Djagno 기반의 웹프로그래밍

이 소 영

yisy0703@naver.com

Web Frameworks for Python(https://wiki.python.org/moin/WebFrameworks)
장고 공식 사이트(https://www.djangoproject.com/)
장고 공식 소스 저장소(http://github.com/django/django)
장고 참조 문서(https://docs.djangoproject.com/ko/5.2/)

Agenda

Django(장고) 기반의 파이썬 웹 프로그래밍

Ch01. Django 시작하기
1. Django 란?
2. 개발 환경 구축
3. Django 구조

Ch02. Django App
1. Django Project
2. Model
3. View

Ch03. Model
1. Model 속성 및 옵션
2. Relationship
3. Migrations
4. Admin App

Ch04. Django SQL
1. Django shell
2. Manager & QuerySet
3. 조회 SQL
4. 생성/수정/삭제 SQL
5. Django-Debug-Toolbar

Ch05. Template
1. Template Loader
2. URL Dispatcher
3. Template 상속
4. Template Engines
5. Template Filter

Ch07. Django View
1. View 기본
2. View 활용

Ch06. Django Form
1. HTML form
2. CSRF
3. HttpRequest/HttpResponse
4. Django Form
5. Django Model Form
6. Form Validation

Ch08. File 관리
1. Static Files
2. Media Files

Ch09. 사용자 인증
1. Auth App
2. 회원가입 구현
3. 로그인/아웃 구현
4. Oauth 라이브러리 활용

Ch 07 Django View

1. View 기본
2. View 활용

Chapter 07. Django View

- Class-based views
  (https://docs.djangoproject.com/en/5.2/topics/class-based-views/)

- django.views.generic
  (https://github.com/django/django/tree/5.2/django/views/generic)

- Built-in class-based views API
  (https://docs.djangoproject.com/en/5.2/ref/class-based-views/)
  (**https://docs.djangoproject.com/ko/5.2/ref/class-based-views/generic-editing**)

- View
  (https://github.com/django/django/blob/master/django/views/generic/base.py)

- ListView
  (https://github.com/django/django/blob/5.2/django/views/generic/list.py)

- DetailView
  (https://github.com/django/django/blob/5.2/django/views/generic/detail.py)

- Generic Editing Views
  (https://github.com/django/django/blob/5.2/django/views/generic/edit.py)

1. View 기본

View

1. View 기본

- **호출 가능한 객체**
  - 첫번째 인자로 HttpReqeust 인스턴스를 받음
  - 리턴값으로 HttpResponse 인스턴스를 반환함

- **함수 기반 뷰(Function Based View)**
  - 함수로 구현한 뷰

- **클래스 기반 뷰(Class Based View)**
  - 클래스로 구현한 뷰

View

1. View 기본

- **함수 기반 뷰**

```
def index(request):
    return HttpResponse("hello, World!")
```

- **클래스 기반 뷰**

```
class DetailView(object):
    def __init__(self, model):
        self.model = model
    def get_object(self, *args, **kwargs):
        return get_object_or_404(self.model, pk=kwargs['pk'])

    def get_template_name(self):
        return '{}/{}_detail.html'.format(self.model._meta.app_label, self.model._meta.model_name)

    def dispatch(self, request, *args, **kwargs):
        return render(request, self.get_template_name(), {self.model._meta.model_name: self.get_object(*args, **kwargs),})

    @classmethod
    def as_view(cls, model):
        def view(request, *args, **kwargs):
            self = cls(model)
            return self.dispatch(request, *args, **kwargs)
        return view
```

Generic View

1. View 기본

- Django에서 범용적으로 사용되는 기능을 구현해 놓은 View 클래스
- 별도의 View 구현 없이 바로 사용할 수 있음
- 필요한 경우 제네릭뷰를 상속하여 속성과 메소드를 오버라이딩하여 커스터마이징 할 수 있음
- 참조문서
  - https://docs.djangoproject.com/en/5.2/topics/class-based-views
- 장고 기본 CBV 패키지
  - https://github.com/django/django/tree/5.2/django/views/generic

제네릭 뷰의 종류

1. View 기본

분류	이름	설명
Base View	View TemplateView RedirectView	모든 뷰들이 상속받는 최상위 뷰 주어진 템플릿으로 렌더링 주어진 URL로 리다이렉트
Generic display View	DetailView ListView	한객체의 상세한 정보를 보여줌 여러개의 객체를 보여줌
Generic edit View	FormView CreateView UpdateView DeleteView	주어진 폼을 보여줌 객체를 생성하는 폼을 보여줌 객체를 수정하는 폼을 보여줌 객체를 삭제하는 폼을 보여줌
Generic Data View	ArchiveIndexView YearArchiveView MonthArchiveView WeekArchiveView DayArchiveView TocayArchiveView DateDetailView	날짜필드를 기준으로 객체들을 보여줌 주어진 연도의 객체들을 보여줌 주어진 년,월의 객체들을 보여줌 주어진 년,주(week)의 객체들을 보여줌 주어진 년,월,일의 객체들을 보여줌 오늘 날짜의 객체들을 보여줌 주어진 년,월,일,기본키의 객체를 보여줌

https://docs.djangoproject.com/en/5.2/topics/class-based-views

STEP 1

1. View 기본

```
# article/views.py
def article_detail(request, pk):
    article = get_object_or_404(Article, pk=pk)
    return render(request,'article/article_detail.html', {'article':article} )

# article/urls.py
path('<pk>/', views.article_detail, name='detail')
```

```
# article/templates/article/article_list.html
{% for article in article_list %}
<li>
    {{article.id}}
    <a href="{% url 'article:detail' article.id %}">{{article.title}}</a>
</li>
{% endfor %}

#article/templates/article/article_detail.html
<h1>{{article.title}}</h1>
{{article.body | linebreaks}}
```

STEP 2

1. View 기본

방법1

```
# article/views.py
from django.views.generic import DetailView

article_detail = DetailView.as_view(model=Article)

#article/urls.py
path('<pk>/', views.article_detail, name='detail')
```

방법2

```
# article/views.py
class ArticleDV(DetailView):
        model = Article

# article/urls.py
path('<pk>/', views.ArticleDV.as_view(), name='detail')
```

2. View 활용

TemplateView

2. View 활용

- 화면에 보여줄 템플릿 파일을 처리하는 간단한 뷰

```python
# views.py
from django.views.generic.base import TemplateView
from articles.models import Article
class HomePageView(TemplateView):
    template_name = "article/home.html"
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['latest_articles'] = Article.objects.all()[:5]
        return context
```

```python
# urls.py
from django.urls import path
from myapp.views import HomePageView
urlpatterns = [
     path('home', HomePageView.as_view(), name='home'),
]
```

```
#home.html
{% for article in latest_articles %}
    <li>{{article.title}}</li>
{% endfor %}
```

ListView

2. View 활용

- class django.views.generic.list.ListView
- 여러 객체의 리스트를 보여주는 뷰
- paginate_by 옵션 : 페이지당 갯수 지정
- 기본 템플릿 경로 : "앱이름/모델명소문자_list.html"
- 기본 context 이름 : "모델명소문자_list" or "object_list"

```
# article/views.py
article_list = ListView.as_view(model=Article, paginate_by=10)

## 웹브라우저에서 다음 URL 요청
http://localhost:8000/article/
http://localhost:8000/article/?page=2
```

urls.py

2. View 활용

```
# article/urls.py

from django.urls import path
from . import views
app_name = "article"
urlpatterns = [
    # path('', views.article_list, name="list"),
    # path('', views.article_list2, name="list"),
    path('', views.ArticleListView.as_view(), name='list'),
    # path('<id>/detail/', views.detail, name='detail'),
    path('<id>/detail/', views.ArticleDetailView.as_view(), name='detail'),
    path('new/', views.article_new, name='new'),
    path('<pk>/edit/', views.article_edit, name='edit'),
    path('<pk>/delete/', views.article_delete, name='delete'),
]
```

ListView (views.py)

2. View 활용

```
def article_list(request):
  q = request.GET.get('q','')
  qs = Article.objects.filter(title__icontains=q).order_by('title')
  return render(request, 'article/article_list.html',
                {'article_list':qs, 'q':q})

article_list2 = ListView.as_view(model=Article)

class ArticleListView(ListView):
  model = Article
  def get_context_data(self, **kwargs):
    context = super().get_context_data()
    q = self.request.GET.get('q','')
    context['article_list'] = Article.objects.filter(title__contains=q)
    context['object_list'] = context['article_list']
    context['q'] = q
    return context
```

ListView (views.py) ; paging 추가

2. View 활용

```
# Import Pagination Stuff
from django.core.paginator import Paginator

class ArticleListView(ListView):
  model = Article
  def get_context_data(self, **kwargs):
    context = super().get_context_data()
    q = self.request.GET.get('q','')
    context['article_list'] = Article.objects.filter(title__contains=q)
    context['object_list'] = context['article_list']
    context['q'] = q
    # Set up Pagination
    p = Paginator(Article.objects.filter(title__icontains=q).order_by('id'),
                  5) # 한페이지당 5개씩 출력
    page = self.request.GET.get('page')
    context['articles'] = p.get_page(page)
    return context
```

article_list.html; paging 추가할 사항

2. View 활용

```
<hr>
   Has Previous : {{articles.has_previous}}<br>
   Current Page : {{articles.number}}<br>
   {{articles}}<br>
   Has Next : {{articles.has_next}}<br>
   Number of Pages : {{articles.paginator.num_pages}}
   <hr>
   {% if articles.has_previous %}
      <a href="?page=1&q={{q}}">&laquo First</a>
      <a href="?page={{articles.previous_page_number}}&q={{q}}">Previous</a>
   {% endif%}

   Page {{ articles.number}} of {{articles.paginator.num_pages}}

   {% if articles.has_next %}
      <a href="?page={{articles.next_page_number}}&q={{q}}">next</a>
      <a href="?page={{articles.paginator.num_pages}}&q={{q}}">Last &raquo</a>
   {% endif %}
```

DetailView

2. View 활용

- 특정 객체에 대한 정보를 보여주는 뷰
- 기본 템플릿 경로 : “앱이름/모델명소문자_detail.html”
- 기본 context 이름 : “모델명소문자 "

```
from django.views.generic import DetailView
from django.utils import timezone

def detail(request, id):
    article = Article.objects.get(pk=id)
    return render(request, 'article/article_detail.html',
      {'article':article})

class ArticleDetailView(DetailView):
  model = Article
  def get_object(self):
    object = get_object_or_404(Article, id=self.kwargs['id'])
    return object
  def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['now'] = timezone.now()
    return context
```

article_detail.html


```
{% extends "base.html" %}
{% block title %}
    {{article.title}} 상세보기
{% endblock %}
{% block content %}
  <h2>{{article.id}}. {{article.title}}</h2>
  {{article.body | linebreaks }}
  <p>
    <a href="{%url 'article:edit' article.id %}">[수정]</a>
    <a href="{%url 'article:delete' article.id %}">[삭제]</a>
    <a href="{% url 'article:list' %}">[목록]</a>
  </p>
  현재 요청시간 : {{now}}
{% endblock %}
```

FormView

2. View 활용

- Form을 보여주기 위한 View
- form_class : Form 클래스 지정
- template_name : 폼을 렌더링 할 페이지
- form_valid( ) : 유효성 검사 성공 후 호출되는 메소드
- success_url : 유효성 검사 성공 후 이동할 페이지

```
#contact.html

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Send message">
</form>
```

CreateView / UpdateView

2. View 활용

- **ModelForm을 통해 모델 객체 생성/수정**
- **속성**
  - model : 필수 속성으로 모델 지정
  - form_class : 폼 객체 지정, 미지정시 ModelForm 생성 후 적용
  - template_name : 기본 템플릿 경로는 "앱이름/모델소문자_form.html"
  - success_url : POST 처리 완료후 이동할 페이지, 미지정시 모델의 get_absolute_url( ) 메소드 반환값 지정
- **GET 요청**
  - 지정된 입력폼을 출력함. 폼 입력후 submit 하면 현재 URL을 POST로 요청함
- **POST 요청 : 입력 데이터에 대해 유효성 검사**
  - 유효성 검사 성공시 : 데이터 저장 후 success_url로 이동
  - 유효성 검사 실패시 : 입력 페이지로 이동

CreateView

2. View 활용

```
# article/views.py
article_new = CreateView.as_view(model=Article, fields='__all__')

# article/urls.py
path('new/', views.article_new, name='new' ),

# models.py
class Article(models.Model):
     ~ 생략 ~
     def get_absolute_url(self):
          return reverse('article:detail', args=[self.id])
```

```
# article/templates/article/article_form.html

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save" />
</form>
```

```
# 웹브라우저 요청
http://localhost:8000/article/new/
```

UpdateView

2. View 활용

```
# article/views.py
article_edit = UpdateView.as_view(model=Article, fields='__all__')

# article/urls.py
path('<pk>/edit/',views.article_edit, name='edit'),

# 웹브라우저 요청
http://localhost:8000/article/1/edit/
```

DeleteView

- 특정 모델 객체 삭제
- model : 필수 속성
- success_url : 필수 속성
- 기본 템플릿 경로 : "앱이름/모델소문자 _confirm_delete.html"
- GET 요청 : 삭제 작업을 진행할지 확인
- POST 요청 : 삭제 작업 후 success_url 로 이동

DeleteView

2. View 활용

```
# article/views.py
article_delete = DeleteView.as_view(model=Article, success_url='/article/')

# article/urls.py
path('<pk>/delete/',views.article_delete, name='delete'),
```

```
# template/article/article_confirm_delete.html
<h2>{{article}} 삭제 확인</h2>

정말 삭제하시겠습니까?
<form action="" method="post">
    {% csrf_token %}
    <a href="{% url 'article:index' %}">아니요. 취소!</a>
    <input type="submit" value="예. 삭제하겠습니다.">
</form>
```

```
# 웹브라우저 요청
http://localhost:8000/article/1/delete/
```

reverse_lazy

2. View 활용

```
# article/views.py
from django.urls import reverse

article_delete =
            DeleteView.as_view(model=Article,success_url=reverse('article:list'))
```

```
# 서버 오류

~ 생략 ~
The included URLconf 'myproject.urls' does not appear to have any patterns in it.
If you see valid patterns in the file then the issue is probably caused by a
circular import.
```

```
# article/views.py
from django.urls import reverse, reverse_lazy

article_delete =
      DeleteView.as_view(model=Article, success_url=reverse_lazy('article:list'))
```