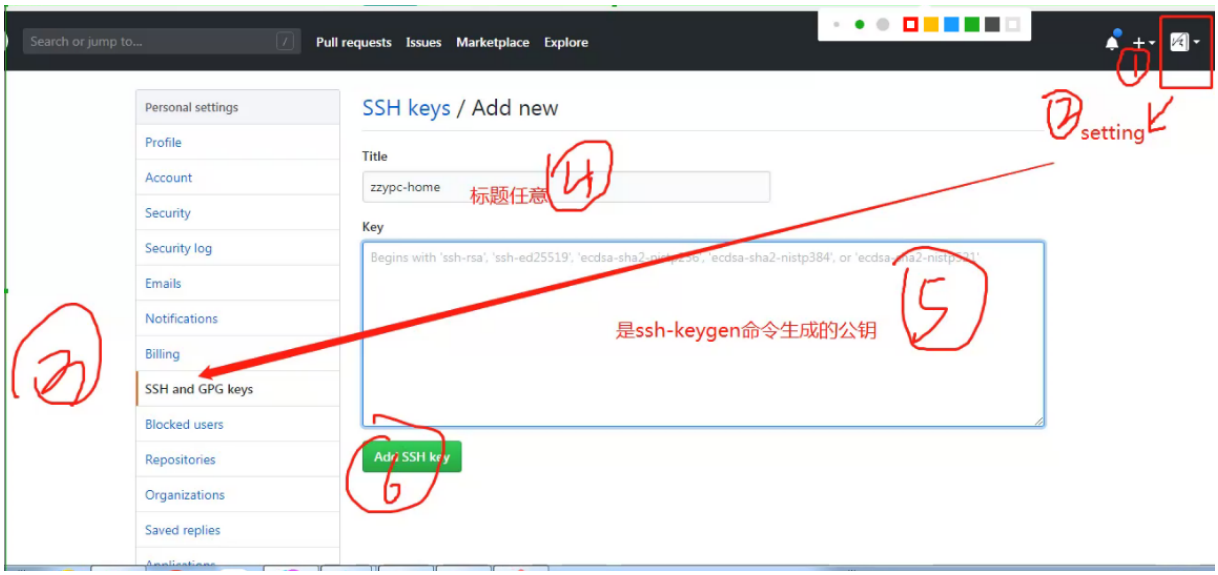
Flask简单的项目创建流程

编辑器：vscode，需要提前安装的插件有：Chinese（汉化），Python（解析器） Git && GitHub账号

注册GitHub账号

- 简单的使用流程
  - 1、设置自己电脑可访问远程共有仓库 **[和GitHub远程仓库加密协议默认为SSH加密]**

```
#1、打开git客户端 生成SSH key---->公有的秘钥地址在c\用户\Administrator\.ssh
ssh-keygen -t rsa -C "youremail@example.com"
#2、登陆GitHub，打开“Account settings”，“SSH Keys”页面:
```



  - 2、创建和维护本地git库/克隆远程空的库到本地
  - 3、本地仓库映射到GitHub中 推送代码

连接GitHub远程库

- 新建远程github公有库

- 本地代码库和远程库关联

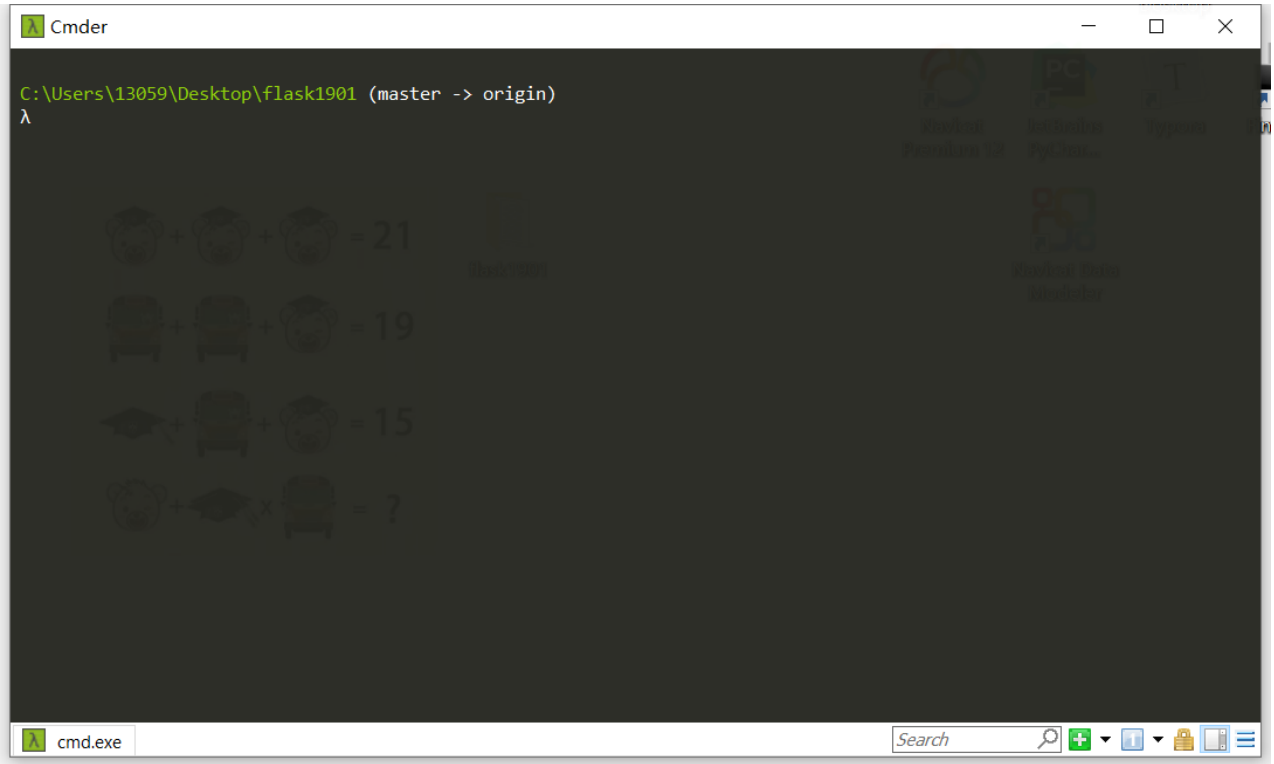
```
git remote add origin git@github.com:apang-ai/task.git
    远程    添加    源       源地址
```

自己按照操作 创建一个github仓库 名字叫 flask1901

命令行增强工具Cmder `https://cmder.net/`

- 首先下载cmder 下载步骤详细请参考链接： `<https://www.pianshen.com/article/6105678389/>`

  在桌面上创建一个文件夹 **flask1901** 文件夹下创建一个 **day1文件夹 day1文件夹下创建一个watchlist文件夹 右击flask1901文件夹** 出现 **Cmder Here** 点击 出现 如图所示界面



执行

```
# 初始化本地仓库
git init
```

新建 .gitignore文件

```
vim .gitignore
```

写入

```
*.pyc
*~
__pycache__
.vscode
```

Ctrl + o 保存 Ctrl + x 退出

切换到watchlist文件夹中执行命令

```
# 创建虚拟环境
python -m venv env
# 安装flask
pip install flask
# 激活虚拟环境
env/Scripts/activate
# 退出虚拟环境
deactivate
```

- ## 本地仓库与远程仓库关联

  1. 指定远程仓库地址

     ```
     git remote add origin git@github.com:apang-ai/1901flask.git
     ```

  2. 第一次提交：

     ```
     git status 查看文件变化

     git add . 追踪文件

     git commit -m "备注" 添加到本地仓库

     git push -u origin master 第一次提交需要指定仓库，默认分支
     ```

- ## 管理环境变量

  将项目拉取到VScode 中 进入虚拟环境

  ```
  # ctrl + `  打开控制台
  cd flask1901/day1/watchlist
  # 执行命令 激活虚拟环境
  env/Scripts/activate
  ```

  启动Flask程序时候，需要用到两个环境变量：FLASK_APP和FLASK_ENV

  启动flask项目后，控制台输出：

  ```
   * Environment: production
     WARNING: This is a development server. Do not use it in a production deployment.
     Use a production WSGI server instead.
   * Debug mode: off
  ```

  Debug mode，调试模式，设置成on，我们可以通过将FLASK_ENV设置成development来开启

  为了不用每次启动项目都在终端中设置环境变量，我们安装用来管理系统环境变量的

  python--dotenv：pip install python-dotenv

  python-dotenv 默认会从项目的根目录下的.flaskenv和.env文件读取环境变量并设置，我们新建这两个文件

  ```
  touch .flaskenv .env
  ```

  .flaskenv 用来存储Flask命令行系统相关的公共环境变量。 设置

  ```
  FLASK_ENV=development
  ```

就打开了debug mode

.env 用来存储敏感数据，不应该提交到git仓库，所以需要在.gitignore文件中设置忽略

```
# watchlist/.gitignore   ctrl+s保存
.vscode
*.pyc
*~
__pycache__
env
.env
```

- **动态URL**

通过输入 http://127.0.0.1:5000/index/apang ，浏览器中展示Hello，Bruce

```
from flask import Flask
app = Flask(__name__)
# 动态URL
@app.route('/index/apang')
def home(name):
    return "<h1>Hello, %s</h1>"%name
```

url_for函数用来生成URL:url_for('home',name="apang")

- **配置及模板 {{ }} 前端模板**

app.py

```
# views
@app.route('/', methods=['GET', 'POST'])

def index():
    name = 'Apang'

    movies = [
        {"title":"大赢家","year":"2020"},
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"},
        {"title":"速度与激情8","year":"2012"},
        {"title":"我的父亲母亲","year":"1995"},
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"},
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"}
    ]

    return render_template('index.html', movies=movies, name=name)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>

    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ user.name }}'s WatchList</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <h2>
        <img class='avatar' src="{{ url_for('static', filename='images/avatar.jpg') }}"
alt="">
        {{ name }}'s WatchList
    </h2>

    <ul class='movie-list'>

        <li>
            {{ movie.title }} - {{ movie.year }}
        </li>
    <footer>
        <small>&copy;2020<a href="#">1901电影列表</a></small>
    </footer>
</body>
</html>
```

- ## 静态文件

在Python脚本中，需要从flask中导入url_for， 在模板里面可以直接用

static文件需要强制刷新清除缓存，之后生产环境我们会加入一个动态的版本号

- ## 数据库操作

SQLite 使用Flask-SQLAlchemy集成SQLAlchemy

1. 安装

   pip install flask-sqlalchemy

2. 初始化

   ```
   from flask_sqlalchemy import SQLAlchemy  # 导入扩展类
   app = Flask(__name__)
   db = SQLAlchemy(app)  # 初始化扩展，传入程序实例app
   ```

3. Flask里面，使用Flask.config接口来写入和获取设置Flask，扩展或者是程序本身的行为配置，配置变量的名称必须大写，写入配置的语句一般会放到扩展类实例化语句之前

4. 创建数据库模型

```
# models
class User(db.Model):
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(20))
class Movie(db.Model):
    id = db.Column(db.Integer,primary_key=True)
    title = db.Column(db.String(60))
    year = db.Column(db.String(4))
```

5. 创建表和数据库文件

```
进入flask shell
>>> from app import db
>>> db.create_all()
>>>
```

如果改动了模型类，想重新生成表模式，需要先db.drop_all()删除表，然后在创建。

将来使用Flask-Migrate扩展

6. 自定义数据库创建和删除的命令

flask initdb 创建数据库

flask initdb --drop 删除表后再创建

7. 增删改查

添加

```
>>> from app import User,Movie
>>> user = User(name="Bruce")  创建记录
>>> movie1 = Movie(title="杀破狼",year="2000")
>>> movie2 = Movie(title="战狼",year="2016")
>>> db.session.add(user) 添加到数据库会话，只是将改动添加到一个临时区域
>>> from app import db
>>> db.session.add(user)
>>> db.session.add(movie1)
>>> db.session.add(movie2)
>>> db.session.commit() 提交数据库会话，只需要最后调用一次
```

查（读取）

模型类.query.<过滤方法>.<查询方法>

Movie.query.first()

过滤方法	作用
filter()	筛选指定规则过滤记录，返回新产生的查询对象
filter_by()	
order_by()	
group_by()	

查询方法	作用
all()	
first()	
get(id)	
count()	
first_or_404()	
get_or_404(id)	
pagination	

更新（update）

```
>>> movie = Movie.query.first()
>>> movie.title
'杀破狼'
>>> movie.title = "shapolang"
>>> db.session.commit()
```

删除（delete）

```
>>> db.session.delete(movie)
>>> db.session.commit()
```

1. 在项目中操作数据库
2. 创建数据库中的数据

- **模板和代码优化**

1. 错误处理函数

```
@app.errorhandler(404)
def page_not_found(e):
    user = User.query.first()
    # 返回模板和状态码
    return render_template('404.html',user=user),404
```

2. 模板上下文处理函数

```
# 模板上下文处理函数
@app.context_processor
def common_user():
    user = User.query.first()
    return dict(user=user)
```

返回的变量会统一注入到每一个模板的上下文环境中，所以可以直接在模板中使用

3. 模板继承
4. 添加百度视频链接

```
<span class="float-right">
            <a class="baidu" href="http://v.baidu.com/v?word={{ movie.title
}}&ct=301989888&rn=67&pn=0&db=0&s=0&fbl=800&ie=utf-8" target="_blank" title="在百度视
频中查找">百度视频</a>
        </span>
```

- **自定义命令**

自定义命令必须使用装饰器 @app.cli.command()

之后运行的时候在命令行中输入 **flask 函数名 即可运行**

```
# 自定义指令
# 新建data.db初始化命令
@app.cli.command()  # 装饰器 注册命令
@click.option('--drop', is_flag=True, help='删除之后在创建')
def initdb(drop):
    if drop:

        db.drop_all()

    db.create_all()

    click.echo('初始化数据库完成')

# 向data.db中写入数据
@app.cli.command()

def forge():

    # name = 'Apang'

    movies = [
        {"title":"大赢家","year":"2020"},
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"},
        {"title":"速度与激情8","year":"2012"},
        {"title":"我的父亲母亲","year":"1995"},
```

```
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"},
        {"title":"囧妈","year":"2020"},
        {"title":"战狼","year":"2020"},
        {"title":"心花路放","year":"2018"}
    ]

    # user = User(name=name)
    # db.session.add(user)
    for i in movies:

        movie1 = Movie(title=i['title'], year=i['year'])
        db.session.add(movie1)

    db.session.commit()
    click.echo('插入数据完成')

# 生成管理员账号
@app.cli.command()
@click.option('--username', prompt=True, help='管理员账号')
@click.option('--password', prompt=True, help='管理员密码', hide_input=True,
confirmation_prompt=True)
def admin(username, password):
    db.create_all()
    user = User.query.first()
    if user is not None:
        click.echo('更新用户信息')
        user.username = username
        user.set_password(password)
    else:
        click.echo('创建用户信息')
        user = User(username=username, name='Admin')
        user.set_password(password)
        db.session.add(user)

    db.session.commit()
    click.echo('管理员创建完成')
```

- **模板继承**

由于前段过多的重复代码 因此为了简化代码量 使用模板继承

创建一个base.html 把所有页面相同的代码提取出来

{% block content %}{% endblock %} 这个就是放每个页面不同的代码

```
# templates/base.html
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>{{ user.name }}'s WatchList</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
    {% endblock %}
</head>
<body>
    <h2>
        <img class='avatar' src="{{ url_for('static', filename='images/avatar.jpg') }}"
alt="">
        {{ user.name }}'s WatchList
    </h2>
    {% for message in get_flashed_messages() %}
        <div class='alert'>{{ message }}</div>
    {% endfor %}
    <nav>
        <ul>
            <li><a href="{{ url_for('index') }}">首页</a></li>
            {% if not current_user.is_authenticated %}
             <li><a href="{{ url_for('login') }}">登录</a></li>
            {% else %}
             <li><a href="{{ url_for('login') }}">{{ current_user.name }}</a></li>
            {% endif %}
            {% if current_user.is_authenticated %}
            <li><a href="{{ url_for('logout') }}">退出</a></li>

            <li><a href="{{ url_for('settings') }}">设置</a></li>
            {% endif %}
        </ul>
    </nav>
    {% block content %}{% endblock %}
    <footer>
        <small>&copy;2020<a href="#">1901电影列表</a></small>
    </footer>
</body>
</html>
```

index.html

{% extends 'base.html' %} 继承的父模板 base.html 中的代码部分

{% block content %}{% endblock %} 就是子模板中不同部分的代码

```
{% extends 'base.html' %}
{% block content %}
    <p>总共 {{ movies|length }}个</p>
    <!-- 表单: 添加电影 -->
    {% if current_user.is_authenticated %}
    <form action="" method="post">
        电影名称: <input type="text" name='title' autocomplete="off" required>
        电影年份: <input type="text" name='year' autocomplete="off" required>
        <input type="submit" value="添加" class='btn' name='submit'>
    </form>
    {% endif %}
    <ul class='movie-list'>
        {% for movie in movies %}
```

```
            <li>
                {{ movie.title }} - {{ movie.year }}
                <span class='float-right'>
                    <a class='baidu' href="http://v.baidu.com/v?word={{ movie.title
}}&pn=0&db=0&s=0&fbl=800&ie=utf-8" target="_blank" title='在百度视频中查找'>百度视频</a>
                </span>
                {% if current_user.is_authenticated %}
                <span class=float-right>
                    <a class='btn' href="{{url_for('edit', movie_id=movie.id)}}">编辑</a>
                </span>
                <span class='float-right'>
                    <form action="{{ url_for('delete', movie_id=movie.id) }}"
method='post'>
                        <input class='btn-delete' type="submit" value="删除"
onclick='return confirm("确定删除吗? ")'>
                    </form>
                </span>
                {% endif %}
            </li>

        {% endfor %}
    </ul>

{% endblock %}
```

- **表单**

1. 表单
2. 处理表单数据

   ```
   405
   Method Not Allowed
   The method is not allowed for the requested URL.
   ```

   原因：默认处理地址请求的index视图函数默认只能接受GET请求

   ```
   @app.route('/',methods=['GET','POST'])
   ```

3. 针对GET和POST方法的请求，要有不同的处理逻辑

   GET：返回渲染的页面

   POST：获取表单数据并保存

4. 错误信息： The session is unavailable because no secret key was set. Set the secret_key on the application to something unique and secret.

   flash()函数内部会把消息存储到Flask提供的session对象里。session用来在请求间存储数据，它会把数据签名后存储到浏览器的Cookie中，所以我们需要设置签名所需的密钥

   ```
   app.config['SECRET_KEY'] = 'watchlist_dev'
   ```

这个密钥在开发时候随便设置，但是考虑的安全性，在部署时候应该设置为随机字符串，也不能明文的写到代码

5. 在模板中显示flash提示

   get_flashed_messages() 返回是一个list类型

```
flash('错误信息')
```

- **用户认证**

1. Flask 依赖的Werkzeug内置了用于生成和验证密码hash值的函数

   werkzeug.security.generate_password_hash() 生成密码

   werkzeug.security.check_password_hash() 检查密码

```
>>> from werkzeug.security import generate_password_hash,check_password_hash
>>> pw_hash = generate_password_hash('123456')
'pbkdf2:sha256:150000$AM0Iq4hT$c0ad2c0b5c6dfb28ac092fc3ff8f12e3315f9b97263fa7e321d1a
2dc31fd61c0'
>>> check_password_hash(pw_hash,'123456')
True
>>> check_password_hash(pw_hash,'12345678')
False
```

2. 登录登出 Flask-Login

   安装 pip install flask-login

   初始化

```
from flask_login import LoginManager
app = Flask(__name__)
login_manager = LoginManager(app) # 实例化登录拓展类

@login_manager.user_loader
def load_user(user_id):
    user = User.query.get(int(user_id))
    return user
```

   current_user是Flask-Login提供的变量，如果用户已经登录，current_user变量的值就是当前用户

   让User继承Flask-Login提供的UserMixin类，继承之后User类多了几个属性，

   is_authenticated属性：如果当前用户已经登录，那么current_user.is_authenticated 会返回True，否则就是False

3. 认证保护

   @login_required

4. 前段模板判断 {% if 条件 %}{% endif %}

```
{% if current_user.is_authenticated %}
    <form action="" method="post">
        电影名称: <input type="text" name='title' autocomplete="off" required>
        电影年份: <input type="text" name='year' autocomplete="off" required>
        <input type="submit" value="添加" class='btn' name='submit'>
    </form>
{% endif %}
```