8253jang@daum.net
장희정

# Web 보안

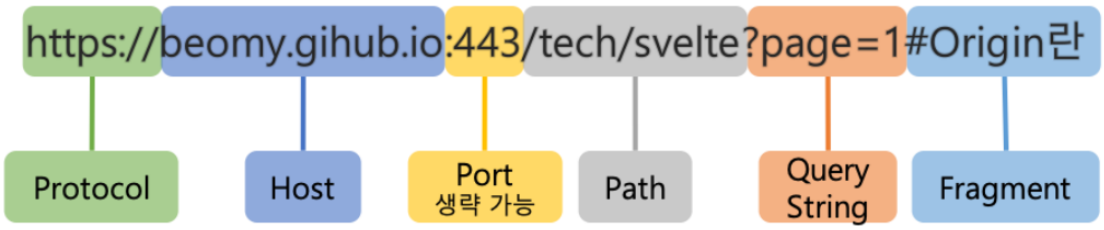
## Same Origin Policy vs CORS(Cross Origin Resource Sharing)

: 웹에서 리소스를 안전하게 주고받기 위한 기본 개념으로 요즘 웹 애플리케이션에서 다른 출처의 리소스를 요청하는 경우가 많은데 보안문제를 방지하기 위해 **브라우저는 Same-Origin Policy를 기본적으로 채택하고** 있다.

### ☞ 동일출처 정책(Same Origin Policy)

: 동일 출처 정책(Same-origin policy)는 다른 출처로부터 조회된 자원들의 읽기 접근을 막아 다른 출처 공격 즉, 같은 출처에서만 리소스를 공유 할 수 있도록 제한하는 보안 메커니즘이다.

**: 동일 출처(Origin)는 Protocol + Host + Port 가 같은 것.**




https://beomy.github.io/tech/browser/cors/

Ex) 1) http://jang.or.kr:80/test.html
2) https://jang.or.kr:80/test.html
위, 1), 2)은 protocol이 다르므로 다른 출처이다.

Ex) 1) http://jang.or.kr:80/test.html
2) http://jang.or.kr:80/test2.html
위, 1), 2)은 동일 출처

Ex) 1) http://jang.or.kr:8080/test.html
2) http://jang.or.kr:80/test.html

8253jang@daum.net
장희정

위, 1), 2)은 다른 출처

**하지만**,
SOP정책은 웹 애플리케이션 개발 시 **다양한 출처의 리소스를 활용해야 하는 현대의 웹 환경에서는 기능구현에 제약이 많다.**

**따라서,**
특정 조건 하에 다른 출처의 리소스 접근을 허용하는 방법이 필요하게 되었고 이러한 필요성에 의해 CORS 정책이 도입되었으며, 이는 Same-Origin Policy를 유지하면서도 보다 유연한 리소스 공유를 가능하게 한다.

## ☞ CORS(Cross Origin Resource Sharing)

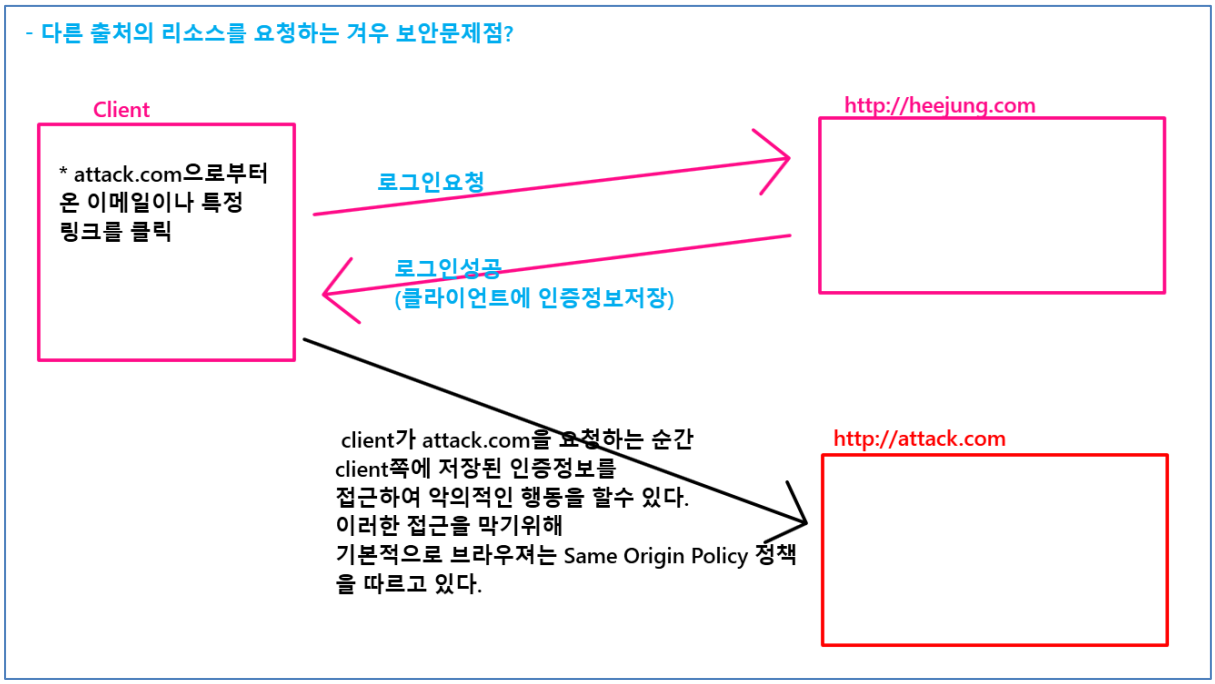
: **출처**가 다른 자원을 공유한다는 뜻으로 한 출처에 있는 자원에서 다른 출처에 있는 자원에 접근하도록 하는 개념이다.

: CORS는 브라우저와 서버 간의 협상을 통해 특정 출처의 리소스 접근을 허용하는 방식으로 작동한다.

: **CORS 요청을 할 때 요청-응답 프로세스는**

**브라우저는** 현재 Origin의 protocol, host, port에 대한 정보가 포함된 'Origin' 헤더를 통해 요청을 보내고, **서버는** 현재 Origin의 헤더를 확인한 후 응답으로 요청된 데이터와 'Access-Control-Allow-Origin' 헤더를 포함하여 응답한다.

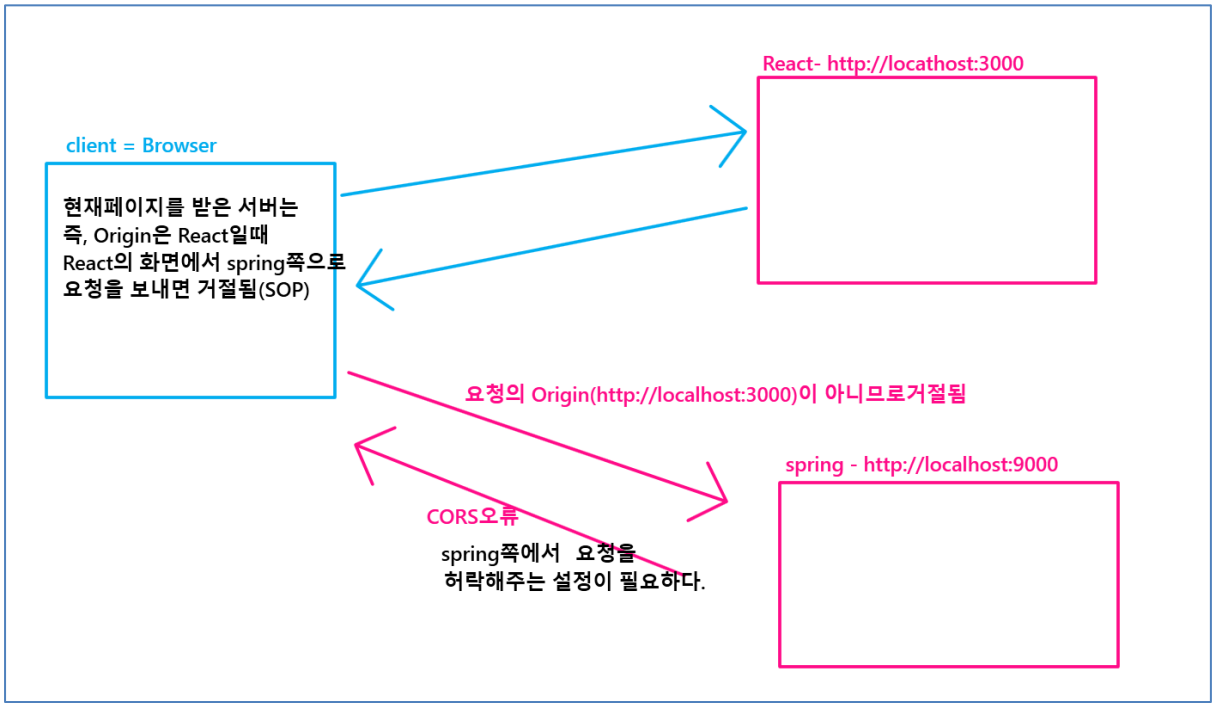
8253jang@daum.net
장희정




**만약,**

Front는 React -> http://localhost:3000

Back은 springBoot-> http://localhost:9000



8253jang@daum.net
장희정

## Spring 환경에서 CORS설정방법

### 1) 전역설정 - CORS Config 클래스 설정

```
2)  package com.spring.board.config;
3)
4)  import org.springframework.context.annotation.Configuration;
5)  import org.springframework.web.servlet.config.annotation.CorsRegistry;
6)  import org.springframework.web.servlet.config.annotation.EnableWebMvc;
7)  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
8)
9)  /**
10)  * WebMvcConfigurer를 이용해서 @CrossOrigin 글로벌 설정
11)  * */
12) @Configuration
13) @EnableWebMvc
14) public class WebMvcConfig implements WebMvcConfigurer {
15)      @Override
16)      public void addCorsMappings(CorsRegistry registry) {
17)          registry.addMapping("/**")
18)                  .allowedOrigins("http://localhost:3000")
19)                  .allowedMethods("OPTIONS","GET","POST","PUT","DELETE");
20)      }
```

**1) addMapping**

addMapping을 사용해 CORS를 적용할 URL 패턴을 정의
위 처럼 "/**" 와일드 카드를 사용할 수도 있고 "/somePath/**" 설정가능

**2) allowedOrigins**

allowedOrigins를 이용해서 자원 공유를 허락할 Origin을 지정

allowedOrigins("*") 가능

.allowedOrigins("http://localhost:8080", "http://localhost:8081");

**3) allowedMethods**

allowedMethods를 이용해서 위 처럼 허용할 HTTP method를 설정
이 때 "*"를 사용해 모든 method를 허용할 수도 있음.

**4) allowedHeaders**

allowedHeaders를 이용해 클라이언트 측의 CORS 요청에 허용되는 헤더를 설정

allowedHeaders("Authorization", "Content-Type")

기본적으로 Content-Type, Accept 및 Origin과 같은 간단한 요청 헤더만 허용.

**5) exposedHeaders**

exposedHeaders를 이용해 클라이언트측 응답에서 노출되는 헤더를 지정.

.exposedHeaders("Custom-Header")

**6) allowCredentials**

allowCredentials를 이용해 클라이언트 측에 대한 응답에 credentials(예: 쿠키, 인증 헤더)를 포함할 수 있는지 여부를 지정. 기본값은 false.

true로 설정하면 클라이언트가 요청에 credentials를 포함하고 응답에서 받을 수 있음. credentials를 사용할 때 응답의 Access-Control-Allow-Origin 헤더가 *로 설정되지 않았는지 확인해야 함. 요청 원본과 명시적으로 일치해야 한다.

.allowCredentials(true)

**7) maxAge**

maxAge를 이용해서 원하는 시간만큼 pre-flight 리퀘스트를 캐싱 해둘 수 있음.

.maxAge(3600);

☞ Preflight 는 단어 그 자체에서 알 수 있듯, 실제 통신을 하기 전에 미리 통신을 한 번 하는 것이다. 미리 통신을 보내서 Access-Control-Allow-Origin 설정은 되어있는지, 우리가 지금 보내려고 하는 메서드가 이 서버에서 사용 가능한지 확인할 수 있다.

8253jang@daum.net
장희정

### 2) 특정 Controller의 설정

```
@RequestMapping("/boards")
@CrossOrigin(origins = "*", allowedHeaders = "*")
 public class BoardController {
        ...
}
```

### 3)특정 메소드의 설정

```
@RestController
@RequestMapping("/boards")
   public class BoardController {
        @CrossOrigin(origins="*")
        @DeleteMapping(value = "/{boardId}")
       public ResponseEntity<BoardResponseDto> delete(@PathVariable Long boardId) throws
       Exception{
         ...
    }
```