Django Exceptions ¶

Django raises some of its own exceptions as well as standard Python exceptions.

Django Core Exceptions ¶

Django core exception classes are defined in `django.core.exceptions`.

`AppRegistryNotReady` ¶

exception `AppRegistryNotReady`[source] ¶

This exception is raised when attempting to use models before the app loading process, which initializes the ORM, is complete.

`ObjectDoesNotExist` ¶

exception `ObjectDoesNotExist`[source] ¶

The base class for `DoesNotExist` exceptions; a `try/except` for `ObjectDoesNotExist` will catch `DoesNotExist` exceptions for all models.

See `get()` for further information on `ObjectDoesNotExist` and `DoesNotExist`.

`EmptyResultSet` ¶

exception `EmptyResultSet`[source] ¶

`EmptyResultSet` may be raised during query generation if a query won't return any results. Most Django projects won't encounter this exception, but it might be useful for implementing custom lookups and expressions.

Changed in Django 1.11:
In older versions, it's only importable from `django.db.models.sql`.

`FieldDoesNotExist` ¶

exception `FieldDoesNotExist`[source] ¶

The `FieldDoesNotExist` exception is raised by a model's `_meta.get_field()` method when the requested field does not exist on the model or on the model's parents.

`MultipleObjectsReturned` ¶

exception `MultipleObjectsReturned`[source] ¶

The `MultipleObjectsReturned` exception is raised by a query if only one object is expected, but multiple objects are returned. A base version of this exception is provided in `django.core.exceptions`; each model class contains a subclassed version that can be used to identify the specific object type that has returned multiple objects.

Contents

SuspiciousOperation ¶

exception `SuspiciousOperation`[source] ¶

The **`SuspiciousOperation`** exception is raised when a user has performed an operation that should be considered suspicious from a security perspective, such as tampering with a session cookie. Subclasses of **`SuspiciousOperation`** include:

- `DisallowedHost`
- `DisallowedModelAdminLookup`
- `DisallowedModelAdminToField`
- `DisallowedRedirect`
- `InvalidSessionKey`
- `RequestDataTooBig`
- `SuspiciousFileOperation`
- `SuspiciousMultipartForm`
- `SuspiciousSession`
- `TooManyFieldsSent`

If a **`SuspiciousOperation`** exception reaches the WSGI handler level it is logged at the **`Error`** level and results in a **`HttpResponseBadRequest`**. See the logging documentation for more information.

PermissionDenied ¶

exception `PermissionDenied`[source] ¶

The **`PermissionDenied`** exception is raised when a user does not have permission to perform the action requested.

ViewDoesNotExist ¶

exception `ViewDoesNotExist`[source] ¶

The **`ViewDoesNotExist`** exception is raised by **`django.urls`** when a requested view does not exist.

MiddlewareNotUsed ¶

exception `MiddlewareNotUsed`[source] ¶

The **`MiddlewareNotUsed`** exception is raised when a middleware is not used in the server configuration.

ImproperlyConfigured ¶

exception `ImproperlyConfigured`[source] ¶

The **`ImproperlyConfigured`** exception is raised when Django is somehow improperly configured – for example, if a value in **`settings.py`** is incorrect or unparseable.

FieldError ¶

exception `FieldError`[source] ¶

You are here:

Getting help

FAQ
Try the FAQ — it's got answers to many common questions.

Index, Module Index, or Table of Contents
Handy when looking for specific information.

django-users mailing list
Search for information in the archives of the django-users mailing list, or post a question.

#django IRC channel
Ask a question in the #django IRC channel, or search the IRC logs to see if it's been asked before.

Django Discord Server
Join the Django Discord Community.

Official Django Forum
Join the community on the Django Forum.

Ticket tracker
Report bugs with Django or Django documentation in our ticket tracker.

Download:

Offline (Django 1.11): HTML | PDF | ePub
Provided by Read the Docs.

Getting Help

Language: **en**

Documentation version: **1.11**

- A field in a model clashes with a field of the same name from an abstract base class
- An infinite loop is caused by ordering
- A keyword cannot be parsed from the filter parameters
- A field cannot be determined from a keyword in the query parameters
- A join is not permitted on the specified field
- A field name is invalid
- A query contains invalid order_by arguments

`ValidationError` ¶

exception `ValidationError`[source] ¶

The `ValidationError` exception is raised when data fails form or model field validation. For more information about validation, see Form and Field Validation, Model Field Validation and the Validator Reference.

`NON_FIELD_ERRORS` ¶

`NON_FIELD_ERRORS` ¶

`ValidationError`s that don't belong to a particular field in a form or model are classified as `NON_FIELD_ERRORS`. This constant is used as a key in dictionaries that otherwise map fields to their respective list of errors.

URL Resolver exceptions ¶

URL Resolver exceptions are defined in `django.urls`.

Deprecated since version 1.10:
In older versions, these exceptions are located in `django.core.urlresolvers`. Importing from the old location will continue to work until Django 2.0.

`Resolver404` ¶

exception `Resolver404`[source] ¶

The `Resolver404` exception is raised by `resolve()` if the path passed to `resolve()` doesn't map to a view. It's a subclass of `django.http.Http404`.

`NoReverseMatch` ¶

exception `NoReverseMatch`[source] ¶

The `NoReverseMatch` exception is raised by `django.urls` when a matching URL in your URLconf cannot be identified based on the parameters supplied.

Database Exceptions ¶

Database exceptions may be imported from `django.db`.

Django wraps the standard database exceptions so that your Django code has a guaranteed common implementation of these classes.

exception `Error`[source] ¶

exception `DatabaseError`[source] ¶

exception `DataError`[source] ¶

exception `OperationalError`[source] ¶

exception `IntegrityError`[source] ¶

exception `InternalError`[source] ¶

exception `ProgrammingError`[source] ¶

exception `NotSupportedError`[source] ¶

The Django wrappers for database exceptions behave exactly the same as the underlying database exceptions. See **PEP 249**, the Python Database API Specification v2.0, for further information.

As per **PEP 3134**, a `__cause__` attribute is set with the original (underlying) database exception, allowing access to any additional information provided. (Note that this attribute is available under both Python 2 and Python 3, although **PEP 3134** normally only applies to Python 3. To avoid unexpected differences with Python 3, Django will also ensure that the exception made available via `__cause__` has a usable `__traceback__` attribute.)

Changed in Django 1.10:

The `__traceback__` attribute described above was added.

exception `models.ProtectedError` ¶

Raised to prevent deletion of referenced objects when using `django.db.models.PROTECT`. `models.ProtectedError` is a subclass of `IntegrityError`.

Http Exceptions ¶

Http exceptions may be imported from `django.http`.

`UnreadablePostError` ¶

exception `UnreadablePostError`[source] ¶

`UnreadablePostError` is raised when a user cancels an upload.

Transaction Exceptions ¶

Transaction exceptions are defined in `django.db.transaction`.

`TransactionManagementError` ¶

exception `TransactionManagementError`[source] ¶

`TransactionManagementError` is raised for any and all problems related to database transactions.

Testing Framework Exceptions ¶

Exceptions provided by the `django.test` package.

`RedirectCycleError` ¶

Getting Help

Language: **en**

Documentation version: **1.11**

`RedirectCycleError` is raised when the test client detects a loop or an overly long chain of redirects.

Python Exceptions ¶

Django raises built-in Python exceptions when appropriate as well. See the Python documentation for further information on the Built-in Exceptions.

‹ `django-admin` and `manage.py`   File handling ›

Getting Help
Language: **en**
Documentation version: **1.11**