Vues

les classes Phalcon\Mvc\View et Phalcon\Mvc\View\Simple permettent la manipulation des vues dans le cadre du design pattern MVC.

-- Intégration des vues avec les contrôleurs

Phalcon passe automatiquement l'exécution à un composant de type vue dès qu'un contrôleur à terminé son chargement. La vue à charger est recherchée dans le dossier views, dans un sous-dossier du même nom que le dernier contrôleur invoqué, le fichier de vue portant le nom de la dernière action exécutée.
Pour prendre un exemple, si une requête est faite vers l'url **http://127.0.0.1/blog/posts/show/301**, Phalcon va parser l'url de la façon suivante :

Root de l'application	blog
Controller	posts
Action	show
Paramètre	301

Le dispatcher recherche “**PostsController**” et son action “**showAction**”.

Exemple de contrôleur :

```
<?php

class PostsController extends \Phalcon\Mvc\Controller{

    public function indexAction(){

    }

    public function showAction($postId){
        // Passage du paramètre $postId (301 dans l'exemple) à la vue
        $this->view->setVar("postId", $postId);
    }

}
```

La méthode **setVar** permet de créer des variables à la demande de façon à ce qu'elles puissent être utilisées dans la vue. L'exemple montre comment passer des paramètres ($postId) à la vue.

-- Scan hiérarchique

Le composant par défaut pour les vues (Phalcon\Mvc\View) gère l'affichage à partir d'une hiérarchie de fichiers. Cette hiérarchie permet d'utiliser des zones de layout (fréquemment utilisées dans les vues) à condition que les templates soient présents dans les dossiers correspondant au contrôleur et à l'action.

Phalcon\Mvc\View utilise PHP par défaut comme moteur de template, à condition que les vues aient l'extension **.phtml**. Si le dossier des vues est app/views (comme défini dans la configuration) le composant view recherchera automatiquement les 3 fichiers suivants :

Rôle	Fichier	Description
Main Layout	app/views/index.phtml	Affiché pour chaque contrôleur et chaque action de l'application
Controller Layout	app/views/layouts/posts.phtml	Vue relative au contrôleur, visible pour l'exécution de chacune des action du contrôleur “**posts**”. Tout le code implémenté dans ce layout sera utilisé pour toutes les actions du contrôleur
Action View	app/views/posts/show.phtml	Vue relative à l'action, visible uniquement si l'action “**show**” est exécutée.

```
<!-- app/views/index.phtml -->
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <h1>main layout!</h1>

        <?php echo $this->getContent() ?>

    </body>
</html>
```

```
<!-- app/views/layouts/posts.phtml -->

<h2>"posts" controller layout!</h2>

<?php echo $this->getContent() ?>
```

```
<!-- app/views/posts/show.phtml -->

<h3>show view!</h3>

<p>Paramètre passé : <?php echo $postId ?></p>
```

Résultat :


```
<!-- app/views/index.phtml -->
<!DOCTYPE html>
<html>
    <head>
        <title>Phalcon PHP Framework Layout example</title>
    </head>
    <body>
        <h1>Main layout!</h1>
<!-- app/views/layouts/posts.phtml -->

<h2>"posts" controller layout!</h2>

<!-- app/views/posts/show.phtml -->

<h3>show view!</h3>

<p>Paramètre passé : 301</p>    </body>
</html>
```

-- Utilisation de templates

Les templates permettent de factoriser et de partager une partie de l'affichage.

Exemple : Utilisation d'un template pour affichage d'une action

```
class PostsController extends \Phalcon\Mvc\Controller{
    public function initialize(){
        $this->view->setTemplateAfter('common');
    }
...
    public function lastAction(){
        $this->flash->notice("Derniers posts");
    }
}
```

La méthode **setTemplateAfter** de la classe **view** applique le template après le controller layout.

Le template **common** :

```
<!-- app/views/layouts/common.phtml -->

<ul class="menu">
    <li><a href="/">Home</a></li>
    <li><a href="/articles">Articles</a></li>
    <li><a href="/contact">Contact</a></li>
</ul>

<div class="content"><?php echo $this->getContent() ?></div>
```

La vue correspondant à l'action **last** :

```
<!-- app/views/posts/show.phtml -->

<h3>last view!</h3>
```

-- Contrôle hiérarchique du niveau d'affichage

La méthode **setRenderLevel** permet de définir le niveau de présentation

```
        $this->view->setRenderLevel(View::LEVEL_NO_RENDER);
```

Il est également possible de désactiver l'affichage,

ponctuellement, dans un contrôleur :

```
        $this->view->disableLevel(View::LEVEL_MAIN_LAYOUT);
```

ou **de manière générale** dans le fichier bootstrap :

```
<?php

use Phalcon\Mvc\View;

$di->set('view', function(){

    $view = new View();

    //Disable several levels
    $view->disableLevel(array(
        View::LEVEL_LAYOUT => true,
```

```
        View::LEVEL_MAIN_LAYOUT => true
    ));

    return $view;

}, true);
```

- Choix spécifique de vues

Il est également possible de déterminer précisément l a vue à afficher avec la méthode **pick** :

```
        $this->view->pick("products/search");
```

Ou de désactiver l'affichage de vues (s'il s'agit d'une réponse ajax ou d'une redirection :

```
        $this->view->disable();
```

-- Scan Simple

L'alternative au scan hierarchique est offert par la classe Phalcon\Mvc\View\Simple.

Ce composant plus classique permet de choisir précisément la vue à afficher.

Le composant par défaut doit être remplacé dans le fichier bootstrap lors de l'initialisation du service container $di :

```
<?php

$di->set('view', function() {

    $view = new Phalcon\Mvc\View\Simple();

    $view->setViewsDir('../app/views/');

    return $view;

}, true);
```

Démarrage de l'application et désactivation du rendering automatique :

```
<?php

try {

    $application = new Phalcon\Mvc\Application($di);

    $application->useImplicitView(false);
```

```
    echo $application->handle()->getContent();

} catch (\Exception $e) {
    echo $e->getMessage();
}
```

Exemples d'affichage de vues :

```
<?php

class PostsController extends \Phalcon\Mvc\Controller{

    public function indexAction(){
        //Render 'views-dir/index.phtml'
        echo $this->view->render('index');

        //Render 'views-dir/posts/show.phtml'
        echo $this->view->render('posts/show');

        //Render 'views-dir/index.phtml' passing variables
        echo $this->view->render('index', array('posts' => Posts::find()));

        //Render 'views-dir/posts/show.phtml' passing variables
        echo $this->view->render('posts/show', array('posts' => Posts::find()));
    }

}
```

-- Inclusion de templates partiels

```
<div class="top"><?php $this->partial("shared/ad_banner") ?></div>

<div class="content">
    <h1>Robots</h1>

    <p>Check out our specials for robots:</p>
    ...
</div>

<div class="footer"><?php $this->partial("shared/footer") ?></div>
```

Il est également possible de passer des variables à la vue partielle :

```
<?php $this->partial("shared/ad_banner", array('id' => $site->id, 'size' => 'big'))
?>
```

-- Passage de variables du contrôleur à la vue

-- Variable simple

```
   public function showAction()
    {
        //Passage du nombre de posts à la vue
        $this->view->setVar("postsCount", 5);
```

```
echo $postCount;
```

-- Passage de plusieurs variables

```
        $this->view->setVars(array(
            'title' => $post->title,
            'content' => $post->content
        ));
```

```
echo $title;
echo $content;
```

-- Mise en cache

voir http://docs.phalconphp.com/en/latest/reference/views.html#caching-view-fragments

-- Evènements sur les vues

Les vues ont la possibilité d'envoyer des évènements à l'eventsManager s'il est présent.

Nom	Evènement	Interruptible ?
beforeRender	avant interprétation	Yes
beforeRenderView	Avant interprétation d'une vue existante	Oui
afterRenderView	Après interprétation d'une vue existante	Non
afterRender	Après interprétation	Non
notFoundView	Déclenché si la vue est inexistante	Non

Exemple de configuration avec eventsManager :

```
$di->set('view', function() {
```

```
    //création de l'eventsManager
    $eventsManager = new Phalcon\Events\Manager();

    //Association d'un listener pour le type "view"
    $eventsManager->attach("view", function($event, $view) {
        echo $event->getType(), ' - ', $view->getActiveRenderPath(), PHP_EOL;
    });

    $view = new \Phalcon\Mvc\View();
    $view->setViewsDir("../app/views/");

    //Associe l'eventsManager au composant $view
    $view->setEventsManager($eventsManager);

    return $view;

}, true);
```

L'exemple suivant montre comment créé un plugin permettant de nettoyer/réparer le code HTML produit avec tidy :

```
<?php

class TidyPlugin{

    public function afterRender($event, $view){

        $tidyConfig = array(
            'clean' => true,
            'output-xhtml' => true,
            'show-body-only' => true,
            'wrap' => 0,
        );

        $tidy = tidy_parse_string($view->getContent(), $tidyConfig, 'UTF8');
        $tidy->cleanRepair();

        $view->setContent((string) $tidy);
    }

}

//Associe le plugin en tant que listener sur l'événement afterRender
$eventsManager->attach("view:afterRender", new TidyPlugin());
```

La suite : View Helpers