Day 3 - 编写ORM

687次阅读

有了db模块，操作数据库直接写SQL就很方便。但是，我们还缺少ORM。如果有了ORM，就可以用类似这样的语句获取User对象：

```
user = User.get('123')
```

而不是写SQL然后再转换成User对象：

```
u = db.select_one('select * from users where id=?', '123')
user = User(**u)
```

所以我们开始编写ORM模块：transwarp.orm。

设计ORM接口

和设计db模块类似，设计ORM也是从上层调用者角度来设计。

我们先考虑如何定义一个User对象，然后把数据库表users和它关联起来。

```
from transwarp.orm import Model, StringField, IntegerField

class User(Model):
    __table__ = 'users'
    id = IntegerField(primary_key=True)
    name = StringField()
```

注意到定义在User类中的__table__、id和name是类的属性，不是实例的属性。所以，在类级别上定义的属性用来描述User对象和表的映射关系，而实例属性必须通过__init__()方法去初始化，所以两者互不干扰：

```
# 创建实例:
user = User(id=123, name='Michael')
# 存入数据库:
user.insert()
```

实现ORM模块

有了定义，我们就可以开始实现ORM模块。

首先要定义的是所有ORM映射的基类Model：

```
class Model(dict):
    __metaclass__ = ModelMetaclass

    def __init__(self, **kw):
        super(Model, self).__init__(**kw)

    def __getattr__(self, key):
        try:
            return self[key]
        except KeyError:
            raise AttributeError(r"'Dict' object has no attribute '%s'" % key)

    def __setattr__(self, key, value):
        self[key] = value
```

Model从dict继承，所以具备所有dict的功能，同时又实现了特殊方法__getattr__()和__setattr__()，所以又可以像引用普通字段那样写：

```
>>> user['id']
123
>>> user.id
123
```

Model只是一个基类，如何将具体的子类如User的映射信息读取出来呢？答案就是通过metaclass：ModelMetaclass：

```
class ModelMetaclass(type):
    def __new__(cls, name, bases, attrs):
        mapping = ... # 读取cls的Field字段
        primary_key = ... # 查找primary_key字段
        __table__ = cls.__talbe__ # 读取cls的__table__字段
        # 给cls增加一些字段：
        attrs['__mapping__'] = mapping
        attrs['__primary_key__'] = __primary_key__
        attrs['__table__'] = __table__
        return type.__new__(cls, name, bases, attrs)
```

这样，任何继承自Model的类（比如User），会自动通过ModelMetaclass扫描映射关系，并存储到自身的class中。

然后，我们往Model类添加class方法，就可以让所有子类调用class方法：

```
class Model(dict):

    ...

    @classmethod
    def get(cls, pk):
        d = db.select_one('select * from %s where %s=?' % (cls.__table__, cls.__primary_key__.name), pk)
        return cls(**d) if d else None
```

User类就可以通过类方法实现主键查找：

```
user = User.get('123')
```

往Model类添加实例方法，就可以让所有子类调用实例方法：

```
class Model(dict):

    ...

    def insert(self):
        params = {}
        for k, v in self.__mappings__.iteritems():
            params[v.name] = getattr(self, k)
        db.insert(self.__table__, **params)
        return self
```

这样，就可以把一个User实例存入数据库：

```
user = User(id=123, name='Michael')
user.insert()
```

最后一步是完善ORM，对于查找，我们可以实现以下方法：

- find_first()

- find_all()
- find_by()

对于count，可以实现：

- count_all()
- count_by()

以及update()和delete()方法。

最后看看我们实现的ORM模块一共多少行代码？加上注释和doctest才仅仅300多行。用Python写一个ORM是不是很容易呢？
