CHAPTER 10

Caching

Introduction to ORM Cache Types

Due to its nature and the way Doctrine 2 works, applications using Doctrine 2 naturally run a bit slower than others. However, with good caching strategies applied, this issue can be almost completely eliminated. Doctrine 2 ORM brings native support for three different types of caches: the "meta cache," the "query cache," and the "result cache." When setting up the entity manager, the different caches can be added to the entity manager's configuration. In addition, the cache instance might be used for custom values as well.

Caching Backends

Cached data can be stored in different so-called caching backends. Doctrine 2 supports multiple technologies to be used as caching backends:

- ApcCache (requires ext/apc)
- ArrayCache (in memory, lifetime of the request)
- FilesystemCache (not optimal for high concurrency)
- MemcacheCache (requires ext/memcache)
- MemcachedCache (requires ext/memcached)
- PhpFileCache (not optimal for high concurrency)
- RedisCache.php (requires ext/phpredis)
- WinCacheCache.php (requires ext/wincache)
- XcacheCache.php (requires ext/xcache)
- ZendDataCache.php (requires Zend Server Platform)

Based on the situation or technologies used, you can choose between the different technologies.

M. Romer, *PHP Persistence*, DOI 10.1007/978-1-4842-2559-2_10

Metadata Cache

The metadata cache holds the entity mapping data given as annotations or external XML or YAML files. Caching this data means that Doctrine 2 does not need to perform reflection or XML or YAML parsing for every single request, which saves a significant amount of processing time.

When setting up the entity manager, the cache can easily be configured:

```
<?php
// [..]
$paths = array(__DIR__ . "/../src/Entity/");
$isDevMode = false;

$dbParams = array(
        'driver' => 'pdo_mysql',
        'user' => 'root',
        'password' => '',
        'dbname' => 'app',
);

$config = Setup::createAnnotationMetadataConfiguration(
        $paths, $isDevMode
);

$config->setMetadataCacheImpl(
        new \Doctrine\Common\Cache\FilesystemCache('/tmp/doctrine2')
);

$em = EntityManager::create($dbParams, $config);
```

In the configuration shown above, we tell Doctrine 2 to cache metadata locally in the filesystem. We could use other caching backends here as well. The caching path needs to be given when using the `FilesystemCache`. Next time, when entities are processed, Doctrine 2 starts setting up a somewhat cryptic files and folders structure in the given caching path. Luckily, you don't need to care about it—Doctrine 2 takes care of all things caching. We only need to make sure that the caching path given is writable to Doctrine 2. When caching via Memcached or Redis, these services need to be up and running and accessible to Doctrine 2, as well. If not, an exception will be raised.

Query Cache

The query cache ensures that DQL statements need to be translated into SQL only once. This again speeds up a Doctrine 2 application significantly:

```
<?php
// [..]
$paths = array(__DIR__ . "/../src/Entity/");
```

```
4   $isDevMode = false;
5
6   $dbParams = array(
7           'driver' => 'pdo_mysql',
8           'user' => 'root',
9           'password' => '',
10          'dbname' => 'app',
11  );
12
13  $config = Setup::createAnnotationMetadataConfiguration($paths,
    $isDevMode);
14  $cachingBackend = new \Doctrine\Common\Cache\FilesystemCache('/tmp/
    doctrine2');
15  $config->setMetadataCacheImpl($cachingBackend);
16  $config->setQueryCacheImpl($cachingBackend);
17  $em = EntityManager::create($dbParams, $config);
```

In the configuration shown above, we first set up a general caching backend, which now powers both the metadata cache and the query cache.

Result Cache

Last but not least, there is the result cache. The use of a result cache prevents executing the same queries against the database again and again:

```
1   <?php
2   // [..]
3   $paths = array(__DIR__ . "/../src/Entity/");
4   $isDevMode = false;
5
6   $dbParams = array(
7           'driver' => 'pdo_mysql',
8           'user' => 'root',
9           'password' => '',
10          'dbname' => 'app',
11  );
12
13  $config = Setup::createAnnotationMetadataConfiguration($paths,
    $isDevMode);
14  $cachingBackend = new \Doctrine\Common\Cache\FilesystemCache('/tmp/
    doctrine2');
15  $config->setMetadataCacheImpl($cachingBackend);
16  $config->setQueryCacheImpl($cachingBackend);
17  $config->setResultCacheImpl($cachingBackend);
18  $em = EntityManager::create($dbParams, $config);
```

Via method `setResultCacheImpl()`, the result cache now is ready for action. In contrast to the two other caching types, you have to actively tell Doctrine 2 to cache results for a given query:

```
1   <?php
2
3   // [..]
4   $query = $em->createQuery($dqlString);
5   $query->useResultCache(true);
```

Now, the result is cached.

Summary

Since object-relational-mapping results in a noticeable runtime overhead, caching is essential for high speed. In fact, in production, caching is a must and should be setup right from the beginning.