Laravel : Mise en cache

1.Introduction

La mise en cache est essentielle pour atteindre des performances élevées et une grande évolutivité. Mettre en œuvre la bonne stratégie de mise en cache dès la phase de développement est essentiel pour éviter le « lag » des API et les temps de chargement des pages trop lents.

Dans ce TP, nous allons explorer des stratégies pour mettre en œuvre et manipuler la mise en cache dans Laravel. Vous découvrirez comment fonctionne la mise en cache Laravel, plusieurs requêtes de mise en cache Laravel et comment vous pouvez gérer la mise en cache sur les applications Laravel.

Voyons aujourd'hui la mise en cache à la sauce Laravel. Je rappelle qu'un cache est destiné à accélérer la génération des pages en gardant en mémoire des informations. Le cas le plus classique est celui de requêtes sur une base de données. Si les données ne changent pas fréquemment il est plus efficace de mettre en cache leur résultat plutôt que d'aller chaque fois interroger la base. On peut aussi carrément mettre en cache des pages complètes.

2.CONFIGURATION

Le fichier de configuration du cache de votre application se trouve au chemin config/cache.php. Dans ce fichier, vous pouvez spécifier le pilote de cache que vous souhaitez utiliser par défaut dans votre application. Laravel prend en charge les backends de mise en cache les plus courants, tels que **Memcached**, **Redis**, **DynamoDB** et les bases de données relationnelles. En outre, un pilote de cache

basé sur les fichiers est disponible, tandis que les pilotes de cache array et "null" fournissent des backends de cache pratiques pour vos tests automatisés.

Le fichier de configuration du cache contient également diverses autres options, qui sont documentées dans le fichier, alors assurez-vous de lire ces options. Par défaut, Laravel est configuré pour utiliser le pilote de cache "file", qui stocke les objets sérialisés et mis en cache sur le système de fichiers du serveur. Pour les applications plus importantes, il est recommandé d'utiliser un pilote plus robuste tel que Memcached ou Redis. Vous pouvez même configurer plusieurs configurations de cache pour le même pilote.

Vous trouvez toutes les possibilités de configuration du cache. Vous voyez en premier la façon dont les informations sont stockées :

- APC
- array
- base de données
- fichier (valeur par défaut)
- Memcached
- Redis

Vous trouvez aussi la localisation du fichier (par défaut **storage/framework/cache**). Le nom de la base et de la table à utiliser dans le cas de stockage en base de données.

Quelle que soit la méthode choisie la classe Cache de Laravel offre les mêmes possibilités de base.

```
<?php

use Illuminate\Support\Str;

return [

    /*
    |--------------------------------------------------------------------------
    | Default Cache Store
    |--------------------------------------------------------------------------
    |
```

```
    | This option controls the default cache connection that gets used while
    | using this caching library. This connection is used when another is
    | not explicitly specified when executing a given caching function.
    |
    */

    'default' => env('CACHE_DRIVER', 'file'),

    /*
    |--------------------------------------------------------------------------
    | Cache Stores
    |--------------------------------------------------------------------------
    |
    | Here you may define all of the cache "stores" for your application as
    | well as their drivers. You may even define multiple stores for the
    | same cache driver to group types of items stored in your caches.
    |
    | Supported drivers: "apc", "array", "database", "file",
    |         "memcached", "redis", "dynamodb", "octane", "null"
    |
    */

    'stores' => [

        'apc' => [
            'driver' => 'apc',
        ],

        'array' => [
            'driver' => 'array',
            'serialize' => false,
        ],

        'database' => [
            'driver' => 'database',
            'table' => 'cache',
            'connection' => null,
            'lock_connection' => null,
        ],

        'file' => [
            'driver' => 'file',
            'path' => storage_path('framework/cache/data'),
        ],

        'memcached' => [
            'driver' => 'memcached',
            'persistent_id' => env('MEMCACHED_PERSISTENT_ID'),
            'sasl' => [
```

```
                env('MEMCACHED_USERNAME'),
                env('MEMCACHED_PASSWORD'),
            ],
            'options' => [
                // Memcached::OPT_CONNECT_TIMEOUT => 2000,
            ],
            'servers' => [
                [
                    'host' => env('MEMCACHED_HOST', '127.0.0.1'),
                    'port' => env('MEMCACHED_PORT', 11211),
                    'weight' => 100,
                ],
            ],
        ],

        'redis' => [
            'driver' => 'redis',
            'connection' => 'cache',
            'lock_connection' => 'default',
        ],

        'dynamodb' => [
            'driver' => 'dynamodb',
            'key' => env('AWS_ACCESS_KEY_ID'),
            'secret' => env('AWS_SECRET_ACCESS_KEY'),
            'region' => env('AWS_DEFAULT_REGION', 'us-east-1'),
            'table' => env('DYNAMODB_CACHE_TABLE', 'cache'),
            'endpoint' => env('DYNAMODB_ENDPOINT'),
        ],

        'octane' => [
            'driver' => 'octane',
        ],

    ],

    /*
    |--------------------------------------------------------------------------
    | Cache Key Prefix
    |--------------------------------------------------------------------------
    |
    | When utilizing the APC, database, memcached, Redis, or DynamoDB cache
    | stores there might be other applications using the same cache. For
    | that reason, you may prefix every cache key to avoid collisions.
    |
    */

    'prefix' => env('CACHE_PREFIX', Str::slug(env('APP_NAME', 'laravel'),
'_').'_cache_'),
```

```
];
```

3.Les commandes de base

a.Récupérer des éléments du cache

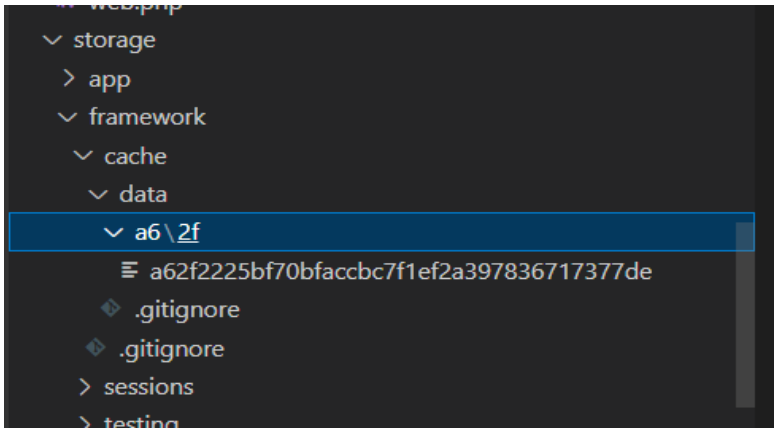
Pour obtenir une instance de cache, vous pouvez utiliser la façade **Cache**, qui est celle que nous utiliserons tout au long de ce TP.

```
public function index()
    {

        Cache::put('key', '1', 60);  //1 Minute
        $value = Cache::get('key','default');
        echo $value;

    }
```

La méthode **put** permet de mémoriser une information, le premier paramètre est la clé, le second la valeur et le troisième la durée de conservation de l'information.

La méthode **get** est l'inverse, elle permet de retrouver une information mémorisée à partir de sa clé.

Regardons maintenant le fichier créé :



Il est possible de conserver indéfiniment une information en utilisant la méthode **forever** :

```
Cache::forever('key', '1');
```

Et la méthode **forget** pour en supprimer une :

```
Cache::forget('key');
```

La méthode has peut être utilisée pour déterminer si un élément existe dans le cache. Cette méthode renvoie également false si l'élément existe mais que sa valeur est nulle :

```
if (Cache::has('key')) {
    // ...
}
```

Il peut arriver que vous souhaitiez récupérer un élément dans le cache, mais aussi stocker une valeur par défaut si l'élément demandé n'existe pas. Par exemple, vous pouvez souhaiter récupérer tous les utilisateurs dans le cache ou, s'ils n'existent pas, les récupérer dans la base de données et les ajouter au cache. Vous pouvez le faire en utilisant la méthode **Cache::remember** :

```
$value = Cache::remember('students',$seconds, function () {
      return DB::table('students')->get();
  });
```

Vous pouvez utiliser la méthode **rememberForever** pour récupérer un élément dans le cache ou le stocker définitivement s'il n'existe pas :

```
$value = Cache::rememberForever('students', function () {
      return DB::table('students')->get();
  });
```

Si vous devez récupérer un élément dans le cache et ensuite le supprimer, vous pouvez utiliser la méthode **pull**. Comme pour la méthode **get**, null sera renvoyé si l'élément n'existe pas dans le cache :

```
$value = Cache::pull('key');
```

- **Incrémentation/décrémentation des valeurs**

Les méthodes d'incrémentation et de décrémentation peuvent être utilisées pour ajuster la valeur d'éléments entiers dans le cache. Ces deux méthodes acceptent un deuxième argument facultatif indiquant le montant de l'incrémentation ou de la décrémentation de la valeur de l'élément :

```
Cache::increment('key');
Cache::increment('key', $amount);
Cache::decrement('key');
Cache::decrement('key', $amount);
```

b.Stockage d'éléments dans le cache

Vous pouvez utiliser la méthode **put** de la façade Cache pour stocker des éléments dans le cache :

```
Cache::put('key', 'value', $seconds = 10);
```

Si la durée de stockage n'est pas transmise à la méthode put, l'élément sera stocké indéfiniment :

```
Cache::put('key', 'value');
```

Vous pouvez vider tout le cache à l'aide de la méthode flush :

Cache::flush() ;

Code :

```
Route::get('/post',function(){
    $posts = Cache::remember('posts',60, function () {
        return DB::table('posts')->get();
    });


       //afficher la vue en passant un parametre
       return view('posts.home',compact('posts'));


});
Route::get('/flush',function(){
    Cache::flush();
});
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>My posts</h1>


    @forelse ($posts as $post )
   <p> {{$post->id}} </p>
    <p> {{$post->title}} </p>
    <p> {{$post->content}} </p>


    @empty
        <p>No posts</p>
    @endforelse
</body>
</html>
```