1. Network (Http / Https / RestFul)

---

1. HTTP

1. HTTP의 정의

⇒ 텍스트 기반의 통신 규약으로 **인터넷에서 데이터를 주고받을 수 있는 프로토콜**

⇒ 모든 프로그램이 이 규약에 맞춰 개발해서 서로 정보를 **교환**할 수 있게 되었다.

2. HTTP의 동작

⇒ 응답(서버)과 요청(클라이언트) 모델을 통해 통신하는 규약

1. 클라이언트가 브라우저를 통해서 어떠한 서비스를 url을 통하거나 다른 것을 통해서 요청
2. 서버에서는 해당 요청사항에 맞는 결과를 찾아서 사용자에게 응답

- 요청 : client → server
- 응답 : server → client

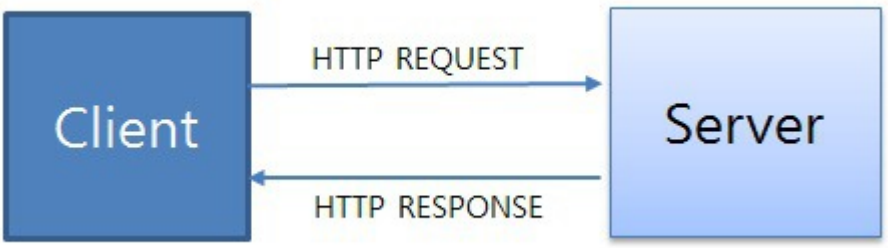
※ 포트 번호 : 80

⇒ HTML 문서만이 HTTP 통신을 위한 유일한 정보 문서는 아니다.

⇒ Plain text 로부터 JSON 데이터 및 XML과 같은 형태의 정보도 통신할 수 있다.

⇒ 보통은 client가 어떠한 정보를 HTML 형태로 받고 싶은지, JSON 형태로 받고 싶은지 명시 해주는 경우가 많다.

3. HTTP의 특징



⇒ HTTP 메시지는 HTTP 서버와 HTTP 클라이언트에 의해 해석

⇒ TCP/ IP를 이용하는 웹 페이지 전송 응용 프로토콜 (Socket을 이용)

- 컴퓨터와 컴퓨터간에 데이터를 전송 할 수 있도록 하는 규약 또는 장치
- 인터넷이라는 거대 통신망을 통해 원하는 데이터 교환 기능을 이용하는 응용 프로토콜

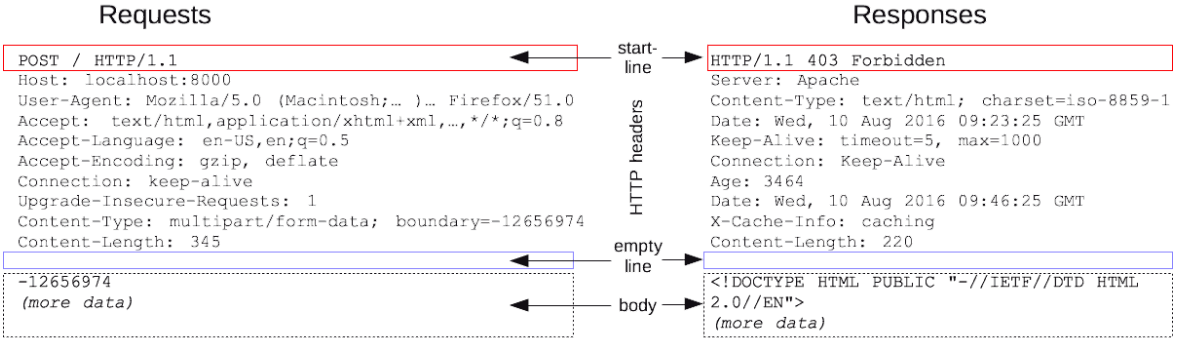
⇒ HTTP는 연결 상태를 유지하지 않는 비연결성 프로토콜

- 이러한 단점을 해결하기 위해 Cookie와 Session이 등장

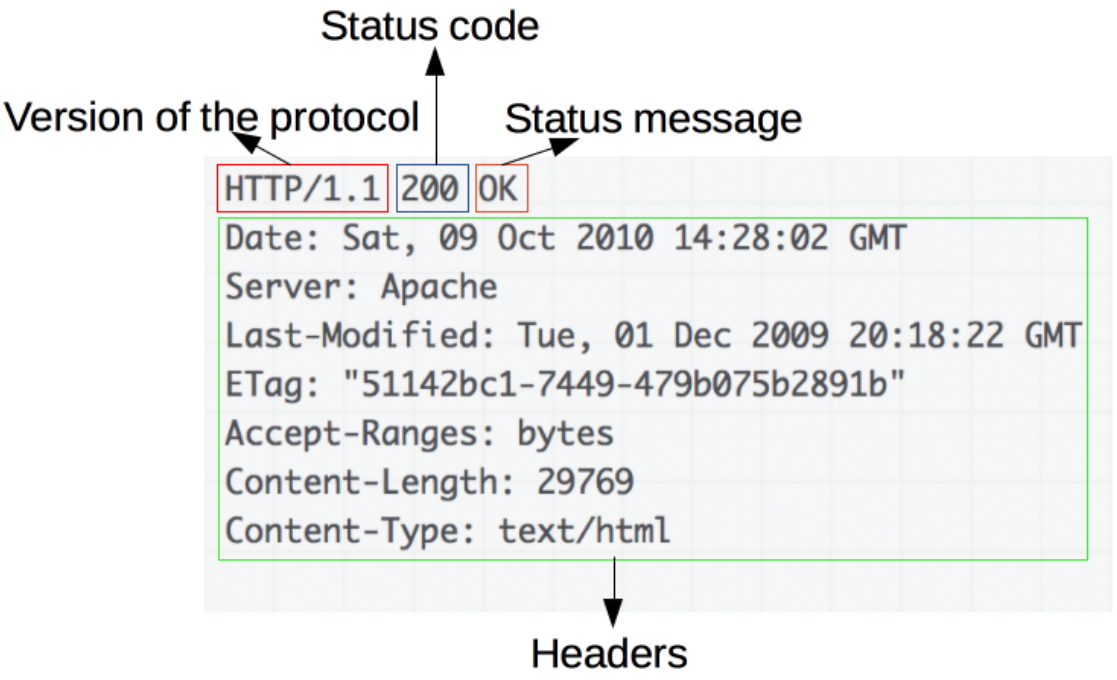
⇒ HTTP는 연결을 유지하지 않는 프로토콜이기 때문에 요청/응답 방식으로 동작

※ HTTP 동작 예시

- **서버** : 어떠한 자료에 대한 접근을 관리하는 네트워크 상의 시스템
  - **(요청에 대한 응답을 보내준다.)**
- **클라이언트** : 그 자료에 접근할 수 있는 프로그램
  - Ex) 웹 브라우저, 핸드폰 어플리케이션 등...



1. 클라이언트 프로그램에서 사용자가 회원가입을 시도
    - 클라이언트와 서버 → TCP socket
    - 클라이언트가 HTTP GET 메시지를 서버로 전송 (요청에 대한 정보를 담아 전송)
2. 서버로 회원정보를 보내게 되고, 서버는 회원 정보를 저장도 추가적으로 해줄 수 있음
    - 서버가 HTTP 응답 메시지를 클라이언트로 전송
    - 서버는 요청 객체를 클라이언트로 전송 (HTTP는 상태가 없음)
3. Client ↔ **클라이언트와 서버 간의 교류가 HTTP라는 규약을 이용하여 발생** ↔ Server



※ HTTP 요청 Method

- **GET** : 자료를 **요청**할 때 사용
- **POST** : 자료의 **생성**을 요청할 때 사용
- **PUT** : 자료의 **수정**을 요청할 때 사용
- **DELETE** : 자료의 **삭제**를 요청할 때 사용

- 그 외) PATCH, HEAD, OPTIONS, CONNECT, TRACE 등...

※ HTTP Response 상태 코드

- 1XX (조건부 응답) : 요청을 받았으며 작업을 계속한다.
- 200 OK
  - 요청한 동작을 수신하여 이해했고 승낙했으며 성공적으로 처리했음을 가리킨다.
- 301 Moved Permanently
  - 클라이언트는 요청을 마치기 위해 추가 동작을 취해야 한다.
- 400 Bad Request & 404 Not Found
  - **클라이언트**에 **오류**가 있음을 나타낸다.
- 505 HTTP Version Not Supported
  - **서버**가 유효한 요청을 명백하게 **수행하지 못했음**을 나타낸다.

4. HTTP의 발전

- **HTTP/1.0**
  - HTTP GET 메소드에 대해서 1개의 TCP 연결 사용
    - 객체 1개 전송에 1개의 TCP 연결 -> 여러 개의 TCP 연결이 1개 웹 페이지에 사용
    - TCP 연결 만들 때 왕복지연시간 문제 -> non-persistent HTTP
- **HTTP/1.1**
  - HTTP GET 메소드 처리에 이전 TCP 연결을 재사용
    - 왕복지연시간 감소 -> persistent HTTP
  - Pipelining 으로 병렬 요청과 응답
    - 요청 : GET HTML + CSS
    - 응답 : HTML + CSS response
- **HTTP/2**
  - 웹 페이지 성능 향상 목표

        - HTTP/1.1에서는 객체가 순차적으로 전송 (1 개의 TCP 연결내에서)
            - Head-of-Line (HoL) 병목
        - 웹 페이지 로딩 시간을 50% 단축 목표
    - 바이너리 프레임
        - 우선순위, 흐름 제어, 서버 푸시
    - 스트림 전송
        - 멀티플렉싱 지원
        - 우선순위 지원
    - 헤더 압축
- **HTTP/3**
    - TCP를 이용하는 HTTP/2의 HoL 병목 현상 해결 위해 개발
        - TCP의 세그먼트 손실은 HTTP에서 보이지 않는 단점

---

2. HTTPS

⇒ HTTP : 암호화가 되지 않은 텍스트를 전송하는 프로토콜

⇒ 누군가 네트워크에서 신호를 가로채면 내용이 노출되는 보안 이슈가 존재

1. HTTPS 정의

⇒ HTTP에 **데이터 암호화가 추가**된 프로토콜

⇒ 인터넷 상에서 정보를 암호화하는 `SSL 프로토콜`을 사용

⇒ 클라이언트와 서버가 자원을 주고 받을 때 사용하는 통신 규약

2. HTTPS 특징

1. HTTP에 Secure Socket가 추가된 형태
    a. 공개키 암호화 방식을 SSL통신 절차에 적용한 형태
2. 기존의 HTTP 통신에 SSL 혹은 TLS 프로토콜를 조합하여 세션 데이터를 암호화

3. 공개키 암호화 방식 사용, 포트 번호 = 433

3. HTTPS 통신 흐름

1. 클라이언트는 서버에 접속하면, 서버인증서(CA)를 받는다.
2. 서버인증서 신뢰 여부 체크 후, 공개키를 추출
3. 클라이언트는 서버와 통신하는 동안만 사용할 대칭키를 임의로 만든다.
    a. 해당 대칭키를 공개키로 암호화 후 전송 한다.
4. 서버는 개인키로 클라이언트가 보낸 메시지를 복호화하여 대칭키를 추출한다.
    a. 해당 대칭키를 이용하여 클라이언트와 통신한다.

---

3. REST API & RestFul

⇒ REST(Representational State Transfer)의 약자로 자원을 이름으로 구분하여, 해당 자원의 상태를 주고 받는 모든 것을 의미한다.

1. HTTP URI(Uniform Resource Identifier)를 통해 **자원**(Resource)을 명시하고,
2. HTTP **Method**(POST, GET, PUT, DELETE, PATCH 등)를 통해
3. 해당 자원(URI)에 대한 **CRUD Operation을 적용**하는 것을 의미한다.

1. REST 구성요소

- **자원(Resource) : HTTP URI**
- **자원에 대한 행위(Verb) : HTTP Method**
- **자원에 대한 행위의 내용 (Representations) : HTTP Message Pay Load**

2. REST의 특징

1. Server-Client(서버-클라이언트 구조)
2. Stateless(무상태)

3. Cacheable(캐시 처리 가능)
4. Layered System(계층화)
5. Uniform Interface(인터페이스 일관성)

※ 장점

- HTTP 프로토콜의 인프라를 그대로 사용
    - REST API 사용을 위한 별도의 인프라를 구축할 필요가 없다.
- HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해 준다.
- **HTTP 표준 프로토콜**에 **따르는 모든 플랫폼**에서 **사용이 가능**하다.
- Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.
- REST API 메시지가 의도하는 바를 명확하게 나타내므로 **의도하는 바를 쉽게 파악**할 수 있다.
- 여러 가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.
- 서버와 클라이언트의 역할을 명확하게 **분리**한다.

※ 단점

- **표준 자체가 존재하지 않아 정의가 필요**하다.
- HTTP Method 형태가 제한적이다.
- 브라우저를 통해 테스트할 일이 많은 서비스인 경우
    - 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
- **구형 브라우저**에서 **호환이 되지 않아** 지원해주지 못하는 동작이 많다.

3. REST API

CRUD	HTTP verbs	Route
resource들의 목록을 표시	GET	/resource
resource 하나의 내용을 표시	GET	/resource/:id
resource를 생성	POST	/resource
resource를 수정	PUT	/resource/:id
resource를 삭제	DELETE	/resource/:id

⇒ **RESPT API :** REST의 원리를 따르는 API

⇒ REST API를 올바르게 설계하기 위해서는 지켜야 하는 규칙 존재

※ API란?

⇒ 데이터와 기능 집합 제공함으로써, 프로그램 간 상호작용 및 통신을 하도록 하는 것

※ REST API 설계 규칙

1. **URI는 동사보다는 명사를, 대문자보다는 소문자를 사용하여야 한다.**

```
http://khj93.com/run/
```

2. **마지막에 슬래시 (/)를 포함하지 않는다.**

```
http://khj93.com/test
```

3. **언더바 대신 하이폰(-)을 사용한다.**

```
http://khj93.com/test-blog
```

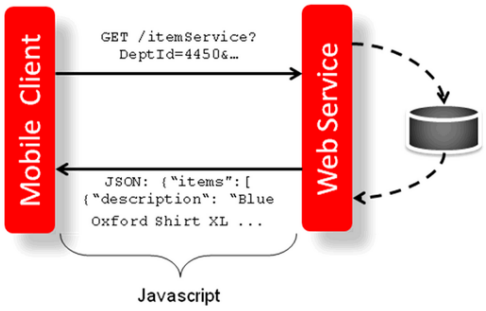
4. **파일확장자는 URI에 포함하지 않는다.**

```
http://khj93.com/photo ( photo.jpg => X )
```

5. **행위를 포함하지 않는다.**

```
http://khj93.com/post/1  ( http://khj93.com/delete-post/1
```

4. RESTful이란?



“RESTful”

⇒ **REST 아키텍쳐 구현**을 따르는 **웹 서비스**, 또는 **시스템**을 의미

- 하지만 REST를 사용했다 하여 모두가 RESTful 한 것은 아님
- 이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것
- RESTful한 API를 구현하는 근본적인 목적이 성능 향상에 있는 것이 아님!
  - 일관적인 컨벤션을 통한 API의 이해도 및 호환성을 높이는 것이 목적
  - 성능이 중요한 상황에서는 굳이 RESTful한 API를 구현할 필요 X!

⇒ REST API의 설계 규칙을 올바르게 지킨 시스템을 RESTful 하다고 한다.