Caché

Symfony cachea de forma automática muchos elementos para mejorar la velocidad de las respuestas:

- Los equivalentes en PHP de los templates de twig
- Los valores de los archivos de configuración
- Las traducciones

  …

Los elementos cacheados se guardan por defecto en el directorio *var/cache*.

Cada entorno de ejecución tiene su propio subdirectorio de cache.

El comando para instalar el componente Cache es:

> *composer require symfony/cache*

El comando cache/clear

El comando *cache:clear* de la consola de symfony borra todos los elementos de la cache, pero además realiza una puesta apunto. Es decir, vuelve a generar algunos elementos de la cache para que la siguiente petición no tenga que hacerlo.

Caché programada

Al margen de todos los elementos que cachea symfony, nosotros podemos utilizar el sistema de caché para cachear nuestros propios elementos.

A partir de la versión 3.3 symfony implementa el denominado “Simple Cache” o PSR-16. Es la que se describe a continuación.

El primer paso es crear un objeto *cache* a partir de una de las clases de cache que vienen con symfony.

```
use Symfony\Component\Cache\Simple\FilesystemCache;

$cache = new FilesystemCache();
```

Usando este objeto, se pueden crear, recuperar, actualizar y borrar elementos de
la caché.

```
// guardar un ítem en la caché
$cache->set('stats.num_products', 4711);

// o guardarlo con un ttl personalizado
// $cache->set('stats.num_products', 4711, 3600);
```

```php
// preguntar si existe un ítem determinado
if (!$cache->has('stats.num_products')) {
    // ...
}

// recuperar el valor almacenado en un ítem
$numProducts = $cache->get('stats.num_products');

// o especificar un varlor por defecto, si no existe
// $numProducts = $cache->get('stats.num_products', 100);

// elimiar un ítem de la caché
$cache->delete('stats.num_products');

// borrar todos los ítems de la caché
$cache->clear();
```

Icluso se pueden trabajar con varios elementos al mismo tiempo

```php
$cache->setMultiple(array(
    'stats.num_products' => 4711,
    'stats.num_users' => 1356,
));

$stats = $cache->getMultiple(array(
    'stats.num_products',
    'stats.num_users',
```

```
));

$cache->deleteMultiple(array(
    'stats.num_products',
    'stats.num_users',
));
```

Clases de caché disponibles

- ApcuCache
- ArrayCache
- ChainCache
- DoctrineCache
- FilesystemCache
- MemcachedCache
- NullCache
- PdoCache
- PhpArrayCache
- PhpFilesCache
- RedisCache
- TraceableCache

“More Advanced Caching” (PSR-6)

Antes de la versión 3.3 de symfony, la única implementación era la PSR-6 (“More Advanced Caching”). A partir de la versión 3.3 se puede utilizar cualquiera de las 2 implementaciones.

Se puede encontrar documentación sobre la implementación PSR-6 en la documentación oficial de symfony

Uso básico

Antes de empezar a cachear información, se debe crear la “cache pool” instanciando alguno de os adaptardores. Por ejemplo:

```
use Symfony\Component\Cache\Adapter\FilesystemAdapter;

$cache = new FilesystemAdapter();
```

Ahora ya podemos utilizar los métodos de la API de cacheo. Los métodos de esta API PSR-6 son muy sencillos y autoexplicativos:

```
// create a new item by trying to get it from the cache
$productsCount = $cache->getItem('stats.products_count');

// assign a value to the item and save it
```

```php
$productsCount->set(4711);
$cache->save($productsCount);

// retrieve the cache item
$productsCount = $cache->getItem('stats.products_count');
if (!$productsCount->isHit()) {
    // ... item does not exists in the cache
}
// retrieve the value stored by the item
$total = $productsCount->get();

// remove the cache item
$cache->deleteItem('stats.products_count');
```

Adapters de PSR-6 en Symfony

- APCu Cache Adapter
- Array Cache Adapter
- Chain Cache Adapter
- Doctrine Cache Adapter
- Filesystem Cache Adapter
- Memcached Cache Adapter
- PDO & Doctrine DBAL Cache Adapter
- Php Array Cache Adapter
- Php Files Cache Adapter
- Proxy Cache Adapter
- Redis Cache Adapter

Cache Contracts

En Symfony 5 (y disponible en symfony 4.4) se han introducido los **cache contracts**.

La idea es que la mayoría de veces que trabajamos con caché, tenemos un código como el siguiente:

```
$productsList = $cache->getItem('products_list');

if (!$productsList->isHit()) {
    // Si no está en la caché, lo volvemos a calcular
    $productsList = $servicio->getProductsList();
    $cache->setItem('products_list', $productsList);
}
```

Con los caché contracts, esa estructura getItem + if(!isHit()) + setItem está incorporada en un único método get().

```
$productsList = $cache->get('products_list', ???)
```

Ese $cache->get() internamente hará el if con el isHit() y en caso de fallo (el valor buscado no está en la caché), ejecutará el código que pasemos como callback en el segundo argumento, y hará el setItem() de lo que devolvamos en el return de la función de callback automáticamente.

De esta forma nos ahorramos varias líneas de código. El ejemplo anterior, con cache contracts, quedaría así:

```
$productsList = $cache->get('products_list', function (ItemInterface
$item) {
    $item->expiresAfter(3600);

    // Si no está en la caché, lo volvemos a calcular
    $data = $servicio->getProductsList();

    return $data;
)
```

El argumento de entrada a la función de callback nos permite establecer un tiempo de caducidad en segundos.

Si queremos eliminar algún dato cacheado, disponemos de la función delete().

```
$cache->delete('products_list');
```

https://symfony.com/doc/current/components/cache.html#cache-contracts