django-phased Documentation

Release 0.6.1

Cody Soyland

September 20, 2013

CONTENTS

A simple two-phase template rendering application useful for caching of authenticated requests.

CHAPTER

ONE

HOW IT WORKS

This technique has been desribed by Adrian Holovaty in this blog post and previously by Honza Kral. The idea is to first render the template with certain blocks denoted as "phased," such that they will not be rendered, and will remain valid template code that can be rendered with a second pass.

The second pass fetches the partially-rendered template from the cache and performs a second render on it, using RequestContext to provide user-specific context to the template. This enables very fast generation of pages that have user-specific content, by bypassing the need to use the `CACHE_MIDDLEWARE_ANONYMOUS_ONLY` setting.

This implementation uses a secret delimiter that makes it safe against the possibility of template code injection vulnerabilities, as it only passes any given text through the template parser once. The phased blocks can also contain cached context.

CHAPTER

TWO

BASIC IMPLEMENTATION

django-phased contains a templatetag, `phased`, which defines blocks that are to be parsed during the second phase. A middleware class, PhasedRenderMiddleware, processes the response to render the parts that were skipped during the first rendering.

A special subclass of UpdateCacheMiddleware that drops the "Vary: Cookie" header from response when it updates the cache is also included, which, if used in place of the standard UpdateCacheMiddleware will prevent the cache middleware from varying the cache key based on cookies, thus enabling caching of pages in authenticated sessions.

2.1 Quickstart

django-phased is an implementation of a two-phase template rendering system to allow caching of pages that otherwise would be uncachable due to user-specific content that needs to be rendered (such as a signed-in notice in the corner of the page). This technique has been described in detail by Adrian Holovaty in this blog post.

2.1.1 Installation

To install django-phased, either check out the source from Github or install from PyPI:

- Check out django-phased from GitHub and run `python setup.py install` in the source checkout

  or

- Run `pip install django-phased`.

2.1.2 Setup

To make django-phased tags available to your templates, add `'phased'` to your `INSTALLED_APPS`.

You can either use `phased` via the *PhasedRenderMiddleware* middleware or the `phasedcache` template tag.

2.1.3 Usage

Middleware

Install the `PhasedRenderMiddleware` to enable second-phase rendering of templates.

If using Django's caching middleware, use `PatchedVaryUpdateCacheMiddleware` to bypass the `Vary: Cookie` behavior of that middleware.

A common setup for middleware classes would be this:

```
MIDDLEWARE_CLASSES = (
    'phased.middleware.PhasedRenderMiddleware',
    'phased.middleware.PatchedVaryUpdateCacheMiddleware',
    ...
    'django.middleware.cache.FetchFromCacheMiddleware',
)
```

See *Settings* for additional settings.

Template Tag

In order to use the `phasedcache` template tag you need to add `'django.core.context_processors.request'` to the `TEMPLATE_CONTEXT_PROCESSORS` settings variable and use `RequestContext` when you render your templates. See the Django docs on how to use RequestContext in your views.

The `phasedcache` template tag works exactly like Django's cache template tag except that it will run a second render pass using the `second_pass_render` function with value returned from the cache.

See `phasedcache` for details.

2.2 Settings

There are 2 settings to control behavior of django-phased:

- `PHASED_SECRET_DELIMITER`

  A custom delimiter to separate prerendered content from content that needs to be rendered with a second phase.

- `PHASED_KEEP_CONTEXT`

  If set to True, this setting will automatically capture and pickle context in phased blocks so the second pass will have access to context variables. Use with caution.

2.3 Middleware

django-phased provides two helpful middleware classes, `PhasedRenderMiddleware` and `PatchedVaryUpdateCacheMiddleware`.

class `phased.middleware.`**`PhasedRenderMiddleware`**

Performs a second-phase template rendering on the response and should be placed before the UpdateCacheMiddleware (or `PatchedVaryUpdateCacheMiddleware`) in the `MIDDLEWARE_CLASES` setting.

`process_response`(*request*, *response*)

If the content-type starts with `text/html` performs a second-phase render on response.content and updates the `Content-Length` header of the response to reflect the change in size after rendering.

class `phased.middleware.`**`PatchedVaryUpdateCacheMiddleware`**

If `Vary: Cookie` is set in the response object, Django's cache middleware will vary the cache key based on the value of the cookie.

This subclass of Django's UpdateCacheMiddleware is designed to cache without varying the cache key on cookie contents.

process_response(*request*, *response*)
This removes the `Vary: Cookie` header prior to running the standard Django `UpdateCacheMiddleware.process_response()` and adds the header back after caching so that in-browser caches are aware to vary the cache on cookies.

2.4 Template tag

`phased.templatetags.phased_tags.`**phased**(*parser*, *token*)
Template tag to denote a template section to render a second time via a middleware.

Usage:

```
{% load phased_tags %}
{% phased with [var1] [var2] .. %}
    .. some content to be rendered a second time ..
{% endphased %}
```

You can pass it a list of context variable names to automatically save those variables for the second pass rendering of the template, e.g.:

```
{% load phased_tags %}
{% phased with comment_count object %}
    There are {{ comment_count }} comments for "{{ object }}".
{% endphased %}
```

Alternatively you can also set the `PHASED_KEEP_CONTEXT` setting to `True` to automatically keep the whole context for each phased block.

Note: Lazy objects such as messages and csrf tokens aren't kept.

`phased.templatetags.phased_tags.`**phasedcache**(*parser*, *token*)
Taken from `django.templatetags.cache` and changed ending tag.

This will cache the contents of a template fragment for a given amount of time and do a second pass render on the contents.

Usage:

```
{% load phased_tags %}
{% phasedcache [expire_time] [fragment_name] %}
    .. some expensive processing ..
    {% phased %}
        .. some request specific stuff ..
    {% endphased %}
{% endphasedcache %}
```

This tag also supports varying by a list of arguments:

```
{% load phased_tags %}
{% phasedcache [expire_time] [fragment_name] [var1] [var2] .. %}
    .. some expensive processing ..
    {% phased %}
        .. some request specific stuff ..
    {% endphased %}
{% endphasedcache %}
```

Each unique set of arguments will result in a unique cache entry. The tag will take care that the phased tags are properly rendered.

It requires usage of `RequestContext` and `django.core.context_processors.request` to be in the `TEMPLATE_CONTEXT_PROCESSORS` setting.

`phased.templatetags.phased_tags.`**`parse`**(*parser*)
Parse to the end of a phased block. This is different than Parser.parse() in that it does not generate Node objects; it simply yields tokens.

2.5 Utils

`phased.utils.`**`second_pass_render`**(*request*, *content*)
Split on the secret delimiter and generate the token list by passing through text outside of phased blocks as single text tokens and tokenizing text inside the phased blocks. This ensures that nothing outside of the phased blocks is tokenized, thus eliminating the possibility of a template code injection vulnerability.

`phased.utils.`**`drop_vary_headers`**(*response*, *headers_to_drop*)
Remove an item from the "Vary" header of an `HttpResponse` object. If no items remain, delete the "Vary" header. This does the opposite effect of django.utils.cache.patch_vary_headers.

`phased.utils.`**`flatten_context`**(*context*, *remove_lazy=True*)
Creates a dictionary from a Context instance by traversing its dicts list. Can remove unwanted subjects from the result, e.g. lazy objects.

`phased.utils.`**`unpickle_context`**(*content*, *pattern=None*)
Unpickle the context from the given content string or return None.

`phased.utils.`**`pickle_context`**(*context*, *template=None*)
Pickle the given Context instance and do a few optimzations before.

2.6 Changelog

2.6.1 0.6.1 (2012-07-11)

- Moved docs to Read The Docs and extended the tutorial section:

  http://django-phased.readthedocs.org/

- Added `phasedcache` template tag for two-phase fragment caching.

2.6.2 0.6 (2012-06-29)

- **backwards-incompatible change**

  Starting in 0.6 django-phased now requires loading template tag libraries inside the `phased` block again. This was done to improve compatibility with Django 1.4 and future versions.

- Started to use Travis CI for testing:

  http://travis-ci.org/codysoyland/django-phased

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*