

第9章

日付と時刻

この章では、Ruby on Rails で日付時刻型のデータを扱う方法について学習します。

9.1 マイグレーションスクリプトの書き換え

第4章で `plan_items` テーブルに必要となりそうな列をリストアップしました。忘れてしまったかもしれないので、もう一度ここに掲載します (表 9.1)。

表 9.1 `plan_items` テーブルの列 (再掲)

名称	型	用途
<code>name</code>	<code>string</code>	件名
<code>description</code>	<code>text</code>	説明
<code>starts_at</code>	<code>datetime</code>	開始日時
<code>ends_at</code>	<code>datetime</code>	終了日時

後半の二つがまだ定義されていません。`datetime` は日付と時刻を格納するための型です。

では、`starts_at` 列と `ends_at` 列を `plan_items` テーブルに追加しましょう。このテーブルのマイグレーションスクリプトを次のように書き換えてください。

第9章 日付と時刻

```
db/migrate/20160813144526_create_plan_items.rb
```

```
1 class CreatePlanItems < ActiveRecord::Migration[5.0]
2   def change
3     create_table :plan_items do |t|
4       t.string :name, null: false
5       t.text :description, null: false
6 +     t.datetime :starts_at, null: false
7 +     t.datetime :ends_at, null: false
8
9       t.timestamps
:
:
```

そして、次のコマンドを実行します。

```
$ rails db:migrate:reset
```

ターミナルに次のようなメッセージが出力されます。

```
Dropped database 'db/development.sqlite3'
Dropped database 'db/test.sqlite3'
Created database 'db/development.sqlite3'
Created database 'db/test.sqlite3'
== 20160813144526 CreatePlanItems: migrating =====
-- create_table(:plan_items)
   => 0.0017s
== 20160813144526 CreatePlanItems: migrated (0.0017s) =====
```

PlanItem クラスのソースコードに書かれた注釈を更新します。

```
$ bin/annotate
```

plan_item.rb のコードが次のように書き換わっていることを確認してください。

```
app/models/plan_item.rb
```

```
:
7 # description :text           not null
8 + # starts_at  :datetime      not null
9 + # ends_at    :datetime      not null
```

```
10 # created_at :datetime          not null
:
```

日付と時刻を格納するための型の名前は DBMS によって異なります。SQLite3 と MySQL では `datetime` と呼ばれますが、PostgreSQL では `timestamp without time zone` という名前になります。しかし、近過去から近未来までの日付を扱う限りは違いを意識する必要はありません。もっとも制限の強い MySQL の `datetime` 型でも西暦 1000 年から 9999 年までを扱えます。

9.2 シードデータの書き換え

シードデータを投入してみましょう。

```
$ rails db:seed
```

すると、次のようなエラーメッセージが出ます。

```
rails aborted!
ActiveRecord::StatementInvalid: SQLite3::ConstraintException: NOT NULL >
constraint failed: plan_items.starts_at: INSERT INTO "plan_items" (">
name", "description", "created_at", "updated_at") VALUES (?, ?, ?, ?>
)
...
```

`starts_at` 列と `ends_at` 列には NOT NULL 制約が設定されているので、現行の `db/seeds.rb` はうまく動きません。次のように書き換えてください。

```
db/seeds.rb
1 + time0 = Time.current.beginning_of_day
2 +
3   item = PlanItem.new
4   item.name = '読書'
5   item.description = '『走れメロス』を読む'
```

第9章 日付と時刻

```
6 + item.starts_at = time0.advance(days: 1, hours: 10)
7 + item.ends_at = time0.advance(days: 1, hours: 11)
8   item.save!
9
10  item = PlanItem.new
11  item.name = '買い物'
12  item.description = '洗剤を買う'
13 + item.starts_at = time0.advance(hours: 16)
14 + item.ends_at = time0.advance(hours: 16, minutes: 30)
15  item.save!
16
17  item = PlanItem.new
18  item.name = '帰省'
19  item.description = "新幹線の指定席を取る。\\n お土産を買う。"
20 + item.starts_at = time0.beginning_of_year.advance(years: 1, days: -2)
21 + item.ends_at = time0.beginning_of_year.advance(years: 1, days: 3)
22  item.save!
```

コードの説明は後回しにしましょう。シードデータを投入してください。

```
$ rails db:seed
```

さきほどのようなエラーメッセージが出なければ OK です。

9.3 日付時刻オブジェクトの操作

では、`db/seeds.rb` の変更箇所を見ていきましょう。まず、1 行目。

```
time0 = Time.current.beginning_of_day
```

`Time.current` は現在の日付と時刻を返します。このメソッドの戻り値は `ActiveSupport::TimeWithZone` クラスのインスタンスです。このオブジェクトは、日付、時刻、および時間帯 (time zone) を保持しています。時間帯に関しては後で説明します。

以後、`ActiveSupport::TimeWithZone` クラスのインスタンスを「日付時刻オブジェクト」と呼びます。

日付時刻オブジェクトの `beginning_of_day` メソッドは、その日の初め（午前 0 時 0 分）まで巻き戻した新しい日付時刻オブジェクトを返します。これを変数 `time0` にセットし、日時計算の基準として使用することにします。

6 行目でその変数 `time0` が使われています。

```
item.starts_at = time0.advance(days: 1, hours: 10)
```

日付時刻オブジェクトの `advance` メソッドは、日付と時刻を前に進めて新たな日付時刻オブジェクトを返します。引数にはハッシュを取ります。このハッシュに指定できるキーは `:years`, `:months`, `:weeks`, `:days`, `:hours`, `:minutes`, `:seconds` の七つです。上記の例では、1 日と 10 時間だけ時計の針を前に進めます。`time0` が今日の午前 0 時 0 分ですから、明日の午前 10 時になります。

20 行目では、ちょっと複雑な計算が行われています。

```
item.starts_at = time0.beginning_of_year.advance(years: 1, days: -2)
```

日付時刻オブジェクトの `beginning_of_year` メソッドは、その年の初め（1 月 1 日午前 0 時 0 分）まで巻き戻した新しい日付時刻オブジェクトを返します。そしてそのオブジェクトに対して `advance` メソッドを呼び出しています。`years: 1` で 1 年進めつつ、`days: -2` で 2 日戻しています。つまり、今年の 12 月 30 日の午前 0 時 0 分になります。

9.4 時間帯の設定

次のコマンドを実行してください。

```
$ rails r 'puts PlanItem.first.starts_at'
```

第9章 日付と時刻

`PlanItem.first` は `plan_items` テーブルの最初のレコードに対応する `PlanItem` オブジェクトを返します。そして、`starts_at` メソッドを呼べば予定の開始日時を取得できます。

コマンドを実行した結果、次のように出力されるはずです（ただし、日付部分は今日または昨日の日付になります）。

```
2016-10-05 10:00:00 UTC
```

末尾にある `UTC` という文字列は、この時刻が**協定世界時**（UTC）であることを示しています。日本標準時（JST）は協定世界時よりも 9 時間進んでいます。つまり、1 番目の予定の開始時刻は日本時間の午前 1 時であるということになります。

かつて国際的な基準時刻は**グリニッジ標準時**（Greenwich Mean Time）と呼ばれ、GMT と略されていましたが、現在、科学技術の分野では「協定世界時」という用語が使われます。ただし、テレビや新聞などでは「グリニッジ標準時」という言葉が依然として使われています。

共通の標準時を使う地域全体を**時間帯**（time zone）と言います。Rails では、地域名またはその地域の代表的な都市名を用いて時間帯を特定します。日本標準時を使用する時間帯は “Tokyo” と呼ばれます。

“Tokyo” の他に “Osaka”、“Sapporo”、“Seoul”、“Yakutsk” を用いることもできます。

`PicoPlanner` を日本標準時で運用する場合、アプリケーションの時間帯を “Tokyo” にセットします。`config/application.rb` を次のように書き換えてください。

```
config/application.rb
:
19 module PicoPlanner
20   class Application < Rails::Application
```

```
21 # Settings in config/environments/* take precedence over those >
    specified here.
22 # Application configuration should go into files in config/initializers
23 # -- all .rb files in that directory are automatically loaded.
24 +
25 + config.time_zone = 'Tokyo'
26 end
27 end
```

`config.time_zone` にセットできる時間帯の名前については、付録 C を参照してください。この付録では「夏時間」の扱い方についても簡単に解説しています。

そして、データベースを作り直します。

```
$ rails db:reset
```

もう一度、さきほどのコマンドを実行してみましょう。

```
$ rails r 'puts PlanItem.first.starts_at'
```

次のような結果が出れば OK です。

```
2016-10-05 10:00:00 +0900
```

末尾の `+0900` は、協定世界時との時差（プラス 9 時間）を示します。

■ コラム: 時間帯を設定する場所について

`config/application.rb` の 22 ~ 23 行に書いてある英文のコメントは、日本語で次のように翻訳できます。

アプリケーションの設定は `config/initializers` ディレクトリ `config/initializers` 内のファイルに書くべきである。そのディレクトリにあるすべての `.rb` ファイルは自動的に読み込まれる。

第9章 日付と時刻

とすれば、`config/initializers` ディレクトリに次のような内容を持つ `time_zone.rb` というファイルを作成してもうまく行きそうですが、実際には効果がありません。

```
Rails.application.config.time_zone = 'Tokyo'
```

実は、`config/application.rb` に書いてある設定が読み込まれた直後に、アプリケーションで使用される時間帯が決定され、その後で `config/initializers` ディレクトリ内のファイル群が読み込まれるようになっています。

このようにコメントの記述が実際の挙動とずれていますので、一種のバグかもしれません。筆者は Rails 5.0.0.1 でこの挙動を確認しましたが、将来のバージョンでは変更される可能性があります。