

Django 自学报告

江灿 2019011325 软件02 jiang-c19@mails.tsinghua.edu.cn

[Django introduction](#)

[overview](#)

[Layer](#)

[Forms](#)

[Admin](#)

[Security](#)

[Internationalization and localization](#)

[Performance and optimization](#)

[Common Web application tools](#)

[预备知识](#)

[Web开发：](#)

[Web框架：](#)

[Django的特性](#)

[设计模式](#)

[Install](#)

[Verifying](#)

[具体使用](#)

[创建](#)

[Request and Responses](#)

[Models](#)

[Database setup](#)

[Creating models](#)

[激活模型](#)

[使用可交互式的API](#)

[Admin](#)

[Views](#)

[Template](#)

[Forms](#)

[Patch for Django](#)

[解释说明](#)

Django introduction

Django is one of the most popular python web frameworks. It suits developers of different levels: from beginners to professionals. Django helps you save time writing code and make it more efficient.

Django has the following pros:

- URL routing.
- Web server support.
- Database schema migration.
- Working with databases: PostgreSQL, MySQL, SQLite and Oracle.

- The native engine of templates.

Django基于MVC模型，即Model + View + Controller 的设计模式

overview

Layer

- model layer: Django 提供了一个抽象层 (“model”) 来构建和操作 Web 应用程序的数据。
- view layer: Django 有视图 (“view”) 的概念来封装负责处理用户请求和返回的逻辑响应。
- template layer: 模板层提供了一种designer-friendly的语法，用于呈现要呈现给用户的信息。

Forms

Django 提供了一个丰富的框架来促进表单的创建和表单数据的操作。(其中Tutorial就是介绍一个Forms的创建)

。

Admin

Django提供了Admin的管理，

Security

Internationalization and localization

Django 提供强大的国际化和本地化框架来帮助开发应用程序，可以适用于多种语言和世界地区。

Performance and optimization

Django 有多种技术和工具可以帮助代码更有效地运行，更快并且使用更少系统资源。

Common Web application tools

Django 提供了 Web 应用程序开发中常用的多种工具：

Authentication、Caching、Logging、Sending emails、Syndication 、feed(RSS/Atom)、Pagination、Messages framework、Serialization、Sessions、Sitemaps、Static files management、Data validation, etc.

预备知识

因为Web网页的开发是一个完全陌生的领域，所以需要先行学习不少预备知识，总结整理如下

Web开发：

Web开发指的是开发基于B/S架构，通过前后端的配合，将后台服务器的数据在浏览器上展现给前台用户的应用。

Web框架：

在早期，没有Web框架的时候，如何创建Web应用的呢？

以使用Python CGI脚本显示数据库中最新添加的10件商品为例：

```
import pymysql

# 首先打印起始标签
print("Content-Type: text/html\n")
print("<html><head><title>products</title></head>")
print("<body>")
print("<h1>products</h1>")
print("<ul>")

#连接数据库，开始查询
connection = pymysql.connect(user='user', passwd='pwd', db='product_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM products ORDER BY create_date DESC LIMIT 10")

# 生成HTML列表项
for row in cursor.fetchall():
    print("<li>%s</li>" % row[0])

# 其他HTML元素
print("</ul>")
print("<p>https://www.liujiangblog.com</p>")
print("</body></html>")

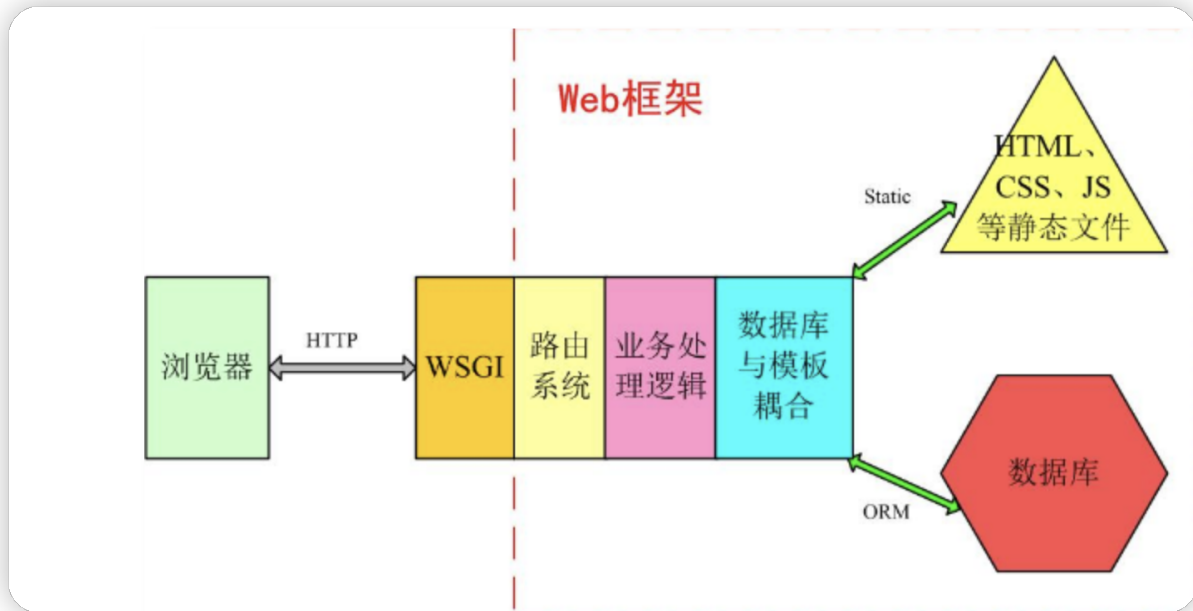
connection.close()
#然后将生成的HTML代码保存到一个.cgi文件中，然后上传到网络服务器上，用户通过浏览器即可访问。
```

虽然上面的例子简单易懂，但是会产生一些问题，如：

1. 网络应用底层的协议、线程、进程如何处理？
2. 如果应用中有多处需要连接数据库会怎样呢？
3. 前端、后端工程师以及数据库管理员集于一身，无法分工配合。
4. 代码复用性低
5. 密码直接写在代码里不安全
6. 代码修改成本高。

于是就有了Web框架，Web框架致力于解决一些共同的问题，为Web应用提供通用的架构，让用户专注于网站应用业务逻辑的开发，而无须处理网络应用底层的协议、线程、进程等方面的问题，从而大大提高开发者的效率和Web应用程序的质量。

最基础的Web框架的架构



Django的特性

- Django是一个全栈Web框架。全栈框架：是指除了封装网络和线程操作，还提供HTTP请求和响应、数据库读写管理、HTML模板渲染等一系列功能的框架。
- 功能完善、要素齐全。
- 完善的文档。经过长期的发展和完善，Django有广泛的实践经验和完善的在线文档。开发者遇到问题时可以搜索在线文档寻求解决方案（但是文档实际上也太长了导致看文档花费不少时间）
- 强大的数据库访问API。Django的Model层自带数据库ORM组件，开发者无须学习其他数据库访问技术（例如SQLAlchemy）。
- 灵活的路由系统。Django具备路由转发、正则表达式、命名空间、URL反向解析等功能。
- 丰富的Template模板功能：Django自带类似jinja的模板语言，不但原生功能丰富，还可以自定义模板标签和过滤器。并且以类似Python的调用机制和视图默契配合。
- 自带后台管理应用admin：只需要通过简单的几行配置和代码就可以实现一个完整的后台数据管理控制平台。这是Django最受欢迎的功能。
- 完整的错误信息提示：在开发调试过程中如果出现运行错误或者异常，Django可以提供非常完整的错误信息帮助定位问题。

设计模式

MVC设计模式：

最早由Trygve Teenskaug在1978年提出，上世纪80年代是程序语言Smalltalk的一种内部架构。后来MVC被其他领域借鉴，成为了软件工程中的一种通用架构模式。MVC把Web框架分为三个基础部分：

模型(Model)：用于封装与应用程序的业务逻辑相关的数据及对数据的处理方法，是Web应用程序中用于处理应用程序的数据逻辑的部分，Model只提供功能性的接口，通过这些接口可以获取Model的所有功能。白

话说，这个模块就是业务逻辑和数据库的交互层，定义了数据表。

视图(View)：负责数据的显示和呈现，是对用户的直接输出。

控制器(Controller)：负责从用户端收集用户的输入，可以看成提供View的反向功能。

这三个部分互相独立，但又相互联系，使得改进和升级界面及用户交互流程，在Web开发过程任务分配时，不需要重写业务逻辑及数据访问代码。

MVC在Python之外的语言中也有广泛应用，例如VC++的MFC，Java的Struts及Spring、C#的.NET开发框架，都非常有名。

MTV设计模式：

MTV和MVC大体上一致，Django对传统的MVC设计模式进行了修改，将视图分成View模块和Template模块两部分，将动态的逻辑处理与静态的页面展示分离开。而Model采用了ORM技术，将关系型数据库表抽象成面向对象的Python类，将数据库的表操作转换成Python的类操作，避免了编写复杂的SQL语句。

模型(Model)：和MVC中的定义一样

模板(Template)：将模型数据与HTML页面结合起来的引擎

视图(View)：负责实际的业务逻辑实现

Install

1. Install python
2. Set up a database
3. Install Django

Verifying

```
>>> import django
>>> print(django.get_version())
```

```
>>> import django
>>> print(django.get_version())
4.1
```

具体使用

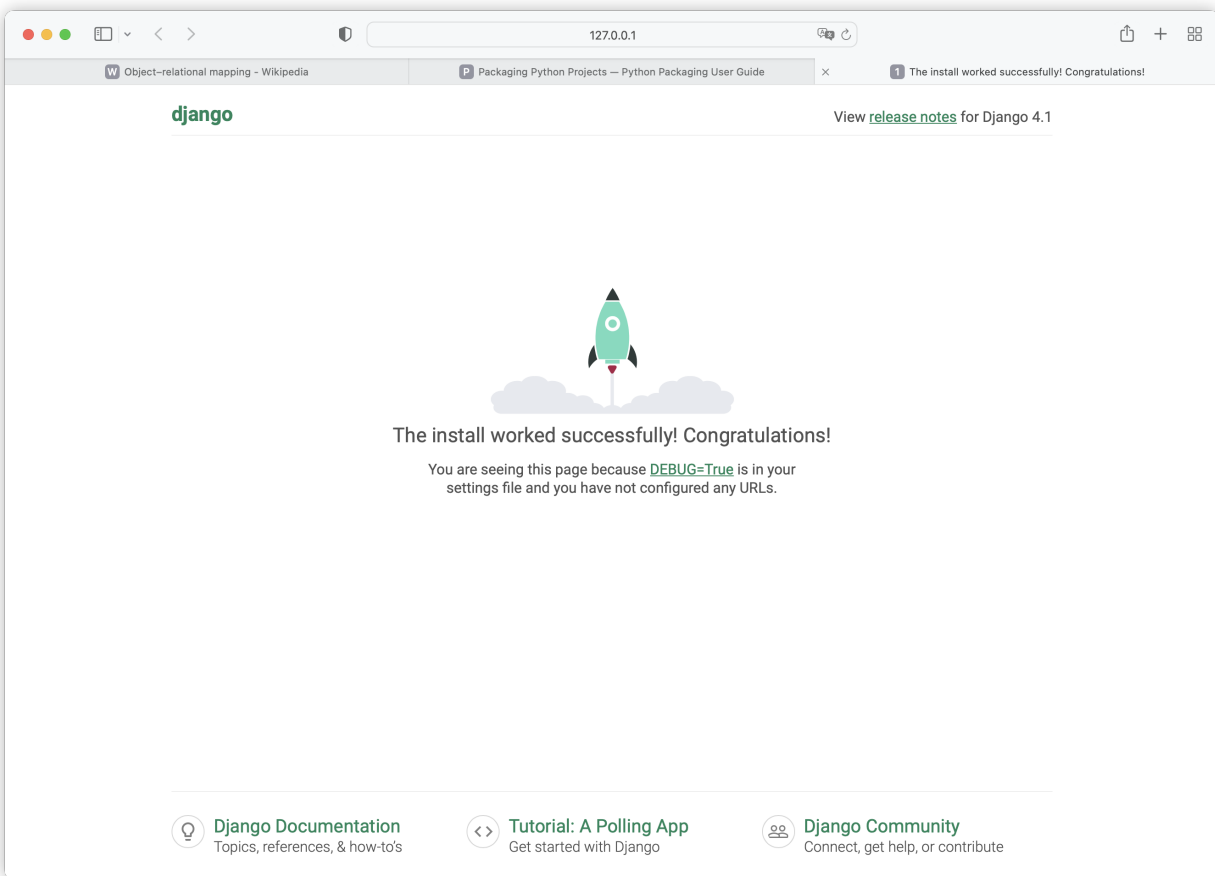
创建

```
$ python -m django --version #验证检查版本
4.1
$ django-admin startproject mysite #使用 startproject 命令开始一个新的项目
```

会创建如下目录

```
mysite/ # a container for your project.
manage.py # A command-line utility that can interact with this Django project in various ways
mysite/ # the actual Python package for your project
__init__.py # An empty file that tells Python that this directory should be considered a Python pack- age.
settings.py # n for this Django project.
urls.py # The URL declarations for this Django project
asgi.py # Asynchronous Server Gateway Interface
wsgi.py # Web Server Gateway Interface
```

`$ python manage.py runserver` 成功开启服务，可在8000端口（默认）访问如下界面



`$ python manage.py startapp polls` 创建polls app，并添加 `urls.py`，文件目录如下

```
polls/
__init__.py
admin.py
apps.py
migrations/
__init__.py
models.py
tests.py
views.py # app的view
urls.py # URLconf
```

Request and Responses

主要通过 `path()` 来实现URL访问。在Django中，在处理请求时，Django从urlpatterns中的第一个模式开始，然后沿着列将请求的URL与每个模式进行比较，直到找到匹配的URL。在查找时，不会查找GET/POST 或者域名。下面是 `path()` 参数的解释

- route: 路由是一个包含URL模式的字符串。
- view: 当Django找到匹配模式时，它会调用以HttpRequest对象为第一个参数的指定视图函数，并将路由中的任何“捕获”值作为关键字参数。
- kwargs
- name: 命名一个URL可以任意的从Django的其他地方引用，这个功能呢可以在只对一个文件进行修改的时候做到全局的更改。

Models

Database setup

`settings.py` 是该项目的配置文件。在默认情况下，配置的数据库使用SQLite。因为第一次接触数据库，所以我也使用的默认的SQLite。配置文件如下

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

同时，在配置文件中，记录了该app的实例文件，初始化如下

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin', # The admin site.
    'django.contrib.auth', # An authentication system.
    'django.contrib.contenttypes', # A framework for content types.
    'django.contrib.sessions', # A session framework.
    'django.contrib.messages', # A messaging framework.
    'django.contrib.staticfiles', # A framework for managing static files.
]
```

Creating models

因为是做一个polls，所以设计两个models，Question&Choice

可以在polls/models.py 修改代码如下

```
from django.db import models

class Question(models.Model):
    # 在这里，每个模型都由一个子类django.db.models.Model的类表示。
```

```

# 每个模型都有许多类变量，每个类变量代表模型中的数据库字段。
question_text = models.CharField(max_length=200)
# 每个字段由字段类的实例表示——例如字符字段的CharField和日期时间的DateTimeField。
# 这告诉Django每个字段持有哪种类型的数据。

# 可以使用字段的可选第一个可选参数来指定人类可读的名称。
# 如果没有提供此字段，Django将使用机器可读的名称。
# 在本例中，只为Question.pub_date定义了一个人类可读的名称。'date published'
# 对于此模型中的所有其他字段，该字段的机器可读名称也能够很好的被人理解。
pub_date = models.DateTimeField('date published')

class Choice(models.Model):
# 使用ForeignKey定义关系。此处告诉Django，每个选择都与一个问题有关。
# Django支持所有常见的数据库关系：多对一、多对多和一对一。
question = models.ForeignKey(Question, on_delete=models.CASCADE)
# 对于CharField，需要提供参数max_length
choice_text = models.CharField(max_length=200)
# 可选参数default
votes = models.IntegerField(default=0)

```

激活模型

写好了上述代码后，需要在setting.py文件告诉Django安装polls.app，添加代码 `'polls.apps.PollsConfig'` 即可，然后执行命令 `$ python manage.py makemigrations polls`

输出如右图所示，代表Question和Choice的修改已经被Django存储。

`makemigrations` 是Django存储模型更改（以及数据库模式）的方式。

之后再使用 `$ python manage.py migrate` 命令迁移，在数据库中创建这些模型。

```

→ mysite
python manage.py makemigrations polls
Migrations for 'polls':
polls/migrations/0001_initial.py
- Create model Question
- Create model Choice

```

使用可交互式的API

`$ python manage.py shell` 可进入可交互式的API界面


```

→ mysite
python manage.py shell

Python 3.10.5 (main, Aug 10 2022, 23:48:16) [Clang 13.1.6 (clang-1316.0.21.2.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Choice, Question
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
>>> q.save()
>>> q.id
1
>>> q.question_text
'What's new?'
>>> q.pub_date
datetime.datetime(2022, 8, 15, 2, 26, 36, 234507, tzinfo=datetime.timezone.utc)
>>> q.question_text = "What's up?"
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>

```

一些简单的运行测试，更多的API

Making queries | Django documentation | Django

Once you've created your data models , Django automatically gives you a database-abstraction API that lets you create, retrieve, update and delete objects. This document explains how to use this API. Refer to the data model reference for full details of all the various model lookup options.

 <https://docs.djangoproject.com/en/4.1/topics/db/queries/>

常用API汇总

- `objects.all()`
- `objects.filter(id=1)`
- `filter(question_text__startswith='What')`
- `objects.get(pub_date__year=current_year)`
- `objects.create()`

Admin

\$ `python manage.py createsuperuser` 创建一个具有高级权限的Admin，启动服务后，可以在 127.0.0.1:8000/admin/ 访问管理员界面。

通过 `admin.site.register(Question)` 在admin.py注册Question之后，即可在管理员界面查看之前的Question界面。

在此界面，不同的model field类型（DateTimeField、CharField）对应于适当的HTML输入部件。Django能够自动的选择采用何种方式来展示。

Views

通过在 `polls.urls` 添加path如

```
path('<int:question_id>/', views.detail, name='detail'),
```

然后在 `views.py` 添加

```
def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)
```

即可完成对于view界面的编写，通过path当URL为<int:question_id>是，通过views.detail获取HttpResponse，然后渲染到页面上。

Template

```
template = loader.get_template('polls/index.html')
```

此处Django会在polls/templates/polls/index的目录下寻找template，所以需要新建一个文件在templates，然后再再子目录里新建一个同名的文件夹，然后再编写相应的html文件，Django会自动的在该目录下寻找匹配的模板

- `render()`

```
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {'latest_question_list': latest_question_list}
    return HttpResponse(template.render(context, request))

#可以简写为
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

在文档中提到

Each view is responsible for doing one of two things: returning an `HttpResponse` object containing the content for the requested page, or raising an exception such as `Http404`.

所以除了正常的HttpResponse外，我们还需要处理404的情况。

处理的方法为引入包 `from django.http import Http404` 然后在 `view.py` 文件中

```
def detail(request, question_id):
    try:
        question = Question.objects.get(pk=question_id)
    except Question.DoesNotExist:
```

```
raise Http404("Question does not exist")
return render(request, 'polls/detail.html', {'question': question})
```

该界面对于debug帮助很大。

同样的，对于404也有简写，简写后代码如下

```
def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

其余的template地方与平常的html编写基本一致，除了一个点，即URL的hardcoded。

通过将 `< li >< a href="/polls/{{ question.id }}">{{ question.question_text }}< / a >< / li >` 改写为 `< li >< a href="{% url 'detail' question.id %}">{{ question.question_text }}< / a >< / li >` 将写为hardcode的/polls/改为使用 url 'detail' 来表示，使得代码更加易于修改。

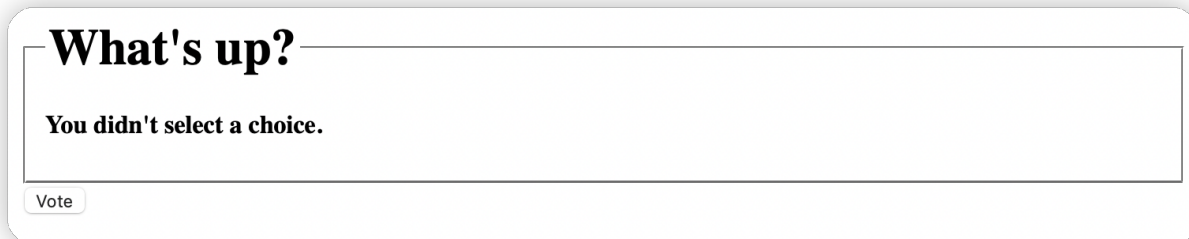
同时，还可以通过增加 Namespace 来使得当项目工程较大时，解决可能会出现的重名的问题。

Forms

最后就是完成一个具体的Forms的编写，通过HTML的<form>元素

```
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
<fieldset>
    <legend><h1>{{ question.question_text }}</h1></legend>
    {% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
    {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
        <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
    {% endfor %}
</fieldset>
<input type="submit" value="Vote">
</form>
```

可以获得如下界面



What's up?

You didn't select a choice.

之后，只需要再往里面填充表单的Question和Choice即可，到此，对于Django的初步学习已经完成，之后便是不断的练习和查阅文档来增强自己对于这个框架的熟练程度。

Patch for Django

因为在此之前，大部分时候都是在GitHub等开源社区clone然后自己学习使用，在学习Django的时候，同时学习了如何为一个开源项目做贡献，分为以下步骤

- Installing Git.
 - Downloading a copy of Django's development version.
 - Running Django's test suite.
 - Writing a test for your patch.
 - Writing the code for your patch.
 - Testing your patch.
 - Submitting a pull request.
-

解释说明

因为感觉难度较大，同时对于Django上手也觉得困难，而且自学python库本来就是想培养我们自己独立去根据文档和源码学习一个新的库的过程，我觉得Django符合这个要求所以将这个项目作为自学报告的项目，要是助教觉得不妥请告知我。