장고 핵심 기능 - Template

템플릿 설정 항목

❖ settings.py

- TEMPLATES
  - 장고의 코어 템플릿 엔진 : DTL Django Template Language
  - Jinja 템플릿 엔진 지원

```
TEMPLATES = [
    {
        'BACKEND' : django.template.backends.django.DjangoTemplates',
         :
]
```

- BACKEND
  - django.template.backends.django.DjangoTemplates
  - django.template.backends.jinja2.Jinja2

- DIRS
  - 템플릿 파일이 위치하는 디렉토리 목록
  - 어플리케이션의 템플릿 보다 우선 적용

```
'DIRS' : [ os.path.join(BASE_DIR, 'templates')],
```

템플릿 설정 항목

❖ settings.py

- APP_DIRS
  - 템플릿 파일을 찾을 때, 애플리케이션 내의 템플릿 디렉토리에서도 찾을지 여부를 지정
  - 디폴트는 False
  - startproject 명령에 의해 settings.py 파일이 만들어질 때는 True로 설정됨

- OPTIONS
  - 템플릿 엔진에 따라 해당하는 옵션 항목들을 설정

템플릿 설정 항목

❖ settings.py

- context_processors :
  - 웹 요청에 들어 있는 파라미터들을 인자로 받아서 커텍스트 데이터로 사용될 사전을 만드는 호출 가능한 객체를 지정
  - 보통 함수로 정의
  - 이 함수들이 반환하는 사전은 최종 컨텍스트 데이터를 만들때 추가
  - 디폴트는 빈 리스트
- debug:
  - 템플릿 디버그 모드를 설정.
- loaders
  - 템플릿 로더 클래스 지정
  - 로더는 템플릿 파일을 찾아 메모리로 로딩하는 역할을 수행
- string_if_invalid
  - 템플릿 변수가 잘못된 경우 대신 사용할 문자열을 지정
  - 디폴트는 공백 문자열
- file_charset
  - 문자셋 지정.
  - 디폴트는 utf-8

템플릿 내부 처리 과정

❖ 템플릿 설정에 따라 Engine 객체를 생성

- settings.py에 설정된 TEMPLATES 설정 항목에 지정된 값들 사용

❖ 템플릿 파일 로딩 및 Template 객체를 생성

- 디폴드 로더
  - django.template.loaders.filesystem.Loader
    - 템플릿 파일을 찾기위해 TEMPLATES의 DIRS 항목에 지정된 디렉토리를 검색
    - DIRS 항목이 비어 있으면 로던느 검색을 수행하지 않음
  - django.template.loaders.app_directories.Loader
    - 템플릿 파일을 찾기위해 각 애플리케이션 디렉터리 하위에 있는 templates/ 디렉토리를 검색
    - INSTALLED_APPS 설정 항목에 등록된 앱들이 대상
    - TEMPLATES 설정 항목의 APP_DIR 항목이 True인 경우만 동작

❖ 렌더링을 실시해, 최종 HTML 텍스트 파일을 생성

템플릿 내부 처리 과정

❖ 컨텍스트 생성 옵션

- 템플릿 엔진 설정 항목의 context_processors 옵션 항목

- django.template.context_processors.debug
  - 웹 요철 처리 과정에서 사용된 SQL 쿼리 정보를 담은 sql_queries 변수가 컨텍스트 데이터에 추가
- django.template.context_processors.request
  - 현 요청의 HttpRequest를 지칭하는 request 변수가 최종 컨텍스트 데이터에 추가
- django.contrib.auth.context_processors.auth
  - 로그인 사용자를 지칭하는 user 변수 및 그 사용자의 권한을 지칭하는 perms 변수가 최종 컨텍스트 데이터에 추가
- django.contrib.messages.context_processors.messages
  - 메시지 리스트를 지칭하는 messages 변수와 메시지 레벨을 지칭하는 DEFAULT_MESSAGE_LEVELS 변수가 최종 컨텍스트 데이터에 추가
- django.template.context_processors.csrf
  - {% csrf_token %} 템플릿 태그 처리에 필요한 토큰이 최종 컨텍스트 데이터에 추가

15.3 템플릿 렌더링 실습

```
$ python manage.py shell

>>> from django.template import Template, Contex
>>> template = Template('My name is {{ my_name }}."
>>> context = Context({"my_name": "John"})
>>> template.render(context)
'My name is John'

>>> context = Context({ "my_name" : "Jane"})
>>> template.render(context)
'My name is Jane'

>>> t = Template("My name is {{ person.first_name }}.")
>>> d = { 'person' : { 'first_name': 'Joe', 'last_name': 'Johnson'}}
>>> t.render(Context(d))
'My name is Joe'
```

15.3 템플릿 렌더링 실습

```
>>> class PersonClass: pass
...
>>> p = PersonClass()
>>> p.first_name = 'Ron'
>>> p.last_name = 'Nasty'
>>> t.render(Context({'person': p}))
'My name is Ron. '

>>> tpl = Template('The first element in the list is {{ test_list.0 }}.')
>>> t_list = ['Larray', 'Curly', 'Moe']
>>> the.render(Context({'test_list': t_list}))
'The first element in the list is Larry.'

>>> class PersonClass2:
...     def name(self):
...         return "Samantha"
...
>>> tt = Template(Context( {'person': PersonClass2}))
'My name is Samantha.'
```

제네릭 뷰의 디폴트 템플릿

❖ **제너릭 뷰는 디폴트 템플릿 명을 가짐**

- `template_name` 속성을 지정하지 않은 경우 사용할 템플릿 파일 이름

- `<app_label>/<model_name 소문자>_<template_name_suffix>.html`

제너릭 뷰 이름	template_name_suffix	예시(blog앱의 Post 모델)
ListView	_list	blog/post_list.html
DetailView	_detail	blog/post_detail.html
ArchiveIndexView	_archive	blog/post_archive.html
YearArchiveView	_archive_year	blog/post_archive_year.html
MonthArchiveView	_archive_month	blog/post_archive_month.html
WeekArchiveView	_archive_week	blog/post_archive_week.html
DayArchiveView	_archive_day	blog/post_archive_day.html
TodayArchiveView	_archive_today	blog/post_archive_today.html
DateDetailView	_detail	blog/post_detail.html
CreateView	_form	blog/post_form.html
UpdateView	_form	blog/post_form.html
DeleteView	_confirm_delete	blog/post_confirm_delete.html

{% include %} 태그

❖ {% include %} 태그

- 공통된 코드를 재활용하면서 코드 중복을 줄임
- 공통적으로 사용할 수 있는 템플릿 파일을 따로 만들어 둠
- {% include %} 태그로 공통 파일을 가져와 사용

```
{% include "foo/bar.html" %}  # 템플릿 파이명을 따움표로 묶음
{% include templage_name %}   # 템플릿 파일명이 들어 있는 변수 사용
```

{% include %} 태그

❖ foo/bar.html

```
{{ greeting }}, {{ person | default:"friend" }}



{% include "foo/bar.html" %}

{% include "foo/bar.html" with person="Jane" greeting="Hello" %}


{% include "foo/bar.html" with greeting="HI" only %}
```

{% static %} 템플릿 태그

❖ {% static %} 템플릿 태그

- 정적 파일
  - Image, 자바스크립트, CSS 파일들

- {% static arg %}
  - STATIC_URL 설정 항목과 arg로 주어진 정적 파일을 합쳐서 URL을 만듦

- settings.py 파일

```
STATIC_URL = '/static/'
```

- 템플릿 *.html 파일

```
{% load static %}
<img src="{% static "images/hi.jpg" %}" alt="Hi!"/>
```

- # 템플릿 *.html 파일에서 {% static %} 태그를 처리한 결과

```
<img src="/static/images/hi.jpg" alt="Hi!" />
```

{% static %} 템플릿 태그

❖ {% static %} 템플릿 태그

- 예

```
{% load static %}
<link rel="stylesheet" href="{% static user_stylesheet %}"
type="text/css" />
```

- URL을 템플릿 변수에 저장

```
{% load static %}
{% static "images/hi.jpg" as myphoto %}
<img src="{{myphoto}}"/>
```

staticfiles 애플리케이션 기능

❖ staticfiles 애플리케이션 기능

- 정적 파일을 처리하기위해 장고는 staticfiles 애플리케이션을 제공
- 애플리케이션 개발환경에서 사용
- 상용 환경에선느 Apache, Nginx 등의 웹 서버를 사용

- 개발환경에서의 웹서버 : runserver
- staticfiles 앱을 사용해서 정적 파일을 처리

- DEBUG 모드가 True인 경우만 staticfiles 앱이 동작

staticfiles 애플리케이션 기능

❖ 정적 파일 처리 순서

- 웹 클라이언트(브라우저)는 URL을 포함한 웹 요청을 서버에 보냄
- 장고는 웹 요청 URL이 STATIC_URL로 시작하는지 검사
- URL이 STATIC_URL로 시작하면, 장고는 staticfiles 앱을 사용해 처리를 시작
- staticfiles 앱은 STATICFILES_FINDERS에 지정된 파인더로 정적 파일을 검색
- 파인더에 따라 검색하는 디렉터리가 달라짐
- 정작 파일을 찾으면 해당 파일을 웹 클라이언트에 응다

staticfiles 애플리케이션 기능

❖ settings.py

```
# startproject 명령 실행 시 등록된 내용
INSTALLED_APPS = (
    :
    'django.contrib.staticfiles',
    :
)

# startproject 명령 실행 시 지정된 내용
STATIC_URL = '/static/'

# 디폴트 설정
STATICFILES_FINDERS = (
    "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder",
)

# 수동으로 추가
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```