Django查询数据库操作详解（一）

在 web 开发过程中，Django 与后台数据库的交互是必不可少的一项，也是实现业务逻辑所需数据的重要方式，所以 Django 的表查询操作就显的尤为重要。在本节我们将全面阐述 Django 的表查询 API。本节知识属于重中之重，希望各位小伙伴能够尽可能的掌握这些 API，懂得活学活用，熟悉每个 API 的使用场景，这将对后续学习 Django 框架有很大的帮助。

1. 返回单条查询结果

我们知道 Manager 查询管理器提供了查询 Model 实例的接口，这些接口通常会返回三种类型：单实例、RawQuerySet、QuerySet。通常的实际的开发工作中，我们会根据给定的条件查询数据库记录。 Django 为实现返回单条查询数据提供了两个查询方法 get 和 get_or_create。

get_or_create方法和 get 区别在于，当被查询数据不存在的时候，get_or_create 方法会创建新的实例对象，而 get 方法会抛出 DoesNotExist异常。而当这两个方法的查询条件都能够匹配多条数据记录时，都会抛出MultipleObjectsReturned异常。

提示：这两个方法都只能返回一条数据。

1) 使用get查询

因为 get 查询可能会抛出异常，所以若在项目使用 get 查询的时，经常会与 try..except 异常处理语句一起使用。示例如下：

```
#使用 Django shell
Author.objects.create(name="Tom",email="456789@163.com") #添加 Tom 此时数据表有两个Tom
Author.objects.get(name="Tom") #查询 name="Tom",就会报错
#报错信息如下
MultipleObjectsReturned: get() returned more than one Author -- it returned 2!
#查询不存在数据
Author.objects.get(id=4)
#报错信息如下：
DoesNotExist：Traceback (most recent call last)
```

2) 使用get_or_create查询

该方法的查询过程与 get 类似，都需要传递查询参数，但是不 get 不同的是，它返回值是一个 tuple 对象，通过举例子来说明。

```
Author.objects.get_or_create(name="Xiaolong")#先查询是否存在若不存在则新建该实例对象
(<Author: 作家：Xiaolong>, True) #返回值是一个元组有两个元素
```

元组中的第一个元素代表实例对象，第二个元素是布尔值，标识返回的实例对象是否是新创建的，其中 True 代表新建实例对象，False 代表原有实例对象。

查询结果为单实例的类似方法还有 first、last 等方法，用法与语法格式别无二致，在这里就不加赘述了。

2. 原生数据库操作方法

所谓原生数据库操作即使用 SQL 语句来进行数据库的相关的查询操作。那你可能会问，Django 已经有 ORM 了，为什么还会提供原生语句的操作呢？其实这个问题也很容易想明白，ORM 虽然为我解决了大部分的有业务场景，但是对于复杂的查询来说，ORM 还是有点力不从心，有时无法满足开发者的要求或者书写起来比相较 SQL 语句来说很麻烦，那么在这个时候使用原生 SQL 语解决问题就是不错的选择。

在 Django 中，可以使用模型管理器（Marager）的 raw 方法来执行 select 语句进行数据的查询。raw 方法的返回值是一个 RawQuerySet 对象，该对象支持索引和切片，同样也可以对它进行迭代得到 Model 实例对象。但是，与 QuerySet 不同的是，它不能执行 fillter、exclude 等方法。下面通过举例说明 raw 方法如何使用。

1) raw方法语法格式

使用 raw 方法的语法格式如下所示：

```
MyModel.objects.raw('sql语句')
```

2) raw方法查询实例

添加如下代码在index¥views.py 中：

```
def BookName(request):
    books=Book.objects.raw("select * from index_book") #书写sql语句
    return render(request,"index/allbook.html",locals())
```

在 index¥templates¥index 中创建 allbook.html，代码如下：

```
{% for book in books %}
<p>{{book.title}}</p>
{% endfor %}
```

最后配置路由，通过访问 127.0.0.1:8000/index/allbook/，可以得到所有书籍的 title。

3) raw方法与SQL注入

在很多场景下，我们需要使用参数化查询，比如根据传递的 title 查询 book 对象，在这个时候我们不能手动填充 SQL 字符串，这会带来 SQL 注入的风险，raw 方法充分考虑到这一点，提供了 params 参数来解决这个问题。对于参数化查询的 SQL 来说，只需要在语句中加上占位符号（%s）即可。如下所示：

```
authors=Author.objects.raw("select id from index_author where name= %s",['Tom'])
    ...: for author in authors:
    ...:     print('%s:%s'%(author.id,author.name))
2:Tom #返回结果
```

raw 方法可以接受一个列表或字典类型的参数，并将 SQL 语句中的占位符替换，最终完成查询。

4) 游标cursor执行SQL语句

那么执行非查询语句呢？如何使用原生的 SQL 语句呢？ Django 提供了游标 cursor 对数据库进行增删改操作，在 Django 中执行非查询语句必须使用游标进行操作。游标 cursor 定义在 django.db.connection 包中，使用前用下面的方式进行导入：

```
from django.db import connection
```

用创建 cursor 类的构造函数创建 cursor 对象，再使用 cursor 对象执行 SQL 语句。为确保能够在出现异常时释放 cursor 游标，通常使用 with 语句进行创建操，如下所示：

```
from django.db import connection
with connection.cursor() as cur:
    cur.execute('执行SQL语句')
```

使用示例如下：

```
from django.db import connection
with connection.cursor() as cur:
#调用游标对象的execute方法, 更新author的名字
cur.execute('update index_author set name="Jack" where id=3;')
with connection.cursor() as cur:
# 删除id为3的一条author记录
cur.execute('delete from index_author where id=3;')
```

虽然 Django 提供了 raw SQL的查询方法以及 cursor 游标对象的使用，但是它们很少的被使用，应为这些方法都需要考虑不同数据库的不同特性。Django 官网建议我们尽量使用 ORM 模块来完成相关操作。