HTTP

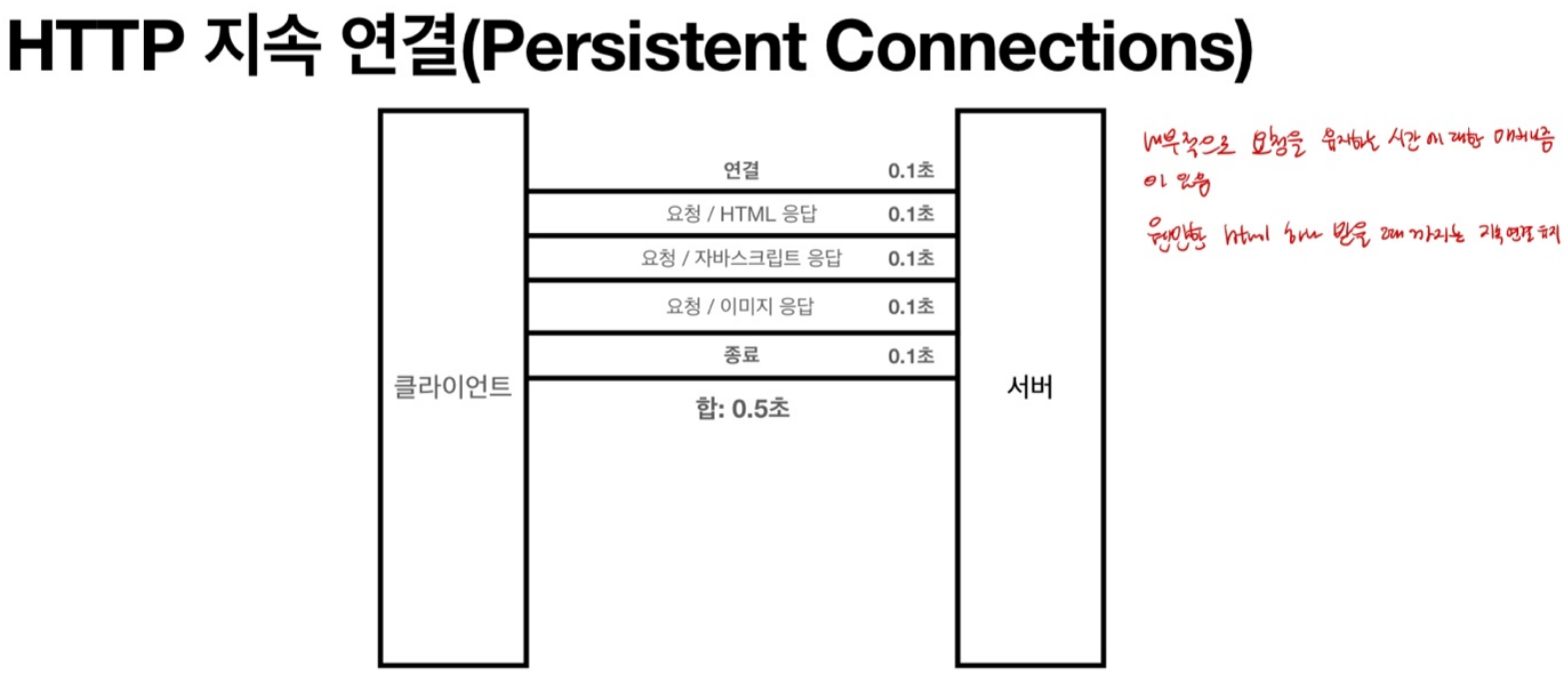
▼ HTTP

HyperText Transfer Protocol

클라이언트-서버 구조로 request/response를 통해 **웹 상에서 정보를 주고받을** 수 있는 **프로토콜**

특징

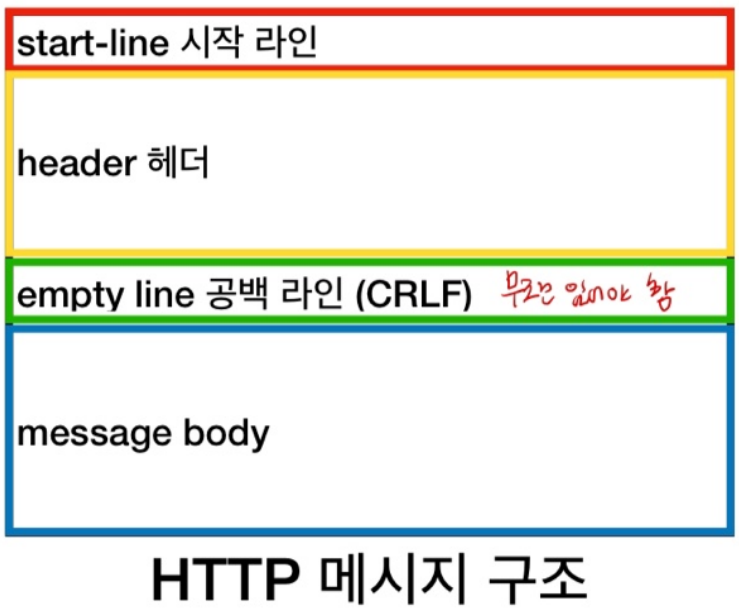
- **클라이언트-서버 구조**
  - **클라이언트**는 서버에 **요청(request)**를 보내고, **서버**는 요청에 대한 결과를 만들어 **응답(response)**을 보냄
  - 이런 구조이기 때문에 클라이언트와 서버가 각각 독립적으로 진화 가능했음
- **TCP/IP** 기반으로 동작, 80번 포트 사용
- **비연결성(Connectionless)**
  - 클라이언트가 **요청**을 서버에 보내고 서버가 적절한 **응답**을 클라이언트에 보내면 바로 **연결이 끊김**
  - 장점: 서버 **자원을 매우 효율적**으로 사용할 수 있음
    - 수천명이 서비스를 사용해도 실제 서버에서 동시에 처리하는 요청은 수십개 이하로 매우 작음
      - ex) 검색하고 한참을 보기 때문에 동시 처리는 적음
  - 단점: TCP/IP 연결 새로맺어야 함 - **3 way handshake 시간 추가**
    - **HTTP 지속 연결(Persistent Connections)**로 문제 해결



- **무상태(Stateless)**
  - **서버**가 **클라이언트의 상태를 보존하지 않음**
    - 연결을 끊는 순간 클라이언트와 서버의 통신은 끝나기 때문에 이전 상태(로그인 유무 등)을 알 수 없음
      - 이걸 해결하기 위해 cookie, session, jwt를 도입
  - 클라이언트가 필요한 데이터를 담아서 요청을 보내서 서버가 장애가 나도 중계서버가 다른 서버로 요청을 보내서 처리 가능
  - 장점: **서버 확장성이 높음(스케일 아웃)**
    - 응답서버를 쉽게 바꿀 수 있기 때문
  - 단점: **클라이언트가 추가 데이터를 전송**
  - **최대한 무상태로 설계**하는 것이 좋음
- 단순하기 때문에 확장이 가능

- **HTTP/1.1을 가장 많이 사용**하고 HTTP/2, HTTP/3은 성능을 개선에 초점을 맞춤 (3은 UDP사용)

HTTP 메시지



- **시작라인**
  - 요청메시지
    - `HTTP메서드` `경로` `HTTP버전`
  - 응답메시지
    - `HTTP버전` `HTTP상태코드` `이유문구`(사람이 이해할 수 있는 짧은 상태코드 설명)
- **헤더**
  - HTTP 전송에 필요한 모든 부가 정보를 담고 있음
- **메시지 바디**
  - 실제 전송할 데이터

단점

- **평문 텍스트**, 즉 암호화되지 않은 텍스트를 전송하는 프로토콜로, **중간자 공격에 취약**
- **변조, 위장, 도청에 취약**

▼ HTTP 메서드

GET

- 리소스 **조회**
- 클라이언트가 서버에게 정보를 요청할 때 사용
- 전달하고 싶은 데이터는 **key-value 쌍 쿼리 스트링**으로 **전달**
- URL에 요청 정보가 이어붙기 때문에 **길이 제한**이 있어서 **대용량의 데이터**를 전송하기 **어려움**
- **캐시가 가능**해서 한번 서버에 GET요청을 한 적 있다면 브라우저가 결과를 저장해 이후 동일한 요청은 브라우저에 저장된 값으로 가져올 수 있음

POST

- **요청 데이터 처리 (주로 생성)**

- 클라이언트가 서버에 요청 데이터를 처리하도록 할 때 사용
- 전달하고 싶은 데이터는 **메시지 바디**를 통해 **전달**
- 리소스 URI에 POST요청이 오면 요청 데이터를 어떻게 처리할지는 리소스마다 따로 정해야 함 → 정해진 것이 없음

1. **새 리소스 생성(등록)**
    - 서버가 아직 식별하지 않은 새 리소스 생성
2. **요청 데이터 처리**
    - **프로세스를 처리하는 경우**
    - POST의 결과로 새로운 리소스가 생성되지 않을 수도 있음
    - ex) 주문에서 결제완료 → 배달시작 처럼 프로세스의 상태 변경되는 경우
3. **다른 메서드로 처리하기 애매한 경우**
    - JSON으로 조회 데이터를 넘겨야 하는데, GET 메서드를 사용하기 어려운 경우

PUT

- 리소스를 **대체**
    - 리소스가 **있으면 완전히 대체**
    - 리소스가 **없으면 생성**
- **클라이언트가 리소스의 위치를 알고 URI를 지정**
    - POST와 차이점

PATCH

- 리소스 **부분 변경**

DELETE

- 리소스 **제거**

HTTP 메서드 속성

- **안전(Safe)**
    - 호출해도 **리소스를 변경하지 않는다**
        - ex) GET
- **멱등(Idempotent)**
    - 한 번 호출하든 100번 호출하든 **결과가 똑같다**
    - ex)
        - GET: 한 번 조회하든, 두번 조회하든 같은 결과가 조회
        - PUT: 결과 대체. 같은 요청을 여러번 해도 최종 결과는 같음
        - DELETE: 결과 삭제. 같은 요청을 여러번 해도 삭제된 결과는 똑같음
        - POST❌: 두번 호출하면 같은 결제가 중복 발생할 수 있음
    - 활용
        - 자동 복구 메커니즘
        - **서버가 정상응답을 못주었을 때, 클라이언트가 같은 요청을 다시 해도 되는가 판단근거**
- **캐시가능(Cacheable)**
    - 응답 결과 리소스를 캐시해서 사용해도 되는가?
        - ex) GET, HEAD, POST, PATCH
    - 실제로는 **GET, HEAD 정도만 캐시로** 사용

- POST,PATCH는 본문 내용까지 캐시키로 고려해야하는데 구현 쉽지 않음

메소드	안전	멱등	캐시 가능
GET	✔	✔	✔
HEAD	✔	✔	✔
POST	✘	✘	✔
PUT	✘	✔	✘
DELETE	✘	✔	✘
CONNECT	✘	✘	✘
OPTIONS	✔	✔	✘
TRACE	✔	✔	✘
PATCH	✘	✘	✔

▼ HTTP 상태코드

클라이언트가 보낸 **HTTP 요청에 대한 서버의 응답 코드**

클라이언트로 부터 받은 request에 대한 서버의 response에 대한 간략할 설명

이를 토대로 클라이언트는 알맞는 대응을 할 수 있음

1xx (정보)

- 요청이 수신되어 처리 중
- 거의 사용되지 않음

2xx (성공)

- **클라이언트가 요청**한 동작을 성공적으로 수신하여 **성공적으로 처리**
- `200 OK`
  - **요청 성공**
  - ex) 잔액조회 성공
- `201 Created`
  - **요청 성공해서 새로운 리소스 생성**
  - ex) 게시글 작성 성공, 회원가입 성공

3xx (리다이렉션)

- 요청을 완료하려면 **추가 행동이 필요**
- **일시적 리다이렉션**
  - 리소스의 **URI가 일시적으로 변경**
  - 모두 기능은 같음
  - `302 Found`
    - 리다이렉트시 **요청 메서드가 GET으로** 변하고, **본문이 제거**될 수 있음
  - `307 Temporary Redirect`

        - 리다이렉트시 **요청 메서드와 본문 유지**(POST로 보내면 POST로 유지)
    - `303 See Other`
        - 리다이렉트시 **요청메서드가 GET으로 변경**

4xx (클라이언트 오류)

- **클라이언트의 요청에 문제가 있음**
    - 똑같은 재시도해도 요청 실패
- `400 Bad Request`
    - **클라이언트가 잘못된 요청**을 해서 서버가 요청을 처리할 수 없음
    - ex) 올바르지 않은 형식의 데이터 입력
- `401 Unauthorize`
    - **인증되지 않은 상태**에서 인증이 필요한 리소스에 접근함
    - ex) 로그인 전에 사용자 정보 요청
- `403 Forbidden`
    - 인증 자격 증명은 있지만 **권한이 없는 리소스에 접근함**
    - ex) 일반 유저가 관리자 메뉴 접근
- `404 Not Found`
    - **요청 리소스를 찾을 수 없음**
        - ex) 존재하지 않는 route에 요청, 클라이언트가 권한이 부족한 리소스에 접근할 때 해당 리소스를 숨기고 싶을 때

5xx (서버 오류)

- **서버 오류**, **서버가 정상 요청을 처리하지 못함**
    - 재시도하면 성공할 수도 있음
- `500 Internal Server Error`
    - **서버 내부 문제로** 오류 발생

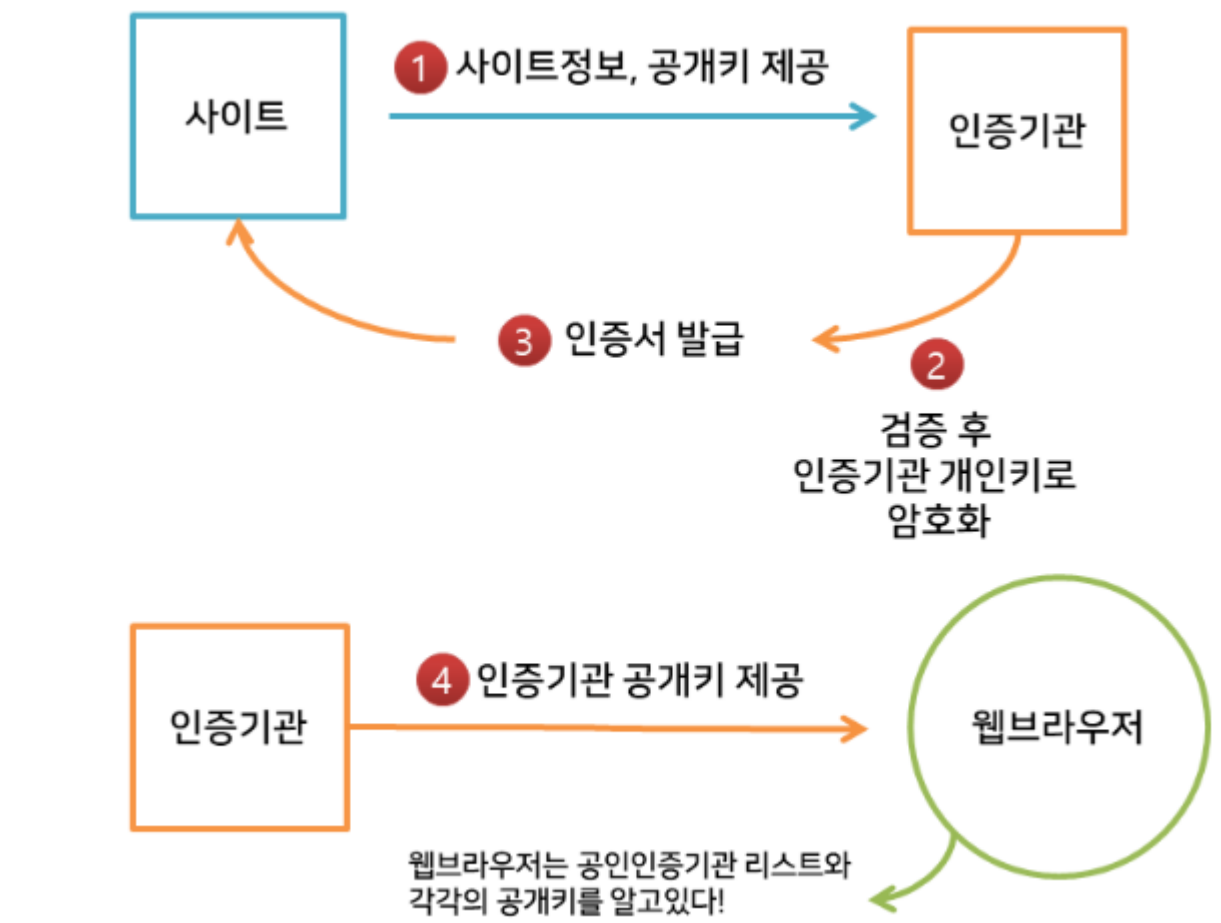
▼ HTTPS

- 웹 통신 프로토콜인 **HTTP의 보안이 강화된 버전**의 프로토콜

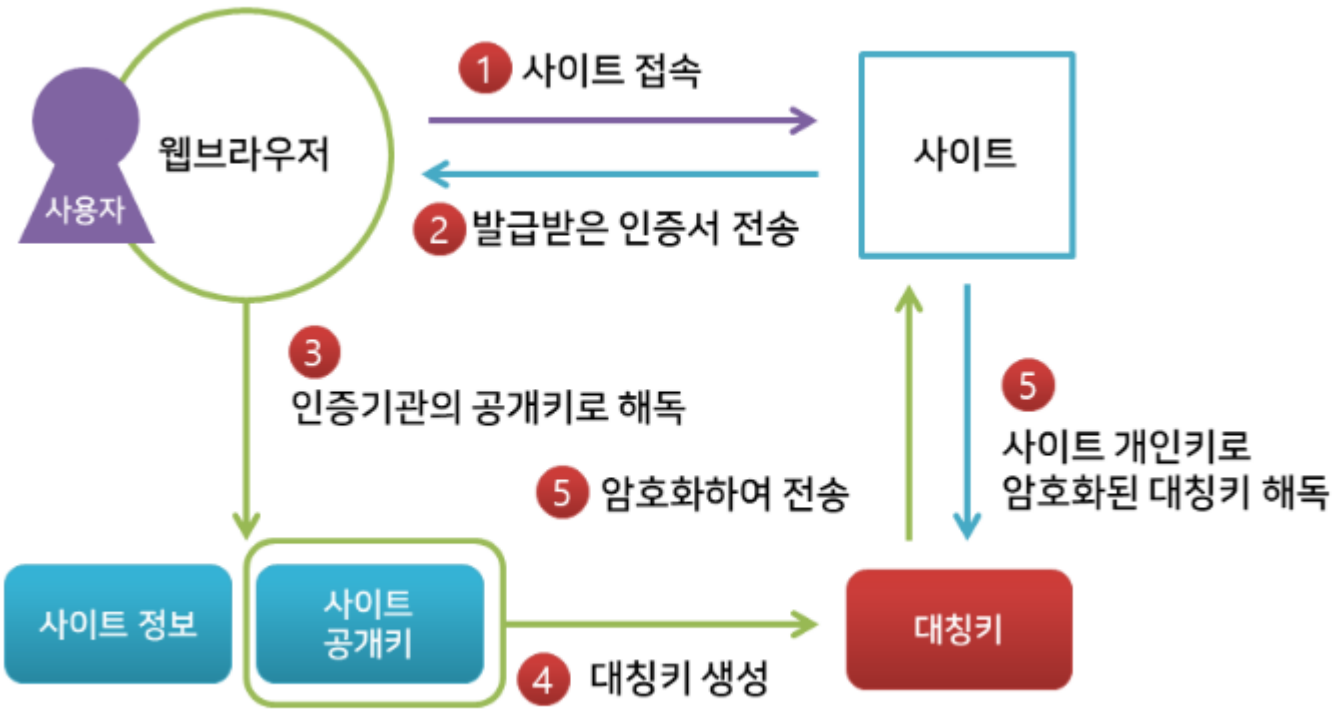
특징

- **TCP/IP**기반으로 **443번 포트** 사용
- 웹 상에서 정보를 암호화하는 `SSL`이나 `TLS` 프로토콜을 통해 **세션 데이터를 암호화**
    - TLS(Transport Layer Security)은 SSL(Secure Socket Layer)에서 발전한 것
    - 두 프로토콜의 주요 목표는 **기밀성(사생활 보호)**, **데이터 무결성**, **ID 및 디지털 인증서**를 사용한 **인증을 제공**하는 것
- **공개키 암호화** 방식으로 **텍스트 암호화**

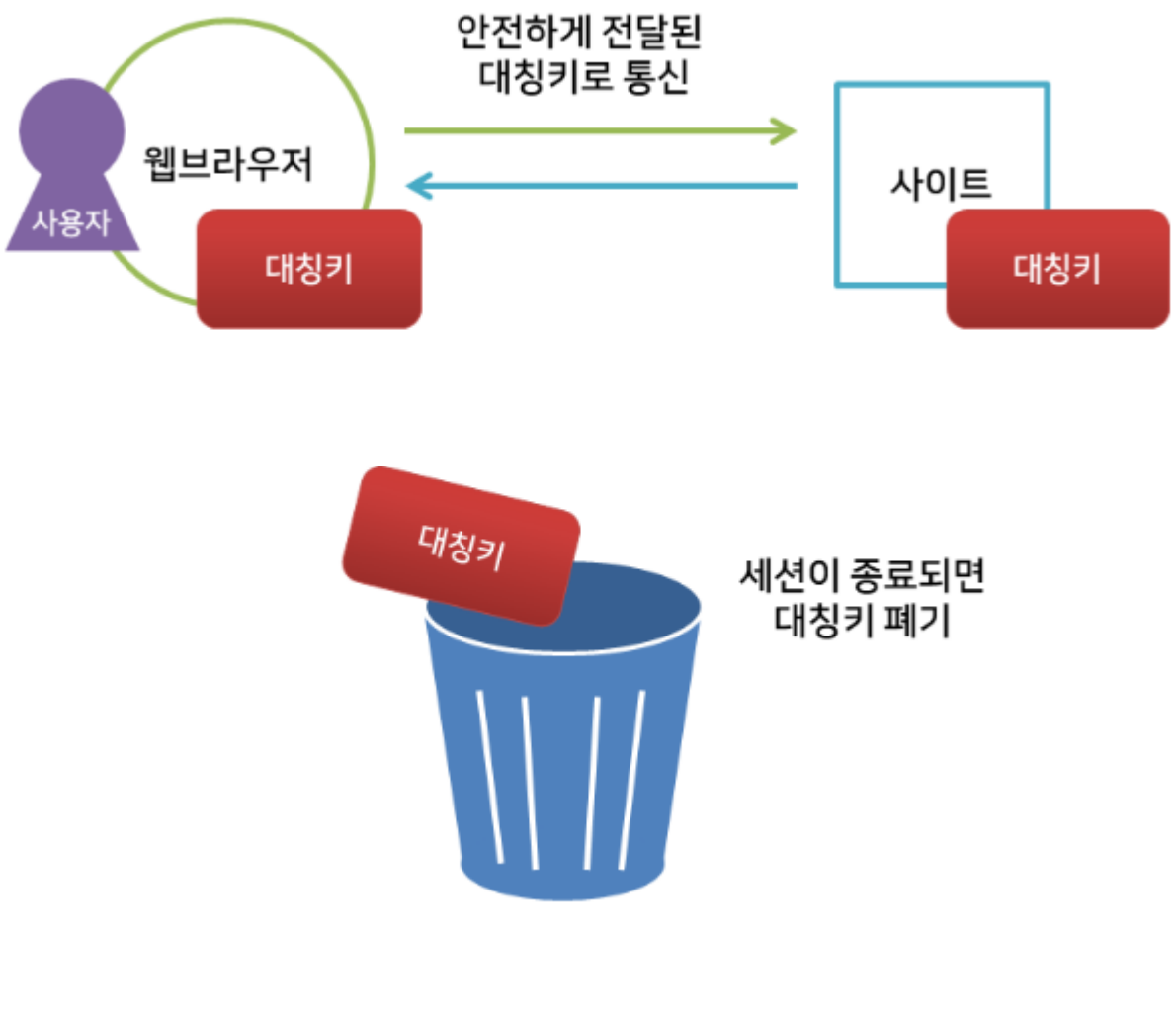
동작과정



1. 인터넷 사이트(서버)는 공개키와 개인키를 만들고, 신뢰할 수 있는 인증 기관(CA)에 자신의 정보와 공개키를 관리해달라고 계약하고 (경우에 따라) 돈을 지불한다.
    - CA: 공개키를 저장해주는 신뢰성이 검증된 민간기업
2. 이 때, 계약을 완료한 인증 기관은 기관만의 공개키와 개인키가 있다. 인증 기관은 사이트가 제출된 데이터를 검증하고, 인증 기관의 개인키로 사이트에서 제출한 정보를 암호화해서 인증서를 만들어 제공한다. 사이트는 인증서를 가지게 되었다.
3. 인증 기관은 웹 브라우저에게 자신의 공개키를 제공한다.



1. 사용자가 사이트에 접속하면 서버는 자신의 인증서를 웹 브라우저(클라이언트)에게 보낸다. 예를 들어, 웹 브라우저가 index.html 파일을 달라고 요청했다면, 서버의 정보를 인증 기관의 개인키로 암호화한 인증서를 받게 되는 것이다.
2. 웹 브라우저는 3.에서 미리 알고 있던 인증기관의 공개키로 인증서를 해독하여 검증한다. 그러면 사이트의 정보와 서버의 공개키를 알 수 있게 된다.이 부분은 보안상의 의미는 없다. 단지 해당 서버로부터 온 응답임을 확신할 수 있게 된다.
3. 이렇게 얻은 서버의 공개키로 대칭키를 암호화해서 다시 사이트에 보낸다.
4. 사이트는 개인키로 암호문을 해독하여 대칭키를 얻게 되고, 이제 대칭키로 데이터를 주고받을 수 있게 된다.



단점

- **암호화**를 하는 과정이 **웹 서버에 부하**를 준다.
- HTTPS는 설치 및 인증서를 유지하는데 **추가 비용이 발생**한다.
- HTTP에 비해 **느리다**.
- 인터넷 **연결이 끊긴 경우 재인증 시간이 소요**된다.
  - HTTP는 비연결형으로 웹 페이지를 보는 중 인터넷 연결이 끊겼다가 다시 연결되어도 페이지를 계속 볼 수 있다.
  - 그러나 HTTPS의 경우에는 소켓(데이터를 주고 받는 경로) 자체에서 인증을 하기 때문에 인터넷 연결이 끊기면 소켓도 끊어져서 다시 HTTPS 인증이 필요하다.

사용이유

- 클라이언트인 **웹브라우저**가 **서버**에 **HTTP를 통해** 웹 페이지나 이미지 **정보를 요청**하면 **서버**는 이 요청에 **응답**하여 **요구하는 정보**를 제공하게 된다.
- 웹 페이지(HTML)는 텍스트이고, **HTTP**를 통해 이런 **텍스트 정보를 교환**하는 것이다.
- 이때 주고받는 **텍스트 정보**에 주민등록번호나 비밀번호와 같이 민감한 정보가 포함된 상태에서 네트워크 상에서 중간에 **제3자가 정보를 가로챈다면 보안상 큰 문제**가 발생한다.
- 즉, 중간에서 정보를 볼 수 없도록 **주고받는 정보를 암호화**하는 방법인 **HTTPS를 사용**하는 것이다.

▼ REST

Representational State Transfer

1. **HTTP URI(Uniform Resource Identifier)**를 통해 **자원(Resource)을 명시**하고,
2. **HTTP Method(POST, GET, PUT, DELETE, PATCH 등)**를 통해
3. **해당 자원(URI)**에 대한 **CRUD Operation**을 **적용**하는 것

구성요소

1. **자원(Resource) : HTTP URI**
2. **자원에 대한 행위(Verb) : HTTP Method**
3. **자원에 대한 행위의 내용 (Representations) : HTTP Message Pay Load**

- JSON 또는 XML

장점

- HTTP 프로토콜의 인프라를 그대로 사용하므로 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 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.
- Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.
- REST API 메시지가 의도하는 바를 명확하게 나타내므로 **의도하는 바를 쉽게 파악할 수 있다**.
- 여러 가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.
- **서버와 클라이언트의 역할을 명확하게 분리**한다.

단점

- **표준**이 자체가 **존재하지 않아** 정의가 필요하다.
- **HTTP Method 형태가 제한적**이다.
- 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
- 구형 브라우저가 아직 제대로 지원해주지 못하는 부분이 존재한다.

사용이유

- 애플리케이션 분리 및 통합/ 다양한 클라이언트의 등장
- 최근의 서버 프로그램은 다양한 브라우저와 안드로이폰, 아이폰과 같은 모바일 디바이스에서도 통신을 할 수 있어야 한다.

⇒ 이러한 **멀티 플랫폼에 대한 지원**을 위해 **서비스 자원에 대한 아키텍처를 세우고 이용**하는 방법을 모색한 결과, REST에 관심을 가지게 되었다.

REST의 특징

- **Server-Client(서버-클라이언트 구조)**
- **Stateless(무상태)**
- **Cacheable(캐시 처리 가능)**
- **Layered System(계층화)**
- **Uniform Interface(인터페이스 일관성)**

REST API

REST의 원리를 따르는 **API**

특징

- 사내 시스템들도 REST 기반으로 시스템을 분산해 **확장성과 재사용성을 높여 유지보수 및 운용을 편리**하게 할 수 있다.
- REST는 HTTP 표준을 기반으로 구현하므로, HTTP를 지원하는 프로그램 언어로 클라이언트, 서버를 구현할 수 있다.
- 즉, REST API를 제작하면 델파이 클라이언트 뿐 아니라, 자바, C#, 웹 등을 이용해 클라이언트를 제작할 수 있다.

설계 규칙

1. **URI는 정보의 자원을 표현해야 한다.**
    - **resource**는 동사보다는 **명사**를, 대문자보다는 **소문자**를 사용한다.
    - resource의 **도큐먼트** 이름으로는 **단수 명사**를 사용해야 한다.
    - resource의 **컬렉션** 이름으로는 **복수 명사**를 사용해야 한다.
    - resource의 **스토어** 이름으로는 **복수 명사**를 사용해야 한다.
    - Ex) `GET /Member/1` -> `GET /members/1`

2. **행위를 포함하지 않는다.**
    - Ex) `GET /members/delete/1` -> `DELETE /members/1`
3. **마지막에 슬래시 (/)를 포함하지 않는다.**
    - Ex) `http://restapi.example.com/houses/apartments/` ❌
4. **언더바(_) 대신 가독성을 높이는데 하이픈(-)을 사용한다.**
5. **URI 경로에는 소문자가 적합하다.**
6. **파일확장자는 URI에 포함하지 않는다.**
7. 리소스 간에는 연관 관계가 있는 경우

   /리소스명/리소스 ID/관계가 있는 다른 리소스명
    - Ex) `GET : /users/{userid}/devices` (일반적으로 소유 'has'의 관계를 표현할 때)

RESTful

REST의 원리를 따르는 **시스템**

'REST API'를 제공하는 웹 서비스를 'RESTful'하다고 할 수 있다.

목적

- **일관적인 컨벤션**을 통한 **API의 이해도 및 호환성을 높이는 것**

https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html