

Rails講義

はじめに ~本講義について~

本講義について

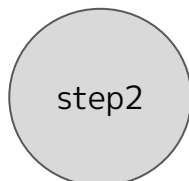
- 本講義は、こちらのスライドと、いくつかのRubyファイル、事前に作成した“test_app”を使って説明していきます。
 - test_appは、rails newとbundle installを済ました状態です。
- 本講義では、RubyやRailsの仕組み、動作の流れを重視して解説していきます。
 - その為、細かいコマンドや、コードの文法上のルールは必要最低限にしています。
 - 講義内で出た文法上の疑問点は、メモをして、講義終了後に調べていただけると幸いです。
- 本講義は、7stepに分かれています。
 - その内、step3, step5, step7終了時にそれぞれ簡単な質疑応答の時間を取ります。
 - 質疑応答の時間内に回答できなかった質問には、講義の最後にまとめて回答させていただきます。
 - 時間の都合上、全ての質問には回答出来ませんがご了承ください。
 - step3の後に小休憩を取ります。

はじめに ~7stepの内訳~

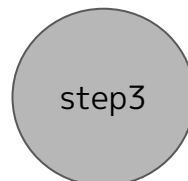
Rubyの基礎 Controller View



Ruby基礎知識

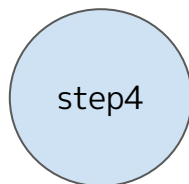


Railsの動作の流れ



VCとインスタンス変数

データベース Model

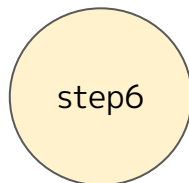


データベースとModel



パラメータ1
詳細画面

投稿機能 バリデーション



パラメータ2
投稿機能



バリデーション

step1

~Rubyの基礎知識~

- step1では、Railsを扱う際に最低限必要になるRubyの知識を復習していきます。

- step1と2では、必要な知識を説明していただけないので聞く方も大変だと思いますが、step3まで聞いていただければどう役立つかが分かります！

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

オブジェクトとは

- WebCampのカリキュラムの説明
 - 操作対象の状態（データ）と便利な機能（メソッド）を1つにまとめたもの
- 生き物、物、概念のような物
- 例を交えて説明いきます

僕に置き換えると、以下のようなデータがあります。

- 年齢: 23歳
- 趣味: ゲーム, ベース

また、以下のような機能があります。

- ご飯を食べる
- プログラミングについて説明する

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

目次

1. オブジェクトの種類
2. 変数
3. メソッドと戻り値

- 飲食店

データ	
種類の名前	内容
ジャンル	“イタリアン”
メニュー数	40



機能(メソッド)	
機能名	内容
提供	お客さんに料理を運ぶ
お会計	お客さんからお金をいただいてレジに入れる

- お店の集合体の“商業施設”もオブジェクト

データ	
種類の名前	内容
住所	“南町田”
店舗	["こなな", "GAP", "CanDo"...]



機能(メソッド)	
機能名	内容
閉店チェック	閉店時間に、全店舗が閉店しているか調べる
定期清掃	共用施設の清掃を行う

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

- 飲食店

データ	
種類の名前	内容
ジャンル	“イタリアン”
メニュー数	40



機能(メソッド)	
機能名	内容
提供	お客さんに料理を運ぶ
お会計	お客さんからお金をいただいてレジに入れる

目次

1. オブジェクトの種類
2. 変数
3. メソッドと戻り値

- お店の集合体の“商業施設”もオブジェクト

データ	
種類の名前	内容
住所	“南町田”
店舗	["こなな", "GAP", "CanDo"...]



機能(メソッド)	
機能名	内容
閉店チェック	閉店時間に、全店舗が閉店しているか調べる
定期清掃	共用施設の清掃を行う

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

Rubyのオブジェクト

- 今のような“人”, “飲食店”, “商業施設”などがオブジェクト
- 以下がRubyの基本的なオブジェクト

※クラス名とは、データの種類の名前のような物です。クラス名の最初の文字は大文字。
※機能の事を、これからメソッドと表します。

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

データ型	クラス名	データの例	メソッド名	メソッドの説明
文字列	String	“abc”	+	文字同士を連結する
数値(整数)	Integer	1	+, .to_s	足し算をする 文字列に変換したオブジェクトを作る
シンボル	Symbol	:abc		あまりない(その分文字列よりも処理が軽い)
配列	Array	[1, 1.0, “abc”, :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{“abc” => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

Rubyのオブジェクト

- 今のような“人”, “飲食店”, “商業施設”などがオブジェクト
- 以下がRubyの基本的なオブジェクト

※クラス名とは、データの種類の名前のような物です。クラス名の最初の文字は大文字。

※機能の事を、これからメソッドと表します。

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

データ型	クラス名	データの例	メソッド名	メソッドの説明
文字列	String	“abc”	+	文字同士を連結する
数値(整数)	Integer	1	+, .to_s	足し算をする 文字列に変換したオブジェクトを作る
シンボル	Symbol	:abc		あまりない(その分文字列よりも処理が軽い)
配列	Array	[1, 1.0, “abc”, :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{“abc” => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

Rubyのオブジェクト

- 今のような“人”, “飲食店”, “商業施設”などがオブジェクト
- 以下がRubyの基本的なオブジェクト

※クラス名とは、データの種類の名前のような物です。クラス名の最初の文字は大文字。

※機能の事を、これからメソッドと表します。

目次

1. オブジェクトの種類
2. 変数
3. メソッドと戻り値

データ型	クラス名	データの例	メソッド名	メソッドの説明
文字列	String	“abc”	+	文字同士を連結する
数値(整数)	Integer	1	+, .to_s	足し算をする 文字列に変換したオブジェクトを作る
シンボル	Symbol	:abc		あまりない(その分文字列よりも処理が軽い)
配列	Array	[1, 1.0, “abc”, :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{“abc” => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

Rubyのオブジェクト

- 今のような“人”, “飲食店”, “商業施設”などがオブジェクト
- 以下がRubyの基本的なオブジェクト

※クラス名とは、データの種類の名前のような物です。クラス名の最初の文字は大文字。

※機能の事を、これからメソッドと表します。

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

データ型	クラス名	データの例	メソッド名	メソッドの説明
文字列	String	“abc”	+	文字同士を連結する
数値(整数)	Integer	1	+, .to_s	足し算をする 文字列に変換したオブジェクトを作る
シンボル	Symbol	:abc		あまりない(その分文字列よりも処理が軽い)
配列	Array	[1, 1.0, “abc”, :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{“abc” => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

Rubyのオブジェクト

- 今のような“人”, “飲食店”, “商業施設”などがオブジェクト
- 以下がRubyの基本的なオブジェクト

※クラス名とは、データの種類の名前のような物です。クラス名の最初の文字は大文字。
※機能の事を、これからメソッドと表します。

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

データ型	クラス名	データの例	メソッド名	メソッドの説明
文字列	String	“abc”	+	文字同士を連結する
数値(整数)	Integer	1	+, .to_s	足し算をする 文字列に変換したオブジェクトを作る
シンボル	Symbol	:abc		あまりない(その分文字列よりも処理が軽い)
配列	Array	[1, 1.0, “abc”, :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{“abc” => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step1. Rubyの基礎知識 ~ オブジェクトの種類 ~

オブジェクトをメソッドで操作してみる

実際にrubyのコードを書いて確認していきます。

目次

1. **オブジェクトの種類**
2. 変数
3. メソッドと戻り値

step1. Rubyの基礎知識 ~ 変数 ~

変数

- オブジェクトは名前を付けて一時保存できる
- この名前を変数と言う
- 変数は、オブジェクトを入れる箱のように振舞う

変数の定義

変数 = オブジェクト

目次

1. オブジェクトの種類
2. **変数**
3. メソッドと戻り値

step1. Rubyの基礎知識 ~ メソッドと戻り値 ~

メソッドの説明の補足

- メソッドとは、“処理(コード)の塊に、名前を付けて保存した部品”
- Rubyには元々定義されているメソッドがたくさんある
- クラスの機能としてのメソッドと、そうでないメソッドがあります
 - ※本来は後者のそうでないメソッドは存在しません
 - クラスの機能としてのメソッドは、先ほど.to_sなど
 - クラスの機能ではないメソッドは、次のスライドで作成する
- メソッドを実行する事を、メソッドを呼び出すという

引数

- メソッド呼び出すとき、メソッドに渡す事が出来るオブジェクトの事

戻り値

- メソッドの呼び出し元に返ってくるオブジェクト(値)の事

目次

1. オブジェクトの種類
2. 変数
- 3. メソッドと戻り値**

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

```
#メソッドの定義
def メソッド名(引数1, 引数2)
  処理
  戻り値 #最終行が戻り値になる
end

#メソッドの呼び出し
メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド
オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド
```

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

#メソッドの定義

```
def メソッド名(引数1, 引数2)
```

処理

戻り値 #最終行が戻り値になる

```
end
```

#メソッドの呼び出し

メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド

オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

```
#メソッドの定義
def メソッド名(引数1, 引数2)
  処理
  戻り値 #最終行が戻り値になる
end

#メソッドの呼び出し
メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド
オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド
```

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

```
#メソッドの定義
def メソッド名(引数1, 引数2)
  処理
  戻り値 #最終行が戻り値になる
end

#メソッドの呼び出し
メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド
オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド
```

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

#メソッドの定義

```
def メソッド名(引数1, 引数2)
```

処理

戻り値 #最終行が戻り値になる

```
end
```

#メソッドの呼び出し

メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド

オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す

step1. Rubyの基礎知識 ~ メソッドをどう使うのか ~

メソッドの定義(作り方)の仕方と呼び出し方

```
#メソッドの定義
def メソッド名(引数1, 引数2)
  処理
  戻り値 #最終行が戻り値になる
end

#メソッドの呼び出し
メソッド名(引数1のオブジェクト, 引数2のオブジェクト) #クラスの機能ではないメソッド
オブジェクト.メソッド名(引数のオブジェクト) #クラスの機能としてのメソッド
```

実際に作ってみる

これから以下のメソッドを作成してみます。

1. 名前と苗字を引数として受け取る
2. 名前と苗字の間に半角スペースを入れた物を、変数に入れる
3. 2を戻り値として返す
4. 実行して、結果をターミナルに出力

目次

1. オブジェクトの種類
2. 変数
3. **メソッドと戻り値**

step1. Rubyの基礎知識 ~ まとめ ~

まとめ

Step1では、以下の学習をしました。

1. オブジェクトはデータと機能をまとめたもので、色々な種類がある
2. 変数は、オブジェクトに名前を付けるもの
3. メソッドはコードの塊に名前を付けたもので、引数と戻り値を設定できる

これらは、これからRailsを触っていく上で大切な知識となります。しっかりと押さえておきましょう。

step2

~Railsの動作の流れ~

- step1では、Rails内部の処理を書くために必要な知識を復習しました。
- step2では、Railsの流れを作るために必要な知識を復習します。

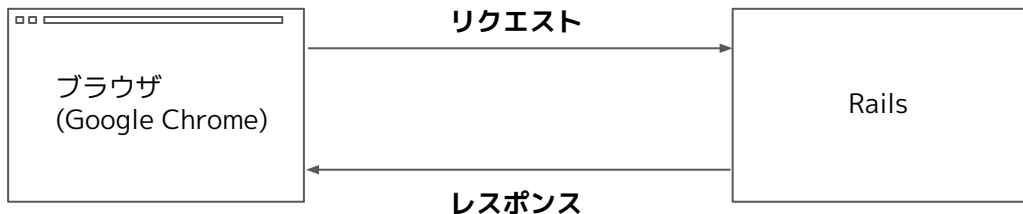
step2. Railsの動作の流れ ~ そもそもRailsは何をしているのか ~

Railsは何をしているのか

1. “ブラウザ”(Google Chrome, Safari等)から“リクエスト”を受け付け
2. リクエストに応じた“処理”を行い
3. ブラウザに“レスポンス”を返す

補足

- リクエスト = HTTPメソッドとURLの組合わせ
- レスポンス = ブラウザに表示させるHTMLなど



目次

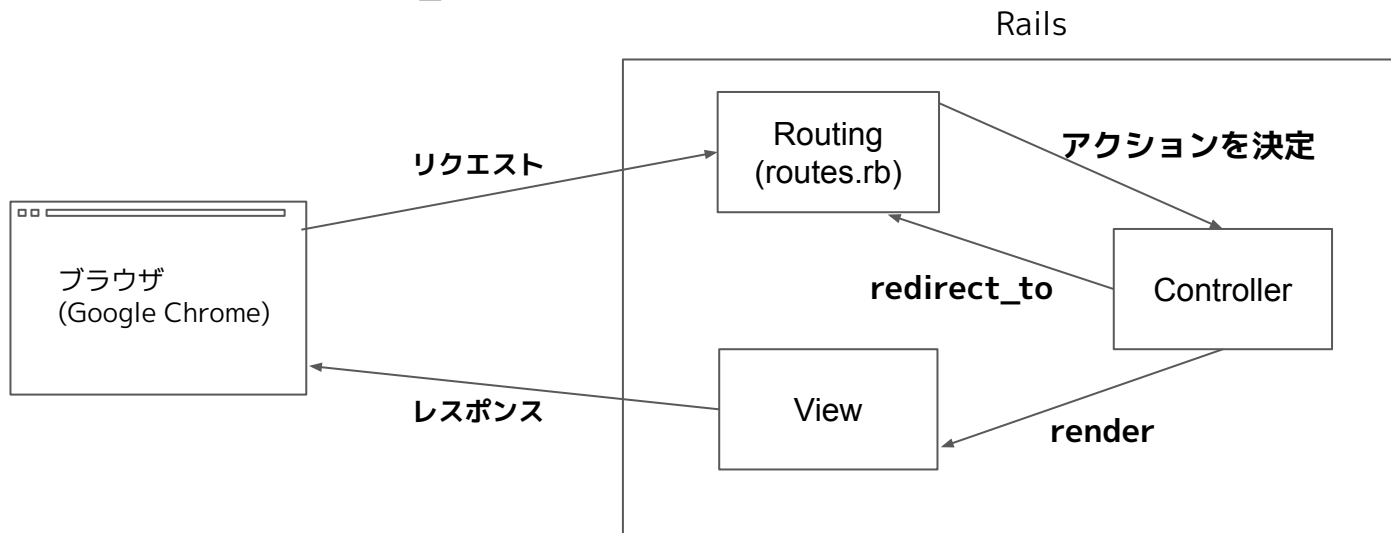
1. そもそもRailsは何をしているのか
2. Rails内部の流れ
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ Rails内部の流れ ~

Rails内部の流れ

※今回はカリキュラムの内容に沿った項目のみ解説します。

1. Routingを元に呼び出すControllerとアクションを決める
2. Controllerを処理
3. render or redirect_to を行う



目次

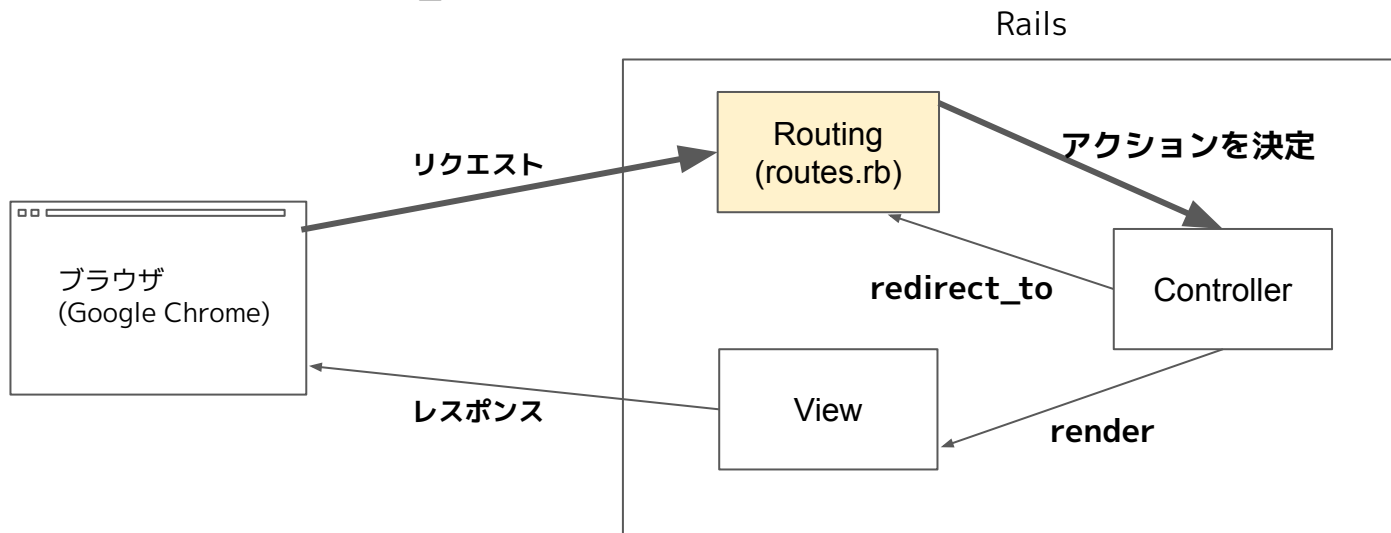
1. そもそもRailsは何をしているのか
- 2. Rails内部の流れ**
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ Rails内部の流れ ~

Rails内部の流れ

※今回はカリキュラムの内容に沿った項目のみ解説します。

1. Routingを元に呼び出すControllerとアクションを決める
2. Controllerを処理
3. render or redirect_to を行う



目次

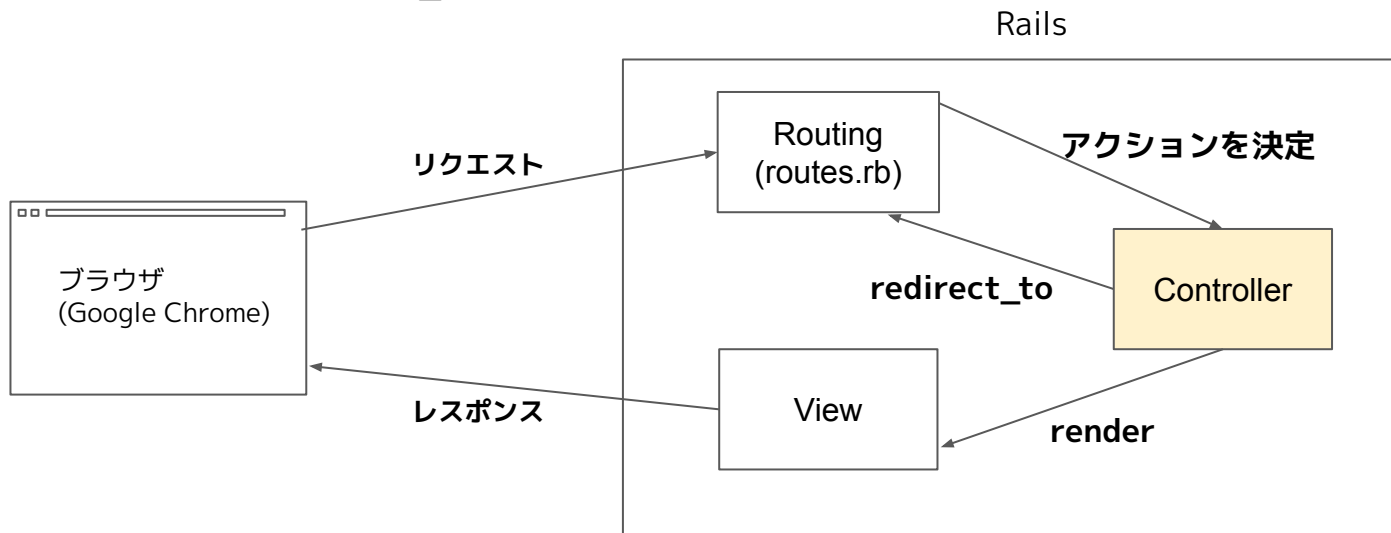
1. そもそもRailsは何をしているのか
2. **Rails内部の流れ**
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ Rails内部の流れ ~

Rails内部の流れ

※今回はカリキュラムの内容に沿った項目のみ解説します。

1. Routingを元に呼び出すControllerとアクションを決める
2. Controllerを処理
3. render or redirect_to を行う



目次

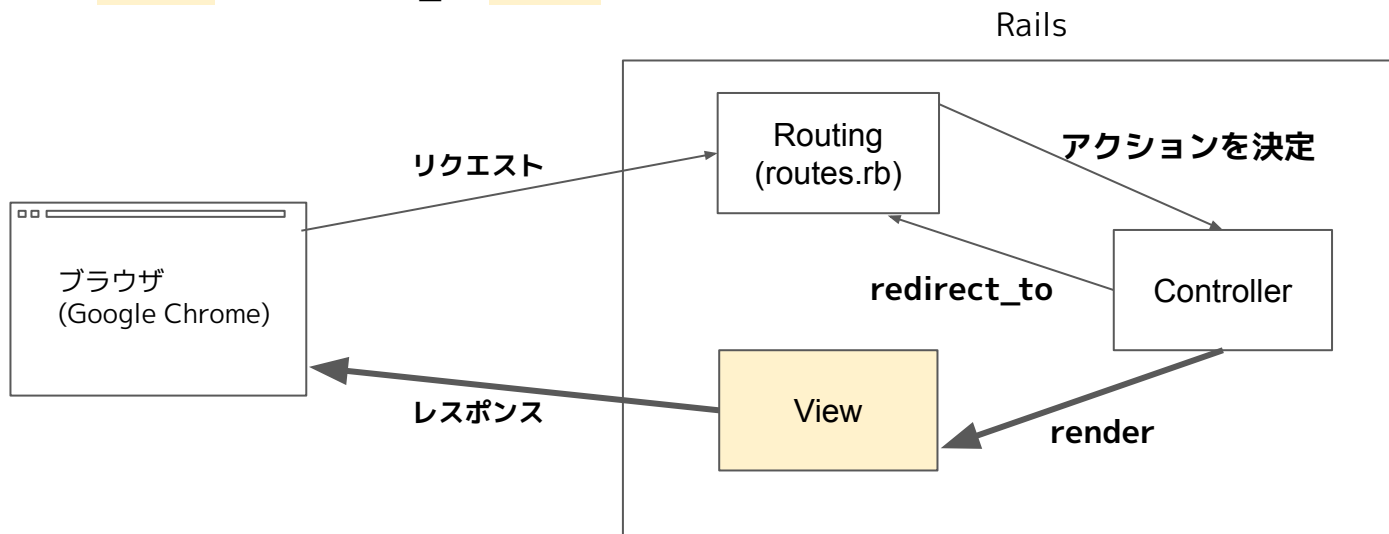
1. そもそもRailsは何をしているのか
2. Rails内部の流れ
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ Rails内部の流れ ~

Rails内部の流れ

※今回はカリキュラムの内容に沿った項目のみ解説します。

1. Routingを元に呼び出すControllerとアクションを決める
2. Controllerを処理
3. `render` or `redirect_to` を行う



目次

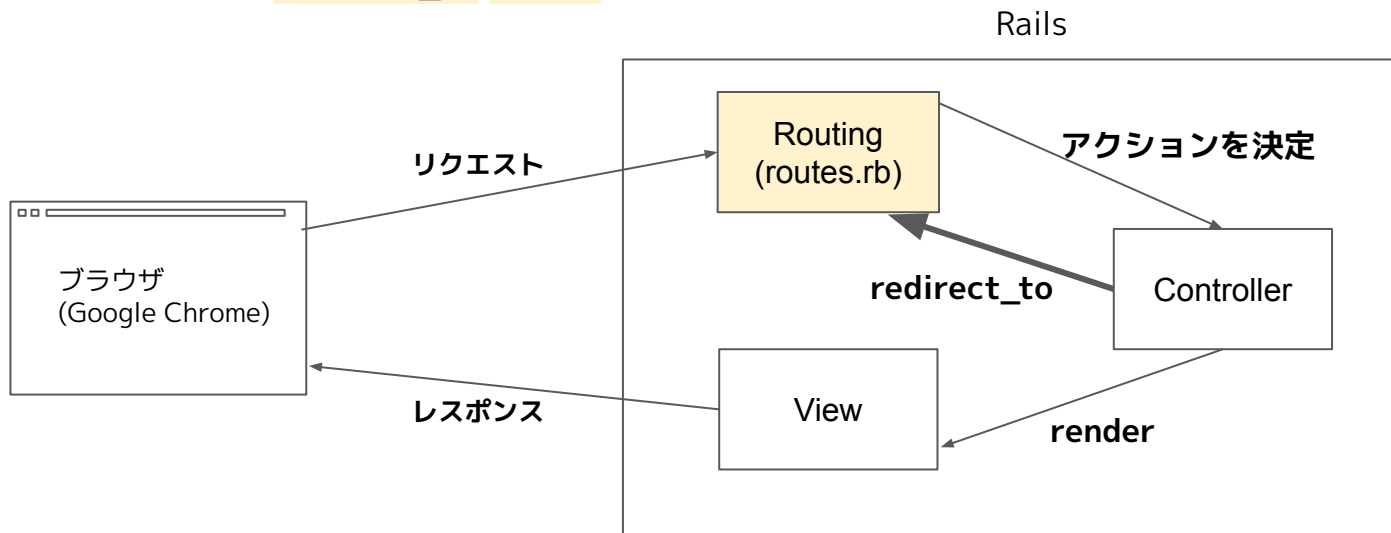
1. そもそもRailsは何をしているのか
- 2. Rails内部の流れ**
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ Rails内部の流れ ~

Rails内部の流れ

※今回はカリキュラムの内容に沿った項目のみ解説します。

1. Routingを元に呼び出すControllerとアクションを決める
2. Controllerを処理
3. render or `redirect_to` を行う



目次

1. そもそもRailsは何をしているのか
- 2. Rails内部の流れ**
3. 実際に作成、動かしてみる
4. ターミナル

step2. Railsの動作の流れ ~ VCの作成とRoutingの設定 / 動作の確認 ~

実際に作成して動かして確認

これから、実際にRoutingとVCを作成して、その動作を確認してみます。

- 内容
 - topというアクションを起動する
 - top.html.erbをブラウザに返す

- 手順

※分かりやすさを重視するため、あえて効率悪く進める場面があります

1. Controllerの作成
2. Controllerにアクションを記述
3. Routingの設定
4. rails routesコマンドで、Routingの確認
5. Controllerにアクションの処理を記述
6. Viewの作成(書き方によっては自動生成される)
7. 実際に動かしてみる

目次

1. そもそもRailsは何をしているのか
2. Rails内部の流れ
- 3. 実際に作成、動かしてみる**
4. ターミナル

step2. Railsの動作の流れ ~ VCの作成とRoutingの設定 / 動作の確認 ~

ターミナル

- RailsでのMVCの動作を確認出来るツール
- Railsで何かを作成する上で**非常に重要**

実際に先ほど作成したアプリの動作を、ターミナルでも確認してみます。

目次

1. そもそもRailsは何をしているのか
2. Rails内部の流れ
3. 実際に作成、動かしてみる
4. **ターミナル**

step2. Railsの動作の流れ ~ まとめ ~

まとめ

Step2では、以下の学習をしました。

1. Railsはリクエストを受けて、レスポンスを返す
2. Rails内部ではRouting -> Controller -> Viewの順で処理している
3. Controllerでredirect_toを行うと、Routingから新たに処理を開始する
4. ターミナルで動作を確認出来る

step3

~VCとインスタンス変数~

- step3では、いよいよ先ほどまでの知識をどう使うのか説明していきます。
- ControllerとViewのコード書き方を順を追って説明していきます。

step3. VCとインスタンス変数 ~ Controllerの記述 ~

ControllerにRubyのコードを書いてみる

手順

1. step1で作成した、full_nameメソッドをController内に作成する
2. topアクション内でfull_nameメソッドを呼び出して、ターミナルに出力

補足

- railsのコントローラーでもpを使用できる
- アクション内で、同じコントローラー内のメソッドを呼び出すことが可能

目次

1. Controllerにコードを記述
2. Viewにコードを記述
3. インスタンス変数

Modelにも同様の記述が出来る

このstepでは触れませんが、Modelにも、Controllerと同じようにRubyのコードが自由に記述できます。

step3. VCとインスタンス変数 ~ Viewの記述 ~

ViewにRubyのコードを書いてみる

- Viewは.rbファイルではなく、html.erbファイルのため、直接はかけない
 - Rubyの処理を書く場合は<% %>で囲う
 - Rubyの処理結果をHTMLに反映させたい場合は<%= %>を使う
- 以下のコードを書いて、実際にViewでRubyが動くか確認する

```
result = 1
result = result + 2
#この結果をh1タグの内容として出力
```

1. resultという変数に、1を代入する
2. さらにresultに2を足す
3. それをHTMLのh1タグとして、HTMLに出力する

目次

1. Controllerにコードを記述
2. **Viewにコードを記述**
3. インスタンス変数

step3. VCとインスタンス変数 ~ インスタンス変数 ~

インスタンス変数

※インスタンス変数というのは、本来Rubyの重要な仕組みの1つなのですが、今回はRailsではどのように動作するか、という事のみに焦点を当てて解説していきます。

- 変数の最初に@をつけると、インスタンス変数になります
- インスタンス変数とは、ControllerからViewにオブジェクトを渡せる変数
 - ControllerからrenderしたViewに、中身が引き継がれる
- インスタンス変数は、定義したメソッドの外でも使える(今回は解説しません)

目次

1. Controllerにコードを記述
2. Viewにコードを記述
3. **インスタンス変数**

注意点

1. インスタンス変数は、Viewの作成が完了する、またはリダイレクトするまで残り続ける → 乱用するとコードが書きづらくなったり、処理が重くなる
2. 定義していないインスタンス変数を記述した時、その変数の中身はnil(空)になる → この仕様を知らないと、ミスに気が付きにくい

step3. VCとインスタンス変数 ~ インスタンス変数 ~

確認する

実際のコードで、以下の事を確認する

- インスタンス変数の中身が、ControllerからViewに引き継がれる
- 定義していないインスタンス変数の中身はnilになる

目次

1. Controllerに
コードを記述
2. Viewにコードを
記述
3. **インスタンス変
数**

step3. VCとインスタンス変数 ~ まとめ ~

まとめ

Step3では、以下の学習をしました。

1. MVCにはRubyのコードを自由にかける
2. ViewにRubyのコードを書くときは、`<% %>` または `<%= %>` の記述が必要
3. インスタンス変数を利用すると、ControllerからViewに値の受け渡しが可能

質疑応答

step4

~データベースとModel~

- step4では、データベースをRailsでどのように扱うかを学んでいきます。
- このstepは、step1, 2と同じような、知識の説明になるのですが、step5でどう役立つのかが分かるので、耐えて覚えてください！

step4. データベースとModel ~ データベース ~

データベースとは

- データベースとは、データを保存しておくシステムの事
- Railsの機能ではなく、使われている言語もRubyではない
- しかしRailsでは、データベースの知識があまりなくともデータベースを扱える工夫がたくさんある

なので今回は最低限の知識のみ解説いたします。

目次

1. データベース

2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

step4. データベースとModel ~ データベース ~

データは表のように保存されている

- データベースのデータは、下の図のExcelの表のように保存されている
- 仕様
 - テーブル: 表の事、テーブル名は表が何に対しての情報なのかを表す
 - カラム: 列の事、カラム名はどのような情報かを表す(名前ならname)
 - レコード: データ1行分の事
 - 値: レコードのカラムの中身

The diagram shows a table with the following structure and annotations:

- カラム (Column):** Indicated by a blue box and arrow pointing to the 'id' column header.
- テーブル名 (Table Name):** Indicated by a red box and arrow pointing to the 'lists' header.
- カラム名 (Column Name):** Indicated by a blue box and arrow pointing to the 'id' column header.
- レコード (Record):** Indicated by a yellow box and arrow pointing to the first data row (1, 勉強, Railsの基礎).
- 値 (Value):** Indicated by a green box and arrow pointing to the value '16時から新宿でデート' in the 'boby' column of the third row.

lists	id	title	boby
	1	勉強	Railsの基礎
	2	息抜き	ギターの練習の続き
	3	休日	16時から新宿でデート

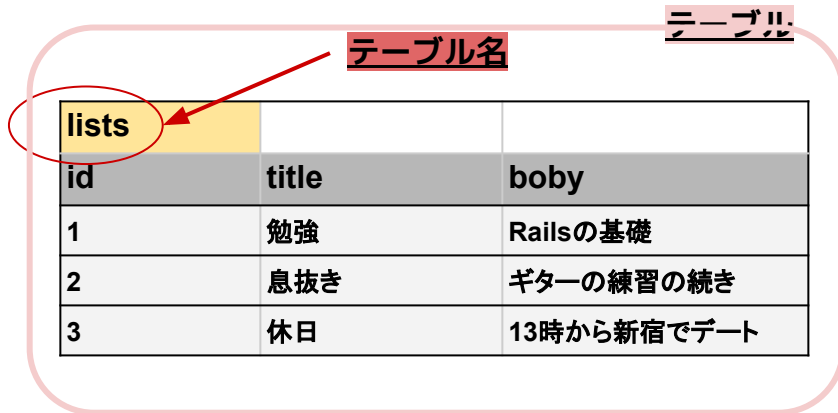
目次

1. データベース
2. データベースを Railsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

step4. データベースとModel ~ データベース ~

データは表のように保存されている

- データベースのデータは、Excelの表のように保存されている
- 仕様
 - テーブル: 表の事、テーブル名は表が何に対しての情報なのかを表す
 - カラム: 列の事、カラム名はどのような情報かを表す(名前ならname)
 - レコード: データ1行分の事
 - 値: レコードのカラムの中身



テーブル		
lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	13時から新宿でデート

目次

1. データベース
2. データベースを Railsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

step4. データベースとModel ~ データベース ~

データは表のように保存されている

- データベースのデータは、Excelの表のように保存されている
- 仕様
 - テーブル: 表の事、テーブル名は表が何に対しての情報なのかを表す
 - カラム: 列の事、カラム名はどのような情報かを表す(名前ならname)
 - レコード: データ1行分の事
 - 値: レコードのカラムの中身

目次

1. データベース

2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

カラム

カラム名

lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	13時から新宿でデート

step4. データベースとModel ~ データベース ~

データは表のように保存されている

- データベースのデータは、Excelの表のように保存されている
- 仕様
 - テーブル: 表の事、テーブル名は表が何に対しての情報なのかを表す
 - カラム: 列の事、カラム名はどのような情報かを表す(名前ならname)
 - レコード: データ1行分の事
 - 値: レコードのカラムの中身

レコード

lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	13時から新宿でデート

目次

1. データベース

2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

step4. データベースとModel ~ データベース ~

データは表のように保存されている

- データベースのデータは、Excelの表のように保存されている
- 仕様
 - テーブル: 表の事、テーブル名は表が何に対しての情報なのかを表す
 - カラム: 列の事、カラム名はどのような情報かを表す(名前ならname)
 - レコード: データ1行分の事
 - 値: レコードのカラムの中身

lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	16時から新宿でデート

値

目次

1. データベース

2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

step4. データベースとModel ~ データベースをRailsで扱うには ~

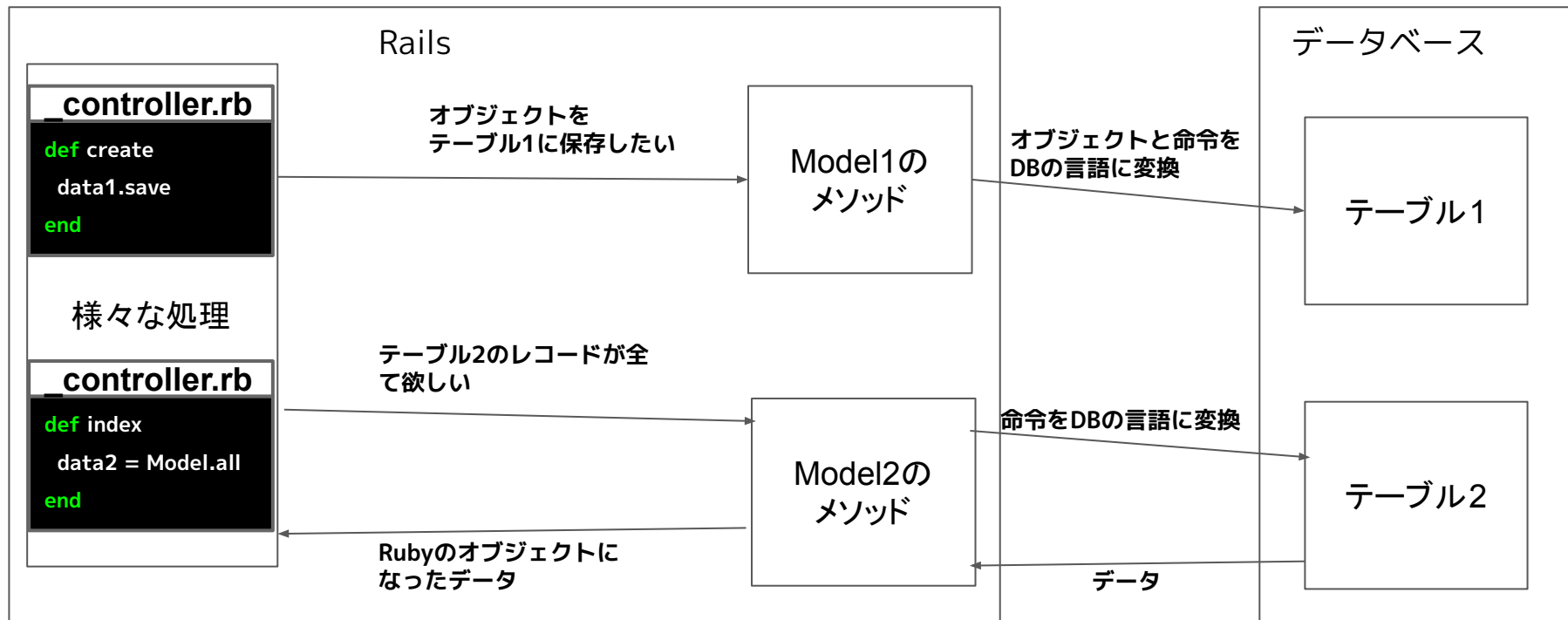
データベースをRailsで扱う方法

- データベースはRailsの機能ではなく、あの表を直接Ruby(Rails)で扱う事はできない
- その問題を解消してくれるがModel
- Modelはデータベースのデータを、Rubyで扱えるオブジェクトにしたもののクラス(データの種類)
- 1つのModelに1つのテーブルが対応している
- Modelにある様々なメソッドは、データベースのデータ \Leftrightarrow Rubyのオブジェクト という風に相互変換してくれる

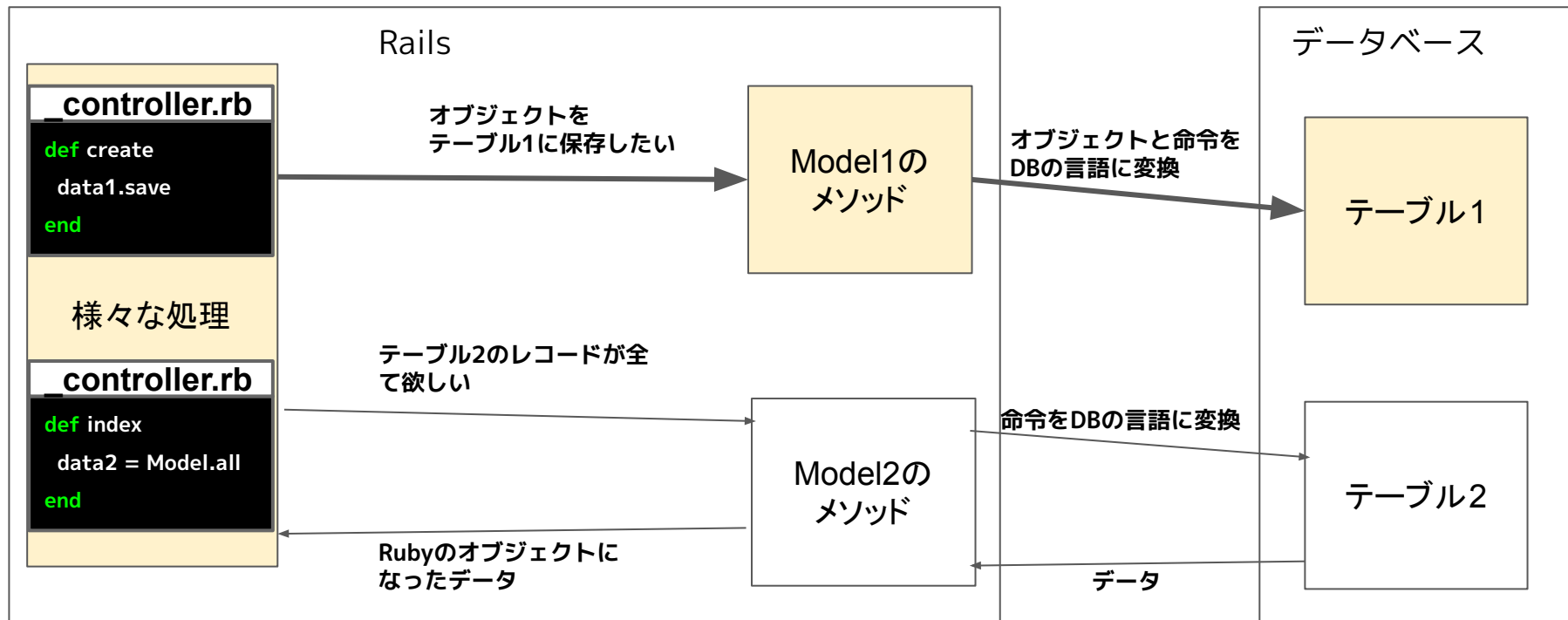
目次

1. データベース
2. データベースをRailsで扱うには
3. Modelの基本的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認

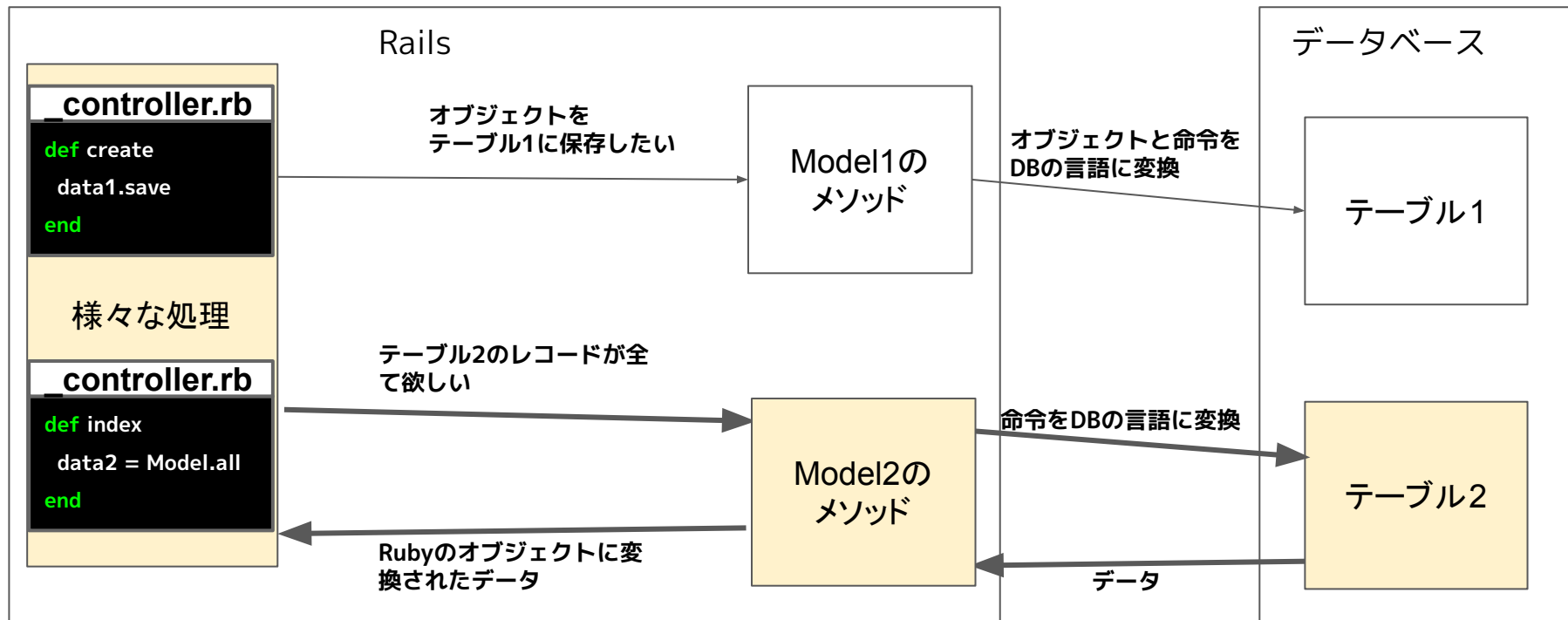
step4. データベースとModel ~ データベースをRailsで扱うには ~



step4. データベースとModel ~ データベースをRailsで扱うには ~



step4. データベースとModel ~ データベースをRailsで扱うには ~



step4. データベースとModel ~ データベースをRailsで扱うには ~

Rubyのオブジェクト

~

データ型	クラス名	データの例	機能名	機能
数値	Integer	1	+	足し算をする
	Float	1.0	.to_s	文字列に変換したオブジェクトを作る
文字列	String	"abc"	+	文字同士をくっつける
シンボル	Symbol	:abc		あまりない(その分文字列よりも容量が軽い)
配列	Array	[1, 1.0, "abc", :abc]	[番号] .each	指定した列の位置のオブジェクトを取り出す オブジェクトを1つずつ取り出して、ループ処理を行う
ハッシュ	Hash	{"abc" => 1, :abc => 2}	[鍵] .each	指定した鍵に対応する値のオブジェクトを取り出す 鍵と値を1つずつ取り出して、ループ処理を行う

この他に自分で新しい種類のオブジェクトを作成する事もできます。

step4. データベースとModel ~ Modelの基礎的なメソッド ~

Modelのメソッド

呼び出し元	メソッド名	引数	戻り値	処理の内容
Model自体	new	カラムにあらかじめ入れておく値	カラムが全て空のオブジェクトに、引数を入れたもの	空のオブジェクトを作成 引数がある場合は、それをカラムの値として入れる
	find	探したいレコードのidカラムの値	引数と一致しているidを持つレコード	引数と一致しているidを持つレコードをテーブルから探す 引数が文字列だった場合、数値に変換してくれる
	all	-	全てのレコード	全てのレコードを取得する
Modelのオブジェクト	カラム名	-	カラムの値	そのオブジェクトのカラムの値を返す
	カラム名 =	-	代入した値	そのオブジェクトのカラムに値を代入する
	save	-	true(成功時)	オブジェクトをテーブルに保存
	update	更新するカラム内容	false(失敗時)	既にテーブルに保存されているオブジェクトを更新
	destroy	-	削除したオブジェクト(成功時) false(失敗時)	オブジェクトをテーブルから削除
	errors	-	Errorsオブジェクト	エラーを取得

step4. データベースとModel ~ Modelの作成とメソッドの動きの確認 ~

実際にModelを作って動かしてみる

- Modelを作成した後、“find”と“カラム名”のメソッドを使う
※今回データベースを作るmigrationファイルの説明は割愛させていただきます。
- 手順
 - Modelの作成
 - migrate と テストデータの挿入(今回は解説しません)
 - Controllerのtopアクションでfindメソッドを使って、テストデータの1行目を取得する
 - 3をインスタンス変数に入れて、topのViewに受け渡す
 - 4で引き継いだオブジェクトの、titleとbodyカラムの中身を表示する

作成したテーブル

lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	13時から新宿でデート

目次

1. データベース
2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. **Modelの作成とメソッドの動きの確認**

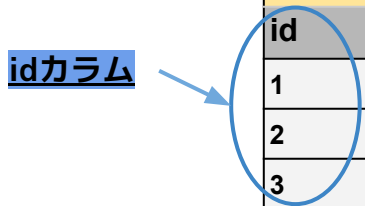
step4. データベースとModel ~ Modelの作成とメソッドの動きの確認 ~

Modelの補足 idカラム

- idカラムは、テーブル作成時に自動で設定されます
- idカラムの値は、1, 2, 3と順番に増えて行きます

目次

1. データベース
2. データベースをRailsで扱うには
3. Modelの基礎的なメソッド(機能)
4. Modelの作成とメソッドの動きの確認



lists		
id	title	boby
1	勉強	Railsの基礎
2	息抜き	ギターの練習の続き
3	休日	13時から新宿でデート

step4. データベースとModel ~ まとめ ~

まとめ

Step4では、以下の学習をしました。

1. データはデータベースのテーブルというところに、表の形で保存する
2. ModelはテーブルのデータをRubyで扱えるオブジェクトとして変換したもの
3. Modelのメソッドにより、Rubyのオブジェクトとデータベースのデータを簡単に変換して扱える
4. findメソッドを使えば、特定のidのデータを取得出来る
5. カラム名メソッドを使えば、オブジェクトのカラムの中身を取得出来る

step5

～パラメータ1 詳細画面～

- step5では、パラメータとはなんなのかと、詳細画面の仕組みについて説明していきます。

step5. パラメータ1 詳細画面 ~ パラメータ ~

鍵と値(keyとvalue)

- 鍵と値のオブジェクトがセットになっている
- やっている事はデータベースのカラムと値一緒
- key => value または key: valueと記述する

key => value

key: value

パラメータとは

- 次のアクションにデータを渡すための機能
- 文字列のみ渡す事が出来る
- 渡すための方法を2つ解説
- 1つ目は、URLにパラメータを設定する

目次

1. パラメータ
2. URLにパラメータを設定
3. パラメータを受け取る
4. 詳細画面の作成

step5. パラメータ1 詳細画面 ~ URLにパラメータを設定 ~

URLにパラメータを設定

RoutingのURLを設定する際、/:~/という部分を追加します

config/routes.rb

```
Rails.application.routes.draw do
  get 'lists/:id' => 'lists#show', as: 'list'
end
```

目次

1. パラメータ
2. URLにパラメータを設定
3. パラメータを受け取る
4. 詳細画面の作成

step5. パラメータ1 詳細画面 ~ URLにパラメータを設定 ~

パラメータを受け取る

- パラメータはControllerやViewで使用可能
- 以下の記述で取得出来る

```
app/controllers/~_controller.rb
```

```
def action名  
  params[鍵]  
end
```

- 鍵の名前は、URLの時に設定した:~の部分

実際にパラメータが渡る様子を見てみましょう、以下手順です

1. showアクションの作成
2. Controllerでパラメータを受け取る
3. 2をインスタンス変数にいれて、Viewに表示させる
4. 確認

目次

1. パラメータ
2. URLにパラメータを設定
3. **パラメータを受け取る**
4. 詳細画面の作成

step5. パラメータ1 詳細画面 ~ 詳細画面の作成 ~

詳細画面(show)の作成

- パラメータとfindメソッドを組み合わせる
- 手順
 1. ControllerのshowアクションにListモデルのfindメソッドを書く
 2. findメソッドの引数をパラメータにする
 3. 2をインスタンス変数に入れて、Viewに受け渡す
 4. 3のカラムの値をViewに表示
 5. ターミナルを見ながら動作確認

目次

1. パラメータ
2. URLにパラメータを設定
3. パラメータを受け取る
- 4. 詳細画面の作成**

step5. パラメータ1 詳細画面 ~ パラメータを設定したURLへのリンク ~

パラメータの補足 パラメータ入りのpathの書き方

- パラメータ入りのurlへのpathは、引数にパラメータとして渡したい値を記述する
- step6の最後で、再度説明します。

```
app/views/~.html.erb
```

```
<%= link_to 'リンク', パス名_path(渡したい値) %>
```

目次

1. パラメータ
2. URLにパラメータを設定
3. パラメータを受け取る
4. **詳細画面の作成**

step5. パラメータ1 詳細画面 ~ まとめ ~

まとめ

Step5では、以下の学習をしました。

1. 鍵と値はペア
2. パラメータは、次のアクションにデータを送れる仕組み
3. URLに/:~を記述することで、パラメータを設定出来る
4. params[鍵]でパラメータを受け取れる
5. ターミナル4行目付近でパラメータを確認
6. findメソッドと組み合わせる事で詳細画面を作成可能
7. pathで、パラメータ付きURLへパラメータを渡す時は、_path(渡したい値)と記述

質疑応答

step6

～パラメータ2 投稿機能～

- step6では、パラメータのもう一つの送り方を利用した投稿機能について、必要な知識を説明していきます。

step6. パラメータ2 投稿機能 ~ パラメータのもう一つの送り方 ~

パラメータのもう一つの送り方

- パラメータを渡す方法2つ目は、formから送信
- form(form_with)とは、パラメータを入力して送れるlink_toのようなもの
- form_withとストロングパラメータの記述ルールを守る事で、スムーズなデータの保存が可能

目次

1. **パラメータのもう一つの送り方**
2. form_with
3. ストロングパラメータ
4. 投稿機能作成

step6. パラメータ2 投稿機能 ~ form_with ~

form_withの記述

- form_withを記述 model, url, method, localの4項目の鍵とその値を設定する
 - modelは、ストロングパラメータを使用する際に必要な記述
 - 新規投稿の場合の値はModel.new
 - 編集の場合の値はfind等で取ってきた、編集対象のオブジェクト
 - urlとmethodの値は、遷移先のリクエスト(URL + HTTPメソッド)
 - リクエストは特定の条件で省略可能
 - localは、1ヶ月目の段階では必ずtrueを指定
- 送りたいデータを入力するタグと、パラメータの鍵を記述
 - ストロングパラメータを使用する際は、鍵の名前をカラム名にする
 - タグには文字入力フォーム、プルダウンなど色々な種類がある
- 遷移するためにsubmitボタンを記述する

目次

1. パラメータのもう一つの送り方
2. **form_with**
3. ストロングパラメータ
4. 投稿機能作成

app/views/~.html.erb

```
<%= form_with model: モデルのオブジェクト, url: 遷移先のurl, method: 遷移先のHTTPメソッド, local: true, do |f| %>
  <%= f.タグ :鍵 #カラム名 %>
  <%= f.submit '表示する文字' %>
<% end %>
```

step6. パラメータ2 投稿機能 ~ form_with ~

form_withの記述

- form_withを記述 model, url, method, localの4項目の鍵とその値を設定する
 - modelは、ストロングパラメータを使用する際に必要な記述
 - 新規投稿の場合の値はModel.new
 - 編集の場合の値はfind等で取ってきた、編集対象のオブジェクト
 - urlとmethodの値は、遷移先のリクエスト(URL + HTTPメソッド)
 - リクエストは特定の条件で省略可能
 - localは、1ヶ月目の段階では必ずtrueを指定
- 送りたいデータを入力するタグと、パラメータの鍵を記述
 - ストロングパラメータを使用する際は、鍵の名前をカラム名にする
 - タグには文字入力フォーム、プルダウンなど色々な種類がある
- 遷移するためにsubmitボタンを記述する

目次

1. パラメータのもう一つの送り方
2. **form_with**
3. ストロングパラメータ
4. 投稿機能作成

app/views/~.html.erb

```
<%= form_with model: モデルのオブジェクト, url: 遷移先のurl, method: 遷移先のHTTPメソッド, local: true, do |f| %>
  <%= f.タグ :鍵 #カラム名 %>
  <%= f.submit '表示する文字' %>
<% end %>
```

step6. パラメータ2 投稿機能 ~ form_with ~

form_withの記述

- form_withを記述 model, url, method, localの4項目の鍵とその値を設定する
 - modelは、ストロングパラメータを使用する際に必要な記述
 - 新規投稿の場合の値はModel.new
 - 編集の場合の値はfind等で取ってきた、編集対象のオブジェクト
 - urlとmethodの値は、遷移先のリクエスト(URL + HTTPメソッド)
 - リクエストは特定の条件で省略可能
 - localは、1ヶ月目の段階では必ずtrueを指定
- 送りたいデータを入力するタグと、パラメータの鍵を記述
 - ストロングパラメータを使用する際は、鍵の名前をカラム名にする
 - タグには文字入力フォーム、プルダウンなど色々な種類がある
- 遷移するためにsubmitボタンを記述する

目次

1. パラメータの送りの送り方
2. **form_with**
3. ストロングパラメータ
4. 投稿機能作成

app/views/~.html.erb

```
<%= form_with model: モデルのオブジェクト, url: 遷移先のurl, method: 遷移先のHTTPメソッド, local: true, do |f| %>
  <%= f.tag :鍵 #カラム名 %>
  <%= f.submit '表示する文字' %>
<% end %>
```

step6. パラメータ2 投稿機能 ~ form_with ~

form_withの記述

- form_withを記述 model, url, method, localの4項目の鍵とその値を設定する
 - modelは、ストロングパラメータを使用する際に必要な記述
 - 新規投稿の場合の値はModel.new
 - 編集の場合の値はfind等で取ってきた、編集対象のオブジェクト
 - urlとmethodの値は、遷移先のリクエスト(URL + HTTPメソッド)
 - リクエストは特定の条件で省略可能
 - localは、1ヶ月目の段階では必ずtrueを指定
- 送りたいデータを入力するタグと、パラメータの鍵を記述
 - ストロングパラメータを使用する際は、鍵の名前をカラム名にする
 - タグには文字入力フォーム、プルダウンなど色々な種類がある
- 遷移するためにsubmitボタンを記述する

目次

1. パラメータの送りの送り方
2. **form_with**
3. ストロングパラメータ
4. 投稿機能作成

app/views/~.html.erb

```
<%= form_with model: モデルのオブジェクト, url: 遷移先のurl, method: 遷移先のHTTPメソッド, local: true, do |f| %>
  <%= f.タグ :鍵 #カラム名 %>
  <%= f.submit '表示する文字' %>
<% end %>
```

step6. パラメータ2 投稿機能 ~ form_with ~

form_withの例

```
<%= form_with model: List.new, local: true do |f| %>  
  <%= f.text_field :title %>  
  <%= f.select :genre ,[['a', 'a'], ['b', 'b']]%>  
  <%= f.submit '投稿' %>  
<% end %>
```

The diagram illustrates the rendered HTML form. It consists of a text input field for the title, a dropdown menu for the genre (with options 'a' and 'b'), and a submit button labeled '投稿'. Colored boxes and arrows link the ERB code to the rendered form elements: a blue box for the title field, a green box for the genre dropdown, and a red box for the submit button.

目次

1. パラメータの
一つの送り方
2. **form_with**
3. ストロングパラ
メータ
4. 投稿機能作成

リクエストを省略出来る条件

- Routingをresourcesで指定している事
- Controller名が、Model名の複数系になっている事

※こちらは正確なルールではありません

step6. パラメータ2 投稿機能 ~ ストロングパラメータ ~

ストロングパラメータ

- データ送信時の不正を防ぐため、保存するカラムを制限する仕組み
- ストロングパラメータは Controllerに定義するメソッド
 - `params.require(モデル名の小文字).permit(許可したいカラム名)`
 - 安全に簡単にフォームのデータを、Modelのオブジェクトに反映出来る
- `private`以下に書く
 - `private`以下のメソッドは、同じ Controller内でしか呼び出せなくなるため、セキュリティが向上する

記述ルール

app/controllers/~_controller.rb

```
...  
def create  
end  
  
private  
def ストロングパラメータのメソッド名  
  params.require(モデル名小文字).permit(許可したいカラム名)  
end
```

目次

1. パラメータのも
う一つの送り方
2. `form_with`
3. **ストロングパラ
メータ**
4. 投稿機能作成

step6. パラメータ2 投稿機能 ~ ストロングパラメータ ~

ストロングパラメータを使ってformのデータを保存する方法

手順

1. ストロングパラメータの戻り値を newメソッドの引数として渡す
2. 上記のオブジェクトから saveメソッドを呼び出す

例

目次

1. パラメータのも
う一つの送り方
2. form_with
3. **ストロングパラ
メータ**
4. 投稿機能作成

app/controllers/~_controller.rb

```
...  
def create  
  list = List.new(list_params)  
  list.save  
end  
  
private  
def list_params  
  params.require(:list).permit(:title, :body)  
end
```


step6. パラメータ2 投稿機能 ~ ストロングパラメータ ~

ストロングパラメータを使ってformのデータを保存する方法

手順

1. ストロングパラメータの戻り値を newメソッドの引数として渡す
2. 上記のオブジェクトから saveメソッドを呼び出す

例

app/controllers/~_controller.rb

```
...  
def create  
  list = List.new(list_params)  
  list.save  
end  
  
private  
def list_params  
  params.require(:list).permit(:title, :body)  
end
```

目次

1. パラメータのも
う一つの送り方
2. form_with
3. ストロングパラ
メータ
4. 投稿機能作成

step6. パラメータ2 投稿機能 ~ ストロングパラメータ ~

ストロングパラメータを使ってformのデータを保存する方法

手順

1. ストロングパラメータの戻り値を newメソッドの引数として渡す
2. 上記のオブジェクトから saveメソッドを呼び出す

例

目次

1. パラメータのもう一つの送り方
2. form_with
3. ストロングパラメータ
4. 投稿機能作成

app/controllers/~_controller.rb

```
...  
  
def create  
  list = List.new(list_params)  
  list.save  
end  
  
private  
  
def list_params  
  params.require(:list).permit(:title, :body)  
end
```

step6. パラメータ2 投稿機能 ~ 投稿機能作成 ~

投稿機能を作成

- form_with, new, ストロングパラメータ, saveを組合わせる
 - 手順
1. 投稿を保存する処理を行う createアクションの作成
 2. topページにform_withを記述
 3. formからのデータを保存する処理を書く
 - a. ストロングパラメータを作成
 - b. createアクションでパラメータを入れた List.newを作成
 - c. createアクションで5をsave
 4. showページへ遷移する記述を書く
 5. 確認

目次

1. パラメータのも
う一つの送り方
2. form_with
3. ストロングパラ
メータ
4. **投稿機能作成**

step6. パラメータ2 投稿機能 ~ まとめ ~

まとめ

Step6では、以下の学習をしました。

1. formを利用して、パラメータを渡す事が可能
2. form_withでは、4つの鍵と値のペア、データを入力するタグ、遷移ボタンになるsubmitを記述する
3. Model.newの引数にストロングパラメータの戻り値を渡すことで、formから送られてきたデータを簡単にオブジェクトにする事ができます
4. 3からsaveメソッドを呼び出す事で、データを保存する

step7

～バリデーション～

- step7では、最後の要素“バリデーションとエラーメッセージ”について解説していきます。

step7. バリデーション ~ バリデーション ~

バリデーションとは

- Modelの機能
- カラム毎に、値を制限できる
 - 空は禁止等

記述

- validates カラム名, 制限を記述
- かける制限、制限の種類を鍵, 内容を値とした鍵と値のペアを記述

```
app/models/~.rb
```

```
class モデル名
  validates :カラム名, 制限の種類: 値
end
```

目次

1. バリデーション
2. エラーメッセージ
3. 投稿失敗時にエラーメッセージを表示

step7. バリデーション ~ バリデーション ~

バリデーションの一例

鍵	値	説明
presence	true	空欄を禁止する
uniqueness	true	値の被りを禁止する
length	文字数の対象を鍵 文字数を値としたハッシュ	文字数を制限する
exclusion	inを鍵、禁止したい値の配列を値としたハッシュ	特定の値を禁止する

目次

1. バリデーション
2. エラーメッセージ
3. 投稿失敗時にエラーメッセージを表示

制限をかける

先ほどの投稿機能に、空白禁止の制限を追加してみます。

step7. バリデーション ~ エラーメッセージ ~

エラーメッセージ

- save等が失敗した時に、Errorsオブジェクト(以下errors)というものが追加されます
- このerrorsに含まれているメッセージが、エラーメッセージ
- save出来なかったオブジェクト.errors.full_messagesで、エラーメッセージの配列を取得
- このerrorsというのは、Modelのメソッドです(次スライド)

目次

1. バリデーション
2. エラーメッセージ
3. 投稿失敗時にエラーメッセージを表示

エラーメッセージの取得

- 先ほどのsaveに失敗した時の処理に、エラーメッセージをターミナルに表示する処理を加える

step7. バリデーション ~ エラーメッセージ ~

Modelのメソッド

呼び出し元	メソッド名	引数	戻り値	処理の内容
Model自体	new	カラムにあらかじめ入れておく値	カラムが全て空のオブジェクトに、引数を入れたもの	空のオブジェクトを作成 引数がある場合は、それをカラムの値として入れる
	find	探したいレコードのidカラムの値	引数と一致しているidを持つレコード	引数と一致しているidを持つレコードをテーブルから探す 引数が文字列だった場合、数値に変換してくれる
	all	-	全てのレコード	全てのレコードを取得する
Modelのオブジェクト	カラム名	-	カラムの値	そのオブジェクトのカラムの値を返す
	カラム名 =	-	代入した値	そのオブジェクトのカラムに値を代入する
	save	-	true(成功時)	オブジェクトをテーブルに保存
	update	更新するカラム内容	false(失敗時)	既にテーブルに保存されているオブジェクトを更新
	destroy	-	削除したオブジェクト(成功時) false(失敗時)	オブジェクトをテーブルから削除
	errors	-	Errorsオブジェクト	エラーを取得

step7. バリデーション ~ 投稿失敗時にエラーメッセージを表示 ~

投稿失敗時にエラーメッセージをViewに表示する

目次

1. バリデーション
2. エラーメッセージ
3. 投稿失敗時にエラーメッセージを表示

- バリデーション、errors、renderを使用する
 - renderを使用して、createアクションでtop.html.erbを表示しなおす
 - redirect_toを使用してtopページに遷移すると、errorsが追加されたオブジェクトの変数がなくなってしまう
- step6で作成した投稿機能に、エラーメッセージ機能を足す
- 手順
 1. save失敗時に、投稿画面をrenderされる処理を記述する
 2. インスタンス変数を使って、render先にエラーメッセージを渡す
 3. Viewにエラーメッセージを表示させる記述をする

step7. パラメータ2 投稿機能 ~ まとめ ~

まとめ

Step7では、以下の学習をしました。

1. バリデーションを設定して、カラムの値に制限をかけれる
2. save等の処理が失敗した際、オブジェクトにerrorsが追加される
3. errorsの中にはエラーメッセージが含まれている
4. バリデーション、エラーメッセージ、renderを使用して、投稿失敗時にエラーメッセージを表示できます

質疑応答