ZF - Session 3

1. Module.php

Antes de llamar a ningún action para atender la petición actual, primero se ejecutan los métodos de la clase Module. Dos de ellos son de configuración como vimos en apartados anteriores. Ahora nos centraremos en otros dos métodos que podemos utilizar para ejecutar código antes de que la petición sea atendida por un action.

1.1. init

Este método se utiliza para inicializar el módulo. En este punto puede suceder que no todos los módulos hayan sido cargados.

Recibe como argumento el Module Manager y a través de este componente tenemos acceso al EventManager y podemos gestionar los módulos a través de sus métodos.

En ZF2 para permitir la Inversión de Control (el framework nos cede el control) se generan varios eventos a lo largo del ciclo de una petición. De manera que podemos inyectar código en cada una de las diferentes etapas por las que pasa una petición.

MvcEvent Events		
Name	**Constant**	**Description**
bootstrap	MvcEvent::EVENT_BOOTSTRAP	Bootstrap the application by creating the ViewManager.
route	MvcEvent::EVENT_ROUTE	Perform all the route work (matching...).
dispatch	MvcEvent::EVENT_DISPATCH	Dispatch the matched route to a controller/action.
dispatch.error	MvcEvent::EVENT_DISPATCH_ERROR	Event triggered in case of a problem during dispatch process (unknown controller...).
render	MvcEvent::EVENT_RENDER	Prepare the data and delegate the rendering to the view layer.
render.error	MvcEvent::EVENT_RENDER_ERROR	Event triggered in case of a problem during the render process (no renderer found...).
finish	MvcEvent::EVENT_FINISH	Perform any task once everything is done.

En la tabla anterior tenemos los diferentes eventos que ZF2 lanza durante el ciclo de una petición.

Estos eventos están definidos en la clase MvcEvent:

```
class MvcEvent extends Event
{
    /**#@+
     * Mvc events triggered by eventmanager
     */
    const EVENT_BOOTSTRAP      = 'bootstrap';
    const EVENT_DISPATCH       = 'dispatch';
    const EVENT_DISPATCH_ERROR = 'dispatch.error';
    const EVENT_FINISH         = 'finish';
    const EVENT_RENDER         = 'render';
    const EVENT_RENDER_ERROR   = 'render.error';
    const EVENT_ROUTE          = 'route';
```

Como vemos el primero de ellos es el evento Bootstrap y es manejado por la clase module en su método onBootstrap (lo veremos en el siguiente punto). A partir de aquí podemos añadir manejadores al resto de eventos.

Como ejemplo crearemos un manejador para el evento MvcEvent::Event_Dispatch:

```
class Module implements AutoloaderProviderInterface
{
    public function init(ModuleManager $moduleManager)
    {
        $em = $moduleManager->getEventManager();
        $em->attach(MvcEvent::EVENT_DISPATCH, array($this, 'onDispatch'));
    }

    public function onDispatch(MvcEvent $e)
    {
        echo 'onDispatch Event';
    }
```

El método 'attach' del Event Manager tiene dos argumentos: el nombre del evento y qué método lo va a manejar.

1.2. onBootstrap

Cuando este método es llamado, todos los módulos han sido cargados.

Este método será llamado en cada petición antes de que ésta sea atendida por ningún 'action'. Por lo tanto, si necesitamos que una acción se lleve a cabo independientemente del 'controller' y del 'action' que vaya a responder, este es el sitio adecuado.

Un ejemplo podría ser la definición de una variable de Layout.

El hecho de que este método sea llamado en cada petición hace que debamos ser también muy cuidadosos con las acciones que llevaremos a cabo en este punto. No es buena idea cargar mucho este método, sólo las acciones indispensables.

La interfaz del método es la siguiente:

```
public function onBootstrap(MvcEvent $e)
```

Mediante el objeto $e tenemos acceso a la petición actual. Podemos obtener el controller que será invocado y su clase:

```
$controller = $e->getController();
```

```
$controllerClass = $e->getControllerClass();
```

Así podríamos decidir qué acción realizar dependiendo del controller.

Podemos también recuperar el Layout:

```
$layout = $e->getViewModel();
```

Y asignar variables de Layout:

```
$layout->layoutVar = 'Value From Layout';
```

También podemos utilizar aquí el ServiceManager para recuperar cualquier servicio del contenedor de servicios. Por ejemplo, vamos a utilizar el servicio 'config' para recuperar el nombre de la aplicación y asignarlo a una variable de layout:

```
public function onBootstrap(MvcEvent $e)
{
    $sm = $e->getApplication()->getServiceManager();
    $config = $sm->get('config');
    $layout = $e->getViewModel();
    $layout->title = $config['application']['name'];
}
```

También podeos utilizar el método onBootstrap para asignar manejadores de eventos a través del Event Manager:

```
public function onBootstrap(MvcEvent $e)
{
    $em = $e->getApplication()->getEventManager();
    $em->attach(MvcEvent::EVENT_DISPATCH, array($this, 'onDispatch'));
}

public function onDispatch(MvcEvent $e)
{
```

```
    echo 'onDispatch Event';
}
```

2. Controller Plugins

Dentro de un controller que herede de AbstractActionController o de AbstractRestfulController o si implementamos el método setPluginManager tendremos acceso a los plugins que ZF2 trae por defecto.

Un plugin es un objeto que tiene métodos que nos van a ser útiles en diferentes controllers.
Éstos son algunos de los plugins que ZF2 nos facilita:

- FlashMessenger() : Mensajes basados en la sesión
- Forward() : Redirige la petición actual a otro controller. No crea una nueva petición.
- Layout() : Nos permite cambiar el template del layout.
- Params() : Recoge parámetros desde diferentes fuentes (fromRoute, fromPost, ...)
- Redirect() : Redirige a una nueva url. Crea una nueva petición.
- Url() : Generación de urls en un controller

Url

Este helper nos sirve para generar urls en un controller a partir de las rutas definidas en el router. Se usa de manera algo diferente a como se hace en las vistas:

```
$view->action =  $this->url()->fromRoute('user\index\updateDo');
```

Podemos encontrar más plugins y ejemplos de uso en la documentación oficial:

http://framework.zend.com/manual/current/en/modules/zend.mvc.plugins.html

2.1. Controller Plugins propios

Vamos a crear un plugin que nos sirva para añadir un determinado número de 0's a la izquierda de un número. Para ilustrar cómo acceder a un servicio del ServiceManager desde el plugin, recogeremos el número de 0's a añadir de un parámetro de configuración del módulo Calculator.

Como nuestro plugin va a requerir el número de 0's en su creación, lo vamos a requerir en el constructor:

```
<?php
/**
 * xenFramework (http://xenframework.com/)
 *
```

```
 * This file is part of the xenframework package.
 *
 * (c) Ismael Trascastro <itrascastro@xenframework.com>
 *
 * @link        http://github.com/xenframework for the canonical source repository
 * @copyright   Copyright (c) xenFramework. (http://xenframework.com)
 * @license     MIT License - http://en.wikipedia.org/wiki/MIT_License
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Calculator\Controller\Plugin;


use Zend\Mvc\Controller\Plugin\AbstractPlugin;

class PrependZerosPlugin extends AbstractPlugin
{
    /**
     * @var int The number of 0's to prepend
     */
    private $digits;

    public function __construct($digits)
    {
        $this->digits = $digits;
    }

    /**
     * __invoke
     *
     * @param int $number The number to prepend zeros in
     *
     * @return string
     */
    public function __invoke($number)
    {
        return sprintf('%0' . $this->digits . 'd', $number);
    }
}
```

Hemos creado el método __invoke ya que nuestro plugin solo tiene un método.

Ahora creamos el Factory para inyectar el número de 0's en el constructor:

```
<?php
/**
 * xenFramework (http://xenframework.com/)
 *
```

```
 * This file is part of the xenframework package.
 *
 * (c) Ismael Trascastro <itrascastro@xenframework.com>
 *
 * @link        http://github.com/xenframework for the canonical source repository
 * @copyright   Copyright (c) xenFramework. (http://xenframework.com)
 * @license     MIT License - http://en.wikipedia.org/wiki/MIT_License
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Calculator\Controller\Plugin\Factory;


use Calculator\Controller\Plugin\PrependZerosPlugin;
use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class PrependZerosPluginFactory implements FactoryInterface
{

    /**
     * Create service
     *
     * @param ServiceLocatorInterface $serviceLocator
     *
     * @return mixed
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $sm = $serviceLocator->getServiceLocator();
        $config = $sm->get('config');
        $digits = $config['Calculator']['digits'];

        return new PrependZerosPlugin($digits);
    }
}
```

Damos de alta el factory en el module.config.php:

```
'controller_plugins' => array(
    'factories' => array(
        'PrependZeros' =>
'Calculator\Controller\Plugin\Factory\PrependZerosPluginFactory',
    ),
),
```

El nombre que hemos puesto en el identificator 'PrependZeros' es el que utilizaremos en el controller para utilizar el plugin.

Creamos la variable de configuración:

```
'Calculator' => array(
    'digits' => 3,
),
```

Lo usamos en el controller:

```
public function addDoAction()
{
    $this->model->setOp1($this->params()->fromPost('op1'));
    $this->model->setOp2($this->params()->fromPost('op2'));
    $this->model->sum();

    $result = $this->PrependZeros($this->model->getResult());

    $title = $this->layout()->getVariable('title');
    $newTitle = $title . ' - Calculator Add Result';
    $this->layout()->setVariable('title', $newTitle);

    return ['result' => $result, 'title' => $newTitle];
}
```

2.2. Cómo llamar a un action de otro controller

Si desde un controller necesitamos llamar a un action o método de otro controller, tenemos dos opciones dependiendo de si el action llamado renderiza devuelve un ViewModel o no. El otro controller puede estar en un módulo diferente.

2.2.1. Forward

Podemos llamar al action de otro controller haciendo uso del plugin forward():

```
$response = $this->forward()->dispatch('TestModule/Controller/Controller2', ['action' => 'test', 'var' => '4']);
```

Podemos recuperar los parámetros en el action destino sin necesidad de tener una ruta creada en el module.config.php:

```
public function testAction()
{
    return $this->params()->fromRoute('var');
}
```

Si queremos recuperar el control y renderizar nosotros desde el primer controller:

```
$phpRendrer = $this->getEvent()
   ->getApplication()
   ->getServiceManager()
   ->get('Zend\View\Renderer\PhpRenderer');

$content = $phpRendrer->render($temp);
```

2.2.2. Controller Manager

Podemos recuperar el Controller Manager mediante el ServiceLocator:

```
class IndexController extends AbstractActionController
{
   public function indexAction()
   {
      $cm = $this->serviceLocator->get('controllerManager');
      $controller2 = $cm->get('OtherModule\Controller\Controller2');
      $msg = $controller2->testAction();

      return ['msg' => $msg];
   }
}
```

O también podríamos crear un factory para el controller poniendo como dependencia al segundo controller:

```
class IndexControllerFactory implements FactoryInterface
{

   /**
    * Create service
    *
    * @param ServiceLocatorInterface $serviceLocator
    * @return mixed
    */
   public function createService(ServiceLocatorInterface $serviceLocator)
   {
      $controller2 = $serviceLocator->get('TestModule\Controller\Controller2');

      return new IndexController($controller2);
   }
}
```

3. View Helpers

Volvamos a nuestro módulo Calculator. En la vista correspondiente al action 'add' mostraremos un formulario. Dicho formulario será idéntico al que mostraremos en las vistas para restar, multiplicar y dividir.

3.1. Partial View Helper

Para evitar repetir código, pondremos el formulario en una vista accesible por todas, en un partial.

```
<div>
    <?php echo $this->partial('partial/menu', array('title' => $title)); ?>
</div>

<div>
    <?php echo $this->partial('partial/form', array('action' => $this-
>url('calculator_addDo'))); ?>
</div>
```

Partial es un View Helper que nos renderiza partes de una vista que están en otros ficheros. Lo único que tenemos que hacer es dar de alta el alias del partial en el fichero module.config.php:

```
'template_map' => array(
    'partial/form'          => __DIR__ . '/../view/calculator/partial/form.phtml',
    'partial/menu'          =>    DIR    . '/../view/calculator/partial/menu.phtml',
),
```

En este caso estamos usando dos partials. El segundo es para mostrar un menú.

3.2. Url View Helper

También estamos haciendo uso del View Helper 'url'. Este View Helper nos genera una ruta a partir de un identificador de ruta. Este identificador debe existir en el apartado routes del fichero module.config.php.

Todos éstos helpers pueden ser utilizados de la misma manera en los layouts.

Hay que reseñar que las variables asignadas al layout no son visibles en las vistas.

3.3. Otros View Helpers

Si repasamos el layout que viene por defecto en el módulo Application observaremos una gran cantidad de helpers:

- doctype()
- headMeta()
- headLink()

- headScript()
- basePath()

Por citar algunos. Todos ellos nos ayudarán en las tareas de generar los tags HTML presentes en cualquier página.

Los anteriores son helpers normalmente útiles en layouts. ZF2 nos ofrece otros helpers que podemos utilizar también en las vistas:

- translate()
- url()
- FlashMessenger()
- escapeHtml()

Para no hacer este manual demasiado extenso y por tanto, difícil de leer. No vamos a entrar a explicar el uso de cada uno de ellos. La información disponible en la documentación oficial es más que suficiente y deberemos acudir a ella a medida que necesitemos usar algunos de los helpers anteriores u otros que no hayamos citado.

3.4. View Helpers propios

Tenemos la posibilidad de crear nuestros propios View Helpers.

Crearemos un View Helper para los formularios que usamos en nuestro módulo Calculator. De esta manera tendremos dos opciones: partials y View Helpers.

Crearemos nuestra clase FormHelper dentro de src/Calculator/View/Helper. Nuestro Helper debe heredar de Zend\View\Helper\AbstractHelper.

Debemos implementar el método __invoke que será el que ZF2 busque cuando lo utilicemos en una vista. Podemos pasarle argumentos a este método. En nuestro caso podemos parametrizar el formulario con el action y el caption del botón submit.

```
<?php
/**
 * xenFramework (http://xenframework.com/)
 *
 * This file is part of the xenframework package.
 *
 * (c) Ismael Trascastro <itrascastro@xenframework.com>
 *
 * @link        http://github.com/xenframework for the canonical source repository
 * @copyright   Copyright (c) xenFramework. (http://xenframework.com)
 * @license     MIT License - http://en.wikipedia.org/wiki/MIT_License
 *
 * For the full copyright and license information, please view the LICENSE
```

```
 * file that was distributed with this source code.
 */

namespace Calculator\View\Helper;


use Zend\View\Helper\AbstractHelper;

class FormHelper extends AbstractHelper
{
    /**
     * @var string
     */
    private $action;

    /**
     * @var string
     */
    private $submit;

    public function __invoke($action, $submit)
    {
        $this->action = $action;
        $this->submit = $submit;

        return $this->render();
    }

    private function render()
    {
        return '
            <form id="calculatorForm" action="' . $this->action . '" method="post">
                <input type="number" name="op1"><br>
                <input type="number" name="op2"><br>
                <input type="submit" value="' . $this->submit . '">
            </form>
        ';
    }
}
```

Lo siguiente que debemos hacer es dar de alta el Helper. Para ello vamos al module.config.php y creamos una nueva entrada:

```
'view_helpers' => array(
    'invokables' => array(
        'formHelper' => 'Calculator\View\Helper\FormHelper',
    ),
),
```

Y ahora ya podemos usarlo en nuestras vistas:

```
<div>
   <?php echo $this->formHelper($this->url('calculator_addDo'), 'Add'); ?>
</div>
```

También podríamos haber registrado el View Helper en el fichero Module.php:

```
public function getViewHelperConfig()
{
   return array(
      'factories' => array(
         'formHelper' => function($m) {
            return new FormHelper();
         }
      ),
   );
}
```

4. Child Views

Una opción alternativa al uso del View Helper partial() es el uso de Child Views (vistas hijas). En este caso creamos una serie de vistas hijas, cada una con su template y se las asignamos a una vista padre. Luego en la vista padre podemos renderizar las vistas hijas donde queramos.

Por ejemplo en el caso del partial menu podríamos haberlo hecho con una vista child:

```
$menu = new ViewModel();
$menu->setTemplate('partial/menu');

$this->layout()->addChild($menu, 'menu');
```

Donde 'partial/menu' es el identificador que le hemos dado a esa vista bajo la key 'view_manager' en el fichero module.config.php:

```
'view_manager' => array(
   'display_not_found_reason' => true,
   'display_exceptions'       => true,
   'doctype'                  => 'HTML5',
   'not_found_template'       => 'error/404',
   'exception_template'       => 'error/index',
   'template_map' => array(
      'layout/layout'           => __DIR__ . '/../view/layout/layout.phtml',
      'calculator/index/index'  => __DIR__ . '/../view/calculator/index/index.phtml',
      'error/404'               =>    DIR    . '/../view/error/404.phtml',
      'error/index'             => __DIR__ . '/../view/error/index.phtml',
      'partial/form'            => __DIR__ . '/../view/calculator/partial/form.phtml',
```

```
        'partial/menu'          => __DIR__ . '/../view/calculator/partial/menu.phtml',
    ),
    'template_path_stack' => array(
            __DIR__ . '/../view',
    ),
),
```

y luego mostrarla en el layout en este caso (se procedería igual si fuera en una vista):

```
<div><?php echo $this->menu; ?> </div>
```

5. Contenido estático en los módulos

http://stackoverflow.com/questions/10296920/how-to-merge-zend-framework-2-module-public-directories

http://ocramius.github.io/blog/asset-manager-for-zend-framework-2/

http://es.slideshare.net/stefanovalle/zf2-asset-management