* 예습
1. 6/6

43. Ajax

TABLE OF CONTENTS

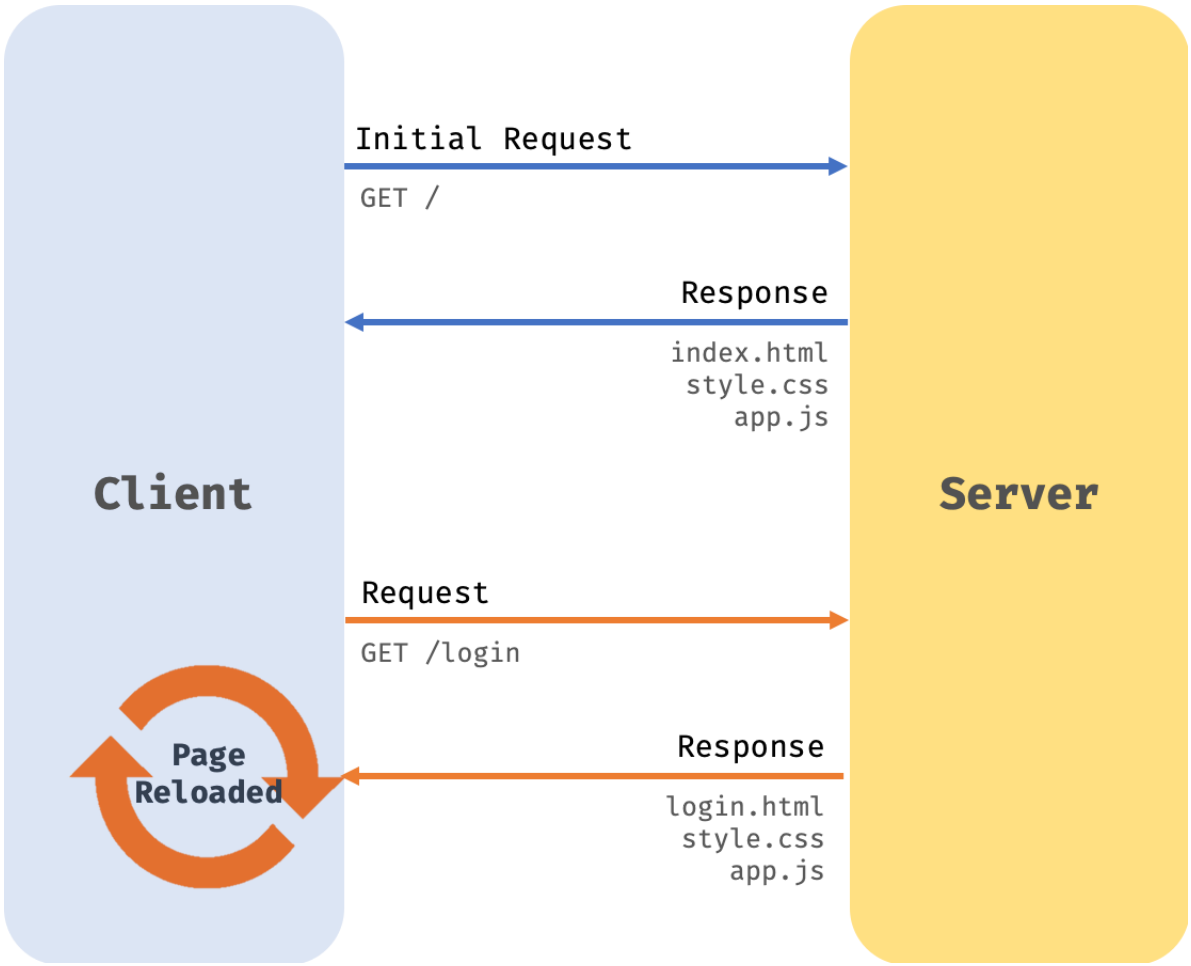
#1. Ajax란?

데이터를 교환(통신)하고 서버로부터 수신한 데이터를 기반으로 웹 페이지를 동적으로 갱신하는 프로그래밍 방식을 의미한다. Ajax는 Web API인 XMLHttpRequest 객체를 기반으로 동작한다.
XMLHttpRequest는 서버와 브라우저 간의 비동기적(Asynchronous) 데이터 통신을 가능케하는 여러 메소드와 프로퍼티를 메소드를 제공한다.

1999년 마이크로소프트가 개발한 XMLHttpRequest은 그다지 큰 주목을 받지 못하다가 2005년 구글이 발표한 Google Maps을 통해 웹 애플리케이션 개발 프로그래밍 언어로서 자바스크립트의 가능성을 확인하는 계기를 마련했다. 웹 브라우저에서 자바스크립트와 Ajax를 기반으로 동작하는 Google Maps

가 데스크톱 애플리케이션과 비교해 손색이 없을 정도의 퍼포먼스와 부드러운 화면 전환 효과를 보여준 것이다.

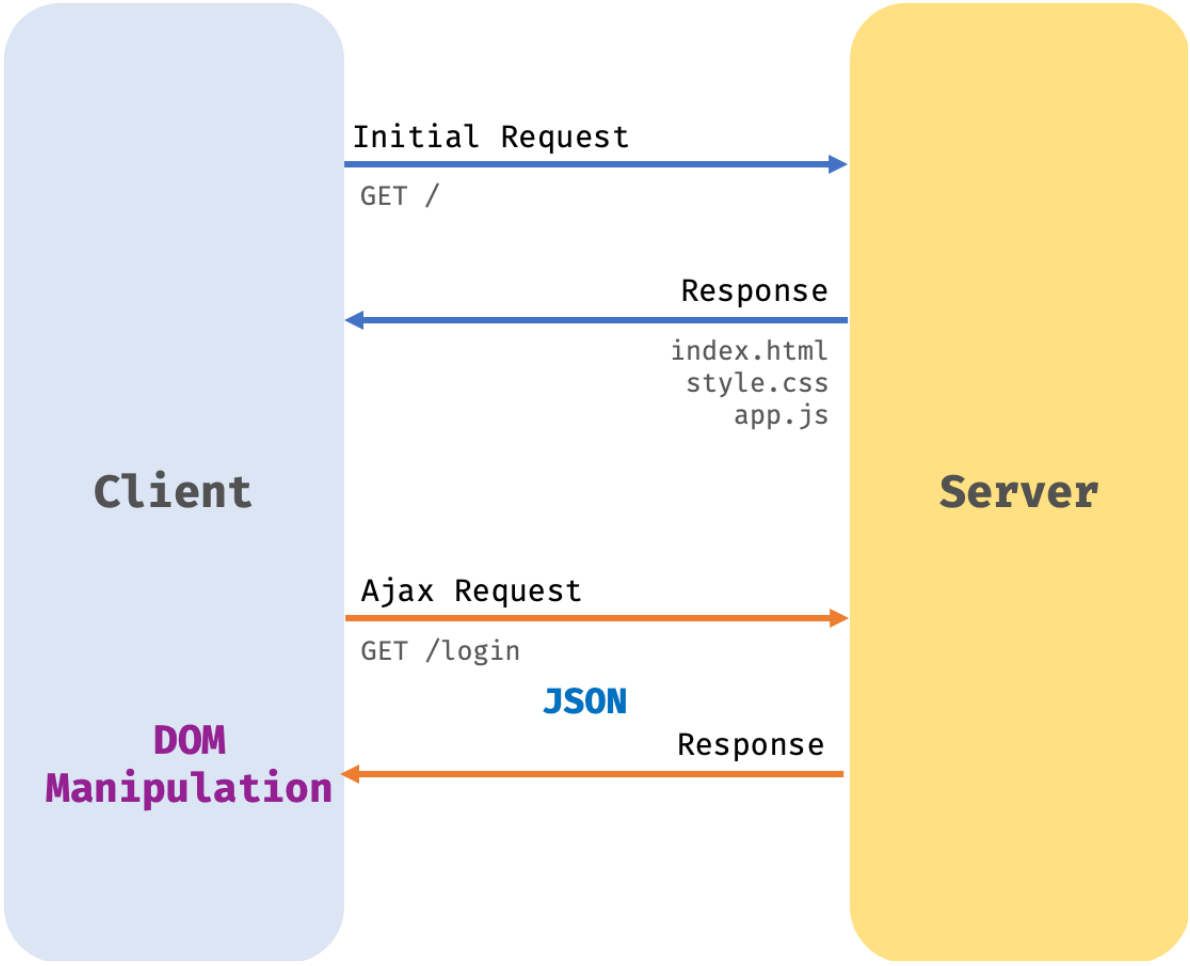
이전의 웹 페이지는 서버로부터 완전한 HTML을 전송 받아 웹 페이지 전체를 처음부터 다시 렌더링하는 방식으로 동작했다. 따라서 화면이 전환되면 서버로부터 새로운 HTML을 전송 받아 웹 페이지 전체를 처음부터 다시 렌더링하였다.



전통적인 웹 페이지의 생명 주기

이러한 전통적인 방식은 아래와 같은 단점이 있다.

1. 변경이 없는 부분까지 포함된 HTML을 서버로부터 매번 다시 전송 받기 때문에 불필요한 데이터 통신이 발생한다.
2. 변경이 필요 없는 부분까지 처음부터 다시 렌더링해야 한다. 이로 인해 화면 전환이 일어나면 화면이 순간적으로 깜박이는 현상이 발생한다.
3. 클라이언트와 서버와의 통신이 동기 방식으로 동작하기 때문에 서버로부터 응답이 있을 때까지 다음 처리는 블로킹된다. Ajax의 등장은 이전의 패러다임을 획기적으로 전환했다. 즉, 서버로부터 웹 페이지의 변경에 필요한 데이터만을 비동기 방식으로 전송 받아 웹 페이지의 변경이 필요 없는 부분은 다시 렌더링하지 않고, 변경이 필요한 부분만을 한정적으로 렌더링하는 방식이 가능해진 것이다. 이로 인해 웹 브라우저에서도 데스크톱 애플리케이션과 유사한 빠른 퍼포먼스와 부드러운 화면 전환이 가능케 되었다.



Ajax

Ajax는 전통적인 웹 페이지 방식과 비교했을 때 아래와 같은 장점이 있다.

1. 변경이 필요한 부분만을 갱신하기 위한 데이터 만을 전송 받기 때문에 불필요한 데이터 통신이 발생하지 않는다.
2. 변경이 필요 없는 부분은 다시 렌더링하지 않는다. 따라서 화면이 순간적으로 깜박이는 현상이 발생하지 않는다.
3. 클라이언트와 서버와의 통신이 비동기 방식으로 동작하기 때문에 서버에게 요청을 보낸 이후, 다른 처리를 계속해서 수행할 수 있다.

2. JSON

JSON(JavaScript Object Notation)은 클라이언트와 서버 간의 통신을 위한 텍스트 데이터 포맷이다. 자바스크립트에 종속되지 않는 언어 독립형 데이터 포맷으로 대부분의 프로그래밍 언어에서 사용할 수 있다.

2.1. JSON 표기 방식

JSON은 자바스크립트의 객체 리터럴과 유사하게 키와 값으로 구성된 순수한 텍스트다.

JSON

```
{
  "name": "Lee",
  "age": 20,
  "alive": true,
  "hobby": ["traveling", "tennis"]
}
```

JSON의 키는 반드시 큰따옴표(작은 따옴표 사용불가)로 묶어야 한다. 값은 객체 리터럴과 같은 표기법을 그대로 사용할 수 있다. 하지만 문자열은 반드시 큰따옴표(작은 따옴표 사용불가) 로 묶어야 한다.

2.2. JSON.stringify

JSON.stringify 메소드는 객체를 JSON 포맷의 문자열로 변환한다. 클라이언트가 서버로 객체를 전송하려면 객체를 문자열화하여야 하는데 이를 직렬화(Serializing)이라 한다. 즉, 직렬화는 객체를 전송 가능한 형태로 변형하는 것을 말한다.

JAVASCRIPT

```
const obj = {
  name: 'Lee',
  age: 20,
  alive: true,
  hobby: ['traveling', 'tennis']
};

// 객체 => JSON
const json = JSON.stringify(obj);
console.log(typeof json, json);
// string {"name":"Lee","age":20,"alive":true,"hobby":["traveling","tenni
s"]}
```

```
// 객체 => JSON 형식의 문자열 + prettify
const prettyJson = JSON.stringify(obj, null, 2);
console.log(typeof prettyJson, prettyJson);
/*
string {
  "name": "Lee",
  "age": 20,
  "alive": true,
  "hobby": [
    "traveling",
    "tennis"
  ]
}
*/

// replacer
// 값의 타입이 Number이면 필터링되어 반환되지 않는다.
function filter(key, value) {
  // undefined: 반환하지 않음
  return typeof value === 'number' ? undefined : value;
}

// 객체 => JSON 형식의 문자열 + replacer + prettify
const strFilteredObject = JSON.stringify(obj, filter, 2);
console.log(typeof strFilteredObject, strFilteredObject);
/*
string {
  "name": "Lee",
  "alive": true,
  "hobby": [
    "traveling",
    "tennis"
  ]
}
*/
```

JSON.stringify 메소드는 객체 뿐만이 아니라 배열도 JSON 포맷의 문자열로 변환한다.

JAVASCRIPT

```
const todos = [
  { id: 1, content: 'HTML', completed: false },
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
];

// 배열 => JSON
const json = JSON.stringify(todos, null, 2);
console.log(typeof json, json); // string [1,5,"false"]
/*
string [
  {
    "id": 1,
    "content": "HTML",
    "completed": false
  },
  {
    "id": 2,
    "content": "CSS",
    "completed": true
  },
  {
    "id": 3,
    "content": "Javascript",
    "completed": false
  }
]
*/
```

2.3. JSON.parse

JSON.parse 메소드는 JSON 포맷의 문자열을 객체로 변환한다. 서버로부터 클라이언트에게 전송된 JSON 데이터는 문자열이다. 이 문자열을 객체로서 사용하려면 JSON 포맷의 문자열을 객체화하여야 하는데 이를 역직렬화(Deserializing)이라 한다. 즉, 역직렬화는 전송된 문자열 등을 다시 객체로 복원하는 것을 말한다.

JAVASCRIPT

```
const obj = {
  name: 'Lee',
  age: 20,
  alive: true,
  hobby: ['traveling', 'tennis']
};

// 객체 => JSON
const json = JSON.stringify(obj);

// JSON ⇒ 객체
const parsed = JSON.parse(json);
console.log(typeof parsed, parsed);
// object {name: "Lee", age: 20, alive: true, hobby: ["traveling", "tenni
s"]}
```

배열이 JSON 포맷의 문자열로 변환되어 있는 경우, JSON.parse는 문자열을 배열 객체로 변환한다. 배열의 요소가 객체인 경우, 배열의 요소까지 객체로 변환한다.

JAVASCRIPT

```
const todos = [
  { id: 1, content: 'HTML', completed: false },
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
];

// 객체 => JSON
const json = JSON.stringify(todos);

// JSON ⇒ 객체
const parsed = JSON.parse(json);
console.log(typeof parsed, parsed);
/*
 object [
  { id: 1, content: 'HTML', completed: false },
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
```

```
]
*/
```

3. XMLHttpRequest

브라우저는 주소창이나 HTML의 form 태그 또는 a 태그를 통해 HTTP 요청 전송 기능을 기본 제공한다. 자바스크립트를 사용하여 HTTP 요청을 전송하려면 XMLHttpRequest 객체를 사용한다. Web API인 XMLHttpRequest 객체는 HTTP 요청 전송과 HTTP 응답 수신을 위한 다양한 메소드와 프로퍼티를 제공한다.

3.1. XMLHttpRequest 객체 생성

XMLHttpRequest 객체는 XMLHttpRequest 생성자 함수를 호출하여 생성한다.

JAVASCRIPT

```
// XMLHttpRequest 객체의 생성
const xhr = new XMLHttpRequest();
```

XMLHttpRequest 객체는 다양한 프로퍼티와 메소드를 제공한다. 대표적인 프로퍼티와 메소드는 아래와 같다.

프로토타입 프로퍼티	설명
readyState	요청의 현재 상태를 나타내는 정수. 이하의 XMLHttpRequest의 정적 프로퍼티를 값으로 갖는다. UNSENT: 0 OPENED: 1 HEADERS_RECEIVED: 2 LOADING: 3 DONE: 4

프로토타입 프로퍼티	설명
status	요청에 대한 응답 상태(HTTP 상태 코드)를 나타내는 정수 예) 200
statusText	요청에 대한 응답 메시지를 나타내는 문자열 예) “OK”
responseType	응답 타입 예) document, json, text, blob, arraybuffer
response	요청에 대한 응답 몸체(response body). responseType에 따라 타입이 다르다.
responseText	서버가 전송한 요청에 대한 응답 문자열

이벤트 핸들러 프로퍼티	설명
onreadystatechange	readyState 프로퍼티 값이 변경된 경우
onloadstart	요청에 대한 응답을 받기 시작한 경우
onprogress	요청에 대한 응답을 받는 도중 주기적으로 발생
onabort	abort 메소드에 의해 요청이 중단되었을 경우
onerror	요청에 에러가 발생한 경우
onload	요청이 성공적으로 완료한 경우
ontimeout	요청 시간이 초과한 경우
onloadend	요청이 완료한 경우. 요청이 성공 또는 실패하면 발생

메소드	설명
open	HTTP 요청 초기화
send	HTTP 요청 전송
abort	이미 전송된 HTTP 요청 중단
setRequestHeader	HTTP 요청 헤더의 값을 설정
getResponseHeader	지정한 HTTP 요청 헤더의 값을 문자열로 반환

정적 프로퍼티	값	설명
UNSENT	0	open 메소드 호출 이전

정적 프로퍼티	값	설명
OPENED	1	open 메소드 호출 이후
HEADERS_RECEIVED	2	send 메소드 호출 이후
LOADING	3	서버 응답 중(응답 데이터 미완성 상태)
DONE	4	서버 응답 완료

3.2. HTTP 요청 전송

HTTP 요청을 전송하는 경우, 아래의 순서를 따른다.

1. XMLHttpRequest.prototype.open 메소드로 HTTP 요청 초기화
2. 필요에 따라 XMLHttpRequest.prototype.setRequestHeader 메소드로 HTTP 요청의 헤더 값 설정
3. XMLHttpRequest.prototype.send 메소드로 HTTP 요청 전송

JAVASCRIPT

```
// XMLHttpRequest 객체의 생성
const xhr = new XMLHttpRequest();

// HTTP 요청 초기화
xhr.open('GET', '/users');

// HTTP 요청 헤더 설정
// 클라이언트가 서버로 전송할 데이터의 MIME-type 지정: json
xhr.setRequestHeader('content-type', 'application/json');

// HTTP 요청 전송
xhr.send();
```

XMLHttpRequest.prototype.open

open 메소드는 서버에게 전송할 HTTP 요청을 초기화한다. open 메소드의 호출 방법은 아래와 같다.

JAVASCRIPT

```
xhr.open(method, url[, async])
```

매개 변수	설명
method	HTTP 요청 메소드 (“GET”, “POST”, “PUT”, “DELETE” 등)
url	HTTP 요청을 전송할 URL
async	비동기 요청 여부. 옵션으로 기본값은 true이며 비동기 방식으로 동작한다.

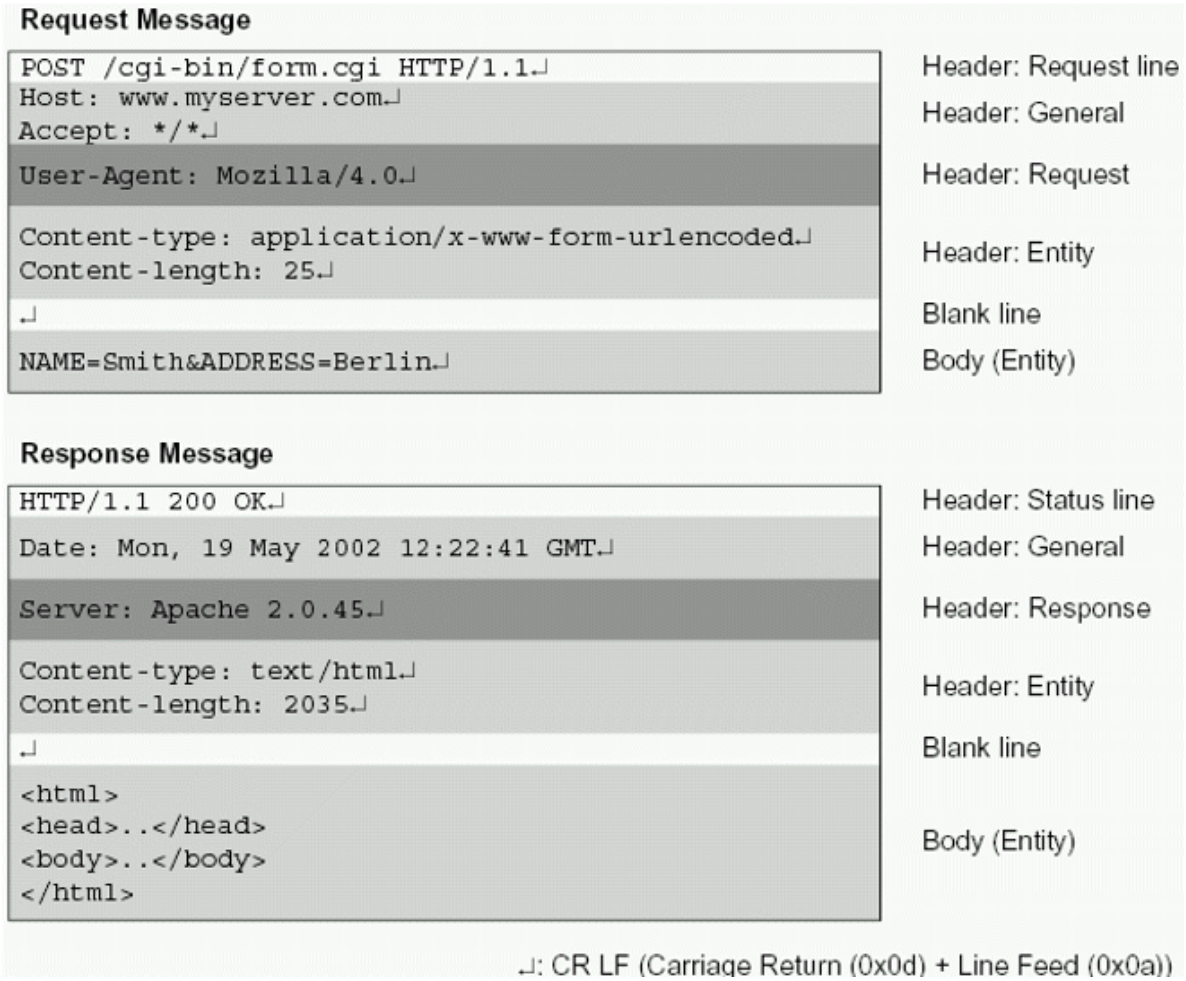
HTTP 요청 메소드는 클라이언트가 서버에게 요청의 종류와 목적(리소스에 대한 행위)을 알리는 방법이다. 주로 5가지의 요청 메소드(GET, POST, PUT, PATCH, DELETE 등)를 사용하여 CRUD를 구현한다.

HTTP 요청 메소드	종류	목적	페이로드
GET	index/retrieve	모든/특정 리소스 취득	x
POST	create	리소스 생성	○
PUT	replace	리소스의 전체 교체	○
PATCH	modify	리소스의 일부 수정	○
DELETE	delete	모든/특정 리소스 삭제	x

XMLHttpRequest.prototype.send

send 메소드는 open 메소드로 초기화된 HTTP 요청을 서버에 전송한다. 기본적으로 서버로 전송하는 데이터는 GET, POST 요청 메소드에 따라 그 전송 방식에 차이가 있다.

- GET 요청 메소드의 경우, 데이터를 URL의 일부분인 쿼리 문자열(query string)로 서버로 전송한다.
- POST 요청 메소드의 경우, 데이터(페이로드)를 요청 몸체(request body)에 담아 전송한다.



HTTP 요청/응답 메시지

send 메소드에는 요청 몸체(request body)에 담아 전송할 데이터(페이로드)를 인수로 전달할 수 있다. 페이로드가 객체인 경우, 반드시 JSON.stringify 메소드를 사용하여 직렬화한 다음, 전달해야 한다.

JAVASCRIPT

```
xhr.send(JSON.stringify([
  { id: 1, content: 'HTML', completed: false },
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
]));
```

만약 HTTP 요청 메소드가 GET인 경우, send 메소드에 페이로드로 전달한 인수는 무시되고 요청 몸체는 null로 설정된다.

XMLHttpRequest.prototype.setRequestHeader

setRequestHeader 메소드는 HTTP 요청의 헤더 값을 설정한다. setRequestHeader 메소드는 반드시 open 메소드 호출 이후에 호출해야 한다. 자주 사용하는 HTTP 요청 헤더인 Content-type과

Accept에 대해 살펴보자.

Content-type은 요청 몸체(request body)에 담아 전송할 데이터의 MIME-type의 정보를 표현한다. 자주 사용되는 MIME-type은 아래와 같다.

MIME-type	서브타입
text	text/plain, text/html, text/css, text/javascript
application	application/json, application/x-www-form-urlencode
multipart	multipart/formed-data

다음은 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정하는 예이다.

JAVASCRIPT

```
// XMLHttpRequest 객체의 생성
const xhr = new XMLHttpRequest();

// HTTP 요청 초기화
xhr.open('POST', '/users');

// HTTP 요청 헤더 설정
// 클라이언트가 서버로 전송할 데이터의 MIME-type 지정: json
xhr.setRequestHeader('content-type', 'application/json');

// HTTP 요청 전송
xhr.send(JSON.stringify([
  { id: 1, content: 'HTML', completed: false },
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
]));
```

HTTP 클라이언트가 서버에 요청할 때 서버가 센드백할 데이터의 MIME-type을 Accept로 지정할 수 있다. 다음은 서버가 센드백할 데이터의 MIME-type을 지정하는 예이다.

JAVASCRIPT

```
// 서버가 센드백할 데이터의 MIME-type 지정: json
xhr.setRequestHeader('accept', 'application/json');
```

만약 Accept 헤더를 설정하지 않으면, send 메소드가 호출될 때 Accept 헤더가 `*/*` 으로 전송된다.

3.3. HTTP 응답 처리

서버가 전송한 응답을 처리하려면 XMLHttpRequest 객체가 발생시키는 이벤트를 캐치하여야 한다. “43.3.1. XMLHttpRequest 객체 생성”에서 살펴본 바와 같이 XMLHttpRequest 객체는 이벤트 핸들러 프로퍼티를 갖는다. 이 이벤트 핸들러 프로퍼티 중에서 readyState 프로퍼티 값이 변경된 경우, 발생하는 readystatechange 이벤트를 캐치하여 아래와 같이 HTTP 응답을 처리할 수 있다.

JAVASCRIPT

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

// HTTP 요청 초기화
xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1');

// HTTP 요청 전송
xhr.send();

// readystatechange 이벤트는 요청의 현재 상태를 나타내는 readyState 프로퍼티가 변경될
 때마다 발생
xhr.onreadystatechange = () => {
  // readyState: 4 ⇒ DONE(서버 응답 완료)
  if (xhr.readyState !== XMLHttpRequest.DONE) return;

  // status는 response 상태 코드를 반환 : 200 => 정상 응답
  if (xhr.status === 200) {
    console.log(JSON.parse(xhr.response));
    // {userId: 1, id: 1, title: "delectus aut autem", completed: false}
  } else {
    console.error('Error', xhr.status, xhr.statusText);
  }
};
```

send 메소드를 통해 서버에 HTTP 요청을 전송하면 서버는 응답을 반환한다. 하지만 언제 응답이 클라이언트에 도달할 지는 알 수 없다. 따라서 readystatechange 이벤트를 통해 HTTP 요청의 현재 상태를

확인해야 한다. readystatechange 이벤트는 요청의 현재 상태를 나타내는 readyState 프로퍼티가 변경될 때마다 발생한다.(“43.3.1. XMLHttpRequest 객체 생성”의 정적 프로퍼티 참고)

onreadystatechange 이벤트 핸들러 프로퍼티에 할당한 이벤트 핸들러는 xhr.readyState가 XMLHttpRequest.DONE인지 확인하여 서버의 응답이 완료되었는지 확인한다.

서버의 응답이 완료되었다면 요청에 대한 응답 상태(HTTP 상태 코드)를 나타내는 xhr.status가 200인지 확인하여 정상 처리와 에러 처리를 구분한다. 정상적으로 요청에 대한 응답이 도착했다면 요청에 대한 응답 몸체(response body)를 나타내는 xhr.response에서 서버가 전송한 데이터를 취득한다.

readystatechange 이벤트 대신 load 이벤트를 캐치해도 좋다. load 이벤트는 요청이 성공적으로 완료된 경우 발생한다. 따라서 load 이벤트를 캐치하는 경우, xhr.readyState가 XMLHttpRequest.DONE인지 확인할 필요가 없다

JAVASCRIPT

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

// HTTP 요청 초기화
xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1');

// HTTP 요청 전송
xhr.send();

// load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
xhr.onload = () => {
  // status는 response 상태 코드를 반환 : 200 => 정상 응답
  if (xhr.status === 200) {
    console.log(JSON.parse(xhr.response));
    // {userId: 1, id: 1, title: "delectus aut autem", completed: false}
  } else {
    console.error('Error', xhr.status, xhr.statusText);
  }
};
```