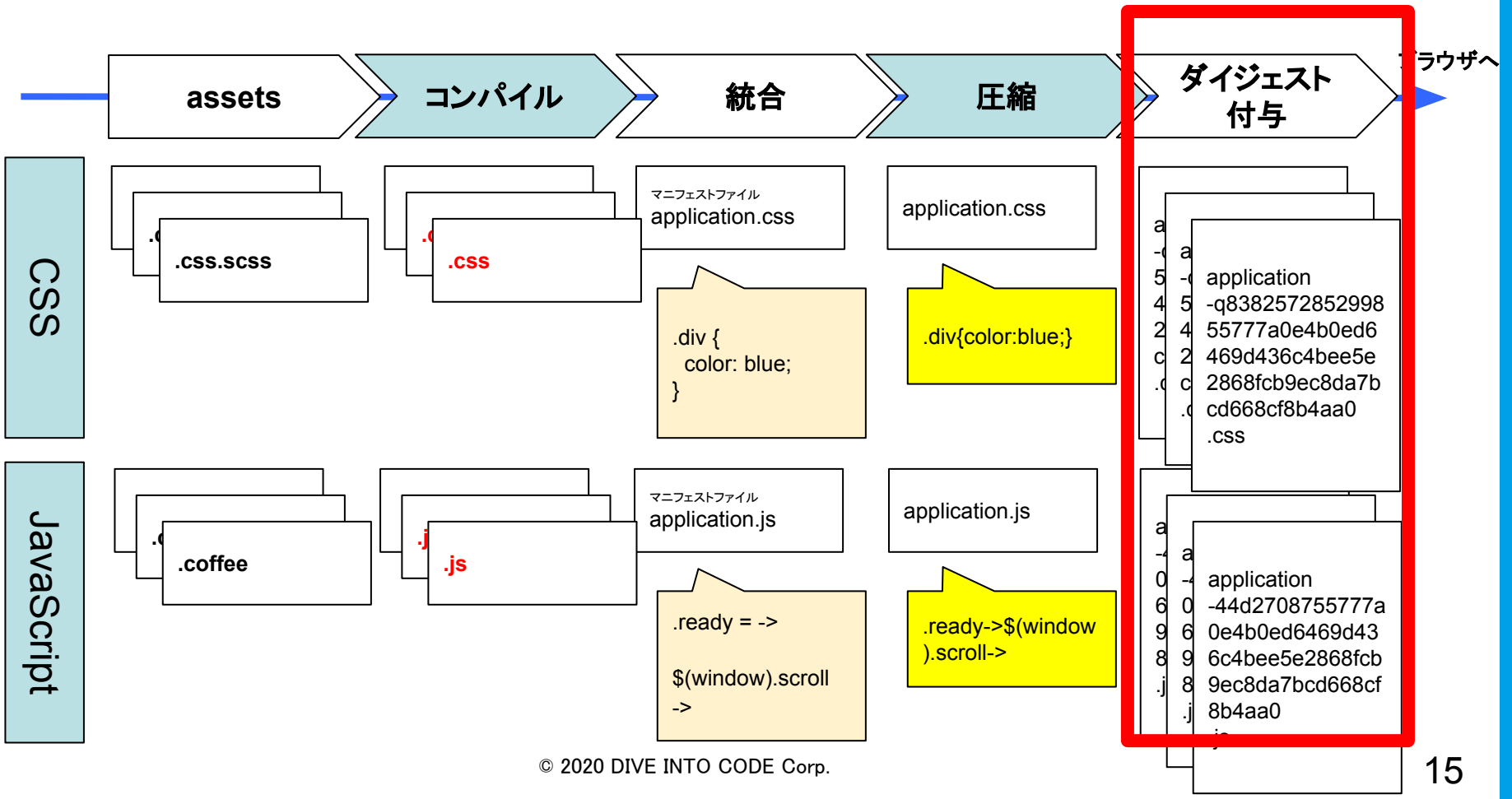
Railsとアセットパイプライン

質問:「なぜ、必要なのか？」



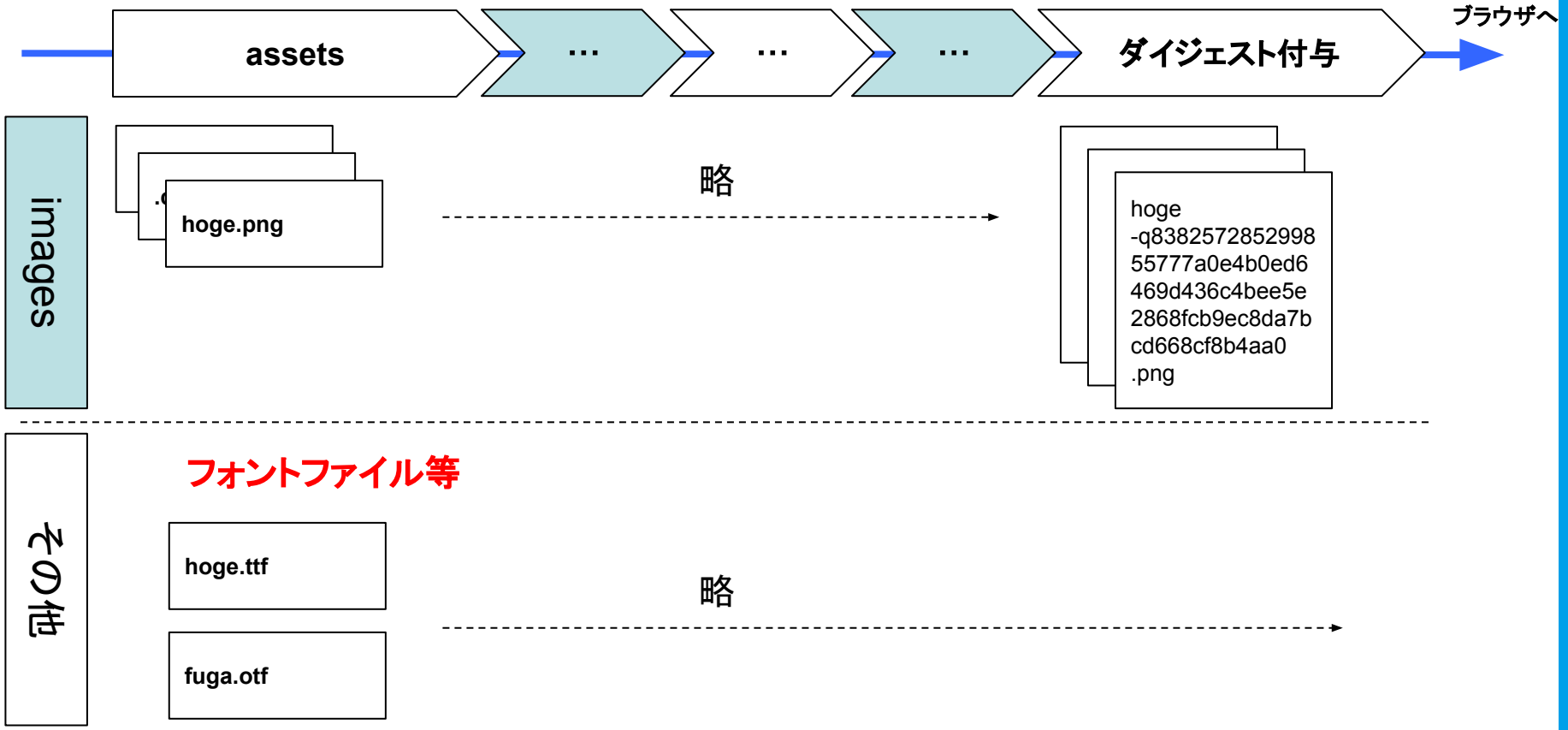
Railsとアセットパイプライン

質問:「なぜ、必要なのか？」



Railsとアセットパイプライン

imagesやassets内に配置された全ディレクトリ内のファイルをブラウザからアクセスできるようにする。



Railsとアセットパイプライン

ViewやSCSSファイルでのアセットの読み込み指定は、Railsや gem sass-rails のヘルパーメソッドを使おう。

使用場所	ヘルパーメソッド	# => 実行結果
ERB (View)	**asset_path**("ファイル名.js")	"/assets/ファイル名-**ダイジェスト**.js”
	image_path("ファイル名.png")	"/assets/images/ファイル名.png”
	font_path("ファイル名.ttf")	"/**fonts**/ファイル名.ttf”
SCSS (sass-rails)	**asset-path**("ファイル名.js")	"/assets/ファイル名-**ダイジェスト**.js”
	image-path("ファイル名.png")	"/assets/images/ファイル名.png”
	font-path("ファイル名.ttf")	"/**fonts**/ファイル名.ttf”

Railsとアセットパイプライン

developmentとproductionで、アセットを読み込む仕様が異なる。ローカルとHerokuの動作環境に気をつけよう。

環境	アセットパイプライン (プリプロセス)	ファイル格納先 public/assets/
development	**自動**実行	**動的**格納
production	**手動**実行 precompile	**静的**格納

ローカルでの環境確認	Herokuでの環境確認
rails c	heroku run rails c
Rails.env	Rails.env

Railsとアセットパイプライン

CSSやJavaScriptのライブラリを使う場合は、マニフェストファイルから参照させるのが一般的な使い方。

jQueryを使う

assets/javascripts/
application.js

読み込む順番に注意！

```
//= require jquery
//= require rails_ujs
//= require_tree .
```

assets/javascripts/***.coffee

```
function(  )
```

jQuery のファイル一式が格納されたディレクトリ
- 方法1. Gem
- 方法2. Yarn

Bootstrapを使う

assets/stylesheets/
application.css

GemとYarnとでは記述内容が違うため要注意！

```
*= require bootstrap/dist/css/bootstrap.min
*= require_tree .
```

assets/stylesheets/***.scss

```
p { }
```

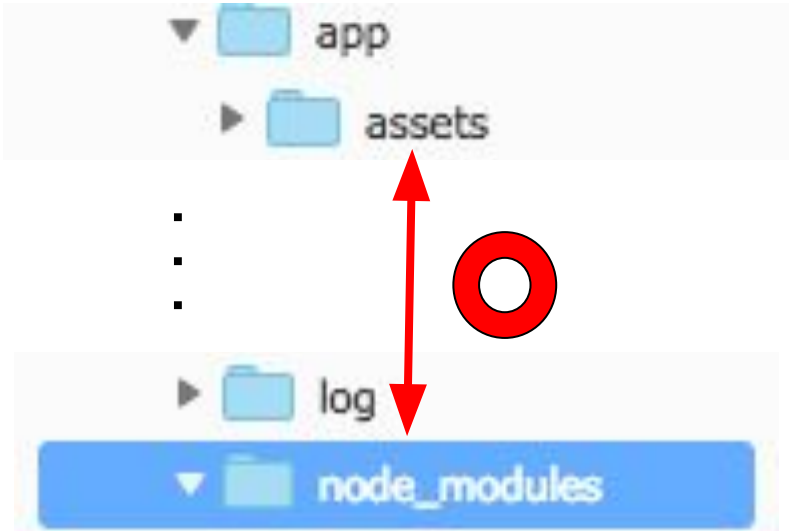
Bootstrap のファイル一式が格納されたディレクトリ
- 方法1. Gem
- 方法2. Yarn

Railsとアセットパイプライン

Rails5は、yarnを活用できるようにnode_modulesディレクトリがデフォルトで読み込まれるようになった。

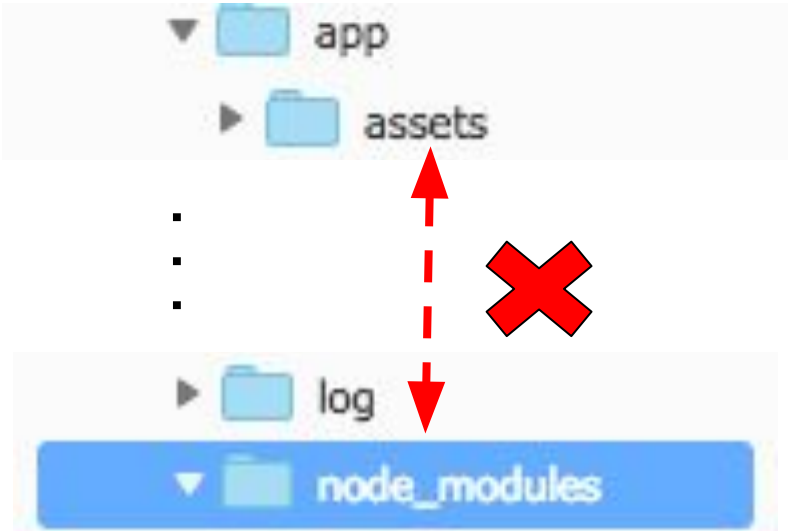
Rails 5.1

Rails から自動的に読み込まれる。
node_modules以下のパスを記述すれば良い。



Rails 5.0 以前

Rails から自動的に読み込まれ**ない**。

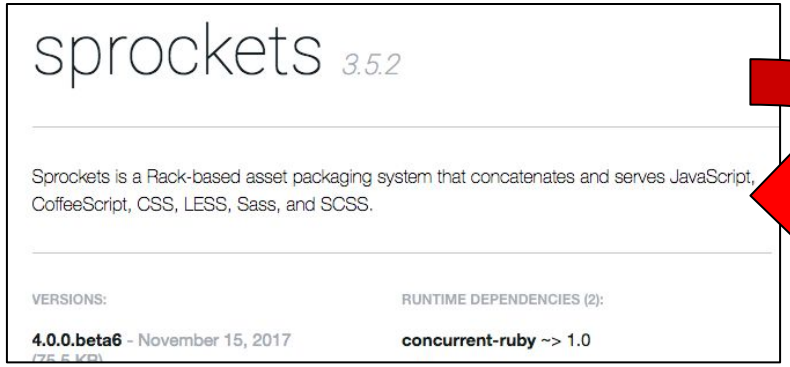


Railsとアセットパイプライン

Rails5は、デフォルトで使われるGem「Sprokets」により下記のディレクトリが自動読み込み（ロード）される。

読み込む仕組み

Sprockets: Rackで開発されたアセット管理Gemが動作している。



https://github.com/rails/sprockets/blob/master/README.md

読み込み先

app/assets/*
lib/assets/*
vendor/assets/*

ターミナル上で確認できる
$ rails c
> Rails.application.config.assets.paths

Railsとアセットパイプライン

production環境では、assets内からプリコンパイルされたファイルを public/assets に配置する。

開発環境内の格納先		プリコンパイル後の格納先
app/assets/* lib/assets/* vendor/assets/*	**画像** images/*.png images/hoge/*.png	public/assets/*_ダイジェスト.png public/assets/hoge/*_ダイジェスト.png
	JavaScript javascripts/*.coffee	public/assets/*_ダイジェスト.js
	CSS stylesheets/*.scss	public/assets/*_ダイジェスト.css
	その他任意 hogehoge/*.hoge	public/assets/*_ダイジェスト.hoge

Railsとアセットパイプライン

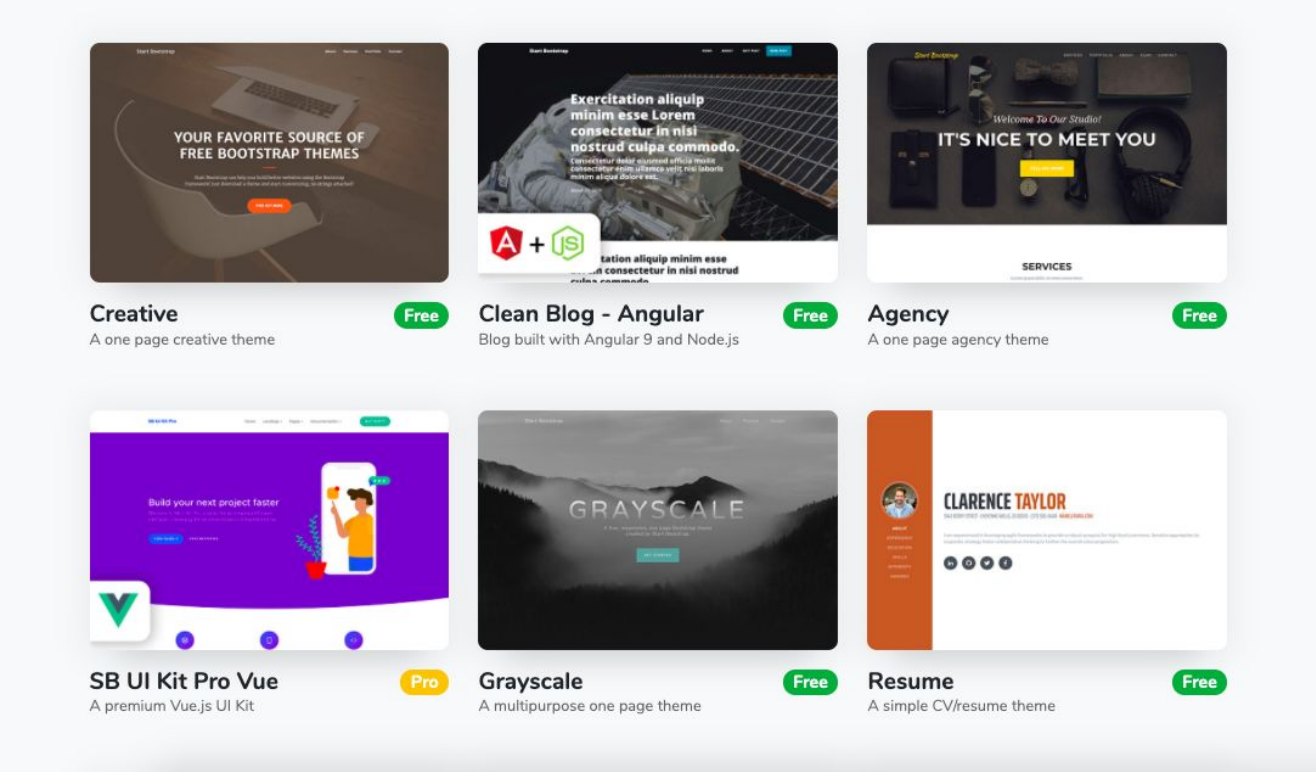
public/assetsディレクトリは、Webサーバから読み込まれ、Webブラウザから直接アクセスができる。

プリコンパイル後の格納先	Webブラウザからのアクセス URL（publicディレクトリは読み込める）
public/assets/*_ダイジェスト.png public/assets/hoge/*_ダイジェスト.png	ドメイン/assets/*_ダイジェスト.png ドメイン/assets/hoge/*_ダイジェスト.png
public/assets/*_ダイジェスト.js	ドメイン/assets/*_ダイジェスト.js
public/assets/*_ダイジェスト.css	ドメイン/assets/*_ダイジェスト.css
public/assets/*_ダイジェスト.hoge	ドメイン/assets/*_ダイジェスト.hoge

Bootstrap

Bootstrapのフリーテンプレートを活用すれば、あっという間に好きなデザインを実装できる。（Freeのもの/Angularと書かれていないものを選択）

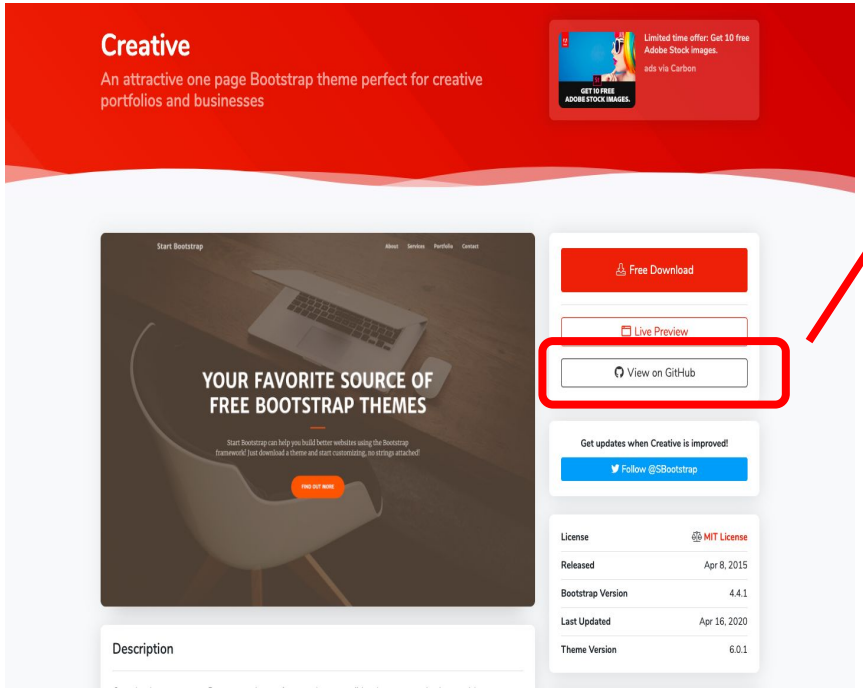
Start Bootstrap https://startbootstrap.com/



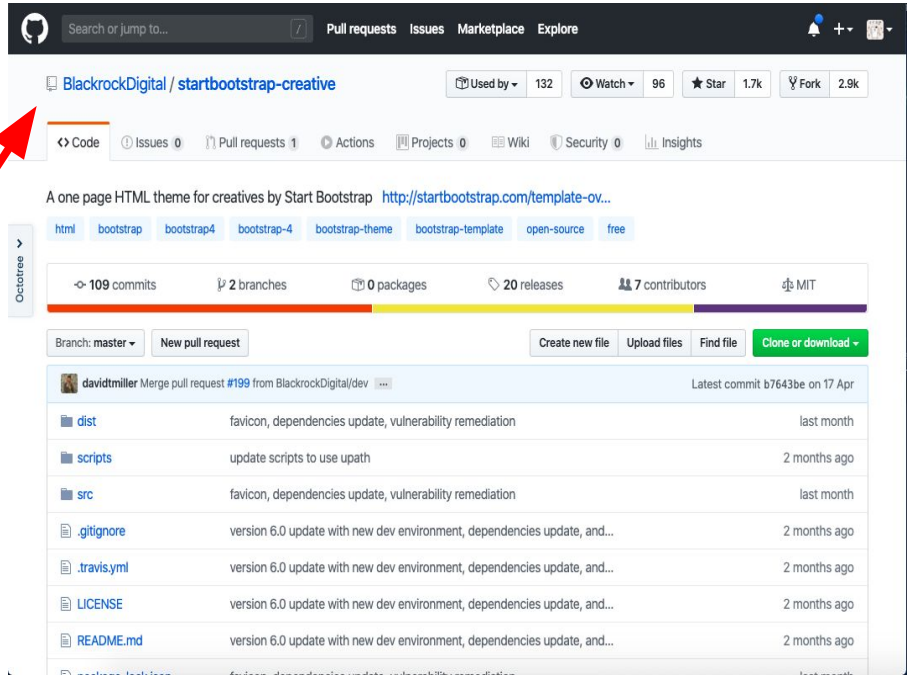
Bootstrap

フリーテンプレートのソースコードは、「View Source」から確認・入手して活用する。

個々のテンプレートのページ



ソースコード
HTML, CSS, JavaScript



ワーク

ペアプログラミング



「二人一組になり、一つの画面・キーボードを共有して実装する」

1人が**ドライバ**としてコードを書き、もう一人が**オブザーバ**としてコードを見ながらアドバイスをすることで知識の共有を促進します。

- 現役エンジニアに学ぶ「ペアプログラミング実践中に重要なポイントとは？」

挨拶をするかのごとく、自然とやろう

グループワーク

テーマ：Start Bootstrapの導入

【流れ】

1. Start Bootstrap https://startbootstrap.com/themes/ にアクセスする
2. CRUD機能つきアプリにStart Bootstrapからテーマを選び、トップページを実装する。トップページtop.html(.erb/.slim/.haml)に実装する（scaffoldを使うこと、railsは5系を使うこと）
3. コントローラ内にtopアクションを実装する
4. トップページをルートでアクセスできるよう実装する
5. 開発が完了したら GitHubリポジトリにプッシュし、slackにGitHubのURLを投稿する
6. 【任意課題】Herokuにデプロイする

まとめ

アセットパイプラインは、エンジニアの開発スピードとWebブラウザの表示スピードを双方共に向上させるためのもの。

- Railsアプリケーションからレスポンスとしてスタイルシートや JavaScript、画像ファイルを返すまでの仕組み。

- SCSSやCoffeeScriptは、マニフェストファイルで集約され、レイアウトファイル内のヘルパーメソッドでリンクが生成される。

- CSSやJavaScriptのライブラリを使う場合は、マニフェストファイルから参照させるのが一般的な使い方。