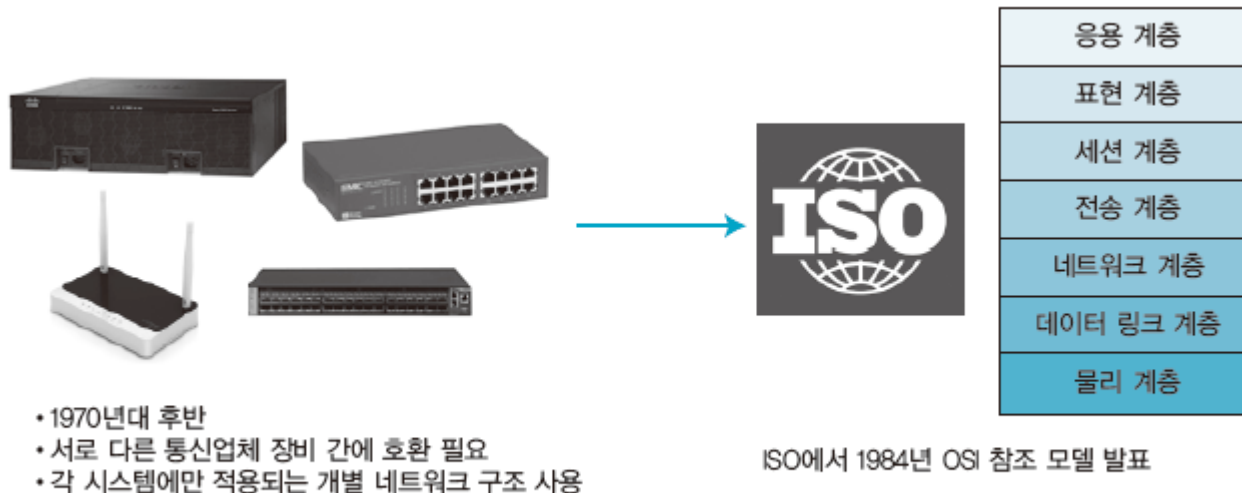


웹 서버 프로그래밍 #1



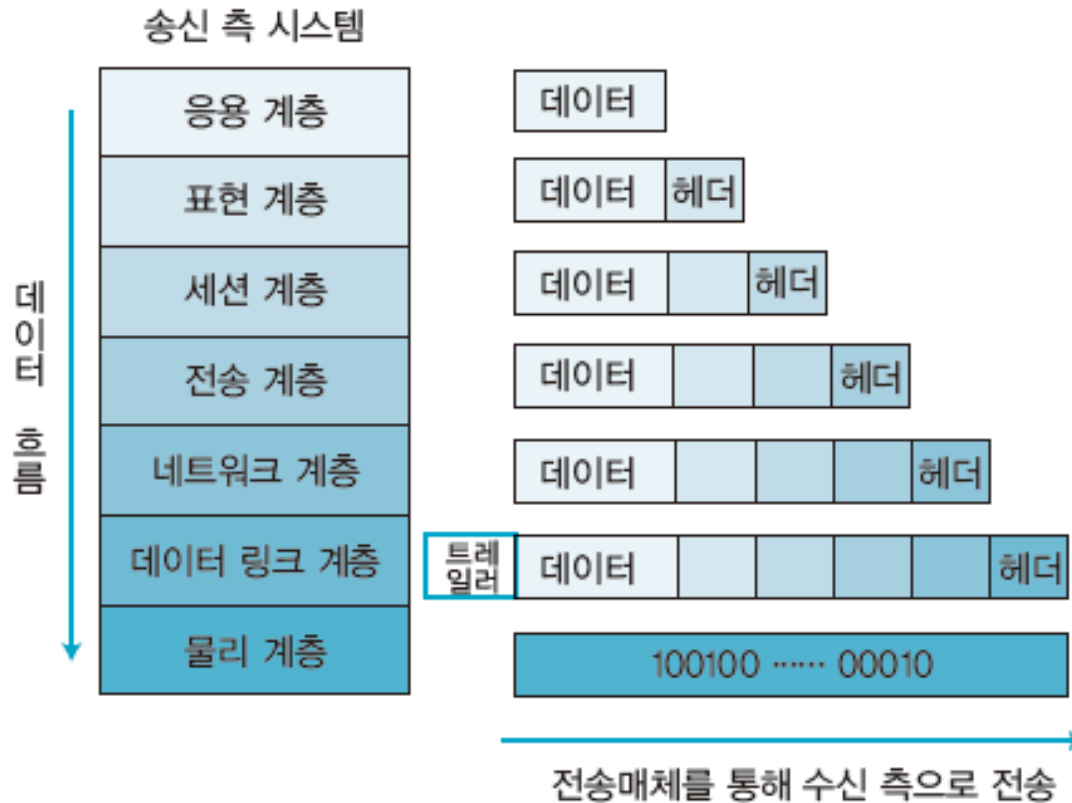
01. OSI 참조 모델의 개요

- 통신 기술의 도입과 통신 기능의 확장을 쉽게 하려고 프로토콜을 몇 개의 계층으로 나누는 것을 '계층화'라 하고, 통신 기능을 7계층으로 분류하여 각 계층마다 프로토콜을 규정한 규격을 'OSI(Open System Interconnection)' 모델이라고 한다.



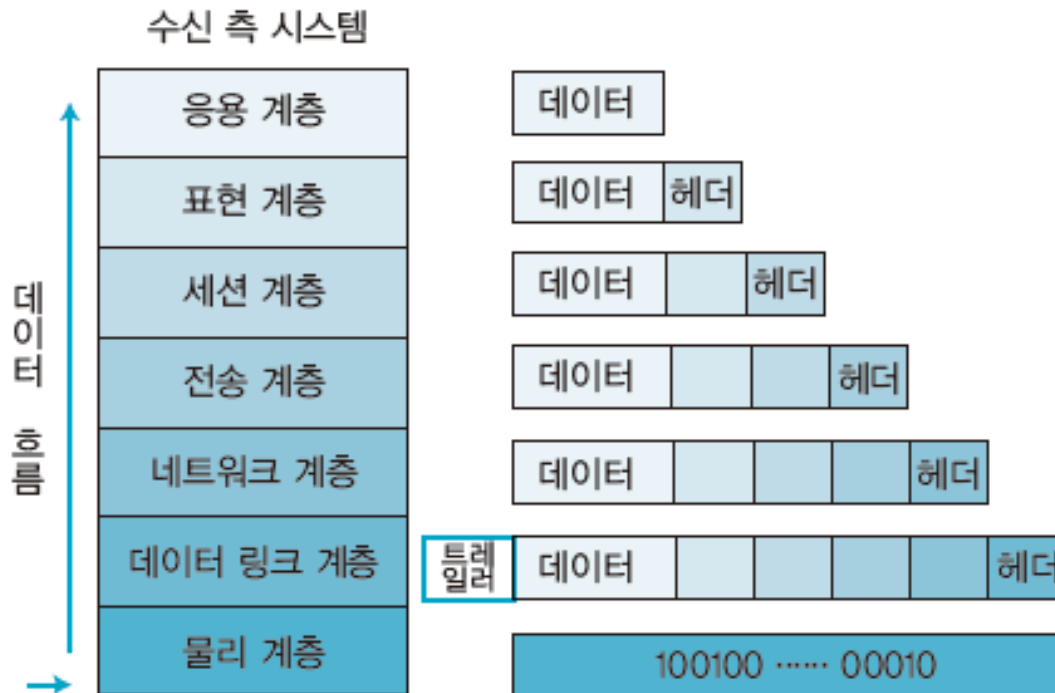
[그림 4-1] OSI 참조 모델

02. OSI 참조 모델의 데이터 전송



[그림 4-4] OSI 참조 모델에서 데이터 전송(송신 측)

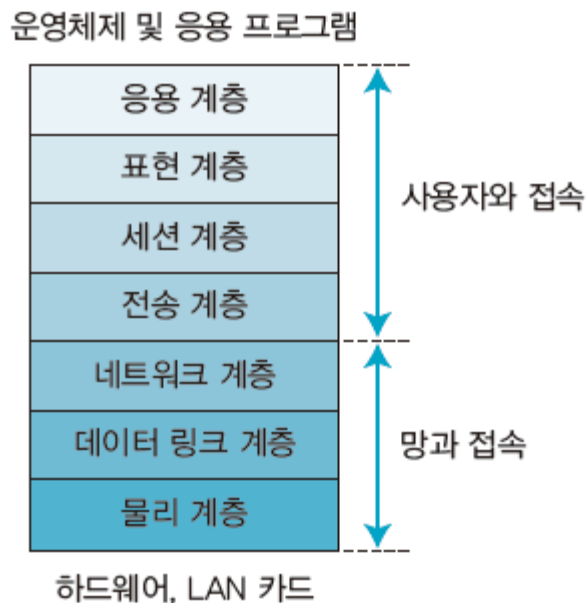
02. OSI 참조 모델의 데이터 전송



[그림 4-5] OSI 참조 모델에서 데이터 전송(수신 측)

03. OSI 참조 모델 7계층

- 계층 7개는 서로 독립적이므로 어느 한 계층의 변경이 다른 계층에는 영향을 미치지 않는다.
- 기능에 필요한 몇 개의 계층만 표준화하면 정상적으로 통신할 수 있다.

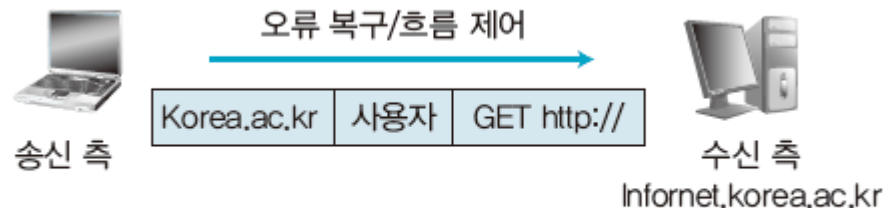


[그림 4-7] OSI 참조 모델 7계층

03. OSI 참조 모델 7계층: 전송 계층

■ 전송 계층(Transport Layer)

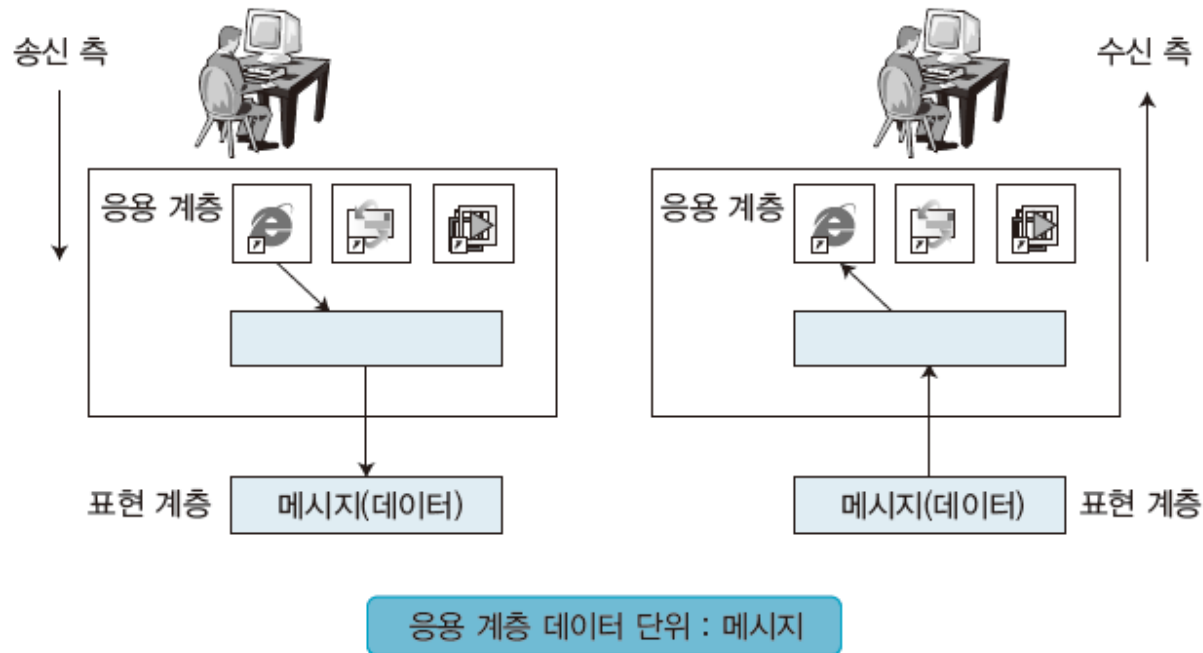
- 프로토콜(TCP, UDP)과 관련된 계층으로 오류 복구와 흐름 제어 등을 담당하며, 두 시스템 간에 신뢰성 있는 데이터를 전송한다.
- 또한 네트워크 계층에서 온 데이터를 세션 계층의 어느 애플리케이션에 보낼 것인지 판독하고, 네트워크 계층으로 전송할 경로를 선택한다.
- OSI 참조 모델 7계층 중 전송 계층은 네 번째 계층으로 시스템 종단 간에 투명한 데이터를 양방향으로 전송하는 계층이다.
- 네트워크 계층에서 전송한 데이터와 실제 운영체제의 프로그램이 연결되는 통신 경로라고 할 수 있다



[그림 4-15] 전송 계층의 예

03. OSI 참조 모델 7계층 : 응용 계층

- 응용 계층과 사용자(사람 또는 소프트웨어), 표현 계층 간의 관계를 보여준다.



[그림 4-23] 응용 계층의 데이터 단위 전송

04. 인터넷 모델 : 전송 계층

■ 전송 계층

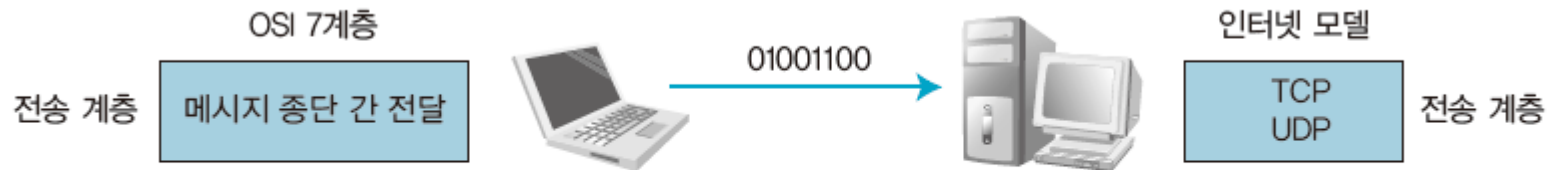
- 인터넷 모델의 전송 계층에는 TCP와 UDP 프로토콜이 두 개 있다.
 - TCP(Transmission Control Protocol : 전송 제어 프로토콜)는 송신지에서 수신지까지 문자 스트림을 전송하는데, 두 응용 계층이 서로 대화하는 것을 허용하는 신뢰성 있는 프로토콜이다.
 - TCP의 성능은 OSI 참조 모델의 전송 계층보다 뛰어나다.
 - UDP(User Datagram Protocol : 사용자 데이터그램 프로토콜)는 OSI 참조 모델에서 정의하는 전송 계층의 일부 역할을 무시하는 단순한 전송 프로토콜이다.
 - UDP는 TCP에 비해 신뢰성이 낮으며, 흐름 제어 및 오류 검출 등의 기능이 없어 패킷을 빠르게 전송해야 하는 응용 계층에서 사용한다



[그림 4-27] 인터넷 모델의 전송 계층

04. 인터넷 모델 : 전송 계층

- TCP를 사용한 전송 계층의 예를 살펴보자. 송신 측에서 데이터 (01001100)를 보내면, TCP의 포트 번호 80번을 이용하여 수신 측으로 데이터를 안전하게 전송한다.
 - 전송 계층에서는 송신지에서 수신지까지 메시지 전송 기능을 제공한다.



[그림 4-28] 인터넷 모델과 OSI 참조 모델의 비교(전송 계층)

04. 인터넷 모델 : 응용계층

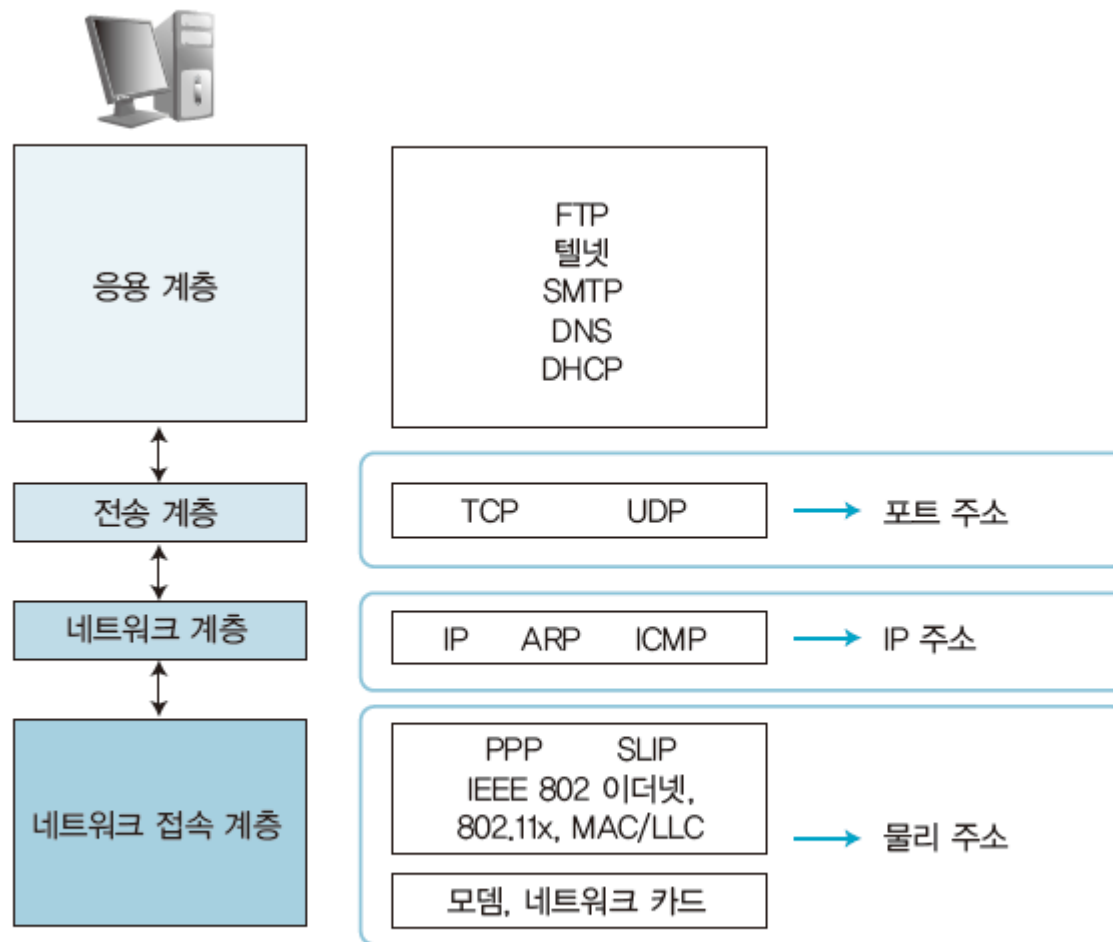
■ 응용 계층

- 인터넷 모델의 응용 계층에 포함되어 있는 프로토콜 일곱 개와 프로그램은 원격으로 컴퓨터 자원에 접속하는 데 사용한다.
- 응용 프로그램들로 제공되는 서비스는 표현 계층과 세션 계층에서 정의하고 있다



[그림 4-25] 인터넷 모델의 응용 계층

01. TCP/IP의 이해



[그림 5-7] 계층과 주소 관계

01. TCP/IP의 이해

■ TCP/IP 주소의 구조

■ 물리 주소

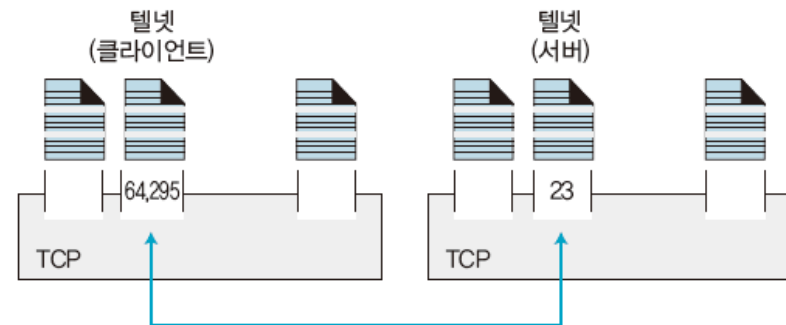
- 물리 주소(MAC 주소)는 링크 주소 또는 통신망에서 정의된 노드의 주소, 이더넷 네트워크인터페이스 카드(NIC) 6바이트(48비트) 주소 등을 말한다.

■ 인터넷 주소

- 인터넷에서는 기존 물리 주소와는 별도로 각 호스트를 식별할 수 있는 유일한 주소를 지정해야 한다.

■ 포트 주소

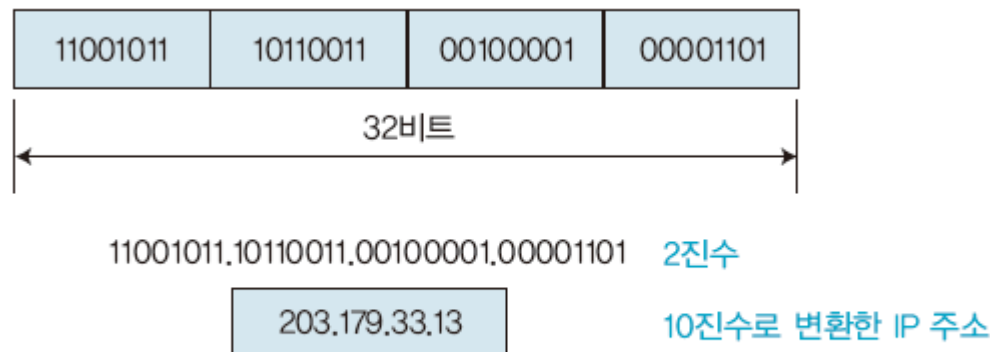
- 수신지 컴퓨터까지 전송하려면 IP 주소와 물리 주소가 필요하다.
- 인터넷 통신의 최종 목적은 한 프로세스가 다른 프로세스와 통신할 수 있도록 하는 것이다.



[그림 5-6] 포트 주소

02. IP(인터넷 프로토콜)

- 인터넷에 연결된 모든 컴퓨터에는 고유의 주소가 부여되는데, 이를 'IP 주소'라고 한다.
 - 현재 사용하는 IP 주소 체계는 IP Ver. 4이다.
 - IP 주소는 8비트 크기의 필드 네 개를 모아서 구성한 32비트(4바이트) 논리 주소다.
 - $x.x.x.x$, 즉 163.152.19.114처럼 .(점)으로 구분한 10진수 형태 네 개로 구성된다.
 - 한 바이트가 가질 수 있는 10진수는 0~255이므로, IP 주소의 값은 0.0.0.0에서 255.255.255.255까지다.

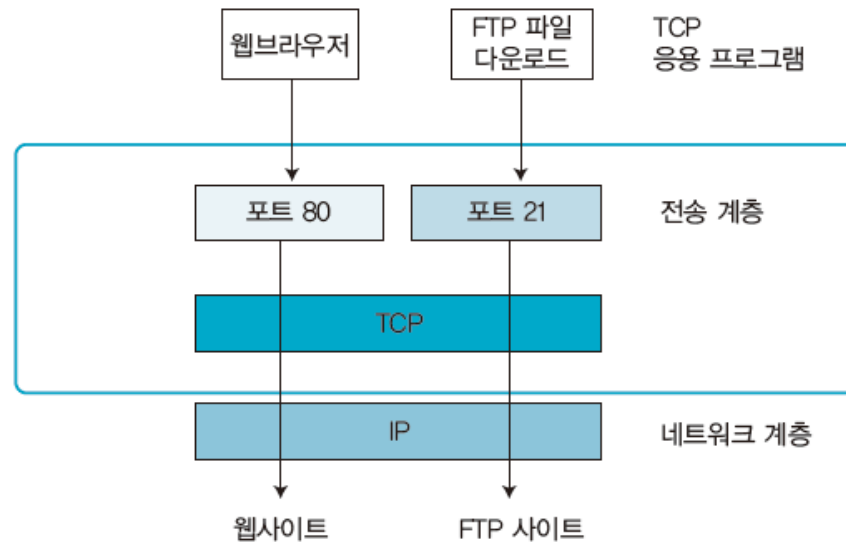


[그림 5-8] IP 주소의 구조

03. TCP(전송 제어 프로토콜)

■ 포트 번호

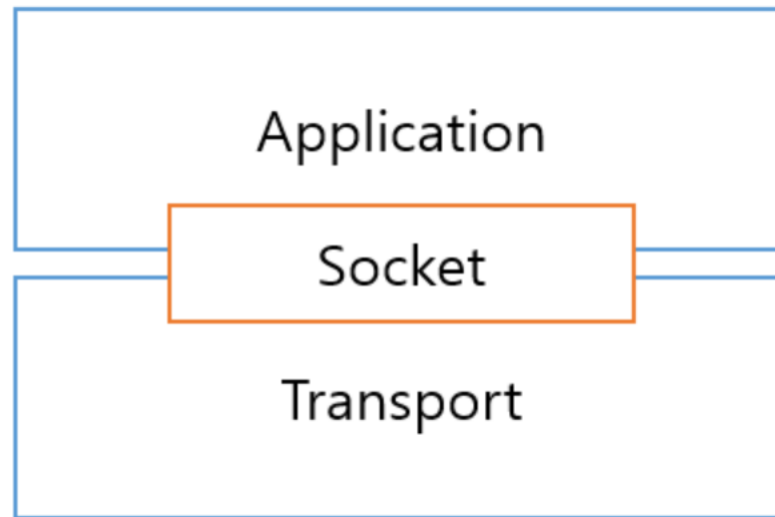
- 포트는 TCP가 상위 계층으로 데이터를 전달하거나 상위 계층에서 TCP로 데이터를 전달할 때 상호 간에 사용하는 데이터의 이동 통로를 말한다.
- 통신할 때 여러 웹사이트에서 파일을 동시에 다운로드할 수 있다.
- 파일을 동시에 다운로드할 수 있는 이유는 TCP 프로토콜이 포트를 여러 개 사용해서 상위 계층의 프로그램과 각각 따로 통신하기 때문이다.
 - TCP 포트 범위는 0~65,534까지의 정수다.



[그림 5-28] TCP 포트를 이용한 통신

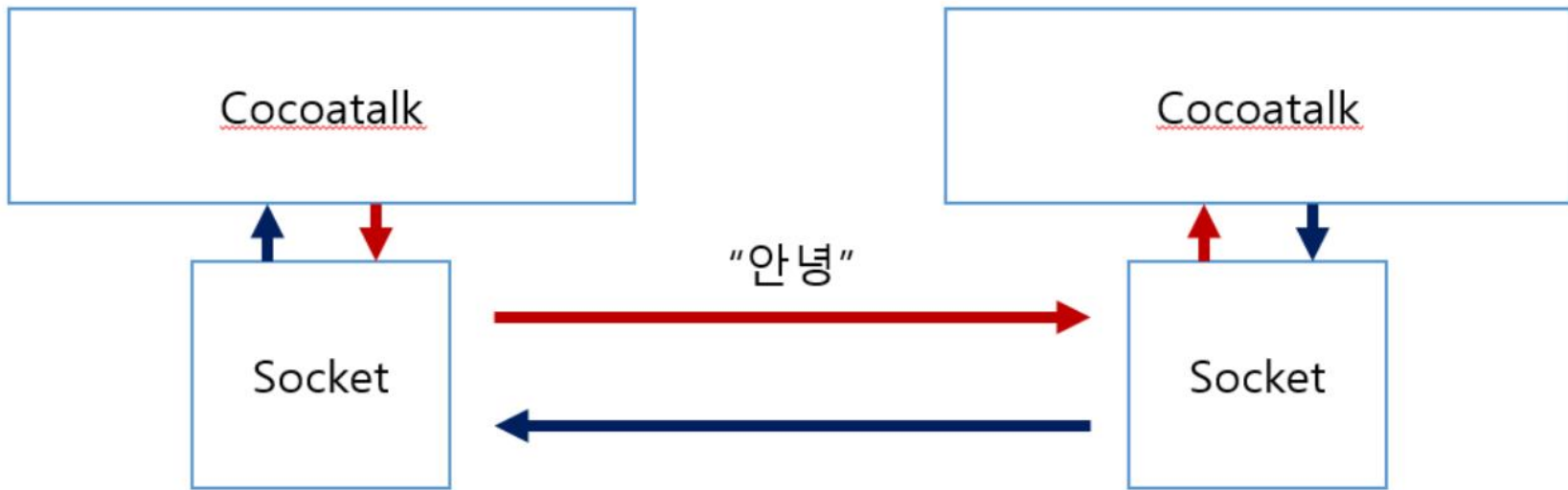


애플리케이션 계층에서 트랜스포트 계층을 조작할 수 있을까요.
소켓은 OS에서 제공하는 인터페이스라고 설
이 소켓으로만 외부 네트워크와 통신을 할 수 있



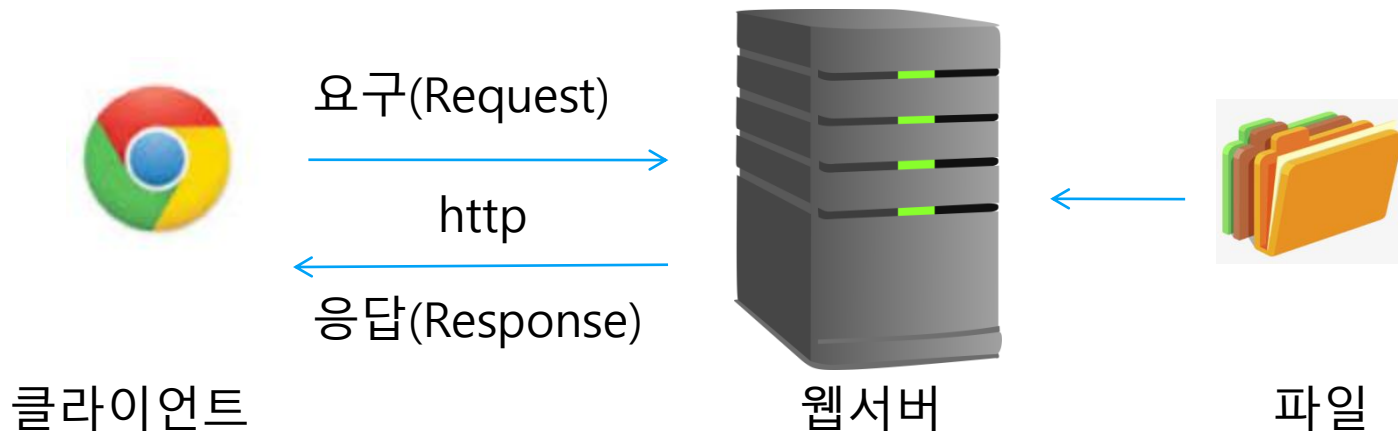


실제로 통신을 하고 있는 것은 소켓들이고,
코코아톡 자체는 외부 네트워크와 아무런
정보도 주고 받지 않습니다



웹 서비스

- 웹 서버는 웹 클라이언트로부터 전송되어 오는 서비스 요청에 대한 결과인 HTML문서를 해당 웹 클라이언트에 제공
- 웹 브라우저 : HTML로 작성된 하이퍼텍스트 문서를 해석하여 화면에 출력
- 가장 대표적인 웹서버로 아파치 사용
 - 안정성, 확장성을 고려한 OpenSource 프로젝트





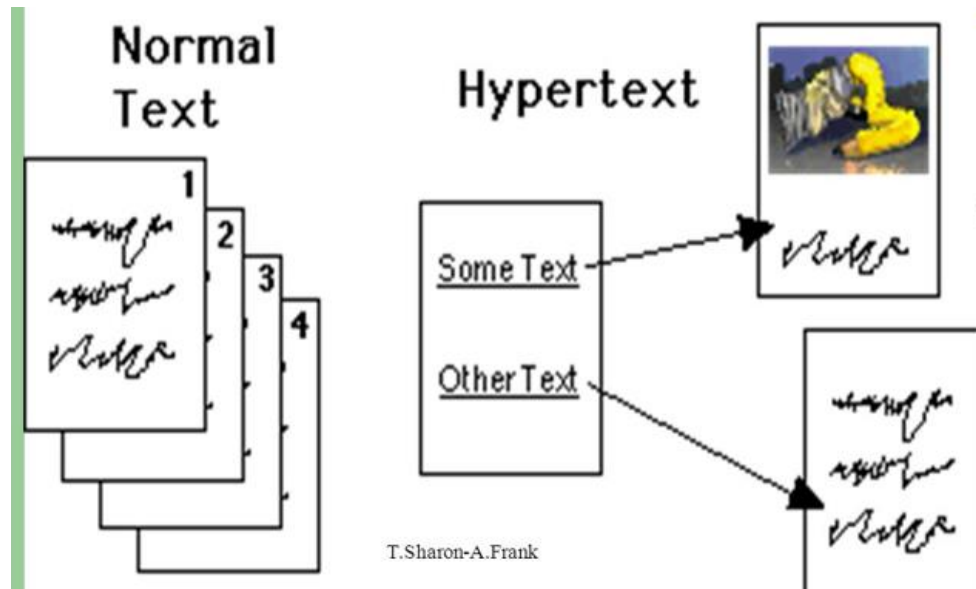
- **HyperText Transfer Protocol**
- 웹에서 정보를 주고 받을 수 있는 가장 기본적인 프로토콜(1996)
- HTML 포맷을 사용한 text 및 멀티미디어 자료 공유
- TCP(80번 port) 프로토콜을 사용한 응용 계층
- http:로 시작하는 URL을 통해 접속
- 대규모 접속을 지원하기 위한 접속을 유지 하지 않는 stateless 프로토콜
- 클라이언트와 서버 사이에 이루어지는 요청/응답 프로토콜



- 웹서버에 요청하는 방식
 - 웹서버에 클라이언트의 data를 전송하기 위한 전송 방법
 - GET과 POST는 전달하는 방식에서 차이가 있음
- GET 방식 : URL을 통해 data 전송
 - GET방식으로 전송할수 있는 data크기는 한계가 있음
 - 전송 URL이 노출됨
 - <http://xxx/service?name1=value1&name2=value2>
 - GET /service?name1=value1&name2=value2
 - Host : xxx
- POST 방식 : Body를 통해 data 전송
 - 길이 제한이 없음
 - POST /service HTTP/1.1
 - Host : xxx
 - name1=value1&name2=value2

HTML 이란

- 많은 수의 문서를 하이퍼 링크로 서로 연결시킬 수 있는 문서의 구조
- 참조([하이퍼링크](#))를 통해 한 문서에서 다른 문서로 즉시 접근할 수 있는 텍스트
- 하이퍼텍스트는 1960년대 컴퓨터 개척자이자 철학자인 [테드 넬슨](#)이 처음 고안

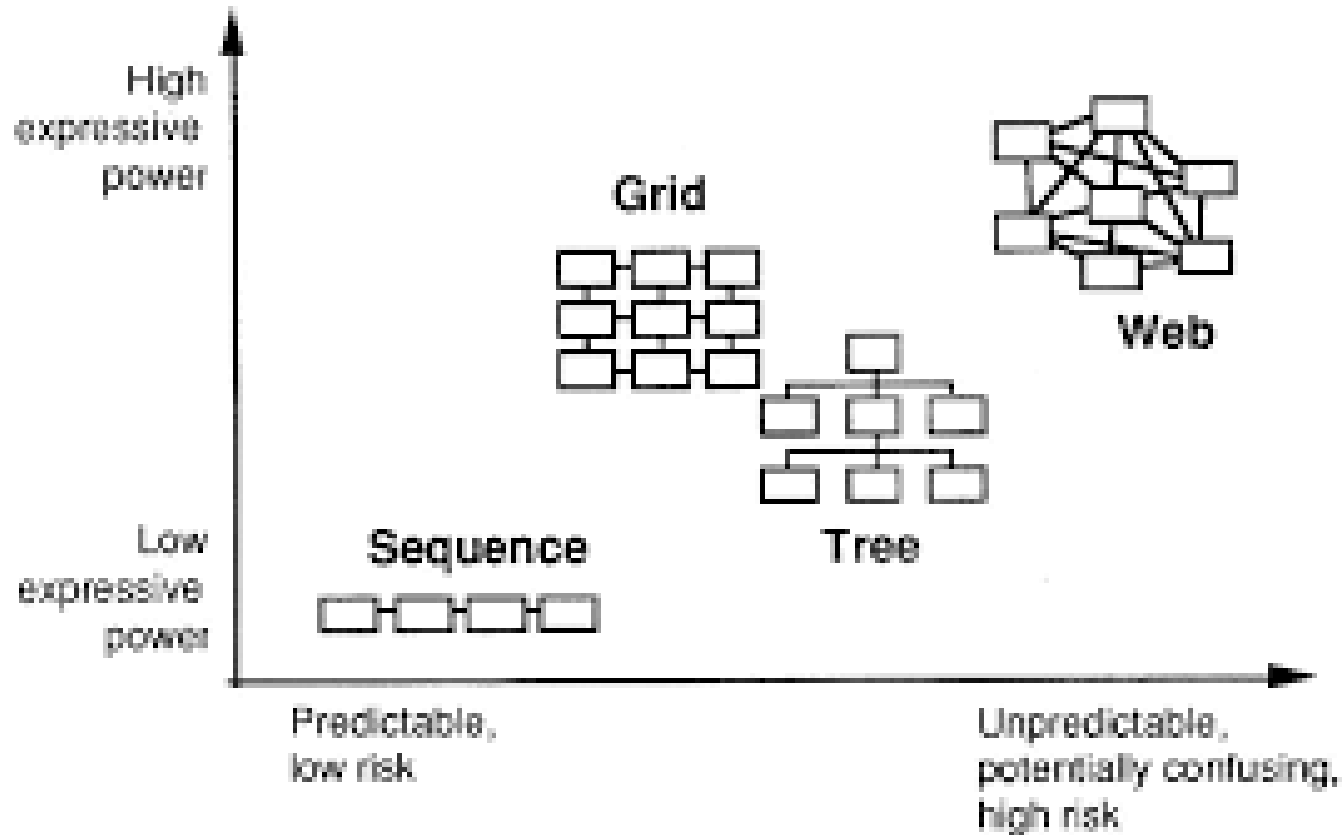


hyper(건너 편의, 초월, 과도한)

` label `

문서의 구조

- 기존 문서 : 순차적, 서열형 구조
- 하이퍼텍스트 : 링크에 따라 그 차례가 바뀌는 임의적이면서 나열형 구조
- 출판된 책처럼 작가의 의도대로 사용자가 따라가는 것이 아닌, 하이퍼링크로 연결된 문서들을 어떠한 행위(클릭)에 따라 자유롭게 이동
- 검색엔진과 더불어 정보습득의 새로운 장을 인류에게 가져다 줌



- tag 는 하나의 키워드
- opening 과 closing tags 이루어짐
- XHTML에서는 가 아닌 사용
- 중첩태그
 - <H1> <I> The Nation </I> </H1>
- 속성을 값을 설정하기 위해 "와 ' 모두 사용
- 주석
 - <!-- -->



1. 명확성

- HTML Tag로만 구성된 소스는 명확하다. 보기 편하다.
- 명확한 코드는 코드 자체를 좀더 엄격하게 제어할 수 있게 해준다.
- 논리적으로 명확한 구조와 코드의 엄격한 제어는 스크린리더 사용과 같은 접근성에도 기여한다.
- 코드가 적어지면 파일 용량도 줄고 속도측면도 잇점이 있다.

2. 확장성

- 다양한 접속환경(운영체제, 브라우저, 미디어)에 대응할 수 있다.
: 구조와 관계없이 표현 부분만 별도의 CSS로 분리하여 각 브라우저별, 미디어별로 사용 가능하다.
: [1개의 HTML] : [IE6용 CSS, 프린터용 CSS, 스마트폰용 CSS]
(브라우저 대응은 '**CSS 핵**'에서, 미디어 대응은 '**미디어쿼리**'에서 자세히 보게 됩니다.)
- 스타일 전환(style switchers)가 가능하다. 커스터마이징이 편해지고 스킨화가 가능하다.

3. 유지보수 편의성

- 명확하고 분리되어 있기 때문에 수정(유지보수)가 편하다.
: 개발자는 HTML을 보고, 퍼블리셔(디자이너)는 CSS를 보고.. 요런식으로도 생각할 수 있다.
- 공통적인 표현은 CSS에 통합되어 있긴 때문에 한방에 수정 가능하다.
- 결국 원가절감에 기여할 수 있다.

모바일 viewport 지원

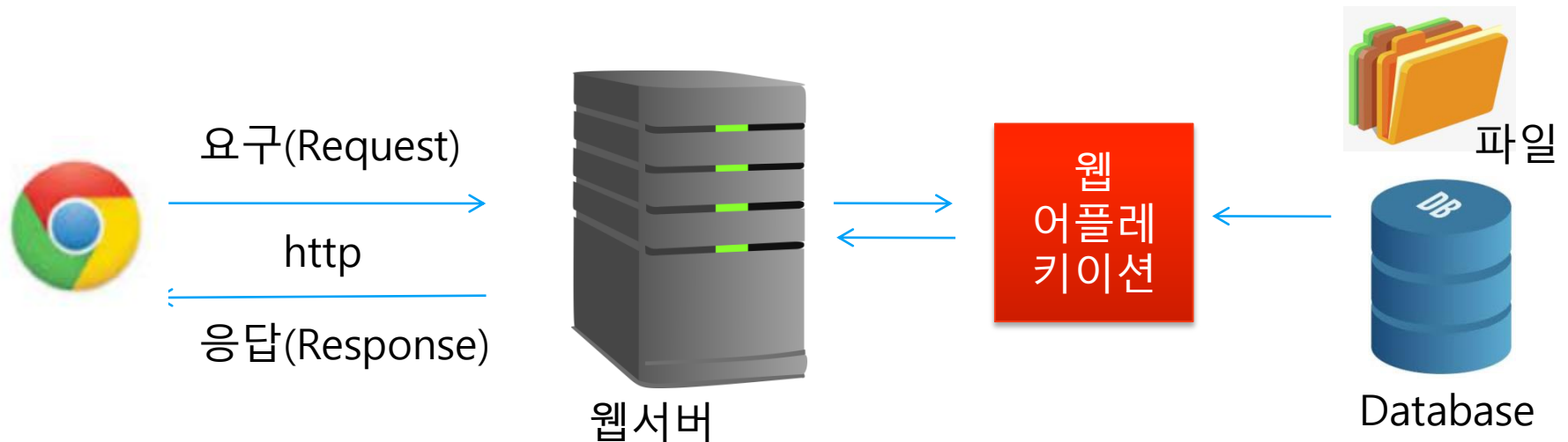
```
<head>  
  <meta charset="utf-8">  
  <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
</head>
```

모바일 Camera 지원

```
<input type = "file" name = "file1"  
accept="image/*" capture="camera" />
```


웹 서버 어플리케이션

- 웹을 기반으로 서버실행되는 프로그램
- 동적 HTML 생성 : 웹 서버 어플리케이션은 HTML을 리턴
- 웹 서버 어플리케이션은 파일, 데이터베이스 등 기존 서버 자원 활동



- Python에서는 경량 web server를 위해 세가지 방법 제공
 - 기본 웹서버(SimpleHTTPServer)

- `python -m http.server 9000`

- 스크립트 기반 웹 서버

```
import SimpleHTTPServer
import SocketServer
```

```
Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", 9000), Handler)
httpd.serve_forever()
```

- Web Framework기반 웹서버

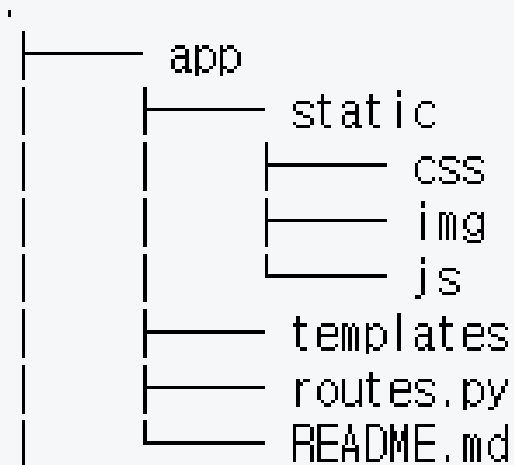
- Flask : 경량 웹 서버
 - Django : MVC 기반의 웹 서버

Python 웹 서버 프로그래밍(Flask)

- 파이썬으로 작성된 마이크로 웹 프레임워크
- 최소한의 기능만 제공하여 유연한 어플리케이션 제작 가능
- 설치 : `pip install Flask`



- 폴더 기본 구조
 - 루트 폴더 : app
 - 정적파일 폴더 : app/static
 - 하위에 css, img, js 폴더 생성
 - 템플릿 폴더 : app/templates
- python 코드는 루트 폴더에 작성



- 실행 : `python home.py`

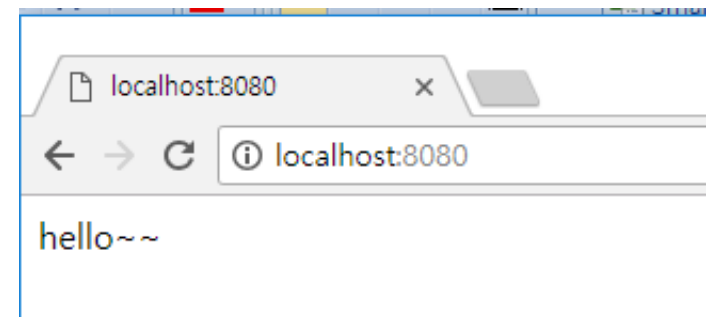
```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')  
def home():  
    return "hello~~"
```

```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=8080)
```

```
(base) D:\wckt\2018\WT\IoT\flask>python home.py  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 383-366-169  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)  
127.0.0.1 - - [14/Jun/2018 14:21:24] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [14/Jun/2018 14:21:24] "GET /favicon.ico HTTP/1.1" 404 -  
* Detected change in 'D:\wckt\2018\WT\IoT\flask\home.py', reloading  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 383-366-169  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```



- 코드 수정 시 서버 자동 reload

```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=8080, debug=True)
```

Anaconda Prompt - python home.py

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)  
127.0.0.1 - - [14/Jun/2018 16:56:27] "POST /form HTTP/1.1" 200 -  
* Detected change in 'D:\w\ck\2018\IoT\flask\home.py', reloading  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 383-366-169  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)  
127.0.0.1 - - [14/Jun/2018 19:11:36] "POST /form HTTP/1.1" 200 -  
* Detected change in 'D:\w\ck\2018\IoT\flask\home.py', reloading  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 383-366-169  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)  
* Detected change in 'D:\w\ck\2018\IoT\flask\home.py', reloading  
* Restarting with stat  
* Debugger is active!
```

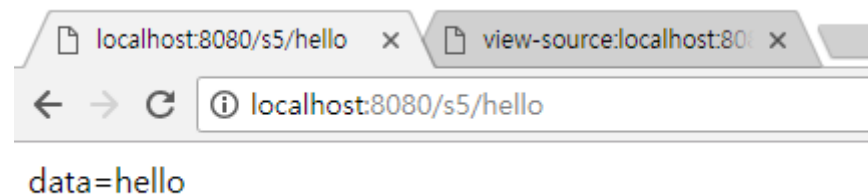
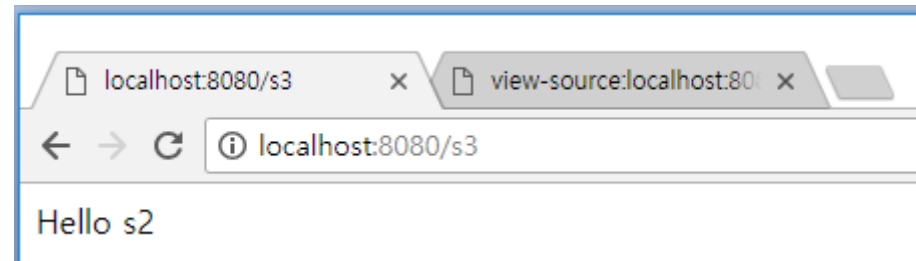
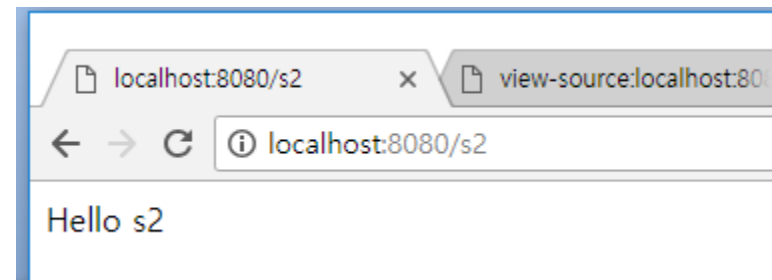
- 하나의 파일에서 다수의 URL 처리

```
@app.route('/')  
def home():  
    return render_template('home.html')
```

```
@app.route('/s1')  
def s1():  
    return "Hello s1 "
```

```
@app.route('/s2')  
@app.route('/s3')  
def s2():  
    return "Hello s2 "
```

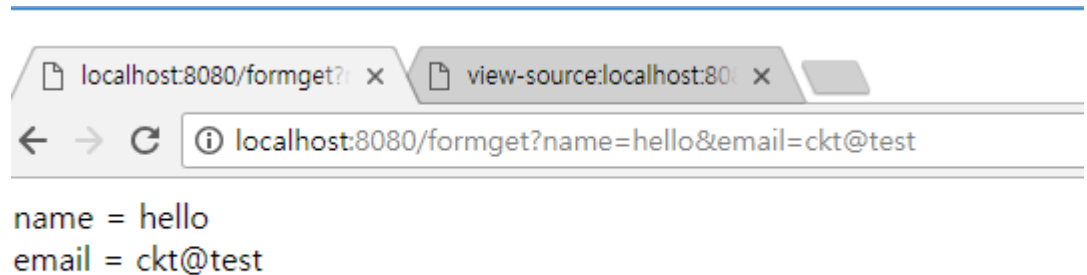
```
@app.route('/s5/<name>')  
def s6(name):  
    return "data=" + name
```



- Get 파라미터는 `request.args.get()` 함수로 값 접근

```
@app.route('/formget', methods=['GET'])
def formget():

    html = "name = " + request.args.get("name") + "<br>"
    html += "email = " + request.args.get("email") + "<br>"
    return html
```



Form GET

← → ↻ ⓘ localhost:8000/form.html

앱 128 bit 출력 128 add sub Catching Integer O... C: multiplin...

name :

email :

```
<form action=formget>
```

```
name : <input type=text name=name> <br>
```

```
email : <input type=text name=email> <br>
```

```
<input type=submit value=send>
```

```
</form>
```

Template 기반 Flask

- Html 코드와 python 코드 분리
- Html 파일은 templates 폴더에 위치

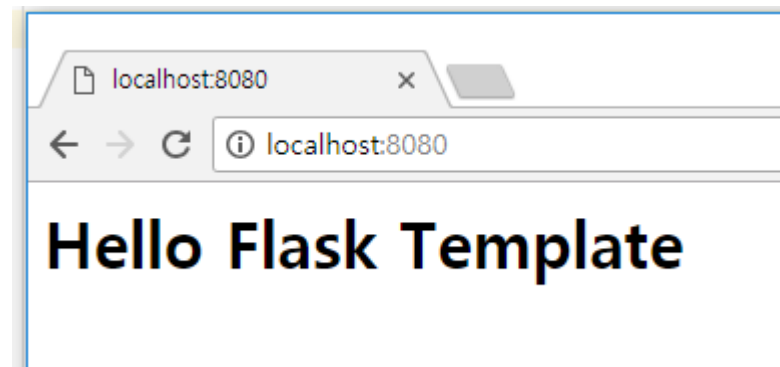
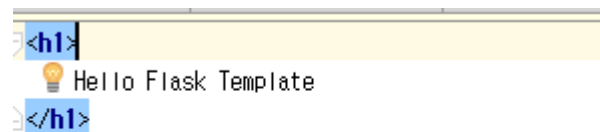
```
from flask import Flask, render_template
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def home():  
    return render_template('home.html')
```

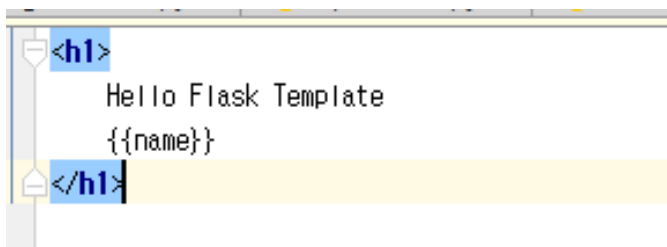
```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=8080)
```



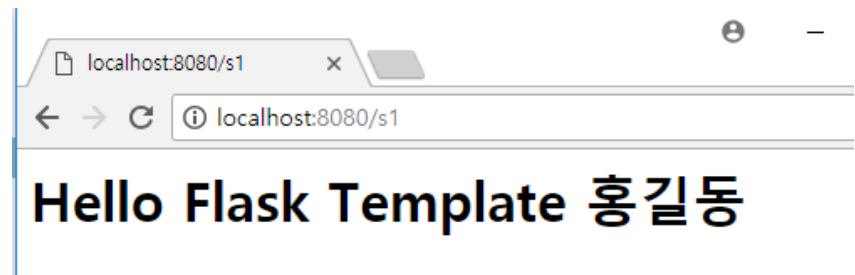
Template 기반 Flask : 변수 공유

- {{변수}} 표현을 사용해 python의 객체 접근

```
return render_template('home.html', name="홍길동")
```



```
<h1>  
Hello Flask Template  
{{name}}  
</h1>
```



Template 기반 Flask : 지시어 for

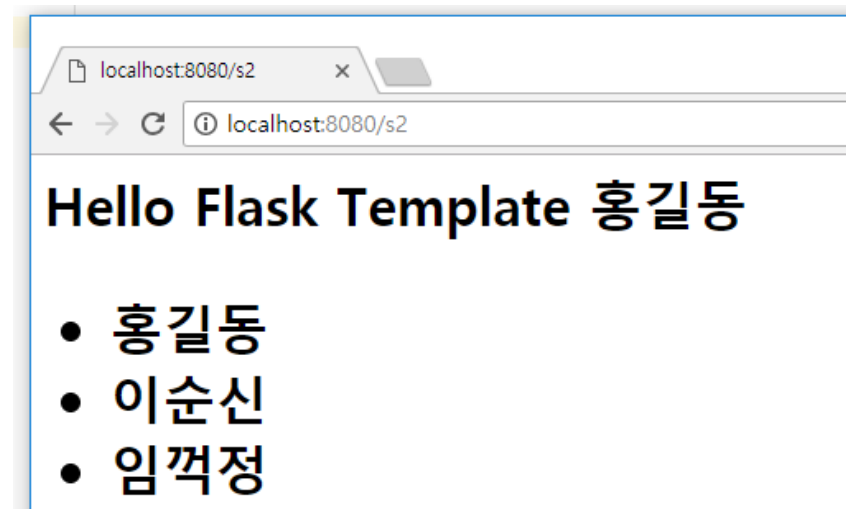
- Html내에서 {% for %}, {% endfor %} 사용해 반복처리

```
@app.route('/s2')
def s2():
    list = ['홍길동', '이순신', '임꺽정']
    return render_template('home.html', name="홍길동", list=list)
```

```
<h1>
Hello Flask Template
{{name}}

<ul>
    {% for s in list %}
        <li> {{s}} </li>
    {% endfor %}

</ul>
</h1>
```





```
<html>
  <body>
    <form action = "/fileUpload" method = "POST"
      enctype = "multipart/form-data">
      <input type = "file" name = "file1" />
      <input type = "submit"/>
    </form>
  </body>
</html>
```

파일 upload



```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route('/fileUpload', methods = ['POST'])
def fileUpload():
    if request.method == 'POST':
        f = request.files['file1']
        f.save("./static/downloads/" + f.filename)
        return '파일 업로드 성공!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True, port=3000)
```



```
from flask import Flask, request, send_file
from io import BytesIO
from PIL import Image, ImageDraw
```

```
app = Flask(__name__)
```

```
@app.route('/get_image')
```

```
def get_image():
```

```
    return send_file("11.jpg", mimetype='image/gif')
```

Dynamic Image 처리

```
@app.route('/get_image2')
def get_image2():
    pil_img = Image.new("RGB", (220, 190))
    img1 = ImageDraw.Draw(pil_img)
    img1.rectangle([(40, 40), (210, 180)], fill="#ffff33",
out_line="red")

    img_io = BytesIO()
    pil_img.save(img_io, 'JPEG', quality=70)
    img_io.seek(0)
    return send_file(img_io, mimetype='image/jpeg')

app.run(host='0.0.0.0', port=8080, debug=True)
```