

220820 Django

1. Django 시작하기

1.1 프레임워크 (Framework)

- 서비스 개발에 필요한 기능들을 미리 구현해서 모아 놓은 것
- 소프트웨어 프레임워크
 - 복잡한 문제를 해결하거나 서술하는 데 사용되는 기본 개념 구조
- Django를 배워야하는 이유
 - python으로 작성된 프레임워크
 - 검증된 웹 프레임워크
 - 유명한 서비스들이 사용한다는 것 → 안정적으로 서비스를 할 수 있다는 검증

1.2 Web

- WWW (World Wide Web)
 - 전 세계에 퍼져 있는 거미줄 같은 연결망
- 인터넷을 이용한다는 건, 전세계 컴퓨터가 연결되어 있는 하나의 인프라를 이용하는 것

1.3 클라이언트와 서버

1.3.1 클라이언트

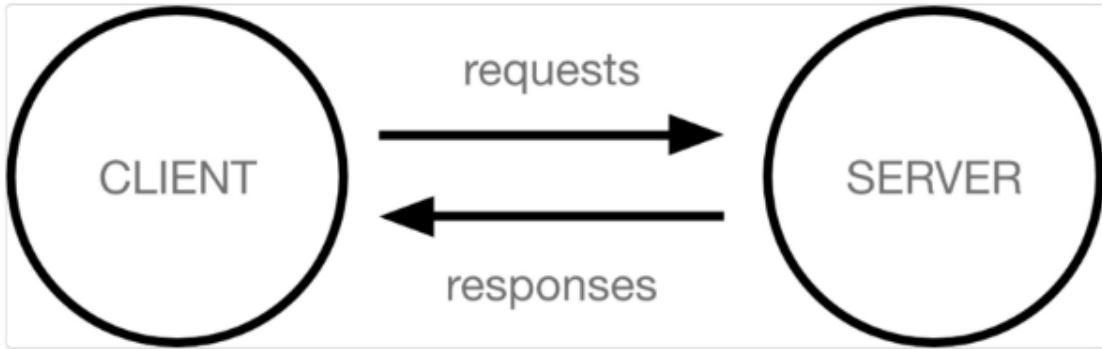
- 서비스를 요청하는 주체
- 웹 사용자의 인터넷에 연결된 장치
 - ex) wi-fi에 연결된 컴퓨터 또는 모바일
- Chrome 또는 Firefox와 같은 웹 브라우저

1.3.2 서버

- 요청에 대해 서비스를 응답하는 주체
- 웹 페이지, 사이트 또는 앱을 저장하는 컴퓨터
- 클라이언트가 웹 페이지에 접근하려고 할 때, 서버에서 클라이언트 컴퓨터로 웹 페이지 데이터를 응답해 사용자의 웹 브라우저에 표시됨

1.3.3 클라이언트와 서버 상호작용

- 대부분의 웹사이트는 클라이언트-서버 구조를 기반으로 동작
- 클라이언트와 서버 역시 하나의 컴퓨터이며, 아래와 같이 상호작용 함



- ex) 구글 홈페이지 접속
 1. 결론적으로 구글 컴퓨터에게 'Google 홈페이지.html' 파일을 달라고 요청하는 것
 2. 요청을 받고 'Google 홈페이지.html' 파일을 인터넷을 통해서 우리 컴퓨터에게 응답해 줌
 3. 전달받은 파일을 웹 브라우저가 우리가 볼 수 있도록 해석해 주는 것
 - 여기서 파일을 달라고 요청한 컴퓨터, 웹 브라우저를 **클라이언트**
 - 파일을 제공한 컴퓨터, 프로그램을 **서버**

1.3.4 정리

- 우리가 사용하는 웹은 클라이언트-서버 구조
- Django는 서버를 구현하는 웹 프레임워크

1.2 Web browser와 Web page

1.2.1 웹 브라우저

- 웹에서 페이지를 찾아 보여주고, 사용자가 하이퍼링크를 통해 다른 페이지로 이동할 수 있도록 하는 프로그램
- 웹 페이지 파일을 우리가 보는 화면으로 바꿔주는 프로그램
→ **렌더링(rendering)** 프로그램
- 예시
 - 우리가 보고있는 웹 페이지는 사실 HTML 문서 파일 하나
 - 웹 페이지 코드를 받으면 우리가 보는 화면처럼 바꿔주는 것이 **웹 브라우저**
 - HTML / CSS / JS 등의 코드를 읽어 실제 사람이 볼 수 있는 화면으로 만들어 줌

1.2.2 웹 페이지

- 웹에 있는 문서
 - 우리가 보는 화면 각각 한 장 한장이 웹페이지
- 정적 웹 페이지
 - **Static Web page**
 - 있는 그대로를 제공하는 것(served-as-is)을 의미
 - 우리가 지금까지 작성한 웹 페이지이며 한번 작성된 HTML 파일의 내용이 변하지 않고 모든 사용자에게 동일한 모습으로 전달되는 것
 - 서버에 미리 저장된 HTML 파일 그대로 전달된 웹 페이지
 - 같은 상황에서 모든 사용자에게 동일한 정보를 표시
- 동적 웹페이지
 - **Dynamic Web page**
 - 사용자의 요청에 따라 웹 페이지에 추가적인 수정이 되어 클라이언트에게 전달되는 웹 페이지
 - 웹 페이지의 내용을 바꿔주는 주체 → **서버**
 - 서버에서 동작하고 있는 프로그램이 웹 페이지를 변경해줌
 - 사용자의 요청을 받아서 적절한 응답을 만들어 주는 프로그램을 쉽게 만들 수 있게 도와주는 프레임워크가 **Django**
 - 다양한 서버사이드 프로그래밍 언어 사용 가능
 - 파일을 처리하고 데이터베이스와의 상호작용이 이루어짐

2. Django 구조 (MTV Desing Pattern)

2.1 Design pattern

- 자주 사용되는 구조를 일반화해서 하나의 공법으로 만들어 둔 것
- 소프트웨어에서의 관점
 - 각기 다른 기능을 가진 다양한 응용 소프트웨어를 개발할 때 공통적인 설계 문제가 존재하며, 이를 처리하는 해결책 사이에도 공통점이 있다는 것을 발견
→ 이러한 유사점이 패턴

2.1.1 소프트웨어 디자인 패턴

- 클라이언트-서버 구조도 소프트웨어 디자인 패턴 중 하나
- 자주 사용되는 소프트웨어 구조를 마치 건축의 공법처럼 일반적인 구조화를 해둔 것
- 특정 맥락에서 공통적으로 발생하는 문제에 대해 재사용 가능한 해결책을 제시
- 소프트웨어 디자인 패턴의 장점
 - 디자인 패턴을 알고 있다면 서로 복잡한 커뮤니케이션이 매우 간단해짐
 - 다수의 엔지니어들이 일반화된 패턴으로 소프트웨어 개발을 할 수 있도록 한 규칙
 - 커뮤니케이션의 효율성을 높이는 기법

2.2 MVC 소프트웨어 디자인 패턴

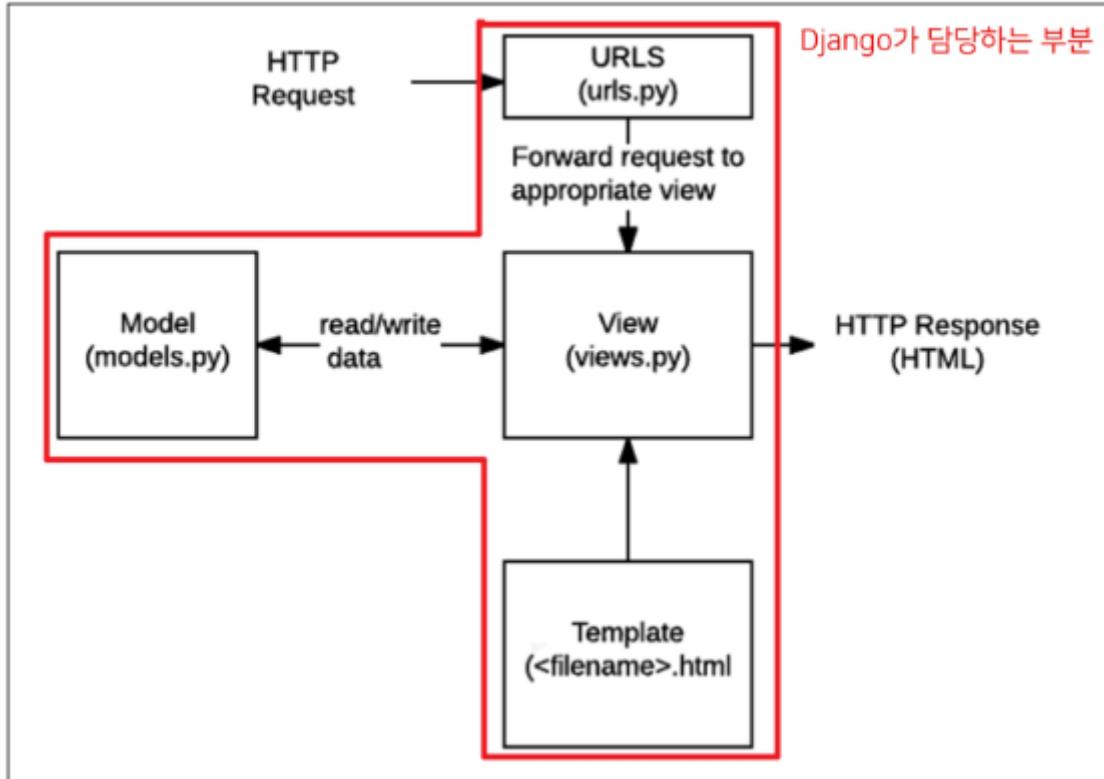
- **MVC (Model-View-Controller)**
 - 데이터 및 논리제어를 구현하는데 널리 사용되는 소프트웨어 디자인 패턴
 - 하나의 큰 프로그램을 세가지 역할로 구분한 개발 방법론
 - **model** : 데이터와 관련된 로직을 관리
 - **View** : 레이아웃과 화면을 처리
 - **Controller** : 명령을 model과 view 부분으로 연결
- MVC 소프트웨어 디자인 패턴의 목적
 - 관심사 분리
 - 더 나은 업무의 분리와 향상된 관리를 제공
 - 각 부분을 독립적으로 개발할 수 있어, 하나를 수정하고 싶을 때 모두 수정하지 않아도 됨
 - 개발 효율성 및 유지보수가 쉬워짐
 - 다수의 멤버로 개발하기 용이함

2.3 Django에서의 디자인 패턴

- **Django** 는 MVC 패턴을 기반으로 한 **MTV 패턴** 을 사용

MVC	MTV
View	Template
Model	Model
Controller	View

- **Model**
 - 데이터와 관련된 로직을 관리
 - 응용프로그램의 데이터 구조를 정의하고 데이터베이스의 기록을 관리
 - MVC 패턴에서 Model의 역할에 해당
- **Template**
 - 레이아웃과 화면을 처리
 - 화면상의 사용자 인터페이스 구조와 레이아웃을 정의
 - MVC 패턴에서 View의 역할에 해당
- **View**
 - Model & Template과 관련된 로직을 처리해서 응답을 반환
 - 클라이언트의 요청에 대해 처리를 분기하는 역할
 - 동작 예시
 - 데이터가 필요하다면 model에 접근해서 데이터를 가져옴
 - 가져온 데이터를 template로 만들어 화면 구성
 - 구성된 화면을 응답을 만들어 클라이언트에게 반환
 - MVC 패턴에서 Controller의 역할에 해당



2.4 정리

- **Django**는 MTV 패턴을 가지고 있음
 - Model : 데이터 관련
 - Template : 화면 관련
 - View : Model & Template 중간 처리 및 응답 반환

3. 기본 설정

3.1 가상환경

- `python -m venv venv` 가상환경 만들기
- `source venv/Scripts/activate` 가상환경 활성화

3.2 Django 설치

```
$ python -m venv venv
$ source venv/Scripts/activate

$ pip install django==3.2.13
$ pip freeze > requirements.txt
$ pip install -r requirements.txt

$ django-admin startproject pjt .
$ python manage.py runserver
$ python manage.py startapp articles
```

- `pip install django==3.2.13` 3.2 LTS 버전을 명시하고 설치
 - LTS (Long Term Support 장기지원버전)
 - 일반적인 경우보다 장기간에 걸쳐 지원하도록 고안된 소프트웨어의 버전
 - 컴퓨터 소프트웨어의 제품 수명주기 관리 정책
 - 배포자는 LTS 확정을 통해 장기적이고 안정적인 지원을 보장함
- `pip freeze > requirements.txt` 패키지 목록 생성
- `pip install -r requirements.txt` 목록 속 패키지 설치

3.3 Django Project

- `django-admin startproject 프로젝트명 .` 프로젝트 생성
 - Project 이름에는 Python이나 Django에서 사용중인 키워드 및 . 사용 불가
 - . 을 붙이지 않을 경우 현재 디렉토리에 프로젝트 디렉토리를 새로 생성하게 됨
- `python manage.py runserver` 서버 실행

3.3.1 프로젝트 구조

- `__init__.py`
 - Python에게 이 디렉토리를 하나의 Python 패키지로 다루도록 지시
 - 별도의 추가 코드 작성 X
- `asgi.py`
 - Django 애플리케이션이 비동기식 웹 서버와 연결 및 소통하는 것을 도움
- `settings.py`
 - Django 프로젝트 설정을 관리
- `urls.py`
 - 사이트와 `url`과 적절한 `views`의 연결을 지정
- `wsgi.py`
 - Django 애플리케이션이 웹 서버와 연결 및 소통하는 것을 도움
- `manage.py`
 - Django 프로젝트와 다양한 방법으로 상호작용 하는 커맨드라인 유틸리티

```
$ python manage.py <command> [options]
```

3.4 Django Application

- `python manage.py startapp articles` 앱 생성
 - 일반적으로 애플리케이션 이름은 **복수형**으로 작성하는 것을 권장
 - 애플리케이션 등록
 - 프로젝트에서 앱을 사용하기 위해서는 반드시 `INSTALLED_APPS` 리스트에 추가

```
# project/settings.py

INSTALLED_APPS = [
    'articles'
]
```

- `INSTALLED_APPS`
 - Django installation에 활성화 된 모든 앱을 지정하는 문자열 목록

3.4.1 애플리케이션 구조

- `admin.py`
 - 관리자용 페이지를 설정하는 곳
- `apps.py`
 - 앱의 정보가 작성된 곳
 - 별도의 추가 코드 작성 X
- `models.py`
 - 애플리케이션에서 사용하는 Model을 정의하는 곳
 - MTV 패턴의 M에 해당
- `tests.py`
 - 프로젝트의 테스트 코드를 작성하는 곳
- `views.py`
 - view 함수들이 정의되는 곳
 - MTV 패턴의 V에 해당

3.5 Project & Application

3.5.1 Project

- 프로젝트는 앱의 집합 (collection of apps)
- 프로젝트에는 여러 앱이 포함될 수 있음
- 앱은 여러 프로젝트에 있을 수 있음

3.5.2 Application

- 앱은 실제 요청을 처리하고 페이지를 보여주는 등의 역할을 담당
- 일반적으로 앱은 하나의 역할 및 기능 단위로 작성하는 것을 권장
- 주의사항
 - 반드시 생성 후 등록
 - INSTALLED_APPS에 먼저 작성(등록)하고 생성하려면 앱이 생성되지 않음
 - 순서 지키기 권장

```
INSTALLED_APPS = [  
    'articles',      # Local apps  
    'haystack',      # Third party apps  
    'Django.contrib.admin' # Django apps  
]
```

4. 요청과 응답

4.1 URL

- URL → VIEW → TEMPLATE 데이터의 흐름 이해하기

```
# urls.py

from django.contrib import admin
from django.urls import path
from articles import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', view.index)
]
```

4.2 View

- HTTP 요청을 수신하고 HTTP 응답을 반환하는 함수 작성
- Template에게 HTTP 응답 서식을 맡김

```
# articles/views.py

def index(request):
    return render(request, 'index.html')
```

4.2.1 `render(request, template_name, context)`

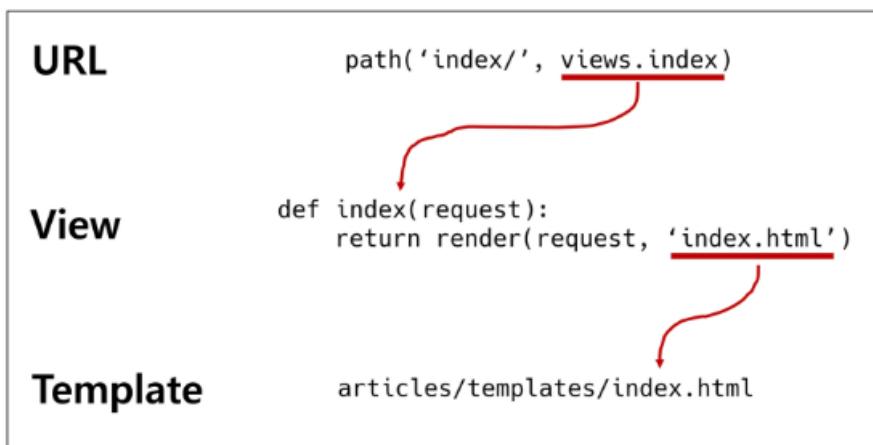
- 주어진 템플릿을 주어진 컨텍스트 데이터와 결합하고 렌더링 된 텍스트와 함께 `HttpResponse(응답)` 객체를 반환하는 함수
- `request`
 - 응답을 생성하는 데 사용되는 요청 객체
- `template_name`
 - 템플릿의 전체 이름 또는 템플릿 이름의 경로
- `context`
 - 템플릿에서 사용할 데이터(딕셔너리 타입으로 작성)

4.3 Templates

- 실제 내용을 보여주는데 사용되는 파일
- 파일의 구조나 레이아웃을 정의
- Template 파일의 기본 경로
 - app 폴더 안의 templates 폴더
 - app_name/templates
- 템플릿 폴더 이름은 반드시 `templates`라고 지정

4.4 코드 작성 순서

- Django에서의 코드 작성은 `URL → VIEW → TEMPLATE` 순으로 작성
- 데이터의 흐름 순서



5. Django Template

- Django Template
 - 데이터 표현을 제어하는 도구이자 표현에 관련된 로직
 - Django Template을 이용한 HTML 정적 부분과 동적 컨텐츠 삽입
- Django Template System
 - 데이터 표현을 제어하는 도구이자 표현에 관련된 로직을 담당
- Django Template Language (DTL)
 - Django template에서 사용하는 built-in template system
 - 조건, 반복, 변수 치환, 필터 등의 기능을 제공
 - python처럼 일부 프로그래밍 구조(if, for 등)를 사용할 수 있지만 python 코드로 실행되는 것은 아님
 - Django 템플릿 시스템은 단순히 Python이 HTML에 포함된 것은 아님
 - 프로그래밍적 로직이 아니라 프레젠테이션을 표현하기 위한 것임을 명심

5.1 DTL Syntax

5.1.1 Variable `{{ variable }}`

- 변수명은 영어, 숫자와 밑줄(_)의 조합으로 구성될 수 있으나 밑줄로는 시작 할 수 없음
 - 공백이나 구두점 문자 또한 사용할 수 없음
- dot(.)를 사용하여 변수 속성에 접근할 수 있음
- render()의 세번째 인자로 `{'key':value}`와 같이 딕셔너리 형태로 넘겨주며,
여기서 정의한 key에 해당하는 문자열이 template에서 사용 가능한 변수명이 됨

5.1.2 Filters `{{ Filters }}`

- 표시할 변수를 수정할 때 사용
 - `{{ name|lower }}` : name 변수를 모두 소문자로 출력
- chained가 가능하며 일부 필터는 인자를 받기도 함
 - `{{ name|truncatetwords:30 }}`

5.1.3 Tags `{% tag %}`

- 출력 텍스트를 만들거나, 반복 또는 논리를 수행하여 제어
흐름을 만드는 등 변수보다 복잡한 일들을 수행
- 일부 태그는 시작과 종료 태그가 필요
 - `{% if %}{% endif %}`

5.1.4 Comments `{# #}`

- Django template에서 라인의 주석을 표현하기 위해 사용
- 아래처럼 유효하지 않은 템플릿 코드가 포함될 수 있음
 - `{# {% if %} text {% endif %} #}`
- 한 줄 주석에만 사용할 수 있음(줄 바꿈이 허용되지 않음)
- 여러 줄 주석은 `{% comment %}` 와 `{% endcomment %}` 사이에 입력

5.2 템플릿 상속(Template inheritance)

- 템플릿 상속은 기본적으로 코드의 재사용성에 초점을 맞춤
- 템플릿 상속을 사용하면 사이트의 모든 공통 요소를 포함하고, 하위 템플릿이 재정의
(override) 할 수 있는 블록을 정의하는 기본 'skeleton' 템플릿을 만들 수 있음

5.2.1 템플릿 상속에 관련된 태그

- `{% extends '' %}`
 - 자식(하위) 템플릿이 부모 템플릿을 확장한다는 것을 알림
 - 반드시 템플릿 최상단에 작성되어야 함 (2개 이상 사용 불가)
- `{% block content %}{% endblock content %}`
 - 하위 템플릿에서 재지정(overridden)할 수 있는 블록을 정의
 - 하위 템플릿이 채울 수 있는 공간
 - 가독성을 높이기 위해 선택적으로 endblock 태그에 이름을 지정할 수 있음

5.2.2 추가 템플릿 경로 추가하기

- 기본 template 경로가 아닌 다른 경로를 추가하기 위해 코드 작성

```
# settings.py

TEMPLATES = [
{
    'DIRS' : [BASE_DIR / 'templates'],
}
]
```

- app_name/templates/ 디렉토리 경로 외 추가 경로를 설정한 것
- `BASE_DIR`

```
# settings.py
BASE_DIR = Path(__file__).resolve().parent.parent
```

 - settings.py에서 특정 경로를 절대 경로로 편하게 작성할 수 있도록 Django에서 미리 지정해둔 경로 값
 - '객체 지향 파일 시스템 경로'
 - 운영체제별로 파일 경로 표기법이 다르기 때문에 어떤 운영체제에서 실행되더라도 각 운영체제 표기법에 맞게 해석될 수 있도록 하기 위해 사용

6. Sending and Retrieving form data

- 데이터를 보내고 가져오기
- HTML form element를 통해 사용자와 애플리케이션 간의 상호작용 이해하기
- Client & Server architecture
 - 웹은 다음과 같이 가장 기본적으로 클라이언트-서버 아키텍처를 사용
 - 클라이언트(일반적으로 웹 브라우저)가 서버에 요청을 보내고, 서버는 클라이언트의 요청에 응답
 - 클라이언트 측에서 HTML form은 HTTP 요청을 서버에 보내는 가장 편리한 방법
 - 이를 통해 사용자는 HTTP 요청에서 전달할 정보를 제공할 수 있음

6.1 Sending form data (Client)

6.1.1 HTML <form> element

- 데이터가 전송되는 방법을 정의
- 웹에서 사용자 정보를 입력하는 여러 방식(text, button, submit 등)을 제공하고, 사용자로부터 할당된 데이터를 서버로 전송하는 역할을 담당
- 데이터를 어디(action)에서 어떤 방식(method)으로 보낼지
- 핵심 속성
 - **action**
 - 입력 데이터가 전송될 URL을 지정
 - 데이터를 어디로 보낼 것인지 지정하는 것이며 이 값은 반드시 유효한 URL이어야 함
 - 만약 이 속성을 지정하지 않으면 데이터는 현재 form이 있는 페이지의 URL로 보내짐
 - **method**
 - 데이터를 어떻게 보낼 것인지 정의
 - 입력 데이터의 HTTP request methods를 지정
 - HTML form 데이터는 오직 2가지 방법으로 전송 가능
 - **GET**
 - **POST**

```

# urls.py
urlpatterns = [
    path('throw/', views.throw)
]

# articles/views.py
def throw(request):
    return render(request, 'throw.html')

# articles/templates/throw.html
{% extend 'base.html' %}

{% block content %}
<h1>Throw</h1>
<form action="#" method="#">
</form>
{% endblock content %}

```

6.1.2 HTML <input> element

- 사용자로부터 데이터를 입력 받기 위해 사용
- 'type' 속성에 따라 동작 방식이 달라짐
 - type을 지정하지 않은 경우, 기본값은 "text"
- 핵심 속성
 - name
 - form을 통해 데이터를 제출(submit)했을 때 name 속성에 설정된 값을 서버로 전송
 - 서버는 name 속성에 설정된 값을 통해 사용자가 입력한 데이터 값에 접근할 수 있음
 - GET/POST 방식으로 서버에 전달하는 파라미터 (name은 key, value는 value)로 매팅하는 것
 - GET 방식에서는 URL에서 '?key=value&key=value/' 형식으로 데이터를 전달

```

# articles/templates/throw.html
{% extends 'base.html' %}

{% block content %}
<h1>Throw</h1>
<form action="#" method="#">
    <label for="message">Throw</label>
    <input type="text" id="message" name="message">
    <input type="submit">
</form>
{% endblock content %}

```

6.1.3 HTTP request methods

- HTTP
 - HTML 문서와 같은 리소스(데이터, 자원)들을 가져올 수 있도록 해주는 프로토콜 (규칙)
- 웹에서 이루어지는 모든 데이터 교환의 기초
- HTTP는 주어진 리소스가 수행 할 원하는 작업을 나타내는 request methods를 정의
- 자원에 대한 행위(수행하고자 하는 동작)을 정의
- 주어진 리소스(자원)에 수행하길 원하는 행동을 나타냄
- HTTP Method 예시
 - GET, POST, PUT, DELETE

6.1.4 GET

- 서버로부터 정보를 조회 하는 데 사용
 - 서버에게 리소스를 요청하기 위해 사용
- 데이터를 가져올 때만 사용해야 함
- 데이터를 서버로 전송할 때 Query String Parameters를 통해 전송
 - 데이터는 URL에 포함되어 서버로 보내짐
- GET과 get은 모두 동일하게 동작하지만 명시적 표현을 위해 대문자 사용 권장

```

# articles/templates/throw.html
{% extends 'base.html' %}

{% block content %}
<h1>Throw</h1>
<form action="#" method="GET">
    <label for="message">Throw</label>
    <input type="text" id="message" name="message">
    <input type="submit">
</form>
{% endblock content %}

```

6.1.4 Query String Parameters

- 사용자가 입력 데이터를 전달하는 방법 중 하나로, url 주소에 데이터를 파라미터를 통해 넘기는 것
- 이러한 문자열은 앤퍼센트(&)로 연결된 key=value 쌍으로 구성되며, 기본 URL과 물음표(?)로 구분됨
 - [http://host:port/path ?key=value&key=value](http://host:port/path?key=value&key=value)
- Query String 이라고 함
- 정해진 주소 이후에 물음표를 쓰는 것으로 Query String이 시작함을 알림
- key=value 로 필요한 파라미터의 값을 적음
 - = 로 key와 value가 구분됨
- 파라미터가 여러 개일 경우 & 을 붙여 여러 개의 파라미터를 넘길 수 있음

6.2 Retrieving the data (Server)

- 데이터 가져오기(검색하기)
- 서버는 클라이언트로 받은 key-value 쌍의 목록과 같은 데이터를 받게 됨

6.2.1 catch 작성

```
# urls.py  
  
urlpatterns = [  
    ...,  
    path('catch/', views.catch),  
]
```

```
# articles/views.py  
  
def catch(request):  
    pass  
    return render(request, 'catch.html')
```

```
<!-- articles/templates/catch.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
    <h1>Catch</h1>  
    <h2>여기서 데이터를 받았어!!</h2>  
    <a href="/throw/">다시 던지려</a>  
{% endblock %}
```

6.2.2 action 작성

- throw 페이지에서 form의 action 부분을 먼저 작성하고 데이터를 보냄

```
<!-- articles/templates/throw.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
    <h1>Throw</h1>  
    <form action="/catch/" method="GET">  
        <label for="message">Throw</label>  
        <input type="text" id="message" name="message">  
        <input type="submit">  
    </form>  
  
    <a href="/index/">뒤로</a>  
{% endblock %}
```

```
<!-- articles/templates/index.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
    ...  
    <a href="/throw/">throw</a>  
{% endblock %}
```

127

6.2.3 catch 작성 마무리

```
def catch(request):  
    message = request.GET.get('message')  
    context = {  
        'message': message,  
    }  
    return render(request, 'catch.html', context)
```

```
<!-- articles/templates/catch.html -->  
  
{% extends 'base.html' %}  
  
{% block content %}  
    <h1>Catch</h1>  
    <h2>여기서 {{ message }} 를 받았어!!</h2>  
    <a href="/throw/">다시 던지려</a>  
{% endblock %}
```

6.2.4 Request and Response objects

- 요청과 응답 객체 흐름
 - 페이지가 요청되면 Django는 요청에 대한 메타데이터를 포함하는 HttpRequest object를 생성
 - 해당하는 적절한 view 함수를 로드하고 HttpRequest를 첫번째 인자로 전달
 - 마지막으로 view 함수는 HttpResponse object를 반환

7. Django URLs

- 웹 어플리케이션은 URL을 통한 클라이언트의 요청에서부터 시작함

7.1 Trailing URL Slashes

- Django는 URL 끝에 /(Trailing slash)가 없다면 자동으로 붙여주는 것이 기본 설정
 - 그래서 모든 주소가 / 로 끝나도록 구성 되어있음

[참고] URL 정규화

- 정규 URL(=オリジ널로 평가되어야 할 URL)을 명시하는 것
- 복수의 페이지에서 같은 콘텐츠가 존재하는 것을 방지하기 위함
- Django에서는 /가 없는 요청에 대해 자동으로 slash를 추가하여 통합된 하나의 콘텐츠로 봄

7.2 Variable routing

- Variable routing의 필요성
 - 템플릿의 많은 부분이 중복되고, 일부분만 변경되는 상황에서 비슷한 URL과 템플릿을 계속 만들 필요가 없음
- URL 주소를 변수로 사용하는 것을 의미
- URL의 일부를 변수로 지정하여 view 함수의 인자로 넘길 수 있음
- 변수 값에 따라 하나의 path()에 여러 페이지를 연결 시킬 수 있음

7.2.1 Variable routing 작성

- 변수는 < >에 정의하여 view 함수의 인자로 할당 됨
- 기본 타입은 string이며 5가지 타입으로 명시할 수 있음
 - str
 - / 를 제외하고 비어 있지 않은 모든 문자열
 - 작성하지 않을 경우 기본 값
 - int
 - 0 또는 양의 정수와 매치

```
# urls.py

urlpatterns = [
    ...,
    # path('hello/<str:name>/', views.hello),
    path('hello/<name>/', views.hello),
]
```

7.2.2 View 함수 작성

- variable routing으로 할당된 변수를 인자로 받고 템플릿 변수로 사용할 수 있음

```
# articles/views.py
def hello(request, name):
    context = {
        'name': name,
    }
    return render(request, 'hello.html', context)


{% extends 'base.html' %}

{% block content %}
<h1>만나서 반가워요 {{ name }}님!</h1>
{% endblock %}
```

[참고] Routing(라우팅)

- 어떤 네트워크 안에서 통신 데이터를 보낼 때 최적의 경로를 선택하는 과정을 뜻함

7.3 App URL mapping

- 앱이 많아졌을 때 urls.py를 각 app에 매팅하는 방법
- app의 view 함수가 많아지면서 사용하는 path() 또한 많아지고, app 또한 더 많이 작성되기 때문에 프로젝트의 urls.py에서 모두 관리하는 것을 프로젝트 유지보수에 좋지 않음
- 각 앱의 view 함수를 다른 이름으로 import할 수 있음
- 하나의 프로젝트의 여러 앱이 존재한다면, 각각의 앱 안에 urls.py를 만들고 프로젝트 urls.py에서 각 앱의 urls.py 파일로 URL 매팅을 위탁할 수 있음
- 각각의 app 폴더 안에 urls.py를 작성하고 수정

```
# articles/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index),
    path('greeting/', views.greeting),
    path('dinner/', views.dinner),
    path('throw/', views.throw),
    path('catch/', views.catch),
    path('hello/<str:name>/', views.hello),
]
```

```
# pages/urls.py
from django.urls import path

urlpatterns = [
]
```

7.3.1 Including other URLconfs

- urlpattern은 언제든지 다른 URLconf 모듈을 포함(include)할 수 있음
- include되는 앱의 urls.py에 urlpatterns가 작성되어 있지 않다면 에러가 발생
- urlpatterns가 빈 리스트라도 작성되어 있어야 함

```
# firstpjt/urls.py

from django.contrib import admin
from django.urls import path, include

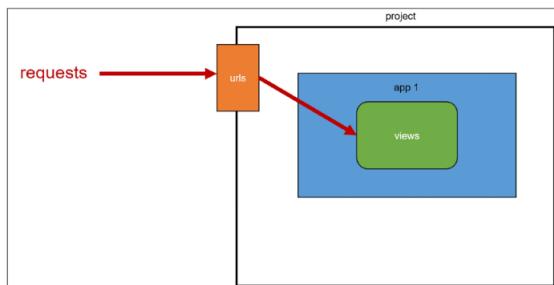
urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', include('articles.urls')),
    path('pages/', include('pages.urls')),
]
```

- `include()`

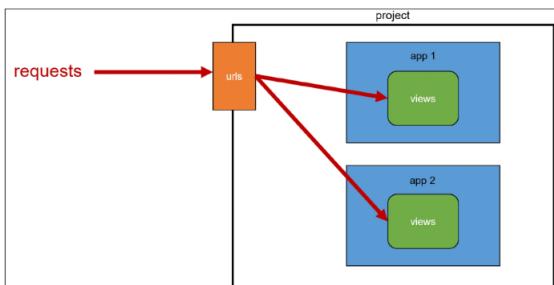
- 다른 URLconf(app1/urls.py)들을 참조할 수 있도록 돋는 함수
- 함수 `include()`를 만나게 되면 URL의 그 시점까지 일치하는 부분을 잘라내고, 남은 문자열 부분을 후속 처리를 위해 `include`된 URLconf로 전달

7.3.2 URL 구조의 변화

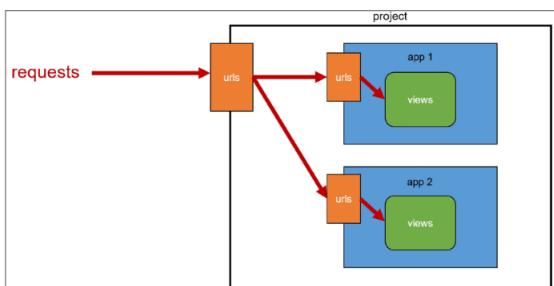
- 앱의 URL을 project의 urls.py에서 관리



- 복수 개의 앱의 URL을 project의 urls.py에서 관리



- 각각의 앱에서 URL을 관리



7.4 Naming URL patterns

- 링크에 URL을 직접 작성하는 것이 아니라 `path()`의 `name` 인자를 정의해서 사용
- DTL의 tag 중 하나인 URL 태그를 사용해서 `path()`에 작성한 name을 사용 가능
- 이를 통해 URL 설정에 정의된 특정한 경로들의 의존성을 제거할 수 있음
- Django에서는 URL에 이름을 지정하는 방법을 제공함으로써, view함수와 템플릿에서 특정 주소를 쉽게 참조할 수 있도록 도움

```
# articles/urls.py

urlpatterns = [
    path('index/', views.index, name='index'),
    path('greeting/', views.greeting, name='greeting'),
    path('dinner/', views.dinner, name='dinner'),
    path('throw/', views.throw, name='throw'),
    path('catch/', views.catch, name='catch'),
    path('hello/<str:name>', views.hello, name='hello'),
]
```

- `{% url '' %}`
 - 주어진 URL 패턴 이름 및 선택적 매개 변수와 일치하는 절대 경로 주소를 반환
 - 템플릿에 URL을 하드 코딩하지 않고도 DRY 원칙을 위반하지 않으면서 링크를 출력하는 방법

```
<!-- catch.html -->
{% extends 'base.html' %}
{% block content %}
<h1>Catch</h1>
<h2>여기서 {{ message }}를 받았어!!</h2>
<a href="{% url 'throw' %}">다시 던지려</a>
{% endblock content %}

<!-- throw.html -->
{% extends 'base.html' %}
{% block content %}
<h1>Throw</h1>
<form action="{% url 'catch' %}" method="GET">
...
</form>
<a href="{% url 'index' %}">뒤로</a>
{% endblock content %}

<!-- index.html -->
{% extends 'base.html' %}
{% block content %}
...
<a href="{% url 'greeting' %}">greeting</a>
<a href="{% url 'dinner' %}">dinner</a>
<a href="{% url 'throw' %}">throw</a>
{% endblock content %}

<!-- dinner, greeting.html-->
<a href="{% url 'index' %}">뒤로</a>
```

161

■ [참고] DRY 원칙

- `Don't Repeat Yourself`의 약어
- 소스 코드에서 동일한 코드를 반복하지 말자 라는 의미
- 동일한 코드가 반복된다는 것은 잠재적인 버그의 위협을 증가시키고 반복되는 코드를 변경해야 하는 경우, 반복되는 모든 코드를 찾아서 수정

8. 마무리

8.1 Django의 설계 철학(Templates System)

- 표현과 로직(view)을 분리
 - 템플릿 시스템은 표현을 제어하는 도구이자 표현에 관련된 로직일 뿐
 - 템플릿 시스템은 이러한 기본 목표를 넘어서는 기능을 지원하지 말아야 함
- 중복을 배제
 - 대다수의 동적 웹사이트는 공통 header, footer, navbar 같은 사이트 공통 디자인을 가짐
 - Django 템플릿 시스템은 이러한 요소를 한곳에 저장하기 쉽게 해 중복 코드를 없애야 함
 - 템플릿 상속의 기초가 되는 철학

8.2 Framework의 성격

- 독선적(Opinionated)
 - 독선적인 프레임워크들은 어떤 특정 작업을 다루는 올바른 방법에 대한 분명한 의견 존재
 - 대체로 특정 문제내에서 빠른 개발방법을 제시
 - 어떤 작업에 대한 올바른 방법이란 보통 잘 알려져 있고 문서화가 잘 되어있기 때문
- 관용적(Unopinionated)
 - 관용적인 프레임워크들은 구성요소를 한데 붙여서 해결해야 한다거나 심지어 어떤 도구를 써야 한다는 올바른 방법에 대한 제약이 거의 없음
 - 이는 개발자들이 특정 작업을 완수하는데 가장 적절한 도구들을 이용할 수 있는 자유도가 높음
 - 하지만 개발자 스스로가 그 도구들을 찾아야 한다는 수고가 필요

8.3 Django Framework의 성격

- 다소 독선적
 - 양쪽 모두에게 최선의 결과를 준다고 강조
- 결국 하고자 하는 말은 현대 개발에 있어서는 가장 중요한 것들 중 하나는 생산성
- 프레임워크는 우리가 하는 개발을 방해하기 위해 규칙, 제약을 만들어 놓은 것이 아님
- 우리가 온전히 만들고자 하는 것에만 집중할 수 있게 도와주는 것