Network Systems

# Index	2
CreatedAt	
Person	A Ally Hyeseong Kim
Status	TODO
Tags	Computer Science Network Systems
UpdatedAt	

References

면접 시리즈4 - 네트워크

Java, JPA, Spring을 주로 다루고 공유합니다.

https://backtony.github.io/interview/2021-12-12-interview-19/

References

1. OSI 7 Layers
2. DNS
3. 4 way-hand shaking
4. Port, Socket
5. CIDR
6. Subnet
7. Cast
8. Maximum Transmission Unit (MTU)
9. TCP, UDP
10. HTTP, HTTPS
11. HTTP, TCP
12. keep-alive header
13. HTTP 1.0 vs HTTP 1.1
14. HTTP/2.0
15. CORS
16. REST API
17. 쿠키, 세션
18. JWT
19. OAuth

---

1. OSI 7 Layers

👨‍💻 TCP/IP 7계층

- Application(응용) 계층 : **최종 목적지로 응용 프로그램과 연관하여 서비스를 수행하는 계층**
  - ex) HTTP, FTP, DNS
- Presentation(표현) 계층 : **데이터 압축, 변환이 이뤄지는 계층**
  - ex) JPEG, MPEG
- Session(세션) 계층 : **데이터가 통신하기 위한 논리적 연결을 담당하는 계층**
  - ex) API, Socket
- Transport(전송) 계층 : **종단 간의 사용자들에게 신뢰성 있는 데이터를 전달하는 계층**
  - 흐름 제어 : 송신측과 수신측 사이의 데이터 처리 속도 차이를 제어
  - 혼잡 제어 : 네트워크 혼잡을 피하기 위해 데이터의 전송 속도 제어
  - 오류 제어 : 오류 검출과 재전송
  - 전송계층은 논리적으로 1:1 연결된 송신과 수신 호스트 즉, 종단 간의 호스트에 신뢰성 관련 기능을 제공하고, 데이터 링크 계층은 물리적으로 1:1 연결된 호스트 사이의 전송 즉, 직접 묶여있는 호스트-노드 또는 노드-노드 사이에서의 신뢰성 관련 기능을 제공한다.
- Network(네트워크) 계층 : **IP를 지정하고 라우터로 경로를 선택해 네트워크를 통해 패킷을 전달하는 계층**
- Data Link(데이터링크) 계층 : **신뢰성 있는 전송을 보장하기 위한 계층**
  - 오류 제어, 흐름 제어, 회선 제어
- Physical(물리) 계층 : **전기적, 물리적 세부 사항을 정의하는 계층**

- 계층을 나눈 이유 : 통신이 일어나는 과정을 단계별로 알 수 있고, 문제가 생기면 그 단계만 수정하면 된다.

1.1. 전송 계층

- TCP 패킷 추적 관리
  - 데이터는 패킷 단위로 쪼개져 같은 목적지로 전송된다.
  - 따라서 패킷에 각각 번호를 부여하여 패킷의 분실 확인 처리를 하기 위해 목적지에서 패킷을 재조립한다.
- 흐름제어
  - 송신측과 수신측 사이의 데이터 처리 속도 차이를 제어하기 위한 기법으로 송신측의 데이터 처리 속도를 조절하여 수신자의 버퍼 오버플로우를 방지한다.
  - 슬라이딩 윈도우를 사용한다.
    - 송신 측에서 0,1,2,3,4,5,6을 보낼 수 있는 프레임을 가지고 있고 0,1을 전송했다면 슬라이딩 윈도우 구조는 2,3,4,5,6으로 변한다.
    - 수신측에서 ACK를 보내게 되면 송신측은 이전에 보낸 데이터 0,1을 수신측에서 정상적으로 받았음을 알게 되고 ACK 개수 만큼 프레임의 수를 오른쪽으로 확장한다.
- 오류제어
  - 프레임이 손상되었거나 손실되었을 경우, 재전송을 통해 오류를 복구한다.
  - 2가지 방식
    - Stop and Wait ARQ
      - 수신측에서 ACK 또는 NAK을 보낸다.
      - 수신측에서 받지 못한 경우 NAK를 보내고 송신측은 해당 데이터를 재전송한다.
      - 만약 분실된 경우 일정 간격의 시간을 두고 타임이웃이 되면 송신측에서 데이터를 재전송한다.
    - Go-Back-n ARQ
      - 손상되거나 분실된 경우 확인된 마지막 프레임 이후로 모든 프레임을 재전송한다.
      - 예를 들어 수신측에서 1, 3을 받게 되면 3을 버리고 2를 받지 못했다고 송신측에 보내고 송신측은 2부터 데이터를 다시 보낸다.

        - SR ARQ
            - Go-Back-n을 보완한 방법으로 손실된 프레임만 재전송한다.
            - 별도의 데이터 재정렬을 수행해야 하며, 별도의 버퍼를 필요로 한다.
- 혼잡제어
    - 네트워크 혼잡을 피하기 위해 송신측에서 보내는 데이터의 전송 속도를 제어하는 것
    - AIMD 방식
        - 처음 패킷 하나를 보내 문제가 없다면 Window Size를 1씩 증가시키는 방식
        - 문제가 발생하면 Window Size를 절반으로 줄인다.
        - 초기에 높은 대역폭을 사용하지 못하여 오랜 시간이 걸리고 네트워크가 혼잡해지는 상황을 미리 감지하지 못한다.
    - Slow Start 방식
        - 처음 패킷을 하나씩 보내는 것은 같지만 매 전송마다 2배씩 증가하여 데이터의 크기가 지수함수적으로 증가한다.
        - 전송되는 데이터의 크기가 임계값에 도달하면 혼잡 회피 단계로 넘어간다.
        - 혼잡 현상이 발생하면 Window size를 1로 줄인다.
        - 혼잡 현상이 발생했던 Window Size 절반 까지는 지수함수 꼴로 증가하고 이후부터는 1씩 증가한다.
    - Fast recovery 빠른 회복 방식
        - 혼잡 시 1로 줄이지 않고 절반으로 줄이고 선형 증가
    - 혼잡 회피
        - 임계값을 넘어가면 1씩만 선형적으로 증가

1.2. TCP/IP 4계층

- 현재는 업데이트되어 5계층으로 불린다.

TCP/IP 5계층

- Application 계층 : Application + Presentation + Session 계층
- Transport 계층
- Network 계층
- Data Link 계층
- Physical 계층

2. DNS

- DNS: IP주소를 문자로 표현한 주소로 바꾸는 시스템 혹은 서버

2.1. www.google.com 접속

1. www.google.com을 브라우저 주소창에 입력한다.
2. 브라우저는 캐싱된 DNS 기록을 통해 www.google.com에 대응되는 IP 주소가 있는지 확인한다.
    - 브라우저 캐시 확인
    - OS 캐시 확인
    - 라우터 캐시 확인
    - ISP 캐시 확인
3. 요청한 URL이 캐시에 없으면 ISP(인터넷 서비스 제공자, kt 등)의 DNS 서버가 www.google.com을 호스팅하고 있는 서버의 IP 주소를 찾기 위해 DNS query를 날려서 찾는다.
4. 브라우저는 IP주소를 받아 서버와 TCP 연결을 한다. (3 way handshaking)
    - 클라이언트가 서버로 접속 요청 SYN 패킷을 보낸다.
    - 서버에서는 수락하는 ACK와 SYN 패킷을 보낸다.
    - 클라이언트는 서버에게 확인 응답으로 ACK 패킷을 보낸다.
5. TCP 연결이 완료되면 브라우저가 웹 서버에 HTTP 요청을 보낸다.
6. 서버는 요청을 처리하고 response를 생성하고 보낸다.
7. 브라우저가 HTML content를 사용자에게 보여준다.

- 서버단에서 클라이언트단으로 index.html을 응답으로 보내게 되는데 여기 안에는 구글의 이미지 google.png가 들어있다. 웹 브라우저는 html파일을 읽어서 해석하는데 이미지 주소가 나오면 다시 해당 url로 서버에 요청을 보내고 위와 같은 요청을 반복한다.

3. 4 way-hand shaking

- HTTP 요청과 응답 과정이 끝나면 연결과정을 종료하는 4-way-handshaking이 진행된다.

1. 클라이언트가 서버로 연결을 종료하겠다는 FIN 패킷 전송
2. 서버는 클라이언트에게 우선적으로 ACK 패킷 전송
3. 서버는 자신의 통신의 끝날 때까지 기다리고 끝나면 클라이언트에게 FIN 패킷 전송
4. 클라이언트는 확인했다는 의미로 ACK 패킷을 서버에게 전송
5. 서버가 보내는 FIN보다 서버가 보내는 데이터가 늦게 보내질 경우를 대비해 클라이언트는 일정 시간 동안 소켓을 닫지 않고 잉여 패킷을 기다림(time wait)
6. 이후에 연결 종료

3.1. TCP

- TCP의 연결 과정에서 3way와 4way가 단계 차이가 나는 이유
  - Client가 데이터 전송을 마쳤다고 하더라도 Server는 아직 보낼 데이터가 남아있을 수 있기 때문에 일단 ACK만 먼저 보내고, 데이터를 모두 전송한 후에 자신도 FIN 메시지를 보내기 때문이다.
- Server에서 FIN 플래그를 전송하기 전에 전송한 패킷이 Routing 지연이나 패킷 유실로 인한 재전송 등으로 인해 FIN 패킷보다 늦게 도작한 상황이 발생하면?
  - 이러한 현상을 대비하여 Client는 Server로부터 FIN 플래그를 수신하더라도 일정기간 time wait동안 세션을 남겨 놓고 잉여 패킷을 기다리는 과정을 거친다.
- 초기 Sequence Number인 ISM을 0부터 시작하지 않고 난수를 생성해서 설정하는 이유
  - 커넥션을 맺을 때 사용하는 포트는 시간이 지남에 따라 재사용된다. 따라서 두 통신이 과거에 사용된 포트 번호 쌍을 사용할 가능성이 생기고 난수가

아닌 순차적 Number가 전송된다면 이전의 커넥션으로부터 오는 패킷으로 인식할 가능성이 생긴다.

4. Port, Socket

- 포트
  - 네트워크를 통해 데이터를 주고받는 프로세스를 식별하기 위해 호스트 내부적으로 프로세스가 할당받는 고유한 값
  - 하나의 IP 주소 내에 개별적으로 부여된 통신 프로세스
- 소켓
  - 네트워크상에서 동작하는 프로그램 간 통신의 종착점(Endpoint)
  - 두 시스템 사이의 네트워크 연결을 나타내는 객체
  - 소켓을 열기 위해선 호스트에 할당된 IP, 포트 번호, 프로토콜이 필요하다.
  - 위 세가지가 소켓을 정의할 수 있지만 유일하게 식별하지는 않는다.
  - 보내는 쪽과 받는 쪽 모두 소켓을 열어야 한다.
  - 하나의 프로세스가 같은 포트를 갖고 여러 개의 소켓을 열 수 있다.

5. CIDR

- 사이더(Classless inter-Domain Routing, CIDR): 클래스없는 도메인간 라우팅 기법
- 기존 IP 주소 할당 방식이었던 클래스를 대체하며 IP주소의 네트워크 영역, 호스트영역을 유연하게 나누어준다.
- 서브넷 마스크는 /를 사용해서 명시한다.

6. Subnet

- Subnetting
  - 기본의 class로 분리하는 IP 주소 체계에서, **네트워크 부분과 호스트 부분으로 IP영역대를 분리하는 것**
  - 분할하는 이유는 브로드캐스트에서 성능 저하가 발생하기 때문
  - 네트워크 부분이 같다면 같은 네트워크에 포함되어있는 호스트이고, 네트워크 부분을 서브넷 마스크를 통해 구분한다.
- Subnet

    - **특정 지역에서 관리되는 IP 영역을 몇 개의 영역으로 나눠서 관리하는 것**
- 장점
    - 네트워크 브로드캐스트 사이즈를 줄일 수 있다.
        - 브로드캐스트 : IP 네트워크에 있는 모든 로컬 네트워크 호스트로 데이터를 전송하는 방식
    - 효율적으로 관리할 수 있다.
    - 네트워크 분리에 따라 보안성 향상
- 10.0.0.0/24 로 예를 들면, 앞에 24비트(서브넷마스크) 10.0.0을 네트워크 영역 뒤에 마지막 0을 호스트 영역이라고 한다. 앞의 24비트는 네트워크 영역으로 같다면 같은 네트워크 대역이다. 뒤쪽의 8비트는 호스트 영역 범위인데 해당 10.0.0 네트워크에 호스트 영역으로 2^8개의 범위, 즉 2^8개의 Ip를 범위를 할당한다는 것이다. 여기서 8은 32비트 - 24비트로 나온 것이다.여기서 4개의 서브넷으로 쪼갠다고 한다면 마지막 8비트의 맨앞 2자리를 서브넷으로 구분하는데 사용한다. 2자리를 사용하는 이유는 2^n>=4 일때 n는 2가 최소값이기 때문이다. 만약 5개의 서브넷으로 쪼갠다면 앞에 3비트를 써야한다.4개의 서브넷으로 쪼갠다면 8비트의 맨앞 2자리가 00, 01, 10, 11로 구분되어 2자리가 서브넷 ID가 되고 나머지 뒤에 6자리가 사용가능한 호스트 ID가 된다.

7. Cast

- 유니 캐스트 : 특정 대상과 1:1 통신
- 멀티 캐스트 : 특정 다수와 1:N 통신
- 브로드 캐스트 : 네트워크에 있는 모든 대상과 통신

8. Maximum Transmission Unit (MTU)

- 패킷이나 프레임의 최대 크기로 데이터의 크기가 크다면 단편화해야 한다.

9. TCP, UDP

- TCP
    - 연결형 서비스를 지원하는 전송계층 프로토콜
    - 흐름 제어(송수신 측의 데이터 처리 속도차이 해결), 혼잡 제어(송신측의 전달과 네트워크 데이터 처리 속도 해결), 오류 제어(오류 검출과 재전송)를 통해 신뢰성을 보장
    - 인터넷 환경(http)에서 기본으로 사용

    - 초기화(3way-hand-shaking)과 종료(4way-hand-shaking) 필요
    - 전이중, 점대점 방식이다.
        - 전이중 : 전송이 양방향으로 동시에 일어날 수 있다.
        - 점대점 : 각 연결이 정확히 2개의 종단점을 가지고 있다.
    - 멀티캐스팅이나 브로드캐스팅을 지원하지 않는다.
- UDP
    - 비연결성 서비스를 지원하는 전송 프로토콜
    - checksum 필드를 통해 최소한의 오류만 검출한다.
    - 신뢰성이 낮지만 속도가 빠르다.(스트리밍 서비스에 이용)
    - 전송을 위한 논리적인 경로가 없다.
    - 전송 순서를 보장하지 않음
    - 수신 여부를 확인하지 않음
- 참고
    - UDP와 TCP는 각각 별도의 포트 주소 공간을 관리하므로 같은 포트 번호를 사용해도 무방하다.
    - 즉, 두 프로토콜에서 동일한 포트 번호를 할당해도 서로 다른 포트로 간주한다.
    - Checksum은 헤더와 데이터의 에러 확인 용도로 사용되나 UDP는 에러 복구를 위한 필드가 불필요하기 때문에 TCP 헤더에 비해 간단하다.

10. HTTP, HTTPS

- HTTP 프로토콜의 특징
    - 비연결 지향(connectionless) : 클라이언트가 request를 서버에 보내고, 서버가 클라이언트에 요청에 맞는 response를 보내면 바로 연결을 끊는다.
    - 상태 정보 유지 안 함(stateless) : 연결을 끊는 순간 클라이언트와 서버의 통신은 끝나며 상태 정보를 유지하지 않는다.
- HTTP
    - WWW 상에서 정보를 주고받을 수 있는 포로토콜
    - 클라이언트와 서버 사이에 이루어지는 요청/응답 프로토콜

    - 주로 HTML 문서를 주고 받는데 사용
    - TCP(HTTP/1, HTTP/2), UDP(HTTP/3)를 사용하며, 80번 포트를 사용한다.
- HTTPS
    - HTTP의 보안상 문제를 해결하기 위해 등장한 프로토콜
    - HTTP는 텍스트로 자원을 주고받기 때문에 네트워크를 가로챈다면 내용이 유출되는 보안 이슈가 발생한다.
    - SSL, TLS(SSL의 최신 버전)를 이용해 암호화하여 주고받음
        - 응용 계층 및 전송 계층 사이에 위치
    - 443 포트 사용
    - 모든 HTTP 요청과 응답 데이터는 네트워크로 보내기지 전에 전송 계층과 응용 계층 사이에서 암호화 된다.
    - HTTP 자체를 암호화하는 것이 아니라 운반하는 HTTP Body 부만 암호화를 진행하고 Header는 암호화하지 않는다.
    - 암호화 통신 방법 (공개키/개인키, 대칭키 방식을 혼합해서 사용)
        - A에서 B로 접속 요청
        - B에서 공개키를 A에게 전달
        - A는 자신의 대칭키를 공개키 A로 암호화해서 B에게 전달
        - B는 개인키로 복호화하여 A의 대칭키를 얻음
        - 얻어낸 대칭키를 이용하여 A와 B가 암호문을 주고 받음
    - 단점
        - 암호화 추가 비용 발생
        - 암호화 과정에서 웹 서버에 부하
        - 연결이 끊기면 재인증 시간이 소요

10.1. HTTP Method

- GET
    - 조회, 리소스 요청
- POST
    - 요청된 데이터 처리

    - 자원을 생성
- PUT
    - 요청된 자원이 없으면 생성
    - 요청된 자원을 새 것으로 전체 갱신
- PATCH
    - 자원의 일부분만 수정
- DELETE
    - 요청된 자원 삭제

10.2. HTTP 멱등성

- 특정 HTTP 메서드를 여러 번 요청을 했을 경우, 매번 요청 결과가 같다면 해당 메소드를 멱등성 메소드라고 한다.
    - GET, PUT, DELETE가 멱등성 메서드에 속하고 POST는 멱등성 메서드가 아니다.
        - 같은 POST를 연속적으로 보낸다면 명령은 여러 번 내린 것처럼 부가적인 결과를 가져온다.(POST 요청을 반복하면 같은 내용이더라도 다른 데이터가 계속 추가된다.)
        - GET은 여러 번 수행해도 서버의 상태가 변하지도 않고 같은 결과를 가져온다.
        - PUT은 여러 번 수행해도 결과적으로 데이터는 요청한 값으로 수정된 항상 같은 상태이다.
        - DELETE도 여러 번 수행해도 이미 존재하든, 존재하지 않든 그 데이터는 DELETE 요청을 보낸 시점에 사라진다.
- cf 안전성: 안전성은 호출해도 리소스를 변경하지 않는 특성으로 서버에 영향을 끼치는 여부로 생각하면 된다.
    - GET 메서드만 안전성 메서드다.

11. HTTP, TCP

- TCP/IP 모델에서 봤을 때, 개념적으로 TCP는 4계층에 존재하고 HTTP는 5계층에 존재한다.
    - 따라서 HTTP는 TCP위에서 동작한다고 할 수 있다.

    - 메시지 형식에 대해서는 HTTP가 제공해주고, 이 메시지를 TCP 방식으로 전달하는 것이다.

12. keep-alive header

- HTTP는 매번 연결을 끊고 새로 생성한다.
- 특정 시간까지는 access가 없더라도 기다리고 연결상태를 유지하는 헤더이다.
- HTTP 1.1부터는 defualt로 keep-alive를 지원한다.

13. HTTP 1.0 vs HTTP 1.1

- HTTP 1.0
    - 요청 컨텐츠마다 TCP 커넥션을 맺고 끊음을 반복한다.
    - 요청1 -> 응답1 -> 요청2 -> 응답2 순으로 순차적으로 진행된다. 즉, 응답을 받아야만 다음 작업을 한다.
- HTTP 1.1
    - 매 요청마다 TCP 커넥션을 맺고 끊음을 반복하지 않고 keep-alive를 통해 일정 시간 동안 커넥션을 유지한다.
    - 클라이언트는 각 요청에 대한 응답을 기다리지 않고 여러개의 요청을 연속적으로 보낸다.(파이프라이닝) 하지만 각 응답의 처리는 순차적으로 처리된다.

14. HTTP/2.0

- 하나의 커넥션으로 동시에 여러 개의 메세지를 주고 받을 수 있으며, response는 순서에 상관없이 stream으로 주고받는다.
- 서버는 클라이언트의 요청에 대해 요청하지 않은 리소스를 마음대로 보낼 수 있다.
- Header의 내용과 중복되는 필드를 재전송하지 않아 데이터를 절약한다.

14.1. Multiplexed Streams

- HTTP/2.0은 한 커넥션에서 여러개의 메시지를 주고받을 수 있으며, 응답은 순서에 상관없이 stream으로 주고받는다.
    - 하나의 커넥션에서 여러 병렬 스트림(3개)가 존재한다. Stream이 뒤섞여서 전송될 경우, 수신측에서 stream number을 이용해 재조합한다.

14.2. Server Push

- 서버는 클라이언트 요청에 대해 요청하지 않은 리소스도 보낼 수 있다.
- HTTP/1.1에서는 HTML문서를 요청하고 받은뒤 그 안에 css,image 파일이 있다면 서버로 재요청했으나, HTTP/2.0에서는 Server Push 기능으로 클라이언트에서 요청하지 않은 (HTML문서에 포함된 리소스) 리소스를 Push 해주는 방법으로 클라이언트의 요청을 최소화하여 성능을 향상시킨다.

14.3. Header Compression

- Header의 내용과 중복되는 필드를 재전송하지 않도록 하여, 데이터를 절약한다.
- 기존에 HTTP Header가 Plain Text(평문)이었지만, HTTP/2에서는 Hash Table과 Huffman Coding을 사용하는 HPACK이라는 Header 압축방식을 이용하여 데이터 전송 효율을 높였다.

15. CORS

- 교차 출처 리소스 공유(Cross-Origin Resource Sharing, CORS)는 추가 HTTP 헤더를 사용하여, 한 출처에서 실행 중인 웹 애플리케이션이 다른 출처의 선택한 자원에 접근할 수 있는 권한을 부여하도록 브라우저에 알려주는 체제이다.웹 애플리케이션은 리소스가 자신의 출처(도메인, 프로토콜, 포트)와 다를 때 교차 출처 HTTP 요청을 실행한다.
- 브라우저는 보안상의 이유로, 스크립트에서 시작한 교차 출처 HTTP 요청을 제한한다.예를 들면, domain-a.com <-> domain-b.com 간의 요청은 CORS 정책 위반으로, 브라우저에서 요청을 제한한다.따라서 다른 출처의 리소스를 불러오기 위해서는, 그 출처에서 교차 출처 리소스 공유에 대한 헤더(CORS)를 응답 시 반환해주어야 한다.

CORS 동작 방법

1. 실제 요청을 보내기 전에 Preflight라는 예비 요청을 보낸다. 이때 HTTP의 OPTIONS 메서드를 이용해 서버에 보낸다.
2. 서버는 예비 요청이 CORS를 위반하고 있는지를 확인하고 정보를 클라이언트에게 보낸다.
3. Preflight에 대해 서버 응답이 안전하다면 실제 요청을 보낸다.

👩‍💻 Spring에서 CORS 동작

- Servlet Filter를 커스텀한 Cors 설정하는 방식
- WebMvcConfigurer를 구현한 Configuration 클래스를 만들어서 addCorsMapping을 재정의하는 방식
- Spring Security에서 CorsConfigurationSource를 빈 등록하고 Config에 추가하는 방식

16. REST API

- REST: HTTP URI를 통해 자원을 명시하고 HTTP Method를 통해 자원을 처리하도록 설계된 아키텍처
- REST API: REST를 기반으로 만든 API
- RESTful: REST 아키텍처를 구현하는 웹서비스를 나타내는 것으로 REST 원리를 따르는 시스템
- HATEOAS: 동적인 API를 제공
  - 모든 관련된 동작을 URI를 통해 알려주어, 클라이언트가 API의 변화에 일일이 대응하지 않아도 된다는 장점이 있다.
- HTTP를 기반으로 동작하다보니 HTTP의 이점을 그대로 가져올 수 있다.(무상태, 캐시처리 등등..)

16.1. REST하지 않다는 것의 의미

- HTTP Method를 한가지만 사용하고 있다.(Post만 사용하고 있다던가)
- URL에 동사를 사용하고 있다.(리소스는 명사적 특성을 갖는다.)
- HTTP의 자원을 제대로 활용하지 않고 있다면 REST하지 않다고 볼 수 있다.

17. 쿠키, 세션

- 기본적으로 HTTP 프로토콜 환경은 “connectionless, stateless”한 특성을 가지기 때문에 서버는 클라이언트가 누구인지 매번 확인해야 한다.이 특성을 보완하기 위해서 쿠키와 세션을 사용한다.
- 쿠키
  - **클라이언트 로컬에 저장되는 키와 값이 들어있는 작은 데이터 파일**
- 세션

    - **일정 시간 동안 같은 브라우저로부터 들어오는 일련의 요구를 하나의 상태로 보고 그 상태를 유지하는 것**
- 세션과 쿠키의 차이점
    - 저장 위치
        - 쿠키는 클라이언트, 세션은 서버
    - 라이프 사이클
        - 쿠키는 만료시간까지 유지, 세션은 브라우저 종료 시 삭제
    - 속도
        - 세션은 정보가 서버에 있기 때문에 처리가 요구되어 쿠키가 더 빠르다.
    - 보안
        - 쿠키는 클라이언트 로컬에 저장되어 취약하지만, 세션은 서버에서 관리해서 비교적 안전
- 세션보다 주로 쿠키를 사용하는 이유
    - 세션은 서버의 자원을 사용하기 때문에 사용자가 많은수록 소모되는 자원이 상당하기 때문

18. JWT

- 보통 로그인 방식으로는 세션과 쿠기를 통해 구현한다.
- 세션에 저장된 정보는 고유 세션ID가 부여되는데 사용자가 로그인을 하면 서버는 쿠키에 세션ID를 실어서 보낸다. 브라우저는 쿠키를 쿠키저장소에 갖고 있다가 다음 페이지를 요청할 때 해당 유저의 쿠키를 다시 요청헤더에 포함해서 전송한다. 서버는 쿠키 내부의 세션ID를 통해 세션 내부에 일치하는 유저 정보를 가져와서 처음에 로그인한 유저가 맞는지 확인하는 작업이 반복되면서 로그인 상태가 유지된다.
- 세션은 서버의 메모리 내부에 저장되기 때문에 많아지는 만큼 메모리에 부하가 걸리게 되고, 확장 시 세션을 분산시키는 기술을 따로 설계해야 한다.
- JWT는 토큰을 만들어 발행하는 방식으로 쿠키에 담거나 HTTP 헤더에 담아서 보낼 수 있다. 따로 서버에서 저장하지 않기 때문에 토큰이 유효한지 검증하는 과정만 있으면 된다. 하지만 탈취에 의해 악의적으로 사용될 가능성이 있기에 보통 유효시간을 짧게 가져가고 Refresh 토큰을 사용하는 방식으로 구현한다.
- 쿠키는 웹브라우저에서 사용할 수 있는 기능이므로 모바일 애플리케이션에서는 JWT를 사용한 인증방식이 최적이다.

19. OAuth

- 자신이 소유한 리소스에 소프트웨어 애플리케이션이 접근할 수 있도록 허용해 줌으로써 접근 권한을 위임해주는 개방형 표준 프로토콜

20. URI, URL, URN

- URI: 인터넷 자원을 식별하기 위한 문자열
  - URI는 인터넷 주소 같은 것으로 정보 리소스를 유일하게 식별하고 위치를 지정할 수 있다.

    `http://test.com/company/location`

    `http://test.com/member/jobhope`
- URL: 웹에서 자원의 위치
  - 특정 서버의 한 리소스에 대해 구체적인 위치를 표현한다.

```
safefood.com 서버에서 food폴더 안의 salad.png를 요청하는 URL
>> http://safefood.com/food/salad.png

http://example.com/mypage.html - 실제 사이트 URL
http://img.naver.net/static/www/dl_qr_naver.png - 네이버 앱QR코드의 이미지에 대한 UR
L
```

- URN: 어떤 리소스가 있을 때 해당 리소스의 위치에 영향을 받지 않는 고유한 이름 역할
  - 객체를 가리키는 실제 객체 이름을 사용하는 방식으로 위치가 바뀌더라도 리소스의 위치를 찾을 수 있게 된다.

21. 중간자 공격

- 중간자 공격: 두 사람이 서로 통신을 주고 받는데, 중간에 제 3의 인물이 끼여서 데이터를 중계한다면, 이 제3자는 그 내용을 모두 알 수 있고, 데이터를 위/변조할 경우 두 사람에게 서로 잘못된 정보를 전달하게 만들 수도 있다. 중간에서 데이터를 가로채는 것
  - 스니핑: 시스템 및 네트워크에서 들어오고 나가는 데이터 패킷을 캡쳐한다.
  - 패킷 주입: 공격자가 일반 데이터와 함께 악의적인 데이터를 주입한다.
  - 세션 하이재킹: 세션 가로채기라고도 불리며 두 시스템 간 연결이 활성화된 상태를 가로채는 것

    - 보안 소켓 계층 스트리핑: 공격자는 SSL/TLS 연결을 차단하여 프로토콜 보안성이 있는 HTTPS에서 안전하지 않은 HTTP로 전환한다.
- 이 공격법을 방어하기 위한 목적으로 만들어진 것이 TLS 통신이다. TLS 통신은 클라이언트가 연결을 요청하면 서버에서 자신의 공개키가 포함된 인증서를 보내고, 클라이언트는 인증서의 내용을 연결 프로그램(웹 브라우저 등) 또는 운영체제에 내장된 루트 인증서 정보와 비교하여 무결성을 검증한 뒤, 인증서에 있는 공개키로 대칭형 암호화 키를 만들 수 있는 난수 정보를 공개키로 암호화해 서버로 보내며, 클라이언트와 서버가 각각 이 난수에서 암호화 키를 만들어내 암호화 연결이 개시된다.

22. WebSocket과 Socket.io

- WebSocket
    - 개념
        - 웹 페이지의 한계에서 벗어나 실시간으로 상호작용하는 웹 서비스를 만드는 표준 기술
    - 배경
        - HTTP 프로토콜을 클라이언트에서 서버로의 단방향 통신을 위해 만들어진 방법이다.
        - 실시간 웹을 구현하기 위해서는 양방향 통신이 가능해야 하는데, WebSocket 이전에는 AJAX를 이용해서 구현했고 각 브라우저마다 구현 방법이 달라 개발이 어려웠다.
        - 이를 위해 HTML5 표준의 일부로 WebSocket이 등장했다.
    - 특징
        - 소켓을 이용하여 자유롭게 데이터를 주고 받는다.
        - http 대신 ws://로 시작하여 streaming과 유사한 방식으로 푸쉬를 지원한다.
        - 브라우저와 마찬가지로 서버에서도 WebSocket을 지원해야 한다.
        - 아직 확정된 상태가 아니라 브라우저별로 지원하는 WebSocket 버전이 다르다.
        - 즉, 시범적 기술로 아직 써볼만한 기술은 아니다.
- Socket.io
    - 개념

        - 다양한 방식의 실시간 웹 기술을 사용할 수 있는 모듈
        - WebSocket, FlashSocket, 등 다양한 방법을 하나의 API로 추상화 한 것
        - Socket.io는 JavaScript를 이용하여 브라우저 종류에 상관없이 실시간 웹을 구현할 수 있도록 한 기술
    - 특징
        - 현재 바로 사용할 수 있는 기술
        - WebSocket을 지원하는 여러 서버 구현체가 있지만 socket.io는 Node.js 밖에 없다.
        - 개발자는 socket.io로 개발하고 클라이언트로 푸쉬 메시지를 보내기만 하면, websocket을 지원하지 않는 브라우저의 경우 브라우저 모델과 버전에 따라 다른 방법으로 내부적으로 푸쉬를 보낸다.
- 실시간으로 상호작용하는 웹 서비스를 만드는 기술은 여러 개가 있고 표준 기술은 WebSocket이다. 아직 시범적인 기술이고, 이런 것들을 추상화 한 것이 Socket.io이다.

23. gRPC

- RPC
    - 프로그램이 네트워크의 세부 정보를 이해하지 않고도 네트워크 안의 다른 컴퓨터에 있는 프로그램에서 서비스를 요청하는 프로토콜
    - 클라이언트에서 서비스를 요청(function call) 하면 서버에서 서비스를 제공하는 client-server 모델
- gRPC
    - 구글에서 만든 RPC프레임워크로 protocol buffer와 HTTP2.0 를 사용
        - protocol buffer는 구조화된 데이터를 직렬화하기 위한 구글의 확장 가능한 메커니즘
        - HTTP2.0을 사용함기에 내부적으로 헤더 압축, protobuf로 통신시점에는 바이너리 데이터로 통신하기에 메시지 크기가 작고 양방향 통신이 가능
    - SSL/TLS를 사용하여 서버를 인증하고 클라이언트와 서버간 교환되는 모든 데이터는 암호화

- IDL(identity Definition Language)만 정의하면 높은 성능을 보장하는 서비스와 메시지에 대한 소스코드가 각 언어에 맞게 자동으로 생성된다. 따라서 개발자들은 생성된 코드를 클라이언트, 서버 간의 사용 언어에 구애받지 않고 사용하기만 하면 되어 정해진 규약을 공통으로 사용하기 때문에 의사소통 비용이 감소하게 된다.(JSON을 사용하는 HTTP의 경우 공식 표준이 없기 때문에 불필요한 의사소통 시간이 생긴다.)