# 3. API参考

## 3.1. 连接中

**mongoengine.connect**( *db = None , alias = 'default' , ** kwargs* )

连接到“db”参数指定的数据库。

如果数据库未在本地主机的默认端口上运行，也可以在此处提供连接设置。如果需要身份验证，请同时提供用户名和密码参数。

使用别名支持多个数据库。提供一个单独的 *别名*以连接到不同的实例： program: *mongod*。

为了替换由给定别名标识的连接，您需要 `disconnect` 先调用

有关所有受支持的 kwargs 的更多详细信息，请参阅*register_connection*的文档字符串。

**mongoengine.register_connection**（*别名，db = None，name = None，host = None，port = None，read_preference = Primary()，username = None，password = None，authentication_source = None，authentication_mechanism = None，authmechanismproperties = None，** kwargs*）

注册连接设置。

**参数:** **别名**——将用于在整个 MongoEngine 中引用此连接的名称

**db** – 要使用的数据库的名称，以与 connect 兼容

**name** – 要使用的特定数据库的名称

**host** – 程序的主机名：要连接到的*mongod实例*

**port** – 程序：*mongod*实例运行的端口

**read_preference** – 集合的读取首选项

**用户名**– 用于验证的用户名

**password** – 用于验证的密码

**authentication_source** – 用于验证的数据库

**authentication_mechanism** – 数据库认证机制。默认情况下，对 MongoDB 3.0 及更高版本使用 SCRAM-SHA-1，对较旧的服务器使用 MONGODB-CR（MongoDB 挑战响应协议）。

**mongo_client_class** – 使用 pymongo.MongoClient 以外的替代连接客户端，例如 mongomock、montydb，它提供类似 pymongo 的接口，但不一定用于连接到真正的 mongo 实例。

**kwargs** – 要传递到 pymongo 驱动程序的临时参数，例如 maxpoolsize、tz_aware 等。有关完整列表，请参阅 pymongo 的*MongoClient文档*。

## 3.2. 文件

*班级* **mongoengine.Document**（**参数，**值*）

用于定义存储在 MongoDB 中的文档集合的结构和属性的基类。继承此类，并添加字段作为类属性来定义文档的结构。然后可以通过创建子类的实例来创建单个文档 `Document`。

默认情况下，用于存储使用 `Document` 子类创建的文档的 MongoDB 集合会将子类的名称转换为 snake_case。可以通过向类定义中的字典 `collection` 提供不同的集合来指定。 `meta`

子`Document`类本身可以被子类化，以创建将存储在同一集合中的文档的专门版本。为了促进这种行为，一个*_cls* 字段被添加到文档中（通过 MongoEngine 接口隐藏）。要启用此行为，请在字典中 设置`allow_inheritance`为。`True` `meta`

A`Document`可以通过在字典中指定和来使用**Capped Collection** 。是允许存储在集合中的最大文档数，并且是集合的最大大小（以字节为单位）。由 MongoDB 内部和之前的 mongoengine 四舍五入到下一个 256 的倍数。也可以使用 256 的倍数以避免混淆。如果未指定并且是，则默认为 10485760 字节 (10MB)。
`max_documents` `max_size` `meta` `max_documents` `max_size` `max_size` `max_size` `max_documents` `max_size`

`indexes`可以通过在字典中指定来创建索引`meta` 。该值应该是字段名称列表或字段名称元组。可以通过在字段名称前加上+或-符号来指定索引方向。

`auto_create_index`可以通过在字典中指定来禁用自动索引创建 `meta`。如果将其设置为 False，则 MongoEngine 将不会创建索引。这在索引创建作为部署系统的一部分执行的生产系统中很有用。

默认情况下，如果 allow_inheritance 为 True，_cls 将添加到每个索引（不包含列表）的开头。这可以通过在特定索引上将 cls 设置为 False 或在文档的元字典上将 index_cls 设置为 False 来禁用。

默认情况下，存储数据中存在但未在模型中声明的任何额外属性都会引发错误`FieldDoesNotExist`。 这可以通过在字典中设置`strict`为来禁用。`False` `meta`

初始化文档或嵌入文档。

**参数:** **values** – 文档的键和值的字典。它可能包含额外的保留关键字，例如“__auto_convert”。

**__auto_convert – 如果为真，提供的值将通过每个字段的***to_python*方法转换为 Python 类型的值。

**_created** – 表示这是一个全新的文档还是之前已经保存过。默认为真。

### `objects`

`QuerySet`在访问时延迟创建的对象。

### `cascade_save`( ***夸格斯*)

递归保存文档中的所有引用和通用引用。

### `clean`( )

在运行验证之前进行文档级数据清理（通常是验证或分配）的挂钩。

此方法引发的任何 ValidationError 都不会与特定字段相关联；它将与 NON_FIELD_ERRORS 定义的字段有特殊情况关联。

### *类方法* `compare_indexes`( )

将 MongoEngine 中定义的索引与数据库中现有的索引进行比较。返回任何缺失/额外的索引。

### *类方法* `create_index`（*键，背景= False，** kwargs*）

如果需要，创建给定的索引。

**参数:** **keys** – 单个索引键或索引键列表（构建多字段索引）；键可以以+ 或-为前缀来确定索引顺序

**background** – 允许在后台创建索引

### `delete`( *signal_kwargs = None , ** write_concern* )

`Document` 从数据库中删除。这仅在文档先前已保存时才会生效。

**参数:** **signal_kwargs** – （可选）要传递给信号调用的 kwargs 字典。

**write_concern** – 传递额外的关键字参数，它们将用作结果 `getLastError` 命令的选项。例如，将等到至少两台服务器记录了写入并强制在主服务器上进行 fsync。`save(..., w: 2, fsync: True)`

### *类方法* `drop_collection`()

`Document` 从数据库中删除与此类型关联的整个集合 。

`OperationError` 如果文档没有集合集则引发（ig 如果它是*abstract*）

### *类方法* `ensure_indexes`()

检查文档元数据并确保所有索引都存在。

可以在元中设置全局默认值 - 请参阅定义文档

默认情况下，这将在第一次与文档集合交互（查询、保存等）时自动调用，因此除非您禁用 *auto_create_index*，否则您不必手动调用它。

*如果auto_create_index_on_save*设置为 True，每次调用 Document.save 时也会调用此方法

如果多次调用，如果索引已经存在，MongoDB 将不会重新创建索引

**笔记**

您可以通过在文档元数据中将*auto_create_index*设置为 False 来禁用自动索引创建

### *类方法* `from_json`( *json_data* , *created = False* , *** kwargs* )

将 json 数据转换为 Document 实例

**参数:** **json_data** ( *str* ) – 要加载到文档中的 json 数据

**创建**（*布尔*） –

布尔值定义是将新实例化的文档视为全新的还是已经存在的：*如果为真，则将文档视为全新的，无论数据是什么

它已加载（即即使加载了 ID）。

如果为 False 且未提供 ID，则认为该文档是全新的。

如果为 False 并提供了一个 ID，则假定该对象已经被持久化（这对后续调用 .save() 有影响）。

默认为 `False`.

### `get_text_score`()

从文本查询中获取文本分数

### *类方法* `list_indexes`()

列出应为文档集合创建的所有索引。它包括来自超类和子类的所有索引。

请注意，它只会返回索引的字段，而不是索引的选项

### `modify`（*查询=无，** 更新*）

对数据库中的文档执行原子更新，并使用更新后的版本重新加载文档对象。

如果文档已更新，则返回 True；如果数据库中的文档与查询不匹配，则返回 False。

笔记

如果该方法返回 True，则拒绝对文档所做的所有未保存的更改。

**参数:** **查询**——只有当数据库中的文档与查询匹配时才会执行更新

**更新**——Django 风格的更新关键字参数

### `my_metaclass`

的别名 `mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

### *财产* `pk`

获取主键。

### *类方法* `register_delete_rule`( *document_cls* , *field_name* ,*规则*)

此方法注册删除此对象时要应用的删除规则。

### `reload`( **领域*， *** kwargs* )

从数据库重新加载所有属性。

**参数:** **fields** – （可选）args 要重新加载的字段列表

**max_depth** – （可选）要遵循的取消引用深度

### `save`（*force_insert = False*，*validate = True*，*clean = True*，*write_concern = None*，*cascade = None*，*cascade_kwargs = None*，*_refs = None*，*save_condition = None*，*signal_kwargs = None*，*** kwargs*）

保存 `Document` 到数据库。如果文档已经存在，它将被更新，否则将被创建。返回保存的对象实例。

**参数:** **force_insert** – 只尝试创建一个新文档，不允许更新现有文档。

**验证**——验证文档；设置 `False` 为跳过。

**clean**——调用文档clean方法，要求*validate*为True。

**write_concern** – 额外的关键字参数被传递给 `save()` OR `insert()` ，它将用作结果 `getLastError` 命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行 fsync。 `save(..., write_concern={w: 2, fsync: True}, ...)`

**级联**- 设置级联保存的标志。您可以通过在文档 __meta__ 中设置“级联”来设置默认值

**cascade_kwargs** – （可选）要传递给级联保存的 kwargs 字典。暗示 `cascade=True`。

**_refs** – 级联保存中使用的已处理参考列表

**save_condition** – 仅当数据库中的匹配记录满足条件（例如版本号）时才执行保存。 `OperationError` 如果不满足条件则引发

**signal_kwargs** – （可选）要传递给信号调用的 kwargs 字典。

*在 0.5 版更改：* 在现有文档中，它仅使用 set / unset 保存更改的字段。保存是级联的，任何 `DBRef` 有更改的对象也会被保存。

*在 0.6 版更改：* 添加级联保存

*在 0.8 版更改：*级联保存是可选的，默认为 False。如果您想要细粒度控制，那么您可以使用文档 meta['cascade'] = True 关闭。您也可以使用 cascade_kwargs 将不同的 kwargs 传递给级联保存，它会用自定义值覆盖现有的 kwargs。

*在 0.26 版更改：* `ensure_indexes()` 除非设置为 True，否则不再调用 save() `meta['auto_create_index_on_save']`。

**select_related**(*最大深度= 1* )

将 `DBRef` 对象的取消引用处理到最大深度，以减少对 mongodb 的查询次数。

**switch_collection**( *collection_name* , *keep_created = True* )

临时切换文档实例的集合。

仅对归档数据和调用*save()*真正有用：

```
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**参数:** **collection_name** ( *str* ) – 用于保存文档的数据库别名

**keep_created** ( *bool* ) – 切换集合后保留 self._created 值，否则重置为 True

**也可以看看**

`switch_db` 如果您需要从另一个数据库读取，请使用

**switch_db**( *db_alias* , *keep_created = True* )

临时切换文档实例的数据库。

仅对归档数据和调用*save()*真正有用：

```
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**参数:** **db_alias** ( *str* ) – 用于保存文档的数据库别名

**keep_created** ( *bool* ) – 切换数据库后保留 self._created 值，否则重置为 True

**也可以看看**

`switch_collection` 如果您需要从另一个集合中读取，请使用

**to_dbref**( )

`DBRef` 返回在 *__raw__* 查询中有用的实例。

**to_json**( **参数，** kwargs* )

将此文档转换为 JSON。

**参数:** **use_db_field** – 序列化字段名称，因为它们出现在 MongoDB 中（与本文档中的属性名称相反）。默认为真。

**to_mongo**( **参数，** kwargs* )

作为 SON 数据返回以供 MongoDB 使用。

#### `update`( ***夸格斯*)

`Document` 对A convenience wrapper执行更新到 `update()`.

`OperationError` 如果在尚未保存的对象上调用则引发。

#### `validate`（*干净=真*）

确保所有字段的值都有效并且存在必填字段。

`ValidationError` 如果发现任何字段的值无效，则引发。

---

### *班级* `mongoengine.EmbeddedDocument`( **参数，** kwargs* )

未 `Document` 存储在其自己的集合中的 。 s 应该通过 字段类型 `EmbeddedDocument` 用作 s 上的字段。 `Document` `EmbeddedDocumentField`

子 `EmbeddedDocument` 类本身可以被子类化，以创建将存储在同一集合中的嵌入文档的专用版本。为了促进这种行为，一个 *_cls* 字段被添加到文档中（通过 MongoEngine 接口隐藏）。要启用此行为，请在字典中 设置 `allow_inheritance` 为。 `True` `meta`

初始化文档或嵌入文档。

**参数:** **values** – 文档的键和值的字典。它可能包含额外的保留关键字，例如“__auto_convert”。

**__auto_convert – 如果为真，提供的值将通过每个字段的***to_python*方法转换为 Python 类型的值。

**_created** – 表示这是一个全新的文档还是之前已经保存过。默认为真。

#### `clean`( )

在运行验证之前进行文档级数据清理（通常是验证或分配）的挂钩。

此方法引发的任何 ValidationError 都不会与特定字段相关联；它将与 NON_FIELD_ERRORS 定义的字段有特殊情况关联。

#### *类方法* `from_json`( *json_data* , *created = False* , *** kwargs* )

将 json 数据转换为 Document 实例

**参数:** **json_data** ( *str* ) – 要加载到文档中的 json 数据

**创建**（*布尔*） –

布尔值定义是将新实例化的文档视为全新的还是已经存在的：*如果为真，则将文档视为全新的，无论数据是什么

它已加载（即即使加载了 ID）。

如果为 False 且未提供 ID，则认为该文档是全新的。

如果为 False 并提供了一个 ID，则假定该对象已经被持久化（这对后续调用 .save() 有影响）。

默认为 `False`.

#### `get_text_score`( )

从文本查询中获取文本分数

#### `my_metaclass`

的别名 `mongoengine.base.metaclasses.DocumentMetaclass`

**to_json**( **参数，** kwargs* )

将此文档转换为 JSON。

**参数:** **use_db_field** – 序列化字段名称，因为它们出现在 MongoDB 中（与本文档中的属性名称相反）。默认为真。

**to_mongo**( **参数，** kwargs* )

作为 SON 数据返回以供 MongoDB 使用。

**validate**（*干净=真*）

确保所有字段的值都有效并且存在必填字段。

`ValidationError` 如果发现任何字段的值无效，则引发。

## *班级* mongoengine.DynamicDocument（**参数，**值*）

允许灵活、可扩展和不受控制的模式的动态文档类。作为 `Document` 子类，与普通文档的作用相同，但具有扩展的样式属性。 `DynamicDocument` 针对非字段 传递或设置的任何数据都会自动转换为字段 `DynamicField`，并且数据可以归因于该字段。

**笔记**

动态文档有一个警告：未声明的字段不能以_*开头*

初始化文档或嵌入文档。

**参数:** **values** – 文档的键和值的字典。它可能包含额外的保留关键字，例如“__auto_convert”。

**__auto_convert – 如果为真，提供的值将通过每个字段的***to_python*方法转换为 Python 类型的值。

**_created** – 表示这是一个全新的文档还是之前已经保存过。默认为真。

**cascade_save**( ***夸格斯*)

递归保存文档中的所有引用和通用引用。

**clean**( )

在运行验证之前进行文档级数据清理（通常是验证或分配）的挂钩。

此方法引发的任何 ValidationError 都不会与特定字段相关联；它将与 NON_FIELD_ERRORS 定义的字段有特殊情况关联。

*类方法* **compare_indexes**( )

将 MongoEngine 中定义的索引与数据库中现有的索引进行比较。返回任何缺失/额外的索引。

*类方法* **create_index**（*键，背景= False，** kwargs*）

如果需要，创建给定的索引。

**参数:** **keys** – 单个索引键或索引键列表（构建多字段索引）；键可以以+ 或-为前缀来确定索引顺序

**background** – 允许在后台创建索引

**delete**( *signal_kwargs = None , ** write_concern* )

`Document`从数据库中删除。这仅在文档先前已保存时才会生效。

**参数:** **signal_kwargs** – （可选）要传递给信号调用的 kwargs 字典。

**write_concern** – 传递额外的关键字参数，它们将用作结果`getLastError`命令的选项。例如，将等到至少两台服务器记录了写入并强制在主服务器上进行 fsync。`save(..., w: 2, fsync: True)`

*类方法* **drop_collection**( )

`Document`从数据库中删除与此类型关联的整个集合 。

`OperationError`如果文档没有集合集则引发（ig 如果它是*abstract*）

*类方法* **ensure_indexes**( )

检查文档元数据并确保所有索引都存在。

可以在元中设置全局默认值 - 请参阅定义文档

默认情况下，这将在第一次与文档集合交互（查询、保存等）时自动调用，因此除非您禁用*auto_create_index*，否则您不必手动调用它。

*如果auto_create_index_on_save*设置为 True，每次调用 Document.save 时也会调用此方法

如果多次调用，如果索引已经存在，MongoDB 将不会重新创建索引

笔记

您可以通过在文档元数据中将*auto_create_index*设置为 False 来禁用自动索引创建

*类方法* **from_json**( *json_data , created = False , ** kwargs* )

将 json 数据转换为 Document 实例

**参数:** **json_data** ( *str* ) – 要加载到文档中的 json 数据

**创建**（*布尔*） –

布尔值定义是将新实例化的文档视为全新的还是已经存在的：*如果为真，则将文档视为全新的，无论数据是什么

它已加载（即即使加载了 ID）。

如果为 False 且未提供 ID，则认为该文档是全新的。

如果为 False 并提供了一个 ID，则假定该对象已经被持久化（这对后续调用 .save() 有影响）。

默认为`False`.

**get_text_score**( )

从文本查询中获取文本分数

*类方法* **list_indexes**( )

列出应为文档集合创建的所有索引。它包括来自超类和子类的所有索引。

请注意，它只会返回索引的字段，而不是索引的选项

**modify**（*查询=无，** 更新*）

对数据库中的文档执行原子更新，并使用更新后的版本重新加载文档对象。

如果文档已更新，则返回 True；如果数据库中的文档与查询不匹配，则返回 False。

笔记

如果该方法返回 True，则拒绝对文档所做的所有未保存的更改。

**参数:** **查询**——只有当数据库中的文档与查询匹配时才会执行更新

**更新**——Django 风格的更新关键字参数

### `my_metaclass`

的别名`mongoengine.base.metaclasses.TopLevelDocumentMetaclass`

### *财产* `pk`

获取主键。

### *类方法* `register_delete_rule`( *document_cls* , *field_name* ,*规则*)

此方法注册删除此对象时要应用的删除规则。

### `reload`( **领域*， *** kwargs* )

从数据库重新加载所有属性。

**参数:** **fields** – （可选）args 要重新加载的字段列表

**max_depth** – （可选）要遵循的取消引用深度

### `save`（*force_insert = False*，*validate = True*，*clean = True*，*write_concern = None*，*cascade = None*，*cascade_kwargs = None*，*_refs = None*，*save_condition = None*，*signal_kwargs = None*，*** kwargs*）

保存`Document`到数据库。如果文档已经存在，它将被更新，否则将被创建。返回保存的对象实例。

**参数:** **force_insert** – 只尝试创建一个新文档，不允许更新现有文档。

**验证**——验证文档；设置`False`为跳过。

**clean**——调用文档clean方法，要求*validate*为True。

**write_concern** – 额外的关键字参数被传递给`save()`OR`insert()`，它将用作结果`getLastError`命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行 fsync。`save(..., write_concern={w: 2, fsync: True}, ...)`

**级联**- 设置级联保存的标志。您可以通过在文档 __meta__ 中设置“级联”来设置默认值

**cascade_kwargs** – （可选）要传递给级联保存的 kwargs 字典。暗示`cascade=True`。

**_refs** – 级联保存中使用的已处理参考列表

**save_condition** – 仅当数据库中的匹配记录满足条件（例如版本号）时才执行保存。`OperationError`如果不满足条件则引发

**signal_kwargs** – （可选）要传递给信号调用的 kwargs 字典。

*在 0.5 版更改：*在现有文档中，它仅使用 set / unset 保存更改的字段。保存是级联的，任何`DBRef`有更改的对象也会被保存。

*在 0.6 版更改：*添加级联保存

*在 0.8 版更改：*级联保存是可选的，默认为 False。如果您想要细粒度控制，那么您可以使用文档 meta['cascade'] = True 关闭。您也可以使用 cascade_kwargs 将不同的 kwargs 传递给级联保存，它会用自定义值覆盖现有的 kwargs。

*在 0.26 版更改：* `ensure_indexes()` 除非设置为 True，否则不再调用 save() `meta['auto_create_index_on_save']`。

### `select_related`(*最大深度= 1* )

将 `DBRef` 对象的取消引用处理到最大深度，以减少对 mongodb 的查询次数。

### `switch_collection`( *collection_name* , *keep_created = True* )

临时切换文档实例的集合。

仅对归档数据和调用*save()*真正有用：

```
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

**参数:** **collection_name** ( *str* ) – 用于保存文档的数据库别名

**keep_created** ( *bool* ) – 切换集合后保留 self._created 值，否则重置为 True

**也可以看看**

`switch_db` 如果您需要从另一个数据库读取，请使用

### `switch_db`( *db_alias* , *keep_created = True* )

临时切换文档实例的数据库。

仅对归档数据和调用*save()*真正有用：

```
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

**参数:** **db_alias** ( *str* ) – 用于保存文档的数据库别名

**keep_created** ( *bool* ) – 切换数据库后保留 self._created 值，否则重置为 True

**也可以看看**

`switch_collection` 如果您需要从另一个集合中读取，请使用

### `to_dbref`( )

`DBRef` 返回在 *__raw__* 查询中有用的实例。

### `to_json`( **参数，** kwargs* )

将此文档转换为 JSON。

**参数:** **use_db_field** – 序列化字段名称，因为它们出现在 MongoDB 中（与本文档中的属性名称相反）。默认为真。

### `to_mongo`( **参数，** kwargs* )

作为 SON 数据返回以供 MongoDB 使用。

#### update( **夸格斯*)

`Document` 对A convenience wrapper执行更新到 `update()`.

`OperationError` 如果在尚未保存的对象上调用则引发。

#### validate（*干净=真*）

确保所有字段的值都有效并且存在必填字段。

`ValidationError` 如果发现任何字段的值无效，则引发。

---

### *班级* mongoengine.DynamicEmbeddedDocument( **参数，** kwargs* )

允许灵活、可扩展和不受控制的模式的动态嵌入式文档类。 `DynamicDocument` 有关动态文档的更多信息，请参阅。

初始化文档或嵌入文档。

**参数:** **values** – 文档的键和值的字典。它可能包含额外的保留关键字，例如“__auto_convert”。

**__auto_convert – 如果为真，提供的值将通过每个字段的***to_python*方法转换为 Python 类型的值。

**_created** – 表示这是一个全新的文档还是之前已经保存过。默认为真。

#### clean( )

在运行验证之前进行文档级数据清理（通常是验证或分配）的挂钩。

此方法引发的任何 ValidationError 都不会与特定字段相关联；它将与 NON_FIELD_ERRORS 定义的字段有特殊情况关联。

#### *类方法* from_json( *json_data , created = False , ** kwargs* )

将 json 数据转换为 Document 实例

**参数:** **json_data** ( *str* ) – 要加载到文档中的 json 数据

**创建**（*布尔*） –

布尔值定义是将新实例化的文档视为全新的还是已经存在的：*如果为真，则将文档视为全新的，无论数据是什么

它已加载（即即使加载了 ID）。

如果为 False 且未提供 ID，则认为该文档是全新的。

如果为 False 并提供了一个 ID，则假定该对象已经被持久化（这对后续调用 .save() 有影响）。

默认为 `False`.

#### get_text_score( )

从文本查询中获取文本分数

#### my_metaclass

的别名 `mongoengine.base.metaclasses.DocumentMetaclass`

#### to_json( **参数，** kwargs* )

将此文档转换为 JSON。

**参数:** **use_db_field** – 序列化字段名称，因为它们出现在 MongoDB 中（与本文档中的属性名称相反）。默认为真。

### `to_mongo`( *参数，** kwargs )

作为 SON 数据返回以供 MongoDB 使用。

### `validate`（*干净=真*）

确保所有字段的值都有效并且存在必填字段。

`ValidationError` 如果发现任何字段的值无效，则引发。

## *班级* `mongoengine.document.MapReduceDocument`（*文档、集合、键、值*）

从 map/reduce 查询返回的文档。

**参数:** **集合**——一个实例 `Collection`

**key** – 文档/结果键，通常是 `ObjectId` 。 `ObjectId` 如果在给定的中以找到的形式提供 `collection` ，则可以通过属性访问该对象 `object` 。

**value** – 此键的结果。

### *财产* `object`

延迟加载 引用的对象 `self.key` 。 `self.key` 应该是 `primary_key` 。

## *班级* `mongoengine.ValidationError`(*消息= ''，** kwargs* )

验证异常。

可能表示验证字段或包含具有验证错误的字段的文档的错误。

**变量:** **errors** – 此文档或列表中字段的错误字典，如果错误是针对单个字段的，则为 None。

### `to_dict`( )

返回文档中所有错误的字典

键是字段名称或列表索引，值是验证错误消息，或嵌入文档或列表的嵌套错误字典。

## *班级* `mongoengine.FieldDoesNotExist`

`Document` 尝试设置未在 a或 中声明的字段时引发 `EmbeddedDocument` 。

为避免数据加载时的这种行为，您应该 在字典中设置 `strict` to 。 `False` `meta`

# 3.3. 上下文管理器

## *班级* `mongoengine.context_managers.switch_db`（*cls，db_alias*）

switch_db 别名上下文管理器。

例子

```
# Register connections
register_connection('default', 'mongoenginetest')
register_connection('testdb-1', 'mongoenginetest2')

class Group(Document):
    name = StringField()

Group(name='test').save()  # Saves in the default db

with switch_db(Group, 'testdb-1') as Group:
    Group(name='hello testdb!').save()  # Saves in testdb-1
```

构建 switch_db 上下文管理器

**参数:** **cls** – 更改已注册数据库的类

**db_alias** – 要使用的特定数据库的名称

### *班级* mongoengine.context_managers.switch_collection( *cls* , *collection_name* )

switch_collection 别名上下文管理器。

例子

```
class Group(Document):
    name = StringField()

Group(name='test').save()  # Saves in the default db

with switch_collection(Group, 'group1') as Group:
    Group(name='hello testdb!').save()  # Saves in group1 collection
```

构建 switch_collection 上下文管理器。

**参数:** **cls** – 更改已注册数据库的类

**collection_name** – 要使用的集合的名称

### *班级* mongoengine.context_managers.no_dereference( *cls* )

no_dereference 上下文管理器。

在上下文管理器期间关闭 Documents 中的所有取消引用：

```
with no_dereference(Group) as Group:
    Group.objects.find()
```

构造 no_dereference 上下文管理器。

**参数:** **cls** – 关闭取消引用的类

### *班级* mongoengine.context_managers.query_counter（*别名= '默认'*）

Query_counter 上下文管理器获取查询次数。这通过更新数据库的*profiling_level*来工作，以便记录所有查询，在上下文的开头重置 db.system.profile 集合并计算新条目。

这是为调试目的而设计的。事实上，它是一个全局计数器，因此其他线程/进程发出的查询可能会干扰它

用法：

```
class User(Document):
    name = StringField()

with query_counter() as q:
    user = User(name='Bob')
    assert q == 0       # no query fired yet
    user.save()
    assert q == 1       # 1 query was fired, an 'insert'
    user_bis = User.objects().first()
    assert q == 2       # a 2nd query was fired, a 'find_one'
```

意识到：

迭代大量文档（>101）使 pymongo 发出*getmore*查询以获取下一批文档（https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/#cursor-batches）

计数器默认忽略某些查询（killcursors、db.system.indexes）

## 3.4. 查询

***班级*** **`mongoengine.queryset.QuerySet`** ***（文档，集合）***

默认查询集，用于构建查询并处理查询返回的一组结果。

包装 MongoDB 游标，提供 `Document` 对象作为结果。

> **`__call__`**( *q_obj = None* , ***查询*** )
>
> `QuerySet` 通过使用查询调用 来过滤选定的文档 。
>
> **参数:** **q_obj——** `Q` 查询中使用的对象； `QuerySet` 使用不同的对象多次过滤， `Q` 只会使用最后一个。
>
> **查询**——Django 风格的查询关键字参数。

> **`aggregate`**（***管道***，* *suppl_pipeline*，** *kwargs*）
>
> 根据您的查询集参数执行聚合函数
>
> **参数:** **管道**- 聚合命令列表，请参阅：http ://docs.mongodb.org/manual/core/aggregation-pipeline/
>
> **suppl_pipeline** – 管道的解压缩列表（添加以支持旧接口的弃用）参数将很快被删除
>
> **kwargs** – （可选）要传递给 pymongo 的聚合调用的 kwargs 字典参见 https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collectio

> **`all`**()
>
> 返回当前 QuerySet 的副本。

> **`all_fields`**()
>
> 包括所有字段。重置所有先前调用的 .only() 或 .exclude()。
>
> ```
> post = BlogPost.objects.exclude('comments').all_fields()
> ```

> **`allow_disk_use`**（***已启用***）
>
> **在处理阻塞排序操作时启用或禁用磁盘上的临时文件。**
>
> （存储超过 100 兆字节系统内存限制的数据）
>
> **参数:** **enabled——**是否使用磁盘上的临时文件

**as_pymongo**( )

不返回 Document 实例，而是从 pymongo 返回原始值。

如果您不需要解除引用并且主要关心数据检索的速度，则此方法特别有用。

**average**（*场*）

对指定字段的值进行平均。

**参数:** **field** – 要平均的字段；使用点符号来引用嵌入的文档字段

**batch_size**(*尺寸*)

限制单个批次中返回的文档数量（每个批次都需要往返服务器）。

有关详细信息，请参阅 http://api.mongodb.com/python/current/api/pymongo/cursor.html#pymongo.cursor.Cursor.batch_size 。

**参数:** **size——每批所需的大小。**

**clear_cls_query**( )

清除默认的“_cls”查询。

默认情况下，为允许继承的文档生成的所有查询都包含一个额外的“_cls”子句。在大多数情况下，这是可取的，但有时如果您清除该默认查询，您可能会获得更好的性能。

*扫一扫_cls_query*二维码获取更多详情。

**clone**( )

创建当前查询集的副本。

**collation**（*整理=无*）

排序规则允许用户指定特定于语言的字符串比较规则，例如字母大小写和重音符号的规则。:param collation: *~pymongo.collation.Collation*或具有以下字段的字典：

{

语言环境：str，caseLevel：bool，caseFirst：str，强度：int，numericOrdering：bool，替代：str，maxVariable：str，向后：str

}

排序规则应该像测试示例一样添加到索引中

**comment**（*文字*）

向查询添加注释。

有关详细信息，请参阅 https://docs.mongodb.com/manual/reference/method/cursor.comment/#cursor.comment 。

**count**( *with_limit_and_skip = False* )

计算查询中选定的元素。

**参数:** **( optional )** ( *with_limit_and_skip* ) –在获取计数时考虑已应用于此游标的任何 `limit()` 或 `skip()`

**create**( ***夸格斯*)

创建新对象。返回保存的对象实例。

### `delete`( *write_concern = None , _from_doc_delete = False , cascade_refs = None* )

删除与查询匹配的文档。

**参数:** **write_concern** – 传递额外的关键字参数，它们将用作结果 `getLastError` 命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行 fsync。 `save(..., write_concern={w: 2, fsync: True}, ...)`

**_from_doc_delete** – 从文档删除调用时为真，因此信号将被触发，因此不要循环。

:返回已删除文档的数量

### `distinct`（*场*）

返回给定字段的不同值列表。

**参数:** **field** – 从中选择不同值的字段

**笔记**

这是一个命令，不会考虑排序或限制。

### `exclude`( **字段*)

与 .only() 相反，排除某些文档的字段。

```
post = BlogPost.objects(...).exclude('comments')
```

**笔记**

*exclude()*是可链接的，并将执行联合 :: 因此，对于以下内容，它将同时排除：*title*和 *author.name*：

```
post = BlogPost.objects.exclude('title').exclude('author.name')
```

`all_fields()` 将重置任何字段过滤器。

**参数:** **字段**——要排除的字段

### `exec_js`（*代码，*字段，**选项*）

在服务器上执行 Javascript 函数。可以提供字段列表，这些字段将被翻译成它们的正确名称并作为函数的参数提供。一些额外的变量被添加到函数的作用域中： `collection` ，这是正在使用的集合的名称； `query` ，是代表当前查询的对象；and `options` ，它是一个对象，包含指定为关键字参数的任何选项。

由于 MongoEngine 中的字段可能在数据库中使用不同的名称（使用构造 `db_field` 函数的关键字参数设置 `Field` ），因此存在一种机制，可以用 Javascript 代码中的数据库字段名称替换 MongoEngine 字段名称。访问字段时，使用方括号表示法，并在 MongoEngine 字段名称前加上波浪号 (~)。

**参数:** **代码**——要执行的一串 Javascript 代码

**fields**——你将在你的函数中使用的字段，它们将作为参数传递给你的函数

options – 您希望函数可用的选项（通过`options`对象在 Javascript 中访问）

**explain**( )

返回游标的解释计划记录`QuerySet`。

**fields**( *_only_called = False* , *** kwargs* )

操纵如何加载此文档的字段。由*.only()* 和*.exclude()*用于操作要检索的字段。如果直接调用，使用一组类似于 MongoDB 投影文档的 kwargs。例如：

仅包括字段的子集：

posts = BlogPost.objects(...).fields(作者=1, 标题=1)

排除特定字段：

posts = BlogPost.objects(...).fields(评论=0)

要检索数组元素的子范围或子列表，支持*切片*和*elemMatch*投影运算符：

posts = BlogPost.objects(...).fields(slice__comments=5) posts = BlogPost.objects(...).fields(elemMatch__comments="test")

**参数:** **kwargs** – 一组关键字参数，用于标识要包含、排除或切片的内容。

**filter**( *** q_objs* , ****查询*)

的别名`__call__()`

**first**( )

检索匹配查询的第一个对象。

**from_json**( *json_data* )

将 json 数据转换为未保存的对象

**get**( *** q_objs* , ****查询*)

检索匹配对象引发`MultipleObjectsReturned`或 *DocumentName.MultipleObjectsReturned*异常（如果有多个结果）和`DoesNotExist`/或 *DocumentName.DoesNotExist*（如果未找到结果）。

**hint**（*索引=无*）

添加了"提示"支持，告诉 Mongo 用于查询的正确索引。

明智地使用提示可以大大提高查询性能。当对多个字段（至少其中一个被索引）进行查询时，将索引字段作为查询的提示传递。

如果相应的索引不存在，提示将不会做任何事情。应用于此游标的最后一个提示优先于所有其他提示。

**in_bulk**( *object_ids* )

通过 ID 检索一组文档。

**参数:** **object_ids** – ObjectId 的列表或元组

**返回类型:** ObjectId 的字典作为键，集合特定的文档子类作为值。

**insert**（*doc_or_docs*，*load_bulk = True*，*write_concern = None*，*signal_kwargs = None*）

批量插入文件

**参数:** **doc_or_docs** – 要插入的文档或文档列表

**(可选)** ( *load_bulk* ) – 如果 True 返回文档实例列表

**write_concern** – 额外的关键字参数被传递给 `insert()` 将用作结果命令的选项 `getLastError` 。例如， 将等到至少有两个服务器记录了写入，并将在每个正在写入的服务器上强制执行 fsync。 `insert(..., {w: 2, fsync: True})`

**signal_kwargs** – （可选）要传递给信号调用的 kwargs 字典。

默认返回文档实例，设置 `load_bulk` 为 False 只返回 `ObjectIds`

### `item_frequencies`（*字段，归一化=假，map_reduce =真*）

返回整个查询文档集中某个字段中存在的所有项目及其相应频率的字典。这对于生成标签云或搜索文档很有用。

> **笔记**
>
> 只能做直接的简单映射，不能跨映射 `ReferenceField` 或 `GenericReferenceField` 更复杂的计数需要手动 map reduce 调用。

如果该字段是 `ListField` ，每个列表中的项目将被单独计算。

**参数:** **字段**——要使用的字段

**归一化**——归一化结果，使它们相加为 1.0

**map_reduce** – 在 exec_js 上使用 map_reduce

### `limit`(*名词*)

将返回文档的数量限制为*n*。这也可以使用数组切片语法（例如 `User.objects[:5]` ）来实现。

**参数:** **n** – 如果 n 大于 0，则返回的最大对象数。

传递 0 时，返回游标中的所有文档

### `map_reduce`( *map_f* , *reduce_f* , *output* , *finalize_f = None* , *limit = None* , *scope = None* )

使用当前查询规范和排序执行映射/归约查询。虽然 `map_reduce` 尊重 `QuerySet` 链接，但它必须是最后一次调用，因为它不会返回 maleable `QuerySet` 。

有关使用示例，请参阅 中的 `test_map_reduce()` 和测试。
`test_map_advanced()` `tests.queryset.QuerySetTest`

**参数:** **map_f** – 映射函数，as `Code` 或 string

**reduce_f** – reduce 函数，as `Code` 或 string

**output** – 输出集合名称，如果设置为 'inline' 将以内联方式返回结果。这也可以是包含输出选项的字典，请参阅：http://docs.mongodb.org/manual/reference/command/mapReduce/#dbcmd.mapReduce

**finalize_f** – finalize 函数，一个可选函数，用于执行任何缩减后处理。

**scope** – 要插入到 map/reduce 全局范围的值。选修的。

**limit** – 当前查询中提供给 map/reduce 方法的对象数量

返回一个迭代器 yielding `MapReduceDocument` 。

### `max_time_ms`（*毫秒*）

在终止服务器上的查询之前等待*ms毫秒*

**参数:** ms – 在服务器上终止查询之前的毫秒数

### `modify`（*upsert = False，full_response = False，remove = False，new = False，**更新*）

更新并返回更新后的文档。

*根据新*参数返回修改之前或之后的文档 。如果没有文档匹配查询并且*upsert*为 false，则返回`None`。如果 upserting 和*new*为 false，则返回`None`。

如果 full_response 参数是`True`，则返回值将是来自服务器的整个响应对象，包括“ok”和“lastErrorObject”字段，而不仅仅是修改后的文档。这很有用，主要是因为“lastErrorObject”文档包含有关命令执行的信息。

**参数:** **upsert** – 如果文档不存在则插入（默认`False`）

**full_response** – 从服务器返回整个响应对象（默认`False`，不适用于 PyMongo 3+）

**remove** – 删除而不是更新（默认`False`）

**new** – 返回更新的而不是原始文档（默认`False`）

**更新**——Django 风格的更新关键字参数

### `no_cache`()

转换为非缓存查询集

### `no_dereference`()

关闭对此查询集结果的任何取消引用。

### `no_sub_classes`()

仅过滤此特定文档的实例。

不要返回任何继承的文档。

### `none`()

返回一个从不返回任何对象的查询集，并且在访问受 django none() 启发的结果时不会执行任何查询https://docs.djangoproject.com/en/dev/ref/models/querysets/#none

### `only`(**字段*)

仅加载此文档字段的一个子集。

```
post = BlogPost.objects(...).only('title', 'author.name')
```

**笔记**

*only()*是可链接的并且将执行联合 :: 因此对于以下内容它将获取：*title*和*author.name*：

```
post = BlogPost.objects.only('title').only('author.name')
```

`all_fields()`将重置任何字段过滤器。

**参数:** **字段**——要包含的字段

### order_by（*键）

按 `QuerySet` 给定的键排序。

可以通过在每个键前面加上“+”或“-”来指定顺序。如果没有前缀，则假定升序。

如果未传递任何键，则会清除现有排序。

**参数:** **keys** – 用于对查询结果进行排序的字段；键可以以“+”或“-”为前缀来确定排序方向。

### read_concern（*阅读关注*）

查询时更改read_concern。

**参数:** **read_concern** – 覆盖 ReplicaSetConnection 级别的首选项。

### read_preference（*阅读偏好*）

查询时更改 read_preference。

**参数:** **read_preference** – 覆盖 ReplicaSetConnection 级别的首选项。

### rewind()

将游标倒回到其未评估的状态。

### scalar(*字段)

不是返回 Document 实例，而是按顺序返回特定值或值元组。

可以与 一起使用 `no_dereference()` 以关闭解除引用。

**笔记**

这会影响所有结果，并且可以通过 `scalar` 不带参数调用来取消设置。自动调用 `only` 。

**参数:** **fields** – 要返回的一个或多个字段，而不是文档。

### search_text（*文字，语言=无*）

使用文本索引开始文本搜索。要求：MongoDB 服务器版本 2.6+。

**参数:** **language** – 确定搜索停用词列表以及词干分析器和分词器规则的语言。如果未指定，则搜索使用索引的默认语言。有关支持的语言，请参阅 *文本搜索语言 <http://docs.mongodb.org/manual/reference/text-search-languages/#text-search-languages>*。

### select_related(*最大深度= 1*)

处理 `DBRef` 对象的取消引用或 `ObjectId` 最大深度，以减少对 mongodb 的查询次数。

### skip(*名词*)

在返回结果之前跳过*n 个文档*。这也可以使用数组切片语法（例如 `User.objects[5:]` ）来实现。

**参数:** **n** – 返回结果前要跳过的对象数

### snapshot（*已启用*）

查询时启用或禁用快照模式。

**参数:** **enabled** – 是否启用快照模式

---

**`sum`**（*场*）

对指定字段的值求和。

**参数:** **field** – 要求和的字段；使用点符号来引用嵌入的文档字段

---

**`timeout`**（*已启用*）

查询时启用或禁用默认的 mongod 超时。（no_cursor_timeout 选项）

**参数:** **enabled** – 是否使用超时

---

**`to_json`**( **参数*，*** kwargs* )

将查询集转换为 JSON

---

**`update`**（*upsert = False*，*multi = True*，*write_concern = None*，*read_concern = None*，*full_result = False*，***更新*）

对查询匹配的字段执行原子更新。

**参数:** **upsert** – 如果文档不存在则插入（默认`False`）

**multi** – 更新多个文档。

**write_concern** – 传递额外的关键字参数，它们将用作结果`getLastError`命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行fsync。`save(..., write_concern={w: 2, fsync: True}, ...)`

**read_concern** – 覆盖操作的读取关注

**full_result** – 返回关联的`pymongo.UpdateResult`而不仅仅是更新项目的数量

**更新**——Django 风格的更新关键字参数

:返回更新文档的数量（除非`full_result`为 True）

---

**`update_one`**（*upsert = False*，*write_concern = None*，*full_result = False*，***更新*）

对查询匹配的第一个文档的字段执行原子更新。

**参数:** **upsert** – 如果文档不存在则插入（默认`False`）

**write_concern** – 传递额外的关键字参数，它们将用作结果`getLastError`命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行fsync。`save(..., write_concern={w: 2, fsync: True}, ...)`

**full_result** – 返回关联的`pymongo.UpdateResult`而不仅仅是更新项目的数量

**更新**——Django 风格的更新关键字参数 full_result

:返回更新文档的数量（除非`full_result`为 True）

---

**`upsert_one`**（*write_concern =无*，*read_concern =无*，***更新*）

覆盖或添加与查询匹配的第一个文档。

**参数:** **write_concern** – 传递额外的关键字参数，它们将用作结果`getLastError`命令的选项。例如， 将等到至少两台服务器记录了写入并强制在主服务器上进行fsync。`save(..., write_concern={w: 2, fsync: True}, ...)`

**read_concern** – 覆盖操作的读取关注

**更新**——Django 风格的更新关键字参数

:返回新的或覆盖的文件

> **using**（*别名*）

如果您使用多个数据库，此方法用于控制将针对哪个数据库评估 QuerySet。

**参数:** **别名**——数据库别名

> **values_list**( **字段*)

标量的别名

> **where**( *where_clause* )

使用子句（Javascript 表达式）过滤`QuerySet`结果。`$where`执行自动字段名称替换，如`mongoengine.queryset.Queryset.exec_js()`.

笔记

使用这种查询模式时，数据库将为集合中的每个对象调用您的函数，或评估您的谓词子句。

> **with_id**( *object_id* )

检索与提供的 ID 匹配的对象。仅使用*object_id*并在应用过滤器时引发 InvalidQueryError。如果不存在具有该 ID 的文档，则返回 *None* 。

**参数:** **object_id** – 要查找的文档 ID 的值

---

*班级* **mongoengine.queryset.QuerySetNoCache**（*文档，集合*）

非缓存查询集

> **__call__**( *q_obj = None , **查询*)

`QuerySet`通过使用查询调用 来过滤选定的文档 。

**参数:** **q_obj**——`Q`查询中使用的对象；`QuerySet`使用不同的对象多次过滤，`Q`只会使用最后一个。

**查询**——Django 风格的查询关键字参数。

> **cache**( )

转换为缓存查询集

---

**mongoengine.queryset.queryset_manager**（*功能*）

允许您在类上定义自定义 QuerySet 管理器的装饰器`Document`。管理器必须是一个接受`Document`类作为其第一个参数和 a `QuerySet`作为其第二个参数的函数。方法函数应该返回一个`QuerySet`，可能与传入的相同，但以某种方式进行了修改。

## 3.5. 领域

---

*班级* **mongoengine.base.fields.BaseField**( *db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs* )

MongoDB 文档中字段的基类。此类的实例可以添加到*Document*的子类中以定义文档的架构。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** – （可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** – （可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** – （可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.StringField** ***（正则表达式=无，最大长度=无，最小长度=无，** kwargs）***

一个 unicode 字符串字段。

**参数:** **正则表达式**——（可选）将在验证期间应用的字符串模式

**max_length** – （可选）将在验证期间应用的最大长度

**min_length** – （可选）将在验证期间应用的最小长度

**kwargs**——传递给父级的关键字参数 `BaseField`

---

***班级*** **mongoengine.fields.URLField** ***( url_regex = None , schemes = None , ** kwargs )***

将输入验证为 URL 的字段。

**参数:** **url_regex** – （可选）覆盖用于验证的默认正则表达式

**schemes** – （可选）覆盖允许的默认 URL 方案

**kwargs**——传递给父级的关键字参数 `StringField`

---

***班级*** **mongoengine.fields.EmailField** ***( domain_whitelist = None , allow_utf8_user = False , allow_ip_domain = False , * args , ** kwargs )***

将输入验证为电子邮件地址的字段。

**参数:** **domain_whitelist** – （可选）验证期间应用的有效域名列表

**allow_utf8_user** – 允许电子邮件的用户部分包含 utf8 字符

**allow_ip_domain** – 允许电子邮件的域部分是 IPv4 或 IPv6 地址

**kwargs**——传递给父级的关键字参数 `StringField`

---

***班级*** **mongoengine.fields.EnumField** ***（枚举，** kwargs）***

枚举字段。值按原样存储在下面，因此它只适用于 bson 可编码的简单类型（str、int 等）

用法示例：

```
class Status(Enum):
    NEW = 'new'
    ONGOING = 'ongoing'
    DONE = 'done'

class ModelWithEnum(Document):
    status = EnumField(Status, default=Status.NEW)

ModelWithEnum(status='done')
ModelWithEnum(status=Status.DONE)
```

可以使用枚举或其值搜索枚举字段：

```
ModelWithEnum.objects(status='new').count()
ModelWithEnum.objects(status=Status.NEW).count()
```

可以使用以下`choices`参数将值限制为枚举的子集：

```
class ModelWithEnum(Document):
    status = EnumField(Status, choices=[Status.NEW, Status.DONE])
```

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.IntField**( *min_value = None , max_value = None , ** kwargs* )

32 位整数字段。

**参数:** **min_value** –（可选）将在验证期间应用的最小值

**max_value** –（可选）将在验证期间应用的最大值

kwargs——传递给父级的关键字参数 `BaseField`

---

*班级* **mongoengine.fields.LongField**( *min_value = None , max_value = None , ** kwargs* )

64 位整数字段。（等同于 IntField 因为不再支持 Python2）

**参数:** **min_value** – （可选）将在验证期间应用的最小值

**max_value** – （可选）将在验证期间应用的最大值

kwargs——传递给父级的关键字参数 `BaseField`

---

*班级* **mongoengine.fields.FloatField**( *min_value = None , max_value = None , ** kwargs* )

浮点数字段。

**参数:** **min_value** – （可选）将在验证期间应用的最小值

**max_value** – （可选）将在验证期间应用的最大值

kwargs——传递给父级的关键字参数 `BaseField`

---

*班级* **mongoengine.fields.DecimalField**( *min_value = None , max_value = None , force_string = False , precision = 2 , rounding = 'ROUND_HALF_UP' , ** kwargs* )

免责声明：由于历史原因保留此字段，但由于它将值转换为浮点数，因此不适合真正的十进制存储。考虑使用 `Decimal128Field` .

定点十进制数字段。默认情况下将值存储为浮点数，除非使用*force_string* 。如果使用浮点数，请注意十进制到浮点数的转换（潜在的精度损失）

**参数:** **min_value** – （可选）将在验证期间应用的最小值

**max_value** – （可选）将在验证期间应用的最大值

**force_string** – 将值存储为字符串（而不是浮点数）。请注意，这会影响查询排序和操作，如 lte、gte（因为应用了字符串比较）并且某些查询运算符将不起作用（例如 inc、dec）

**precision** – 要存储的小数位数。

**四舍五入**–

来自 python decimal 库的舍入规则：

decimal.ROUND_CEILING（接近无穷大）

decimal.ROUND_DOWN（接近零）

decimal.ROUND_FLOOR（朝向 -Infinity）

decimal.ROUND_HALF_DOWN（最接近零）

decimal.ROUND_HALF_EVEN（最接近的是最接近的偶数）

decimal.ROUND_HALF_UP（从零到最接近）

decimal.ROUND_UP（远离零）

decimal.ROUND_05UP（如果向零四舍五入后的最后一位数字为 0 或 5，则远离零；否则向零）

默认为： `decimal.ROUND_HALF_UP`

kwargs——传递给父级的关键字参数 `BaseField`

---

*班级* **mongoengine.fields.BooleanField**( *db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse =*

*False , ** kwargs )*

布尔字段类型。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.DateTimeField**( *db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs* )

日期时间字段。

使用 python-dateutil 库（如果可用）或者使用 time.strptime 来解析日期。注意：python-dateutil 的解析器功能齐全，安装后您可以使用它来将各种类型的日期格式转换为有效的 python 日期时间对象。

注意：要将该字段默认为当前日期时间，请使用：DateTimeField(default=datetime.utcnow)

> **注意：微秒四舍五入到最接近的毫秒。**
>
> UTC 前微秒支持已被有效破坏。 `ComplexDateTimeField` 如果您需要精确的微秒支持，请使用。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.ComplexDateTimeField**( *separator = ':' , ** kwargs* )

ComplexDateTimeField 精确处理微秒，而不是像 DateTimeField 那样四舍五入。

从 StringField 派生，因此您可以在过滤/排序字符串时通过使用字典序比较来进行*gte*和*lte过滤*。

存储的字符串具有以下格式：

YYYY,MM,DD,HH,MM,SS,NNNNNN

*其中 NNNNNN 是表示的日期时间*的微秒数。作为分隔符的,可以 在初始化字段时通过传递*separator*关键字轻松修改。

注意：要将该字段默认为当前日期时间，请使用：DateTimeField(default=datetime.utcnow)

**参数:** **separator** – 允许自定义用于存储的分隔符（默认`,`）

**kwargs——传递给父级的关键字参数`StringField`**

---

*班级* **mongoengine.fields.EmbeddedDocumentField**（*文档类型，** kwargs*）

嵌入式文档字段 - 带有已声明的文档类型。只有有效值是 的子类`EmbeddedDocument`。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

*班级* **mongoengine.fields.GenericEmbeddedDocumentField**( *db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs* )

通用嵌入式文档字段 - 允许 `EmbeddedDocument` 存储任何内容。

只有有效值是 的子类 `EmbeddedDocument` 。

笔记

您可以使用 choices 参数来限制可接受的 EmbeddedDocument 类型

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

*班级* **mongoengine.fields.DynamicField**( *db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs* )

真正的动态字段类型，能够处理不同类型的数据。

用于 `DynamicDocument` 处理动态数据

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

### *班级* `mongoengine.fields.ListField`*（字段=无，最大长度=无，** kwargs）*

包装标准字段的列表字段，允许该字段的多个实例用作数据库中的列表。

如果与 ReferenceFields 一起使用，请参阅：Many to Many with ListFields

**笔记**

必需意味着它不能为空 - 因为 ListFields 的默认值是 []

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

### *班级* `mongoengine.fields.EmbeddedDocumentListField`*（文档类型，** kwargs）*

专门设计`ListField`用于保存嵌入式文档列表以提供额外的查询助手。

**笔记**

唯一有效的列表值是 的子类`EmbeddedDocument`。

**参数:** **document_type**`EmbeddedDocument` –列表的类型 。

**kwargs**——传递给父级的关键字参数`ListField`

*班级* **mongoengine.fields.SortedListField**（*领域，** kwargs*）

一个 ListField，它在写入数据库之前对其列表的内容进行排序，以确保始终检索到已排序的列表。

**警告**

处理列表时存在潜在的竞争条件。如果您设置/保存整个列表，那么其他试图保存整个列表的进程也可能会覆盖更改。附加到列表的最安全方法是执行推送操作。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

*班级* **mongoengine.fields.DictField**（*字段=无，* args，** kwargs*）

包装标准 Python 字典的字典字段。这类似于嵌入式文档，但未定义结构。

**笔记**

必需意味着它不能为空 - 因为 DictFields 的默认值是 {}

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.MapField**（*字段=无，* *args，** *kwargs*）

将名称映射到指定字段类型的字段。类似于 DictField，除了每个项目的“值”必须匹配指定的字段类型。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.ReferenceField**( *document_type , dbref = False , reverse_delete_rule = 0 , ** kwargs* )

对将在访问时自动解除引用（延迟）的文档的引用。

请注意，这意味着您每次访问此字段时都将获得数据库 I/O 访问权限。这是必要的，因为该字段返回的 `Document` 精确类型取决于数据库中文档中存在的*_cls 字段的值*。简而言之，使用这种类型的字段可能会导致性能不佳（特别是如果您访问此字段只是为了检索它在取消引用之前已知的*pk 字段*）。要解决这个问题，您应该考虑使用 `LazyReferenceField`.

如果字段引用的文档被删除，请使用*reverse_delete_rule 来处理应该发生的情况*。EmbeddedDocuments、DictFields 和 MapFields 不支持 reverse_delete_rule， 如果尝试设置这些文档/字段类型之一，将引发*InvalidDocumentError* 。

选项是：

DO_NOTHING (0) - 什么都不做（默认）。

NULLIFY (1) - 更新对 null 的引用。

CASCADE (2) - 删除与引用关联的文档。

DENY (3) - 阻止删除引用对象。

`ListField` PULL (4) - 从引用中拉取引用

注册删除规则的替代语法（在实现双向删除规则时很有用）

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

初始化参考字段。

**参数:** **document_type** – 将被引用的文档类型

**dbref** – 将引用存储为 `DBRef` 或存储为 `ObjectId`.

**reverse_delete_rule** – 确定删除引用对象时要执行的操作

**kwargs——传递给父级的关键字参数** `BaseField`

**笔记**

*无论dbref* `DBRef` 的值如何，对抽象文档类型的引用始终存储为 a 。

---

*班级* **mongoengine.fields.LazyReferenceField**( *document_type , passthrough = False , dbref = False , reverse_delete_rule = 0 , ** kwargs* )

对文档的真正懒惰引用。与它不同的是， `ReferenceField` 它 **不会**在访问时自动（懒惰地）取消引用。相反，access 将返回一个 `LazyReference` 类实例，允许访问*pk*或通过使用 `fetch()` 方法手动取消引用。

初始化参考字段。

**参数:** **dbref** – 将引用存储为.id `DBRef` 或作为 `ObjectId` .id 。

**reverse_delete_rule** – 确定删除引用对象时要执行的操作

**passthrough——尝试访问未知字段时，** `LazyReference` 实例将自动调用*fetch()*并尝试检索已获取文档中的字段。请注意，这仅适用于获取字段（而不是设置或删除）。

---

*班级* **mongoengine.fields.GenericReferenceField**( **参数，** kwargs* )

对将在访问时自动取消引用（延迟）的*任何子* 类的引用。 `Document`

请注意，此字段的工作方式与 相同 `ReferenceField` ，在第一次访问时进行数据库 I/O 访问（即使是访问它 `pk` 或 `id` 字段）。要解决这个问题，您应该考虑使用 `GenericLazyReferenceField` .

**笔记**

任何用作通用参考的文件都必须在文件登记处登记。导入模型将自动注册它。
您可以使用 choices 参数来限制可接受的文档类型

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** – （可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** – （可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** – （可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.GenericLazyReferenceField**( **参数*, *** kwargs* )

*对任何子* 类的引用`Document`。与它不同的是，`GenericReferenceField`它 **不会**在访问时自动（懒惰地）取消引用。相反，access 将返回一个`LazyReference`类实例，允许访问*pk*或通过使用`fetch()`方法手动取消引用。

笔记

任何用作通用参考的文件都必须在文件登记处登记。导入模型将自动注册它。
您可以使用 choices 参数来限制可接受的文档类型

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** – （可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** – （可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** – （可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.CachedReferenceField**（*document_type*，*fields = None*，*auto_sync = True*，*** kwargs*）

带有用于伪连接的缓存字段的参考字段

初始化缓存引用字段。

**参数:** **document_type** – 将被引用的文档类型

**fields** – 要缓存在文档中的字段列表

**auto_sync** – 如果 True 文档自动更新

**kwargs——传递给父级的关键字参数** `BaseField`

---

*班级* **mongoengine.fields.BinaryField**( *max_bytes = None* , *** kwargs* )

二进制数据字段。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择——**（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.FileField**( *db_alias = 'default'* , *collection_name = 'fs'* , *** kwargs* )

一个 GridFS 存储字段。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.ImageField**（*大小=无，缩略图大小=无，collection_name = 'images'，** kwargs*）

图像文件存储字段。

**参数:** **size** – 存储图像的最大尺寸，提供为 (width, height, force) 如果更大，它将自动调整大小（例如：size=(800, 600, True)）

**thumbnail_size** – 生成缩略图的大小，提供为（宽度、高度、力）

**db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**——（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **mongoengine.fields.SequenceField**( *collection_name = None , db_alias = None , sequence_name = None , value_decorator = None , * args , ** kwargs* )

**提供顺序计数器见：**

https://docs.mongodb.com/manual/reference/method/ObjectId/#ObjectIDs-SequenceNumbers

笔记

虽然传统数据库经常使用递增的序号作为主键。在 MongoDB 中，首选方法是使用对象 ID。这个概念是，在一个非常大的机器集群中，创建一个对象 ID 比拥有全局的、均匀增加的序列号更容易。

**参数:** **collection_name** – 计数器集合的名称（默认为“mongoengine.counters”）

**sequence_name** – 集合中序列的名称（默认为“ClassName.counter”）

**value_decorator** – 任何可用作计数器的调用（默认 int）

使用任何可调用的*value_decorator*将计算的计数器转换为适合您需要的任何值，例如默认整数计数器值的字符串或十六进制表示。

笔记

如果在抽象文档中定义了计数器，它将对所有继承的文档都是通用的，默认序列名称将是抽象文档的类名。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** –（可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** –（可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** –（可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

***班级*** **`mongoengine.fields.ObjectIdField`***( db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs )*

围绕 MongoDB 的 ObjectIds 的字段包装器。

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** –（可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** – （可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** – （可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** ****kwargs** –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.UUIDField**(*二进制= True，** kwargs*)

UUID 字段。

将 UUID 数据存储在数据库中

**参数:** **二进制**- 如果 False 存储为字符串。

---

*班级* **mongoengine.fields.GeoPointField**(*db_field = None , required = False , default = None , unique = False , unique_with = None , primary_key = False , validation = None , choices = None , null = False , sparse = False , ** kwargs*)

存储经度和纬度坐标的列表。

笔记

这表示 2D 平面中的通用点和表示地理点的传统方式。它接受 2d 索引，但不接受 MongoDB > 2.4 中的“2dsphere”索引，这对于建模地理空间点更自然。请参见地理空间索引

**参数:** **db_field** – 存储该字段的数据库字段（默认为字段名称）

**required** – 如果该字段是必需的。它是否必须具有价值。默认为假。

**default** – （可选）此字段的默认值，如果未设置任何值，如果该值设置为 None 或已取消设置。它可以是可调用的。

**unique** – 字段值是否唯一（创建索引）。默认为假。

**unique_with** – （可选）此字段应该唯一的其他字段（创建索引）。

**primary_key** – 将此字段标记为主键（（创建索引））。默认为假。

**validation** – （可选）可调用以验证字段的值。可调用对象将值作为参数，如果验证失败则应引发 ValidationError

**选择**––（可选）有效的选择

**null** – （可选）如果存在默认值时字段值可以为 null。如果不设置，默认值

将在具有默认值的字段设置为 None 的情况下使用。默认为假。:param sparse: (可选) *sparse=True*结合*unique=True*和*required=False*

意味着不会对*None*值强制执行唯一性（创建索引）。默认为假。

**参数:** **kwargs –

（可选）此字段的任意无间接寻址元数据可以作为附加关键字参数提供并作为字段属性访问。不得与任何现有属性冲突。常见的元数据包括*verbose_name*和 *help_text*。

---

*班级* **mongoengine.fields.PointField**( *auto_index = True , * args , ** kwargs* )

存储经度和纬度坐标的 GeoJSON 字段。

数据表示为：

```
{'type' : 'Point' ,
 'coordinates' : [x, y]}
```

您可以传递包含完整信息的字典或列表来设置值。

要求 mongodb >= 2.4

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

---

*班级* **mongoengine.fields.LineStringField**( *auto_index = True , * args , ** kwargs* )

存储一行经度和纬度坐标的 GeoJSON 字段。

数据表示为：

```
{'type' : 'LineString' ,
 'coordinates' : [[x1, y1], [x2, y2] ... [xn, yn]]}
```

您可以传递包含完整信息或点列表的字典。

要求 mongodb >= 2.4

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

---

*班级* **mongoengine.fields.PolygonField**( *auto_index = True , * args , ** kwargs* )

存储经度和纬度坐标的多边形的 GeoJSON 字段。

数据表示为：

```
{'type' : 'Polygon' ,
 'coordinates' : [[[x1, y1], [x1, y1] ... [xn, yn]],
                  [[x1, y1], [x1, y1] ... [xn, yn]]}
```

您可以传递包含完整信息的字典或 LineStrings 列表。第一个 LineString 是外面，其余的是孔。

要求 mongodb >= 2.4

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

---

*班级* **mongoengine.fields.MultiPointField**( *auto_index = True , * args , ** kwargs* )

存储点列表的 GeoJSON 字段。

数据表示为：

```
{'type' : 'MultiPoint' ,
 'coordinates' : [[x1, y1], [x2, y2]]}
```

您可以传递包含完整信息的字典或列表来设置值。

要求 mongodb >= 2.6

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

*班级* **mongoengine.fields.MultiLineStringField**( *auto_index = True , * args , ** kwargs* )

存储 LineString 列表的 GeoJSON 字段。

数据表示为：

```
{'type' : 'MultiLineString' ,
 'coordinates' : [[[x1, y1], [x1, y1] ... [xn, yn]],
                  [[x1, y1], [x1, y1] ... [xn, yn]]]}
```

您可以传递包含完整信息或点列表的字典。

要求 mongodb >= 2.6

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

*班级* **mongoengine.fields.MultiPolygonField**( *auto_index = True , * args , ** kwargs* )

存储多边形列表的 GeoJSON 字段。

数据表示为：

```
{'type' : 'MultiPolygon' ,
 'coordinates' : [[
       [[x1, y1], [x1, y1] ... [xn, yn]],
       [[x1, y1], [x1, y1] ... [xn, yn]]
   ], [
       [[x1, y1], [x1, y1] ... [xn, yn]],
       [[x1, y1], [x1, y1] ... [xn, yn]]
   ]
}
```

您可以传递包含完整信息的字典或多边形列表。

要求 mongodb >= 2.6

**参数:** **auto_index** ( *bool* ) – 自动创建一个“2dsphere”索引。默认为*True*。

*班级* **mongoengine.fields.GridFSError**

*班级* **mongoengine.fields.GridFSProxy**( *grid_id = None , key = None , instance = None , db_alias = 'default' , collection_name = 'fs'* )

代理对象，用于处理向 GridFS 写入和读取文件

*班级* **mongoengine.fields.ImageGridFsProxy**( *grid_id = None , key = None , instance = None , db_alias = 'default' , collection_name = 'fs'* )

ImageField 的代理

*班级* **mongoengine.fields.ImproperlyConfigured**

## 3.6. 嵌入式文档查询

*0.9 版中的新功能。*

当使用 存储嵌入文档列表时，可以使用对 `EmbeddedDocumentListField` 嵌入文档的附加查询。

使用以下方法将嵌入文档列表作为特殊列表返回：

---

***班级*** **mongoengine.base.datastructures.EmbeddedDocumentList** ***（列表项、实例、名称）***

**count**()

列表中嵌入文档的数量。

**退货:** 列表的长度，相当于 的结果 `len()` 。

**create**(***值*)

创建 EmbeddedDocument 的新实例并将其附加到此 EmbeddedDocumentList。

笔记

EmbeddedDocument 的实例不会自动保存到数据库中。您仍然需要在父文档上调用 .save() 。

**参数:** **values** – 嵌入文档的值字典。

**退货:** 新的嵌入式文档实例。

**delete**()

从数据库中删除嵌入的文档。

笔记

调用此方法后，嵌入式文档的更改不会自动保存到数据库中。

**退货:** 删除的条目数。

**exclude**(***夸格斯*)

通过排除具有给定关键字参数的嵌入文档来过滤列表。

**参数:** **kwargs** – 与要排除的字段对应的关键字参数。*多个参数被视为好像它们被 AND 运算在一起。*

**退货:** `EmbeddedDocumentList` 包含不匹配嵌入文档的新文件。

`AttributeError` 如果给定关键字不是嵌入文档类的有效字段，则引发。

**filter**(***夸格斯*)

通过仅包含具有给定关键字参数的嵌入文档来过滤列表。

此方法仅支持简单比较（例如.filter(name='John Doe')），不支持像queryset.filter 那样的 __gte、__lte、__icontains 等运算符

**参数:** **kwargs** – 与要过滤的字段对应的关键字参数。*多个参数被视为好像它们被 AND 运算在一起。*

**退货:** `EmbeddedDocumentList` 包含匹配嵌入文档的新文件。

`AttributeError` 如果给定关键字不是嵌入文档类的有效字段，则引发。

**first**()

返回列表中的第一个嵌入文档，或者 `None` 如果为空。

**get**(***夸格斯*)

检索由给定关键字参数确定的嵌入文档。

**参数:** **kwargs** – 与要搜索的字段对应的关键字参数。*多个参数被视为好像它们被 AND 运算在一起。*

**退货:** 与给定关键字参数匹配的嵌入文档。

`DoesNotExist` 如果用于查询嵌入文档的参数未返回任何结果，则引发。
`MultipleObjectsReturned` 如果返回多个结果。

**save**(**参数，** kwargs*)

保存祖先文件。

**参数:** **args** – 传递给祖先 Document 的保存方法的参数。

**kwargs** – 传递给祖先文档的保存方法的关键字参数。

**update**(***更新*)

使用给定的替换值更新嵌入文档。此函数不支持 mongoDB 更新操作符，例如 `inc__`.

笔记

调用此方法后，嵌入式文档的更改不会自动保存到数据库中。

**参数:** **update** – 应用于每个嵌入文档的更新值字典。

**退货:** 更新的条目数。

## 3.7. 杂项

**mongoengine.common._import_class**(*cls_name*)

导入缓存机制。

由于循环导入的复杂性，mongoengine 需要在函数中做大量的内联导入。这是低效的，因为在整个 mongoengine 代码中重复导入类。一些需要内联导入的递归函数使情况更加复杂。

`mongoengine.common` 提供一个点来导入所有这些类。循环导入不是问题，因为它会在第一次需要时动态导入类。对 的后续调用 `_import_class()` 可以直接从 中检索类 `mongoengine.common._class_registry_cache`。