Crystal と Web

Crystal-JP、msky　著

2017-12-29 版　　発行

第 1 章

Crystal と Web

本稿では Crystal を Web 方面で活用する方法について記載します。本稿は技術書典 1 および 3 で公開した「Crystal の本」および「Crystal の本 3」で掲載されております、同稿の改訂版となります。Crystal および Kemal の安定バージョン（2017 年 12 月執筆時点）に準拠します。本サンプルで使用する Crystal のバージョン、および、Kemal のバージョンは以下の通りです。

- Crystal
  - 0.23.1
- Kemal
  - 0.20.0

1.1 Crystal で Web アプリを作成する

Crystal では標準で Web サーバの機能を提供しています。公式のサンプルをそのまま掲載しますが、以下のコードをビルド、実行するとブラウザから「http://localhost:8080」でアクセスするとそのまま「Hello World」が表示されます。

http_sample.cr

```
# A very basic HTTP server
require "http/server"

server = HTTP::Server.new(8080) do |context|
  context.response.content_type = "text/plain"
  context.response.print "Hello world"
end

puts "Listening on http://0.0.0.0:8080"
server.listen
```

ビルド、実行は以下の手順です。

```
crystal build --release http_sample.cr
./http_sample
```

もっと複雑な Web アプリを作成したい場合は、役立つフレームワークが幾つか有志の手で作られております。以前第 3 回 Crystal 勉強会で紹介させていただいた [Frost](https://github.com/ysbaddaden/frost) や、同勉強会で @pine613 さんが紹介された [Kemal](https://speakerdeck.com/pine613/crystal-teshi-ji-falseuehusahisuhazuo-rerufalseka) などが代表的ですが、他の情報については「[Awesome Crystal](https://github.com/veelenga/awesome-crystal)」で取り上げられているので本稿では割愛します。本稿では Kemal を主に例として挙げ、Crystal でのミニブログ機能を持つ Web アプリの制作方法を簡単に紹介します。

説明に使用したサンプルは以下に格納しております。

https://github.com/msky026/kemal-sample

1.2 Kemal

Crystal で現在最も活発に開発されている Web フレームワークが Kemal です。Ruby でいうところの Sinatra 風の設計を思想で作られており、シンプルな実装で Web アプリを作成することが出来ます。

- URL http://kemalcr.com/
- github https://github.com/kemalcr/kemal

Kemal インストール

適当な作業用のディレクトリ以下で、以下のコードを実行してみます。ディレクトリが作成され幾つか雛形が作成されます。（本稿ではサンプルアプリを「kemal-sample」として進めます）

```
crystal init app kemal-sample
cd kemal-sample
```

`shard.yml` ファイルに以下の内容を追記します。

```
dependencies:
  kemal:
    github: kemalcr/kemal
    version: 0.20.0
```

以下のコマンドを実行することで Kemal 本体をインストールすることが出来ます。

```
shards install
```

Kemal FirstStep

インストールまで問題なく動かせたら続いて簡単なサンプルを作成してみましょう。カレントの src ディレクトリ以下に以下の名前のファイルとディレクトリを作成します。

src/kemal-sample.cr

```
require "kemal"

# http://localhost:3000/ または
# http://localhost:3000/articles/ のどちらでもアクセスできるようにする
["/", "/articles"].each do |path|
  get path do |env|
    articles = [
      {"id" => 1, "title" => "title1", "body" => "body1"},
      {"id" => 2, "title" => "title2", "body" => "body2"},
      {"id" => 3, "title" => "title3", "body" => "body3"},
    ]

    render "src/views/index.ecr"
  end
end

Kemal.run
```

src/views/index.ecr

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>post item</title>
</head>
<body>
  <% articles.each do |article| %>
  <table>
    <tr>
      <td><%=article["title"] %></td>
    </tr>
    <tr>
      <td><%=article["body"] %></td>
    </tr>
  </table>
  <% end %>
```

```
</body>
</html>
```

記述したら、以下のコマンドでビルド、実行します。

```
crystal build --release src/kemal-sample.cr
./kemal-sample
```

ブラウザで「http://localhost:3000/」でアクセスします。以下の内容が表示されていれば取り敢えず問題ありません。

```
title1
body1
title2
body2
title3
body3
```

また、「http://localhost:3000/articles」でアクセスしても同様の表示がされます。

DB と連動する

Kemal で作ったアプリから DB にアクセスすることも出来ます。ライブラリを使用することで PostgreSQL か、もしくは MySQL を使用することが出来ます。今回は PostgreSQL を使用します。

テーブル名:artiles

カラム名 型
id serial
title text
body text

DB を作成します。

```
createdb kemal_sample -O your_owner
```

DDL を作成しロードします。

create_articles.sql

```
create table articles (
  id serial primary key,
  title  text,
  body   text
);
```

```
psql -U postgres -d kemal_sample -f create_articles.sql
```

作成した後、shard.yml ファイルに以下の内容を追記します。

```
dependencies:
  db:
    github: crystal-lang/crystal-db
  pg:
    github: will/crystal-pg
```

編集後、以下のコマンドを実行します。

```
shards install
```

投稿一覧ページの編集

本章からいよいよ Web アプリらしく記事の一覧ページと詳細ページ、新規投稿ページをそれぞれ作成していきます。また、全ページで共通で使用するテンプレートは別に作成し、テンプレートヘッダに新規投稿ページと投稿リストページへのリンクを表示し、ページ内に各ページのコンテンツを表示するように修正していきます。

レイアウトページの作成

まず雛形のページを ‘application.ecr‘という名前で作成します。

src/views/application.ecr

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
```

```
  <title>kemal sample</title>
  <!-- bootstrap を使用する -->
  <link
   rel="stylesheet"
   href="//maxcdn.bootstrapcdn.com/bootstrap /3.3.6/css/bootstrap.min.css">
  <link rel="stylesheet" href="/css/custom.css">
</head>
<body>
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <a id="logo">sample app</a>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="/articles">ArticleList</a></li>
        <li><a href="/articles/new">新規投稿</a></li>
      </ul>
    </nav>
  </div>
</header>
  <div class="container">
    <%= content %>
  </div>
</body>
</html>
```

本サンプルでは BootStrap を使用します。CDN を使用しますので特にダウンロード不要ですが、ダウンロードする場合は別途「 http://getbootstrap.com/ 」から必要なファイルをダウンロードし、プロジェクトカレントの/public 以下に配置してください。

CSS の作成

ページ修飾用の CSS を作成します。本サンプルでは rails tutorial(http://railstutorial.jp/) をそのまま参考にします。

public/css/custom.css

```
body {
  padding-top: 60px;
}

section {
  overflow: auto;
}

textarea {
  resize: vertical;
}

.center {
  text-align: center;
}
```

```
.center h1 {
  margin-bottom: 10px;
}

h1, h2, h3, h4, h5, h6 {
  line-height: 1;
}

h1 {
  font-size: 3em;
  letter-spacing: -2px;
  margin-bottom: 30px;
  text-align: center;
}

h2 {
  font-size: 1.2em;
  letter-spacing: -1px;
  margin-bottom: 30px;
  text-align: center;
  font-weight: normal;
  color: #777;
}

p {
  font-size: 1.1em;
  line-height: 1.7em;
}

#logo {
  float: left;
  margin-right: 10px;
  font-size: 1.7em;
  color: #fff;
  text-transform: uppercase;
  letter-spacing: -1px;
  padding-top: 9px;
  font-weight: bold;
}

#logo:hover {
  color: #fff;
  text-decoration: none;
}
```

一覧ページの修正

記事一覧ページを以下の内容で修正します。

src/views/index.ecr

```
<h2>Article List</h2>
<table class="table table-striped">
<thead>
  <tr>
    <td>title</td>
  </tr>
<tbody>
  <% articles.each do |article| %>
  <tr>
    <td>
      <a href="/articles/<%=article["id"]? %>" target="_top">
        <%=article["title"]? %>
      </a>
    </td>
  </tr>
  <% end %>
</tbody>
</table>
```

新規投稿ページ

記事の記事投稿用のページを新規作成します。

以下のファイルを追加します。

データ追加用画面

src/views/articles/new.ecr

```
<h2>新規投稿</h2>
<form method="post", action="/articles">
  <input type="text" name="title" size="10" maxlength="10" />
  <br />
  <br />
  <textarea name="body" cols="40" rows="4"></textarea>
  <br />
  <br />
  <input type="submit" value="post">
</form>
```

詳細ページ

一覧ページから記事タイトルをクリックした時に遷移する詳細ページを作成します。

データの詳細表示画面

src/views/articles/show.ecr

```
<h2>Article</h2>
<% articles.each do |article| %>
  <h3><%=article["title"] %></h3>
  <p><%=article["body"] %></p>
<br />
<% end %>
```

Kemal プログラム改修

kemal-sample.cr を以下の内容で修正します。

src/kemal-sample.cr

```
require "kemal"
require "db"
require "pg"

database_url = if ENV["KEMAL_ENV"]? && ENV["KEMAL_ENV"] == "production"
  ENV["DATABASE_URL"]
else
  "postgres://preface@localhost:5432/kemal_sample"
end

db = DB.open(database_url)

["/", "/articles"].each do |path|
  get path do |env|
    articles = [] of Hash(String, String | Int32)
    # クエリの実行
    db.query("select id, title, body from articles") do |rs|
      rs.each do
        article = {} of String => String | Int32
        article["id"] = rs.read(Int32)
        article["title"] = rs.read(String)
        article["body"] = rs.read(String)
        articles << article
      end
    end
    db.close
    render "src/views/index.ecr", "src/views/application.ecr"
  end
end

get "/articles/new" do |env|
  render "src/views/articles/new.ecr", "src/views/application.ecr"
end

post "/articles" do |env|
  # env.params.body で form の value を取得できます
  title_param = env.params.body["title"]
  body_param = env.params.body["body"]
  params = [] of String
```

```
  params << title_param
  params << body_param
  # update, insert, delete は以下のように exec でアップデートを実行します
  db.exec("insert into articles(title, body) values($1::text, $2::text)", params)
  db.close
  env.redirect "/"
end

get "/articles/:id" do |env|
  articles = [] of Hash(String, String | Int32)
  article = {} of String => String | Int32
  id = env.params.url["id"].to_i32
  params = [] of Int32
  params << id
  sql = "select id, title, body from articles where id = $1::int8"
  article["id"], article["title"], article["body"] =
    db.query_one(sql, params, as: {Int32, String, String})
  articles << article
  db.close
  render "src/views/articles/show.ecr", "src/views/application.ecr"
end

Kemal.run
```

まずクエリは以下のように記述します。

```
db.query("select id, title, body from articles") do |rs|
  rs.each do
    article = {} of String => String | Int32
    article["id"] = rs.read(Int32)
    article["title"] = rs.read(String)
    article["body"] = rs.read(String)
    articles << article
  end
end
```

query メソッドに SQL クエリを記述し、ループ内で結果を格納していきます。1 件だけ取得したい場合は query_one もしくは query_one?を使います。後者は 1 件もデータが無い場合がありうる場合に使います。

```
  sql = "select id, title, body from articles where id = $1::int8"
  article["id"], article["title"], article["body"] =
    db.query_one(sql, params, as: {Int32, String, String})
```

query_one の戻り値は、as で指定した型の Touple になります。上記の場合ですと、`Touble(Int32, String, String)` になります。

render "src/views/articles/show.ecr", "src/views/application.ecr" の形式で ecr ファイルをレンダリングすることで、rails のようにファイルを入れ子の形で描画することが出来ます。

Web アプリ再起動後に http://localhost:3000/にアクセスすると、まだ記事が投稿されていないので空欄になっています。http://localhost:3000/articles/new にアクセスすると、タイトルと本文を入力する画面が表示されており、入力後に一覧に遷移すると成功です。記事タイトルをクリックすると詳細画面に遷移すると成功です。

オブジェクトの CRUD

Kemal は RESTful に対応しており、get、post 以外にも、put、delete、patch にも対応しております。記事編集機能をサンプルに追加しながら説明します。

新規ページで src/views/articles/edit.ecr を追加します。また、記事詳細ページから edit.ecr へのリンクを追加します。

データの詳細表示画面

src/views/articles/show.ecr

```
<h2>Article</h2>
<% articles.each do |article| %>
  <h3><%=article["title"] %></h3>
  <p><%=article["body"] %></p>
<br />
<!-- 追加 -->
<a href="/articles/<%=article["id"] %>/edit" target="_top" class="btn btn-primary">edit</a>
<% end %>
```

データの編集画面

src/views/articles/edit.ecr

```
<h2>投稿編集</h2>
<% articles.each do |article| %>
<form method="post", action="/articles/<%=article["id"] %>">
  <!--
   hidden フィールドに name="_method"、
   value に put を設定する。
  -->
  <input type="hidden", name="_method", value="put" />
  <input type="text" name="title" size="10" maxlength="10" value="<%=article["title"] %>" />
  <br />
  <br />
  <textarea name="body" cols="40" rows="4"><%=article["body"] %></textarea>
  <br />
  <br />
  <input type="submit" value="edit" class="btn btn-primary">
</form>
<% end %>
```

ブラウザによっては form が get、post 以外には対応していない場合、以下の hidden フィールドの設定で put や delete で送信することが出来ます。

kemal-sample.cr を以下の内容で修正します。

src/kemal-sample.cr

```
(中略)
get "/articles/:id/edit" do |env|
  articles = [] of Hash(String, String | Int32)
  article = {} of String => String | Int32
  id = env.params.url["id"].to_i32
  params = [] of Int32
  params << id
  sql = "select id, title, body from articles where id = $1::int8"
  article["id"], article["title"], article["body"] =
    db.query_one(sql, params, as: {Int32, String, String})
  articles << article
  db.close
  render "src/views/articles/edit.ecr", "src/views/application.ecr"
end

put "/articles/:id" do |env|
  id = env.params.url["id"].to_i32
  title_param = env.params.body["title"]
  body_param = env.params.body["body"]
  params = [] of String | Int32
  params << title_param
  params << body_param
  params << id
  db.exec("update articles set title = $1::text, body = $2::text where id = $3::int8", params)
  db.close
  env.redirect "/articles/#{id}"
end

Kemal.run
```

これで、データの追加から一覧表示、データの編集までの一連の流れが行えるようになります。また、delete を追加する場合は以下のように行います。

```
delete "/articles/:id" do |env|
  id = env.params.url["id"].to_i32
  params = [] of Int32
  params << id
  db.exec("delete from articles where id = $1::int8", params)
  db.close
  env.redirect "/"
end
```

update や delete の場合は exec メソッドを使用します。

```
db.exec("update articles set title = $1::text, body = $2::text where id = $3::int8", params)
```

1.3 セッションについて

Kemal ではセッションの機能を標準でサポートします。下記ファイルを編集し、shards update を実行します。

shard.yml

```
dependencies:
  kemal-session:
    github: kemalcr/kemal-session
```

本サンプルでは、簡単な認証機能を追加してみます。新規投稿を行う場合は認証済みのユーザでなければ出来ない（新規投稿画面に遷移できない）ようにします。

ソースを以下の内容で修正します。kemal-sample.cr は以下の通りです。

src/kemal-sample.cr

```
require "kemal"
require "kemal-session"
(中略)

Kemal::Session.config do |config|
  config.cookie_name = "session_id"
  config.secret = "some_secret"
  config.gc_interval = 2.minutes # 2 minutes
end

def authorized?(env)
  env.session.string?("username")
end

get "/login" do |env|
  render "src/views/login.ecr", "src/views/application.ecr"
end

post "/login" do |env|
  user_id_param = env.params.body["user_id"]
  password_param = env.params.body["password"]
  if user_id_param == "user1" && password_param == "pass1"
    env.session.string("username", "user1")
    env.redirect "/"
  else
    env.redirect "/login"
```

```
  end
end

get "/logout" do |env|
  env.session.destroy
  env.redirect "/"
end
```

ログイン画面を新規で追加します。

src/views/login.ecr

```
<h2>ログイン</h2>
<form method="post", action="/login">
  <input type="text" name="user_id" size="10" maxlength="10" />
  <br />
  <br />
  <input type="password" name="password" size="10" maxlength="10" />
  <br />
  <br />
  <input type="submit" value="post">
</form>
```

ヘッダの内容を変更します。

src/views/application.ecr

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>kemal sample</title>
  <link
    rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    integrity="sha384-1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
    crossorigin="anonymous">
  <link rel="stylesheet" href="/css/custom.css">
</head>
<body>
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <a id="logo">sample app</a>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="/articles">ArticleList</a></li>
        <% if env.session.string?("username") %>
          <li><a href="/articles/new">新規投稿</a></li>
          <li><a href="/logout">ログアウト</a></li>
        <% else %>
```

```
          <li><a href="/login">ログイン</a></li>
        <% end %>
      </ul>
    </nav>
  </div>
</header>
  <div class="container">
    <%= content %>
  </div>
</body>
</html>
```

主な設定箇所について解説します。まずセッションの設定を行います。

```
Kemal::Session.config do |config|
  config.cookie_name = "session_id"
  config.secret = "some_secret"
  config.gc_interval = 2.minutes # 2 minutes
end
```

本サンプルでは cookie にセッションを保存します。`cookie_name` と `secret` で `cookie` の設定を行います。`gc_interval` で有効期間を設定します。デフォルトでは 4 分です。その他の設定は [kemal-session](https://github.com/kemalcr/kemal-session) を参照してみてください。

その他追記箇所についてはログイン、ログアウトのパスを新規で作っています。ユーザ ID とパスワードの組み合わせが正しい場合はセッションを作り、そうでない場合はリダイレクトします。本運用を考える場合は、設定を DB に持たせるなどします。

画面の方を以下の内容に修正します。

src/views/application.ecr

```
  <li><a href="/articles">ArticleList</a></li>
  <% if env.session.string?("username") %>
    <li><a href="/articles/new">新規投稿</a></li>
    <li><a href="/logout">ログアウト</a></li>
  <% else %>
    <li><a href="/login">ログイン</a></li>
  <% end %>
```

セッションの有無で表示を切り分けます。本稿では扱いませんが、その他ライブラリを別途使用することで Redis にセッションを保持することも可能になります。

1.4 Heroku へのデプロイ

Crystal で作成したアプリを Heroku にデプロイすることが出来ます。いくつかビルドバックが公開されていますが、以下の点が満たせなかったので自作しました。

- デプロイに使用する Crystal のバージョンは任意で指定したい
- ビルドするファイル名は適宜指定したい
- ビルド時の環境変数を適宜指定したい
- ビルドオプションは適宜指定したい

URL は以下のとおりです。

- heroku-buildpack-crystal
- https://github.com/msky026/heroku-buildpack-crystal

実装にあたっては Elixir のビルドパックを参考にさせていただきました。

- heroku-buildpack-elixir
- https://github.com/HashNuke/heroku-buildpack-elixir

ビルドパックの使い方

Heroku アプリ作成時に以下のコマンドを実行します。

```
heroku create --buildpack \
  "https://github.com/msky026/heroku-buildpack-crystal.git"
```

本ビルドパックでは Crystal のバージョンやビルドコマンドを適宜指定するため、コンフィグファイルをリポジトリに加える必要があります。コンフィグファイルはファイル名を ‘crystal_buildpack.config‘として設定します。以下の設定をファイルに記述します。

設定例

```
# Crystal version
# 使用する Crystal のバージョン（必須）
crystal_version=0.23.1

# Always rebuild from scratch on every deploy?
# ダウンロードした Crystal 本体など
# をクリアし再ダウンロードするか
# Crystal のバージョンを変えて
# リビルドするときは true に設定する
always_rebuild=false
```

```
# Export heroku config vars
# ビルド時に渡される Heroku の環境変数
config_vars_to_export=(DATABASE_URL)

# A command to run right after compiling the app
# コンパイル後に実行するコマンド
# 必要なければ未設定で可
post_compile="pwd"

# Build command
# デプロイ時にビルドするコマンドを設定（必須）
# 複数設定可能でその場合は空白区切りで記述
build_command=("make db_migrate" "make build")
```

尚、ファイルは作成しなくても動かせますが、その場合以下の設定がデフォルトで適用されます。

```
# Use latest version
crystal_version=latest

# ダウンロードした
# Crystal 本体は再ダウンロードしない
always_rebuild=false

# ビルド時に渡される環境変数は"DATABASE_URL"
config_vars_to_export=(DATABASE_URL)

# ビルドコマンドは"make run"
build_command=("make run")
```

アプリのデプロイ

では実際にデプロイするまでを行ってみます。まず前項でも記載しましたが、以下のコマンドで Heroku にアプリを作成します。（アプリ名は任意で、重複しなければなんでも構いません）

```
heroku create kemal-sample-160601 --buildpack "https://github.com/msky026/heroku-buildpack-crystal.git"
```

データベースを設定します。PostgreSQL を追加します。（Hobby パックが指定されます）

```
heroku addons:add heroku-postgresql
```

Heroku の PostgerSQL にテーブルを作成します。

```
heroku pg:psql
kemal-sample-160601::DATABASE=> \i /path/to/file/create_articles.sql
```

config ファイルと Procfile を作成し、Heroku に push します。

crystal_buildpack.config

```
crystal_version=0.23.1
always_rebuild=false
config_vars_to_export=(DATABASE_URL)
post_compile="pwd"
build_command=("crystal build --release ./src/kemal-sample.cr")
```

Procfile

```
web: ./kemal-sample -p $PORT -b 0.0.0.0
```

設定後は以下のコマンドでデプロイします。

```
git add .
git commit -m "add"
git push heroku master
```

デプロイ後に `https://kemal-sample-160601.herokuapp.com/`にアクセスすると、エラーが表示されなく、かつ `/articles/new` からデータが追加できれば成功です。

Crystal と Web

2017 年 12 月 29 日　初版第 1 刷　発行

著　者　Crystal-JP、msky