Django Form

구분	Django

💡 Article Model Class에 필드에 대한 정보를 작성하고 이를 Form에 맵핑하기 위해서는 Form Class에 필드를 재정의해야 한다.
→ Model과 중복 되는 부분이 많아 번거롭다.
→ ModelForm을 사용하면 이러한 Form을 더 쉽게 작성할 수 있다!!!

Django ModelForm

ModelForm Class

- Model을 통해 Form Class를 만들 수 있는 helper class 이다.
- ModelForm은 Form과 똑같은 방식으로 View 함수에서 사용한다.

ModelForm 선언

- form 라이브러리에서 파생된 ModelForm 클래스를 상속받음
- 정의한 ModelForm 클래스 안에 Meta 클래스를 선언
- 어떤 모델을 기반으로 form을 작성할 것인지에 대한 정보를 Meta 클래스에 지정

```
# articles/forms.py

from django import forms
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'
```

Meta Class

- ModelForm의 정보를 작성하는 곳

- ModelForm을 사용할 경우 참조할 모델이 있어야 하는데, Meta class의 model 속성이 이를 구성
- 참조하는 모델에 정의된 field 정보를 Form에 적용
  - fields 속성에 `'__all__'`를 사용하면 모델의 모든 필드를 포함할 수 있음

```
class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'
```

  - exclude 속성을 사용하면 모델에서 포함하지 않을 필드를 지정할 수 있음

```
class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        exclude = ('title',)
```

ModelForm 구현하기 - view 함수 작성

`is_valid()` method

- 유효성 검사를 실행하고 데이터가 유효한지 여부를 boolean 으로 반환
- 데이터 유효성 검사를 보장하기 위한 많은 테스트에 대해 Django는 `is_valid()`를 제공하여 개발자의 편의를 도움

The `save()` method

- form 인스턴스에 바인딩된 데이터를 통해 데이터베이스 객체를 만들고 저장
- ModelForm 하위 클래스는 키워드 인자 `instance` 여부를 통해 생성할지, 수정할지 결정
  - 제공되지 않은 경우 save()는 지정된 모델의 새 인스턴스를 만듦 (`CREATE`)
  - 제공되면 save()는 해당 인스턴스를 수정(`UPDATE`)

```
# CREATE

form = ArticleForm(request.POST)
article = form.save()
```

```
# UPDATE

form = ArticleForm(request.POST, instance=article)
form.save()
```

create

- 유효성 검사를 통과하면 데이터 저장 후 상세 페이지로 리다이렉트
- 통과하지 못하면 작성 페이지로 리다이렉트

```
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save()
            return redirect('articles:detail', article.pk)
        return redirect('articles:create')
    else:
        form = ArticleForm()
        context = {'form':form }
        return render(request, 'articles/create.html', context)
```

- 중복되는 로직 정리하면 코드는 아래와 같다.

```
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save()
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm()

    context = {'form':form }
    return render(request, 'articles/create.html', context)
```

Update

- ModelForm의 인자 instance는 수정 대상이 되는 객체(기존 객체)를 지정
- `request.POST` : 사용자가 form을 통해 전송한 데이터(새로운 데이터)
- `instance` : 수정이 되는 대상

update view 수정

```
def update(request, pk):
    article = Article.objects.get(pk=pk)
    if request.method == 'POST':
        form = ArticleForm(request.POST, instance=article)
        if form.is_valid():
            form.save()
            return redirect('articles:detail', pk=article.pk)
    else:
        form = ArticleForm(instance=article)
    context = {'form': form,
               'article':article }
    return render(request, 'articles/update.html', context)
```

update template 수정

```
{% extends 'base.html' %}

{% block content %}
  <h1>글수정</h1>
  <hr>

  <form action="{% url 'articles:update' article.pk %}" method="POST">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit">
  </form>

  <hr>
  <a href="{% url 'articles:detail' article.pk %}">돌아가기</a>
{% endblock content %}
```

Form과 ModelForm

- ModelForm이 Form보다 더 좋은 것이 아니라 각자 역할이 다른 것
- Form
  - 사용자의 입력을 필요로 하며 직접 입력 데이터가 DB 저장에 사용되지 않거나 일부 데이터만 사용될 때
  - 예 - 로그인, 사용자의 데이터를 받아 인증 과정에서만 사용 후 별도로 DB에 저장하지 않음
- ModelForm

- 사용자의 입력을 필요로 하며 입력을 받은 것을 그대로 DB 필드에 맞춰 저장할 때
- 데이터의 유효성 검사가 끝나면 데이터를 각각 어떤 레코드에 맵핑 해야 할지 이미 알고 있기 때문에 곧바로 save() 호출이 가능하다.