Django Concurrency Documentation

Release 1.4

Stefano Apostolico

May 23, 2017

Contents

CHAPTER 1

Overview

django-concurrency is an optimistic locking library for Django Models

It prevents users from doing concurrent editing in Django both from UI and from a django command.

- easy to add to existing Models (just add *VersionField*)
- can be added with Django internal models (ie *auth.User* or *auth.Group*)
- handle http post and standard python code (ie. django management commands)
- complete test suite (*Test suite*)
- Admin integration. Handle *actions* and *list_editable* (solves issue #11313)
- can handle external updates (see *TriggerVersionField*)

CHAPTER 2

How it works

Overview

django-concurrency works adding a *concurrency.fields.VersionField* to each model, each time a record is saved the version number changes (the algorithm used depends on the implementation of *concurrency.fields.VersionField* used (see *Fields*).

Each update is converted in the following SQL clause like:

```
UPDATE mymodel SET version=NEW_VERSION, ... WHERE id = PK AND version = VERSION_NUMBER
```

CHAPTER 3

Table Of Contents

Install

Using `pip`:

```
pip install django-concurrency
```

Go to https://github.com/saxix/django-concurrency if you need to download a package or clone the repo.

django-concurrency does not need to be added into ‘‘INSTALLED_APPS‘ unless you want to run the tests or use the templatetags and/or admin integration

Test suite

django-concurrency comes with a set of tests that can simulate different scenarios

- basic versioned model
- inherited model
- inherited model from abstract model
- inherited model from external project model
- django User model
- models with custom save
- proxy models
- admin actions

How to run the tests

```
$ pip install tox
$ tox
```

Fields

VersionField

class concurrency.fields.**VersionField**(**args*, ***kwargs*)
Base class

IntegerVersionField

class concurrency.fields.**IntegerVersionField**(**args*, ***kwargs*)
Version Field that returns a "unique" version number for the record.

The version number is produced using time.time() * 1000000, to get the benefits of microsecond if the system clock provides them.

AutoIncVersionField

class concurrency.fields.**AutoIncVersionField**(**args*, ***kwargs*)
Version Field increment the revision number each commit

TriggerVersionField

class concurrency.fields.**TriggerVersionField**

This field use a database trigger to update the version field. Using this you can control external updates (ie using tools like phpMyAdmin, pgAdmin, SQLDeveloper). The trigger is automatically created during `syncdb()` or you can use the *triggers* management command.

Warning: Before **django-concurrency** 1.0 two triggers per field were created, if you are upgrading you must manually remove old triggers and recreate them using *triggers* management command

trigger_name

`TriggerVersionField.`**`trigger_name`**

Starting from 1.0 you can customize the name of the trigger created. Otherwise for each *TriggerVersionField* will be created two triggers named:

```
'concurrency_[DBTABLENAME]_[FIELDNAME]'
```

> **Warning:** Any name will be automatically prefixed with `concurrency_`

triggers management command

Helper command to work with triggers:

- `list` : list existing triggers for each database
- `drop` : drop exisitng triggers
- `create` : create required triggers

example

```
sax@: (concurrency) django-concurrency [feature/triggers*] $ ./demo/manage.py␣
↪triggers create
DATABASE             TRIGGERS
default              concurrency_concurrency_triggerconcurrentmodel_u
```

ConditionalVersionField

This field allow to configure which fields trigger the version increment so to limit the scope of the concurrency checks.

```
class User(models.Model):
    version = ConditionalVersionField()
    username = models.CharField(...)
    password = models.PasswordField(...)

    class ConcurrencyMeta:
        check_fields = ('username',)
```

ConcurrencyMiddleware

You can globally intercept *RecordModifiedError* adding *ConcurrencyMiddleware* to your `MIDDLEWARE_CLASSES`. Each time a *RecordModifiedError* is raised it goes up to the ConcurrencyMiddleware and the handler defined in *CONCURRENCY_HANDLER409* is invoked.

Example

`settings.py`

```
MIDDLEWARE_CLASSES=('django.middleware.common.CommonMiddleware',
                    'concurrency.middleware.ConcurrencyMiddleware',
                    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
                    'django.middleware.csrf.CsrfViewMiddleware',
                    'django.contrib.auth.middleware.AuthenticationMiddleware',
                    'django.contrib.messages.middleware.MessageMiddleware')

CONCURRENCY_HANDLER409 = 'demoproject.demoapp.views.conflict'
CONCURRENCY_POLICY = 2  # CONCURRENCY_LIST_EDITABLE_POLICY_ABORT_ALL
```

views.py

```
from diff_match_patch import diff_match_patch
from concurrency.views import ConflictResponse
from django.template import loader
from django.utils.safestring import mark_safe
from django.template.context import RequestContext

def get_diff(current, stored):
    data = []
    dmp = diff_match_patch()
    fields = current._meta.fields
    for field in fields:
        v1 = getattr(current, field.name, "")
        v2 = getattr(stored, field.name, "")
        diff = dmp.diff_main(unicode(v1), unicode(v2))
        dmp.diff_cleanupSemantic(diff)
        html = dmp.diff_prettyHtml(diff)
        html = mark_safe(html)
        data.append((field, v1, v2, html))
    return data

def conflict(request, target=None, template_name='409.html'):
    template = loader.get_template(template_name)
    try:
        saved = target.__class__._default_manager.get(pk=target.pk)
        diff = get_diff(target, saved)
    except target.__class__.DoesNotExists:
        saved = None
        diff = None

    ctx = RequestContext(request, {'target': target,
                                   'diff': diff,
                                   'saved': saved,
                                   'request_path': request.path})
    return ConflictResponse(template.render(ctx))
```

409.html

```
{% load concurrency %}
<table>
    <tr>
        <th>
            Field
        </th>
        <th>
            Current
        </th>
        <th>
            Stored
        </th>
```

```
        <th>
            Diff
        </th>

    </tr>
    <tr>
        {% for field, current, stored, entry in diff %}
            {% if not field.primary_key and not field|is_version %}
                <tr>
                    <td>
                        {{ field.verbose_name }}
                    </td>
                    <td>
                        {{ current }}
                    </td>
                    <td>
                        {{ stored }}
                    </td>
                    <td>
                        {{ entry }}
                    </td>
                </tr>
            {% endif %}
        {% endfor %}
    </tr>
</table>
```

If you want to use ConcurrentMiddleware in the admin and you are using *concurrency.admin.ConcurrentModelAdmin* remember to set your ModelAdmin to NOT use *concurrency.forms.ConcurrentForm*

```
from django import forms

class MyModelAdmin(ConcurrentModelAdmin):
    form = forms.ModelForm  # overrides default ConcurrentForm
```

Admin Integration

Handle `list_editable`

django-concurrency is able to handle conflicts in the admin's changelist view when `ModelAdmin.list_editable` is enabled. To enable this feature simply extend your ModelAdmin from *ConcurrentModelAdmin* or use *ConcurrencyListEditableMixin*

See also:

list_editable_policies

Check admin's action execution for concurrency

Extend your ModelAdmin with *ConcurrencyActionMixin* or use *ConcurrentModelAdmin*

Update existing actions templates to be managed by concurrency

You ca use the *identity* filter to pass both `pk` and `version` to your ModelAdmin. Each time you use `{{ obj.pk }}` simply change to `{{ obj|identity }}`. So in the `admin/delete_selected_confirmation.html` will have:

```
{% for obj in queryset %}
<input type="hidden" name="{{ action_checkbox_name }}" value="{{ obj|identity }}" />
{% endfor %}
```

API

Forms

ConcurrentForm

class concurrency.forms.**ConcurrentForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)

Simple wrapper to ModelForm that try to mitigate some concurrency error. Note that is always possible have a RecordModifiedError in model.save(). Statistically form.clean() should catch most of the concurrent editing, but is good to catch RecordModifiedError in the view too.

VersionWidget

class concurrency.forms.**VersionWidget**(*attrs=None*)

Widget that show the revision number using

Usually VersionField use *HiddenInput* as Widget to minimize the impact on the forms, in the Admin this produce a side effect to have the label *Version* without any value, you should use this widget to display the current revision number

Exceptions

VersionChangedError

class concurrency.exceptions.**VersionChangedError**(*message*, *code=None*, *params=None*)

class concurrency.exceptions.**RecordModifiedError**

RecordModifiedError

class concurrency.exceptions.**RecordModifiedError**(**args*, ***kwargs*)

InconsistencyError

> **Warning:** removed in 0.7

class concurrency.exceptions.**InconsistencyError**

VersionError

class concurrency.exceptions.**VersionError**(*message=None*, *code=None*, *params=None*, **args*, ***kwargs*)

Admin

ConcurrentModelAdmin

class concurrency.admin.**ConcurrentModelAdmin**(*model*, *admin_site*)

ConcurrencyActionMixin

class concurrency.admin.**ConcurrencyActionMixin**

ConcurrencyListEditableMixin

class concurrency.admin.**ConcurrencyListEditableMixin**

Middleware

class concurrency.middleware.**ConcurrencyMiddleware**

ConcurrencyMiddleware

See also:

ConcurrencyMiddleware

class concurrency.middleware.**ConcurrencyMiddleware**(*get_response=None*)
Intercept RecordModifiedError and invoke a callable defined in CONCURRECY_HANDLER409 passing the request and the object.

concurrency.views.conflict

concurrency.views.**conflict**(*request*, *target=None*, *template_name=u'409.html'*)
409 error handler.

Parameters

- **request** – Request
- **template_name** – *409.html*
- **target** – The model to save

Helpers

apply_concurrency_check()

Add concurrency check to existing classes.

Note: With Django 1.7 and the new migrations management, this utility does not work anymore. To add concurrency management to a external Model, you need to use a migration to add a *VersionField* to the desired Model.

Note: See `demo.auth_migrations` for a example how to add *`IntegerVersionField`* to `auth.Group`)

```
operations = [
    # add version to django.contrib.auth.Group
    migrations.AddField(
        model_name='Group',
        name='version',
        field=IntegerVersionField(help_text=b'Version', default=1),
    ),
]
```

and put in your settings.py

```
MIGRATION_MODULES = {
    ...
    ...
    'auth': '<new.migration.package>',
}
```

disable_concurrency()

Context manager to temporary disable concurrency checking.

Starting from version 0.9, *disable_concurrency* can disable both at Model level or instance level, depending on the passed object. Passing Model is useful in django commands, load data or fixtures, where instance should be used by default

Is now possible use *disable_concurrency* without any argument to disable concurrency on any Model. This features has been developed to be used in django commands

Does not raise an exception if a model not under concurrency management is passed as argument.

examples

```
@disable_concurrency()
def recover_view(self, request, version_id, extra_context=None):
    return super(ReversionConcurrentModelAdmin, self).recover_view(request,
                                                                   version_id,
                                                                   extra_context)
```

```
def test_recover():
    deleted_list = revisions.get_deleted(ReversionConcurrentModel)
```

```
    delete_version = deleted_list.get(id=5)

    with disable_concurrency(ReversionConcurrentModel):
        deleted_version.revert()
```

concurrency_disable_increment()

Context manager to temporary disable version increment. Concurrent save is still checked but no version increment is triggered, this creates 'shadow saves',

It accepts both a Model or an instance as target.

Templatetags

identity

concurrency.templatetags.concurrency.**identity**(*obj*)
returns a string representing "<pk>,<version>" of the passed object

version

concurrency.templatetags.concurrency.**version**(*obj*)
returns the value of the VersionField of the passed object

is_version

concurrency.templatetags.concurrency.**is_version**(*field*)
returns True if passed argument is a VersionField instance

Test Utilties

ConcurrencyTestMixin

class concurrency.utils.**ConcurrencyTestMixin**
Mixin class to test Models that use *VersionField*

this class offer a simple test scenario. Its purpose is to discover some conflict in the *save()* inheritance:

```
from concurrency.utils import ConcurrencyTestMixin
from myproject.models import MyModel

class MyModelTest(ConcurrencyTestMixin, TestCase):
    concurrency_model = TestModel0
    concurrency_kwargs = {'username': 'test'}
```

Signining

VersionField is 'displayed' in the Form using an `django.forms.HiddenInput` widget, anyway to be sure that the version is not tampered with, its value is *signed*. The default VersionSigner is `concurrency.forms.VersionFieldSigner` that simply extends `django.core.signing.Signer`. If you want change your Signer you can set *CONCURRENCY_FIELD_SIGNER* in your settings

mysigner.py

```
class DummySigner():
    """ Dummy signer that simply returns the raw version value. (Simply do␣
↪not sign it) """
    def sign(self, value):
        return smart_str(value)

    def unsign(self, signed_value):
        return smart_str(signed_value)
```

settings.py

```
CONCURRENCY_FIELD_SIGNER = "myapp.mysigner.DummySigner"
```

Settings

Available settings

Here's a full list of all available settings, in alphabetical order, and their default values.

Note: Each entry **MUST** have the prefix `CONCURRENCY_` when used in your settings.py

ENABLED

Default: `True`

enable/disable concurrency

CALLBACK

Default: `concurrency.views.callback`

Handler invoked when a conflict is raised. The default implementation simply raise *RecordModifiedError*

Can be used to display the two version of the record and let the user to force the update or merge the values.

FIELD_SIGNER

Default: `concurrency.forms.VersionFieldSigner`

Class used to sign the version numbers.

See also:

Signining

HANDLER409

Default: `concurrency.views.conflict`

Handler to intercept *`RecordModifiedError`* into *`ConcurrencyMiddleware`*. The default implementation (concurrency.views.conflict) renders `409.html` while passing into the context the object that is going to be saved (`target`)

See also:

ConcurrencyMiddleware

MANUAL_TRIGGERS

Default: `False`

If false do not automatically create triggers, you can create them using *triggers management command* management command or manually using your DB client.

POLICY

Default: `CONCURRENCY_LIST_EDITABLE_POLICY_SILENT`

IGNORE_DEFAULT

Default: `True`

Determines whether a default version number is ignored or used in a concurrency check. While this configuration defaults to True for backwards compatibility, this setting can cause omitted version numbers to pass concurrency checks. New implementations are recommended to set this to `False`.

Note: For security reasons, starting from version 1.5, default value will be `False`.

CONCURRENCY_LIST_EDITABLE_POLICY_SILENT

Used by admin's integrations to handle `list_editable` conflicts. Do not save conflicting records, continue and save all non-conflicting records, show a message to the user

CONCURRENCY_LIST_EDITABLE_POLICY_ABORT_ALL

Used by admin's integations to handle `list_editable`. Stop at the first conflict and raise *`RecordModifiedError`*. Note that if you want to use *`ConcurrencyMiddleware`* based conflict management you must set this flag.

See also:

Handle list_editable, *ConcurrencyMiddleware*

Cookbook

Unable to import data ?

Sometimes you need to temporary disable concurrency (ie during data imports)

Temporary disable per Model

```
from concurrency.api import disable_concurrency

with disable_concurrency(instance):
    Model.object
```

Add version management to new models

models.py

```
from concurrency.fields import IntegerVersionField

class ConcurrentModel( models.Model ):
    version = IntegerVersionField( )
```

tests.py

```
a = ConcurrentModel.objects.get(pk=1)
b = ConcurrentModel.objects.get(pk=1)
a.save()
b.save() # this will raise ``RecordModifiedError``
```

Add version management to Django and/or plugged in applications models

Concurrency can work even with existing models, anyway if you are adding concurrency management to an existing database remember to edit the database's tables:

your_app.models.py

```
from django.contrib.auth import User
from concurrency.api import apply_concurrency_check

apply_concurrency_check(User, 'version', IntegerVersionField)
```

If used with Django>=1.7 remebber to create a custom migration.

Manually handle concurrency

Use concurrency_check

```
from concurrency.api import concurrency_check


class AbstractModelWithCustomSave(models.Model):
    version = IntegerVersionField(db_column='cm_version_id', manually=True)


def save(self, *args, **kwargs):
    concurrency_check(self, *args, **kwargs)
    logger.debug(u'Saving %s "%s".' % (self._meta.verbose_name, self))
    super(SecurityConcurrencyBaseModel, self).save(*args, **kwargs)
```

Test Utilities

ConcurrencyTestMixin offer a very simple test function for your existing models

```
from concurrency.utils import ConcurrencyTestMixin
from myproject.models import MyModel

class MyModelTest(ConcurrencyTestMixin, TestCase):
    concurrency_model = TestModel0
    concurrency_kwargs = {'username': 'test'}
```

Recover deleted record with django-reversion

Recovering delete record with diango-reversion produce a `RecordModifeidError`. As both pk and version are present in the object, **django-concurrency** try to load the record (that does not exists) and this raises `RecordModifedError`. To avoid this simply:

```
class ConcurrencyVersionAdmin(reversionlib.VersionAdmin):
    def render_revision_form(self, request, obj, version, context, revert=False,␣
↪recover=False):
        with disable_concurrency(obj):
            return super(ConcurrencyVersionAdmin, self).render_revision_form(request,␣
↪obj, version, context, revert, recover)
```

or for (depending on django-reversion version)

```
class ConcurrencyVersionAdmin(reversionlib.VersionAdmin):

   @disable_concurrency()
   def recover_view(self, request, version_id, extra_context=None):
        return super(ReversionConcurrentModelAdmin, self).recover_view(request,
                                                               version_id,
                                                               extra_context)
```

Changelog

This section lists the biggest changes done on each release.

Release 1.4 (dev)

- Django 1.11 compatibility
- some minor support for Django 2.0

Release 1.3.2 (10 Sep 2016)

- fixes bug in ConditionalVersionField that produced 'maximum recursion error' when a model had a ManyToManyField with a field to same model (self-relation)

Release 1.3.1 (15 Jul 2016)

- just packaging

Release 1.3 (15 Jul 2016)

- drop support for Python < 3.3

- add support Django>=1.10
- change license
- fixes issue #36. (thanks claytondaley)
- new `IGNORE_DEFAULT` to ignore default version number

Release 1.2 (05 Apr 2016)

- better support for django 1.9 (`TemplateDoesNotExist` is now in `django.template.exceptions`
- improved eror message in `ConcurrencyListEditableMixin` issue #63 issue #64
- fixes issue #61. Error in `ConditionalVersionField` (thanks ticosax)
- fixes `skipif` test in pypy

Release 1.1 (13 Feb 2016)

- drop support for django<1.7
- add support for pypy
- new `concurrency.fields.ConditionalVersionField`
- new decorator `concurrency.api.concurrency_disable_increment`
- `concurrency.api.disable_concurrency` is now a noop if applied to a model not under concurrency management

Release 1.0.1

- fixes issue #56 "Can't upgrade django-concurrency to 1.0" (thanks oppianmatt).

Release 1.0

- **BACKWARD INCOMPATIBLE**:: dropped support for Django prior 1.6
- code clean
- fixes issue #54 "Incorrect default for IntegerVersionField" (thanks vmspike).
- fixes issue #53. updates Documentation
- *disable_concurrency()* can now disable concurrency in any model
- *disable_concurrency()* is now also a decorator
- **BACKWARD INCOMPATIBLE**:: removed custom backends. `TriggerVerionField` can be used with standard Django
- new way to create triggers (thanks Naddiseo)
- new trigger code
- new **:method:'TriggerVersionField.check'**.
- new *TriggerVersionField.trigger_name*.
- new `CONCURRECY_ENABLED` to fully disable concurrency

- new *CONCURRECY_MANUAL_TRIGGERS* to disable triggers auto creation fixes issue #41 (thanks Naddiseo)

Release 0.9

- Django 1.8 compatibility
- python 3.4 compatibility
- **BACKWARD INCOMPATIBLE :function:'disable_concurrency'** works differently if used with classes or instances
- better support for external Models (models that are part of plugged-in applications)
- fixes issue with TriggerVersionField and Proxy Models (thanx Richard Eames)

Release 0.8.1

- avoid to use concurrency when selecting all items (select_across)

Release 0.8

- Django 1.7 compatibility
- fixes typo in `delete_selected_confirmation.html` template
- python 3.2/3.3 compatibility

Release 0.7.1

- **backward compatibility updates. Do not check for concurrency if *0* is passed as version value** (ie. no value provided by the form)

Release 0.7

- new *concurrency.fields.TriggerVersionField*
- start using pytest
- moved tests outside main package
- new protocol see:ref:*protocols*
- it's now possible disable concurrency in Models that extends concurrency enabled models
- fixed issue #23 (thanks matklad)
- new `USE_SELECT_FOR_UPDATE`

Release 0.6.0

- new *disable_concurrency()* context manager
- added documentation for *concurrency.middleware.ConcurrencyMiddleware*
- **BACKWARD INCOMPATIBLE** Fixed typo: `CONCURRECY_SANITY_CHECK` now `CONCURRENCY_SANITY_CHECK`

- added `disable_sanity_check` context manager
- added configuration
- check admin actions for concurrent deletion
- added concurrency check for admin's *Handle list_editable*

Release 0.5.0

- python 3.x compatibility
- new *CONCURRENCY_FIELD_SIGNER*

Release 0.4.0

- **start deprecation of `concurrency.core.VersionChangedError`, `concurrency.core.RecordModifiedErro`**
  `concurrency.core.InconsistencyError`,moved in `concurrency.exceptions`
- start deprecation of `concurrency.core.apply_concurrency_check`, `concurrency.core.concurrency_check` moved in `concurrency.api`
- added `CONCURRECY_SANITY_CHECK` settings entry
- signing of version number to avoid tampering (*ConcurrentForm*)
- added *ConcurrencyTestMixin* to help test on concurrency managed models
- changed way to add concurrency to existing models (*apply_concurrency_check()*)
- fixed issue #4 (thanks FrankBie)
- removed RandomVersionField
- new *concurrency_check*
- added *ConcurrentForm* to mitigate some concurrency conflict
- select_for_update now executed with `nowait=True`
- removed some internal methods, to avoid unlikely but possible name clashes

Release 0.3.2

- fixed issue #3 (thanks pombredanne)
- fixed issue #1 (thanks mbrochh)

FAQ

- *South support ?*
- *How is managed* update_fields

South support ?

South support has been removed after version 1.0 when Django <1.6 support has been removed as well.

If needed add these lines to your `models.py`:

```
from south.modelsinspector import add_introspection_rules
add_introspection_rules([], ["^concurrency\.fields\.IntegerVersionField"])
```

How is managed *update_fields*

It is possible to use *save(update_fields=...)* parameter without interfree with the concurrency check algorithm

```
x1 = MyModel.objects.create(name='abc')
x2 = MyModel.objects.get(pk=x1.pk)

x1.save()
x2.save(update_fields=['username'])  # raise RecordModifiedError
```

anyway this will NOT raise any error

```
x1 = MyModel.objects.create(name='abc')
x2 = MyModel.objects.get(pk=x1.pk)

x1.save(update_fields=['username'])  # skip update version number
x2.save()  # saved
```

Note: *TriggerVersionField* will be always updated

CHAPTER 4

Links

- Project home page: https://github.com/saxix/django-concurrency
- Issue tracker: https://github.com/saxix/django-concurrency/issues?sort
- Download: http://pypi.python.org/pypi/django-concurrency/
- Docs: http://readthedocs.org/docs/django-concurrency/en/latest/

Index