Zend_Cache_Backend_MongoDb - Robert Allen

<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_Backend_MongoDb Component Proposal

Proposed Component Name	Zend_Cache_Backend_MongoDb
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Cache_Backend_MongoDb
Proposers	Robert Allen
Zend Liaison	TBD
Revision	1.0 - 1 March 2012: Initial Draft. (wiki revision: 7)

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

Zend_Cache_Backend_MongoDb is an extended Cache Backend supporting tags, automatic_cleaning, expire_read, infinite_lifetime and get_list functionality. By taking advantage of the speed, reliability and distributed features of MongoDb it is possible to employ advanced caching features in complex and fragmented environments.

2. References

- Zend_Cache_Backend_MongoDb
- MongoDb.org
- Mongo docs on php.net
- CappedCollections: Usage and Restrictions

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** employ MongoDb as a storage engine for cached records.

- This component **will** support access via Zend_Cache_Manager
- This component **will** allow explicit declaration of the MongoDb object.
- This component **will** include a factory method.
- This component **will** support tagging.
- This component **will** automatic cleaning.
- This component **will** support infinite lifetime.
- This component **will** support tag and ID listing.
- This component **will not** support priority.

4. Dependencies on Other Framework Components

- MongoDb pecl extension
- Zend_Cache_Backend_ExtendedInterface
- Zend_Cache_Backend

5. Theory of Operation

This adapter functions in a similar manner as the sqlite backend; however by employing MongoDb as the storage engine we are no longer restricted to the file sytem and gain the benefits of replication and high availability in clustered/load-balanced environments.

6. Milestones / Tasks

- Milestone 1: [DONE] Develop the cache backend class.
- Milestone 2: [DONE] Develop and pass unit tests.
- Milestone 3: Incorporate feedback and change sets from review.
- Milestone 4: Write Documentation.

7. Class Index

- Zend_Cache_Backend_MongoDb
- MongoDb
- MongoCollection

8. Use Cases

UC-01

```php
<?php
$frontendOptions = array(
    'lifetime' => 7200,
    'automatic_serialization' => true
);
$backendOptions = array(
    'database_name' => 'zend_cache',
    'collection'    => 'cache'
);
$cache = Zend_Cache::factory('Core', 'MongoDb', $frontendOptions, $backendOptions);
$cache->save($data, $id, array('tag1','acct_id:1234');
$data = $cache->load($id);
```

UC-02

```php
<?php
$cache = new Zend_Cache_Backend_MongoDb(
    array('database_name' => 'zend_cache',
    'database_urn' => 'mongodb://localhost:27717'
    'collection' => 'cache')
);
$cache->getTags();
$cache->getIds();
$cache->getIdsMatchingAnyTags();
$cache->getIdsMatchingTags(array('tag1','tag2'));
$cache->getIdsNotMatchingTags(array('tag1','tag2'));
$cache->getMetadatas($id);
```

UC-03

```php
<?php
$mongodb = new MongoDb();
$cache = new Zend_Cache_Backend_MongoDb(array('collection' => 'cache'));
$cache->setDatabase($mongodb);
$cache->load($id);
```

UC-04

```
resources.cachemanager.mongodb.frontend.name = Core
resources.cachemanager.mongodb.frontend.customFrontendNaming = false
resources.cachemanager.mongodb.frontend.options.lifetime = 7200
resources.cachemanager.mongodb.frontend.options.automatic_serialization = true
resources.cachemanager.mongodb.backend.name = MongoDb
resources.cachemanager.mongodb.backend.customBackendNaming = false
resources.cachemanager.mongodb.backend.options.database_name = "zend_cache"
resources.cachemanager.mongodb.backend.options.collection = "cache"
resources.cachemanager.mongodb.frontendBackendAutoload = false

<?php
if($bootstrap->hasResource('cachemanager')){
    $cache = $bootstrap->getResource('cachemanager')->getCache('mongodb');
    $cache->load($id);
}
```

9. Class Skeletons

```
<?php
/**
 * @category   Zend
 * @package    Zend_Cache
 * @subpackage Zend_Cache_Backend
 */


/**
 * @see Zend_Cache_Backend_Interface
 */
require_once 'Zend/Cache/Backend/ExtendedInterface.php';

/**
 * @see Zend_Cache_Backend
 */
require_once 'Zend/Cache/Backend.php';

/**
 * Notes:
 * Do not use a capped collection there are limitations that make it untennable
 * as a cache store.
 * @link http://www.mongodb.org/display/DOCS/Capped+Collections#CappedCollections-UsageandRestrictions
 *
 * @package    Zend_Cache
 * @subpackage Zend_Cache_Backend
 */
class Zend_Cache_Backend_MongoDb extends Zend_Cache_Backend implements
Zend_Cache_Backend_ExtendedInterface
{
    /**
     * @todo complete this parameters documentation
     * Available options
     *
     * ====> (array) database_name :
     *       Database name possessing the collection.
     * ====> (string) database_urn :
     *       URN of the MongoDb connections
     * ====> (string) collection :
     *     - The Collection used to store the cache items
     *
     *
     * @var array Available options
     */
    protected $_options = array(
        'database_name' => null,
        'database_urn' => 'mongodb://localhost:27017',
        'options' => array(),
        'collection' => null
    );
    /**
     *
     * @var MongoDb|null
     */
    protected $_database = null;

    /**
     * MongoCollection
     *
     * @var MongoCollection|null $_collection
     */
    private $_collection = null;

    /**
     * Constructor
```

```
     *
     * @param  array $options Associative array of options
     * @throws Zend_cache_Exception
     * @return void
     */
    public function __construct(array $options = array())
    {
    }
    /**
     * Fluent method allowing for the setting of the MongoDb instance
     * @param MongoDb $db
     * @return Zend_Cache_Backend_MongoDb
     */
    public function setDb(MongoDb $db)
    {
    }
    /**
     * @return Zend_Cache_Backend_MongoDb|null
     */
    public function getDb()
    {
    }
    /**
     *
     * @param MongoCollection $collection
     * @return Zend_Cache_Backend_MongoDb
     */
    public function setCollection(MongoCollection $collection)
    {
    }
    /**
     *
     * @return MongoCollection|null
     */
    public function getCollection()
    {
        return $this->_collection;
    }
    /**
     *
     * @see Zend_Cache_Backend_ExtendedInterface::getFillingPercentage()
     *
     * Return the filling percentage of the backend storage
     *
     * @return int integer between 0 and 100
     */
    public function getFillingPercentage()
    {
    }
    /**
     * Test if a cache is available for the given id and (if yes) return it (false else)
     *
     * @param  string  $id                     Cache id
     * @param  boolean $doNotTestCacheValidity If set to true, the cache validity won't be tested
     * @return string|false Cached datas
     */
    public function load($id, $doNotTestCacheValidity = false)
    {
    }

    /**
     * Test if a cache is available or not (for the given id)
     *
     * @param string $id Cache id
     * @return mixed|false (a cache is not available) or "last modified"
     * timestamp (int) of the available cache record
     */
    public function test($id)
    {
    }
```

```
    /**
     * Save some string datas into a cache record
     *
     * Note : $data is always "string" (serialization is done by the
     * core not by the backend)
     *
     * @param  string $data             Datas to cache
     * @param  string $id               Cache id
     * @param  array  $tags             Array of strings, the cache record will be tagged by each
string entry
     * @param  int    $specificLifetime If != false, set a specific lifetime for this cache record
(null => infinite lifetime)
     * @throws Zend_Cache_Exception
     * @return boolean True if no problem
     */
    public function save($data, $id, $tags = array(), $specificLifetime = false)
    {
    }

    /**
     * Remove a cache record
     *
     * @param  string $id Cache id
     * @return boolean True if no problem
     */
    public function remove($id)
    {
    }

    /**
     * Clean some cache records
     *
     * Available modes are :
     * Zend_Cache::CLEANING_MODE_ALL (default)    => remove all cache entries ($tags is not used)
     * Zend_Cache::CLEANING_MODE_OLD              => remove too old cache entries ($tags is not used)
     * Zend_Cache::CLEANING_MODE_MATCHING_TAG     => remove cache entries matching all given tags
     *                                               ($tags can be an array of strings or a single
string)
     * Zend_Cache::CLEANING_MODE_NOT_MATCHING_TAG => remove cache entries not {matching one of the
given tags}
     *                                               ($tags can be an array of strings or a single
string)
     * Zend_Cache::CLEANING_MODE_MATCHING_ANY_TAG => remove cache entries matching any given tags
     *                                               ($tags can be an array of strings or a single
string)
     *
     * @param  string $mode Clean mode
     * @param  array  $tags Array of tags
     * @return boolean True if no problem
     */
    public function clean($mode = Zend_Cache::CLEANING_MODE_ALL, $tags = array())
    {
    }

    /**
     * Return an array of stored cache ids
     *
     * @return array array of stored cache ids (string)
     */
    public function getIds()
    {
    }

    /**
     * Return an array of stored tags
     *
     * @return array array of stored tags (string)
     */
    public function getTags()
```

```
    {
    }

    /**
     * Return an array of stored cache ids which match given tags
     *
     * In case of multiple tags, a logical AND is made between tags
     *
     * @param array $tags array of tags
     * @return array array of matching cache ids (string)
     */
    public function getIdsMatchingTags($tags = array())
    {
    }

    /**
     * Return an array of stored cache ids which don't match given tags
     *
     * In case of multiple tags, a logical OR is made between tags
     *
     * @param array $tags array of tags
     * @return array array of not matching cache ids (string)
     */
    public function getIdsNotMatchingTags($tags = array())
    {
    }

    /**
     * Return an array of stored cache ids which match any given tags
     *
     * In case of multiple tags, a logical AND is made between tags
     *
     * @param array $tags array of tags
     * @return array array of any matching cache ids (string)
     */
    public function getIdsMatchingAnyTags($tags = array())
    {
    }

    /**
     * Return an array of metadatas for the given cache id
     *
     * The array must include these keys :
     * - expire : the expire timestamp
     * - tags : a string array of tags
     * - mtime : timestamp of last modification time
     *
     * @param string $id cache id
     * @return array array of metadatas (false if the cache id is not found)
     */
    public function getMetadatas($id)
    {
    }

    /**
     * Give (if possible) an extra lifetime to the given cache id
     *
     * @param string $id cache id
     * @param int $extraLifetime
     * @return boolean true if ok
     */
    public function touch($id, $extraLifetime)
    {
    }

    /**
     * Return an associative array of capabilities (booleans) of the backend
     *
     * The array must include these keys :
     * - automatic_cleaning (is automating cleaning necessary)
```

```
     * - tags (are tags supported)
     * - expired_read (is it possible to read expired cache records
     *                 (for doNotTestCacheValidity option for example))
     * - priority does the backend deal with priority when saving
     * - infinite_lifetime (is infinite lifetime can work with this backend)
     * - get_list (is it possible to get the list of cache ids and the complete list of tags)
     *
     * @return array associative of with capabilities
     */
    public function getCapabilities()
    {
        return array(
            'automatic_cleaning' => true,
            'tags' => true,
            'expired_read' => true,
            'priority' => false,
            'infinite_lifetime' => true,
            'get_list' => true
        );
    }

    /**
     * PUBLIC METHOD FOR UNIT TESTING ONLY !
     *
     * Force a cache record to expire
     *
     * @param string $id Cache id
     */
    public function ___expire($id)
    {
    }

    /**
     * Return the connection resource
     *
     * If we are not connected, the connection is made
     *
     * @throws Zend_Cache_Exception
     * @return MongoCollection
     */
    private function _getCollection()
    {
    }

    /**
     * Register a cache id with the given tag
     *
     * @param  string $id  Cache id
     * @param  string $tag Tag
     * @return boolean True if no problem
     */
    private function _registerTag($id, $tag) {
    }

    /**
     * Clean some cache records
     *
     * Available modes are :
     * Zend_Cache::CLEANING_MODE_ALL (default)    => remove all cache entries ($tags is not used)
     * Zend_Cache::CLEANING_MODE_OLD              => remove too old cache entries ($tags is not used)
     * Zend_Cache::CLEANING_MODE_MATCHING_TAG     => remove cache entries matching all given tags
     *                                               ($tags can be an array of strings or a single
string)
     * Zend_Cache::CLEANING_MODE_NOT_MATCHING_TAG => remove cache entries not {matching one of the
given tags}
     *                                               ($tags can be an array of strings or a single
string)
     * Zend_Cache::CLEANING_MODE_MATCHING_ANY_TAG => remove cache entries matching any given tags
     *                                               ($tags can be an array of strings or a single
string)
```

```
     *
     * @param  string $mode Clean mode
     * @param  array  $tags Array of tags
     * @return boolean True if no problem
     */
    private function _clean($mode = Zend_Cache::CLEANING_MODE_ALL, $tags = array())
    {
```

```
        }

    }
```

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