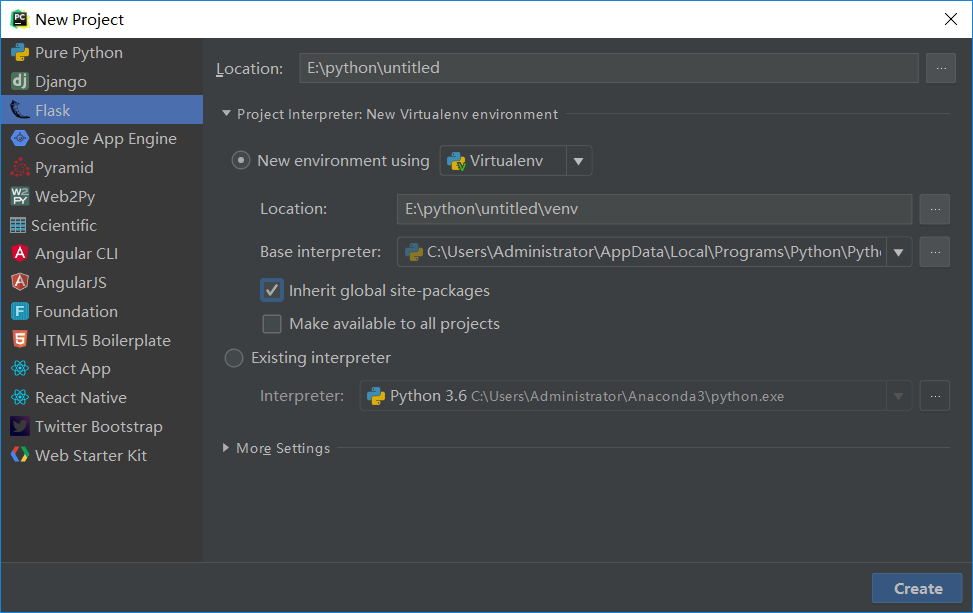
1. **Flask框架入门项目**

**任务**：爬取网站的新闻和天气信息，展示在自己的页面（git项目自己撸一次）

**IDE**：Pycharm 2018.1专业版

1. 使用pycharm创建项目时最好使用虚拟环境并将当前本地的模块导入



**备注**：使用虚拟环境需要提前下载pip下载安装virtualenv模块，虚拟环境可以使项目和当前系统的python版本以及模块隔离，命令行创建虚拟路径后需要激活后才可使用

1. 在导包的过程中遇到的小问题

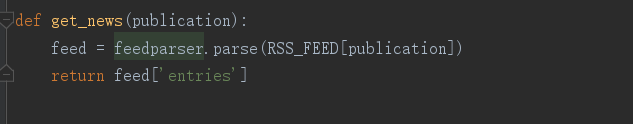
使用pychram写python代码导入模块的过程中，有可能会遇到失败的情况，这事需要在设置中修改source路径，并且导入的模块在未使用的情况下颜色是灰色的（不是报错），使用过程中会恢复正常

1. 关于requests模块

Requests模块的部分功能和urllib相同，据说目前爬虫获取网站资源的过程中使用requests比较多，正好之前简单了解过

1. 关于feedparser模块

简介：feedparser 号称是一个 universal feed parser，使用它我们可轻松地实现从任何 RSS 或 Atom 订阅源得到标题、链接和文章的条目了，RSS是RDF Site Summary 的缩写（RDF是Resource Description Framework的缩写 ），是指将网站摘要用xml语言描述。



代码中此处用feedparser.parse函数解析出来的是一个字典dict，解析出的内容如下图所示：



每个 RSS 和 Atom 订阅源都包含一个标题（d.feed.title）和一组文章条目（d.entries）。

通常，每个文章条目都有一段摘要（d.entries[i].summary），或者是包含了条目中实际文本的描述性标签（d.entries[i].description），此处解析后通过response返回给home.html之后可以直接读出页面标题和文章的概述

1. 关于make\_response

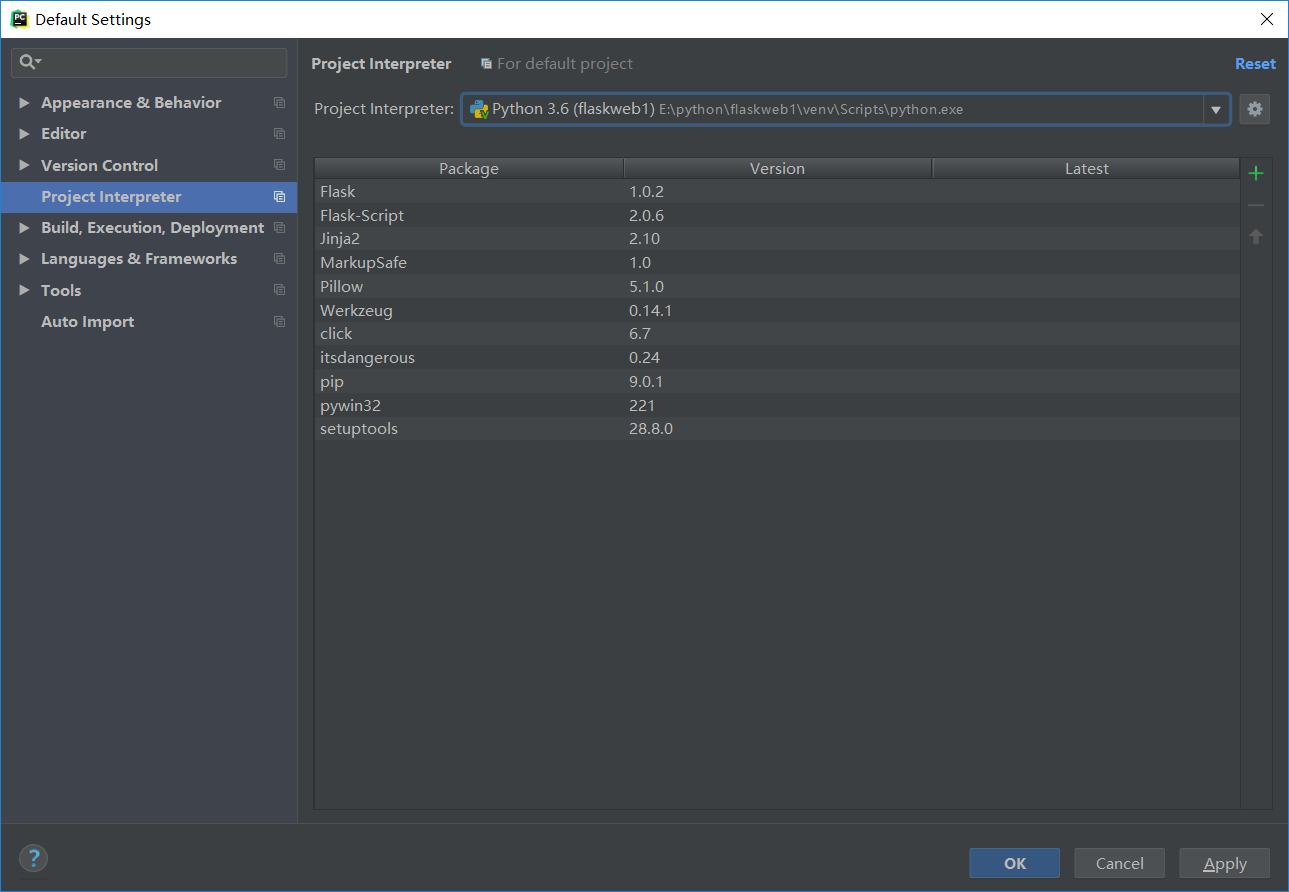
Flask框架中的响应类，命名很贴切，叫Response。不过Flask应用中很少直接调用这个类，Flask框架其实允许应用将默认的响应类，替换为自定义类，make\_response就是用来创建自定义响应类的

1. 关于flask中的request

在进行Flask开发中，前端需要发送不同的请求及各种带参数的方式，比如GET方法在URL后面带参数和POST在BODY带参数，有时候又是POST的表单提交方式，这个时候就需要从request提取参数。

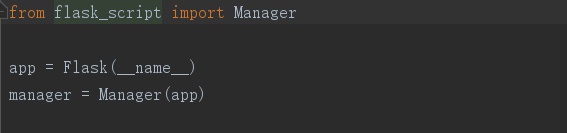
1. **《flask web 开发》问题总结**
2. Pycharm使用Flask扩展

在pycharm的flask项目若想使用扩展需要在File-》Default Setting-》Project interpretor中搜索想要的扩展模块并下载（最好在虚拟环境中），此问题在使用Flask-Script扩展时遇到，一直导包错误，后来通过此方式解决



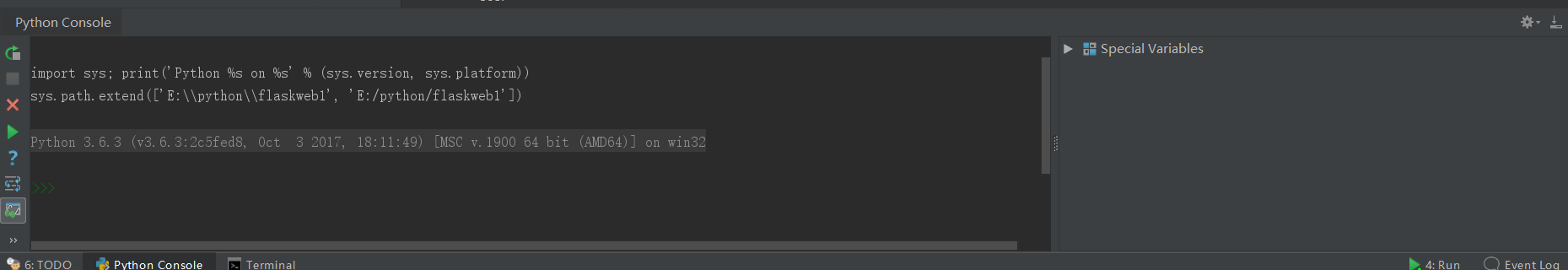
1. falsk.ext.script与falsk\_script

在python3.4+之后使用falsk\_xxx的方式导入flask的扩展而非书中的falsk.ext.script方式

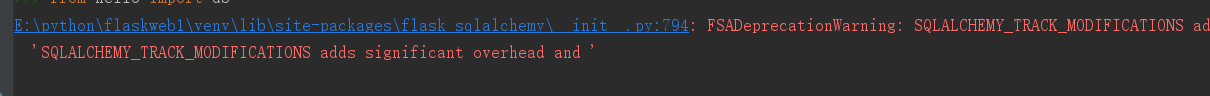


1. 关于pycharm进入shell模式

在数据库操作部分，问了让程序根据模型创建数据库表，需要进入shell模式，在pycharm中需要点击界面左下角选择python console进入shell模式

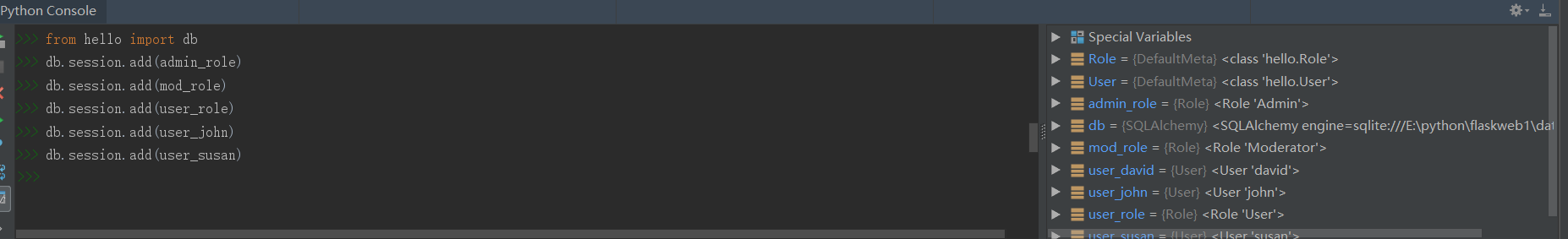


根据书上的代码执行‘from hello import db’时会报错，是因为SQLALCHEMY\_TRACK\_MODIFICATIONS:如果设置成True(默认情况)，Flask-SQLAlchemy 将会追踪对象的修改并且发送信号。这需要额外的内存， 如果不必要的可以禁用它。如果你不显示的调用它，在最新版的运行环境下，会显示警告。



1. 将对象写入数据库

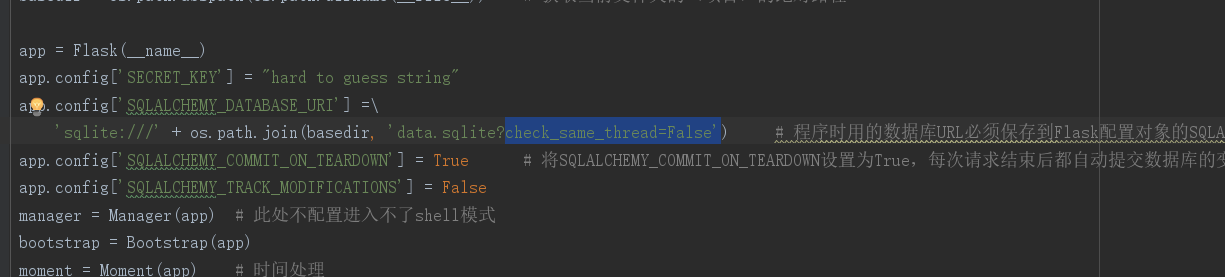
创建了角色和用户之后，数据还只存在于python中，未写入数据库，在Flask-SQLAlchemy中，会话由db.session表示，现将其添加到会话中，之后调用commit（）方法提交会话：



1. 关于使用Shell操作数据库多线程错误

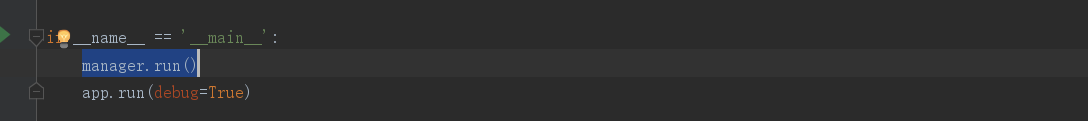
按照教程在pycharm中使用shell操作数据库时会出现多线程报错：sqlalchemy.exc.ProgrammingError: (sqlite3.ProgrammingError) SQLite objects created in a thread can only be used in that same thread.The object was created in thread id 139878844028672 and this is thread id 139878951810816 (Background on this error at: [http://sqlalche.me/e/f405)](http://sqlalche.me/e/f405)" \t "https://segmentfault.com/q/_blank)

后经过查资料，需要将sqlite的check\_same\_thread属性配置为False，故在数据库配置的url中将url改为：



1. 关于数据库迁移

按照书中代码，没有注册manager，需要导入falsk\_script中的Manager，注册manager，即manager = Manager（app），注册完毕后，还会出现错误，此处需要加如下代码：

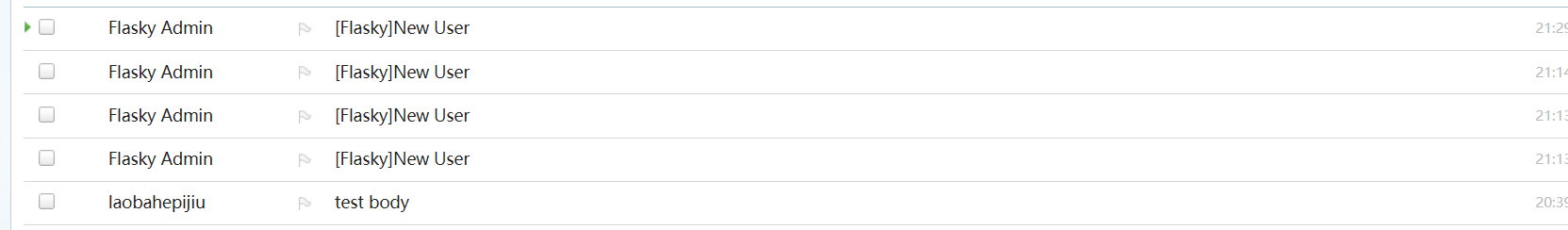


1. 关于电子邮件

教程中使用gmail，但由于国内用gmail过于肉疼，在自己写的时候使用163邮箱，在使用时会出现账号验证错误：smtplib.SMTPAuthenticationError: (535, 'Authentication failed')，但是自己的账号密码肯定没错，查资料后发现是由于163的smtp服务默认是关闭的，需要在设置中打开并设置自己的登录码：

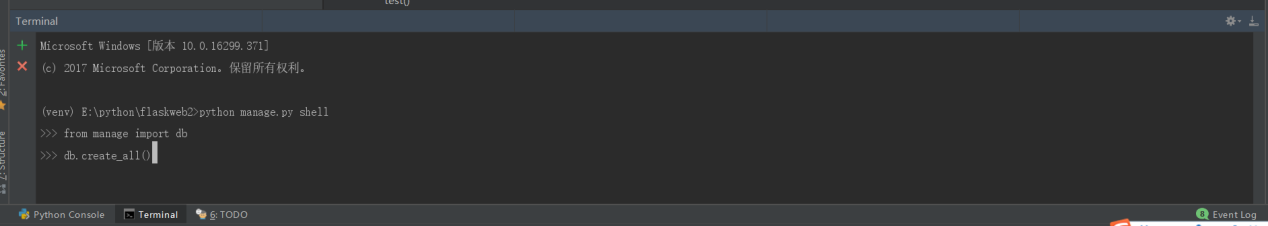


开启上图中的服务后，在非网易客户端登录时使用自己设置的登陆码进行登录，再次尝试之后邮件发送成功，测试截图：

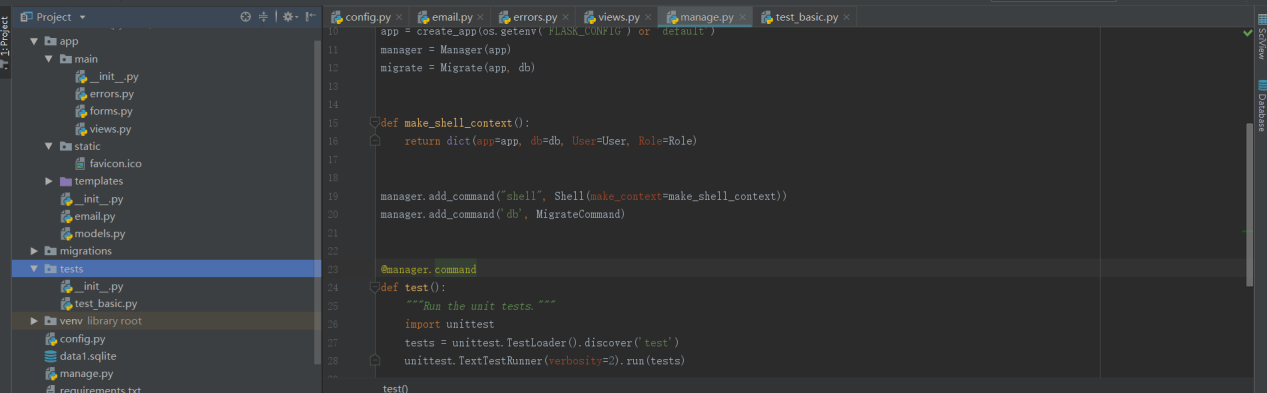


1. 关于‘大型程序结构’整合

为了清晰，将原来的单文件项目保留，新创建一个项目命名为flaskweb2并在pycharm中按照教程将原来的单文件项目重构为多文件的大型程序，重构之后直接运行出错，后来发现需要在shell模式下按照之前的教程让程序按照模型创建数据库，由于程序重构，本次需要在pycharm的Terminal中利用启动脚本manage.py进入shell模式并执行命令：

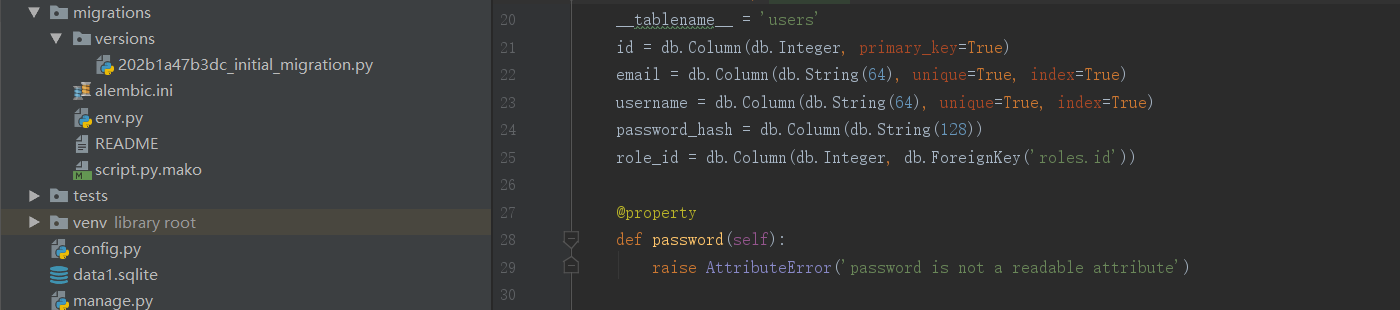


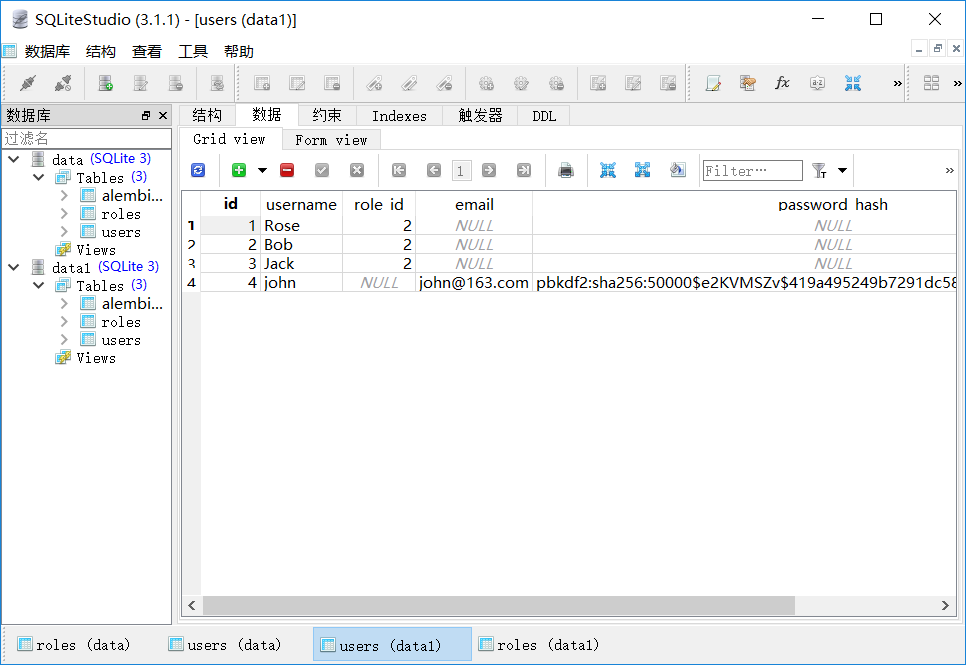
最终重构并运行成功，重构之后pycharm中的项目结构如图所示：

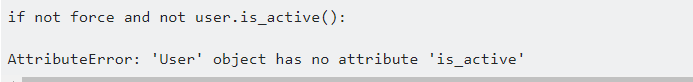


1. 关于用户登录模块

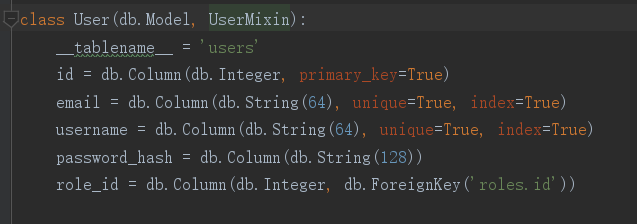
首先，登录模块对数据模型有了转变，在运行前需要按照第五章的内容将数据库进行迁移，执行在shell中执行upgrade操作，执行完毕之后，会发现数据库已经按照新的模型改变完毕，并在migration中生成了本次升级的脚本：



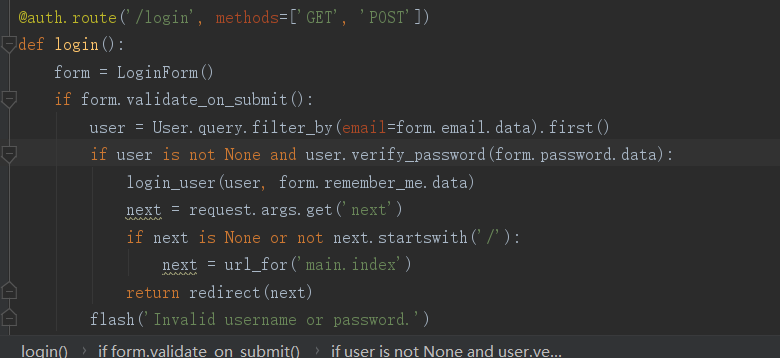


在shell中模拟注册用户之后，运行登录程序，发现服务器报错：

查资料后发现是由于User模型没有对UserMixin进行继承之后User模型就实现了is\_active方法（按照理解应该是这样），之后按照这样让User模型继承了该模块，程序可运行：



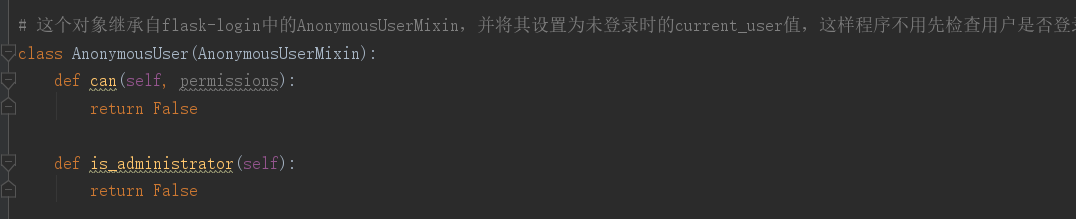
1. 关于‘next’



提交登录密令的POST请求最后也做了重定向，不过目标URL有两种可能，用户访问未授权的URL时会显示登录表单，Flask\_Login会把原地址保存在查询字符串的next参数中，这个参数可以从request.args字典中读取到，若读取不到则跳到首页。

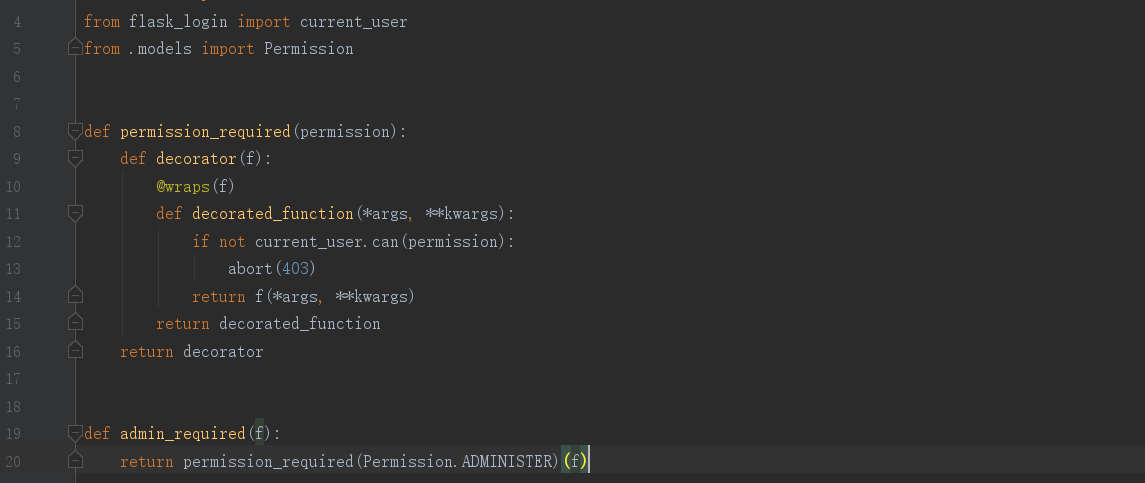
1. 关于AnonymousUserMIxin

定义AnonymousUser类是出于一致性的考虑，并实现了can方法和is\_administrator方法，这个类继承自Flask-Login中的AnonymousUserMixin类，将其设置为用户未登录时的current\_user的值，这样程序无需先检查用户是否登录，就可自由调用current\_user.can()方法和current\_user.is\_administratot()方法。



12、关于检查用户权限自定义修饰器

此处定义修饰器的作用是让视图函数只对具有特定权限的用户开放，两个修饰器一个用来检查常规权限，一个专门用来检查管理员权限，两个修饰器均使用了Python标准库中的functools包，若用户不具有指定权限，则返回403（HTTP禁止错误）。



1. 关于用户头像gravatar

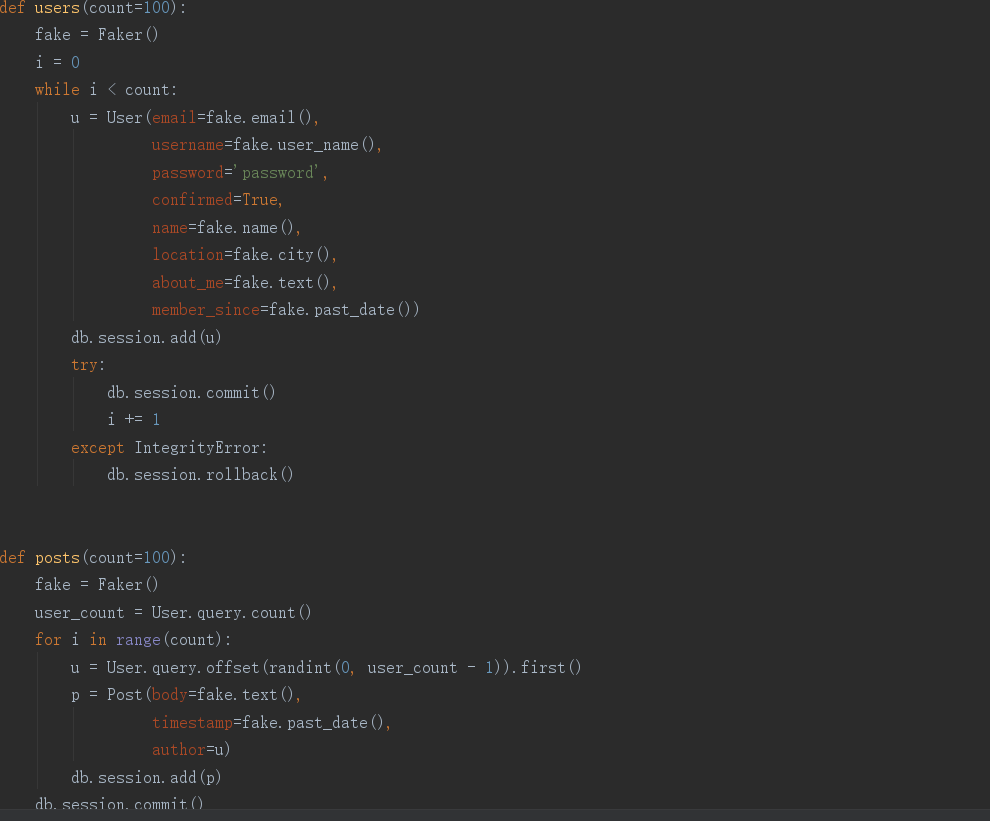
Gravatar是一个头像服务网站，头像和电子邮件关联起来，生成头像时需要用到用户电子邮件的md5码，然后在http://www.gravatar.com/avatar/后面加上用户电子邮件地址的md5码散列值。

1. 关于\_get\_current\_object()方法

用户博文中文章作者author属性表达式是current\_user.\_get\_current\_object()，变量current\_user由flask-login提供，和所有的上下文变量一样，也是通过线程内的代理对象实现，这个对象类似用户对象，但实际上是一个轻度包装，里面包含了真正的用户对象，数据库需要真正的用户对象，因此调用\_get\_current\_object()方法。

1. 关于创建虚拟（伪造）用户和博文数据

书中使用了forgerypy依赖包并在User和Post中创建了静态方法，而实际开发时使用了Faker包：



通过循环，利用fake.xxx()方法来生成相应的虚拟数据，该文件放在项目目录中的app目录下，想运行时，在shell模式下执行如下命令即可（一定要在该项目虚拟环境的shell模式下，否则会出现数据库db无法导入的错误！）



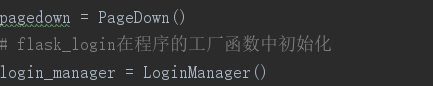
1. 关于分页

为了显示某页的数据，要把all()方法换成flask-sqlalchemy提供的paginate方法，页数是paginate()方法的第一个参数，也是唯一必须的参数，可选参数包括per\_page，用来显示每页显示的数量，若没有指定则默认20条，error\_out参数设置为True时，若请求的页数超过了范围，则会返回404错误，设置为False时则返回空列表，paginate返回的对象拥有很多属性，常用items属性来表示当前页面的记录。（当前渲染的页数从request.args.get(page,1,int)获取）

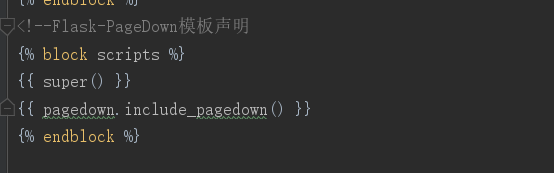


1. 关于富文本文章

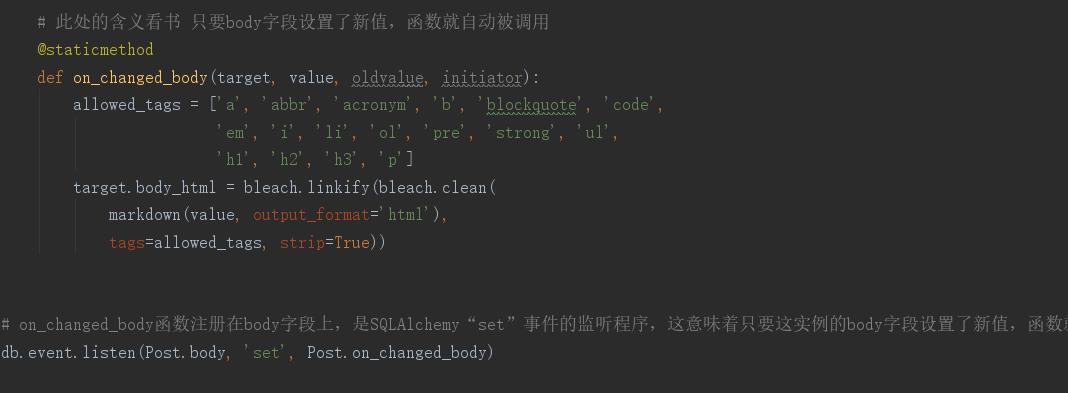
Flask-PageDown扩展定义了一个PageDownField类，这个类和WTForms中的TextAreaField接口一致，使用PageDownField之前要先初始化扩展：



初始化扩展之后在博文编辑器表单中将多行文本编辑器替换成MarkDown富文本编辑器即可（body = PageDownField(...)），而MarkDown预览使用PageDown生成，因此要在模板中做如下相应修改：

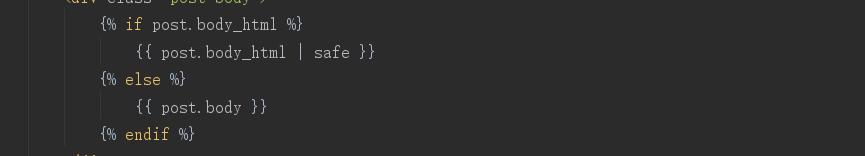


提交表单后，POST请求只会发送纯MarkDown文本，页面中显示的HTML预览会被丢掉，和表单一起发送生成的HTMLyulan会有安全隐患，因为攻击者能轻易地修改HTML代码，让其和MarkDown源不匹配，然后再提交表单。安全起见，只能提交MarkDown源文本，在服务器上使用MarkDown（使用Python编写的MarkDown到HTML转换程序）将其转换成HTML。得到HTML后，再使用Bleach进行清理，确保其中只能包含指定的几个HTML标签。为了避免重复工作，我们在创建博客博文时进行一次性转换，转换后存在Post模型中的字段中，转换函数如下：



这个转换的静态方法注册在body字段上，是SQLAlchemy“set”事件的监听程序，这意味着只要这实例的body字段设置了新值，函数就被自动调用！bleach.clean()函数对初步生成的html进行清洗，删除所有不在白名单中的标签，转换的最后一步由linkify()函数完成，这个函数有Bleach提供，把出文本中的URL转换成适当的<a>链接，最后一步是很有必要的，因为MarkDown规范没有为自动生成链接提供官方支持，而PageDown以扩展的形式实现了这个功能。

如果post.body\_html字段存在，还要把post.body转换成post.body\_html，转换如下所示：



渲染html格式内容时使用 | safe后缀，其目的是告诉Jinja2不要转义HTML元素。出于安全考虑，默认情况下Jinja2会转义所有模板变量，MarkDown转换成的HTML在服务器上生成，因此可以放心渲染。