Framework

Un ***framework***, **entorno de trabajo**[1] o **marco de trabajo**[2] es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar

En el desarrollo de software, un entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de *software*, que puede servir de base para la organización y desarrollo de *software*. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de *software* que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.[3]

Índice

Introducción

Los marcos de trabajo tienen como objetivo principal ofrecer una funcionalidad definida, auto contenida, siendo construidos usando patrones de diseño, y su característica principal es su alta cohesión y bajo acoplamiento.

Para acceder a esa funcionalidad, se construyen piezas, objetos, llamados objetos calientes, que vinculan las necesidades del sistema con la funcionalidad que este presta.

Esta funcionalidad, está constituida por objetos llamados fríos, que sufren poco o ningún cambio en la vida del framework, permitiendo la portabilidad entre distintos sistemas.

Algunos entornos de trabajo conocidos son Spring Framework o Hibernate, donde lo esencial para ser denominados entornos de trabajo es estar constituidos por objetos casi estáticos con funcionalidad definida a nivel grupo de objetos y no como parte constitutiva de estos, por ejemplo en sus métodos, en cuyo caso se habla de una API o librería.

Algunas características notables que se pueden observar:

- La inversión de control: en un *framework*, a diferencia de las bibliotecas, el flujo de control no es dictado por el programa que llama, sino por el mismo.[3]
- La funcionalidad o comportamiento predeterminado: un marco tiene un comportamiento predeterminado. Este comportamiento por defecto debe ser un comportamiento útil, definido e identificable.

- Su escalabilidad: un marco puede ser ampliado para proporcionar una funcionalidad específica. El frame, en general, no se supone que deba ser modificado, excepto en cuanto a extensibilidad. Los usuarios pueden ampliar sus características, pero no deben ni necesitan modificar su código.

Básicos

No es más que una base de programación que atiende a sus descendientes (manejado de una forma estructural o en cascada), posibilitando cualquier respuesta ante las necesidades de sus miembros, o en secciones de una aplicación (web), satisfaciendo así las necesidades más comunes del programador

Arquitectura

Dentro de este aspecto, podemos basarnos en el modelo-vista-controlador o MVC (Controlador => Modelo => Vista), ya que debemos fragmentar nuestra programación. Tenemos que contemplar estos aspectos básicos en cuanto a la implementación de nuestro sistema:

Modelo

Este miembro del controlador maneja las operaciones lógicas, y de manejo de información (previamente enviada por su ancestro), para resultar de una forma explicable y sin titubeos. Cada miembro debe ser meticulosamente llamado, con su correcto nombre y en principio, con su verdadera naturaleza: el manejo de información, su complementación directa.

Vista

Al final, a este miembro de la familia le corresponde dibujar, o expresar la última forma de los datos: la interfaz gráfica que interactúa con el usuario final del programa (GUI). Después de todo, a este miembro le toca evidenciar la información obtenida hasta hacerla llegar al controlador. Solo (e inicialmente), nos espera demostrar la información.

Controlador

Con este apartado podemos controlar el acceso (incluso todo) a nuestra aplicación, y esto puede incluir: archivos, *scripts*, o programas; cualquier tipo de información que permita la interfaz. Así, podremos diversificar nuestro contenido de forma dinámica, y estática (a la vez); pues, solo debemos controlar ciertos aspectos (como se ha mencionado antes).

Estructura

Dentro del controlador, modelo o vista, se pueden manejar datos, y depende de cada uno cómo interpretar y manejar esos datos. Se sabe que el único dato de una dirección estática web es: conseguir un archivo físico en el disco duro o de Internet, etc., e interpretado o no, el servidor responde.

El modelo, al igual que el controlador y la vista, maneja todos los datos que se relacionen consigo (solo es el proceso medio de la separación por capas que ofrece la arquitectura MVC). Y solo la vista, puede demostrar dicha información. Con lo cual ya se ha generado la jerarquía del programa: controlador, modelo y vista.

Lógica

Al parecer, debemos inyectar ciertos objetos dentro de sus parientes en esta aplicación, solo así compartirán herencia y coherencia en su aplicación.

Rápidamente, para una aplicación web sencilla debemos establecer estos objetos:

- Una base (MVC)
  - Controlador: este debe ser capaz de manejar rutas, archivos, clases, métodos y funciones.
  - Modelo: es como un *script* habitual en el servidor, solo que agrupado bajo un *modelo* reutilizable.
  - Vista: como incluyendo cualquier archivo en nuestra ejecución, muy simple.

- Un sistema
  - Ruteador: con él podemos dividir nuestras peticiones sin tantas condicionales.
  - Cargador

Ejemplos

```
    // Index.php
// -----

// ------ Clases ------
class Base {}
class Controller extends Base {
 function load($name) {
     require_once $this->$name &= new $name();
 }
}
class Model extends Controller {
 function view($name, $data) {
    extract($data);

    include "app/view/" . $name . ".php";
 }
}

// ------ Router & Loader ------
function _route($controller, $model) {
 if (is_file("app/$controller.php")) {
     require_once "app/" . $controller . ".php";

     $object = new $controller();

     $object->$model();
 }
}

// ----- Rutina -----
_route($_GET['section'], $_GET['name']);
```

Esto cumple con algunas necesidades de simpleza informática. Ahora solo nos basta controlar estos procesos, ampliarlos y complementarles con algunos*scripts* más.

Aplicar

Si nuestro archivo se llama Foo (clase), y nuestro otro archivo, bar (método) tenemos que crear el siguiente archivo dentro de la carpeta **app/**.

```
// app/Foo.php
    // -----

    class Foo extends Controller {
        function __construct() {
            $this->load('Test');
        }

        function bar() {
         echo '<b>Testing</b>';
         echo $this->test->does();
     }
    }
```

Como resultado al solicitar (por ejemplo,*?section=foo&name=bar*), deberíamos ver el siguiente texto:**Testing**

Extender

Podremos extender nuestro sistema con clases, o funciones propias o de algún 'plugin' o Biblioteca ajena. Solo que queremos extenderles sobre nuestro sistema actual, nuestro objeto básico.

```
// app/model/Test.php
    // -----

    class Test extends Model {
        function does() {
            echo '<ins>Hecho esta!</ins>' ;
            echo $this->view('look', array('my_var' => 'my_value'));
     }
    }
```

Entonces, debemos usar la siguiente sentencia dentro de nuestro programa **Foo**:

$this->load($this, 'test') o *_load($this, 'test')*

Ya con esto, podremos utilizar las llamadas a **$this->test->does()** dentro del objeto o clase *Foo*.

Ver

Para mostrar los resultados de todo nuestro computo necesitamos de vistas, o archivos de inclusión: plantillas, bloques o *scripts*. Suponiendo que ya ha sido todo, debemos de visualizarlo:

```
 // app/view/Look.php
 // -----

 echo 'Variable: ' . $my_var;
```

Para poder ejecutar esto, se debe llamar a esta sentencia: `$this->view('look', array ('my_var' => 'my_value'))` obteniendo como resultado:

```
Variable: my_value
```

Véase también

- Dojo toolkit
- Entorno de desarrollo integrado
- Framework para aplicaciones web
- Marco de aplicaciones
- Modelo-vista-controlador
- Plataforma
- Hamlets

Referencias

1. Oscar, La Red Martínez, David L.; Acosta, Julio César; Mata, Liliana E.; Bachmann, Noemí G.; Vallejos, (1 de enero de 2012). *Aprendizaje combinado, aprendizaje electrónico centrado en el alumno y nuevas tecnologías* (http://sedici.unlp.edu.ar/handle/10915/19306) Consultado el 11 de abril de 2017
2. «Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE» (http://openaccess.uoc.edu/webapps/o2/bitstream/10609/876/1/00765tfc.pdf) *Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE*
3. Riehle, Dirk (2000), *Framework Design: A Role Modeling Approach* (http://www.riehle.org/computer-science/research/dissertation/diss-a4.pdf) *Swiss Federal Institute of Technology*