6장 AJAX

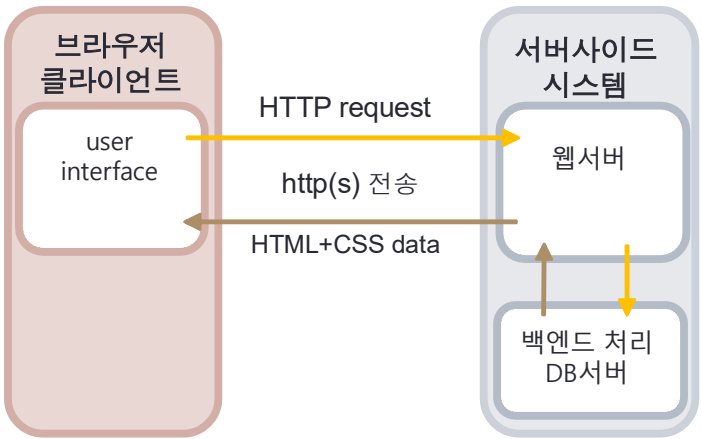
1. AJAX 기초
2. REST(Representational Safe Transfer)
3. XMLHttpRequest 객체
4. JSON 요청과 응답
5. 더 간편한 프로미스(Promise) 구현을 위한 패치(fetch)
6. HTTP 응답 상태 코드 표(Response Status Code)
7. 타이머를 이용한 지연 실행과 반복 실행

AJAX란

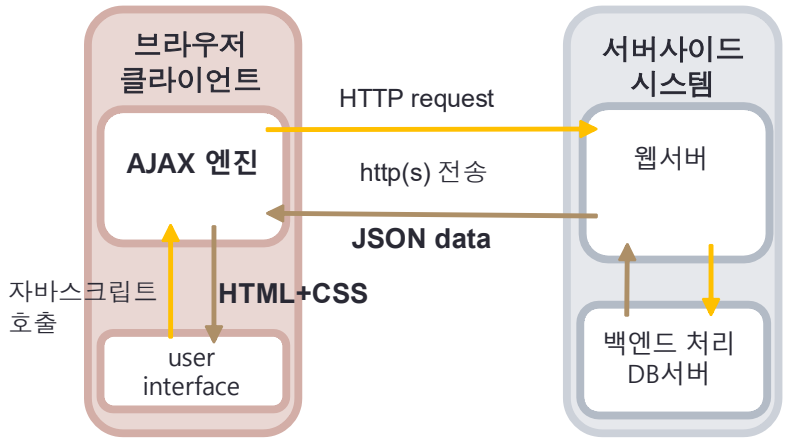
- Asynchronous JavaScript and XML(비동기방식의 자바스크립트와 XML)
- 비동기 방식으로 데이터를 주고받기 위해 개발된 자바스크립트 기술로서 적은 양의 데이터만 받아서 웹페이지의 일부만을 갱신하는 방법
- XMLHttpRequest 객체는 자바스크립트 내장 객체로서 비동기 통신 구현을 위한 객체
- AJAX 사용 사이트
  - http://map.google.com 의 지도
  - http://www.naver.com 의 검색어 입력 기능

기존 방식과 AJAX 방식 비교

- 전통적인 웹애플리케이션 방식
  - 서버는 화면에 필요한 모든 데이터(HTML페이지)를 만들어서 브라우저에 전송
  - 브라우저는 단순 뷰어 역할
- AJAX 웹애플리케이션 방식
  - 모바일 환경이 대두되면서 서버의 역할이 달라짐
  - 서버는 브라우저나 모바일에서 필요한 순수 데이터(JSON)만 전달하는 API서버의 형태로 변화
  - 브라우저나 모바일에서 전달받은 데이터를 가공해서 사용자에게 보여줌



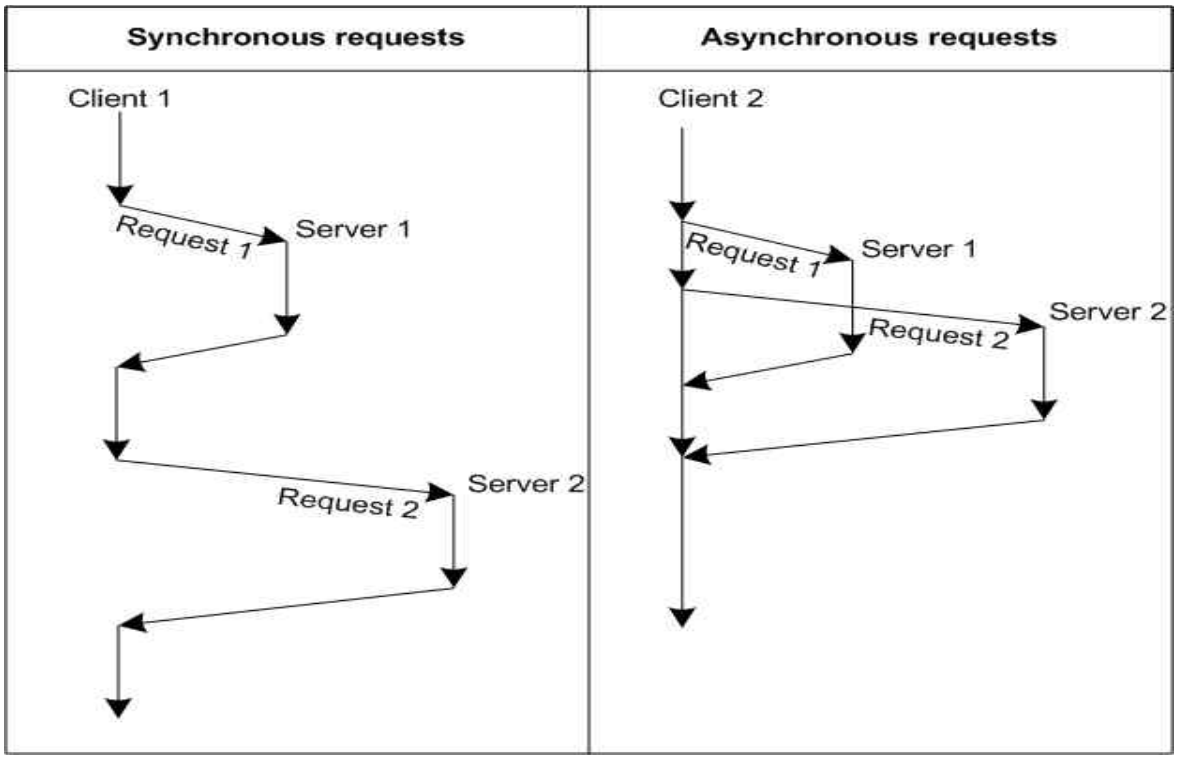
전통적인 웹 애플리케이션 모델



AJAX 웹 애플리케이션 모델

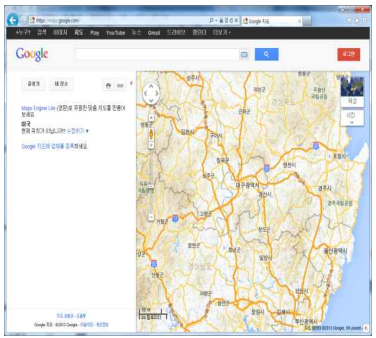
동기와 비동기식 요청 비교

동기식 비동기식



AJAX를 이루는 구성요소

클라이언트(웹브라우저)

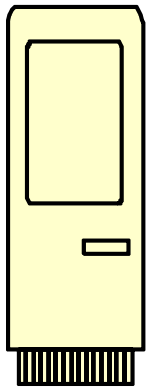


3. 콜백함수 호출
DOM API를 이용하여
페이지 일부분만 수정

1. 비동기 요청
XMLHttpRequest

2. 응답결과
String(csv)
XML,
JSON

서버



REST(Representational Safe Transfer)

- 모바일과 같은 다양한 클라이언트의 등장하면서 Backend 하나로 다양한 Device를 대응하기 위해 REST의 필요성이 증대
- 하나의 URI는 하나의 고유한 리소스(자원)을 대표하도록 설계하고 전송방식(Method)을 결합해서 원하는 작업을 지정
  - POST(등록) ← http://localhost/users
  - GET(조회) ← http://localhost/users/choi
  - PUT(수정) ← http://localhost/users
  - DELETE(삭제) ← http://localhost/users/choi

기존의 웹 접근 방식과 REST API 방식과의 차이점

▶ REST 요청

HTTP Method	REST URI
POST	/users
GET	/users
GET	/users/hong
PUT	/users/hong
DELETE	/users/hong

REST 요청방식은 4가지 메소드를 모두 사용하여 CRUD를 처리하며 URI는 제어하려는 자원을 나타냄

Action
Create(insert)
Read(select)
read
Update
Delete

▶ 기존의 요청

HTTP Method	URI
POST	/insertUser.do
GET	/getUserList.do
GET	/getUser.do?id=hong
POST	/updateUser.do
GET	/deleteUser.do?id=hong

기존의 요청방식은 GET과 POST만으로 CRUD를 처리하며, URI는 액션을 나타냄

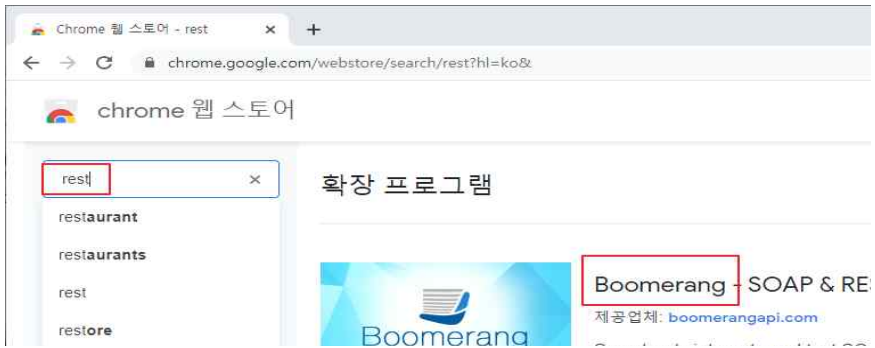
REST 서버

- 공공 데이터 포털
  - https://www.data.go.kr/
- 영화진흥위원회
  - https://www.kobis.or.kr/kobisopenapi/homepg/apiservice/searchServiceInfo.do
- Fake Online REST server
  - https://jsonplaceholder.typicode.com/
- JSON-SERVER
  - https://www.npmjs.com/package/json-server
- Yedam REST 서버(spring framework)
  - https://github.com/cyannara/ajaxserver.git

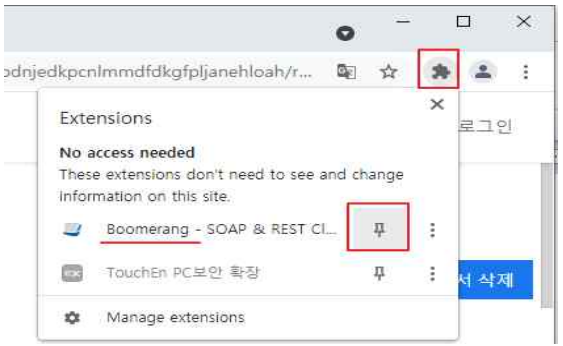
REST 클라이언트 도구 rest를 테스트하는 도구

- 개발된 REST API를 테스트할 수 있고, 팀원들간 공유를 할 수 있게 해주는 플랫폼
- 부메랑 앱 설치
  - 크롬웹스토어: https://chrome.google.com/webstore/category/extensions?hl=ko

▶ rest 검색해서 부메랑 앱 선택



▶ 핀고정



AJAX 예제 (xhrTest.html)

```
<button type="button" onclick="loadDoc()">ajax 요청</button>

<script>
  function loadDoc() {
    const ajax = new XMLHttpRequest();                              //1. XMLHttpRequest 객체 생성
    const url = "https://jsonplaceholder.typicode.com/todos/1"; //2. 접속할 서버 URL
    ajax.onload = function () {                                     //3. 응답을 처리하는 콜백 함수
      if (ajax.status >= 200 && ajax.status < 300) {
        successCallback(ajax.response);
      } else {
        errorCallback(new Error(ajax.statusText));
      }
    };
    ajax.onerror = errorCallback;                                   //4. 에러 이벤트 발생시 처리할 에러 콜백 함수
    ajax.open("GET", url);                                          //5. 비동기 연결을 시작
    ajax.setRequestHeader('Accept', 'application/json');            //6. 요청헤더 설정(수신 데이터의 mime-type)
    ajax.send();                                                    //7. 전송을 시작
  }

  function successCallback(response) {  console.log('response', response);   }
  function errorCallback(err) {    console.log('error', err.message)   }
</script>
```

XMLHttpRequest 객체

https://xhr.spec.whatwg.org/ XHR api 참조

	속성	설명
request	setRequestHeader(속성, 값)	헤더 정보를 설정
	open(method, url, Asynchronous)	요청준비
	send()	서버에 요청 전송
response	status, statusText	서버로부터의 응답 상태를 코드나 텍스트로 나타냄
	getResponseHeader()	응답헤더 정보
	response	responseType 값에 JavaScript 개체 또는 DOMString을 반환
	responseText	응답 결과를 string(csv, json)으로 받아오는 경우
	responseXML	응답 결과를 XML로 받아오는 경우
이벤트	onload	응답을 처리하는 콜백함수
	onError	에러 이벤트 발생 시 처리할 에러 콜백 함수

open() 함수

```
ajax.open(method, url, Asynchronous)
```

- method
  - GET
    - GET이 POST에 비해 더 빠르고 간단해서 GET을 사용
  - POST
    - 서버에서 파일이나 데이터베이스를 업데이트하는 등의 작업할 때(no-cache: cache된 값을 가져와서는 안될 때)
    - 큰 용량의 데이터를 서버에 보낼 때
    - user input을 보낼 때(POST가 훨씬 보안에 좋고 안정적)
- url
  - 서버의 자원을 지칭.
  - 자원은 txt, xml, asp, php, jsp/서블릿 등이 될 수 있으며 그 결과값이 전달
- asynchronus
  - true : 비동기식 요청. send() 함수가 호출된 뒤 곧바로 다음의 코드가 실행
  - false : 동기식 요청. send() 함수가 호출되고, 서버와의 통신이 완전히 완료(콜백함수 실행)된 이후에 다음 코드가 실행

open() 함수 - async(비동기여부)

- 비동기식

```
ajax.open("GET", url, true);
```

```
send 호출
success 콜백
response {
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

- 동기식

```
ajax.open("GET", url, false);
```

```
success 콜백
response {
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
send 호출
```

```
function loadDoc() {
 const ajax = new XMLHttpRequest();
 const url = "https://jsonplaceholder.typicode.com/todos/1";
 ajax.onload = function() {
  if(ajax.status >=200 && ajax.status < 300) {
    console.log("success 콜백")    // 응답완료 후 로그출력
    successCallback(ajax.response);
  } else {
    errorCallback(new Error(ajax.statusText));
  }
 };
 ajax.onerror = errorCallback;
 ajax.open("GET", url, true);     // true:비동기 false:동기식
 ajax.send();
 console.log("send 호출")         // send()후에 로그출력
}
```

open() 함수

- 질의문자열 만들기
  - 템플릿 리터럴을 이용

```
const queryString1 = `userId=${myForm.userId.value}&title=${myForm.title.value}`;
```

  - URLSearchParams 함수 이용

```
const queryString2 = new URLSearchParams(new FormData(myForm)).toString();
```

- method
  - get 방식

```
ajax.open("GET", url+"?"+queryString);   // get이면 url 뒤에 queryString 연결
ajax.send();
```

  - post 방식

```
ajax.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
ajax.open("POST", url);
ajax.send(queryString1);      // post이면 send함수에 지정
```

CORS

- Cross Origin Resource Sharing
- XmlHttpRequest 객체를 이용한 비동기 통신은 서버 쪽에서 원격 요청에 대한 허용을 한 경우에만 요청에 대한 응답 데이터를 받을 수 있다.
- 서버에서 원격 요청에 대한 차단 정책을 사용하는 경우 CORS 에러를 발생시키고 접속이 차단됨
- 해결책
  1. 서버에서 설정 변경
  2. 서버 프로그램
  3. 콜백함수

서버 응답 처리

- ajax.responseText
  - html
  - JSON string
- ajax.responseXml
  - xml
- ajax.response

json 응답

- 응답 결과는 단순 문자열이므로 객체로 변환을 해야 함
- JSON 객체 이용

```
let obj = JSON.parse(ajax.responseText);
```

- eval 함수 이용

```
let obj = eval('(' + ajax.responseText + ')' );
```

json 요청

- JSON 객체를 문자열로 변환

```
let param = { name : 'choi' , age : 20  }
let str = JSON.stringify( param );
```

console.log(typeof param); ← object
console.log(typeof str); ← string

"로 묶이면 string 문자,상수값 {} []는 object=객체로 속성+매서드로 구성(object와 객체가 같은지정확하지 않음 수정)
제이슨 구조로 되어 있는 스트링 값을 오브젝트로 바꾸는게 parse
오브젝트를 스트링으로 바꾸는 것 stringify param은 오브젝트

실습예제

- 실습1 – 객체단건

```
const url = "https://jsonplaceholder.typicode.com/todos/1";
function successCallback(response){
  let todo = JSON.parse(response);
  console.log(todo.title + ":" + todo.completed ) ;
}
```

```
{ "userId": 1,
  "id": 1,
  "title": "delectus",
  "completed": false
}
```

- 실습2 – 배열

```
const url = "https://jsonplaceholder.typicode.com/todos";
function successCallback(response) {
  let todolist = JSON.parse(response);
  for (let i = 0; i < todolist.length; i++) {
    console.log(todolist[i].title + ':' +
todolist[i].completed);
  }
}
```

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "delectus",
    "completed": false
  },
  {
    "userId": 1,
    "id": 2,
    "title": "quis",
    "completed": false
  }
]
```

실습예제

- 실습3 – 영화 일별박스오피스

- 실습4 – 영화 상세정보

fetch() 함수 p.106

프로미스 p.58

- 비동기통신인 프로미스 기술을 기반으로 더 간편하게 사용할 수 있도록 구현된 기술
- 내부 구현은 XMLHttpRequest 객체를 이용.
- fetch 함수

```
fetch(url, {method:"post"} )
.then(res => console.log(res))
```

콜백함수

- fetch 옵션

옵션	사용가능 값	기본값
method	GET, POST, PUT, DELETE	GET
mode	cors, no-cors, same-origin	same-origin
headers	'Content-Type' : application/json, text/plain, application/x-www-form-urlencoded	text/plain
body	BLOB, JSON, FormData	Content-Type 과 같은 타입

fetch 예제 (fetchTest.html)

```
<script>
function loadDoc() {
  const url = "https://jsonplaceholder.typicode.com/todos/1";
  fetch(url)
  .then(response => response.text() )
  .then(response => successCallback(response) )
  .catch(err => errorCallback(err) ) ;
}

function successCallback(response){
  console.log('response', response)
}

function errorCallback(err){
  console.log('error', err)
}
</script>
```

post 요청

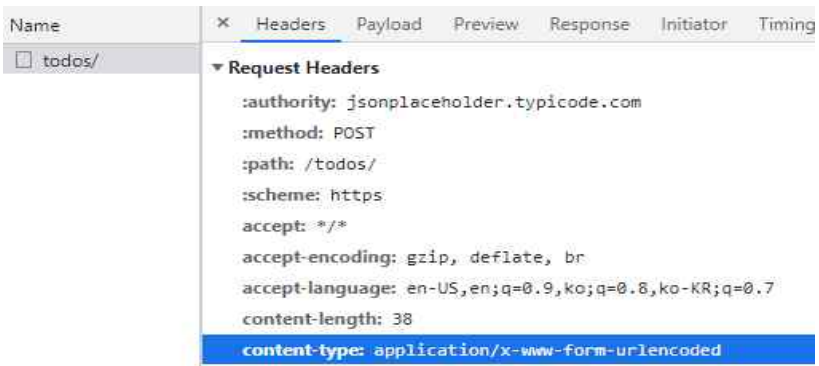
- 요청 파라미터

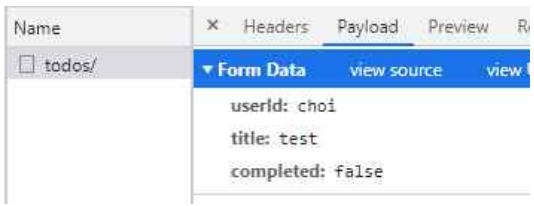
```
const queryString ='userId=choi&title=test&completed=false';
```

```
const formData = new FormData(myForm);
```

- post 요청

```
  fetch(url, {method: 'post',
              headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
              body: queryString  })
```





fetch() - json

- json 응답

```
  fetch(url)
  .then(response=> response.json() )
```

- json 요청 : get 방식은 json 지원안함

```
const url = "https://jsonplaceholder.typicode.com/todos/1";
const obj = { userId:myForm.userId.value,
              title:myForm.title.value,
              completed:myForm.completed.value };

  fetch(url, {method:'put',
              headers: { 'Content-Type': 'application/json' },
              body: JSON.stringify(obj) })
```

HTTP 응답 상태 코드 표(Response Status Code)

- 서버로부터의 응답 상태를 나타냄.
- 서버에서 처리가 올바르게 실행했는지 에러가 발생했는지 서버가 전달한 상태 코드

에러타입	값	텍스트	설명
2XX 성공	200	OK	요청 성공
	204	No Content	응답 헤더와 상태코드는 수신했으나 본문 컨텐츠 내용이 없음
3XX 리다이렉션	301	Moved Rermanently	UIR이 다른 URI로 변경되었음을 알림
	302	Found	URI가 존재하지만 일시적으로 변경되었으며 새 URI 나중에 생성될 예정임을 알림
	304	Not Modifed	브라우저는 더 이상 요청을 하지 않고 캐싱된 파일을 보여줌
4XX 클라이언트 에러	400	Bad Request	서버가 요청을 이해하지 못함
	403	Forbiddon	요청받은 페이지에 대한 접근이 차단됨
	404	Not Found	페이지 없음
	405	Method Not Allowed	허용받지 않은 방법으로 요청함
5XX 서버에러	500	Internal Server Error	서버 오류 발생
	503	Service Unavailable	서버가 요청을 처리할 준비가 되지 않음

지연실행(setTimeout)

- 지연실행
  - 정해진 일정 시간이 경과한 후 콜백 함수를 실행
  - let timer1 = setTimeout(콜백함수, 지연시간(ms))
- 지연 실행 정지
  - 지연실행으로 지연되고 있는 타이머 객체를 해제하면 지연된 콜백함수가 해제
  - clearTimeout(timer1); timer1 = undefined실습

반복지연실행(setInterval)

- 일정 시간 주기로 반복 실행하는 기능.
- 서버에서 일정시간 주기로 데이터를 가져와 화면에 데이터를 갱신하는 경우 이용(채팅)
- 반복지연실행

```
let timer2 = setInterval(콜백함수, 지연시간(ms))
```

- 반복지연실행의 정지

```
clearInterval(timer2);
timer2 = undefined
```

실습예제 (1초마다 시간 출력)

```
<head>
<title>시계</title>
<script type="text/javascript">
function printTime() {
  var clock = document.getElementById("clock");
  var now = new Date();
  clock.innerHTML = '<br>'+ now.getFullYear() + '년 '+ '</br>'
                    (now.getMonth()+1) + '월 '+
                     now.getDate() +'일 '+
                     now.getHours() +'시 '+
                     now.getMinutes() +'분 '+
                     now.getSeconds() +'초';
                     setTimeout("printTime()", 1000);
}

window.onload = function() {
  printTime();
}
</script>
</head>
<body>
   현재 시간은 <span id="clock"></span> 입니다.
</body>
```

실습예제 (delayTest.html)

```
let alarm = {
  notice: function (delay) {
    this.clear();
    this.timer = setTimeout(this.msg, delay * 1000);
  },
  clear: function () {
    if (this.timer) {
      clearTimeout(this.timer);
      this.timer = undefined;
    }
  },
  msg: function () {
    console.log('alarm!')
  }
}
alarm.notice(3);
```