**HTTP 및 URL 통신에 관하여(2주차 과제)**

1. **REST API**

REST(Representational State Transfer) API는 웹 서비스를 위한 아키텍처 스타일 중 하나로, HTTP 프로토콜을 기반으로 한다. REST API는 클라이언트와 서버 간의 통신을 위한 규칙과 규약을 정의한다. 이를 통해 다양한 클라이언트(웹 애플리케이션, 모바일 앱 등)가 서버(웹 서버, 데이터베이스 등)의 리소스에 접근하고 조작할 수 있는 방법을 제공한다.

REST API의 주요 특징)

* 1. Resource-Centric : 모든 자원에 고유한 식별자(URI)를 할당하고, 해당 자원에 대한 CRUD(Create, Read, Update, Delete) 작업을 수행한다.
  2. State-Transfer : 클라이언트와 서버 간의 통신은 상태를 전이시키는 것으로 이루어진다. 클라이언트의 요청에 따라 서버의 상태가 변경되고, 그 결과가 클라이언트에게 전달된다.
  3. Stateless : 각각의 요청은 서버에 저장되는 세션 정보 등의 상태를 가지지 않는다. 클라이언트의 모든 요청은 충분한 정보를 포함하고 있어야 하며, 서버는 이를 기반으로 요청을 처리한다. 얼핏 State-Tranfer와 상충하는 거 같지만, 요청 간에 클라이언트의 상태를 함유하지 않는다는 것을 뜻한다는 점에서 충돌이 없다
  4. Layered System : . 클라이언트는 서버에 직접 접근하는 대신, 중간 서버(로드 밸런서, 캐시 등)를 통해 간접적으로 접근할 수 있다.
  5. Cacheable : 응답은 캐시될 수 있어야 하며, 클라이언트는 캐시된 응답을 재사용할 수 있다.
  6. Interface Consistency : 일관된 인터페이스를 제공하기 위해, URI 구조와 HTTP 메서드를 통해 자원에 접근하고 조작한다.

1. **URL 통신**

브라우저에 URL을 입력하고 요청을 보내면, 다음과 같은 과정이 발생한다:

1. URL(Uniform Resource Locator) 해석 : 브라우저는 입력된 URL을 해석하여 호스트(HOST)명, 포트(PORT), 경로(PATH) 등의 요소로 분해한다.
2. DNS(Domain Name System) 조회\* : 호스트명을 IP 주소로 변환하기 위해 DNS서버에 요청을 보낸다. DNS는 호스트명을 해당 호스트의 IP 주소로 매핑해주는 역할을 한다.
3. TCP(Transmission Control Protocol) 연결\* : 브라우저는 해당 호스트의 IP 주소와 포트를 기반으로 TCP/IP 연결을 시도한다. 이를 위해 TCP의 3-way 핸드셰이크(handshake) 과정이 발생한다. 클라이언트(브라우저)가 서버와 연결을 설정하고, 데이터 전송을 위한 준비를 마친다.
4. HTTP(HyperText Transfer Protocol) 요청\* : TCP 연결이 설정되면, 브라우저는 HTTP 요청 메시지를 생성하여 서버로 전송한다. 이 요청 메시지에는 요청하는 자원의 메서드(GET, POST 등), 경로, 헤더(추가 정보), 본문(POST 요청일 경우), 쿼리 파라미터 등이 포함될 수 있다.
5. 서버 처리 : 서버는 요청을 받고, 해당 요청에 대한 처리를 시작한다. 이 단계에서 서버는 요청된 자원을 찾고, 필요한 데이터베이스나 다른 서비스에 접근하여 요청을 처리한다.
6. HTTP 응답 : 요청에 대한 처리가 완료되면, HTTP 응답 메시지를 생성하여 클라이언트로 전송한다. 응답 메시지에는 응답 상태 코드(200 OK, 404 Not Found 등), 헤더(추가 정보), 본문(응답 데이터) 등이 포함될 수 있다.
7. TCP 연결 종료 : 응답 메시지 전송 후, 서버와 클라이언트 사이의 TCP 연결이 종료된다. 이를 위해 TCP의 4-way 핸드셰이크\* 과정이 발생한다.
8. 응답 처리 : 브라우저는 받은 HTTP 응답 메시지를 해석하고, 필요에 따라 화면에 표시한다. 이는 HTML, CSS, JavaScript 등을 이용하여 브라우저에 웹 페이지를 렌더링하는 과정을 포함할 수 있다.

이와 같은 과정을 거쳐 브라우저는 사용자가 입력한 URL에 대한 서버의 응답을 화면에 표시하게 된다.

1. **DNS\***

DNS(Domain Name System)는 인터넷 상에서 도메인 이름과 IP 주소를 매핑하는 시스템이다. 인터넷에서 통신할 때는 컴퓨터나 기기의 IP 주소를 이용하여 통신하는 것이 일반적이지만, 사람이 이해하기 쉬운 도메인 이름을 사용하여 통신하는 것이 편리하다. DNS는 이러한 도메인 이름을 실제 네트워크에서 사용되는 IP 주소로 변환해주는 역할을 한다.

DNS는 전 세계적으로 분산된 계층적인 구조를 가지고 있으며, 다양한 DNS 서버로 구성되어 있다. 이러한 구조를 통해 도메인 이름에 대한 요청이 빠르고 효율적으로 처리될 수 있다.

DNS의 주요 기능:

도메인 이름 해석: DNS는 사용자가 입력한 도메인 이름을 해당 도메인에 대응되는 IP 주소로 해석한다. 이를 통해 사용자는 도메인 이름을 통해 웹사이트나 서비스에 접속할 수 있다.

도메인 이름 등록: DNS는 새로운 도메인 이름을 등록하고, 이에 대응되는 IP 주소를 관리한다.

DNS 캐싱: DNS 서버는 이전에 해석한 도메인 이름에 대한 정보를 일정 기간 동안 캐싱하여 저장한다. 이를 통해 동일한 도메인 이름에 대한 반복적인 요청을 처리할 때, 더 빠르고 효율적으로 응답할 수 있다.

DNS는 인터넷 통신에서 필수적인 요소로, 사용자가 웹사이트에 접속하거나 이메일을 보내는 등의 작업을 할 때 뒷받침하는 핵심 기술 중 하나이다.

1. **TCP \***

TCP(Transmission Control Protocol)는 네트워크 통신에서 사용되는 주요한 프로토콜 중 하나이다. TCP는 데이터의 신뢰성과 정확성을 보장하기 위해 설계되었으며, 인터넷 프로토콜 스위트(IP 프로토콜을 포함한 프로토콜 집합)의 핵심 부분이다.

TCP의 주요 특징:

1. Credibility : TCP는 데이터를 전송하는 과정에서 손실, 중복, 순서 교환 등의 문제를 해결하여 신뢰성 있는 데이터 전송을 보장한다. 이를 위해 데이터 전송 시 데이터의 확인과 재전송 등의 메커니즘을 사용한다.
2. 연결 지향성(Connection-oriented): TCP는 통신을 시작하기 전에 먼저 연결을 설정해야 한다. 연결 설정 후에 데이터를 주고받게 되며, 통신이 완료되면 연결을 해제한다. 이러한 연결 지향성은 데이터의 신뢰성을 높이는데 기여한다.
3. 흐름 제어(Flow Control): TCP는 수신자가 자신의 처리 속도에 비해 느린 송신자의 데이터 전송을 제어하기 위한 흐름 제어 메커니즘을 제공한다. 수신자는 자신이 처리할 수 있는 양의 데이터만을 수용하도록 송신자에게 알려주는 방식으로 동작한다.
4. 혼잡 제어(Congestion Control): TCP는 네트워크 내의 혼잡 상황을 감지하고, 이에 따라 데이터 전송 속도를 조절하여 네트워크 혼잡을 방지한다. 이를 통해 네트워크 전체의 성능을 향상시키고 데이터의 신뢰성을 보장한다.
5. 전이중(Full-duplex) 통신: TCP 연결은 전이중 통신을 지원한다. 즉, 데이터를 송신하고 수신하는 것이 동시에 가능하다.
6. TCP는 주로 웹 브라우징, 이메일, 파일 전송 및 원격 접속과 같은 애플리케이션에서 사용된다. 신뢰성과 정확성이 요구되는 통신에 적합한 프로토콜로 널리 사용되고 있다.
7. **HTTP 요청\***

HTTP(HyperText Transfer Protocol)는 인터넷 상에서 데이터를 주고받기 위한 프로토콜 중 하나이다. HTTP는 클라이언트와 서버 간의 통신을 위해 설계되었으며, 주로 웹 브라우저와 웹 서버 간에 사용된다.

HTTP 통신의 주요 특징은 다음과 같다:

1. 요청-응답 구조: HTTP는 클라이언트가 서버에 요청(Request)을 보내면, 서버는 요청에 대한 응답(Response)을 반환하는 구조를 가지고 있다. 이 요청과 응답은 메시지 형태로 이루어진다.
2. 무상태(Stateless): HTTP는 무상태 프로토콜이다. 즉, 각각의 요청은 이전 요청과 독립적으로 처리된다. 서버는 클라이언트의 상태를 기억하지 않으므로, 각각의 요청은 독립적으로 처리된다.
3. 텍스트 기반 프로토콜: HTTP는 텍스트 기반 프로토콜로, 요청과 응답은 ASCII 텍스트로 이루어진다. 이러한 특징으로 인해 사용자가 직접 이해할 수 있고, 디버깅이 용이하다.
4. TCP/IP 기반: HTTP는 TCP/IP 위에서 동작한다. 클라이언트와 서버 간의 안정적인 데이터 전송을 보장하기 위해 TCP를 사용한다.
5. 메서드(Method): HTTP 요청은 여러 가지 메서드를 사용하여 원하는 동작을 지정할 수 있다. 가장 일반적으로 사용되는 메서드는 GET(리소스 요청), POST(리소스 생성), PUT(리소스 업데이트), DELETE(리소스 삭제) 등이 있다.
6. HTTP는 웹 브라우저를 통해 웹 페이지를 요청하고 표시하는 데 사용되지만, 다른 응용 프로그램과 서버 간에도 데이터를 전송하는 데 사용될 수 있다. 예를 들어, REST API와 같은 서비스에서도 HTTP를 통해 데이터를 주고받는다.
7. **4-way 핸드셰이크\***

"4-way 핸드셰이크"는 네트워크 통신에서 사용되는 보안 프로토콜 중 하나인 TLS (Transport Layer Security) 또는 SSL (Secure Sockets Layer)에서 발생하는 과정을 가리킨다. 이 과정은 클라이언트와 서버 간에 안전한 통신을 확립하기 위해 사용되며, 보통은 TLS에서 사용되지만 SSL에서도 비슷한 원리로 동작합니다.

4-way 핸드셰이크가 이루어지는 과정은 다음과 같다:

* 1. 클라이언트는 서버에게 연결을 요청하고, 지원하는 암호화 방식 목록을 전송.
  2. 서버는 클라이언트의 요청을 받고, 암호화된 세션 키를 생성한 다음 이를 클라이언트에게 전송.
  3. 클라이언트는 서버가 보낸 세션 키를 받고, 이를 사용하여 암호화된 데이터를 생성하여 서버에게 전송.
  4. 서버는 클라이언트가 보낸 데이터를 받고, 서버가 생성한 세션 키를 사용하여 이를 복호화.

1. **파싱**

"파싱(Parsing)"은 주어진 데이터를 구문 분석하여 원하는 정보를 추출하는 과정을 말한다. 일반적으로 파싱은 컴퓨터 과학에서 특히 프로그래밍 분야에서 많이 사용되는 용어이다.

파싱은 주로 텍스트나 데이터의 구조를 이해하고 분해하여 원하는 정보를 추출하거나 처리하는 작업에 사용된다. 예를 들어, 다음과 같은 상황에서 파싱이 필요할 수 있다:

* 1. 프로그래밍 언어에서의 파싱: 소스 코드를 컴파일하거나 인터프리터에서 실행하기 위해서는 소스 코드를 해석하고 분해하여 프로그램의 구문을 이해해야 한다. 이를 위해 파서(Parser)가 사용되어 소스 코드를 토큰(Token)으로 분해하고 구문 트리(Syntax Tree)를 생성한다.
  2. 데이터 포맷에서의 파싱: JSON, XML, CSV와 같은 데이터 형식에서 원하는 데이터를 추출하기 위해 파싱이 사용된다. 데이터 포맷의 구조를 이해하고, 해당 구조에 맞게 데이터를 분해하여 필요한 정보를 추출한다.
  3. 웹 페이지에서의 파싱: 웹 스크래핑이나 웹 크롤링과 같은 작업에서는 웹 페이지의 HTML 코드를 파싱하여 원하는 정보를 추출한다. 이를 통해 웹 페이지의 제목, 본문, 링크 등을 수집할 수 있다.
  4. 자연어 처리(Natural Language Processing): 자연어 처리에서는 텍스트 데이터를 파싱하여 문장 구조, 단어의 의미 등을 분석한다. 이를 통해 텍스트에서 정보를 추출하거나 언어 모델을 생성하는 등의 작업을 수행한다.