JavaScript 101 - (12) HTTP와 Restful API

AJAX

AJAX는 **A**synchronous **J**avaScript **A**nd **X**ML의 약자로서, 웹 페이지에서 서버와 비동기적으로 데이터를 주고받을 수 있게 해주는 기법입니다. AJAX를 사용하면 웹 페이지의 일부분만 업데이트할 수 있으므로 전체 페이지를 다시 로드하지 않아도 되고, 사용자 경험을 향상시킬 수 있습니다.

AJAX의 핵심은 XMLHttpRequest 객체입니다. 이 객체는 자바스크립트로 서버에 요청을 보내고 응답을 받을 수 있게 해줍니다. 응답은 XML 형식일 수도 있지만, 텍스트나 JSON 형식일 수도 있습니다.

```
// XMLHttpRequest 객체 생성
var xhr = new XMLHttpRequest();

// 요청 방식과 URL 설정
xhr.open("GET", "[https://jsonplaceholder.typicode.com/users](https://jsonplaceholder.typicode.com/users)");

// 요청 전송
xhr.send();

// 요청 상태에 따른 콜백 함수 등록
xhr.onreadystatechange = function() {
  // 요청이 완료되었고, 응답이 성공적이라면
  if (xhr.readyState === 4 && xhr.status === 200) {
    // 응답 텍스트를 JSON으로 파싱
    var users = JSON.parse(xhr.responseText);
    // 파싱된 배열을 순회하며 이름과 이메일 출력
    for (var i = 0; i < users.length; i++) {
      console.log(users[i].name + ": " + users[i].email);
    }
  }
};
```

miniserver-example과 소통하는 예제

```
// XMLHttpRequest 객체 생성
var xhr = new XMLHttpRequest();

// 뉴스 데이터를 요청하는 함수
function getNews() {
  // 요청 방식과 URL 설정
  xhr.open("GET", "http://localhost:5000/data/dummy-news");

  // 요청 전송
  xhr.send();

  // 요청 상태에 따른 콜백 함수 등록
  xhr.onreadystatechange = function() {
    // 요청이 완료되었고, 응답이 성공적이라면
    if (xhr.readyState === 4 && xhr.status === 200) {
      // 응답 텍스트를 JSON으로 파싱
      var news = JSON.parse(xhr.responseText);
      // 파싱된 객체의 data 속성을 순회하며 제목과 내용 출력
      for (var i = 0; i < news.data.length; i++) {
        console.log(news.data[i].title + ": " + news.data[i].content);
      }
    }
  };
}

// 날씨 데이터를 요청하는 함수
function getWeather() {
  // 요청 방식과 URL 설정
  xhr.open("GET", "http://localhost:5000/data/mini-weather");

  // 요청 전송
  xhr.send();

  // 요청 상태에 따른 콜백 함수 등록
  xhr.onreadystatechange = function() {
    // 요청이 완료되었고, 응답이 성공적이라면
    if (xhr.readyState === 4 && xhr.status === 200) {
      // 응답 텍스트를 JSON으로 파싱
```

```
      var weather = JSON.parse(xhr.responseText);
      // 파싱된 객체의 속성들을 출력
      console.log("날씨: " + weather.status);
      console.log("온도: " + weather.tempature);
      console.log("미세먼지: " + weather.finedust);
    }
  };
}

// 뉴스 데이터와 날씨 데이터를 모두 요청하는 함수
function getAllData() {
  getNews();
  getWeather();
}

// getAllData 함수 호출하여 AJAX 요청 보내기
getAllData();
```

HTTP

- HTTP(HyperText Transfer Protocol)는 웹에서 클라이언트와 서버가 데이터를 주고받기 위한 통신 규약입니다.
- HTTP는 요청(Request)과 응답(Response)으로 구성되며, 각각에는 메서드(Method), 헤더(Header), 바디(Body) 등의 요소가 있습니다.
- HTTP는 상태가 없는(stateless) 프로토콜이므로, 각 요청과 응답은 독립적이며 서로 연관성이 없습니다. 따라서 상태 정보를 유지하기 위해서는 쿠키(Cookie), 세션(Session), 토큰(Token) 등의 방법을 사용합니다.

REST

- REST(Representational State Transfer)는 웹에서 사용되는 모든 데이터나 자원(Resource)을 HTTP URI로 표현하고, 표준 HTTP 메서드(GET, POST, PUT, DELETE 등)를 이용하여 자원에 대한 CRUD(Create, Read, Update, Delete) 연산을 수행하는 아키텍처 스타일입니다.
- REST는 로이 필딩(Roy Fielding)이 2000년에 박사학위 논문에서 제안한 개념으로, 웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 소프트웨어 아키텍처 원리입니다.
- REST는 다음과 같은 제약 조건을 가집니다.
  - 클라이언트-서버(Client-Server): 클라이언트와 서버가 분리되어 서로의 역할을 명확하게 구분합니다.
  - 무상태성(Stateless): 각 요청과 응답은 독립적이며 서로 연관성이 없습니다. 즉, 서버는 클라이언트의 상태 정보를 저장하지 않고 매번 새롭게 처리합니다.
  - 캐시 가능(Cacheable): 클라이언트는 서버의 응답을 캐시할 수 있으며, 캐시된 응답은 유효기간 내에 재사용할 수 있습니다. 이를 통해 네트워크 부하를 줄일 수 있습니다.
  - 계층화(Layered System): 클라이언트와 서버 사이에 여러 계층(프록시, 게이트웨이 등)을 두어 시스템의 구조를 단순화하고 확장성을 높일 수 있습니다. 단, 클라이언트는 중간 계층에 대한 지식 없이도 통신할 수 있어야 합니다.
  - 코드 온 디맨드(Code on Demand): 선택적으로 사용할 수 있는 제약 조건으로서, 서버가 실행 가능한 코드(자바스크립트 등)를 클라이언트에 전송하여 기능을 확장할 수 있습니다.
  - 자체 표현적(Self-Descriptive): 요청과 응답은 메시지가 스스로 설명되어야 한다는 것입니다. 메시지에는 상태가 없으므로 그 메시지만 보고서 뜻을 파악할 수 있어야 합니다.

REST API

- REST API는 REST 아키텍처 스타일을 따르는 웹 서비스의 API(Application Programming Interface)입니다.
- REST API는 자원의 식별자(URI), 자원에 대한 행위(HTTP 메서드), 자원에 대한 표현(페이로드 포맷)으로 구성됩니다.
- REST API는 다음과 같은 특징을 가집니다.
  - 유니폼 인터페이스(Uniform Interface): 요청과 응답이 일관된 형식을 갖도록 합니다. 예를 들어, URI는 명사로 표현하고, HTTP 메서드는 동사로 표현합니다.
  - 주소 지정 가능(Addressable): 각 자원은 고유한 URI를 가지고 있으며, 해당 URI로 접근할 수 있습니다. 예를 들어, /users/1은 id가 1인 사용자를 나타냅니다.
  - 연결성(Connectivity): 하나의 자원이 다른 자원과 연결될 수 있습니다. 예를 들어, /users/1/orders는 id가 1인 사용자의 주문 목록을 나타냅니다.

    - 상태 전이(State Transition): 클라이언트는 서버의 상태를 변경하기 위해 요청을 보내고, 서버는 응답으로 적절한 상태 코드와 메시지를 전달합니다. 예를 들어, POST /users는 새로운 사용자를 생성하고, 201 Created와 Location 헤더를 응답합니다.

HTTP 메서드

- HTTP 메서드는 클라이언트가 서버에게 요청하는 작업의 종류를 나타내는 단어입니다.
- HTTP 메서드에는 GET, POST, PUT, PATCH, DELETE 등이 있으며 각각 다음과 같은 의미와 용도를 가집니다.
    - GET: 서버에게 자원의 조회를 요청합니다. GET 요청은 서버의 상태나 데이터베이스에 영향을 주지 않아야 합니다(idempotent). 예를 들어, GET /users/1은 id가 1인 사용자의 정보를 조회합니다.
    - POST: 서버에게 자원의 생성을 요청합니다. POST 요청은 서버의 상태나 데이터베이스에 영향을 줍니다(non-idempotent). 예를 들어, POST /users는 새로운 사용자를 생성합니다.
    - PUT(PATCH): 서버에게 자원의 전체 수정을 요청합니다. PUT 요청은 동일한 요청을 여러 번 보내도 결과가 같아야 합니다(idempotent). 예를 들어, PUT /users/1은 id가 1인 사용자의 정보를 전체 수정합니다.
    - PATCH: 서버에게 자원의 부분 수정을 요청합니다. PATCH 요청은 동일한 요청을 여러 번 보내면 결과가 달라질 수 있습니다(non-idempotent). 예를 들어, PATCH /users/1은 id가 1인 사용자의 정보 중 일부를 수정합니다.
    - DELETE: 서버에게 자원의 삭제를 요청합니다. DELETE 요청은 서버의 상태나 데이터베이스에 영향을 줍니다(non-idempotent). 예를 들어, DELETE /users/1은 id가 1인 사용자를 삭제합니다.

HTTP 상태 코드

- HTTP 상태 코드는 서버가 클라이언트에게 응답할 때 전달하는 세 자리 숫자로서, 요청의 처리 결과를 나타냅니다.
- HTTP 상태 코드에는 다음과 같은 범주가 있으며 각각 다른 의미와 용도를 가집니다.
    - 1xx(Informational): 요청이 수신되었으며 처리 중임을 나타냅니다. 예를 들어, 100 Continue는 클라이언트가 바디를 계속 전송해도 된다는 것을 의미합니다.
    - 2xx(Success): 요청이 성공적으로 처리되었음을 나타냅니다. 예를 들어, 200 OK는 정상적으로 처리되었다는 것을 의미하고, 201 Created는 새로운 자원이 생성되었다는 것을 의미합니다.
    - 3xx(Redirection): 요청을 완료하기 위해 추가 작업이 필요함을 나타냅니다. 예를 들어, 301 Moved Permanently는 자원의 URI가 영구적으로 변경되었다는 것을 의미하고, 302 Found는 자원의 URI가 일시적으로 변경되었다는 것을 의미합니다.
    - 4xx(Client Error): 클라이언트의 요청이 잘못되었음을 나타냅니다. 예를 들어, 400 Bad Request는 잘못된 문법으로 인해 서버가 요청을 이해할 수 없다는 것을 의미하고, 401 Unauthorized는 인증이 필요하거나 실패했다는 것을 의미합니다.
    - 5xx(Server Error): 서버가 정상적인 요청에 대해 처리할 수 없음을 나타냅니다. 예를 들어, 500 Internal Server Error는 서버에 내부 오류가 발생했다는 것을 의미하고, 503 Service Unavailable은 서버가 일시적으로 요청을 처리할 수 없다는 것을 의미합니다.

HTTP 헤더와 바디

- HTTP 헤더는 요청과 응답에 대한 추가적인 정보를 나타내는 키-값 쌍의 집합입니다.
- HTTP 헤더에는 다음과 같은 종류가 있으며 각각 다른 의미와 용도를 가집니다.
    - 일반 헤더(General Header): 요청과 응답 모두에 적용되는 정보를 나타냅니다. 예를 들어, Date는 메시지가 생성된 날짜와 시간을 나타냅니다.
    - 요청 헤더(Request Header): 요청에 대한 정보를 나타냅니다. 예를 들어, Host는 요청이 전송되는 서버의 도메인 이름을 나타냅니다.
    - 응답 헤더(Response Header): 응답에 대한 정보를 나타냅니다. 예를 들어, Location은 리다이렉션할 URI를 나타냅니다.
    - 엔티티 헤더(Entity Header): 바디에 포함된 자원에 대한 정보를 나타냅니다. 예를 들어, Content-Type은 바디의 미디어 타입을 나타냅니다.
    - 확장 헤더(Extension Header): 사용자 정의의 정보를 나타냅니다. 예를 들어, X-Requested-With는 Ajax 요청임을 나타내기 위해 사용됩니다.
- HTTP 바디는 요청과 응답에 실제로 전달되는 데이터입니다.
- HTTP 바디의 형식은 Content-Type 헤더에 따라 달라질 수 있으며, 일반적으로 REST API에서는 JSON(JavaScript Object Notation)이나 XML(eXtensible Markup Language) 등의 구조화된 데이터 포맷을 사용합니다.

JSON

- JSON(JavaScript Object Notation)은 자바스크립트의 객체 표기법을 기반으로 한 경량의 데이터 교환 형식입니다.
- JSON은 키-값 쌍의 집합으로 구성되며, 키는 문자열이고 값은 문자열, 숫자, 불리언, 배열, 객체, null 중 하나입니다.
- JSON은 다른 프로그래밍 언어에서도 쉽게 파싱할 수 있으며, REST API에서 주로 사용되는 페이로드 포맷입니다.
- 예를 들어, 다음은 JSON 형식의 데이터입니다.

```
{
  "name": "Alice",
  "age": 25,
  "hobbies": ["reading", "cooking", "traveling"],
  "address": {
    "city": "Seoul",
    "country": "Korea"
  },
  "married": false
}
```