软件详细设计

CS3604 第二组

我们要设计的项目为“校园兴趣爱好分享社区”，一个论坛网站。我们的开发语言为 Python，并使用了 Flask 作为框架。为了保证代码的可靠性和程序的可复用性，我们使用 GoF 设计模式对软件进行详细设计。下面列出了 Façade 和 Bridge 两种设计模式在系统中的使用。

一、 Façade 模式用于数据库操作的封装

数据库的增删改查操作是论坛网站后端开发的重要部分。比如用户发布新帖子时，数据库中将新建相关帖子的记录；用户删除帖子时，记录也将删除；用户的点赞、评论等行为将会导致对“点赞数”、“评论数”等字段的修改；每当用户点击浏览某个帖子，后端都需要从链接中解析该帖子的 id，在数据库中查找对应帖子的内容等信息，再加载到页面中。

我们使用 Python 库 SQLAlchemy，连接 SQLite 数据库，以进行数据库操作。如果使用 SQLAlchemy 的原生语法，涉及到对 Model 类、Session 对象、Query 对象等的操作，比如对 User 数据库的操作可能用到如下代码：

```
db = SQLAlchemy()              # 创建 SQLAlchemy 对象
class User(db.Model): …        # 定义 User 数据库
user=User.query.get(user_id) # 通过 user_id 查询用户
user.username='new_name'       # 修改 username 字段
db.session.add(new_user)       # 新增用户
db.session.commit()            # 提交当前更改
```

可画出数据库子系统的类图（简单起见，这里只画出了 User, Post, Notification 三个数据库，并且只写出了 get、add、commit 操作相关方法）：

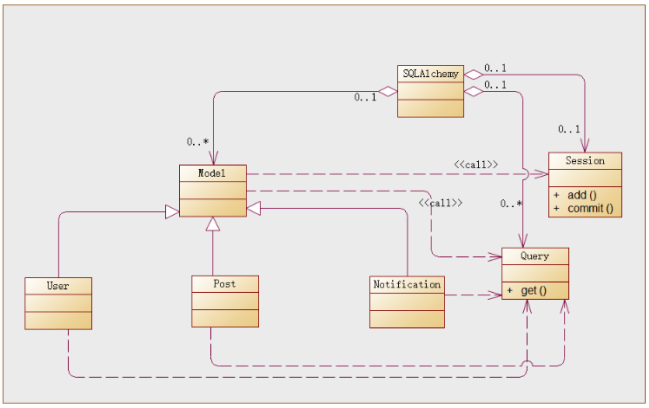

图 1：数据库子系统的类图

从图 1 可以看出，SQLAlchemy 对象与许多类有关联，而本项目又涉及到相当多的数据库，子系统较为复杂。然而做点赞、评论、通知等逻辑处理的处理时，开发者只需要知道当前该对数据库做什么操作，而并不用知道 session、query 等的存在。因此，考虑使用 Façade 模式提供一个简单接口 Database，用户通过 Database 就可访问数据库子系统。

使用 Façade 模式后，画出类图如下：

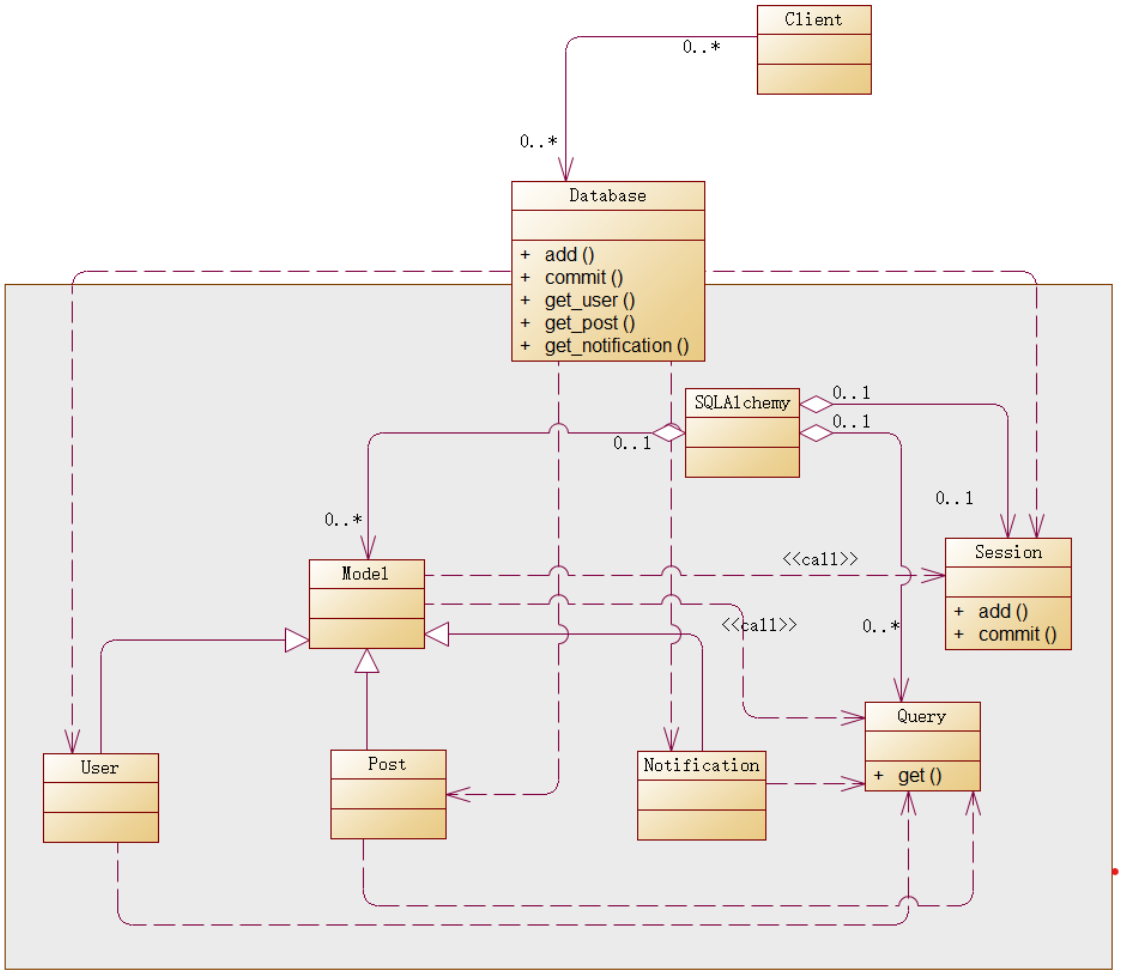

图 2：Façade 模式用于数据库子系统

在使用 Façade 模式之后，用户可以只通过实例化 Database 对象，再调用 add(), commit(), get_user()等方法来访问和操作数据库，大大降低了代码复杂度。此外，在开发阶段，尤其是测试时，常常涉及对数据库的批量操作，比如“删除帖子 11、14、16”，也可以在 Database 类中增加方法 delete_post_by_group() 等，提高了代码的可重用性。

二、 用 Bridge 模式设计“动作——作用对象”的处理

网站用户在使用网站浏览帖子时，可能产生多种“动作”(Action)，比如“点赞”(Like)、“评论”(Reply)、“删除”(Delete)、“举报”(Report)等。而这些 Action 的“作用对象”(Object)又有两类，“主帖”(Post) 和“回帖”(Comment)。不同的 Action 和 Object 的组合，将使用不同的处理逻辑。因此，我们把 Action 看作抽象类，把 Object 看作行为实现类，将二者分离，使用 Bridge 模式进行设计。

下面以点赞、评论、举报三种 Action 为例，画出 Bridge 模式的类图：

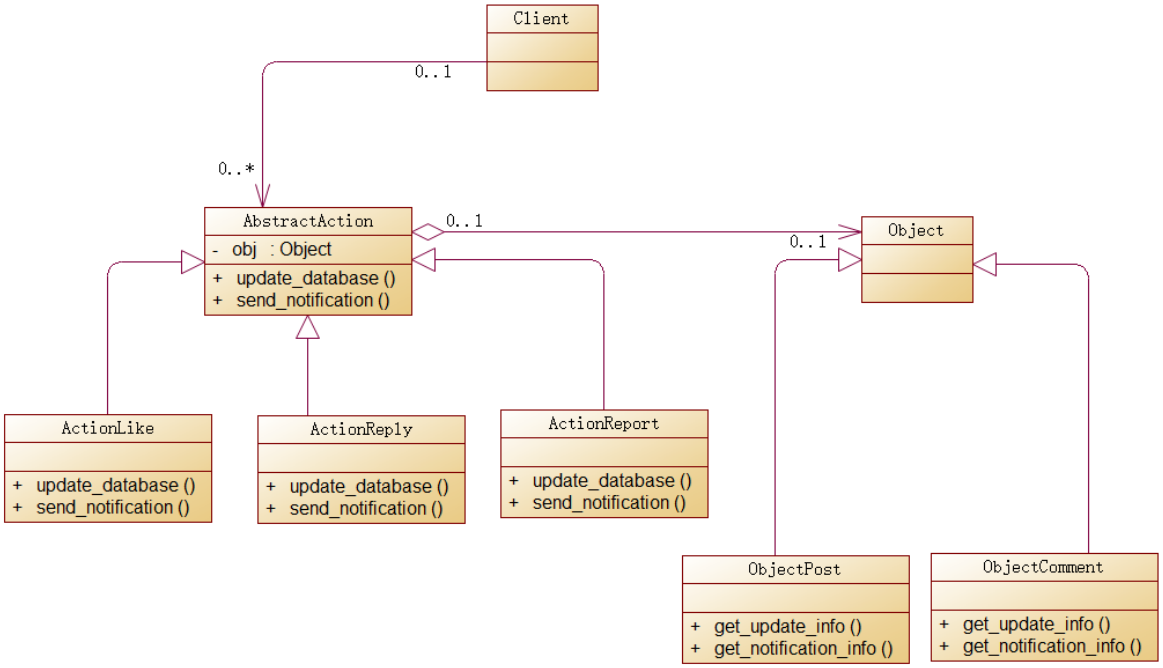

图 3：Bridge 模式设计“动作——作用对象”组合

其中，update_database() 将调用 obj.get_update_info()，而 send_notification() 将调用 obj.get_notification_info()。通过 Bridge 模式，我们用 3+2 个类实现了 3×2 种组合。事实上，用户还有取消点赞(Unlike)、删除(Delete)等动作，使用 Bridge 模式设计将大大提高可扩展性。

事实上，我们使用的 Flask 框架也本身也运用了许多设计模式，比如工厂方法、builder 模式等。正是这些设计模式的使用，给我们的开发带来了很大便利。