



DJANGO



Django Form
Class View
Generic View

폼 처리하기

◆ 장고에서 폼을 처리하기 위한 기능을 제공

◆ HTML에서의 폼

- 사용자로부터 입력을 받기 위해 <form> 태그 활용
- 텍스트 입력, 체크박스 등의 간단한 폼은 기본 위젯을 사용
- 달력이나 슬라이드 바 등의 복잡한 요소는 자바스크립트나 CSS를 사용
- 폼을 구성하는 요소

입력을 위한 <input> 태그

폼의 내용을 전송할 곳을 지정하는 action 속성

보내는 형식을 지정하는 method 속성(장고는 POST만 지원)

폼 처리하기

◆ 장고의 폼 기능

- 장고는 폼의 기능을 단순화하고 자동화하여 안전하게 처리를 지원
 - 폼 생성에 필요한 데이터를 폼 클래스로 구조화
 - 폼 클래스의 데이터를 렌더링하여 HTML 폼 만들기
 - 사용자로부터 제출된 폼과 데이터를 수신하고 처리
- 장고의 폼 클래스는 폼을 기술하고 폼의 작동 및 결과화면을 결정
- 폼 클래스로 폼을 정의한 후 다음과 같이 렌더링 절차를 거침
 - 렌더링 할 객체를 뷰로 가져옴
 - 가져온 객체를 템플릿 시스템으로 전달
 - 템플릿 문법을 처리하여 HTML언어로 변환
- 폼 객체에는 데이터가 없을 수도 있다는 것을 고려(사용자가 입력해야 하므로)
- 언바운드 폼(데이터가 없는 폼)은 빈 폼이나 디폴트 값으로 채워진 폼을 사용자에게 전달
- 바운드 폼은(데이터가 있는 폼)은 데이터의 유효성을 검사하는 데 사용

폼 처리하기

◆ 예제

- 장고에서 <form> 엘리먼트의 기능을 제공하는 폼 클래스를 정의

```
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

- 위 코드가 템플릿 시스템을 통해 렌더링 되면 다음과 같은 코드가 나온다.

```
<label for="Your name">Your name: </label>
<input id="your_name" type="text" name="your_name" max_length="100">
```

- 각 색깔 별로 의미를 연결하여 이해

폼 처리하기

◆ 예제

- 렌더링 결과에 <form>태그나 submit버튼은 포함되지 않으므로 개발자가 직접 넣어줘야 함
- 템플릿 코드를 작성한다면 다음과 같이 할 수 있다.

```
<form action="req-path" method="post">
  {% csrf_token %}
  {{ form }}
  <input type="submit" value="Submit" />
</form>
```

- 위의 {{ form }} 부분은 폼 클래스의 객체를 담고 있는 변수
- 뷰에서 컨텍스트 변수에 포함되어 템플릿 시스템으로 넘겨짐
- 위에서 렌더링한 결과가 포함되어 사용자에게 보여지게 된다.

폼 처리하기

◆ 뷰에서 폼 클래스가 처리되는 흐름을 확인해 본다.

- 앞에서 작성한 NameForm 클래스와 name.html 템플릿을 사용하여 폼을 보여주고 폼 데이터를 전송 받아 값을 처리하는 뷰를 작성하는 실습 진행

➤ 실습 요구 조건

- ✓ 폼을 처리하는 뷰는 2개가 필요

폼을 보여주는 뷰

전송 받은 폼의 데이터를 처리하는 뷰

- ✓ 하나의 뷰로 두 개의 폼을 통합할 수 있다.(장고에서 권장하는 방식)

하나의 뷰에서 2가지 기능을 처리하는 경우 구분을 위해 HTTP메서드로 구분(GET/POST)

- ✓ NameForm클래스를 렌더링하여 name.html템플릿으로 변환되는 과정을 확인

폼 처리하기

◆ 테스트

- URLconf설정에 name/요청 경로 추가 - views.get_name 뷰를 사용하도록

```
urlpatterns = [  
    path('polls/', include('polls.urls', namespace='polls')),  
    path('admin/', admin.site.urls),  
    path('base/', views.inheTest ),  
    path('name/', views.get_name),  
]
```

폼 처리하기

◆ 테스트

- polls/views.py 파일에 URL요청에 응답할 get_name 뷰 작성

```
from django.http import HttpResponseRedirect
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)

def get_name(request):
    #POST방식이면, 데이터가 담긴 제출된 폼으로 간주하여 처리
    if request.method == 'POST':
        #request에 담긴 데이터로 클래스 폼을 생성
        form = NameForm(request.POST)
        #폼 데이터 유효성 체크
        if form.is_valid():
            #폼 데이터가 유효하면 데이터를 cleaned_data로 복사
            new_name = form.cleaned_data['your_name']
            #기타 추가적인 처리
            #
            #
            #결과를 렌더링한다. 또는 새로운 URL로 리다이렉션 하도록 한다.
            return render(request, 'name_ret.html', {'name': new_name})
            #return HttpResponseRedirect('/thanks/')

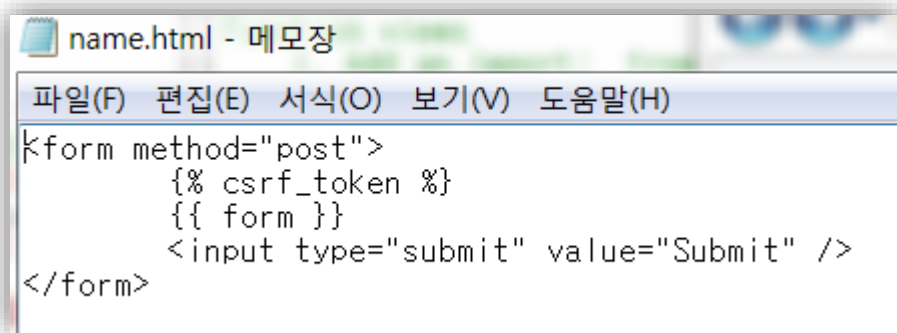
        #POST가 아니면(일반적으로 GET) 빈 폼을 보여준다.
        else:
            form = NameForm()

    return render(request, 'name.html', {'form': form})
```


폼 처리하기

◆ 테스트

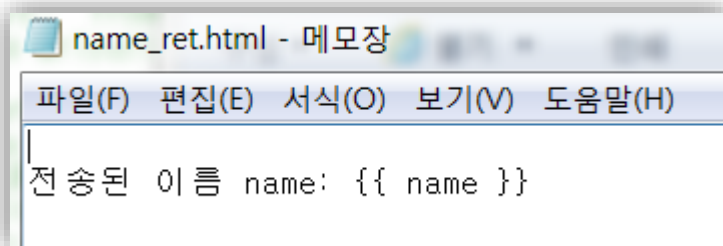
- GET요청 결과에 사용될 템플릿 파일 - polls/templates/name.html



A screenshot of a text editor window titled "name.html - 메모장". The menu bar includes "파일(F)", "편집(E)", "서식(O)", "보기(V)", and "도움말(H)". The text area contains the following HTML code:

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

- POST요청 결과에 사용될 템플릿 파일 - polls/templates/name_ret.html



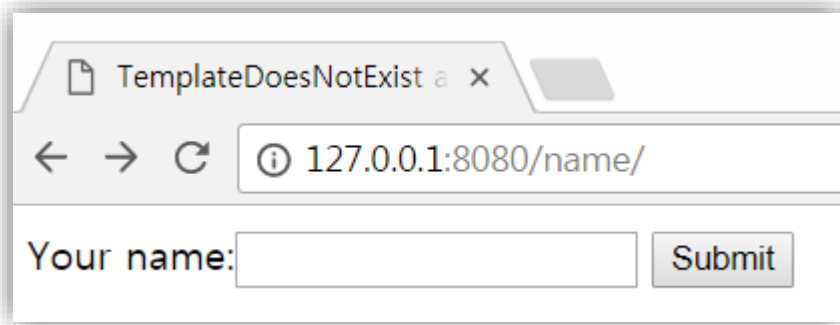
A screenshot of a text editor window titled "name_ret.html - 메모장". The menu bar includes "파일(F)", "편집(E)", "서식(O)", "보기(V)", and "도움말(H)". The text area contains the following HTML code:

```
전송된 이름 name: {{ name }}
```

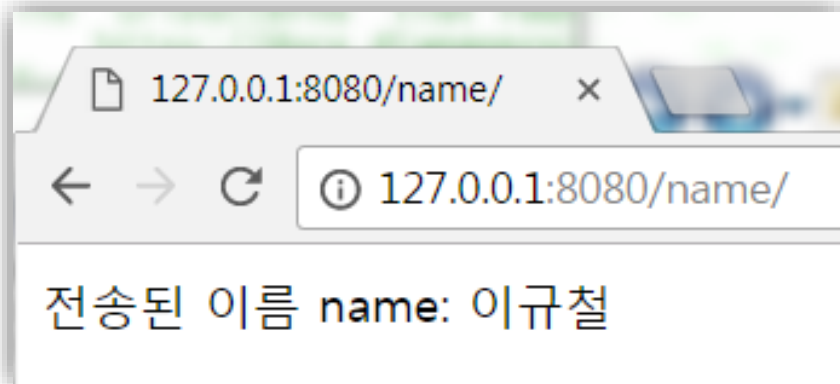
폼 처리하기

◆ 테스트

- name/으로 요청(GET)



- Submit - POST로 요청 결과



폼 처리하기

◆ 템플릿 시스템에서 폼 클래스를 템플릿으로 변환

- 템플릿 시스템에서는 템플릿 문법 및 폼 객체를 해석하여 HTML 템플릿으로 파일을 생성함
- 앞에서 작성한 {{ form }} 구문은 HTML의 <label>과 <input> 태그로 렌더링
- 이외에 다음과 같은 옵션도 있음

{{ form.as_table }} - <tr> 태그로 감싸서 테이블 셀로 렌더링

{{ form.as_p }} - <p> 태그로 감싸도록 렌더링

{{ form.as_ul }} - 태그로 감싸도록 렌더링

- 위의 경우 <table> 태그나 태그는 개발자가 직접 추가해야함
- 추가적으로 <label> 태그의 텍스트는 각 필드를 정의할 때 명시가능
- 지정하지 않으면 디폴트 레이블 텍스트로 지정됨

필드명의 첫 글자를 대문자로하고 _(밑줄)은 빈칸으로 변경하여 이름을 만들

<input> 의 id 속성도 id_필드명으로 만들어진다. <label for>에도 적용

클래스형 뷰

◆ 클래스형 뷰

- 뷰는 요청을 받아서 응답을 반환해주는 호출 가능한 객체이다.
- 장고는 뷰를 함수 또는 클래스로 작성할 수 있다.
- 간단한 뷰는 함수로 작성
- 함수형 보다는 클래스형 뷰를 사용하는 것이 유리
- 클래스의 개념과 구조를 활용할 수 있고 제네릭 뷰가 클래스 형으로 작성되어 있음

◆ 클래스형 뷰를 사용하는 경우 URLconf에 함수가 아닌 클래스형 뷰 사용에 대한 선언 필요

- 선언 예)

```
#urls.py
from django.urls import re_path
from myapp.views import MyView

urlpatterns = [
    re_path(r'^about/', MyView.as_view()),
]
```

클래스형 뷰

◆ as_view()메서드

- 장고의 URLresolver는 요청 파라미터를 함수에 전달
- 클래스형 뷰의 함수에 진입하기 위한 진입점에 해당되는 메서드
- as_view()메서드의 역할은 클래스의 인스턴스를 생성
- 생성된 인스턴스의 dispatch()메서드를 호출
- dispatch()메서드는 요청을 분석하여 GET/POST 등의 메서드를 알아내서 해당 이름의 메서드로 요청을 중계
- 만일 요청을 전달할 메서드가 정의되어 있지 않다면 HttpResponseNotAllowed익셉션 발생

클래스형 뷰

◆ MyView클래스 작성 예

- views.py에 작성

```
#views.py  
from django.http import HttpResponse  
from django.views.generic import View  
  
class MyView(View):  
    def get(self, request):  
        #뷰 로직 작성  
        return HttpResponse('result')
```

- MyView클래스는 View클래스를 상속
- View클래스에 as_view()와 dispatch()메서드가 정의되어 있다.

클래스형 뷰

◆ 클래스형 뷰의 장점 - 효율적인 메서드 구분

◆ 함수형 뷰와 비교하여 2가지 장점

- GET/POST 등의 메서드에 따른 처리 기능을 코딩할 때 if를 사용하지 않아도 됨(메서드명으로 구분)
- 다중상속과 같은 객체지향 기술이 가능
- 제네릭뷰 및 믹스인 클래스 등을 사용할 수 있기 때문에 재사용성 및 생산성이 증대됨

클래스형 뷰

◆ 첫 번째 장점에 대한 이야기

- 첫 번째 장점인 메서드 구분은 위의 예제에서 보여지듯이 뷰 함수 작성 시 HTTP메서드 이름을 사용(소문자로) - get, post, head 등

```
def get(self, request):
```

- 상속한 View클래스에 정의된 dispatch()메서드가 메서드 이름으로 구분(중계)
- post나 head등의 요청에 응답할 뷰가 필요하다면 클래스 내에 정의하면 됨.

◆ 두 번째 장점에 대한 이야기

- 장고에서 제공해주는 제네릭 뷰를 상속받아 작성
- 공통 기능을 반복적으로 정의하는 것을 줄이기 위해 제네릭 뷰사용
- 공통기능을 추상화하여 장고에 미리 만들어진 클래스형 뷰를 제네릭 뷰라 함

클래스형 뷰

◆ 장고에서 제공하는 제네릭 뷰의 4가지 분류

- Base View: 뷰 클래스를 생성하고 다른 Generic View의 부모
- Generic Display View: 객체의 리스트 또는 특정 객체의 상세 정보를 보여줌
- Generic Edit View: 폼을 통해 객체를 생성, 수정, 삭제하는 기능 제공
- Generic Date View: 날짜 기반 객체의 년/월/일 페이지로 구분하여 보여줌

클래스형 뷰

◆ 각 뷰에 대한 기능과 역할(일부만 표시하였음)

Generic View	Generic View 이름	기능 또는 역할
Base View	View	기본. 최상위 지네릭 뷰
	TemplateView	템플릿이 주어지면 해당 템플릿을 렌더링
	RedirectView	URL이 주어지면 해당 URL로 리다이렉트
Generic Display View	DatailView	하나의 객체에 대한 자세한 정보를 보여줌
	ListView	조건에 맞는 여러 개의 객체를 보여줌
Generic Edit View	FormView	폼이 주어지면 해당 폼을 보여줌
	CreateView	객체를 생성하는 폼을 보여줌
	UpdateView	기존 객체를 수정하는 폼을 보여줌
	DeleteView	기존 객체를 삭제하는 폼을 보여줌
Generic Date View	YearArchiveView	년도가 주어지면 해당 년도 객체를 보여줌
	MonthArchiveView	월이 주어지면 해당 월 객체를 보여줌
	DayArchiveView	날짜가 주어지면 해당 날짜 객체를 보여줌

클래스형 뷰와 폼

◆ 클래스형 뷰에서 폼을 처리하기

- 유효하지 않은 폼 데이터를 받은 경우도 처리할 수 있도록 작성
- 최초의 GET요청: 폼은 비어있거나 미리 채워진 데이터를 가짐
- 유효한 데이터를 가진 POST요청: 데이터를 처리. 주로 리다이렉트 하도록 함
- 유효하지 않은 데이터를 가진 POST요청: 일반적으로 에러메시지와 함께 폼을 다시 전송함

클래스형 뷰와 폼

◆ 클래스형 뷰 예시

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.views.generic import View

from .forms import MyForm

class MyFormView(View):
    form_class = MyForm
    initial = {'key': 'value'}
    template_name = 'form_template.html'

    def get(self, request, *args, **kwargs):
        form = self.form_class(initial=self.initial)
        return render(request, self.template_name, {'form': form})

    def post(self, request, *args, **kwargs):
        form = self.form_class(request.POST)
        if form.is_valid():
            #유효한 데이터를 가진 POST처리
            #cleaned_data를 이용하여 로직 처리
            return HttpResponseRedirect('/success/')
        #유효하지 않은 데이터를 가진 POST처리
        return render(request, self.template_name, {'form': form})
```

클래스형 뷰와 폼

- ◆ 앞의 예제는 View를 상속하여 작성했는데 폼처리용 지네릭 뷰인 FormView를 상속받아 처리하면 간결해 질 수 있다.
 - 다음과 같이 수정 가능

```
from django.views.generic.edit import FormView

from .forms import MyForm

class MyFormView(FormView):
    form_class = MyForm
    template_name = 'form_template.html'
    success_url = '/success/'

    def form_valid(self, form):
        #cleaned_data를 이용하여 로직 처리
        return super(MyFormView, self).form_valid(form)
```

장고 폼과 클래스형 뷰

- ◆ FormView 지네릭 뷰는 get, post 등을 구분이 내부에 정의되어 있음
- ◆ 다음 사항을 유의하여 필요한 부분만 만들어주면 됨
 - form_class
사용자에게 보여줄 폼을 정의한 forms.py파일 내의 클래스 이름
 - template_name
폼을 포함하여 렌더링 할 템플릿 파일 이름
 - success_url
MyFormView의 처리가 성공적으로 완료되었을 때 리다이렉트할 URL
 - form_valid() 함수
유효한 폼 데이터로 처리할 로직 코딩. 반드시 super()함수 호출

Generic View의 Attributes

속성	설명	디폴트
model	작업대상 데이터가 들어있는 모델을 지정 model대신 queryset으로 지정 할 수 있음.	
queryset	작업대상이 되는 QuerySet 객체를 지정	
template_name	렌더링할 템플릿 파일명을 지정	<model_name>_list.html <model_name>_detail.html 외
context_object_name	템플릿 파일에서 사용할 컨텍스트 변수명을 지정	object_list, object 외
paginate_by	페이지기능이 활성화된 경우 페이지당 몇개항목을 출력할지 정수로 지정	
date_field	날짜기반뷰에서 기준이 되는 필드를 지정, 이 필드를 기준으로 년/월/일을 검사	
form_class	폼을 만드는데 사용할 클래스를 지정	
initial	폼에 사용할 초기데이터를 사전{ }으로 지정	
fields	폼에 사용할 필드를 지정	
success_url	폼에 대한 처리가 성공한 후 리다이렉트할 url지정	

Generic View의 methods

메소드	설명
get_queryset()	출력객체를 검색하기 위한 대상 QuerySet 객체 또는 출력대상인 객체리스트를 반환한다. 디폴트는 queryset속성값을 반환한다. queryset 속성이 지정되지 않는 경우는 모델 매니저 클래스의 all()메소드를 호출해 QuerySet 객체를 생성하고 이를 반환한다.
get_context_data(**kwargs)	템플릿에서 사용한 컨텍스트 데이터를 반환
form_valid(form)	제출된 폼이 유효성검사를 통과하면 호출되는 메소드 또는 get_success_url()메소드가 반환하는 url로 리디렉트를 수행

오버라이딩 가능 여부

◆ Attributes

	TemplateView	ListView	DetailView	FormView	CreateView UpdateView	DeleteView	DateView
model		○	○		○	○	○
queryset		○	○		○	○	○
template_name	○	○	○	○	○	○	○
context_object_name		○	○	○	○	○	○
paginate_by		○					○
date_field							○
form_class				○	○		
initial				○	○		
fields					○		
success_url				○	○	○	

◆ Methos

	TemplateView	ListView	DetailView	FormView	CreateView UpdateView	DeleteView	DateView
get_queryset()		○	○		○	○	○
get_context_data()	○	○	○	○	○	○	○
form_valid()				○	○		

