Cache Machine

Release 0.8

June 30, 2015

Contents

Cache Machine provides automatic caching and invalidation for Django models through the ORM. The code is hosted on github.

CHAPTER 1

Settings

Before we start, you'll have to update your `settings.py` to use one of the caching backends provided by Cache Machine. Django's built-in caching backends don't allow infinite cache timeouts, which are critical for doing invalidation (see below). Cache Machine extends the `locmem` and `memcached` backends provided by Django to enable indefinite caching when a timeout of `0` is passed. If you were already using one of these backends, you can probably go on using them just as you were. If you were caching things with a timeout of `0`, there will be problems with those entities now getting cached forever. You shouldn't have been doing that anyways.

For memcached:

```
CACHE_BACKEND = 'caching.backends.memcached://localhost:11211'
```

For locmem (only recommended for testing):

```
CACHE_BACKEND = 'caching.backends.locmem://'
```

Cache Machine will not work properly with the file or database cache backends.

If you want to set a prefix for all keys in Cache Machine, define `CACHE_PREFIX` in settings.py:

```
CACHE_PREFIX = 'weee:'
```

1.1 Django 1.3

With Django 1.3 or higher, you should use the new `CACHES` setting:

```
CACHES = {
    'default': {
        'BACKEND': 'caching.backends.memcached.MemcachedCache',
        'LOCATION': [
            'server-1:11211',
            'server-2:11211',
        ],
        'PREFIX': 'weee:',
    },
}
```

Note that we have to specify the class, not the module, for the `BACKEND` property, and that the `PREFIX` is optional. The `LOCATION` may be a string, instead of a list, if you only have one server.

If you require the default cache backend to be a different type of cache backend or want Cache Machine to use specific cache server options simply define a separate `cache_machine` entry for the `CACHES` setting, e.g.:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': 'server-1:11211',
    },
    'cache_machine': {
        'BACKEND': 'caching.backends.memcached.MemcachedCache',
        'LOCATION': [
            'server-1:11211',
            'server-2:11211',
        ],
        'PREFIX': 'weee:',
    },
}
```

Note: Cache Machine also supports the other memcache backend support by Django >= 1.3 based on **pylibmbc_**: `caching.backends.memcached.PyLibMCCache`.

1.2 COUNT queries

Calls to `QuerySet.count()` can be cached, but they cannot be reliably invalidated. Cache Machine would have to do a full select to figure out the object keys, which is probably much more data than you want to pull. I recommend a short cache timeout; long enough to avoid repetitive queries, but short enough that stale counts won't be a big deal.

```
CACHE_COUNT_TIMEOUT = 60  # seconds, not too long.
```

1.3 Empty querysets

By default cache machine will not cache empty querysets. To cache them:

```
CACHE_EMPTY_QUERYSETS = True
```

CHAPTER 2

Cache Manager

To enable caching for a model, add the `CachingManager` to that class and inherit from the `CachingMixin`. If you want related lookups (foreign keys) to hit the cache, `CachingManager` must be the default manager. If you have multiple managers that should be cached, return a *`CachingQuerySet`* from the other manager's `get_query_set` method instead of subclassing `CachingManager`, since that would hook up the post_save and post_delete signals multiple times.

Here's what a minimal cached model looks like:

```
from django.db import models

import caching.base

class Zomg(caching.base.CachingMixin, models.Model):
    val = models.IntegerField()

    objects = caching.base.CachingManager()
```

Whenever you run a query, `CachingQuerySet` will try to find that query in the cache. Queries are keyed by `{prefix}:{sql}`. If it's there, we return the cached result set and everyone is happy. If the query isn't in the cache, the normal codepath to run a database query is executed. As the objects in the result set are iterated over, they are added to a list that will get cached once iteration is done.

Note: Nothing will be cached if the QuerySet is not iterated through completely.

Caching is supported for normal `QuerySets` and for `django.db.models.Manager.raw()`. At this time, caching has not been implemented for `QuerySet.values` or `QuerySet.values_list`.

To support easy cache invalidation, we use "flush lists" to mark the cached queries an object belongs to. That way, all queries where an object was found will be invalidated when that object changes. Flush lists map an object key to a list of query keys.

When an object is saved or deleted, all query keys in its flush list will be deleted. In addition, the flush lists of its foreign key relations will be cleared. To avoid stale foreign key relations, any cached objects will be flushed when the object their foreign key points to is invalidated.

During cache invalidation, we explicitly set a None value instead of just deleting so we don't have any race condtions where:

- Thread 1 -> Cache miss, get object from DB
- Thread 2 -> Object saved, deleted from cache
- Thread 1 -> Store (stale) object fetched from DB in cache

The foundations of this module were derived from Mike Malone's django-caching.

CHAPTER 3

Manual Caching

Some things can be cached better outside of the ORM, so Cache Machine provides the function `caching.base.cached()` for caching arbitrary objects. Using this function gives you more control over what gets cached, and for how long, while abstracting a few repetitive elements.

CHAPTER 4

Template Caching

Cache Machine includes a Jinja2 extension to cache template fragments based on a queryset or cache-aware object. These fragments will get invalidated on using the same rules as `CachingQuerySets`.

First, add it to your template environment:

```
env = jinja2.Environment(extensions=['caching.ext.cache'])
```

Now wrap all your queryset looping with the `cache` tag.

```
{% cache objects %}  {# objects is a CachingQuerySet #}
  {% for obj in objects %}
    ...
  {% endfor %}
{% endcache %}
```

...and for caching by single objects:

```
{% cache object %}
  ...expensive processing...
{% endcache %}
```

The tag can take an optional timeout.

```
{% cache objects, 500 %}
```

If someone wants to write a template tag for Django templates, I'd love to add it.

CHAPTER 5

Classes that May Interest You

class caching.base.**CachingQuerySet**

Overrides the default QuerySet to fetch objects from cache before hitting the database.

C