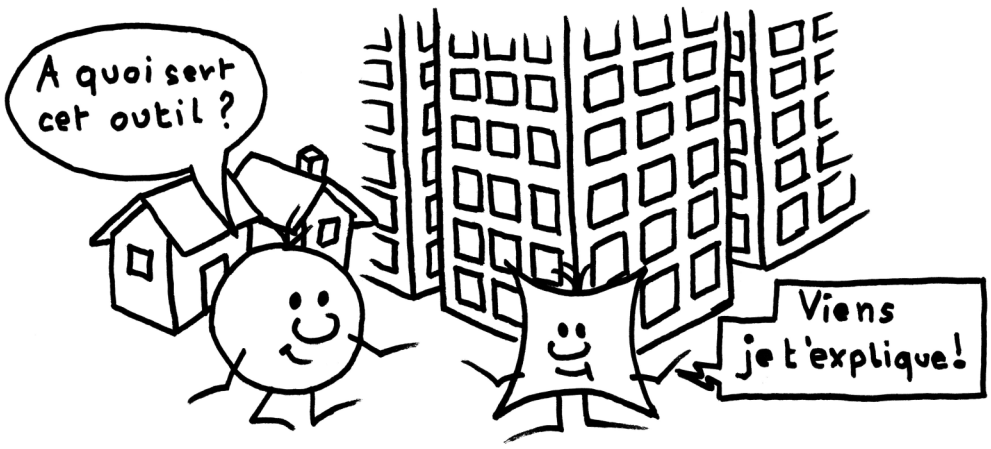
chapitre 6



Architecture MVC

Rares sont les applications web qui n'utilisent pas de près ou de loin une architecture de type MVC. Motif de conception spécialisé dans l'organisation globale d'une application, MVC propose une séparation entre le design, la gestion des données et la logique de navigation.

SOMMAIRE

COMPOSANTS

- Zend_Controller
- Zend_Layout
- Zend_View

MOTS-CLÉS

- MVC
- modèle
- vue
- contrôleur
- architecture
- template
- action
- navigation

Zend Framework propose sa propre implémentation de MVC. Celle-ci est pensée pour être souple et paramétrable. Le composant `Zend_Controller`, au cœur de cette implémentation, peut être utilisé simplement avec sa configuration par défaut ou, de manière avancée, grâce à un système de routage et de configuration étendu.

Ce chapitre est divisé en trois grandes parties :

- `Zend_Controller` – *utilisation simple* : une introduction pratique et accessible sur ce composant central. Cette partie permettra de créer une application MVC en quelques minutes en adoptant une architecture minimale.
- `Zend_Layout`, `Zend_View` : de ces composants dépend toute la logique d'affichage client de Zend Framework. Nous abordons ici leur fonctionnement et leurs possibilités étendues.
- *Le modèle* : extraction et structuration des données à afficher.

Enfin, le chapitre suivant est consacré à l'utilisation avancée de `Zend_Controller`. Son objectif sera de vous faire comprendre le fonctionnement interne de ce composant et de vous aider à en maîtriser toute la souplesse et les possibilités. Nous verrons entre autres l'utilisation de modules, la création de *plugins* et les paramétrages avancés de tous les objets au cœur du modèle MVC de Zend Framework.

PRÉREQUIS MVC

Avant d'aborder ce chapitre, il est important d'avoir compris en théorie ce qu'est une architecture MVC. L'annexe E est consacrée à la présentation théorique de MVC.

Zend_Controller : utilisation simple

Nous vous proposons ici de mettre en place en quelques minutes une architecture MVC minimale avec Zend Framework. Ainsi, nous allons structurer notre application dans les règles de l'art.

Mettre en place l'architecture

PRATIQUE Zend Studio

Le logiciel Zend Studio For Eclipse permet de mettre sur pied un projet type, en quelques clics seulement. Vous trouverez de plus amples informations à ce sujet au chapitre 14.

Avant toute chose, nous devons créer des dossiers et des fichiers de base – ils sont représentés sur la figure 6-1. Il vous suffit pour cela de créer les dossiers représentés ainsi que les fichiers, qui, dans un premier temps, seront vides.

Dans une application Zend Framework, la partie MVC est située dans un dossier à part, nommé par défaut `application`. Trois sous-dossiers `controllers`, `views` et `models` ont des noms explicites quant à leur contenu.

- Dans le dossier `controllers`, le fichier `IndexController.php` contient le contrôleur principal de l'application, c'est-à dire celui qui est appelé par défaut.

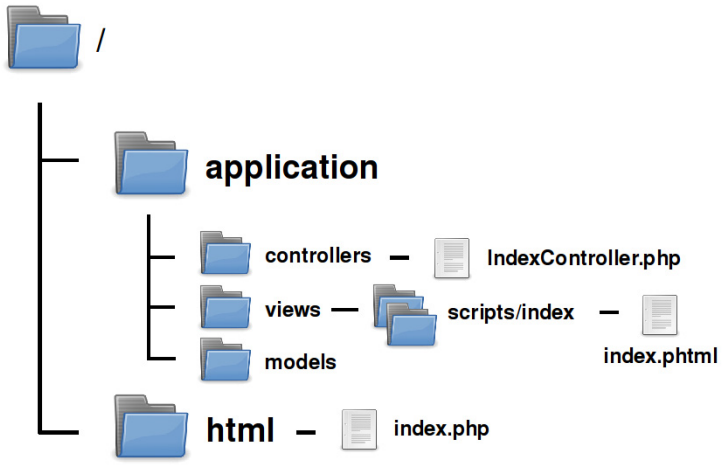


Figure 6–1
Dossiers et fichiers pour une architecture MVC minimale

- Le fichier `index.phtml`, dans `views/index`, correspond à la vue du contrôleur principal. En fait, il s'agit de la vue de l'action `index` (nom du fichier) du contrôleur `index` (nom du dossier contenant).
- Le fichier `html/index.php` est ce que l'on appelle un *bootstrap* ou « fichier d'amorçage ». C'est vers ce fichier que toutes les requêtes HTTP sont redirigées, mis à part celles des fichiers statiques (images, CSS, JavaScript...).

PERFORMANCE Bootstrap

Le *bootstrap* (ou fichier d'amorçage) n'est pas spécifique au Zend Framework. On le retrouve dans de nombreux projets basés sur MVC. Il est le point d'entrée de toute requête sollicitant la création d'une page dynamique. Cela fait aussi de lui le goulet d'étranglement de l'application ! Il est donc important de veiller à la présence de chacun des objets, de ne pas en créer d'inutiles, ou encore d'utiliser des caches aussi souvent que possible.

Parcours d'une requête HTTP

Une fois munis des dossiers et fichiers principaux, il est important de comprendre le parcours de notre requête HTTP. La figure 6-2 illustre cette opération avec un exemple de requête vers un export de réservations.

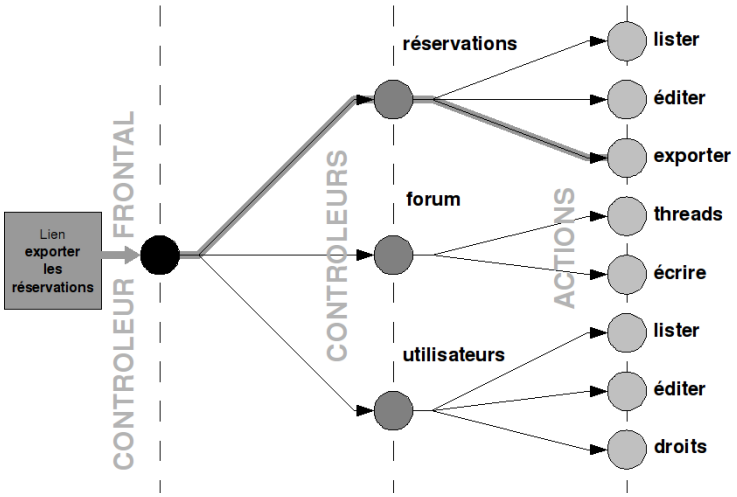


Figure 6–2
Appel de l'action correspondant à la requête HTTP

Module

Le module est une dimension supplémentaire du modèle MVC proposé par `Zend_Controller`. Reservés aux sites de taille importante, les modules sont des dossiers qui encapsulent chacun un trio Modèle-Vue-Contrôleur. Conceptuellement, nous pouvons assimiler les modules à des sous-sites, inclus dans le site actuel.

Routage

Cette règle qui lie le format de l'URL aux appels des actions est un comportement par défaut dans Zend Framework. Il est possible de modifier tout ou partie de ce comportement grâce aux mécanismes de routage qui seront abordés plus loin dans ce chapitre.

Notre requête passe d'abord par le *contrôleur frontal* qui est instancié dans le bootstrap (`html/index.php`). Ce contrôleur frontal va déterminer quel *contrôleur* et quelle *action* doivent être appelés. C'est ainsi, par exemple, que le contrôleur `réservations` est instancié et que l'action `exporter` est appelée. Techniquement, un contrôleur est une classe, et l'action une méthode de cette dernière.

Dernière chose importante à savoir : dans Zend Framework, par défaut, le contrôleur et l'action à appeler dépendent de l'URL. Ce mécanisme est illustré par la figure 6-3.

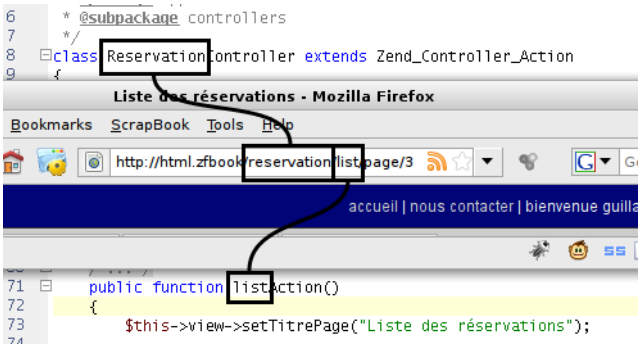


Figure 6–3
Appel de l'action correspondant aux paramètres de l'URL

À la racine de l'URL, le premier mot entre deux caractères « / » est le nom du contrôleur à appeler et le deuxième est le nom de l'action. Lorsqu'on ne spécifie pas le contrôleur et l'action dans l'URL, ils sont par défaut nommés `index` et `index`. De cette manière, la méthode `indexAction()` de la classe `IndexController` sera automatiquement appelée.

Exemple simple d'utilisation de Zend_Controller

Passons maintenant au code source. Une fois les dossiers et les fichiers créés, puis le principe compris, il reste l'implémentation. Voici dans un premier temps à quoi ressemble l'instanciation du contrôleur frontal dans le bootstrap :

Contenu du bootstrap html/index.php

```
// Utilisation de Zend_Loader
require_once 'Zend/Loader.php';

// Chargement automatique des classes
Zend_Loader::registerAutoload();
```

```
// Appel du contrôleur frontal,
// qui se charge de traiter la requête
Zend_Controller_Front::run('../application/controllers');
```

Les première et deuxième lignes correspondent simplement à l'appel de l'autoload qui permet le chargement automatique des classes. C'est la troisième ligne qui nous intéresse ici.

La méthode `run()` de la classe `Zend_Controller_Front` instancie le contrôleur frontal avec, comme paramètre, le chemin vers les contrôleurs. Il est bien entendu possible de faire cela de manière plus minutieuse, mais nous nous limitons volontairement ici à cette version minimale, par souci de clarté.

Voyons ensuite ce que contient le fichier `IndexController.php`, qui renferme le contrôleur et l'action :

Contenu du contrôleur application/controllers/IndexController.php

```
// La classe correspondant au contrôleur index
// (contrôleur par défaut)
class IndexController extends Zend_Controller_Action
{
    // L'action index
    public function indexAction()
    {}
}
```

Toujours en version minimale, nous pouvons déduire deux informations essentielles de ce script :

- le contrôleur est la classe `IndexController`. Tout contrôleur Zend Framework étend la classe `Zend_Controller_Action` ;
- l'action est la méthode `indexAction()` de la classe `IndexController`. Toute action Zend Framework est une méthode suffixée par le mot-clé `Action`.

Enfin, il nous reste à voir le contenu de la vue `views/index/index.phtml` :

Contenu de la vue views/index/index.phtml

```
<p>Bonjour le monde !</p>
```

La vue contient simplement du code HTML. Bien sûr, il pourra y avoir un peu de PHP par la suite, ce que nous verrons dans la section `Zend_View`. Zend Framework appelle automatiquement la vue. Si celle-ci n'existe pas, une erreur est générée.

RAPPEL Autoload

L'autoload est une fonctionnalité PHP abordée en annexe B. `Zend_Loader` est une classe encapsulant un mécanisme d'autoload ; elle est détaillée dans le chapitre 4.

Mettre en place le squelette de l'application

Reprenons ici notre application de gestion de salles. Nous allons simplement créer les fichiers qui correspondent aux contrôleurs et aux vues, puis les classes qui correspondent à nos contrôleurs.

Un contrôleur `ReservationController` sera tout particulièrement important dans notre application. Il a déjà été implicitement abordé dans les illustrations 6-2 et 6-3 précédentes. C'est à travers ce contrôleur que nous accéderons aux fonctionnalités principales de notre application.

Contrôleurs	Actions	Vues

`IndexController`	`indexAction()`
	`contactAction()`
	`languageAction()`
		`index/index`
		`index/contact`
		`pas de vue`

`LoginController`	`indexAction()`
	`loginAction()`
	`logoutAction()`
		`login/welcome ou login/loginform`
		`pas de vue`
		`pas de vue`

`Reservation Controller`	`indexAction()`
	`listAction()`
	`editAction()`
	`deleteAction()`
	`exportAction()`
		`reservation/index`
		`reservation/list+`
		`reservation/edit`
		`pas de vue`
		`reservation/export`

`ErrorController`	`errorAction()`	`error/error`

`Webservice Controller`	`rssAction()`
	`soapAction()`
	`restAction()`
		`pas de vue`
		`pas de vue`
		`pas de vue`

Figure 6–4
Squelette prévisionnel de l'application

La figure 6-4 présente les classes, actions et vues qui interviennent dans le squelette prévisionnel de notre application. Cette organisation peut changer au fur et à mesure des développements, mais il est intéressant d'avoir un aperçu de ce squelette afin d'assurer une répartition cohérente des fonctionnalités dans les contrôleurs et les actions.

Voici ce que nous pouvons dire de cette répartition prévisionnelle :

- Le contrôleur `IndexController` va gérer la page d'accueil générale de l'application. Il comportera la page d'accueil avec sa vue associée, un formulaire de contact et assurera aussi le changement de langue. Seul le changement de langue n'a pas besoin de vue, car nous souhaitons que cette opération se fasse sur la page courante.
- `LoginController` est responsable du traitement du formulaire d'identification (*login*) situé en haut à droite de toute page. Ce contrôleur donne un exemple d'action ayant plusieurs vues qui ne représentent pas des pages, mais seulement des blocs faisant partie du gabarit (*layout*) de l'application.

RENVOI **MVC avancé**

Par défaut, une action est toujours liée à une vue qui porte son nom. Cette politique, bien que conseillée pour une organisation cohérente, n'est pas une obligation. Il est possible de lier une action à une ou plusieurs vues manuellement, ou à aucune, en fonction des besoins. Ce sera le cas dans notre application, comme nous pouvons le voir sur la figure 6-4. Ces changements par rapport aux comportements par défaut seront expliqués ultérieurement.

- `ReservationController` comportera les fonctionnalités principales de l'application de réservation. Il permettra l'accès aux fonctionnalités de lecture et d'édition des réservations. L'action `list` est liée à une fonctionnalité spéciale appelée `ContextSwitch` que nous aborderons plus tard et qui permet de sélectionner une vue parmi plusieurs en fonction d'un contexte (HTML, RSS...).
- `ErrorController` est appelé automatiquement lorsqu'une exception est levée dans le modèle MVC. Nous aborderons ce contrôleur spécial plus loin.
- Enfin, `WebserviceController` sera spécialement dédié aux services web et aux flux. Ces mécanismes ne nécessitent pas de vue, comme nous le verrons dans le chapitre 12 consacré aux services web.

Code du squelette

Le contenu de chaque contrôleur est basé sur le même principe : une classe qui représente le contrôleur et des méthodes qui correspondent aux actions, comme nous l'avons vu précédemment. Voici un exemple de squelette pour le contrôleur `ReservationController` :

Squelette de ReservationController

```
class ReservationController extends Zend_Controller_Action
{
    // fait appel à la vue reservation/index.phtml
    public function indexAction()
    {
    }

    // fait appel aux vues reservation/list.*.phtml
   // (contextswitch)
    public function listAction()
    {
    }

    // fait appel à la vue reservation/edit.phtml
    public function editAction()
    {
    }

    // ne fait appel à aucune vue, appelle la page précédente
    public function deleteAction()
    {
    }

    // fait appel à la vue reservation/export.phtml
    public function exportAction()
    {
    }
}
```

Les vues sont situées comme prévu dans le dossier `views/scripts`. Toute vue doit comporter l'extension `*.phtml`. Dans un premier temps, nos fichiers de vues seront vides. Il est néanmoins nécessaire de les créer, sinon une erreur sera signalée.

Attribuer des paramètres à la vue

Revenons à notre apprentissage de l'implémentation MVC proposée par Zend Framework. Comme nous venons de le voir, par défaut, un contrôleur est lié à une vue. L'emplacement de la vue dépend du contrôleur et de l'action sollicités. Par exemple, l'appel de l'action `ReservationController::listAction()` est lié à la vue `views/scripts/controller/list.phtml`.

Il est possible d'attribuer des paramètres à la vue depuis le contrôleur. Cette opération est très courante (c'est le rôle théorique des contrôleurs), car elle permet d'afficher toutes les informations dynamiques telles que le contenu de la base de données. Pour reprendre notre exemple simple, nous allons juste attribuer un paramètre `$title` à la vue depuis notre contrôleur :

Attribution d'un paramètre dynamique à la vue (méthode 1)

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $this->view->assign('title', 'Bonjour le monde');
    }
}
```

Voici au passage une autre manière de déclarer le paramètre `$title` en utilisant les *méthodes magiques* internes à `Zend_View`. Cette méthode est la plus utilisée.

Attribution d'un paramètre dynamique à la vue (méthode 2)

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $this->view->title = 'Bonjour le monde';
    }
}
```

L'affichage du contenu de ce paramètre dans la vue est simple. Il n'y a qu'une seule chose à comprendre : la vue est située dans l'objet

Méthodes magiques

Les méthodes magiques ont des noms prédéfinis qui commencent par deux traits de soulignement « __ ». Elles proposent des automatismes spécifiques à l'implémentation objet de PHP 5. Vous trouverez des informations détaillées sur les méthodes magiques dans l'annexe C consacrée à la POO.

`$this->view` de notre contrôleur. En d'autres termes, la variable spéciale `$this` dans la vue correspond à `$this->view` dans le contrôleur (agrégation). Il est alors aisé de récupérer le titre pour l'afficher :

Affichage du paramètre title dans la vue

```
<p><?php echo $this->title; ?> !</p>
```

Toute attribution de paramètre se fait de cette manière. Cette méthode permet entre autres de filtrer et d'isoler tous les paramètres qui sont passés à la vue.

Manipulation des données HTTP

L'accès direct aux variables superglobales du type `$_POST`, `$_GET`, ainsi que l'appel direct de fonctions telles que `header()`, sont déconseillés avec Zend Framework. Les contrôleurs d'action disposent de méthodes `getRequest()` et `getResponse()` qui permettent un accès à des objets `Request` et `Response`. Passer par ces objets permet, en théorie, d'éliminer les aléas du langage (configuration et évolutions au fil des nouvelles versions).

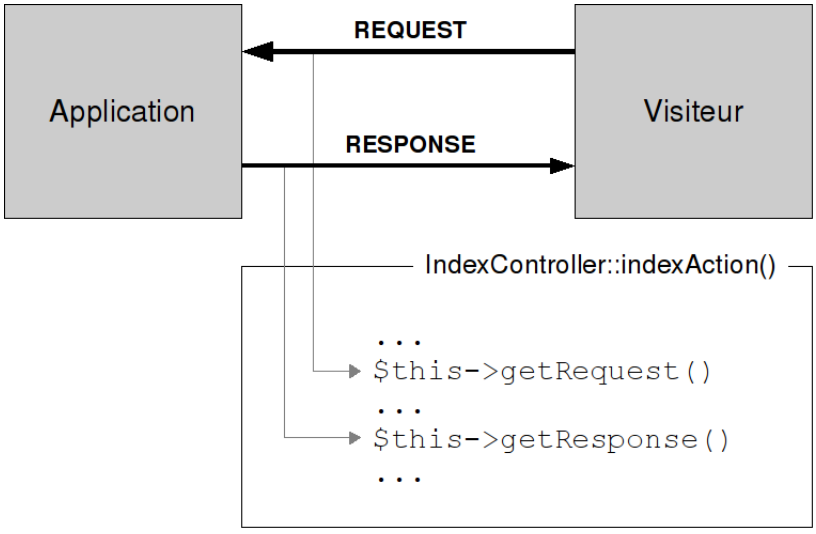


Figure 6–5
Accès aux objets Request et Response de Zend Framework

La requête et la réponse dont il est question ici font référence au visiteur (client HTTP). Il est important de prendre cela en compte de manière à ne pas les confondre. La figure 6-5 illustre la manière dont on accède à ces objets et ce à quoi ils correspondent :

- `getRequest()` donne accès à l'objet `Zend_Controller_Request_Http` qui contient :
  - les paramètres HTTP de formulaires ou de barres d'adresse `$_POST` et `$_GET`, mais aussi `$_COOKIE`, `$_SERVER` et `$_ENV` ;
  - les noms du module, du contrôleur et de l'action courante.
- `getResponse()` donne accès à l'objet `Zend_Controller_Response_Http` qui contient les données de la réponse HTTP :
  - les en-têtes HTTP ;
  - le contenu de la réponse (`body`) ;
  - les éventuelles exceptions rencontrées lors de la construction de la réponse ;
  - le code de la réponse et d'autres informations concernant la réponse (redirection, exceptions potentiellement levées par le *renderer*, etc.).
- `_getParam()` est un raccourci qui permet de récupérer des paramètres de l'objet requête. L'appel `$this->getRequest->getParam()` a le même effet.

Ces méthodes seront largement utilisées dans les pages qui comportent des formulaires et des mécanismes liés aux paramètres HTTP. Dans ce chapitre, reportez-vous aux sections `Zend_Layout` pour un exemple de récupération du contrôleur, et à la gestion des erreurs pour un exemple avec `_getParam()`.

Afin de se familiariser avec la requête et la réponse, voici une petite action qui effectue une copie de sauvegarde (*dump*) de ces deux objets :

L'action IndexController::infoAction() qui effectue le dump

```
/**
* Dump de la requête et de la réponse
*/
public function infoAction()
{
    if ($this->getInvokeArg('debug') == 1) {
        $this->getResponse()->setHeader('Cache-control', 'no-cache');
        $this->view->setTitrePage("Contenu de request et response");
        $this->view->request = $this->getRequest();
        $this->view->response = $this->getResponse();
    }
}
```

Cette action, qui est traitée uniquement en mode *debug*, ajoute un en-tête de réponse HTTP via `getResponse->setHeader()` et passe les objets `Request` et `Response` à la vue. Celle-ci affiche simplement un dump de ces paramètres :

Dump des paramètres request et response (index/info.phtml)

```
<div style="float: right">
  <?php var_dump($this->response); ?>
</div>
<?php var_dump($this->request); ?>
```

Le résultat de ces dumps est illustré par la figure 6-6.

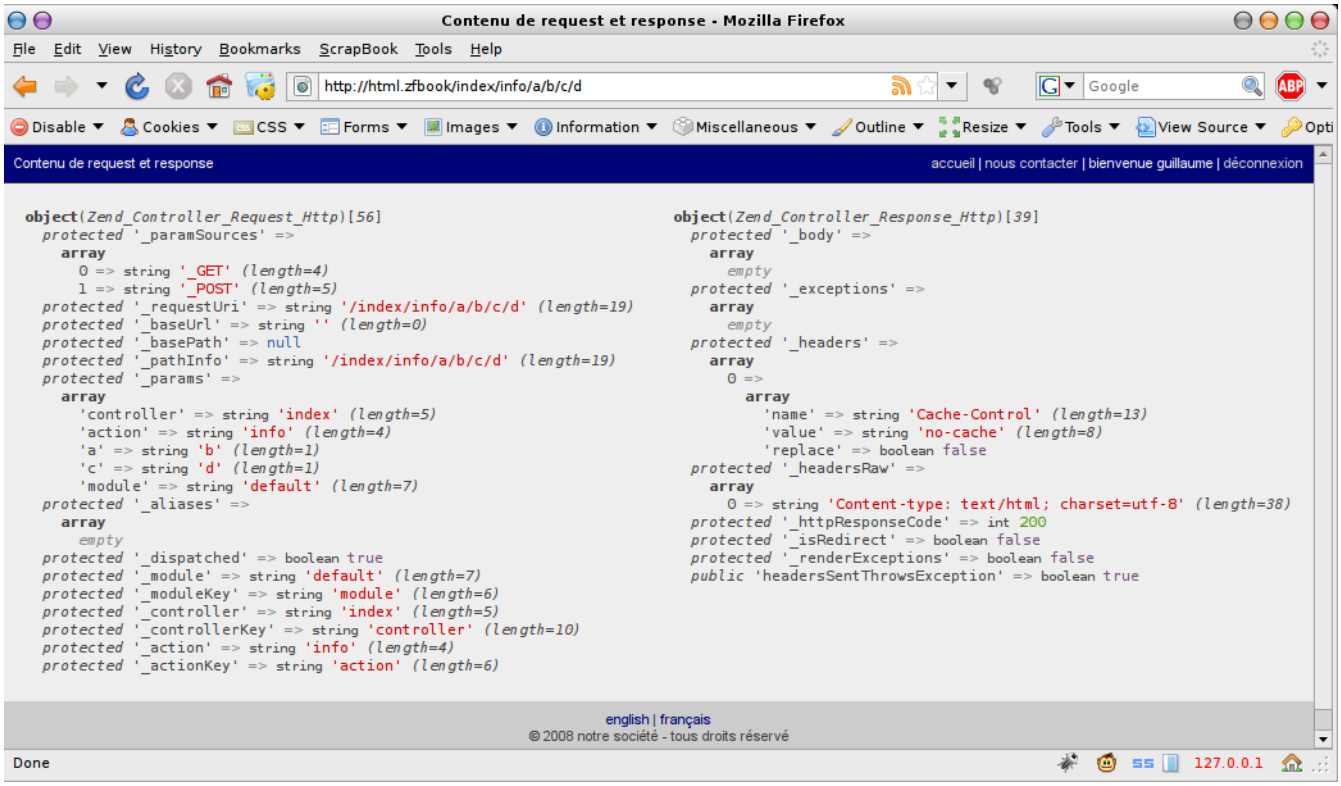


Figure 6–6 Dump des objets Request et Response

Initialisation et postdispatch

Il est possible de factoriser des traitements communs effectués en début ou en fin de plusieurs actions grâce aux méthodes d'initialisation et de *post-dispatching* :

- la méthode `init()` est appelée à la construction du contrôleur d'action ;
- la méthode `preDispatch()` est appelée avant chaque action ;
- la méthode`postDispatch()` est appelée après chaque action.

Nous pouvons utiliser la méthode `init()`, par exemple, dans le contrôleur `ReservationController`, afin de déclarer un titre par défaut :

Déclaration d'un titre par défaut dans la méthode spéciale init()

```
class ReservationController extends Zend_Controller_Action
{
    public function init()
    {
        $this->view->setTitrePage("Réservation des salles");
    }
}
```

Comme nous pouvons le voir, on peut développer dans la méthode `init()` de la même manière que dans n'importe quelle action. L'accès à la vue est possible, de même bien sûr qu'à `getRequest()`, `getResponse()`, et à tout autre objet de notre contrôleur.

Un exemple avec `postDispatch()` est illustré dans le chapitre des services web. Celui-ci explique comment faire appel au mécanisme de chargement du service demandé de manière similaire pour chaque type de traitement RSS, SOAP ou REST.

Zend_Layout : créer un gabarit de page

Il est courant que chaque page comporte des parties communes, telles que l'en-tête et le pied de page, le menu et les styles CSS. `Zend_Layout` va nous permettre de créer un gabarit qui nous évitera de dupliquer du code HTML d'une vue à l'autre.

Pour simplifier notre apprentissage, considérons que l'application ne comporte qu'un seul gabarit de page. Mais que cela ne nous limite pas dans l'absolu, il est bien sûr possible d'avoir plusieurs gabarits utilisés séparément ou en même temps.

La figure 6-7 illustre la composition du gabarit de toute page HTML liée à notre application. Ce gabarit est composé de cinq parties principales :

- les paramètres, situés dans la balise `<head>` de la page ;
- l'en-tête (`header`) de la page, comportant le titre et le formulaire de login ;
- le sous-menu, lorsque celui-ci est actif, ce qui sera le cas dans le contrôleur `ReservationController`pour afficher des liens vers les pages correspondant aux actions ;