Zend Server UI:
A ZF2 Case Study

ZF2.0 Evolution from a developer perspective
v1.1
Yonni Mendes

Thoughts, feedback: yonni.m@zend.com

ZF2 - Highlights

- Modern, up-to-date, hip and upbeat
- Zend & Community
  - GitHub, forums
  - Freenode #zftalk
- New and interesting toys
  - Events
  - Services
  - Uniform plugins/helpers
- Tried and True concepts
  - MVC pattern
  - Customizability
  - Di & resources

ZF2 - Lighter, stronger

ZS5 ZF1 UI	ZS6 ZF2 UI
100,657 lines of php code	78,724 lines of php code
8MB of php & phtml files	4.4MB of php & phtml files
1,161 php & phtml files	758 php & phtml files
518 directories (read: namespaces)	404 directories (read: namespaces)

* numbers were taken a week ago from the trunk

Eye catchers in ZF2

- Events
  - Promote modularity and encapsulation
  - Mitigates tight coupling between components
- Di & ServiceManager
  - Move object provisioning out of the application
    - but not necessarily into the configuration
  - Avoid singletons and promote use of factories
  - Avoid static calls
- ModuleManager
  - Compartmentalize your application
  - Promote reuse across applications
  - Promote extensibility by 3rd party

Eye catchers: Zend\Form

- Complete rewrite
  - Decorators were annihilated
  - Validators were extracted and are not part of the element
  - Factory functionality in a separate set of classes
  - Factory has annotations' support
- However
  - No more <?php echo $form ?> :(
  - Not even a view helper!
  - Some of the elements are tricky to use
    - checkbox
    - multiple selection

Eye pokers in ZF2

- Lambdas. Lots of Lambdas
  - Like really allot of them
- Avoid inheritance
  - ServiceManagers hate inheritance
  - Inject dependencies instead

Eye pokers in ZF2, more

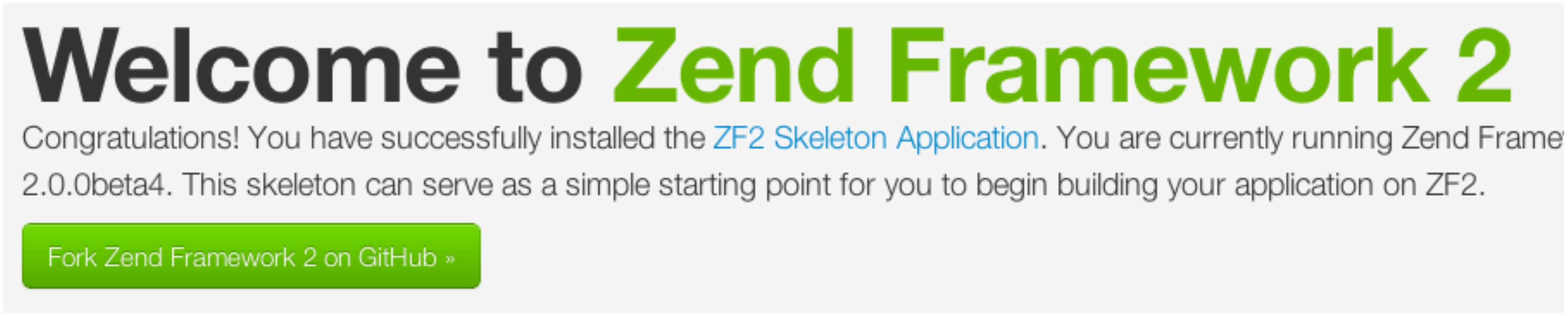
- No More Brokers
  - No more static brokers
  - ServiceManagers are the new brokers
  - Uniform configuration formats
  - Helpers and plugins are sort of the same thing now
- The Framework really likes itself
  - Overriding internal functionality / classes is not immediately obvious
  - Some components suffer from lack of extensibility options, some enforce arbitrary limitations

Initializing a ZF2 MVC application

Do, do more and don't

Oooh Spooky: The Skeleton Application

- MVC implementation based on ZF2
- Basic accepted practices
  - Modular structure
  - Separation of layers and responsibilities
- Getting used to
  - Modules and namespaces alignment
  - Views and dependencies are separated
- Unzip. Bam! it works



Welcome to Zend Framework 2

Congratulations! You have successfully installed the ZF2 Skeleton Application. You are currently running Zend Frame
2.0.0beta4. This skeleton can serve as a simple starting point for you to begin building your application on ZF2.

The Initialization

- `Module::init()`
  - Provided by `InitProviderInterface`
  - Called immediately after instantiation
  - Gets the module manager as a parameter
  - No Events, services have been started yet!

The Initialization, Cont'd

- LoadModules.post event
  - Triggered on the shared eventsManager
  - After all Module objects were initialized
  - Listeners get a generic ManagerEvent parameter
  - Configuration has been merged at this point
  - Setup application services like logging, configuration, caching...

The Initialization, Cont'd

- `Module::onBootstrap()`
  - "Should"
  - Provided by `BootstrapProviderInterface`
  - Called when called by Mvc\Application
  - Gets an MvcEvent parameter (Request, response ...)
  - Shared ServiceManagers are available at this point

Initialization fact to fun

- Distributed initialization
  - Separate bootstrap and init per module
  - Attach listeners to 'LoadModules.post' in `init()`
  - Attach listeners to route/dispatch in `onbootstrap()`
  - Do not attach anything to 'bootstrap' event
- 'LoadModules.post' execution
  - Runs all listeners in-order
  - Avoid dependencies between modules' initialization
- Template your `onBootstrap()`
  - Add initializers and Abstract Factories to ServiceManagers
  - Call things like ACL and View layout initialization

ZS6 Module::init() function

```
public function init(ModuleManagerInterface $manager =
null) {
     $manager->getEventManager()->
       attach('loadModules.post',
       array($this, 'initializeConfig'));

           ...

     $manager->getEventManager()->
       attach('loadModules.post',
       array($this, 'initializeDebugMode'));
}
```

ZS6 Module::onBootstrap()

```
public function onBootstrap(EventInterface $e) {
        $app = $this->application;
        $this->application = $e->getApplication();
        $baseUrl = static::config('baseUrl');
        $app->getRequest()->setBaseUrl($baseUrl);
...
            $this->initializeLog($e);
            $this->initializeRouter($e);
            $this->initializeSessionControl($e);
            $this->initializeACL($e);
            $this->initializeView($e);
            $this->initializeWebAPI($e);
            $this->initializeViewLayout($e);
...
        $this->detectTimezone();
}
```

Failure is NOT an option

Failure during initialization and bootstrap is problematic. A few ideas:

- Ignore errors
- Stop event propagation
- Signal failure on the event
- Throw an exception (burn!)

- Trigger a new event (dispatch.error)

The failure option

```
try {
....
} catch (\Exception $ex) {// $e is a MvcEvent, $ex is an exception
  $events = $this->application->getEventManager();
  $error = $e;
  $error->setError(\Zend\Mvc\Application::ERROR_EXCEPTION);
  $error->setParam('exception', new Exception('..', null, $ex));
  $results = $events->trigger(MvcEvent::EVENT_DISPATCH_ERROR,
$error);
  $e->stopPropagation(true);
  $e->setResult($results);
...
}
```

Authentication & Authorization

The three headed dragon

Authentication requirements

- Multiple users
- Secure passwords
- Different authentication options (simple, extended - ldap)
- Must provide for WebAPI authentication
- This is NOT session control!

Simple is as simple does

- At first
  - Authentication action plugin
  - Zend\Auth\AuthenticationService
  - Digest Adapter
- Not good enough for cluster, moved to DbTable adapter
- Had to extend DbTable and override
  - Credential treatment is hardcoded to be in SQL
  - Wanted to return an Identity Object, instead of a string

Extended Authentication

Essentially similar to Simple

- Extended Zend\Auth\Ldap
  - Add support for Identity class
  - Add groups membership handling for ACL
- Custom authentication for Zend Server
  - Specify a custom "Adapter" class in ui configuration
  - Support either groups or simple roles
  - Example and start up code in github, fork away!

Permissions' requirements

- System wide ACL: affect all aspects of the UI
- Per-Application access for extended authentication
- Two user "levels"
    - Administrator
    - Developer
- Administrator has full access to everything
- Developer has access to read-only actions

MVC ACL integration, cont'd

- Zend\Permissions\Acl
    - Initialized with permissions' details from database
    - Initialization is performed during bootstrap
    - Information Tree is immutable, whatever the user that's logged in - caching in the future?
- MVC actions and ACL
    - Events manager to the rescue!
    - Call acl::isAllowed() before every action
        - Resource: Controller name
        - Privilege: Action name
        - User role from Identity object

```
$app->getEventManager()->attach('route',
array($this, 'allow'));
```

WebAPI output requirements

- Change output flow without affecting functionality
  - Controller actions should behave in the same way
  - Controller output should be uniform regardless of view script functionality
- Affect rendering behavior from different stages of execution
  - Different output formats (json, xml)
  - Different output view scripts
  - Different output functionality - view helpers

WebAPI output planning

```
public function initializeWebAPI(ManagerEvent $e) {
  $app = $e->getParam('application');
  if ($this->detectWebAPIRequest($app)) {
    $app->getEventManager()->
      attach('route', array($this,
'limitedWebapiOutput'));
    $app->getEventManager()->
      attach('dispatch', array($this,'applyWebAPIVersion'));
    $app->getEventManager()->
      attach('render', array($this,  'applyWebAPILayout'));
  }
}
```

WebAPI output, error handling

```
$events                   = $app->getEventManager();
/// Remove default error handling
...
$exceptionStrategy =
$locator->get('Zend\Mvc\View\Http\ExceptionStrategy');
$exceptionStrategy->detach($events);


....
/// Introduce webapi error handling
$exceptionStrategy =
$locator->get('WebAPI\Mvc\View\Http\ExceptionStrategy');
$events->attachAggregate($exceptionStrategy);
```

Dependencing your injection

Di, SM, Locator and other kinky things

D-Wha?!

Locator == Di == Service manager
It's all different names for the same thing

```
class IndexController extends ActionController
{
    public function indexAction() {
        $monitorUiModel = $this->getLocator()->get
('MonitorUi\Model\FilteredMapper');
    }
}
```

ZF2 Evolution: Using Di

Dependency injection

- Class name / instance name
- Parameters
    - Class names or actual values
    - constructor parameters
    - getter/setter
- Heavy on reflection
- Very strict behavior

ZF2 Di configuration

```
'definition' => array (
    'class' => array (
        'Zsd\DbConnector' => array(
            'methods' => array('factory' => array('required' => true, 'context' => array
      ('required' => true)))
        ),
        'PDO' => array('instantiator' => array('Zsd\DbConnector', 'factory'))
    )
),
'instance' => array(
    'zsdDbPDO' => array('parameters' => array('context' => 'zsd')),
    'zsdDbDriver' => array('parameters' => array('connection' => 'zsdDbPDO')),
    'zsdDbAdapter' => array('parameters' => array('driver' => 'zsdDbDriver')),
    'zsdServers_tg' => array('parameters' => array(
            'table' => 'ZSD_NODES',
            'adapter' => 'zsdDbAdapter',
        ))
)
```

ZF2 ServiceManager compared

```
array(
    'aliases' => array(
        'AuthAdapterSimple' => 'AuthAdapterDbTable',
        'AuthAdapterExtended' => 'AuthAdapterLdap',
    ),
    'invokables' => array(
        'index' => 'Application\Controller\IndexController',
        'Settings' => 'Application\Controller\SettingsController',
    ),
    'factories' => array(
        'Zend\Authentication\AuthenticationService' => function($sm) {
            $service = new AuthenticationService();
            $sessionConfig = new SessionConfig();
            $sessionConfig->setName('ZS6SESSID');
            $manager = new SessionManager($sessionConfig);
            $service->setStorage(new Session(null, null, $manager));
            $service->setMapper($sm->get('MonitorUi\Model\FilteredMapper'));
            return $service;
        }
    ))
```

Using Service Manager

- Service manager supplants Di
- Tells a human readable "story"
- Sectioned configuration
  - invokables
  - factories
  - abstractFactories
  - aliases
  - initializers
- Factories can be
  - Lambdas
  - method/function names
  - FactoryInterface implementing classes

ZS6 Di Evolution

Started with Di, moved to Service Manager

- Transition from Di to SM is difficult
- Similar systems, similar terms, different results and implementation
- Lots of functionality resided in Di
- Bridge the gap in onBootstrap:

```
$di = $this->serviceManager->get('Di');
$this->serviceManager->addAbstractFactory(
new DiAbstractServiceFactory($di));
```

- Factories caveat: does not lend to inheritance

Common initializers, the lack thereof

- Initializers are callables, usually for injecting objects into "awareness" interfaces
- Load initializers using ServiceManager::addInitializer

The problem:

- MVC native objects are produced by different ServiceManagers
- Only the "global" service manager is immediately available
- Consistency of SM behavior suggests Initializers should be shared ... they ain't

Common Initializers, solution

```
$initializers = array(
    function ($instance) use ($serviceManager) {
....
    },
    ....
);

$serviceLocators = array(
    $serviceManager,
    $serviceManager->get('ControllerLoader'),
    $serviceManager->get('ControllerPluginManager'),
    $serviceManager->get('ViewHelperManager'),
);

foreach ($serviceLocators as $serviceLocator) {
    foreach ($initializers as $initializer) {
        $serviceLocator->addInitializer($initializer);
    }
}
```

Identity Awareness

Are YOU aware?

Evolving a solution, requirements

- A user, in Zend Server 6 may be able to
  - see only a particular application
    - or a group of applications
  - Affect the application itself
  - Affect application-related information

"Affect" means filter

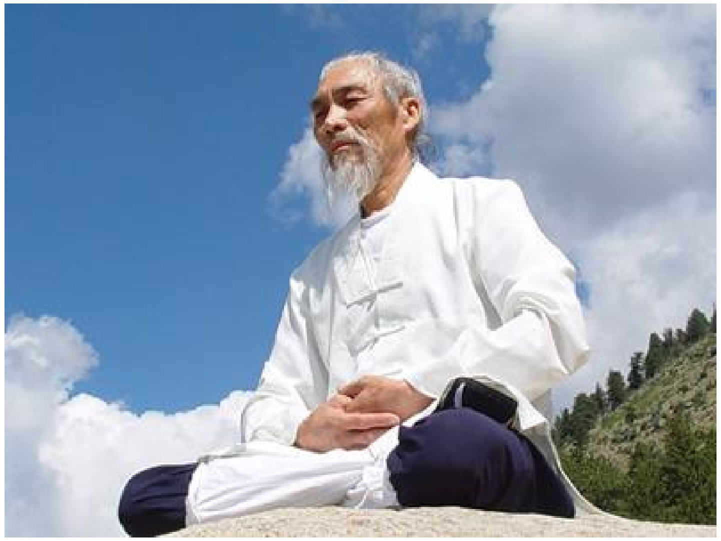
- The user may opt to filter by application id
- The application must enforce his permissions on the filter

Evolving a solution, complications

- A few different components
  - Monitor Events
  - Monitor Rules
  - JobQueue
  - Page Cache
  - Codetracing
- Each component has a different filter structure
- Each component handles applications' relations differently

Solutions, choices and ZF2

- Controller plugin
- Event driven
- Initializer based



The plugin solution

Create an Action Plugin class that accepts the full list of applications and the mapper's output

Problems:

- Diabolically complex
  - Difficult to extend and scale to other data types
- Different data structures and arbitrary differences between components
- Breaks MVC: requires the controller to be involved in business logic

The event driven solution

- Create an event listener class which modifies a predefined filter structure
- Cause the mapper to throw out an event before filtering
- The listener modifies the filter by consulting an available list of allowed applications
- Mapper continues using the filter object normally

The event driven solution

- Pros
  - Centralized functionality
  - Modular behavior - attach a listener or don't
  - Modular behavior 2 - adding more, future activities to the filter will be easier
- Cons
  - Filters' varying structure means either a complex listener
    or
  - Multiple listeners for multiple classes
  - Listener's behavior is difficult to change on the fly
    - either its hidden and hard to get at
      or
    - it's exposed and slowly becomes redundant

The initializer solution

Introduce the necessary functionality into the class that performs the operation

- Introduce a new class which can retrieve application ids from the identity object
    - Inject the user's Identity into this class
- Inject the new class into the data mapper
- Implement the identity filter internally in the mapper
- Continue normally

The initializer solution

Problems

- This is a complex solution
- Requires integration in each mapper
- It requires introducing new dependencies

However

- MVC separation is preserved
- Mapper encapsulation is preserved
- It is easy to extend in an environment with multiple authentication methods

```php
$this->trigger('complete', array('Thanks!'));
```

Thank you!

Thoughts, feedback: yonni.m@zend.com