Zend_Cache_Manager - Pádraic Brady

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Cache_Manager Component Proposal

Proposed Component Name	Zend_Cache_Manager
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Cache_Manager
Proposers	Pádraic Brady
Revision	1.0 - 25 January 2009/28 July 2009 (wiki revision: 8)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

As a springboard to other proposals, Zend_Cache_Manager is intended to be a centralised Manager which is capable of creating, adapting, storing and otherwise tinkering with caches from one single location. It's other advantage from an ease of use view is that it offers a collection of lazy loaded preconfigured caches to play with before you even write a configuration file.

Since it's a springboard proposal, its intention is to simplify the development of dependent classes like helpers and plugins which will delegate some of their core functionality to the Manager and focus themselves on linking the Manager enabled features to the application layer they are targeting.

2. References

- Building A Better Page Cache
- Controller Based Cache Management
- Tagging For Static File Caches

Source Code (In Development)
git repository

3. Component Requirements, Constraints, and Acceptance Criteria

- **MUST** implement a fully funtional Cache Manager storing cache objects and configuration templates
- **MUST** provide sufficient features to be delegated to by related helpers and plugins

- **WILL** host other functionality after refacoring of helpers/plugins

4. Dependencies on Other Framework Components

- Zend_Cache_*
- Zend_Config
- Zend_Exception

5. Theory of Operation

The Cache Manager is the core proposal in an effort to integrate simple to use caching into the MVC architecture of the Zend Framework through the use of helpers and plugins. Adding these elements individually would inevitably lead to coupling and code duplication, and the Cache Manager will offset these detrimental effects by offering a specific decoupled component dedicated to Cache Management which can be delegated to by other elements by proxy.

As a Cache Manager, the component has a simple set of facets:

1. Store all caches in a centralised location, but which are not statically accessible, to favour composition by external helpers/plugins.
2. Contain the creation logic for a collection of default caches usable across an application minimising setup time for developers.
3. Accept user configuration for all default caches to tailor the existing sensible options built into the Manager.
4. Allow the storage of Configuration Templates to support the lazy loading of requested caches based on these templates, and minimise setup code spreading at all application levels.
5. Adopt any duplicated roles common to future implemented plugins and helpers to minimise code duplication.

As you can see, the list is not that extensive. The Manager is a simple concept with simple use cases.

By storing caches internally, they can be accessed by helpers and plugins elsewhere and it likely will not rely on Zend_Registry since there's no real need to import another class dependency. Indeed, if you need a static instance - just store the Manager to the default registry.

The Manager will offer some very basic preconfigured caches to utilise in applications. These will follow the lowest common denominator approach, so they will favour filesystem storage over memory for example, since no PHP enviromental assumptions should be made.

The role of these caches is to ensure developers need not instantiate multiple small caches unnecessarily to support caching operations in helpers and plugins (which have predictable needs) - instead everything should work right away but allow developers to tweak configurations as needed between development, testing and production environments. These default caches will number approximately 3-4 and are all lazy loaded on demand assuming (where applicable) a file path to the cache directory of /cache (parallel to /applications). It is also assumed any sub-directories for these caches should be created by the Manager itself if not already existing.

The fourth role addresses the possibility of future refactoring to extract functionality from the proposed helpers/plugins which make Cache Manager access easier to minimise code duplication. It would require the completion of similar helpers/plugins before such refactoring oppotunities are identified (assuming they even materialise at all). To demonstrate where Zend_Cache_Manager can see a simple use, please see the related proposal on static page caching where the Manager is used to write a simple Action Helper and preDispatch() task.

6. Milestones / Tasks

- Milestone 1: Complete an initial version for comment
- Milestone 2: Complete final feature list incorporating feedback
- Milestone 3: Verify operation using Unit Tests
- Milestone 4: Complete documentation

7. Class Index

- Zend_Cache_Manager

8. Use Cases

UC-01: Storing a new cache

```
$cache = Zend_Cache::factory('Core','File');
$manager = new Zend_Cache_Manager;
$manager->setCache('Generic', $cache);
```

UC-02: Retrieving previously stored cache

```
$manager = new Zend_Cache_Manager;
$manager->getCache('Generic');
```

UC-03: storing a cache configuration template for lazy loading

```
$manager = new Zend_Cache_Manager;
$manager->setCacheTemplate('ExpiresHourly'
    array('Core', 'File', array('lifetime'=>3600))
);
```

UC-04: Fetch a cache, which lazy loads instantiation from a previous configuration template

```
$manager = new Zend_Cache_Manager;
$manager->getCache('ExpiresHourly');
```

UC-05: Check if a cache (or configuration template) exists

```
$manager = new Zend_Cache_Manager;
if ($manager->hasCache('myCustomCache')) {
    echo 'Found!';
}
```

UC-06: Ensure default caches cannot be overwritten once instantiated

```
$cache = Zend_Cache::factory('Core','File');
$manager = new Zend_Cache_Manager;
try {
    $manager->setCache('page', $cache);
} catch (Exception $e) {
    // What $e->getMessage() may state
    exit('Default caches cannot be overridden once instantiated; to reset cache type please configure
before utilising the cache');
}
```

9. Class Skeletons

Zend_Cache_Manager

```
<?php

/** Zend_Cache_Exception */
require_once 'Zend/Cache/Exception.php';

class Zend_Cache_Manager
{

    /**
```

```
     * Array of caches stored by the Cache Manager instance
     *
     * @var array
     */
    protected $_caches = array();

    /**
     * Array of ready made configuration templates for lazy
     * loading caches.
     *
     * @var array
     */
    protected $_configTemplates = array(
        // Null Cache (Enforce Null/Empty Values)
        'skeleton' => array(
            'frontend' => array(
                'name' => null,
                'options' => array()
            ),
            'backend' => array(
                'name' => null,
                'options' => array()
            )
        ),
        // Simplest Common Default
        'default' => array(
            'frontend' => array(
                'name' => 'Core',
                'options' => array(
                    'automatic_serialization' => true
                )
            ),
            'backend' => array(
                'name' => 'File',
                'options' => array(
                    'cache_dir' => '../cache'
                )
            )
        ),
        // Static Page HTML Cache
        'page' => array(
            'frontend' => array(
                'name' => 'Output',
                'options' => array(
                    'ignore_user_abort' => true
                )
            ),
            'backend' => array(
                'name' => 'Static',
                'options' => array(
                    'public_dir' => '../public',
                )
            )
        )
    );

    /**
     * Set a new cache for the Cache Manager to contain
     *
     * @param string $name
     * @param Zend_Cache_Core $cache
     * @return void
     */
    public function setCache($name, Zend_Cache_Core $cache){}

    /**
     * Check if the Cache Manager contains the named cache object, or a named
     * configuration template to lazy load the cache object
     *
     * @param string $name
```

```
     * @return bool
     */
    public function hasCache($name){}

    /**
     * Fetch the named cache object, or instantiate and return a cache object
     * using a named configuration template
     *
     * @param string $name
     * @return Zend_Cache_Core
     */
    public function getCache($name){}

    /**
     * Set a named configuration template from which a cache object can later
     * be lazy loaded
     *
     * @param string $name
     * @param array $config
     * @return void
     */
    public function setCacheTemplate($name, array $config){}

    /**
     * Check if the named configuration template
     *
     * @param string $name
     * @return bool
     */
    public function hasCacheTemplate($name){}

    /**
     * Get the named configuration template
     *
     * @param string $name
     * @return array
     */
    public function getCacheTemplate($name){}

    /**
     * Pass an array containing changes to be applied to a named configuration
     * template
     *
     * @param string $name
     * @param array $config
     * @return void
     */
    public function setTemplateConfig($name, array $config){}

    /**
     * Simple method to merge two configuration arrays
     *
     * @param array $current
     * @param array $config
     * @return array
     */
```

```
    protected function _mergeConfigs(array $current, array $config){}

  }
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>