综合案例：学生成绩管理

新建项目目录students,并创建虚拟环境

```
mkvirtualenv students
```

安装开发中使用的依赖模块

```
pip install flask==0.12.4
pip install redis
pip install flask-session
pip install flask-script
pip install flask-mysqldb
pip install flask-sqlalchemy
pip install flask-migrate
pip install flask_wtf
```

在pycharm中打开项目目录编写manage.py启动项目的文件

创建 `manage.py` 文件

```
from flask import Flask

app = Flask(__name__)

@app.route('/index')
def index():
    return 'index'

if __name__ == '__main__':
    app.run()
```

mange.py终不能存放大量的开发代码, 在开发中应该体现的是一种分工精神,所以我们可以把flask中各种功能代码进行分类分文件存储.

创建项目目录结构:

```
项目根目录/
├── application/              # 项目主要逻辑代码保存目录
|   ├── settings/             # 项目配置存储目录
|   |   ├ dev.py              # 开发阶段的配置文件
|   |   ├ prop.py             # 生产阶段的配置文件
|   ├── __init__.py           # 项目初始化文件
├── manage.py                 # 项目的终端管理脚本文件
```

配置文件

`settings/__init__.py` 代码:

```
from redis import StrictRedis

class Config(object):
    """项目配置核心类"""
    # 调试模式
    DEBUG = True

    # todo 配置日志
    pass

    # mysql数据库的配置信息
    SQLALCHEMY_DATABASE_URI = "mysql://root:123@127.0.0.1:3306/students?charset=utf8"
    # 动态追踪修改设置，如未设置只会提示警告
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    # 查询时会显示原始SQL语句
    SQLALCHEMY_ECHO= False

    # 配置redis
    REDIS_HOST = '127.0.0.1'  # 项目上线以后，这个地址就会被替换成真实IP地址，mysql也是
    REDIS_PORT = 6379

    # 设置密钥，可以通过 base64.b64encode(os.urandom(48)) 来生成一个指定长度的随机字符串
    SECRET_KEY = "ghhBljAaOuzw2afLqJOXrukORE4BlkTY/1vaMuDh6opQ3uwGYtsDUyxcH62Aw3ju"

    # flask_session的配置信息
    SESSION_TYPE = "redis" # 指定 session 保存到 redis 中
    SESSION_USE_SIGNER = True # 让 cookie 中的 session_id 被加密签名处理
    SESSION_REDIS = StrictRedis(host=REDIS_HOST, port=REDIS_PORT,db=1) # 使用 redis 的实
例
    PERMANENT_SESSION_LIFETIME = 24 * 60 * 60 # session 的有效期，单位是秒
```

`settings/dev.py` 代码:

```
from . import Config
class DevelopementConfig(Config):
    """开发模式下的配置"""
    # 查询时会显示原始SQL语句
    SQLALCHEMY_ECHO= True
```

`settings/prop.py` 代码:

```
from . import Config
class ProductionConfig(Config):
    """生产模式下的配置"""
    DEBUG = False
```

项目主应用中初始化项目

在`application/__init__.py`文件中，创建flask应用并加载配置

```
from flask import Flask
from application.settings.dev import DevelopementConfig
from application.settings.prop import ProductionConfig

config = {
    "dev": DevelopementConfig,
    "prop": ProductionConfig,
}

def init_app(config_name):
    """项目的初始化函数"""
    app = Flask(__name__)

    # 设置配置类
    Config = config[config_name]

    # 加载配置
    app.config.from_object(Config)

    return app
```

在manage.py 中调用 init_app 函数,启动项目

```
from application import init_app

app = init_app("dev")

@app.route("/")
def index():
    return "index"

if __name__ == '__main__':
    app.run()
```

在`application/__init__.py`项目初始化文件中加载redis或者mysql的初始化代码

```
from flask import Flask
from redis import StrictRedis
from flask_wtf.csrf import CSRFProtect
from flask_session import Session

from application.settings.dev import DevelopementConfig
from application.settings.prop import ProductionConfig

config = {
```

```
    "dev": DevelopementConfig,
    "prop": ProductionConfig,
}

# 为了方便redis的连接对象在函数外部可以使用,预先设置一个全局变量,接下来在函数中用于保存redis的连接
redis_store = None

def init_app(config_name):
    """项目的初始化功能"""
    app = Flask(__name__)

    # 设置配置类
    Config = config[config_name]

    # 加载配置
    app.config.from_object(Config)

    # redis的链接初始化
    global redis_store
    redis_store = StrictRedis(host=Config.REDIS_HOST, port=Config.REDIS_PORT,db=0)

    # 开启CSRF防范功能
    CSRFProtect(app)

    # 开启session功能
    Session(app)

    # TODO 注册蓝图对象到app应用中

    return app
```

增加数据库配置

```
# from flask import Flask
# from redis import StrictRedis
# from flask_wtf.csrf import CSRFProtect
# from flask_session import Session
from flask_sqlalchemy import SQLAlchemy
#
# from application.settings.dev import DevelopementConfig
# from application.settings.prop import ProductionConfig
#
# config = {
#     "dev": DevelopementConfig,
#     "prop": ProductionConfig,
# }
#
# # 为了方便redis的连接对象在函数外部可以使用,预先设置一个全局变量,接下来在函数中用于保存redis的连接
```

```
# redis_store = None
db = SQLAlchemy()
#
# def init_app(config_name):
#     """项目的初始化功能"""
#     app = Flask(__name__)
#
#     # 设置配置类
#     Config = config[config_name]
#
#     # 加载配置
#     app.config.from_object(Config)
#
#     # redis的链接初始化
#     global redis_store
#     redis_store = StrictRedis(host=Config.REDIS_HOST, port=Config.REDIS_PORT,db=0)
#
#     # 开启CSRF防范功能
#     CSRFProtect(app)
#
#     # 开启session功能
#     Session(app)
#
    # 配置数据库链接
    db.init_app(app)
#
#     # TODO 注册蓝图对象到app应用中
#
#     return app
```

因为前面已经在settings中设置了数据库的配置信息,所以接下来,创建对应的数据库

```
create database students charset=utf8;
```

在manage启动文件中新增关于启动过程中的相关功能

在项目根目录下`manage.py`中设置项目启动程序并调用`__init__.py`的app

```
from application import init_app,db
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand

app = init_app("dev")

# 使用终端脚本工具启动和管理flask
manager = Manager(app)

# 启用数据迁移工具
Migrate(app, db)
# 添加数据迁移的命令到终端脚本工具中
```

```
manager.add_command('db', MigrateCommand)


@app.route("/")
def index():
    return "index"

if __name__ == '__main__':
    manager.run()
```

日志

Python 自身提供了一个用于记录日志的标准库模块：logging。

日志的等级

```
FATAL/CRITICAL = 致命的，危险的
ERROR = 错误
WARNING = 警告
INFO = 信息
DEBUG = 调试
NOTSET = 没有设置
```

把日志设置封装成一个函数

```
import logging
from logging.handlers import RotatingFileHandler

# 把日志相关的配置封装成一个日志初始化函数
def setup_log(Config):
    # 设置日志的记录等级
    logging.basicConfig(level=Config.LOG_LEVEL)  # 调试debug级
    # 创建日志记录器，指明日志保存的路径、每个日志文件的最大大小、保存的日志文件个数上限
    file_log_handler = RotatingFileHandler("logs/log", maxBytes=1024 * 1024 * 300,
backupCount=10)
    # 创建日志记录的格式 日志等级 输入日志信息的文件名 行数 日志信息
    formatter = logging.Formatter('%(levelname)s %(filename)s:%(lineno)d %(message)s')
    # 为刚创建的日志记录器设置日志记录格式
    file_log_handler.setFormatter(formatter)
    # 为全局的日志工具对象（flaskapp使用的）添加日志记录器
    logging.getLogger().addHandler(file_log_handler)
```

在 `init_app` 方法中调用上一步创建的方法，并传入 `config_name`

```
    # 启用日志功能
    setup_log(Config)
```

在配置文件`settings/__init__.py`中,设置默认日志等级

```
class Config(object):
    """项目配置核心类"""
    # 调试模式
    DEBUG = True

    # todo 配置日志
    LOG_LEVEL = "DEBUG"
```

新增日志以后的项目目录结构

```
项目根目录/
├── docs/                    # 项目开发相关文档
├── logs/                    # 项目运行日志保存目录
|   ├── log                  # 日志文件
├── application/             # 项目主要逻辑代码保存目录
|   ├── settings/            # 项目配置存储目录
|   |   ├ dev.py             # 开发阶段的配置文件
|   |   ├ prop.py            # 生产阶段的配置文件
|   ├── __init__.py          # 项目初始化文件
├── manage.py                # 项目的终端管理脚本文件
```

经过上面的改造，我们接下来就可以开始创建蓝图了。

创建蓝图目录

在applications下创建apps目录，apps以后专门用于保存每一个项目的蓝图，

并在apps创建index蓝图目录,并在`__init__.py`文件中创建蓝图对象

```
from flask import Blueprint

index_blu = Blueprint("index_blu",__name__)
```

在index蓝图目录中新增对应的视图文件,代码:

```
from . import index_blu

@index_blu.route("/")
def index():
    return "首页"
```

在`__init__.py`中引入当前蓝图下所有的视图文件

```
from flask import Blueprint

index_blu = Blueprint("index_blu",__name__)

from .views import *
```

在项目初始化文件`application/__init__.py`文件中,注册蓝图对象

```
    # TODO 注册蓝图对象到app应用中
    # 首页模块
    from .apps.index import index_blu
    app.register_blueprint(index_blu,url_prefix='')
```

声明了蓝图目录以后的项目目录结构

```
项目根目录/
├── application/              # 项目主要逻辑代码保存目录
|   ├── settings/             # 项目配置存储目录
|   |   ├ dev.py              # 开发阶段的配置文件
|   |   ├ prop.py             # 生产阶段的配置文件
|   ├── __init__.py           # 项目初始化文件
|   ├── statics/              # 保存项目中所有的静态资源文件[img/css/js]
|   ├── modules/              # 保存项目中所有蓝图的存储目录
|   |   ├── index             # 蓝图目录
|   |   |   ├── __init__.py # 蓝图的初始化问年间
|   |   |   └── views.py      # 蓝图的视图函数文件
|   |   ├── __init__.py
├── manage.py                 # 项目的终端管理脚本文件
```

模型代码:

```
# coding=utf-8
from application import db

# 创建关系表,不再创建模型,一般用于表与表之间的多对多场景
"""
表关系变量 = db.Table(
    "关系表表名",
    db.Column('字段名', 字段类型, 字段选项),  # 普通字段
    db.Column("字段名", 字段类型, db.ForeignKey("表名.id")),
    db.Column("字段名", 字段类型, db.ForeignKey("表名.id")),
)
"""
achievement = db.Table(
    "achievement",
```

```
    db.Column('score', db.Numeric, comment="分数"),
    db.Column('student_id', db.Integer, db.ForeignKey('student.id')),
    db.Column('course_id', db.Integer, db.ForeignKey('course.id'))
)

class Student(db.Model):
    """学生信息"""
    __tablename__ = "student"
    id = db.Column(db.Integer, primary_key=True, comment="主键ID")
    name = db.Column(db.String(64), index=True, comment="姓名" )
    sex = db.Column(db.Boolean, default=True, comment="性别")
    class_number = db.Column(db.String(32), nullable=True, index=True, comment="班级")
    age = db.Column(db.SmallInteger, comment="年龄")
    description = db.Column(db.Text, comment="个性签名")
    courses = db.relationship(
        'Course', # 模型名称
        secondary=achievement, # 表关系变量
        backref='students', # 当外键反过来获取主键信息时,使用的字段名称,可以自定义,接下来的使用例如: course.students 获取某个课程下所有的学生
        lazy='dynamic'
    )

class Course(db.Model):
    """课程信息"""
    __tablename__ = "course"
    id = db.Column(db.Integer, primary_key=True,comment="主键ID")
    name = db.Column(db.String(64), unique=True,comment="课程名称")
```

添加测试数据

```
insert into student values
(1,"赵华",1,307,22,"对于勤奋的人来说，成功不是偶然；对于懒惰的人来说，失败却是必然。"),
(2,"程星云",1,301,20,"人生应该如蜡烛一样，从顶燃到底，一直都是光明的。"),
(3,"陈峰",1,504,21,"在不疯狂，我们就老了，没有记忆怎么祭奠呢？"),
(4,"苏礼就",1,502,20,"不要为旧的悲伤，浪费新的眼泪。"),
(5,"张小玉",0,306,18,"没有血和汗水就没有成功的泪水。"),
(6,"吴杰",1,307,19,"以大多数人的努力程度之低，根本轮不到去拼天赋"),
(7,"张小辰",0,405,19,"人生的道路有成千上万条，  每一条路上都有它独自的风景。")
```

闪现信息[flash]

使用后，只会出现一次的信息，叫“闪现信息”，用于在验证代码失败，或者一些只需要显示一次性提示的场景。

使用步骤：

视图中验证有误,则在显示模板之前设置flash

```
# 视图函数代码
from flask import flash

flash("对不起，您尚未登录，请登录！")
```

模板代码:

```
# 模板代码
{% for message in get_flashed_messages() %}
    <span>{{message}}</span>
{% endfor %}
```