# 강의 링크

<https://www.inflearn.com/course/%EC%8A%A4%ED%94%84%EB%A7%81-%EC%9E%85%EB%AC%B8-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8>

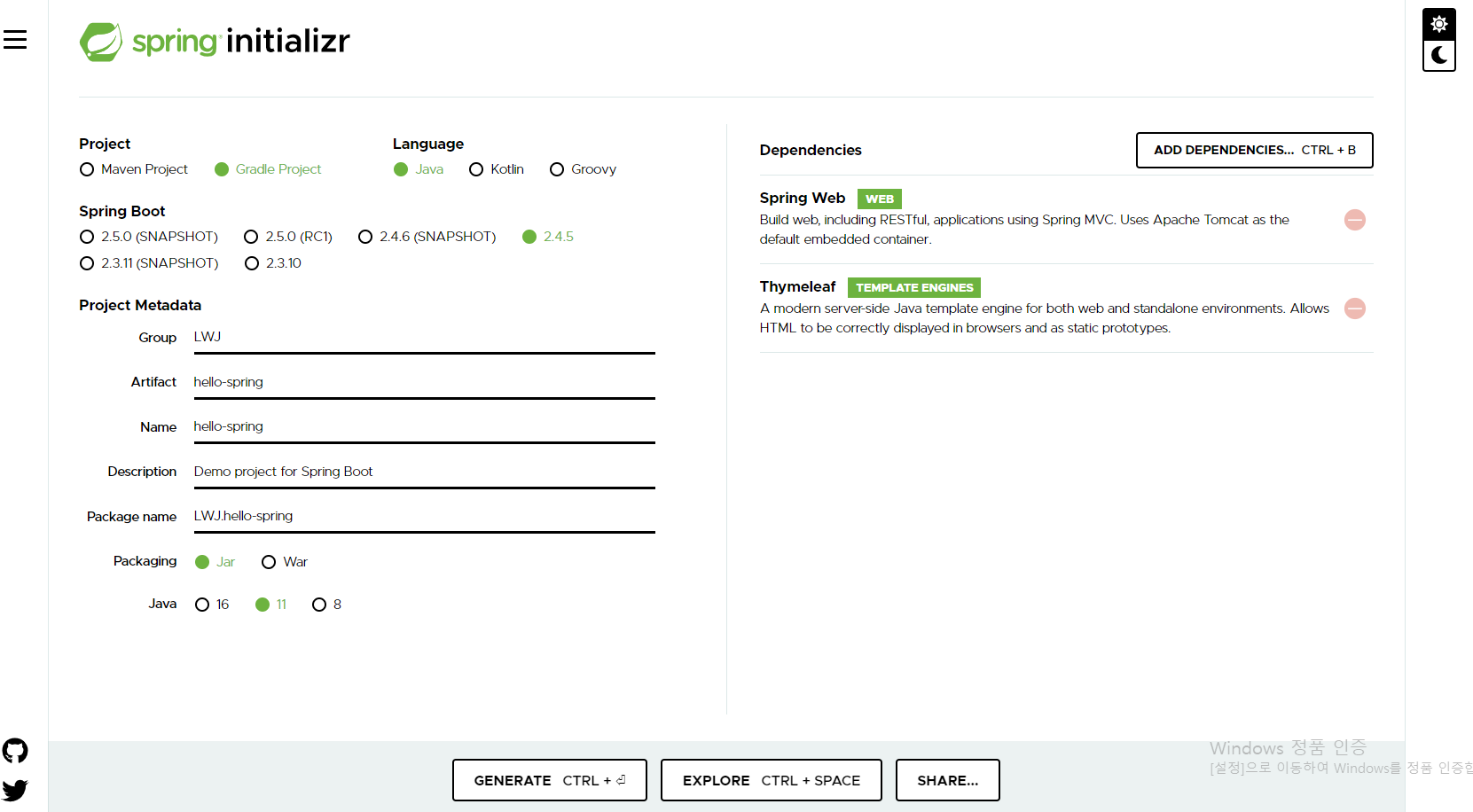
(김영한님)

# 환경 설정

1. java 11 설치 (oracle 가입 필요)
2. IntelliJ 설치
   1. 프로젝트 열고 빌드 시 - 환경설정에서 gradle 검색해보면 build Tools 아래에 Gradle 나올 것, 여기서 Build and run using, Run tests using 모두 IntelliJ IDEA 로 변경해야 빌드 빨리될 것.

# 프로젝트 생성

1. start.spring.io



1. project : Gradle 선택
2. Spring Boot

SNAPSHOT 은 아직 만들고있는 버전

다른것도 아직 정식 릴리즈된 버전이 아님

아무것도 안적혀있는 최신으로.

1. Project Metadata

Group 에 보통 기업 명

Artifact 는 빌드되어 나오는 결과물

1. Dependencies

ADD DEPENDENCIES 클릭해서

WEB - web 검색 후 SPRING WEB (tomcat 내장)

TEMPLETE ENGINES - Thymeleaf (html 관련)

1. GENERATE 클릭하면 압축파일 받아지고, 이걸 압축 풀면됨
2. IntelliJ

open 눌러서 다운받은 폴더 가서 build.gradle 찾아서 이걸 오픈 (Open as Project)

처음 오픈하면 이것저것 다운로드 받을거 (네트워크 연결 필요)

1. 실행

src/main/java 아래에 코드들 있을거.

src/test 는 테스트코드

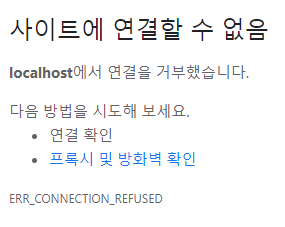
src/main/java 아래에 HelloSpringApplication 에 main 있을거고, 실행하면

~embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)

이런거 나올거고, 인터넷으로 localhost:8080 들어가보면



이런거 나올 것. 아무것도 없으므로 이렇게 나오는게 성공.



이게 실패

1. gradle

build.gradle 에 보면 dependencies 나옴.

# 첫 화면 생성

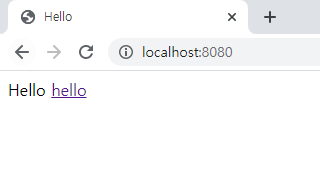
1. Welcome Page

resources/static/index.html 생성

<index.html>

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>Hello</title>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  </head>  <body>  Hello  <a href="/hello">hello</a>  </body>  </html> |

후 실행하면 localhost:8080 에서 아래와 같이 뜰 것.



스프링 부트가 제공하는 welcome page 기능.

resource/static/index.html 을 만들면 이걸 참조해서 뿌려줌.

이러한 기능들 찾아서 할 수 있음. (spring.io 들어가서 project/spring boot/learn 의 reference doc 들어가면 spring boot feature 에서 welcome 또는 index.heml 검색)

1. 템플릿 엔진 (thymeleaf) / controller

controller package 하나 만들고, 아래에 HelloController 생성

|  |
| --- |
| package LWJ.hellospring.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  @Controller  public class HelloController {  @GetMapping("hello") // localhost:8080/hello 에서의 hello 를 의미함.  public String hello(Model model) {  model.addAttribute("data", "hello!!");  return "hello"; // templates/hello.html 를 찾아감.  }  } |

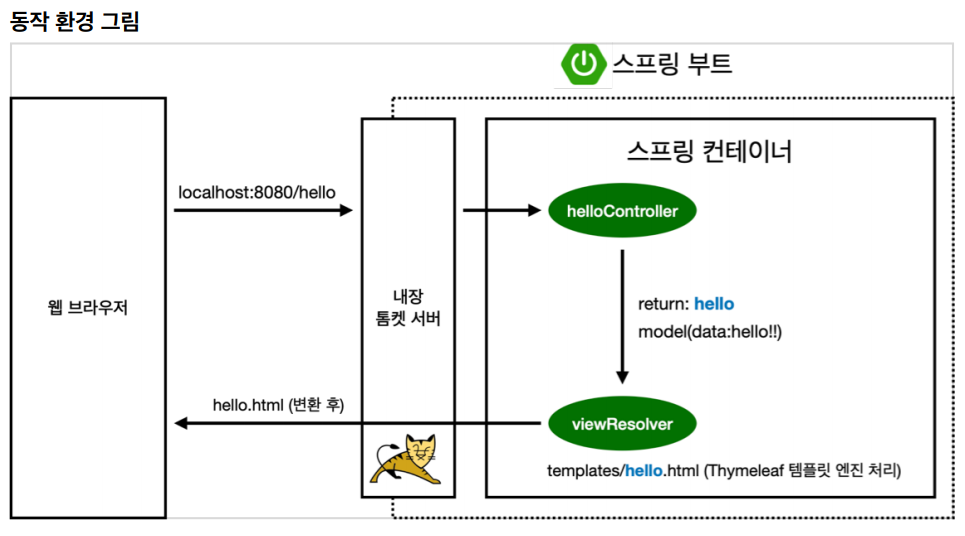
@Controller -> 이거 annotation 적어줘야함

@GetMapping -> 이게 사이트 주소 매핑해주는 역할

@GetMapping(“asdf”) 로 하면 localhost:8080/asdf 와 연결됨

return “hello” 는 templates/hello.html 을 찾을 수 있도록 함.

<참고 그림>



# 빌드 및 실행

1. jar 배포 및 실행

local 에서 cmd 실행 후 hello-spring 에서

(window) ./gradlew.bat build

(linux) ./gradlew build

실행하면 빌드될거고, build/libs/hello-spring-0.0.1-SNAPSHOT.jar 가 생성됨.

그러면

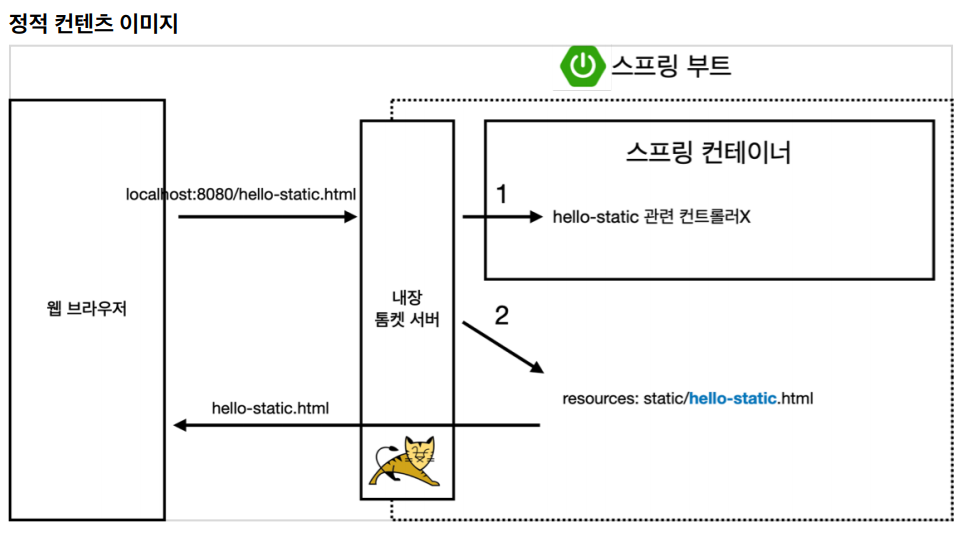
java -jar hello-spring-0.0.1-SNAPSHOT.jar 하면 실행되어서 localhost:8080 가면 페이지 나올 것.

**=> 따라서 배포 시 .jar 파일만 서버에 넣고 서버에서 이 파일 실행시키면 됨.**

# 분류

1. 정적 컨텐츠

파일을 그대로 전달하는 것



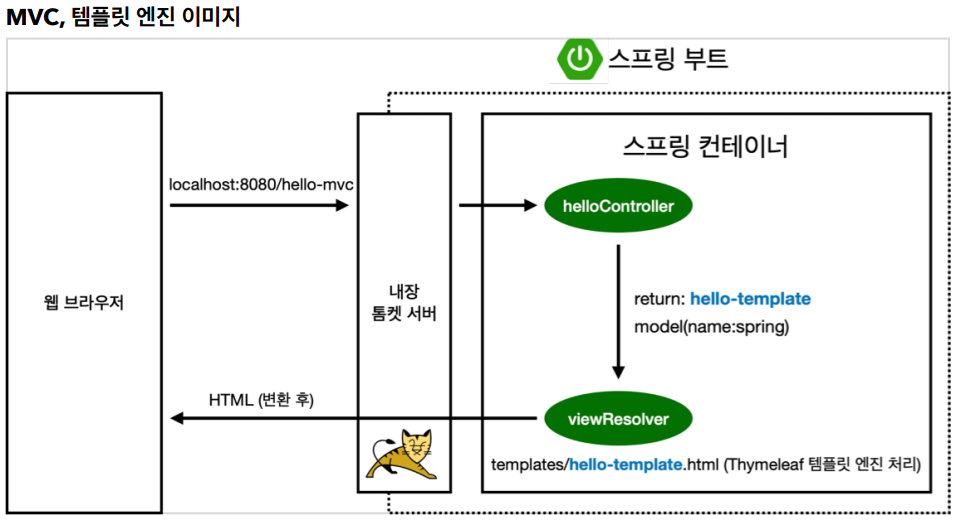
컨트롤러 탐색이 우선순위

없으면 static에서 탐색

1. MVC (Model View Controller)

html 을 뭔가 프로그래밍해서 변경해서 동적으로 전달하는 것

* 1. controller 는 내부 처리에 집중, view 는 화면에 보여지는 것에 집중



1. API

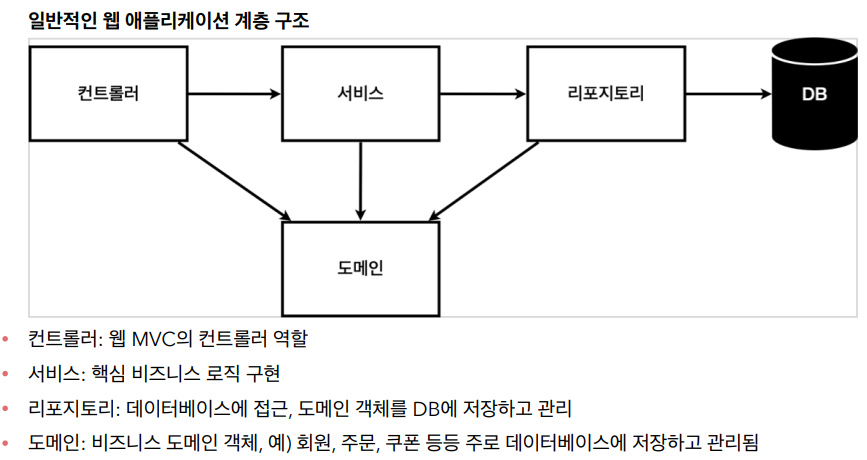
서로 통신하는 것. 어쨌든 data 전달. 이걸 front로 보내서 html을 만들던가 서버로 보내서 뭔가 처리를 하던가 등등.

1. 코드 (정적 컨텐츠 / MVC / API)

|  |
| --- |
| package LWJ.hellospring.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.ResponseBody;  @Controller  public class HelloController {  @GetMapping("hello") // localhost:8080/hello 에서의 hello 를 의미함.  public String hello(Model model) {  model.addAttribute("data", "hello!!");  return "hello"; // templates/hello.html 를 찾아감.  }  @GetMapping("hello-mvc")  public String helloMVC(@RequestParam("name") String name, Model model) {  // RequestParam"name" 으로 받는건 주소에서 들어옴.  // localhost:8080/hello-mvc?name=LWJ => 여기에서 ?name=LWJ 부분.  // 이 name(LWJ) 을 동일하게 value에 담아서 model에 담아서 보냄  model.addAttribute("name", name);  return "hello-template";  }  @GetMapping("hello-string")  @ResponseBody // html에 body 부분에 이걸 직접 넣어주겠다는 의미  public String helloString(@RequestParam("name") String name) {  return "hello " + name;  // 이건 view 이런게 없음. 그냥 문자가 그대로 내려감.  // 그래서 소스보기 하면 html 같은거 없이 진짜 그냥 그대로 내려감. (소스보기 하면 그냥 hello LWJ 로 나올 것)  }  @GetMapping("hello-api")  @ResponseBody // 객체를 반환하면 기본으로 json으로 반환함.  public Hello helloApi(@RequestParam("name") String name) {  Hello hello = new Hello();  hello.setName(name);  return hello;  }  static class Hello {  private String name;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  }  } |

# 회원 관리 예제

1. 구현



<https://github.com/wjlees/LWJ/tree/main/SPRING/hello-spring> (구현 및 테스트코드)

<https://github.com/wjlees/LWJ/commit/5f84c9ec3e4d7a3741a2ce4e363e3280e0b93d06>

Optional 은 null 일 가능성이 있을 때 Optional 로 감싸주기 위해서 붙임

Optional<Member> result = memberRepository.findByName(member.getName());

이렇게 하면 결과값이 null 일 수 있음.

이게 결과값을 Optional 로 한번 감싸주는게 되는데, 그럼으로써 Optional result 의 많은 함수들을 사용할 수 있게 된다.

if (result != null) {~~} 이 아니라, result.ifPresent( m->{~~}); 로 사용 가능해진다.

result.get() 으로 꺼내는건 optional 으로 감싸져있는걸 꺼내는 것.

최종적으로는

memberRepository.findByName(member.getName())

.ifPresent( m -> {

throw new IllegalStateException(“~!”);

});

로 한다.

1. 테스트 코드 작성

테스트 코드는 src/test 아래에 작성하고, naming 규칙은 패키지의 경우 동일하게, 클래스의 경우 동일 + Test 를 붙인다.

보통 assertThat 을 이용하여 확인한다.

Assertions.assertThat(result).isEqualTo(member1);

같으면 그냥 넘어갈거고 다르면 뭔가를 띄울 것.

[ Ctrl + Shift + t ] : 해당 코드에 맞는 테스트 코드 틀 자동 생성

//given

//when

//then

이 주석부터 달고, 이거 기반으로 채우면 보기 쉬움

1. Throw Exception의 Test

IllegalStateException e = assertThrows(IllegalStateException.class, () -> memberService.join(member2));

Assertions.assertThat(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");

1. 의존 관리 - 컴포넌트 스캔 방식

@Controller 적어주면 Spring Container 에서 관리함

이걸 스프링 빈으로 관리된다고 함

1. @Controller, @Service, @Repository

해당 클래스가 @~~ 라고 Spring 에게 알려줌.

안붙이면 그냥 순수 자바 클래스인데, 붙여주면 Spring에서 관리함

1. @Autowired

Controller 내에서 Service를 사용한다고 하면,

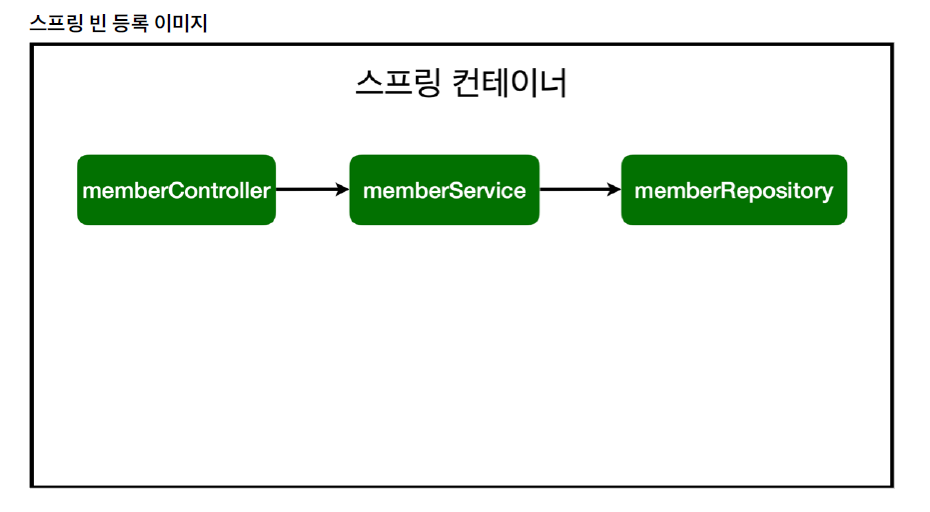
Service 변수를 매번 new 로 생성하지 않고 service 에서 가져올거면

service 변수를 constructor 로 받아서 넣고, Autowired로 연결한다.

이렇게 하면 Dependency가 생기게 된다.

=> Dependency Injection (DI)

Service가 있고, 여기껄 받아서 Controller에서 사용함.



|  |
| --- |
| <MemberController.java>  **@Controller**  public class MemberController {  private final MemberService memberService;  **@Autowired**  public MemberController(MemberService memberService) {  this.memberService = memberService;  }  }  <MemberService.java>  **@Service**  public class MemberService {  private final MemberRepository memberRepository;  **@Autowired**  public MemberService(MemberRepository memberRepository) {  this.memberRepository = memberRepository;  }  <MemoryMemberRepository>  **@Repository**  public class MemoryMemberRepository implements MemberRepository{  } |

이게 가능한 범위는 @SpringBootApplication 의 package 아래.

(@ComponentScan)

Autowired 는 bean 등록이 되어서 관리되고 있어야 동작함.

참고 : 생성자가 하나만 있고, bean으로 등록되면 Autowired 생략할 수 있음.

1. 의존관계 - 자바 코드

|  |
| --- |
| <SpringConfig.java>  **@Configuration**  public class SpringConfig {  **@Bean**  public MemberService memberService() {  return new MemberService(memberRepository());  }  **@Bean**  public MemberRepository memberRepository() {  return new MemoryMemberRepository();  }  } |

이렇게 bean 등록 및 의존관계 설정 가능함.

1. GetMapping, PostMapping

html에서 method = “post” 로 했을 때, PostMapping 된 함수가 호출된다.

<form action="/members/new" method="post">

get 은 보통 조회할 때 많이 쓰고, post는 보통 등록할 때 많이 쓴다.

members/new 로 주소 바로 접근했을 때는 GetMapping 으로 갔을거고,

@GetMapping("/members/new")

public String createForm() {

return "members/createMemberForm";

}

에 따라서 template 아래의 members/CreateMemberForm.html 에 접근했을 것.

해당 페이지에서 등록을 누르면 그건 method=”post” 로 되어있기 때문에 PostMapping 의 함수 호출됨.

ex. <https://dhlottery.co.kr/gameResult.do?method=byWin> 에서 회차별 당첨정보 볼 때 postmapping 되어있음.

# Layer - Controller, Service, Repository

1. Service Layer

Service Layer의 역할이 뭘까 - 핵심 로직.

Controller는 사용자와의 End point, Repository는 DB와의 End point

이 두 접점외에 내부 처리는 모두 Service 이어야 한다고 생각.

1. 참고 링크

<https://velog.io/@sumusb/Spring-Service-Layer%EC%97%90-%EB%8C%80%ED%95%9C-%EA%B3%A0%EC%B0%B0>

# 스프링 DB 접근

1. H2 설치

h2database.com 가서 다운받고 설치하면 됨.

H2/bin 아래에 h2.bat 실행하면 창이 뜰거고, 그게 안꺼짐.

그 상태에서 트레이에 나올거고, 우클릭후 status 클릭하면 H2 Console URL 뜸

이거 전체 블록 지정해서 복사하고 크롬에서 열면 열릴거고, 실행하면 test.mv.db 만들어짐.

~ 대신 내가 원하는 위치 넣으면 됨.

지금 이건 파일로 접근하는거고, 그러면 웹콘솔, 어플리케이션 동시접근하다가 오류 발생할 수 있음.

그래서 이렇게 test.mv.db 파일 만들고나서는 ~ 앞에 tcp://localhost/ 를 붙여주면 파일에 바로 접근하는게 아니라 소켓을 통해서 접근하게 됨. 이렇게 해야 여러군데서 동시에 접근 가능함.

그리고 sql/ddl.sql 과 같이 DB 코드도 같이 관리해주면 좋음.

1. 순수 JDBC

JdbcMemberRepository 만들고나서,

연결 후 SpringConfig에서 MemberRepository 만 return 을 Jdbc~~로 바꿔주면 됨.

MemberRepository 는 interface 만들어놨고, 이걸 Memory로 구현했었는데, 이걸 Jdbc로 구현하고 SpringConfig에서 이걸로 갈아끼우면 되니까 편리함. 이게 OOP의 장점.

DB와 연결하려면 DataSource 라는게 필요함.

application.properties 에서

spring.datasource.url=jdbc:h2:tcp://localhost/E:/Workspace/Study/SPRING/test

spring.datasource.driver-class-name=org.h2.Driver

세팅 해놨으면 Spring에서 DataSource를 준비해놓음.

이 dataSource를 받아서 .getConnection() 하면 연결을 얻을 수 있음.

얻어서 사용한 후에는 반드시 close 해줘야함.

얻고 - 사용하고 - 놓아주고 / 얻으면 잡고있는듯함.

conn에 prepareStatement 하고, 여기에 sql string 넣고, executeUpdate 하면 됨.

1. Spring 사용 및 DB 사용 테스트

@SpringBootTest 를 적으면 Spring 실제로 올려서 테스트 할 수 있음.

@Transactional 적으면 DB에 쿼리 다 날리고 커밋은 안하고 롤백함.

DB에는 commit을 해야 실제 데이터에 반영이 됨. 이를 이용해서 테스트 시 쿼리는 다 날리고 이것저것 하지만 commit 안하고 rollback 해버리면 DB는 다시 돌아옴. AfterEach 로 다 하는 대신 @Transactional 적어주면 이렇게 해줌.

1. JDBC Template

connection 얻고 끊고 등등 정해져있는 행동들을 다 알아서 해주도록 함.

1. JPA / Spring Data Jpa

ORM(Object Relational Mapping)

Spring Data Jpa 는 인터페이스만으로 끝낼 수 있음. Naming 규칙으로 가능. 하지만 모든걸 커버하지는 못할 수 있기 때문에 JPA 기본 베이스는 있어야함.

# AOP

# Crawling

1. Setting

build.gradle에서 jsoup 을 dependency implementation 해야함.

|  |
| --- |
| dependencies {  implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  implementation 'org.springframework.boot:spring-boot-starter-web'  testImplementation 'org.springframework.boot:spring-boot-starter-test'  **implementation 'org.jsoup:jsoup:1.13.1'**  } |

넣고 나서 gradle sync 하면 받아짐.

1. 받아오기

(Android 개발 시 했던것과 동일)

|  |
| --- |
| public class Crawling {  private Document web;  private Document mobile;  public Crawling() throws IOException {  this.web = Jsoup.connect("https://dhlottery.co.kr/common.do?method=main").get();  this.mobile = Jsoup.connect("https://m.dhlottery.co.kr/common.do?method=main").get();  }  public Document getWeb() {  return web;  }  public Document getMobile() {  return mobile;  }  } |

Jsoup.connect(url).get() 으로 받아옴

받은거 타입은 Document 타입.

get method로 구현된건 주소 그대로 가져오면 되는데,

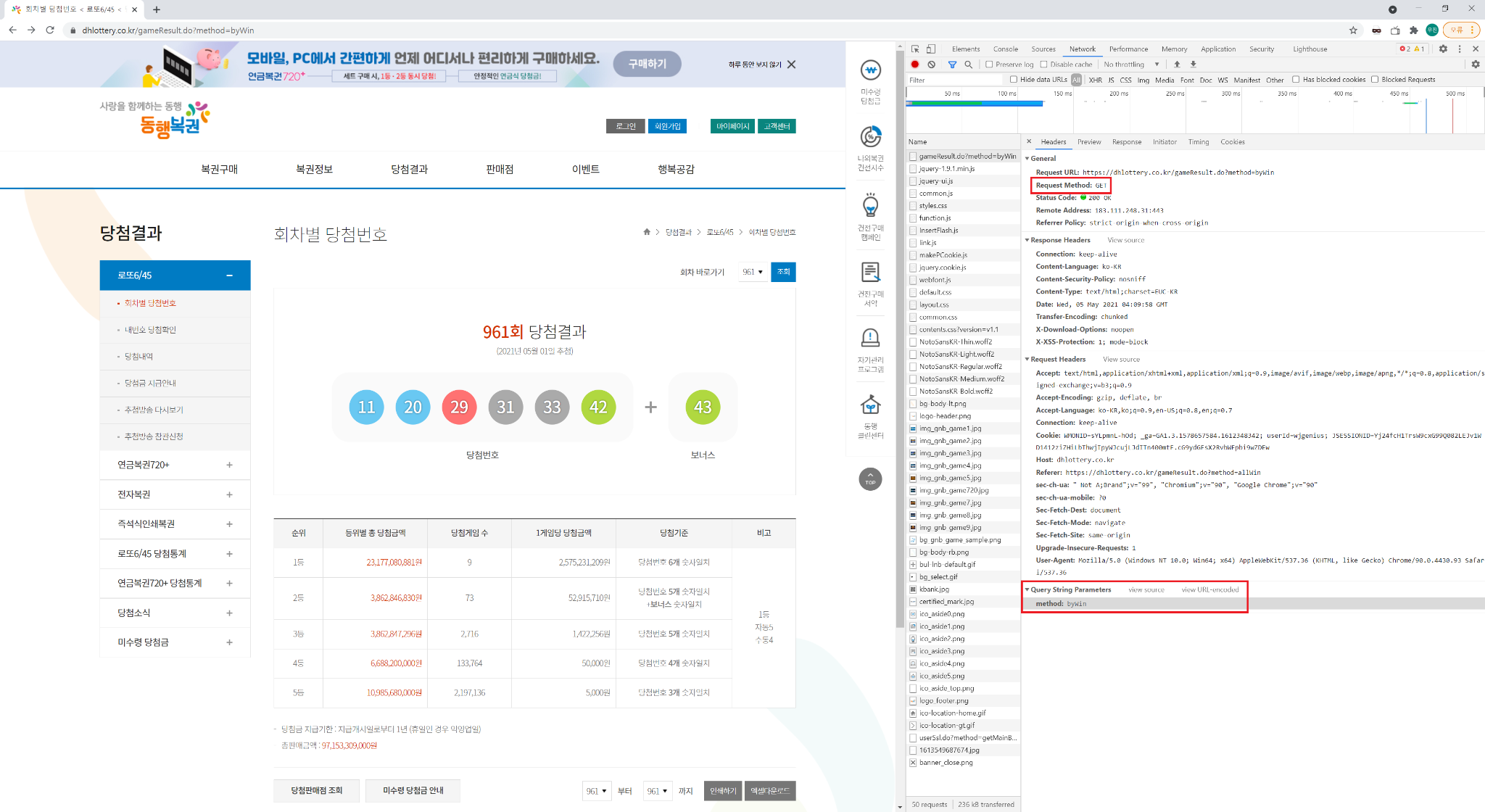
post method 로 구현된건 주소가 그대로라 못가져옴…

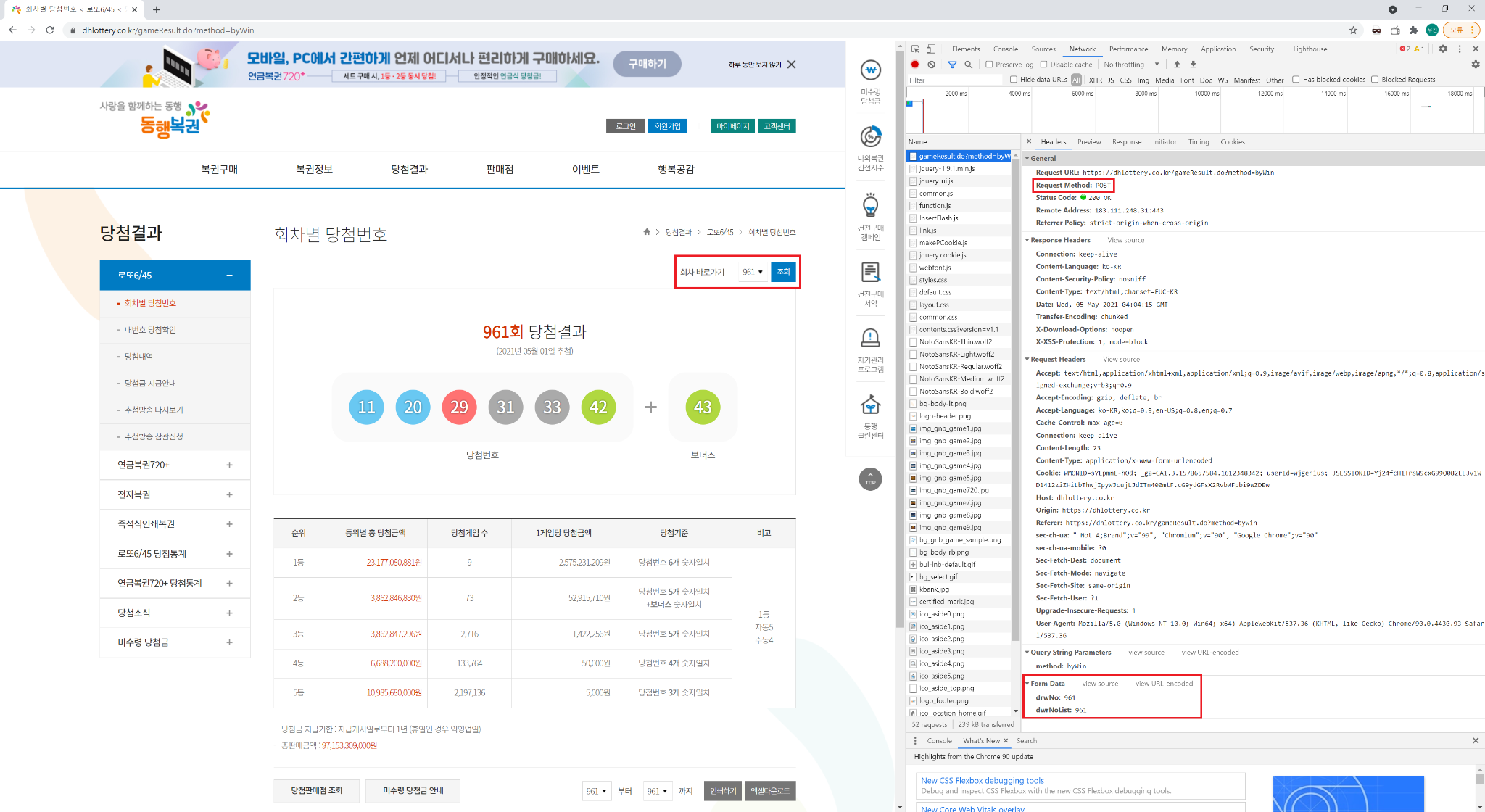
1. post method 로 구현된 부분 크롤링해오기

<https://dhlottery.co.kr/gameResult.do?method=byWin>

여기에서 회차별 당첨번호가 post method로 구현되어 1회 클릭해도 주소는 그대로다.

(처음에는 Get 으로 뜰거고, 조회 버튼 한번 누르고나면 Post 로 가져올테니까 Post로 바뀜)





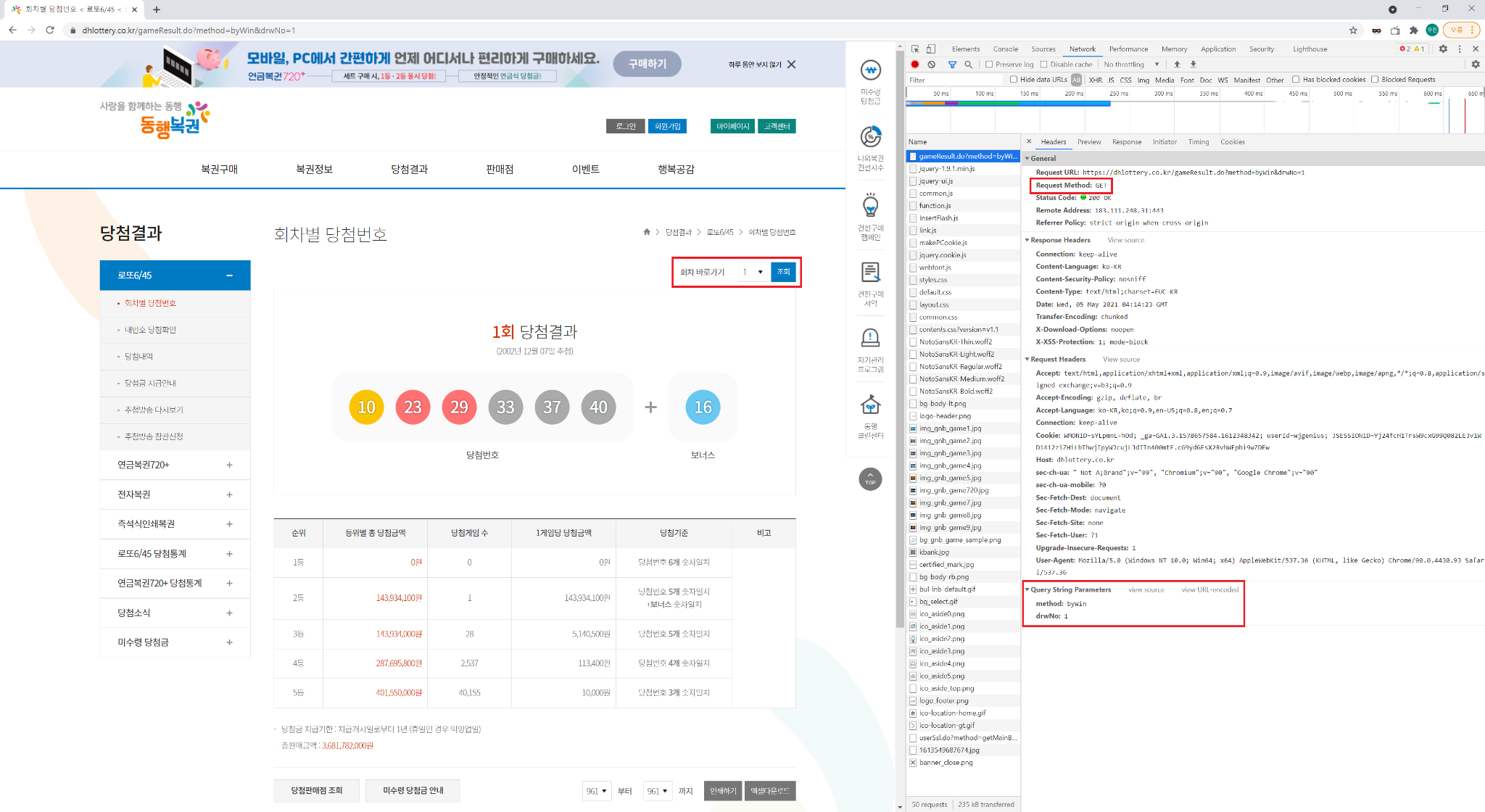
F12에서 Network 가장 처음꺼 보면 Post 방식임을 알 수 있음

Post로 나왔다면, From Data 부분이 생기는데, **이게 parameter 인 듯함.**

이걸 &로 붙여서

<https://dhlottery.co.kr/gameResult.do?method=byWin&drwNo=1>

으로 접근하면 1회 당첨결과가 나옴.



그 상태에서 다시 F12 보면 주소에 넣어서 접근했으니까 Get 방식으로 나옴.

**애초에 페이지 개발할 때 Get/Post 두 방식 모두 접근되도록 코딩해서 가능한건지는 잘 모르겠음.**

Query String Parameter 부분에 drwNo 가 추가됨.

이걸로 Crawling 가능

# HTTPS 설정

1. 배경:

Android App 에서 custom Server 접근하려고 하면 http 접근이라 접근이 불가능하다.

|  |
| --- |
| E/AndroidRuntime: FATAL EXCEPTION: Thread-5  Process: com.example.emptyapp, PID: 21203  java.io.IOException: Cleartext HTTP traffic to 152.70.248.64 not permitted  at com.android.okhttp.HttpHandler$CleartextURLFilter.checkURLPermitted(HttpHandler.java:127)  at com.android.okhttp.internal.huc.HttpURLConnectionImpl.execute(HttpURLConnectionImpl.java:462)  at com.android.okhttp.internal.huc.HttpURLConnectionImpl.connect(HttpURLConnectionImpl.java:131) |

1. 참고 링크:

<https://ayoteralab.tistory.com/entry/Spring-Boot-24-https-TLS-SSL-%EC%A0%81%EC%9A%A9%ED%95%98%EA%B8%B0>

1. keytool 위치(local):

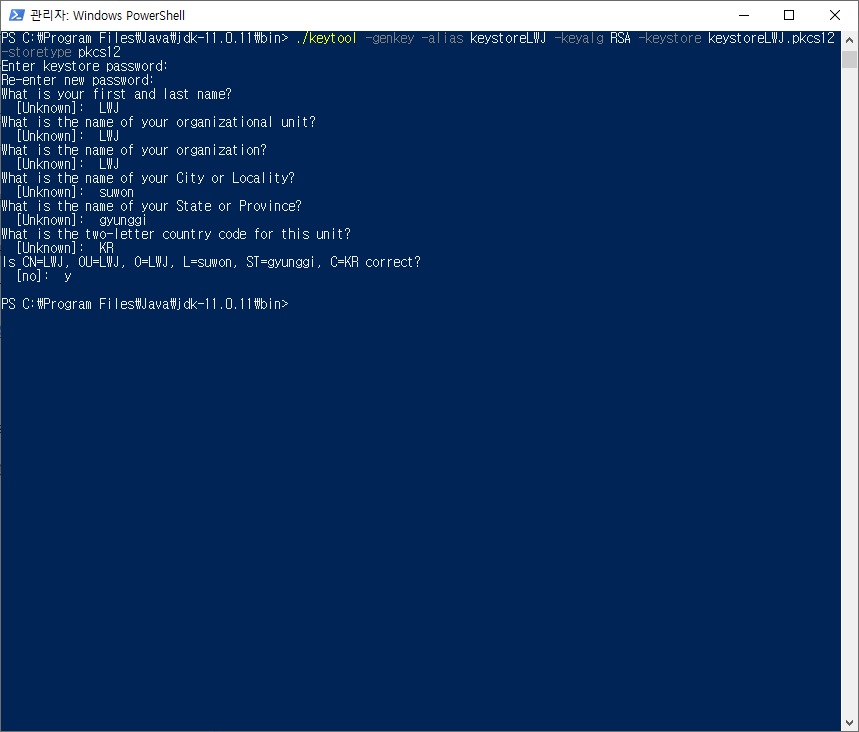
C:\Program Files\Java\jdk-11.0.11\bin

1. key 저장소(keystore) 생성

./keytool -genkey -alias [keystore 별칭] -keyalg RSA -keystore [keystore 파일이름] -storetype pkcs12

./keytool -genkey -alias keystoreLWJ -keyalg RSA -keystore keystoreLWJ.pkcs12 -storetype pkcs12

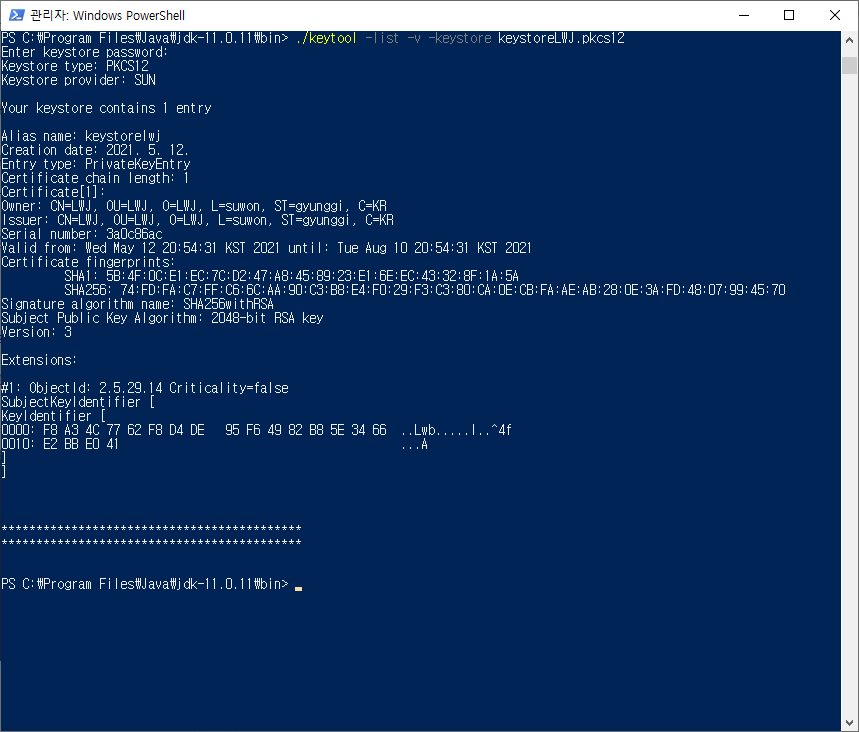
비밀번호 keystoreLWJ



1. keystore 생성 확인

./keytool -list -v -keystore [keystore 파일이름]

./keytool -list -v -keystore keystoreLWJ.pkcs12

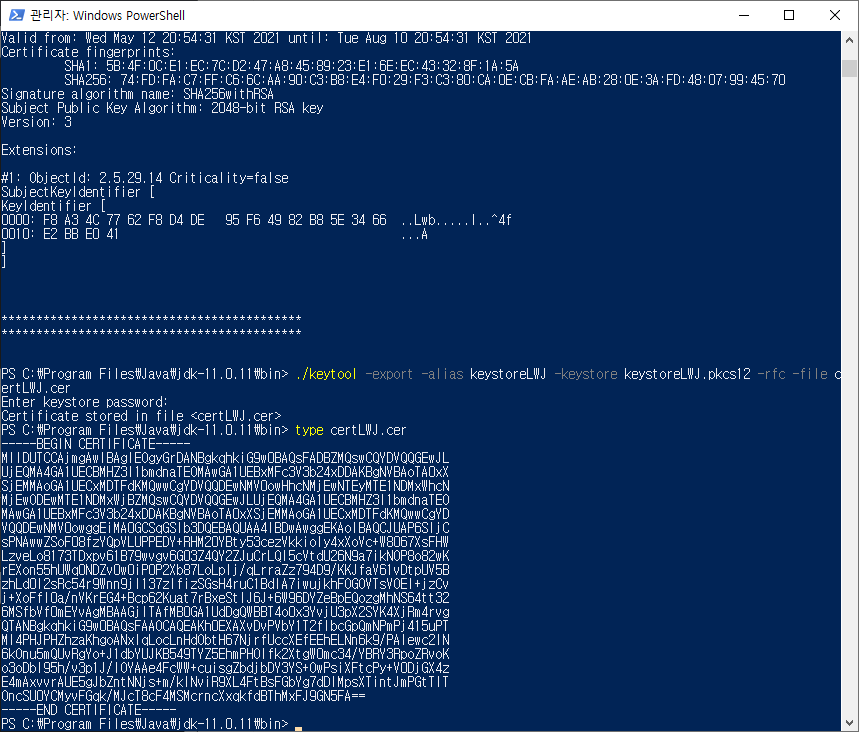


1. keystore 에서 인증서 추출

./keytool -export -alias [keystore 별칭] -keystore [keystore 파일이름] -rfc -file [인증서파일이름]

./keytool -export -alias keystoreLWJ -keystore keystoreLWJ.pkcs12 -rfc -file certLWJ.cer

확인은 type certLWJ.cer



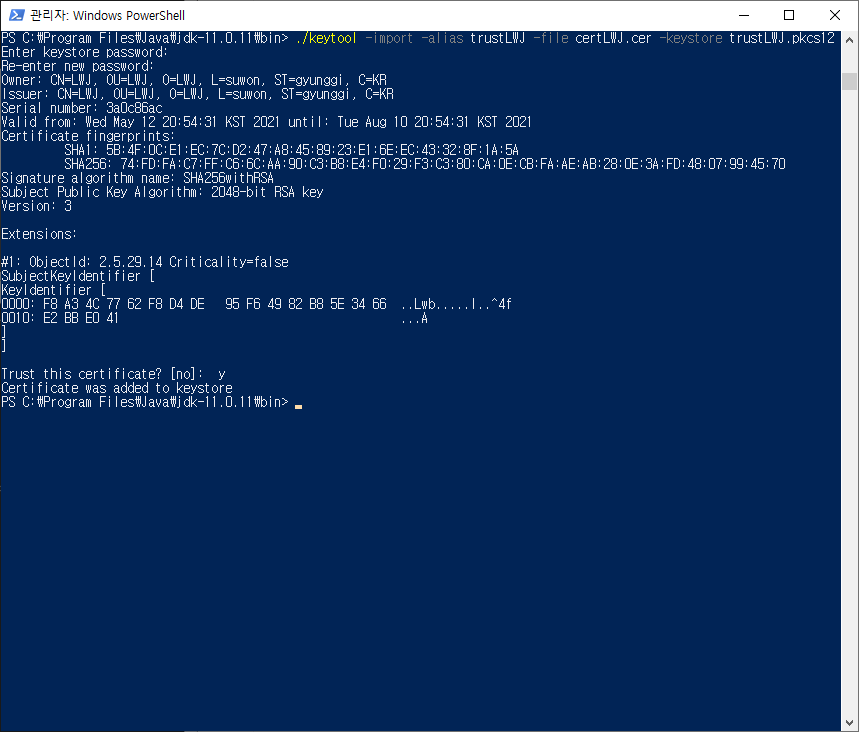
1. trust store 제작

./keytool -import -alias [trust store 별칭] -file [인증서파일이름] -keystore [trust store 파일이름]

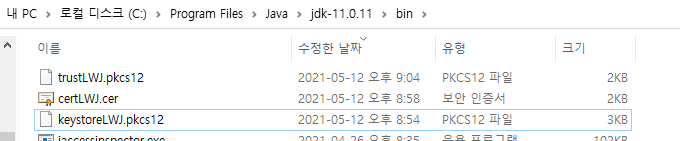
./keytool -import -alias trustLWJ -file certLWJ.cer -keystore trustLWJ.pkcs12

비밀번호 trustLWJ

(만약 cer가 인증기관 인증서라면... -trustcacerts 옵션을 추가)



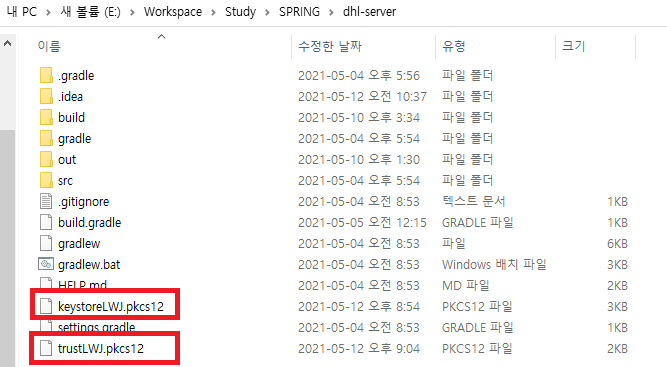
1. 생성된 파일 확인



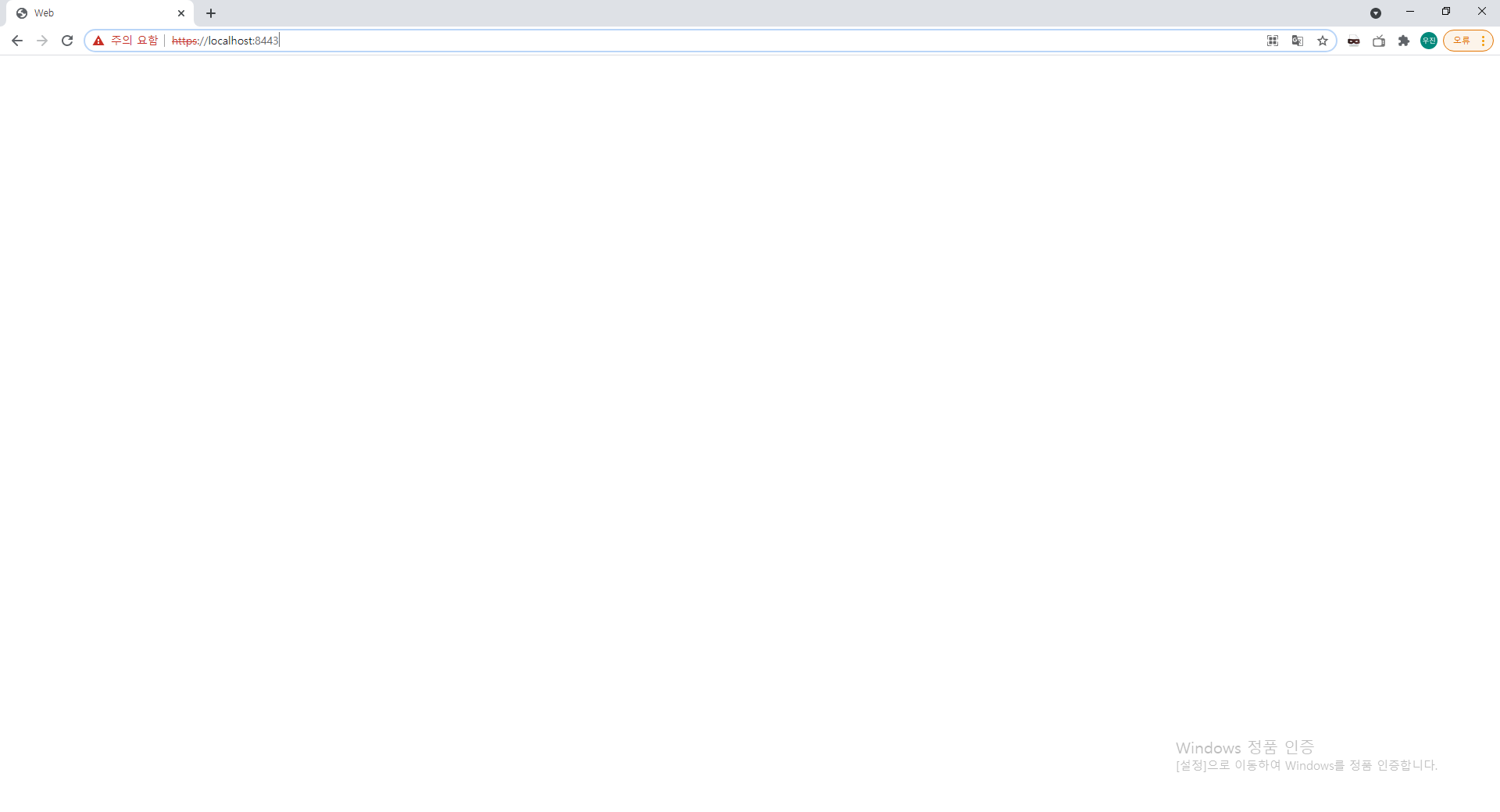
1. application.properties 에 추가

|  |
| --- |
| server.port=8443  server.ssl.enabled=true  server.ssl.key-store-type=PKCS12  server.ssl.key-store=keystoreLWJ.pkcs12  server.ssl.key-store-password=keystoreLWJ  server.ssl.trust-store=trustLWJ.pkcs12  server.ssl.trust-store-password=trustLWJ |

1. 실제 폴더에 추가



1. https로 접속 확인



1. 배포 시 .jar 파일 뿐만 아니라 keystore 파일들까지 같이 배포해야함.

