Server-side web framework

Created by 이 윤 준 (yoonjoon.lee@gmail.com)

May, 2019

- Explore how web frameworks can simplify the tasks, such as the communication between web clients and servers and what a server-side web application needs to do in order to respond to requests from a web browser.
- And give us an idea of how you'd choose a framework for our first server-side web application.

Overview

Server-side web frameworks (a.k.a. "web application frameworks") are software frameworks that make it easier to write, maintain and scale web applications.

They provide tools and libraries that simplify common web development tasks, including routing URLs to appropriate handlers, interacting with databases, supporting sessions and user authorization, formatting output (e.g. HTML, JSON, XML), and improving security against web attacks.

What can a web framework do for you?Section

Web frameworks provide tools and libraries to simplify common web development operations. You don't *have* to use a server-side web framework, but it is strongly advised — it will make your life a lot easier.

Work directly with HTTP requests and responses

Web frameworks allow the developers to write simplified syntax that will generate server-side code to work with these requests and responses. This means that they will have an easier job, interacting with easier, higher-level code rather than lower level networking primitives.

Every "view" function (a request handler) receives an `HttpRequest` object containing request information, and is required to return an `HttpResponse` object with the formatted output (in this case a string).

```python
# Django view function
from django.http import HttpResponse

def index(request):
    # Get an HttpRequest (request)
    # perform operations using information from the reques
    # Return HttpResponse
    return HttpResponse('Output string to return')
```

Route requests to the appropriate handler

Most sites will provide a number of different resources, accessible through distinct URLs. Handling these all in one function would be hard to maintain, so web frameworks provide simple mechanisms to map URL patterns to specific handler functions.

The Flask (Python) web framework adds routes to view functions using a decorator.

```
@app.route("/")
def hello():
    return "Hello World!"
```

Django expects developers to define a list of URL mappings between a URL pattern and a view function.

```
urlpatterns = [
    url(r'^$', views.index),
    # example: /best/myteamname/5/
    url(r'^best/(?P<team_name>\w.+?)/(?P<team_number>[0-9]
]
```

Make it easy to access data in the request

Data can be encoded in an HTTP request in a number of ways.

An HTTP `GET` request to get files or data from the server may encode what data is required in URL parameters or within the URL structure.

An HTTP `POST` request to update a resource on the server will instead include the update information as "POST data" within the body of the request.

The HTTP request may also include information about the current session or user in a client-side cookie.

The `HttpRequest` object that Django passes to every view function contains

- methods and properties for accessing the target URL,
- the type of request (e.g. an HTTP GET), GET or POST parameters,
- cookie and session data, etc.

Django can also pass information encoded in the structure of the URL by defining "capture patterns" in the URL mapper (see the last code fragment in the section above).

Abstract and simplify database access

Websites use databases to store information both to be shared with users, and about users. Web frameworks often provide a database layer that abstracts database read, write, query, and delete operations. This abstraction layer is referred to as an Object-Relational Mapper (ORM).

Two benefits:

- Can replace the underlying database without necessarily needing to change the code.
- Basic validation of data can be implemented within the framework.

The Django web framework provides an ORM, and refers to the object used to define the structure of a record as the model.

The model specifies the field types to be stored, which may provide field-level validation on what information can be stored (e.g. an email field would only allow valid email addresses). The field definitions may also specify their maximum size, default values, selection list options, help text for documentation, label text for forms etc.

The model doesn't state any information about the underlying database as that is a configuration setting that may be changed separately of our code.

The code snippet shows a very simple Django model for a `Team` object.

```python
#best/models.py

from django.db import models

class Team(models.Model):
    team_name = models.CharField(max_length=40)

    TEAM_LEVELS = (
        ('U09', 'Under 09s'),
        ('U10', 'Under 10s'),
        ('U11', 'Under 11s'),
        ...  #list our other teams
    )
    team_level = models.CharField(max_length=3,
                   choices=TEAM_LEVELS, default='U11')
```

This stores the team name and team level as character fields and specifies a maximum number of characters to be stored for each record.

The `team_level` is a choice field, so we also provide a mapping between choices to be displayed and data to be stored, along with a default value.

The Django model provides a simple query API for searching the database.

This can match against a number of fields at a time using different criteria, and can support complex statements.

```python
#best/views.py

from django.shortcuts import render
from .models import Team

def youngest(request):
    #  for displaying all of our U09 teams
    list_teams = Team.objects.filter(team_level__exact="U0
    context = {'youngest_teams': list_teams}
    return render(request, 'best/index.html', context)
```

Rendering data

Web frameworks often provide templating systems to specify the structure of an output document, using placeholders for data that will be added when a page is generated.

Web frameworks often provide a mechanism to make it easy to generate other formats from stored data, including JSON and XML.

The Django template system allows you to specify variables using a "double-handlebars" syntax (e.g. `{{ variable_name }}`), which will be replaced by values passed in from the view function when a page is rendered.

```
#best/templates/best/index.html

<!DOCTYPE html>
<html lang="en">
<body>

 {% if youngest_teams %}
    <ul>
    {% for team in youngest_teams %}
        <li>{{ team.team_name }}</li>
    {% endfor %}
    </ul>
{% else %}
    <p>No teams are available.</p>
{% endif %}

</body>
</html>
```

How to select a web framework

Numerous web frameworks exist for almost every programming language. Some of the factors that may affect your decision are:

- Effort to learn
- Productivity
    - Framework purpose
    - Opinionated vs. unopinionated
    - Bateeries included vs. get it yourself
    - Whether or not the framework encourages good development practices

- Performance of the framework/programming language
- Caching suppor
- Scalability
- Web security

A few good web frameworks?

The server-side frameworks below represent a few of the most popular available at the time of writing. All of them have everything you need to be productive — they are open source, are under active development, have enthusiastic communities creating documentation and helping users on discussion boards, and are used in large numbers of high-profile websites.

Django (Python)

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Django follows the "Batteries included" philosophy and provides almost everything most developers might want to do "out of the box".

It all works together, follows consistent design principles, and has extensive and up-to-date documentation. It is also fast, secure, and very scalable. Being based on Python, Django code is easy to read and to maintain.

Popular sites using Django include: Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest, Open Stack.

Flask (Python)

Flask is a microframework for Python.

While minimalist, Flask can create serious websites out of the box. It contains a development server and debugger, and includes support for Jinja2 templating, secure cookies, unit testing, and RESTful request dispatching. It has good documentation and an active community.

Particularly for developers who need to provide web services on small, resource-constrained systems (e.g. running a web server on a Raspberry Pi, Drone controllers, etc.)

Express (Node.js/Javascript)

Express is a fast, unopinionated, flexible and minimalist web framework for Node.js (node is a browserless environment for running JavaScript). It provides a robust set of features for web and mobile applications and delivers useful HTTP utility methods and middleware.

Express is extremely popular, partially because it eases the migration of client-side JavaScript web programmers into server-side development, and partially because it is resource-efficient.

There are many excellent independent components, but sometimes it can be hard to work out which is the best for a particular purpose!

Many popular server-side and full stack frameworks (comprising both server and client-side frameworks) are based on Express, including Feathers, ItemsAPI, KeystoneJS, Kraken, LoopBack, MEAN, and Sails.

Ruby on Rails (Ruby)

Rails (usually referred to as "Ruby on Rails") is a web framework written for the Ruby programming language.

Rails follows a very similar design philosophy to Django. Like Django it provides standard mechanisms for routing URLs, accessing data from a database, generating HTML from templates and formatting data as JSON or XML. It similarly encourages the use of design patterns like DRY ("dont repeat yourself" — write code only once if at all possible), MVC (model-view-controller) and a number of others.

Rails has been used for high profile sites, including: Basecamp, GitHub, Shopify, Airbnb, Twitch, SoundCloud, Hulu, Zendesk, Square, Highrise.

Laravel (PHP)

Laravel is a web application framework with expressive, elegant syntax. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as:

- Simple, fast routing engine.
- Powerful dependency injection container.
- Multiple back-ends for session and cache storage.
- Expressive, intuitive database ORM.
- Database agnostic schema migrations.
- Robust background job processing.
- Real-time event broadcasting.

ASP.NET

ASP.NET is an open source web framework developed by Microsoft for building modern web applications and services. With ASP.NET we can quickly create web sites can be based on HTML, CSS, and JavaScript, scale them for use by millions of users and easily add more complex capabilities like Web APIs, forms over data, or real time communications.

One of the differentiators for ASP.NET is that it is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language.

Mojolicious (Perl)

Mojolicious is a next generation web framework for the Perl programming language.

Some of the features provided by Mojolicious are:

- Real-time web framework, to easily grow single file prototypes into well-structured MVC web applications;
- RESTful routes, plugins, commands, Perl-ish templates, content negotiation, session management, form validation, testing framework, static file server, CGI/PSGI detection, first class Unicode support;

- Full stack HTTP and WebSocket client/server implementation with IPv6, TLS, SNI, IDNA, HTTP/SOCKS5 proxy, UNIX domain socket, Comet (long polling), keep-alive, connection pooling, timeout, cookie, multipart and gzip compression support;
- JSON and HTML/XML parsers and generators with CSS selector support;
- Very clean, portable and object-oriented pure-Perl API with no hidden magic; Fresh code based upon years of experience, free and open source.

Spring Boot (Java)

Spring Boot is one of a number of projects provided by Spring. It is a good starting point for doing server-side web development using Java.

Although definitely not the only framework based on Java it is easy to use to create stand-alone, production-grade Spring-based Applications that we can "just run".

It is an opinionated view of the Spring platform and third-party libraries but allows to start with minimum fuss and configuration.

It can be used for small problems but its strength is building larger scale applications that use a cloud approach.

Usually multiple applications run in parallel talking to each other, with some providing user interaction and others just do back end work.

Load balancers help to ensure redundancy and reliability or allow geolocated handling of user requests to ensure responsiveness.