

Data:

Nombre del módulo: Cache::SizeAwareSharedMemoryCache
Autor: DeWitt Clinton
URL: <http://search.cpan.org/~dclinton/Cache-Cache-1.04/lib/Cache/SizeAwareSharedMemoryCache.pm>
Autor del Doc: Marcelo A. Liberatto ([N3krodamus](#))
Grupo: [Perlmongers de Capital Federal \(CAFE PM\)](#)
Fecha: Febrero del 2006

Este módulo nos permite generar un pool de datos en memoria que se comparte entre procesos; para quienes hayan usado **IPC (Inter Process Communication)** les resultara familiar sobre todo si desde su consola linux ejecutan el comando " *ipcs* " luego de haber corrido el programa.

Imaginemos que tenemos un sistema de autenticación de usuarios para acceder a una sección de un site; podríamos tener una aplicación que autentique al usuario con un script principal Login.pl, el cual valida al usuario contra una base de datos y si la autenticación es satisfactoria nos devuelve un id de usuario que guardamos en memoria asociado con una cookie que nosotros generamos; que luego entregamos al browser del usuario.

Finalmente en el resto de las aplicaciones del sitio lo único que tienen que hacer es pedir la cookie y consultar la cache, si la cookie no existe en la cache entonces redireccionamos al usuario hacia la aplicación Login.pl.

Esta es una aplicación posible (y muy simplificada por cierto), si investigamos la documentación del módulo Cache::Cache, del cual se desprende entre otros SizeAwareSharedMemoryCache, veremos que existen diferentes módulos que utilizan esta misma interface y de los cuales uno de las mas interesantes es Cache::FileCache. (Si tienen ganas vean la doc).

Antes de empezar es bueno recordar que este módulo en particular requiere que tengamos instalado el IPC::Sharelite (no lo dice la doc pero si los warnings del perl :)), ya que utiliza las facilidades de este para administrar la memoria.

<Ayuda memoria>

Para instalar un módulo desde el cpan en nuestra maquina linux.

```
perl -MCPAN -e shell  
install IPC::Sharelite  
exit
```

Si todo anduvo bien tenemos el módulo de IPC instalado.

</Ayuda memoria>

Veamos un poco como usar este módulo.

Como primer paso cargamos el módulo a usar.

```
use Cache::SizeAwareSharedMemoryCache;
```

Luego inicializamos el objeto.

```
my $cache =  
new Cache::SizeAwareSharedMemoryCache(  
    {  
        'namespace' => 'MyMemory',  
        'max_size' => 10000,  
        'default_expires_in' => 600  
    }  
) or croak( "No se pudo inicializar MyMemory" );
```

Las opciones que podemos setear al iniciar el objeto son:

namespace (Valor por defecto: Default)

Este es el namespace asociado con esta cache, podemos tener tantas caches como queramos.

default_expires_in (Default: \$EXPIRES_NEVER)

Tiempo de vida de cada elemento de la cache, seteando esto nos ahorramos el problema de tener que expirar nosotros mismos cada elemento de la cache. El módulo tiene su propia rutina de limpieza que se encarga de eso. También tenemos la posibilidad de pedir que un objeto no caduque nunca.

auto_purge_interval

Aquí indicamos cada cuanto tiempo tiene que ejecutarse la rutina de limpieza, que se puede correr cada vez que hacemos una llamada de lectura o escritura, o puede correr cada cierto tiempo.

auto_purge_on_set (Boolean)

Con esta opción hacemos lo que se explico en el ítem anterior, cada vez que se hace una lectura o escritura se corren las rutinas de limpieza.

auto_purge_on_get (Boolean)

Aquí podemos explicitar que las rutinas de limpieza sean ejecutadas solo cuando hacemos un get (lectura de datos).

max_size (tamaño en bytes)

Especifica el tamaño máximo de la cache en bytes.

Bien, ya hemos inicializado el objeto cache asi que ahora vamos a ver como guardar datos en el mismo.

```
$cache->set($cache_key,$data,$expire);
```

```
$cache->set($cookie,$user_id,'10 minutes');
```

Breve explicación de lo que hacemos aquí.

La cache en memoria está organizada como un hash, mediante keys tenemos todos los valores ingresados dentro de un namespace. De esta manera cuando necesitamos leer o escribir un valor simplemente lo referenciamos como si fuese un hash, que en el ejemplo anterior es *\$cookie*.

Para el valor de tiempo de vida el módulo tiene dos constantes predefinidas, *\$EXPIRES_NEVER* y *\$EXPIRES_NOW*, por defecto si no se especifica valor se toma *\$EXPIRES_NEVER*. Las unidades de medida que reconoce este método son variadas, a saber :

Para segundos:

s, second, seconds, sec

Para minutos:

m, minute, minutes, min

Para horas:

h, hour, hours

Para días:

d, day, days

Para semanas:

w, week, weeks

Para meses:

M, month, months

Para años:

y, year, years

A continuación explicaré algunos de los métodos que podemos usar con el módulo.

Cuando queremos leer la información guardada simplemente lo hacemos así:

```
$user_id = $cache->get($cookie);
```

Si deseamos obtener un listado de todos los namespaces usados utilizamos el siguiente método:

```
$cache->get_namespaces();
```

Si lo que queremos es un listado de todas las keys de un namespace:

```
$cache->get_keys();
```

Para eliminar un key en particular lo podemos hacer de la siguiente manera:

```
$cache->remove_key($key);
```

Y para eliminar por completo un namespace:

```
$cache->Clear();
```

Esa es toda la magia....

Ahora vamos a ver un ejemplo práctico, ya que es la mejor manera de ver como funciona todo. Es una versión muy simple para consola del sistema de usuarios dado como ejemplo mas arriba, el script pide al usuario que elija entre dar de alta un usuario nuevo y listar los usuarios cargados. Si elegimos realizar un alta se nos piden 3 datos, email, nombre y color preferido; la cache indexa a los usuarios mediante el campo email. Se pueden realizar tantas altas como se desee pero el limite de memoria a usar es de 1000 Bytes y el tiempo de vida de los datos es de 600 segundos.

-----Copiar y pegar-----

```
#!/usr/bin/perl
```

```
use strict ;  
use Cache::SizeAwareSharedMemoryCache;
```

```
&main();
```

```
#####  
## Funcion main  
##
```

```

##
sub main
{
my $opcion;

print "Menu de opciones:
      1. Nuevo usuario
      2. Lista usuarios
      3. Eliminar cache
";

print "Opcion: ";
chop ($opcion = <STDIN>);

if ($opcion == 1)
{
    # alta usuario
    &alta_usr();
}

if ($opcion == 2)
{
    # listado
    &lista_usr();
}

if ($opcion == 3)
{
    # listado
    &elimina_cache();
}
exit;
}

#####
##    Alta de usuario
##
##
sub alta_usr
{
my $color;
my $nombre;
my $email;

print "Email:";
chop ($email = <STDIN>);

print "Nombre:";

```

```

chop ($nombre = <STDIN>);

print "Color preferido:";
chop ($color = <STDIN>);

my $cache = &init_cache();
$cache->set($email,join('#',$nombre,$color) );

print " Datos almacenados.

        Email: $email
        Nombre: $nombre
        Color preferido: $color \n\n";
exit;
}

#####
##      Lista usuarios
##
##
sub lista_usr
{
my $cache = &init_cache();
my @usrs = $cache->get_keys();
my $prefs;
my ($nombre,$color);

print "Usuarios almacenados";
foreach ( @usrs )
{
        $prefs = $cache->get($_);
        ($nombre,$color) = split(/^\#/, $prefs);

        print "
                Nombre: $nombre
                Color: $color
                Email: $_
                -----
        \n";
}
exit;
}

#####
##      Lista usuarios
##
##
sub elimina_cache

```

```

{
my $cache = &init_cache();
my @usrs = $cache->get_keys();
my $prefs;
my ($nombre,$color);

$cache->Clear();

print "Usuarios eliminados";
foreach ( @usrs )
{
    print " Email: $_
           -----\n";
}
exit;
}

```

```

#####
##   Inicializa la cache
## Devuelve: $cache
##
sub init_cache
{
my $cache =
new Cache::SharedMemoryCache(
    {
        'namespace' => 'MyCache', # Nombre de la cache
        'max_size' => 1000 ,
        'default_expires_in' => 600
    }
) or croak( "No se pudo inicializar" );
return($cache);
}

```

Fin.

Espero que les haya servido leer este minipaper, también les recomiendo que lean la documentación de los módulos ya que hay una gran variedad de implementaciones del sistema de cache y todos ellos tienen sus pros y sus contras.

N3krodamus
(n3krodamus-at-gmail-dot-com)