laravel 如何利用路由进行 http 缓存 (http_cache) 呢？而非路由配置信息缓存！

作者：ieras

路由缓存概念

路由缓存？也就是根据路由(url)进行的数据缓存策略，它可以帮助你快速针对接口增加缓存，再也不写数据缓存了～路由缓存，一步到位～

配置参数

本实现方案支持如下参数

- 1.路由,这不用说，基础的
- 2.header参数
- 3.get参数(也就是url里拼接的参数例如:/api/{uid}/info)
- 4.自定义参数，避免缓存key有冲突
- 5.缓存时长配置参数

下面说说实现方案吧！简单说就是 中间件

实现过程一

首先，我们先创建一个中间件，用来写主逻辑(连接缓存驱动，如redis，创建缓存、删除缓存等)，直看代码吧！

- /laravel/app/Http/Middleware/HttpCacheMiddleware.php

```
<?php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Database\Eloquent\Model;

class HttpCacheMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if (config('http_cache.enabled')) {
            $response = self::get($request);
            if (is_null($response)) {
                $response = $next($request);
                self::cache($request, $response);
            }
            return $response;
        } else {
            return $next($request);
        }
    }

    /**
     * Cache a request defined in http cache config file.
     *
     * @param Request $request
     * @param $response
     */
    private static function cache(Request $request, $response)
    {
        $uri = $request->route()->uri;
        $method = strtolower($request->method());
        if ($method != 'get' || !app('config')->has('http_cache.' . $uri)) {
            return;
        }

        list($routeParamFields, $inputFields, $headerFields, $cacheGroupKey, $cacheSeconds) =
            app('config')->get('http_cache.' . $uri);

        $routeParamKey = self::assemble($routeParamFields, $request->route()->parameters());
        $inputKey      = self::assemble($inputFields, $request->input());
        $headerKey     = self::headerAssemble($headerFields, $request->header());
        $cacheKey      = implode(':', [$routeParamKey, $inputKey, $headerKey]);

        cache()->tags('HttpCache:' . $cacheGroupKey)->put($cacheKey, $response, $cacheSecon
s);
    }

    /**
     * Get a request cache defined in http cache config file.
```

```
 *
 * @param Request $request
 * @return mixed|null
 */
private static function get(Request $request)
{
    $uri = $request->route()->uri;
    $method = strtolower($request->method());

    if ($method != 'get' || !app('config')->has('http_cache.' . $uri)) {
        return null;
    }

    list($routeParamFields, $inputFields, $headerFields, $cacheGroupKey, $cacheSeconds) =
        app('config')->get('http_cache.' . $uri);

    $routeParamKey = self::assemble($routeParamFields, $request->route()->parameters());
    $inputKey      = self::assemble($inputFields, $request->input());
    $headerKey     = self::headerAssemble($headerFields, $request->header());
    $cacheKey      = implode(':', [$routeParamKey, $inputKey, $headerKey]);

    return cache()->tags('HttpCache:' . $cacheGroupKey)->get($cacheKey);
}

/**
 * Assemble cache key components.
 *
 * @param array $fields
 * @param array $values
 * @return string
 */
private static function headerAssemble($fields, $values)
{
    $result = [];
    foreach ($fields as $field) {
        $field = strtolower($field);

        $value = array_get($values, $field);
        if (is_array($value)) {
            $value = head($value);
        }

        $result[$field] = strtolower($value);
    }
    return http_build_query($result);
}

/**
 * Assemble cache key components.
 *
 * @param array $fields
 * @param array $values
 * @return string
 */
```

```
    private static function assemble($fields, $values)
    {
        $result = [];
        foreach ($fields as $field) {
            $value = array_get($values, $field);
            if ($value instanceof Model) {
                $value = $value->id;
            }
            $result[$field] = $value;
        }
        return http_build_query($result);
    }

    /**
     * Forget a http cache defined in http cache config file.
     *
     * @param $uri
     */
    public static function forget($uri)
    {
        if (!app('config')->has('http_cache.' . $uri)) {
            return;
        }
        $cacheGroupKey = app('config')->get('http_cache.' . $uri . '.3');
        cache()->tags('HttpCache:' . $cacheGroupKey)->flush();
    }
}
```

实现过程二

代码中，你肯定发现了“config('http_cache.enabled')”，没错，我们还得搞一个配置文件

- /laravel/config/http_cache.php

```
<?php
return [
    'enabled' => env('ROUTE_CACHE_ENABLED', true),

    'sample_uri' => [           // only for GET method
        ['route_param_key1'],   // route param fields
        ['input1', 'input2'],   // input fields
        ['header1', 'header2'], // header fields
        'cache_group_key',      // cache key
        24 * 60,                // cache seconds
    ],


    'api/config'                => [
        [],//url路由里的动态参数
        ['appid', ],//传递参数
        [],//header参数
        'app_config',
        3600,
    ],
//    'api/info/index' => [
//        [],//url路由里的动态参数
//        ['type', ],//传递参数
//        [],//header参数
//        'infos',
//        5,
//    ],

    'api/info/conditions' => [
        [],//url路由里的动态参数
        ['show_all', 'level'],//传递参数
        [],//header参数
        'infos',
        60,
    ],
];
```

实现过程三

最后，就是如何使用这个中间件了，很简单，配置一下即可！

- /laravel/app/Http/Kernel.php

```
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    /**
     * 应用程序的全局HTTP中间件堆栈。
     *
     * 这些中间件在对您的应用程序的每次请求期间运行。
     *
     * @var array
     */
    protected $middleware = [
        \App\Http\Middleware\TrustProxies::class,
        \App\Http\Middleware\CheckForMaintenanceMode::class,
        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
        \App\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
        \App\Http\Middleware\Cors::class,
    ];

    /**
     * 应用程序的路由中间件组。
     *
     * @var array
     */
```

```php
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        // \Illuminate\Session\Middleware\AuthenticateSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:10000,1',
        'bindings',
        'http_cache',// 路由缓存生效的地方
    ],
];

/**
 * 应用程序的路由中间件。
 *
 * 这些中间件可以分配给组，也可以单独使用。
 *
 * @var array
 */
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    //'cros' => \App\Http\Middleware\Cors::class,
    'http_cache' => \App\Http\Middleware\HttpCacheMiddleware::class,//路由缓存配置
];

/**
 * 优先级排序的中间件列表。
 *
 * 这将强制非全局中间件始终处于给定顺序
 *
 * @var array
 */
protected $middlewarePriority = [
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    \App\Http\Middleware\Authenticate::class,
    \Illuminate\Session\Middleware\AuthenticateSession::class,
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
    \Illuminate\Auth\Middleware\Authorize::class,
];
```

```
}
```

整体就完成了～从此接口速度提升百倍，就靠它了～

理论上支持laravel全部版本，不同版本用户自行修改一下代码即可～

原文链接：laravel 如何利用路由进行 http 缓存 (http_cache) 呢？而非路由配置信息缓存！