Low-Level Cache API

Sometimes, caching an entire rendered page doesn’t gain you very much and is, in fact, inconvenient overkill.

Perhaps, for instance, your site includes a view whose results depend on several expensive queries, the results of which change at different intervals. In this case, it would not be ideal to use the full-page caching that the per-site or per-view cache strategies offer, because you wouldn’t want to cache the entire result (since some of the data changes often), but you’d still want to cache the results that rarely change.

For cases like this, Django exposes a low-level cache API. You can use this API to store objects in the cache with any level of granularity you like. You can cache any Python object that can be pickled safely: strings, dictionaries, lists of model objects, and so forth.

django.core.cache.cache

Low-Level Cache API

from django.core.cache import cache

cache.set(key, value, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to set cache.

Where,

key –It should be str.

value – It can be any pickleable Python object.

timeout – It is number of seconds the value should be stored in the cache. Timeout of None will cache the value forever. A timeout of 0 won't cache the value.

version – It is an int. You can set cache with same key but different version.

cache.get(key, default=None, version=None) – This method is used to get cache. If the key doesn't exists in the cache, it returns None.

Where,

default – This specifies which value to return if the object doesn't exist in the cache.

Low-Level Cache API

from django.core.cache import cache

cache.add(key, value, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to add a key only if it doesn't already exist. It takes the same parameters as set(), but it will not attempt to update the cache if the key specified is already present. If you need to know whether add() stored a value in the cache, you can check the return value. It will return True if the value was stored, False otherwise.

cache.get_or_set(key, default, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to get a key's value or set a value if the key isn't in the cache. It takes the same parameters as get() but the default is set as the new cache value for that key, rather than returned. You can also pass any callable as a default value.

cache.set_many(dict, timeout) – This method is used to set multiple values more efficiently, use set_many() to pass a dictionary of key-value pairs.

cache.get_many(keys, version=None) - There's also a get_many() interface that only hits the cache once. get_many() returns a dictionary with all the keys you asked for that actually exist in the cache (and haven't expired).

Low-Level Cache API

from django.core.cache import cache

cache.delete(key, version=None) – This method is used to delete keys explicitly to clear the cache for a particular object.

cache.delete_many(keys, version=None) – This method is used to clear a bunch of keys at once. It can take a list of keys to be cleared.

cache.clear() – This method is used to delete all the keys in the cache. Be careful with this; clear() will remove everything from the cache, not just the keys set by your application.

cache.touch(key, timeout=DEFAULT_TIMEOUT, version=None) – This method is used to set a new expiration for a key. touch() returns True if the key was successfully touched, False otherwise.

Low-Level Cache API

from django.core.cache import cache

cache.incr(key, delta=1, version=None)

cache.decr(key, delta=1, version=None)

You can also increment or decrement a key that already exists using the incr() or decr() methods, respectively. By default, the existing cache value will be incremented or decremented by 1. Other increment/decrement values can be specified by providing an argument to the increment/decrement call. A ValueError will be raised if you attempt to increment or decrement a nonexistent cache key.

Low-Level Cache API

from django.core.cache import cache

cache.close() - You can close the connection to your cache with close() if implemented by the cache backend.