How to run long-running console commands

This page describes how to execute long-running console commands, to make sure they don't run out of memory. An example is a custom import command or the indexing command provided by the Solr Bundle.

Reducing memory usage

To avoid quickly running out of memory while executing such commands you should make sure to:

1. Always run in prod environment using: `--env=prod`
    a. *See Using environments page for further information on Symfony environments.*
    b. *See Logging & Debug configuration for some of different features enabled in development environments, which by design uses memory.*
2. Avoid Stash *(Persistence cache)* using to much memory in prod:
    a. If your system is running, or you need to use cache, then disable Stash InMemory cache as it does not limit the amount of items in cache and grows exponentially:

    config_prod.yml (snippet, not a full example for stash config)

    ```
    stash:
        caches:
            default:
                inMemory: false
    ```

    Also if you use FileSystem driver, make sure `memKeyLimit` is set to a low number, default should be 200 and can be lowered like this:

    config_prod.yml

    ```
    stash:
        caches:
            default:
                FileSystem:
                    memKeyLimit: 100
    ```

    b. If your setup is offline and cache is cold, there is no risk of stale cache and you can actually completely disable Stash cache. This will improve performance of import scripts:

    config_prod.yml (full example)

    ```
    stash:
        caches:
            default:
                drivers: [ Blackhole ]
                inMemory: false
    ```

3. For logging using monolog, if you use either the default `fingers_crossed`, or `buffer` handler, make sure to specify `buffer_size` to limit how large the buffer grows before it gets flushed:

config_prod.yml (snippet, not a full example for monolog config)

```
monolog:
    handlers:
        main:
            type: fingers_crossed
            buffer_size: 200
```

4. Run PHP without memory limits using: `php -d memory_limit=-1 app/console <command>`
5. Disable `xdebug` *(PHP extension to debug/profile php use)* when running the command, this will cause php to use much more memory.

Note: Memory will still grow
Even when everything is configured like described above, memory will grow for each iteration of indexing/inserting a content item with at least *1kb* per iteration after the initial first 100 rounds. This is expected behavior; to be able to handle more iterations you will have to do one or several of the following:

- Change the import/index script in question to use process forking to avoid the issue.
- Upgrade PHP: *newer versions of PHP are typically more memory-efficient.*
- Run the console command on a machine with more memory (RAM).

Process forking with Symfony

The recommended way to completely avoid "memory leaks" in PHP in the first place is to use processes, and for console scripts this is typically done using process forking which is quite easy to do with Symfony.

The things you will need to do:

1. Change your command so it supports taking slice parameters, like for instance a batch size and a child-offset parameter.
    a. *If defined, child-offset parameter denotes if a process is child, this could have been accomplished with two commands as well.*
    b. *If not defined, it is master process which will execute the processes until nothing is left to process.*
2. Change the command so that the master process takes care of forking child processes in slices.
    a. For execution in-order, you may look to our platform installer code used to fork out solr indexing after installation to avoid cache issues.
    b. For parallel execution of the slices, see Symfony doc for further instruction.

Related topics

- Using environments
- Symfony Process Component [symfony.com]
- Contribution Tutorials