## 个人博客

1. 明确工作任务。

我们首先应该清楚我们需要做什么，我们首先拿到的模板是一个静态网页的模板。

我们需要做两件事情：

一件是将网页的内容数据连接数据库，让网页的数据可以迁移储存，也可以通过网页各项需求从数据库提取相对应的数据信息，获得动态的数据；

另一件事情是将数据与模板相结合，让动态数据通过模板得以完美的展示，也就是使用模板渲染动态数据。

1. 分析需求，确定模型搭建

首先，我们需要完成的是数据库表的建立，这是动态数据的依赖。

建立数据库表，首先我们需要在模型模块中【models】创建相应的类。

所以，我们第一步需要做的是分析拿到的网页结构，看看我们都需要哪些数据。这些数据可以如何分类，选择一个合适的分类，将网页数据分为几类。

然后我们需要确定表与表之间的相互关联关系，确定外键的定义，最好是新建一个文档记录一下，划分清晰。

最后我们需要根据我们划分完成的类在我们的模型模块【models】中创建性对应的类。然后迁移、执行，生成对应表。

# 需要注意的是我们在创建相对应的类属性【列|标签】的时候，最好选择合适的数据类型。因为我们在之后的网页中，如果需要通过forms生成表单的时候，这些类属性就是一个个input标签，类属性的数据类型就对应相应的标签输入框的类型，这一点可以提前考虑。还可以考虑一下属性数据的必要性，因为标签默认是属性不能为空的，可以为不必要的属性设置可以为空，还要注意在需要设置最大长度的属性内设置最大长度。

1. 完善配置和模块文件

生成表之后，我们需要创建超级管理员账户，用于对我们数据的管理和调试。

然后我们需要在根目录创建模板【teleplate】文件夹，将模板拷贝进去，然后在setting中配置模板文件的路径。

我们还需要在根目录创建样式文件夹【static】，然后在setting中做出相对应的配置。

1. 处理模板模块

前端的一套网页模板，有好多页面和样式都是重复的，我们需要分析对比相似度高的网页模板。提取其中的共性，创建base模板，供其他子模板继承。子模板只保留自己相对的特性，其它的样式与标签均可以从父模板继承。这样做可以大大减少我们的代码量，也可以减少我们对网页数据修改的工作量，查看起来也会很方便，一目了然。

# 需要注意的是，我们要注意对于{{}}和{%%}的使用，分清我们的语句是变量，还是逻辑语言，使用错误会造成错误。然后我们需要在里面分析清楚我们需要修改的东西，或者添加的东西，然后在父模板留下相对应的重写块。

1. 调通其中的路径

首先，我们需要建立相对应页面的视图类，然后根据需要重写其中的get或者post方法，渲染或者重定向相应的页面，重定向页面在这一步可能会有些困难，如果觉的困难可以在之后具体到post请求的时候再做处理，其实也不太难考虑。

然后我们需要根据我们视图类调通虚拟路径，连接我们的模板网页。

最后在模板文件中配置相应的页面跳转的请求链接。

还需要在模板文件中更改文件样式的路径，因为我们已经改变了样式文件的存储位置，所以说一定要在父模板中查找清楚，全部修改完成，不要有遗漏。

1. 完善数据传递【完成动态数据的添加】，处理post请求。

分析我们HTML文件中的表单请求，为表单请求完善提交路径等信息。

分析页面需要显示数据的需求对象，在views视图文件中获取相应的对象传递过来，然后将页面中需要展示的数据全部替换为我们从数据库得到的动态数据。

在视图类中完善post方法，与数据库结合处理表单提交的数据。将处理完成的数据返回。

# 注意，在这一步，我们需要完成的是我们整个流程的核心。一定要注意数据的获取和传递方式、路径。

# 一定要注意在HTML文件中添加数据时候的语法规范。

# 在处理表单数据时，一定要返回表单的对象，调用表单对象属性时也要注意是要使用表单对象哪条属性的哪个属性~~切记！

# 以下为需要注意的操作。

1. 使用Django自定义标签
2. 在应用下方创建模块名字必须为templatetags（含有\_\_init\_\_.py）
3. 在文件夹中新建文件 my\_fun.py

from django.template import Library

register = Library()

@register.simple\_tag()

def mytags(num=3)

return result

1. 在模板中使用

在html顶部{% load my\_fun %}

{% mytags args as result %}

1. Django使用RSS订阅

RSS是一种获取最新内容方式，可以将网站包装成XML格式，用户只需要订阅对应的RSS地址，即可获取最新通知，RSS包装网站的结果给RSS聚合工具查看

1. 新建feed.py用于重写django自带feed类，将文章包装成XML格式

使用django自带模块将最新更新包装成XML

from django.contrib.syndication.views import Feed

from .models import Article

class ArticleFeed(Feed):

title = “”

description = “”

link = “/”

def items(self):

return result

def item\_title(self, item):

return item.title

def item\_description(self, item):

return item.content

def item\_link(self, item):

return “/single/%s” % (item.id)

1. 添加RSS路由地址

from .feed import ArticleFeed

url(r’^rss$’, ArticleFeed(), name=’rss’)

1. 在页面添加超级链接

{% url ‘blog:rss’ %}

1. 使用markdown编写内容
2. 在当前虚拟环境下安装markdown

pip install markdown

1. 对指定的字段使用markdown渲染成html格式

1).获取markdown实例

md = markdown.Markdown(extensions=[

‘markdown.extensions.extra’,

‘markdown.extensions.codehilite’,

‘markdown.extensions.toc’,

])

2).使用markdown实例渲染指定字段

article.content = md.convert(article.content)

3).将md的目录对象赋予article

Article.toc = md.toc

1. 修改文章，使用markdown格式编写
2. 插入目录

在指定地方使用[TOC]

{{ article.content|safe }}

{{ article.toc|safe }}

1. 分页

引入分页模块：

From django.core.paginator import Paginator

获取GET到的页面number：

pagenum = req.get.GET(‘page’)

创建对象,传入分页的数据列表和每页数量：

paginator = Paginator(article, 1)

判断输入的页面number是否为空，默认显示第一页：

Pagenum = 1 if pagenum == None else pagenum

得出请求页面的信息：

page = paginator.get\_page(pagenum)

设置虚拟路径:

page.path = ‘/’

返回数据：

Return render(req, ‘blog/index.html’, {‘page’: page}

上一页：

{% if page.has\_previous %}

<li><a href="{{ page.path }}?page={{ page.previous\_page\_number }}">上一页</a></li>

{% endif %}

数字页码：

{% for i in page.paginator.page\_range %}

{% if i == page.number %}

<li class="current"><a href="#">{{ i }}</a></li>

{% else %}

<li><a href="{{ page.path }}?page={{ i }}">{{ i }}</a></li>

{% endif %}

{% endfor %}

下一页：

{% if page.has\_next %}

<li><a href="{{ page.path }}?page={{ page.next\_page\_number }}">下一页</a></li>

{% endif %}