Day 4 - 编写Model

363次阅读

有了ORM，我们就可以把Web App需要的3个表用Model表示出来：

```
import time, uuid

from transwarp.db import next_id
from transwarp.orm import Model, StringField, BooleanField, FloatField, TextField

class User(Model):
    __table__ = 'users'

    id = StringField(primary_key=True, default=next_id, ddl='varchar(50)')
    email = StringField(updatable=False, ddl='varchar(50)')
    password = StringField(ddl='varchar(50)')
    admin = BooleanField()
    name = StringField(ddl='varchar(50)')
    image = StringField(ddl='varchar(500)')
    created_at = FloatField(updatable=False, default=time.time)

class Blog(Model):
    __table__ = 'blogs'

    id = StringField(primary_key=True, default=next_id, ddl='varchar(50)')
    user_id = StringField(updatable=False, ddl='varchar(50)')
    user_name = StringField(ddl='varchar(50)')
    user_image = StringField(ddl='varchar(500)')
    name = StringField(ddl='varchar(50)')
    summary = StringField(ddl='varchar(200)')
    content = TextField()
    created_at = FloatField(updatable=False, default=time.time)

class Comment(Model):
    __table__ = 'comments'

    id = StringField(primary_key=True, default=next_id, ddl='varchar(50)')
    blog_id = StringField(updatable=False, ddl='varchar(50)')
    user_id = StringField(updatable=False, ddl='varchar(50)')
    user_name = StringField(ddl='varchar(50)')
    user_image = StringField(ddl='varchar(500)')
    content = TextField()
    created_at = FloatField(updatable=False, default=time.time)
```

在编写ORM时，给一个Field增加一个default参数可以让ORM自己填入缺省值，非常方便。并且，缺省值可以作为函数对象传入，在调用insert()时自动计算。

例如，主键id的缺省值是函数next_id，创建时间created_at的缺省值是函数time.time，可以自动设置当前日期和时间。

日期和时间用float类型存储在数据库中，而不是datetime类型，这么做的好处是不必关心数据库的时区以及时区转换问题，排序非常简单，显示的时候，只需要做一个float到str的转换，也非常容易。

初始化数据库表

如果表的数量很少，可以手写创建表的SQL脚本：

```
-- schema.sql
```

```
drop database if exists awesome;

create database awesome;

use awesome;

grant select, insert, update, delete on awesome.* to 'www-data'@'localhost' identified by 'www-data';

create table users (
    `id` varchar(50) not null,
    `email` varchar(50) not null,
    `password` varchar(50) not null,
    `admin` bool not null,
    `name` varchar(50) not null,
    `image` varchar(500) not null,
    `created_at` real not null,
    unique key `idx_email` (`email`),
    key `idx_created_at` (`created_at`),
    primary key (`id`)
) engine=innodb default charset=utf8;

create table blogs (
    `id` varchar(50) not null,
    `user_id` varchar(50) not null,
    `user_name` varchar(50) not null,
    `user_image` varchar(500) not null,
    `name` varchar(50) not null,
    `summary` varchar(200) not null,
    `content` mediumtext not null,
    `created_at` real not null,
    key `idx_created_at` (`created_at`),
    primary key (`id`)
) engine=innodb default charset=utf8;

create table comments (
    `id` varchar(50) not null,
    `blog_id` varchar(50) not null,
    `user_id` varchar(50) not null,
    `user_name` varchar(50) not null,
    `user_image` varchar(500) not null,
    `content` mediumtext not null,
    `created_at` real not null,
    key `idx_created_at` (`created_at`),
    primary key (`id`)
) engine=innodb default charset=utf8;
```

如果表的数量很多，可以从Model对象直接通过脚本自动生成SQL脚本，使用更简单。

把SQL脚本放到MySQL命令行里执行：

```
$ mysql -u root -p < schema.sql
```

我们就完成了数据库表的初始化。

编写数据访问代码

接下来，就可以真正开始编写代码操作对象了。比如，对于User对象，我们就可以做如下操作：

```
# test_db.py

from models import User, Blog, Comment
```

```
from transwarp import db

db.create_engine(user='www-data', password='www-data', database='awesome')

u = User(name='Test', email='test@example.com', password='1234567890', image='about:blank')

u.insert()

print 'new user id:', u.id

u1 = User.find_first('where email=?', 'test@example.com')
print 'find user\'s name:', u1.name

u1.delete()

u2 = User.find_first('where email=?', 'test@example.com')
print 'find user:', u2
```

可以在MySQL客户端命令行查询，看看数据是不是正常存储到MySQL里面了。
