2018.4.23

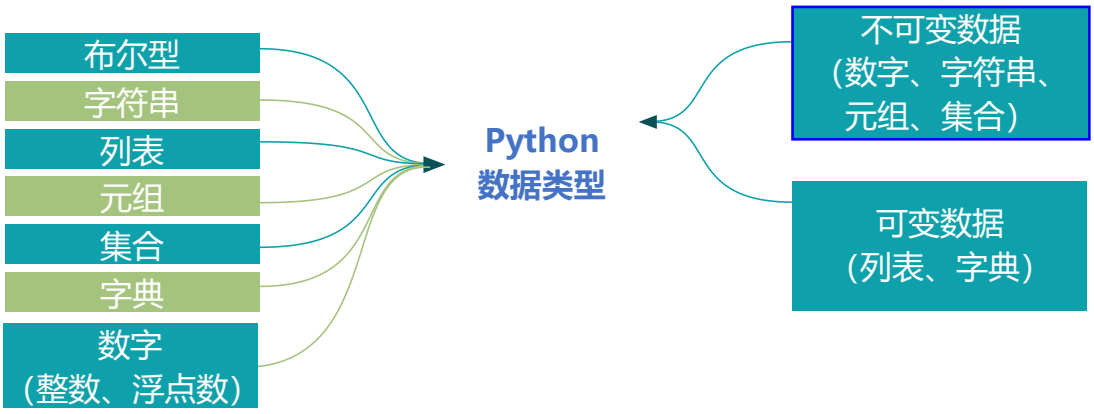
金融软件工程·作业

学号：161220095

姓名：牛皓玥

一、Python 基本成分

- 数据



- 运算成分

Python 语言支持基本运算。并且在官方文档中有内容表示，Python 中的一些比较方式更贴近我们的数学思维，也是与其他语言的不同之处。不过其他语言中没有这种语法。

4.3. Comparisons

There are eight comparison operations in Python. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, `x < y <= z` is equivalent to `x < y and y <= z`, except that *y* is evaluated only once (but in both cases *z* is not evaluated at all when `x < y` is found to be false).

以下是 Python 运算符的相关汇总

➢ 算术运算符

运算符	描述
`+ - * / % ** //`	加减乘除，模运算、指数运算、地板除

➢ 比较（关系）运算符

运算符	描述
`== != > < >= <=`	相等、不等、大于、小于、大于等于、小于等于

- 赋值运算符

运算符	描述
`= += -= *= /= %= **= //=`	基本操作都可以进行相应的赋值

- 逻辑运算符

运算符	描述
`and or not`	如果两个操作数都为真，则条件成立。
`or`	如果两个操作数中的任何一个非零，则条件成为真。
`not`	用于反转操作数的逻辑状态。

- 按位运算符

运算符	描述
`& | ^ ~ << >>`	按位与、按位或、按位亦或、按位取反、二进制左（右）移

- 成员运算符

运算符	描述
`in`	如果在指定的序列中找到一个变量的值，则返回 `true`，否则返回 `false`。
`not in`	如果在指定序列中找不到变量的值，则返回 `true`，否则返回 `false`。

- 身份运算符

运算符	描述
`is`	如果运算符任一侧的变量指向相同的对象，则返回 `True`，否则返回 `False`。
`is not`	如果运算符任一侧的变量指向相同的对象，则返回 `True`，否则返回 `False`。

- 控制成分

  包括基本的控制结构：顺序、条件和循环结构

- 传输成分

  输入：

  raw_input():获取输入后，返回一个 String 类型。

  input(): 支持表达式（此时只返回结果）、数字、字符串

  区别：input( )可以获取任何形式的输入并返回相应的不同类型，而 raw_input( )只能返回 String 类型对象。input( )本质上还是由 raw_input( )输入之后，再调用 eval( )来最终得到 input( )的结果。

  输出：print()

二、Python 语言特性

- 动态强类型：

  动态类型语言：在运行期进行类型检查的语言，也就是在编写代码的时候可以不指定变量的数据类型。

  强类型语言：一个变量不经过强制转换，它永远是这个数据类型，不允许隐式的类型转换。

- 优缺点分析

  优点——

(1) 适合阅读。Python 的伪代码本质使你能够专注于解决问题而不是去搞明白语言本身。

(2) 易学。python 虽然是用 c 语言写的，但是它摈弃了 c 中非常复杂的指针，简化了 python 的语法。

(3) Python 是 FLOSS（自由/开放源码软件）之一。可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。

(4) 可移植性————由于它的开源本质，Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。

(5) 不再需要担心如何编译程序，如何确保连接转载正确的库等等，程序更加易于移植。

(6) Python 既支持面向过程的函数编程也支持面向对象的抽象编程。

(7) 可扩展性和可嵌入性。若需要一段关键代码运行得更快或者希望某些算法不公开，可以把部分程序用 C 或 C++编写，然后在 Python 程序中使用它们。也可以把 Python 嵌入 C/C++程序，从而向程序用户提供脚本功能。

(8) 丰富的库。Python 标准库很庞大，并且除了标准库以外，还有许多其他高质量的库。

(9) 规范的代码。Python 采用强制缩进的方式使得代码具有极佳的可读性。

  缺点——

(1) 运行速度。不过对于用户根本感觉不出来这种速度的差异。

(2) python 的开源性使 Python 语言不能加密。

(3) 构架选择太多（没有像 C#这样的官方.net 构架，也没有像 ruby 由于历史较短，构架开发的相对集中。Ruby on Rails 构架开发中小型 web 程序天下无敌）。

三、代码分析

程序名：Django 源码/db/models/options.py

来源：https://github.com/django/django/blob/master/django/db/models/options.py

代码行：828 lines (734 sloc)

程序语言：Python

注释来源	内容
36 DEFAULT_NAMES = ()	（解释版本更新信息） # For backwards compatibility with Django 1.11. RemovedInDjango30Warning
42-46 def normalize_together(option_together):	（序言性） """option_together can be either a tuple of tuples, or a single tuple of two strings. Normalize it to a tuple of tuples, so that calling code can uniformly expect that."""
55 def normalize_together(option_together):	（解释操作） # Normalize everything to tuples
58-59 def normalize_together(option_together):	（解释分支情况） # If the value of option_together isn't valid, return it verbatim; this will be picked up by the check framework later.
110-114 def __init__(self, meta, app_label=None): self.proxy_for_model = None	（解释变量） # For any class that is a proxy (including automatically created classes for deferred object loading), proxy_for_model tells us which class this model is proxying. Note that proxy_for_model can create a chain of proxy models. For non-proxy models, the variable is always None.
116-118 def __init__(self, meta, app_label=None): self.concrete_model = None	（解释变量） # For any non-abstract class, the concrete class is the model in the end of the proxy_for_model chain. In particular, for concrete models, the concrete_model is always the class itself.
124-125 def __init__(self, meta, app_label=None): self.related_fkey_lookups = []	（解释变量） # List of all lookups defined in ForeignKey 'limit_choices_to' options from *other* models. Needed for some admin checks. Internal use only.
128 def __init__(self, meta, app_label=None): self.apps = self.default_apps	（解释变量） # A custom app registry to use, if you're making a separate model set.
143 def app_config(self):	（功能性） # Don't go through get_app_config to avoid triggering imports.
156 def contribute_to_class(self, cls, name): self.object_name = cls.__name__ self.model_name=self.object_name.lower() self.verbose_name=camel_case_to_spaces(self.object_name)	（解释操作） # First, construct the default values for these options.
161-162	（解释操作）

def contribute_to_class(self, cls, name): self.original_attrs = {}	# Store the original user-defined values for each option,for use when serializing the model definition

165
def contribute_to_class(self, cls, name):
if self.meta:
	（功能性，解释模块操作）
	# Next, apply any overridden values from 'class Meta'.

169-171
def contribute_to_class(self, cls, name):
if name.startswith('_'):
	（使用提示）
	# Ignore any private attributes that Django doesn't care about.
	# NOTE: We can't modify a dictionary's contents while looping over it, so we loop over the *original* dictionary instead.

185-186
def contribute_to_class(self, cls, name):
if self.verbose_name_plural is None:
	（解释分支设置原因）
	# verbose_name_plural is a special case because it uses a 's' by default.

190 def contribute_to_class(self, cls, name): self._ordering_clash = bool(self.ordering and self.order_with_respect_to)	（解释操作） # order_with_respect_and ordering are mutually exclusive.

193
def contribute_to_class(self, cls, name):
if meta_attrs != {}:
	（解释分支设置原因）
	# Any leftover attributes must be invalid.

200
def contribute_to_class(self, cls, name):
if not self.db_table:
	（解释分支内操作）
	# If the db_table wasn't provided, use the app_label + model_name.

207-208 def _prepare(self, model):	（解释操作原因） # The app registry will not be ready at this point, so we cannot use get_field().
226-227 def _prepare(self, model):	（解释操作） # Promote the first parent link in lieu of adding yet another field.
229-231 def _prepare(self, model):	（解释操作及原因） # Look for a local field with the same name as the first parent link. If a local field has already been created, use it instead of promoting the parent
250-253 def add_field(self, field, private=False):	（功能性） # Insert the given field in the order in which it was created, using the "creation_counter" attribute of the field. Move many-to-many related fields from self.fields into self.many_to_many.
262-269 def add_field(self, field, private=False):	（解释分支原因） # If the field being added is a relation to another known field,expire the cache on this field and the forward cache on the field being referenced,

	because there will be new relationships in the cache. Otherwise, expire the cache of references *to* this field.The mechanism for getting at the related model is slightly odd -ideally, we'd just ask for field.related_model. However, related_model is a cached property, and all the models haven't been loaded yet, so we need to make sure we don't cache a string reference.
285-288 def setup_proxy(self, target):	（功能性） """Do the internal setup so that the current model is a proxy for"target".""""
300-303 def can_migrate(self, connection):	（功能性） """Return True if the model can/should be migrated on the `connection`. `connection` can be either a real connection or a connection alias.""""
317 def verbose_name_raw(self):	（功能性） """Return the untranslated verbose name.""""
323-329 def swapped(self):	（功能性） """ Has this model been swapped out for another? If so, return the model name of the replacement; otherwise, return None. For historical reasons, model name lookups using get_model() are case insensitive, so we make sure we are case insensitive here.""""
336-339 def swapped(self): except ValueError:	（解释操作原因） #setting not in the format app_label.model_name #raising ImproperlyConfigured here causes problems with test cleanup code - instead it is raised in get_user_model or as part of validation.
374 def base_manager(self):	（解释操作） # Get the first parent's base_manager_name if there's one.
402 def default_manager(self):	（解释操作） # Get the first parent's default_manager_name if there's one.
424-431 def fields(self):	（功能性） """Return a list of all forward fields on the model and its parents,excluding ManyToManyFields.Private API intended only to be used by Django itself; get_fields() combined with filtering of field properties is the public API for obtaining this field list.""""
432-438 def fields(self):	（解释操作原因） # For legacy reasons, the fields property should

	only contain forward fields that are not private or with a m2m cardinality. Therefore we pass these three filters as filters to the generator.The third lambda is a longwinded way of checking f.related_model - we don't use that property directly because related_model is a cached property,and all the models may not have been loaded yet; we don't want to cache the string reference to the related_model.
458-464 def concrete_fields(self):	（功能性） """Return a list of all concrete fields on the model and its parents.Private API intended only to be used by Django itself; get_fields() combined with filtering of field properties is the public API for obtaining this field list."""
471-477 def local_concrete_fields(self):	（功能性） """Return a list of all concrete fields on the model.Private API intended only to be used by Django itself; get_fields()combined with filtering of field properties is the public API for obtaining this field list."""
484-490 def many_to_many(self):	（功能性） """Return a list of all many to many fields on the model and its parents.Private API intended only to be used by Django itself; get_fields()combined with filtering of field properties is the public API forobtaining this list."""
498-506 def related_objects(self):	（功能性） """Return all related objects pointing to the current model. The related objects can come from a one-to-one, one-to-many, or many-to-many field relation type.Private API intended only to be used by Django itself; get_fields()combined with filtering of field properties is the public API for obtaining this field list."""
519-521 def _forward_fields_map(self):	（解释操作） # Due to the way Django's internals work, get_field() should also be able to fetch a field by attname. In the case of a concrete field with relation, includes the *_id name too
534-536 def fields_map(self):	（解释操作） # Due to the way Django's internals work, get_field() should also be able to fetch a field by attname. In the case of a concrete field with

	relation, includes the *_id name too
544-546 def get_field(self, field_name):	（功能性） """Return a field instance given the name of a forward or reverse field. """
548-549 def get_field(self, field_name):	（解释操作） # In order to avoid premature loading of the relation tree(expensive) we prefer checking if the field is a forward field.
552-553 def get_field(self, field_name):	（解释操作） # If the app registry is not ready, reverse fields are unavailable, therefore we throw a FieldDoesNotExist exception.
562-563 def get_field(self, field_name):	（解释操作） # Retrieve field instance by name from cached or just-computed field map.
569-573 def get_base_chain(self, model):	（功能性） """Return a list of parent classes leading to `model` (ordered from closest to most distant ancestor). This has to handle the case where `model` is a grandparent or even more distant relation."""
586-589 def get_parent_list(self):	（功能性） """Return all the ancestors of this model as a list ordered by MRO.Useful for determining if something is an ancestor, regardless of lineage."""
597-604 def get_ancestor_link(self, ancestor):	（功能性） """Return the field on the current model which points to the given "ancestor". This is possible an indirect link (a pointer to a parent model, which points, eventually, to the ancestor). Used when constructing table joins for model inheritance.Return None if the model isn't an ancestor of this one."""
608 def get_ancestor_link(self, ancestor):	（解释操作） # Tries to get a link field from the immediate parent
611-613 def get_ancestor_link(self, ancestor):	（解释操作） # Tries to get a link field from the immediate parent
617-621 def get_path_to_parent(self, parent):	（功能性） """Return a list of PathInfos containing the path from the current model to the parent model, or an empty list if parent is not a parent of the current model."""

624 def get_path_to_parent(self, parent):	（解释操作） # Skip the chain of proxy to the concrete proxied model.
647-651 def get_path_from_parent(self, parent):	（功能性） """ Return a list of PathInfos containing the path from the parent model to the current model, or an empty list if parent is not a parent of the current model."""
655-656 def get_path_from_parent(self, parent):	（解释操作） # Get a reversed base chain including both the current and parent models.
660 def get_path_from_parent(self, parent):	（解释操作） # Construct a list of the PathInfos between models in chain.
669-674 def _populate_directed_relation_graph(self):	（功能性） """This method is used by each model to find its reverse objects. As this method is very expensive and is accessed frequently (it looks up every field in a model, in every app), it is computed on first access and then is set as a property on every model."""
680-681 def _populate_directed_relation_graph(self):	（解释操作） # Abstract model's fields are copied to child models, hence we will see the fields from the child models.
693-697 def _populate_directed_relation_graph(self):	（解释操作） # Set the relation_tree using the internal __dict__. In this way we avoid calling the cached property. In attribute lookup, __dict__ takes precedence over a data descriptor (such as @cached_property). This means that the _meta._relation_tree is only called if related_objects is not in __dict__.
700-701 def _populate_directed_relation_graph(self):	（解释操作） # It seems it is possible that self is not in all_models, so guard against that with default for get().
708 def _expire_cache(self, forward=True, reverse=True):	（解释操作） # This method is usually called by apps.cache_clear(), when the registry is finalized, or when a new field is added.
722-730 def get_fields(self, include_parents=True, include_hidden=False):	（功能性） """Return a list of fields associated to the model. By default, include forward and reverse fields,

	fields derived from inheritance, but not hidden fields. The returned fields can be changed using the parameters: - include_parents: include fields derived from inheritance - include_hidden: include fields that have a related_name that starts with a "+" """
737-747 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（功能性） """Internal helper function to return fields of the model. * If forward=True, then fields defined on this model are returned. * If reverse=True, then relations pointing to this model are returned. * If include_hidden=True, then fields with is_hidden=True are returned. * The include_parents argument toggles if fields from parent models should be included. It has three values: True, False, and PROXY_PARENTS. When set to PROXY_PARENTS, the call will return all fields defined for the current model or any of its parents in the parent chain to the model's concrete model."""
750-755 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False, seen_models=None):	（解释操作） # This helper function is used to allow recursion in ``get_fields()`` # implementation and to provide a fast way for Django's internals to access specific subsets of fields. We must keep track of which models we have already seen. Otherwise we could include the same field multiple times from different models.
761 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # Creates a cache key composed of all arguments
765-766 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # In order to avoid list manipulation. Always return a shallow copy of the results.
772-773 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # Recursively call _get_fields() on each parent, with the same options provided in this call.
776-778	（解释操作）

def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False, seen_models=None):	# In diamond inheritance it is possible that we see the same model from two different routes. In that case, avoid adding fields from the same parent again.
790-792 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # Tree is computed once and cached until the app cache is expired.It is composed of a list of fields pointing to the current model from other models.
795-796 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # If hidden fields should be included or the relation is not intentionally hidden, add to the fields dict.
803-807 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # Private fields are recopied to each child model, and they get a different model as field.model in each child. Hence we have to add the private fields separately from the topmost call. If we did this recursively similar to local_fields, we would get field instances with field.model != self.model.
811-812 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # In order to avoid list manipulation. Always return a shallow copy of the results
815 def _get_fields(self, forward=True, reverse=True, include_parents=True, include_hidden=False,seen_models=None):	（解释操作） # Store result into cache for later access
821 def _property_names(self):	（功能性） """Return a set of the names of the properties defined on the model."""

分析：由于是项目源码，代码中的注释比重很大。这部分集中了数据库的操作。注释中不仅将模块的功能说明，还在读者容易困扰的地方解释了操作的原因。代码用户友好，适于学习。