최종보고서

2021년 12월 13일

2021년 2학기 소프트웨어시스템실습

담당교수: 최성종

변형조

2017440061@office.uos.ac.kr

서울시립대학교 전자전기컴퓨터공학부

차 례

[1. 요약 1](#_Toc90159140)

[2. 요구사항 2](#_Toc90159141)

[2.1. 전국적 정보 표시 2](#_Toc90159142)

[2.2. 지역적 정보 표시 3](#_Toc90159143)

[2.3. 세부내용 확인 5](#_Toc90159144)

[3. 설계 6](#_Toc90159145)

[3.1. 사용 기술 6](#_Toc90159146)

[3.2. 소프트웨어 구조도 7](#_Toc90159147)

[3.3. 사용자 입력에 따른 흐름도 7](#_Toc90159148)

[4. 구현 8](#_Toc90159149)

[4.1. 필수 라이브러리 8](#_Toc90159150)

[4.2. 사용한 클래스 9](#_Toc90159151)

[4.3. API 및 함수 설명 24](#_Toc90159152)

[5. 토론 33](#_Toc90159153)

[5.1. 요구사항의 완성도 33](#_Toc90159154)

[5.2. 선택할 수 있는 다양한 기술 분석 35](#_Toc90159155)

[6. 결론 36](#_Toc90159156)

[7. 부록 37](#_Toc90159157)

[8. 참고자료 47](#_Toc90159158)

# 요약

코로나19의 등장으로 재난 정보에 대한 중요성이 높아졌고 개인의 주변에 어떤 재난 상황이 있는지에 대한 관심도 증가했다. 현재 개인이 받는 재난 문자는 개인이 위치한 지역 근처의 문자를 받는다. 그러나 재난의 심각성을 알기 위해서는 전국적인 재난의 분포를 알아야 할 경우가 있다. 따라서 이번 프로젝트에서는 행정안전부에서 발송하는 재난문자가 어디서 발송되었는지 지도에 표시하는 웹 페이지를 만들었다. 웹 페이지에서 전국단위와 지역단위로 재난문자방송 발령 현황을 조회할 수 있고 각 조회 결과로 발령 지역이 지도에 마커로 표시된다. 마커를 클릭하면 정보 창을 띄우거나 지도를 확대할 수 있다.

웹페이지의 제작을 위해 HTML5, CSS, JAVA Script,AJAX,XML,JSON등의 기술이 사용되었고 행정안전부 재난문자방송 발령현황 API와 Naver Map API를 사용하였다.

결과적으로 웹페이지는 필요한 기능을 잘 구현하였고 사용자의 행동별로 예외사항이나 추가 기능을 부여해 사용자가 편리하게 사용하도록 하였다.

# 요구사항

## 전국적 정보 표시

### 사용 시나리오

사용자가 전국에서 발생한 재난문자 정보를 보기위해 “전국”버튼을 누른다.

지도이(가) 표시된 사진

자동 생성된 설명

### 요구사항

기능적으로 재난문자 발송 지역에 마커가 표시되어 마커가 밀집되어 있는 지역이 재난 상황이 많음을 알 수 있도록 한다. 밀집도가 눈에 들어오기 위해서는 충분히 많은 마커가 필요하여 최근 100건 이상의 재난문자 현황을 표시하도록 하였다.

비기능적인 면에서 마커가100개 이상이 표시되려면 사용자가 지연을 느낄 만큼의 로딩 시간이 필요하다. 사용자가 로딩 중에 다른 행동을 하지 않도록 주의를 주기 위해 전국 버튼 클릭시 기다려 달라는 경고를 띄운다. 또한 재난문자 안내 해당 범위가 여러 지역인 경우 각 지역마다 마커를 표시하도록 하였다. 따라서 메모리는 마커 정보를 담을 배열 공간이 필요하다.

지도이(가) 표시된 사진

자동 생성된 설명지도이(가) 표시된 사진

자동 생성된 설명

## 지역적 정보 표시

### 사용 시나리오

사용자가 특별시, 광역시, 도 단위 지역으로 재난문자 정보를 보고자 할 때 지도의 경계 내부를 클릭한다. 만약 군, 구 단위로 재난문자 정보를 알고 싶다면 검색창에 입력하면 된다. 다만 행정구역 단위를 두 개 이상 적을 때는 띄어쓰기가 필요하다.

지도이(가) 표시된 사진

자동 생성된 설명

### 요구사항

기능적 요구사항으로 해당 지역에서 발생한 재난문자 발송 현황을 보여주기 위해 마커를 표시한다. 지역별 재난문자 현황은 너무 많은 마커가 표시되면 보기에 불편하므로 20건만 출력되도록 했다. 비기능적으로 지역별 재난문자 발송현황은 전국적 재난문자 발송 현황에 비해 짧은 시간에 표시가 된다. 전국적 재난문자 상황과 동일하게 안내 범위가 여러 곳인 문자는 동일한 내용에 대해 여러 개의 마커가 표시될 수 있다. 따라서 클릭한 지역 외에도 마커가 표시될 수 있다. 따라서 메모리 요구사항으로는 마커 정보를 저장할 배열 공간이 필요하다.

## 세부내용 확인

### 사용 시나리오

지도에 표시된 마커를 클릭하면 마커에 해당하는 정보창이 표시된다. 자세한 위치를 보기 위해 정보창이 표시된 마커를 우클릭하면 지도가 확대된다. 정보창이 나와있는 마커를 다시 클릭하면 정보창이 닫힌다.

### 요구사항

기능적 요구사항으로 마커를 클릭하면 올바른 정보창이 표시되어야 한다. 또한 마커를 우클릭하면 해당 지점으로 확대되어야 한다. 마커를 클릭하기 위해서는 같은 지점에 대한 서로 다른 두 마커가 있을 때 둘이 어느정도 떨어져 있어 클릭할 수 있도록 되어 있어야 한다. 그리고 확대 한 상태에서 지도를 잘못 클릭하여 지역 선택이 되지 않도록 어느정도 지도가 확대된 상태에서는 지역 선택이 되지 않도록 하였다. 또 정보창이 열린 상태에서 새로운 지역에 대한 마커를 불러오면 정보창이 닫히지 않아 새로운 마커를 검색할 때 자동으로 이전에 열려 있던 정보창은 닫히도록 하였다. 비기능적 요구사항으로 지도의 줌 정보와 마커의 클릭 여부, 정보창이 표시되고 있는지 여부 등을 기록해 두어야 한다. 해당 기능은 네이버 지도 API의 각 기능에 포함되어 있다.

# 설계

## 사용 기술

1.HTML5, CSS, JAVA Script

웹페이지의 구조제작, 스타일 지정, 상호작용을 위한 기술이다.

2. AJAX

웹서버와 비동기적으로 데이터를 교환하기 위한 기술이다. 공공데이터포털과 네이버 API서버에서 정보를 받아오는데 사용되었다.

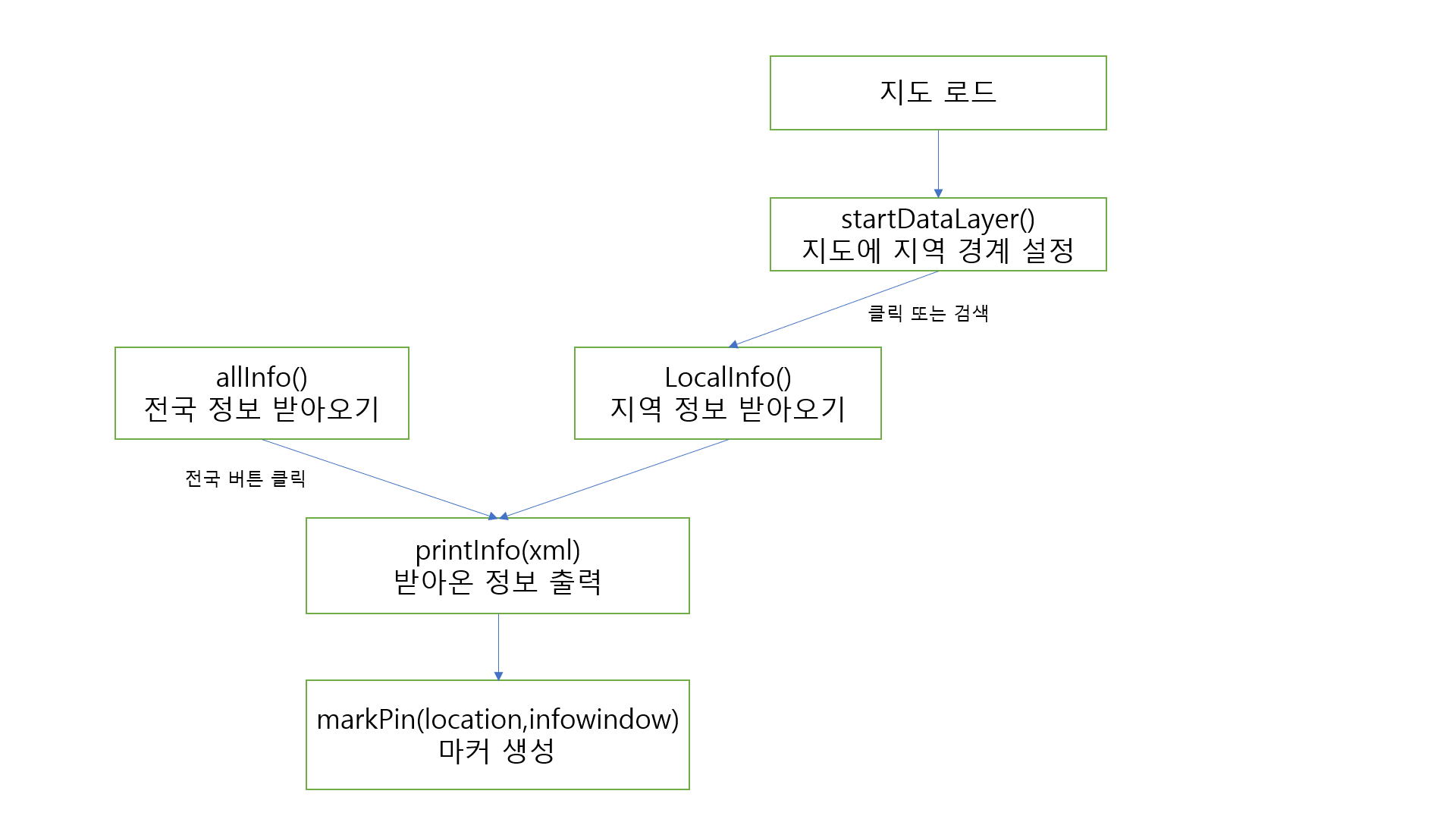
3.XML

인터넷에서 정보를 쉽게 교환하기 위해 정보 작성 방식을 구조화해둔 마크업 언어이다. 공공데이터 포털과 네이버 지도 API에서 받아온 정보가 xml형식을 가진 것이 있다.

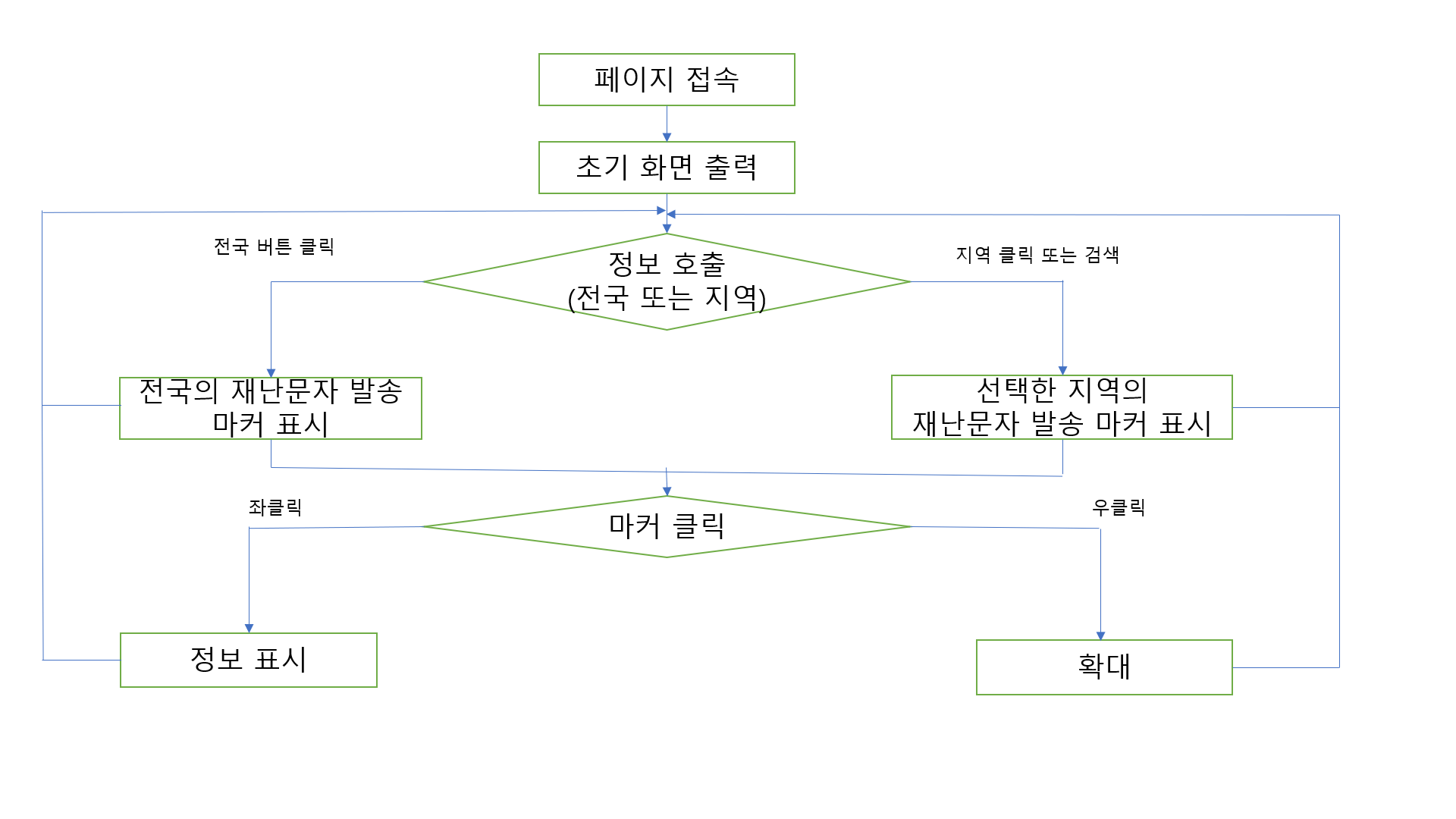
4.JSON

JavaScript 객체를 전달하기 위한 형식이다. 공공데이터 포털과 네이버 지도 API의 정보 일부가 json형식으로 이루어져 있다.

## 소프트웨어 구조도



## 사용자 입력에 따른 흐름도



정보를 다시 호출할 경우 이미 마커나 정보창이 띄워져 있으면 모두 초기화 된 후에 새로운 정보가 표시된다.

# 구현

## 필수 라이브러리

1. Naver Map API - Web Dynamic Map

웹에서 지도를 보여주는 네이버 지도 API

2. Naver Map API – Geo coding

지역 이름에 해당하는 좌표를 알려주는 API

3.공공데이터 포털 API(재난문자방송 발령 현황)

행정안전부에서 제공하는 재난문자방송 발령 현황을 알려주는 API

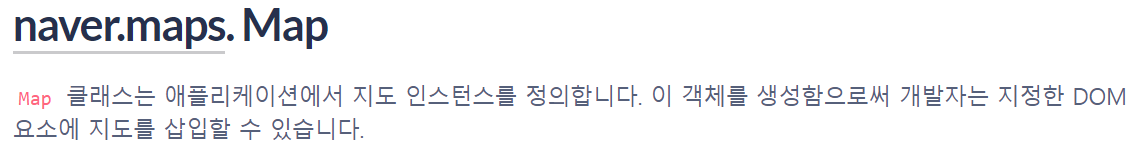
4. JQuary

네이버 서버에 저장된 행정구역 경계 정보를 가져오기 위해 필요.

## 사용한 클래스

### naver.maps.Map

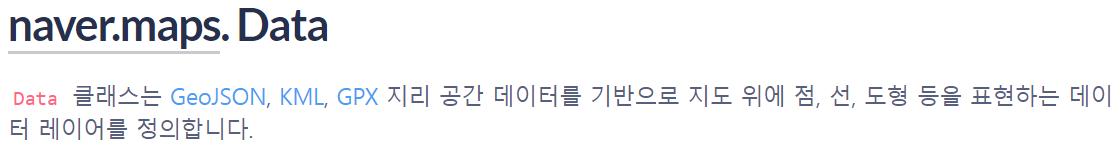
#### 클래스의 주된 목적



#### 사용한 프로퍼티

1)map.data

**1.1) 프로퍼티의 주된 기능**



**1.2) 코드 내에서 기능**

addGeoJson, setStyle, addListener 메소드로 각각 GeoJson데이터를 저장, 스타일 지정, 이벤트 리스너 등록을 한다. naver.maps.Data항목에 추가로 설명이 있다.

#### 사용한 메소드

1) getZoom()

**1.1) 메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.2) 코드 내에서 기능**

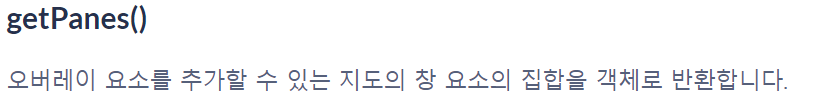
텍스트이(가) 표시된 사진

자동 생성된 설명

현재 보고 있는 zoom정도를 알아내어 9이상 확대되어 있으면 지도 위치를 움직이기 편하도록 지역 선택 기능이 작동하지 않도록 하였다.

2) getPanes()

**1.1) 메소드의 주된 기능**

****

**1.2)코드 내에서 기능**



지도의 창 요소에 툴팁을 적용할 때 사용되었다.

### naver.maps.LatLng

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.1)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

map객체를 생성할 때 지도가 비출 중심점 좌표를 등록해 준다. Map option객체로 naver.maps.Map의 파라미터로 입력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마커의 위치를 전달해 주기 위해서도 사용되었다.

### naver.maps. MapTypeRegistry

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 사용한 메소드

1) set(mapTypeId, mapType)

**1.1)메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

mapTypeId로 ‘normal’을 입력하고 있다.

### naver.maps.Event

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 사용한 메소드

1) once(target, eventName, listener)

**1.1) 메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

‘init\_stylemap’이벤트가 발생하면 ajax로 정보를 불러온 후 startDataLayer함수를 호출한다.

2)addListener

**2.1)메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**2.2)코드 내에서 기능**

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

“click”과 “dbclick”등의 이벤트가 발생할 때 data에 수행할 이벤트를 등록한다.

### naver.maps. Data

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 사용한 메소드

1) addGeoJson(geojson, autoStyle)

**1.1)메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

AJAX로 받아온 GeoJSON정보를 startDataLayer를 이용하여 data에 등록한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

2) setStyle(style)

**2.1)메소드의 주된 기능**

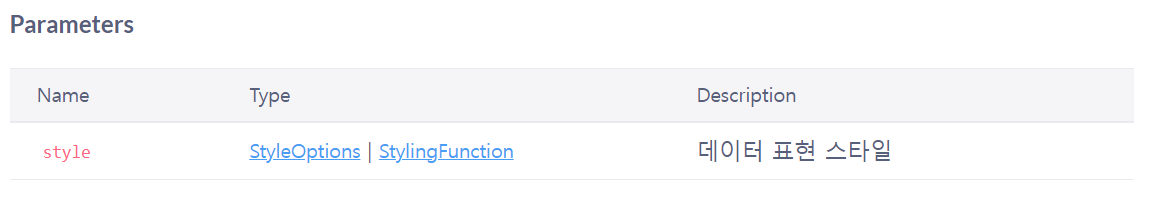
텍스트이(가) 표시된 사진

자동 생성된 설명

**2.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

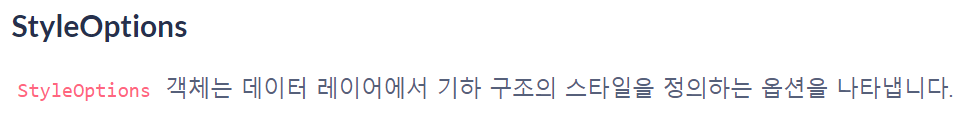
자동 생성된 설명



여기서는 StylingFunction을 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명



3) overrideStyle(feature, style)

**3.1)메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**3.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

마우스가 올라가면 데이터 표현 스타일을 재정의 한다.

4) revertStyle(feature)

**4.1)메소드의 주된 기능**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**4.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

마우스가 자리에서 나가면 스타일을 원래대로 되돌린다.

### naver.maps.GroundOverlay

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 사용한 메소드

1) getBounds()

**1.1)메소드의 주된 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

더블 클릭된 영역의 경계를 얻는다.

### XMLHttpRequest()

#### 클래스의 주된 목적

서버와 데이터를 비동기적으로 받아오기 위해 사용된다.

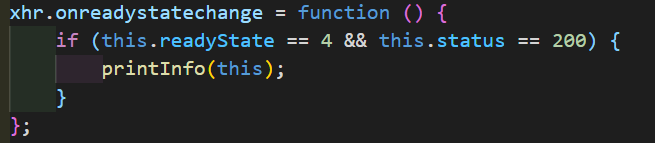
#### 사용한 프로퍼티

**1) onreadystatechange**

**1.1) 프로퍼티의 주된 기능**

readyState 어트리뷰트가 변경될때마다 호출되는 event handler

**1.2) 코드 내에서 기능**



공공데이터 포털에서 XML다운로드가 완료되면 정보를 출력하는 함수를 호출한다.

**2)responseXML**

**2.1)프로퍼티의 주된 기능**

요청에 대한 응답을 XML또는 HTML형태로 반환한다.

**2.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

XML형태로 받은 응답에서 getElementsByTagName으로 필요한 정보를 얻는다.

#### 사용한 메소드

**1)open(method, url)**

**1.1) 메소드의 주된 기능**

새로 생성되거나 기존에 있던 요청을 초기화 한다.

**1.2) 코드 내에서 기능**

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

‘GET’을 url로 요청한다.

**2)send()**

**2.1)메소드의 주된 기능**

요청을 전송한다.

**2.2)코드 내에서 기능**



초기화 된 xhr에대해 요청을 전송한다.

### naver.maps.InfoWindow

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 사용한 메소드

**1)open(map,anchor)**

**1.1) 메소드의 주된 기능**

정보창을 연다. anchor가 지정되면 anchor위에 정보창을 연다.

**1.2)코드 내에서 기능**



**2)close()**

**2.1)메소드의 주된 기능**

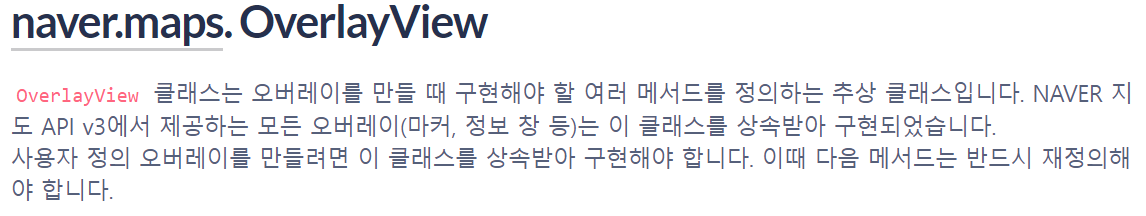
정보 창을 닫고 이벤트를 발생 시킨다.

**2.2)코드 내에서 기능**



### naver.maps.OverlayView

#### 클래스의 주된 목적



#### 사용한 메소드

**1)getMap()**

**1.1)메소드의 주된 기능**

현재의 오버레이가 추가된 map객체를 반환

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

infowindow가 map위에 overlay되어 있으면 닫는다. 즉 열린 창을 닫는다.

**1)setMap(map)**

**1.1)메소드의 주된 기능**

오버레이를 map에 추가한다. map에 null이 주어지면 지도에서 삭제한다.

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

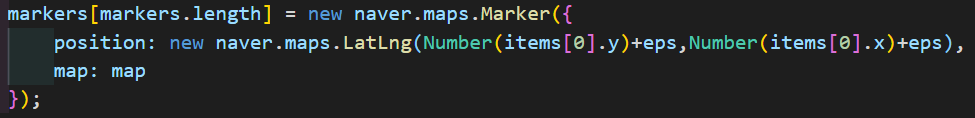
지도에 표시되어 있는 마커와 정보 창을 닫는다.

### naver.maps.Marker

#### 클래스의 주된 목적

텍스트이(가) 표시된 사진

자동 생성된 설명



#### 사용한 메소드

**1)getPosition()**

**1.1) 메소드의 주된 기능**

마커의 위치를 표시한다.

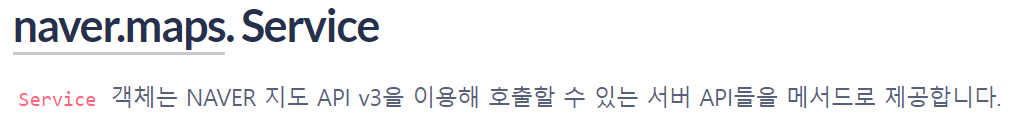
**1.2)코드 내에서 기능**



선택한 마커가 있는 지점으로 지도의 중심을 이동시킨다.

### Service

#### Static 객체의 주된 목적



#### 사용한 프로퍼티

**1)Status**

**1.1)프로퍼티의 주된 기능**

서비스의 응답 상태를 나타낸다.

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

geocode의 응답이 이루어지지 않으면 alert를 발생시킨다.

#### 사용한 메소드

**1)geocode(options, callback)**

**1.1)메소드의 주된 기능**

특정 주소의 좌표를 반환하는 geocode API를 호출한다.

**1.2)코드 내에서 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

사용자가 검색어를 입력하거나 지역을 클릭하면 location을 받아와 해당 지역의 좌표를 얻기 위해 사용된다.

## API 및 함수 설명

### 행정안전부 재난문자방송 발령현황

행정안전부에서 재공하는 재난문자방송 발령 현황을 확인할 수 있는 open API이다. 전국적 정보와 지역별 정보를 확인 할 수 있는 두 종류의 API가 있다.

전국인 경우 요청 변수와 출력 결과는 아래와 같다.



텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

지역별인 경우 요청 변수와 출력 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

### Naver Map API

네이버에서 제공하는 지도 API이다. 지도 표시, 좌표 찾기, 경로 찾기, 스트리트 뷰 등 여러가지 기능을 제공하지만 지도 표시기능과 좌표 찾기 기능을 사용하였다. 지도 표시 API는 Web Dynamic Map이고 좌표 찾기 기능은 Geocoding이다. 여러가지 기능은 공식 문서를 참고하였다.

<https://navermaps.github.io/maps.js.ncp/docs/tutorial-2-Getting-Started.html>

### 자체 정의 함수

#### allInfo()

전국 재난문자방송 발령 현황을 표시해주는 함수이다. 웹페이지의 “전국”버튼을 클릭하면 호출된다. 내부적으로 행정안전부 재난문자방송 발령 현황 API를 호출하여 xml데이터를 받아온 다음 해당 정보를 가공하도록 printInfo에 입력으로 xml데이터를 넘겨주며 함수를 호출한다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

#### localInfo()

allInfo와 동일한 과정으로 함수가 수행되지만 행정안전부 재난문자방송 발령 현황(지역별) API를 호출하기 위해 지역 이름을 전달해야 한다. 지역 이름은 사용자가 웹페이지에서 지도의 행정구역을 클릭하거나 검색창에 지역 이름을 입력한 값을 이용한다. allInfo()와 동일하게 내부적으로 printInfo를 호출하여 받아온 정보를 가공한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

#### printInfo(xml)

행정안전부 재난문자방송 발령 현황 API에서 받아온 xml정보를 가공하는 함수이다. 재난문자방송 발령 시간, 발령 지역, 내용을 추출하여 정보 창 객체를 만들고 마커를 만들기 위해 내부적으로 markPin(location, infowindow)함수를 호출한다. 처음에 호출되면 기존에 표시되어 있던 마커와 정보 창을 전부 지도에서 제거한다. 이후 xml에서 정보를 추출하고 지역의 수만큼 정보 창과 마커를 만든다. 마커를 만드는 일은 markPin(location, infowindow)함수로 수행한다. 다만 하나의 문자에 지역이 여러 개 있을 경우는 해당하는 모든 지역에 정보를 표시하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

#### markPin(location, infowindow)

지역의 이름과 정보 창을 입력으로 받으면 함수 내부적으로 geocoding API를 사용하여 지역 이름에 해당하는 좌표를 얻어온다. 각 좌표에 해당하는 부분에 마커를 만들고 마커에 정보 창을 연결한다. 단 동일한 좌표에 여러 마커가 생기는 것을 방지하고자 얻어온 좌표에 약간의 random값을 더해주어 마커가 서로 구별되도록 했다. 좌표의 오차는 구 단위 면적을 벗어나지 않는다.

마커에는 정보 창 말고도 이벤트를 등록하는데 마우스 왼쪽을 클릭하면 정보 창이 나오고 다시 클릭하면 닫힌다. 마우스 오른쪽을 클릭하면 지도가 확대되며 마커가 위치하는 지점으로 중심이 맞춰진다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

#### startDataLayer()

지도에 행정구역 경계를 표시하고 해당 구역 안에 마우스가 올라오고 나가거나 지역을 클릭할 때 발생하는 이벤트를 등록하는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 화면, 스크린샷, 은색이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

# 토론

## 요구사항의 완성도

#### 전국적 정보 표시

전국적 정보 표시는 “전국”버튼을 클릭하면 전국 단위로 최근 100건의 재난문자방송 발령 현황을 보여줘야한다. 완성된 웹페이지는 해당 기능을 잘 수행할 수 있다. 100개의 마커를 지도상에 잘 표시하며 이를 저장하기 위한 메모리 공간은 현재 대부분 컴퓨터에서 감당할 수 있을 정도로 작다. 다만 응답 시간은 응답이 올 때까지 사용자가 예측하지 못한 행동을 할 수 있을 정도로 길다. 응답 시간 자체는 API의 문제이므로 줄일 수 없다고 해도 로딩이 될 동안 기다려 달라는 경고에 의존할 것이 아니라 입력을 차단할 방법을 추가적으로 구상하면 완성도가 높아질 것이다.

#### 지역적 정보 표시

지역을 검색하거나 지도상에서 클릭하면 해당 지역과 관련된 재난문자방송 발령 현황 20개를 마커로 표시한다. 차지하는 메모리 공간은 충분히 작고 응답시간도 즉각적이다. 이후 다른 지역에 대해 검색하고자 하면 기존의 마커를 지우고 새로운 지역에 대해 마커를 표시하기 때문에 시각적으로 깔끔하게 제작하였다. 다만 선택한 지역 외에 다른 지역도 포함하는 문자가 있다면 다른 지역에도 마커가 표시되어 혼동을 줄 가능성이 있다. 마커를 생성하는 과정에서 간단한 수정을 통해 지역명과 다른 곳에는 마커가 생성되지 않도록 할 수 있지만 해당 재난이 다른 영향을 미치는 범위를 알 수 있다는 장점이 있어 수정하지 않고 다른 지역도 표시되도록 하였다.

#### 세부내용 확인

마커를 클릭하면 정보 창이 표시되어야 한다. 정보 창이 표시되어 있는 마커에 우클릭을 할 경우 해당 부분이 확대되도록 하였다. 이를 위해 정보 창을 저장할 메모리 공간은 충분히 작으며 정보창을 표시하거나 지도가 확대되는 시간도 즉각적이다. 정보 창이 표시되지 않은 마커는 우클릭을 해도 확대되지 않도록 하였고 정보 창이 표시되어 있는 상태로 다른 마커를 클릭하면 기존의 정보 창은 사라져 시각적으로 좋게 완성도를 높였다. 또한 마커를 우클릭하여 지도가 확대된 상태에서 지도를 조작하면 처음에 지도를 클릭하여 지역을 검색하는 이벤트가 발생하기 때문에 특정 줌 레벨 이상인 경우 지도를 클릭하여 지역적 검색을 하는 기능이 비활성화 되도록 하였다. 따라서 사용자가 주의 하지 않고 편안하게 이용해도 사용자의 의도대로 작동하도록 완성도를 높였다.

## 선택할 수 있는 다양한 기술 분석

#### Naver Map API

Naver Map API대신 Kakao 지도 API나 Google 지도 API를 사용할 수 있다.

#### Geocoding

Geocoding대신 공공데이터 포털에서 제공하는 행정구역 정보를 활용할 수 있다. 또는 GIS의 행정구역에 대한 SHP파일을 GeoJSON형태로 바꿔 지도 위에 그릴 수도 있다.

# 결론

재난문자방송 발령 현황을 보여주는 웹페이지의 주요 기능은 전국 단위 발령 현황 검색, 지역 단위 발령 현황 검색, 그리고 그들의 세부 내용 표시이다. 전국 단위 발령 현황은 “전국”버튼으로 작동하며 100개의 마커를 정상적으로 표시할 수 있지만 지연시간이 걸린다. 지역 단위 발령 현황은 지도 입력과 검색어로 검색할 수 있고 20개의 마커를 정상적으로 표시할 수 있다. 전국과 지역 단위 모두 마커에 대해 정보 창 표시, 확대의 기능을 사용할 수 있다. 모든 기능은 줌 상태로 지도 이동시 지역 클릭이 되지 않는 등 최대한 사용자가 이용하기 편리하도록 설계되었다. 또한 매번 검색마다 이전 검색 내용이 지워지도록 하는 등 사용자 입력에 따라 예측하지 못한 결과 발생 가능성을 사전에 차단하였다.

향후 전국 단위 정보 표시에서 지연시간을 줄이거나 해당 시간동안 입력을 원천 차단하는 방안으로 안정성이 개선될 수 있다. 또 같은 위치에 표시되는 마커들은 random으로 약간씩 위치 조정을 하는 것이 아니라 클러스터 형식으로 묶고 이를 클릭하면 개별적인 정보를 볼 수 있는 방식으로 개선이 가능하다. 마지막으로 표시되는 정보가 재난 관련 정보인 만큼 주변에 경찰서, 소방서, 보건소 등의 안내를 추가하는 개선이 가능하다.

# 부록

### 활용한 기술 설명

#### HTML, CSS, JAVAScript

HTML은 브라우저가 HTML문서를 읽고 해석하여 웹에 표시할 정보의 구조를 만드는데 사용된다. CSS는 웹페이지에 들어갈 요소들의 스타일을 지정하는데 사용된다. 마지막으로 JAVAScript는 웹페이지의 내용을 동적으로 바꾸거나 응용프로그램을 작성하는데 사용된다.

#### AJAX

웹서버와 비동기적으로 데이터를 교환하기 위한 기술이다. AJAX로 필요한 데이터만을 웹서버에 요청해서 받은 후 클라이언트에서 데이터에 대한 처리를 할 수 있다. 따라서 웹페이지를 다시 로드 하지 않고 받아온 데이터만 표시한다. 공공데이터포털과 네이버 API서버에서 정보를 받아오는데 사용되었다.

#### XML

인터넷에서 정보를 쉽게 교환하기 위해 정보 작성 방식을 구조화해둔 마크업 언어이다. HTML과 같이 태그로 구성되어 있고 구조적 관계가 있다. 공공데이터 포털과 네이버 지도 API에서 받아온 정보가 xml형식을 가진 것이 있다.

#### JSON

JavaScript 객체를 전달하기 위한 형식으로 “키-값”쌍으로 이루어져 있다. XML과 유사한 목적으로 인터넷에서 자료를 교환하기 쉽도록 개발되었다. 공공데이터 포털과 네이버 지도 API의 정보 일부가 json형식으로 이루어져 있다.

### 전체 소스코드

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no">

    <title>재난문자 지도</title>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <script type="text/javascript" src="https://openapi.map.naver.com/openapi/v3/maps.js?ncpClientId=besgcxdew4&submodules=geocoder"></script>

</head>

<style>

    table,th,td {

        border : 1px solid black;

        border-collapse: collapse;

    }

    th,td {

        padding: 5px;

        font-size: 15px;

    }

</style>

<body>

<div id="map" style="width:100%;height:500px;"></div>

<input id = "text" type="text" style="display: inline-block;" onclick = "eraseForm()" value="시/군/구 단위 띄어쓰기">

<button type="button" onclick="localInfo()">검색</button>

<button type="button" onclick="allInfo()">전국</button>

<table id="info"></table>

<script>

var markers = [];

var InfoList = [];

function eraseForm(){

    document.getElementById("text").value = "";

}

function allInfo(){

    alert("로딩중... 잠시만 기다려주세요.");

    var xhr = new XMLHttpRequest();

    var url = 'http://apis.data.go.kr/1741000/DisasterMsg3/getDisasterMsg1List'; /\*URL\*/

    var queryParams = '?' + encodeURIComponent('serviceKey') + '='+'qlZbl32c93Xg3kzmvwouzkCFGDViSRH%2Fheq2JQEZ3o%2BSyYLCws8Y2oX%2FMjYF3HOvGQkC1ni%2FW5kWilcNYKR0bg%3D%3D'; /\*Service Key\*/

    queryParams += '&' + encodeURIComponent('pageNo') + '=' + encodeURIComponent('1'); /\*\*/

    queryParams += '&' + encodeURIComponent('numOfRows') + '=' + encodeURIComponent('100'); /\*\*/

    queryParams += '&' + encodeURIComponent('type') + '=' + encodeURIComponent('xml'); /\*\*/

    xhr.open('GET', url + queryParams);

    xhr.onreadystatechange = function () {

        if (this.readyState == 4 && this.status == 200) {

            printInfo(this);

        }

    };

    xhr.send('');

    }

function localInfo(){

    var region = document.getElementById("text").value;

    var xhr = new XMLHttpRequest();

    var url = 'http://apis.data.go.kr/1741000/DisasterMsg4/getDisasterMsg2List'; /\*URL\*/

    var queryParams = '?' + encodeURIComponent('serviceKey') + '='+'qlZbl32c93Xg3kzmvwouzkCFGDViSRH%2Fheq2JQEZ3o%2BSyYLCws8Y2oX%2FMjYF3HOvGQkC1ni%2FW5kWilcNYKR0bg%3D%3D'; /\*Service Key\*/

    queryParams += '&' + encodeURIComponent('pageNo') + '=' + encodeURIComponent('1'); /\*\*/

    queryParams += '&' + encodeURIComponent('numOfRows') + '=' + encodeURIComponent('20'); /\*\*/

    queryParams += '&' + encodeURIComponent('type') + '=' + encodeURIComponent('xml'); /\*\*/

    queryParams += '&' + encodeURIComponent('create\_date') + '=' + encodeURIComponent('2021/06/22 00:00:00'); /\*\*/

    queryParams += '&' + encodeURIComponent('location\_name') + '=' + encodeURIComponent(region); /\*\*/

    xhr.open('GET', url + queryParams);

    xhr.onreadystatechange = function () {

        if (this.readyState == 4 && this.status == 200) {

            printInfo(this);

        }

    };

    xhr.send('');

}

function printInfo(xml) {

    //마커 초기화 후 다시 표시

    for(let i=0;i<markers.length;i++)

        markers[i].setMap(null);

    markers = [];

    for(let i=0;i<InfoList.length;i++)

        InfoList[i].setMap(null);

    InfoList = [];

    var i, j;

    var xmlDoc = xml.responseXML;

    //var table="<tr><th>날짜</th><th>지역</th><th>내용</th></tr>";

    let date = xmlDoc.getElementsByTagName("create\_date");

    var loc = xmlDoc.getElementsByTagName("location\_name");

    let msg = xmlDoc.getElementsByTagName("msg");

    for(i=0;i<loc.length;i++){

        let raw = loc[i].innerHTML;

        let loc\_list = raw.split(",");

        for(j=0;j<loc\_list.length;j++){

            let location = loc\_list[j];

            location = location.replace("전체","");

            contentString = date[i].innerHTML +"</br>"+ loc[i].innerHTML +"</br>"+ msg[i].innerHTML;

            var infowindow = new naver.maps.InfoWindow({

            content: contentString

            });

            InfoList[InfoList.length] = infowindow;

            markPin(location,infowindow);

        }

    }

    /\*

    for (i = 0; i <loc.length; i++) {

        table += "<tr><td>" +

        date[i].innerHTML+

        "</td><td>"+

        loc[i].innerHTML+

        "</td><td>" +

        msg[i].innerHTML+

        "</td></tr>";

    }

    document.getElementById("info").innerHTML = table;

    \*/

}

function markPin(location, infowindow){

    naver.maps.Service.geocode({

            query: location

        }, function(status, response) {

            if (status !== naver.maps.Service.Status.OK) {

                return alert(status);

            }

            var result = response.v2, // 검색 결과의 컨테이너

                items = result.addresses; // 검색 결과의 배열

            var eps = Math.random()/100;

            markers[markers.length] = new naver.maps.Marker({

                position: new naver.maps.LatLng(Number(items[0].y)+eps,Number(items[0].x)+eps),

                map: map

            });

            var marker = markers[markers.length-1];

            naver.maps.Event.addListener(marker, "click", function(e) {

            //현재 overlay가 추가된 객체 반환 naver.maps.OverlayView

            if (infowindow.getMap()) {

                infowindow.close();}

            else {infowindow.open(map, marker);}

            });

            naver.maps.Event.addListener(marker, 'rightclick', function() {

                if (infowindow.getMap()) {

                    map.setZoom(10);

                    map.setCenter(marker.getPosition());

                }

            });

        });

}

</script>

<script>

var map = new naver.maps.Map(document.getElementById('map'), {

    zoom: 5,

    mapTypeId: 'normal',

    center: new naver.maps.LatLng(37.3586524, 127.1060678)

});

var HOME\_PATH = "https://navermaps.github.io/maps.js/docs"

    urlPrefix = HOME\_PATH +'/data/region',

    urlSuffix = '.json',

    regionGeoJson = [],

    loadCount = 0;

//일회성 이벤트 정의(GeoJSON)

naver.maps.Event.once(map, 'init\_stylemap', function () {

    for (var i = 1; i < 18; i++) {

        var keyword = i +'';

        if (keyword.length === 1) {

            keyword = '0'+ keyword;

        }

        $.ajax({

            url: urlPrefix + keyword + urlSuffix,

            success: function(idx) {

                return function(geojson) {

                    regionGeoJson[idx] = geojson;

                    loadCount++;

                    if (loadCount === 17) {

                        startDataLayer();

                    }

                }

            }(i - 1)

        });

    }

});

var tooltip = $('<div style="position:absolute;z-index:1000;padding:5px 10px;background-color:#fff;border:solid 2px #000;font-size:14px;pointer-events:none;display:none;"></div>');

tooltip.appendTo(map.getPanes().floatPane);

function startDataLayer() {

    map.data.setStyle(function(feature) {

        var styleOptions = {

            fillColor: '#ff0000',

            fillOpacity: 0.0001,

            strokeColor: '#ff0000',

            strokeWeight: 2,

            strokeOpacity: 0.4

        };

        if (feature.getProperty('focus')) {

            styleOptions.fillOpacity = 0.6;

            styleOptions.fillColor = '#0f0';

            styleOptions.strokeColor = '#0f0';

            styleOptions.strokeWeight = 4;

            styleOptions.strokeOpacity = 1;

        }

        return styleOptions;

    });

    //map에 GeoJSON데이터 등록

    regionGeoJson.forEach(function(geojson) {

        map.data.addGeoJson(geojson);

    });

    //여러가지 eventListener등록

    map.data.addListener('click', function(e) {

        if(map.getZoom()<9){

            var feature = e.feature;

            var region = feature.getProperty('area1');

            document.getElementById("text").value = region;

            localInfo();

        }

    });

    map.data.addListener('mouseover', function(e) {

        //9이상 줌을 하면 붉은 색 안나오게

        if(map.getZoom()<9){

            var feature = e.feature,

                regionName = feature.getProperty('area1');

            tooltip.css({

                display: '',

                left: e.offset.x,

                top: e.offset.y

            }).text(regionName);

            map.data.overrideStyle(feature, {

                fillOpacity: 0.6,

                strokeWeight: 4,

                strokeOpacity: 1

            });

        }

    });

    map.data.addListener('mouseout', function(e) {

        tooltip.hide().empty();

        map.data.revertStyle();

    });

}

</script>

</body>

</html>

# 참고자료

1)Naver Map API 공식 문서

<https://navermaps.github.io/maps.js.ncp/docs/tutorial-2-Getting-Started.html>

2)Naver Map API (Geocoder) 공식 문서

<https://navermaps.github.io/maps.js.ncp/docs/tutorial-Geocoder-Geocoding.html>

3)행정안전부 재난문자방송 발령현황 API

<https://www.data.go.kr/data/3058822/openapi.do>