3-创建 Django 博客的数据库模型

1、设计博客的数据库表结构

博客最主要的功能就是展示我们写的文章，它需要从某个地方获取博客文章数据才能把文章展示出来，通常来说这个地方就是数据库。我们把写好的文章永久地保存在数据库里，当用户访问我们的博客时，Django 就去数据库里把这些数据取出来展现给用户。

博客的文章应该含有标题、正文、作者、发表时间等数据。一个更加现代化的博客文章还希望它有分类、标签、评论等。为了更好地存储这些数据，我们需要合理地组织数据库的表结构。

我们的博客初级版本主要包含博客文章，文章会有分类以及标签。一篇文章只能有一个分类，但可以打上很多标签。

数据库存储的数据其实就是表格的形式，例如存储博客文章的数据库表长这个样子：

文章 id	标题	正文	发表时间	分类	标签
1	title 1	text 1	2016-12-23	Django	Django 学习
2	title 2	text 2	2016-12-24	Django	Django 学习
3	title 3	text 3	2016-12-26	Python	Python 学习

多对多的关系

一对多的关系

其中文章 ID 是一个数字，唯一对应着一篇文章。当然还可以有更多的列以存储更多相关数据，这只是一个最基本的示例。

数据库表设计成这样其实已经可以了，但是稍微分析一下我们就会发现一个问题，这 3 篇文章的分类和标签都是相同的，这会产生很多重复数据，当数据量很大时就浪费了存储空间。

不同的文章可能它们对应的分类或者标签是相同的，所以我们把分类和标签提取出来，做成单独的数据库表，再把文章和分类、标签关联起来。下面分别是分类和标签的数据库表：

分类 id	分类名

1	Django
2	Python

标签 id	标签名
1	Django 学习
2	Python 学习

2、编写博客模型代码

以上是自然语言描述的表格，数据库也和编程语言一样，有它自己的一套规定的语法来生成上述的表结构，这样我们才能把数据存进去。一般来说这时候我们应该先去学习数据库创建表格的语法，再回来写我们的 Django 博客代码了。但是 Django 告诉我们不用这么麻烦，它已经帮我们做了一些事情。Django 把那一套数据库的语法转换成了 Python 的语法形式，我们只要写 Python 代码就可以了，Django 会把 Python 代码翻译成对应的数据库操作语言。用更加专业一点的说法，就是 Django 为我们提供了一套 ORM（Object Relational Mapping）系统。

例如我们的分类数据库表，Django 只要求我们这样写：

【blog/models.py】

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=100)
```

`Category` 就是一个标准的 Python 类，它继承了 `models.Model` 类，类名为 `Category` 。`Category` 类有一个属性 `name`，它是 `models.CharField` 的一个实例。

这样，Django 就可以把这个类翻译成数据库的操作语言，在数据库里创建一个名为 category 的表格，这个表格的一个列名为 name。还有一个列 id，Django 则会自动创建。可以看出从 Python 代码翻译成数据库语言时其规则就是一个 Python 类对应一个数据库表格，类名即表名，类的属性对应着表格的列，属性名即列名。

我们需要 3 个表格：文章（Post）、分类（Category）以及标签（Tag），下面就来分别编写它们对应的 Python 类。模型的代码通常写在相关应用的 models.py

文件里。已经在代码中做了详细的注释，说明每一句代码的含义。但如果你在移动端下阅读不便的话，也可以跳到代码后面看正文的里的讲解。

注意：代码中含有中文注释，如果你直接 copy 代码到你的文本编辑器且使用了 Python 2 开发环境的话，会得到一个编码错误。因此请在文件最开始处加入编码声明：# coding: utf-8。

【blog/models.py】

```
from django.db import models
from django.contrib.auth.models import User

class Category(models.Model):
    """
    Django 要求模型必须继承 models.Model 类。
    Category 只需要一个简单的分类名 name 就可以了。
    CharField 指定了分类名 name 的数据类型，CharField 是字符型，
    CharField 的 max_length 参数指定其最大长度，超过这个长度的分类名就不能被存入数据库。
    当然 Django 还为我们提供了多种其它的数据类型，如日期时间类型 DateTimeField、整数类型 IntegerField 等等。
    Django 内置的全部类型可查看文档：
    https://docs.djangoproject.com/en/1.10/ref/models/fields/#field-types
    """
    name = models.CharField(max_length=100)

class Tag(models.Model):
    """
    标签 Tag 也比较简单，和 Category 一样。
    再次强调一定要继承 models.Model 类！
    """
    name = models.CharField(max_length=100)

class Post(models.Model):
    """
    文章的数据库表稍微复杂一点，主要是涉及的字段更多。
    """

    # 文章标题
    title = models.CharField(max_length=70)

    # 文章正文，我们使用了 TextField。注意不需要加参数
    # 存储比较短的字符串可以使用 CharField，但对于文章的正文来说可能会是一大段文本，因此使用 TextField 来存储大段文本。
    body = models.TextField()

    # 这两个列分别表示文章的创建时间和最后一次修改时间，存储时间的字段用 DateTimeField 类型。
```

```
    created_time = models.DateTimeField()
    modified_time = models.DateTimeField()

    # 文章摘要，可以没有文章摘要，但默认情况下 CharField 要求我们必须存入数据，否则就会报错。
    # 指定 CharField 的 blank=True 参数值后就可以允许空值了。
    excerpt = models.CharField(max_length=200, blank=True)

    # 这是分类与标签，分类与标签的模型我们已经定义在上面。
    # 我们在这里把文章对应的数据库表和分类、标签对应的数据库表关联了起来，但是关联形式稍微有点不同。
    # 我们规定一篇文章只能对应一个分类，但是一个分类下可以有多篇文章，所以我们使用的是 ForeignKey，即一对多的关联关系。
    # 而对于标签来说，一篇文章可以有多个标签，同一个标签下也可能有多篇文章，所以我们使用 ManyToManyField，表明这是多对多的关联关系。
    # 同时我们规定文章可以没有标签，因此为标签 tags 指定了 blank=True。
    # 如果你对 ForeignKey、ManyToManyField 不了解，请看教程中的解释，亦可参考官方文档：
    # https://docs.djangoproject.com/en/1.10/topics/db/models/#relationships
    category = models.ForeignKey(Category)
    tags = models.ManyToManyField(Tag, blank=True)

    # 文章作者，这里 User 是从 django.contrib.auth.models 导入的。
    # django.contrib.auth 是 Django 内置的应用，专门用于处理网站用户的注册、登录等流程，User 是 Django 为我们已经写好的用户模型。
    # 这里我们通过 ForeignKey 把文章和 User 关联了起来。
    # 因为我们规定一篇文章只能有一个作者，而一个作者可能会写多篇文章，因此这是一对多的关联关系，和 Category 类似。
    author = models.ForeignKey(User)
```

3、博客模型代码代码详解

首先是 Category 和 Tag 类，它们均继承自 model.Model 类，这是 Django 规定的。Category 和 Tag 类均有一个 name 属性，用来存储它们的名称。由于分类名和标签名一般都是用字符串表示，因此我们使用了 CharField 来指定 name 的数据类型，同时 max_length 参数则指定 name 允许的最大长度，超过该长度的字符串将不允许存入数据库。除了 CharField ，Django 还为我们提供了更多内置的数据类型，比如时间类型 DateTimeField、整数类型 IntegerField 等等。

在本教程中我们会教你这些类型的使用方法，但以后你开发自己的项目时，你就需要通过阅读 Django 官方文档 关于字段类型的介绍 来了解有哪些数据类型可以使用以及如何使用它们。

Post 类也一样，必须继承自 model.Model 类。文章的数据库表稍微复杂一点，主要是列更多，我们指定了这些列：

  - title。这是文章的标题，数据类型是 CharField，允许的最大长度 max_length = 70。
  - body。文章正文，我们使用了 TextField。比较短的字符串存储可以使用 CharField，但对于文章的正文来说可能会是一大段文本，因此使用 TextField 来存储大段文本。
  - created_time、modified_time。这两个列分别表示文章的创建时间和最后一次修改时间，存储时间的列用 DateTimeField 数据类型。
  - excerpt。文章摘要，可以没有文章摘要，但默认情况下 CharField 要求我们必须存入数据，否则就会报错。指定 CharField 的 blank=True 参数值后就可以允许空值了。

- category 和 tags。这是分类与标签，分类与标签的模型我们已经定义在上面。我们把文章对应的数据库表和分类、标签对应的数据库表关联了起来，但是关联形式稍微有点不同。我们规定一篇文章只能对应一个分类，但是一个分类下可以有多篇文章，所以我们使用的是 ForeignKey，即一对多的关联关系。而对于标签来说，一篇文章可以有多个标签，同一个标签下也可能有多篇文章，所以我们使用 ManyToManyField，表明这是多对多的关联关系。同时我们规定文章可以没有标签，因此为标签 tags 指定了 blank=True。

- author。文章作者，这里 User 是从 django.contrib.auth.models 导入的。django.contrib.auth 是 Django 内置的应用，专门用于处理网站用户的注册、登录等流程。其中 User 是 Django 为我们已经写好的用户模型，和我们自己编写的 Category 等类是一样的。这里我们通过 ForeignKey 把文章和 User 关联了起来，因为我们规定一篇文章只能有一个作者，而一个作者可能会写多篇文章，因此这是一对多的关联关系，和 Category 类似。

4、理解多对一和多对多两种关联关系

我们分别使用了两种关联数据库表的形式：ForeignKey 和 ManyToManyField。

ForeignKey

`ForeignKey` 表明一种一对多的关联关系。比如这里我们的文章和分类的关系，一篇文章只能对应一个分类，而一个分类下可以有多篇文章。反应到数据库表格中，它们的实际存储情况是这样的：

文章 ID	标题	正文	分类 ID
1	title 1	body 1	1
2	title 2	body 2	1
3	title 3	body 3	1
4	title 4	body 4	2

分类 ID	分类名
1	Django
2	Python

可以看到文章和分类实际上是通过文章数据库表中 分类 ID 这一列关联的。当要查询文章属于哪一个分类时，只需要查看其对应的分类 ID 是多少，然后根据这个分类 ID 就可以从分类数据库表中找到该分类的数据。例如这里文章 1、2、3 对应的分类 ID 均为 1，而分类 ID 为 1 的分类名为 Django，所以文章 1、2、3 属于分类 Django。同理文章 4 属于分类 Python。

反之，要查询某个分类下有哪些文章，只需要查看对应该分类 ID 的文章有哪些即可。例如这里 Django 的分类 ID 为 1，而对应分类 ID 为 1 的文章有文章 1、2、3，所以分类 Django 下有 3 篇文章。

希望这个例子能帮助你加深对多对一关系，以及它们在数据库中是如何被关联的理解，更多的例子请看文末给出的 Django 官方参考资料。

ManyToManyField

`ManyToManyField` 表明一种多对多的关联关系，比如这里的文章和标签，一篇文章可以有多个标签，而一个标签下也可以有多篇文章。反应到数据库表格中，它们的实际存储情况是这样的：

文章 ID	标题	正文
1	title 1	body 1
2	title 2	body 2

3	title 3	body 3
4	title 4	body 4

标签 ID	标签名
1	Django 学习
2	Python 学习

文章 ID	标签 ID
1	1
1	2
2	1
3	2

多对多的关系无法再像一对多的关系中的例子一样在文章数据库表加一列 分类 ID 来关联了，因此需要额外建一张表来记录文章和标签之间的关联。例如文章 ID 为 1 的文章，既对应着 标签 ID 为 1 的标签，也对应着 标签 ID 为 2 的标签，即文章 1 既属于标签 1：Django 学习，也属于标签 2：Python 学习。

反之，标签 ID 为 1 的标签，既对应着 文章 ID 为 1 的文章，也对应着 文章 ID 为 2 的文章，即标签 1：Django 学习下有两篇文章。

希望这个例子能帮助你加深对多对多关系，以及它们在数据库中是如何被关联的理解，更多的例子请看文末给出的 Django 官方参考资料。

假如你对多对一关系和多对多关系还存在一些困惑，强烈建议阅读官方文档对这两种关系的说明以及更多官方的例子以加深理解：

- Django ForeignKey 简介
- Django ForeignKey 详细示例
- Django ManyToManyField 简介
- Django ManyToManyField 详细示例