//190102

/서브릿은 뒤에 있어서 안 나오지만 spring이 그거까지 포함해서 돌려 줌.

/지금까지 필요한 것 자바 + DB(여러 DB 시스템 중 아무거나, 예를 들어 우리는 Oracle) + Mybatis(그 둘을 이어주는) + 웹 view (HTML + CSS + JavaScript) + Servlet (자바 기반 서버측 구성 프로그램) + JSP(이것도 서버측, 다 조합한 거, 수많은 언어가 한 데 조합)

/이러한 수많은 조합들이 모여서 하나의 웹을 완성하는 것. 그렇다 보니 드라이러도 여러 개고, 버전이 중간에 하나가 바뀌면 뒤에 다 도미노처럼 다 안 되고, 이렇게 됨. 그래서 드라이버 관리도 일임. 그 드라이버 파일들 다 들고 다녀야 해. 따라서 그런 드라이버들을 설정만 하면 그에 맞춰서 프로젝트에 넣어 주는 기능이 있음.

/필요한 드라이버나 파일 다운 받아서 이클립스에 넣든가,

그냥 STS(spring tool suite) 받아 쓰든가. STS는 이클립스랑 똑같은데 드라이버들 설정 구현해놓은

/Mybatis는 DB와 Java 연결해놓는 거

+)Java

+)Oracle

+) HTML / CSS / JavaScript

+) Servlet + Spring / JSP

+) Tomcat

/spring 설정 확인 - compiler 1.8 맞지 / Tomcat 8.0v 맞지 / server - runtime environment 에서 tomcat 압출 푼 폴더

/새 웹 프로젝트 - 즉, 새로운 사이트 하나를 만든다고

spring legacy project 선택하고, 이걸 처음부터 만들기에는 너무 다운 받아야 할 것도 많고

그래. 그래서 템플릿 미리 짜 놓은 거 바탕으로 만듦. spring MVC project

/여기부터는 패키지 이름을 막 지을 수 없어. 무조건 세 자리 이상. 보통은 우리가 쓰는 웹 주소를 거꾸로 돌려서 한칸씩. ex) web1.soc.global -> global.soc.web1 / 베이스 패키지

/결국엔 우리가 짜 놓은 프로그래밍이 웹에 설치가 돼 있어야, 사용자한테 보여지는 거..

/요즘엔 자바 코드로 설정 넣는 게 유행이기는 하지만 거의 기본적으로 xml로 설정 만듦.

+) 상단은 그냥 아파치 톰캣 주소랑 버전 / 우리가 정한 프로젝트 경로, 이름 / 중요한 건

properties - java 버전 / spring framwork 버전 /

/ dependency 부터 /dependency 까지가 한 단위. 드라이버 하나를 받는 코딩

/ 프로젝트 우클릭 - properties - web project settings - context root

이거가 숏컷

/ 포트 8888이 사용 중이다? 안쓰는 거 확인하고 우리가 만든 건데? 이런 경우는

이전에 우리가 사용할 때 톰캣이 정상 종료가 안 돼서 아직까지 사용하는 걸로 인식 됌.

컴퓨터 재부팅하는 게 답.

/life cycle exception

+) C:\Users\user\.m2\repository 이 폴더에는 이 프로그램 돌리는 데에 필요한

여러가지 드라이버 or 라이브러리를 자동으로 다운 받아 놓은 걸 모아놓은 곳.

오류 걸렸을 때 이거를 삭제해서 새로 실행하면 다시 처음부터 받아 옴.

/ 사용자 화면에 (우리가 보내는 거에) 한글이 깨진다 -> jsp 파일 page에서 page encoding 추가

사용자로부터 받아오는 한글이 깨진다 -> web.xml 에서 processes application requests 에 <filter> ~ </filter mapper> 까지 복붙해서 web xml 에 넣기.

/xml 만지면 톰캣을 재실행 해야. 끈 후 다시

/실제 파일명 말고 가상 주소로만

/ spring 관련 설정 파일 : root-context.xml + servel-context.xml

/ 경로 잘 생각해야 함. - src 에 webapp 이라는 폴더는 사실 없음

/ src/main/java 에서 global.sec.web1.java 에서, 우리가 브라우저로 검색창에

입력 할 때 쓰는 숏컷

190103

/ Maven Dependencies 이게 뭐냐면 pom.xml 에서 버전 정보 입력해놓은 것들에 맞게 맞춰서

Maven Dependencies 에 다운받아서 모아 놓음. 여기 없으면 뭔가 잘못 된 거. pom에

빠뜨렸다거나 등등. 만약 다 됐는데 멀쩡한데 작동이 안 된다 -> 이 레포지토리를 지우고 다시

/ web.xml은 web.inf 폴더 바로 아래에 있음. web 설정이 들어 있음. 요청이나 작업을

spring 통해서 수행하라는 설정 + 필터 같은 거

/servlet-context.xml => "/web-inf/views/" 에다가 ".jsp"에서 찾아라. 설정을. 그런 듯

/refactor 로 rename 했으면 자바에서 file 이름과 그 관련된 호출 정의 등 name만 바뀔 뿐이지

spring 전체에 지원해주지는 않음. 우리가 일일이 바꿔야 함.

+) servlet-context.xml / pom.xml / log4j.xml

+) pom에서 group id 랑 artifact id 등은 안 바꿔줘도 되긴 함. 실행은 됨. 뭐지?

/이제 확인용 디버깅용 출력을 System.out.print("여기서 오류"); 이런 거 쓰지 말게. 왜냐면 나중에

확인하고 나서 작동될 때 다 일일이 지워야 해. 뭐 할 수는 있지만, 일은 그렇게 하는 게 아니지.

+) HomeController.java 에서 logger 가 사용자에게 출력되는 객체임.

/spring 말고 주된 설정 파일 -> resources

+) log4j.xml 도 누가 만들어 놓은 프레임워크. Mybatis 처럼. log4j는 출력 도와주는 프레임워크

+) logger name 에서 패키지 이름 제대로 됐는지?

+) level 은 어느 정도 단계까지 출력을 해줄지? info는 다 나옴, debug는 우리가 한번 볼라고,

warning 은 진짜 경고. error 는

완전 에러.

/MVC (Model - View - Controller) 의미

+) 각 클래스, 각 부품, 각 객체, 각 조각이 맡은 역할. 하는 역할. 그것을 구상하는 게

디자인 패턴이라고 함.

+) view 는 jsp 담당. 화면에 뭐 기사 내용이 뿌려지거나, 가입할 때 우리가 입력하거나 등

+) controller 는 자바 클래스가 담당. 요청이 들어오거나 나가거나. 이거 보고 싶어요, 이거 입력했어요, 서버로 보내거나 서버로부터 받음. 이러한 흐름을 컨트롤

+) view와 controller 가 주거니 받거니.

+) model 은. 실질적 정보 처리. view에서 사용자가 뭐 보고, 뭐 요청하면, controller는

그 요청에 알맞는, 적합한, 그 일을 처리할 수 있는 녀석에게 요청을 처리하라고 보냄..

+) spring은 controller 가 틀이 잡혀 있음.

+) 실제로 길고 복잡한 코딩이 @controller 이런 어노테이션( annotation) 단축키 비슷한 걸로 축약.

/ 어노테이션( annotation)

+) 점점 이런 어노테이션, 축약어로 일처리를 해내는 것들이 늘어나고 있음.

+) 우리가 가장 기본적으로 외워야 하는 어노테이션. @Controller / @RequestMapping

+) 시작하면 일단 이 annotation 붙은 것들을 다 읽어 놓음.

+) 요청 들어오면 일단 @controller 붙은 클래스들 다 뒤져 봄

+) @RequestMapping value = "요청 받는 주소" , method = RequestMethod.GET or RequestMethod.POST

+) return testview; 하면, servlet.xml 에서 우리가 미리 작성해 놓은 prefix를 앞에 붙이고,

suffix를 뒤에 붙여서, /web-inf/views/testview.jsp 를 실행함.

+) 이때, testview.jsp 라는 파일이 src/main/webapp/WEB-INF/views/ 에 있어야 함.

/우리가 혼자 서버와 클라이언트를 겸용하다보니, 자꾸 망각하는 사실이 있는데,

서버가 먼저 static 처럼 있고, 그 서버에 client가 뭐좀 보여ㅊ달라고 요청. 그럼 서버는

그 요청에 맞는 처리를 해서, 그 결과를 다시 client에 보내줌. 그럼 client는 그 결과를

받아서, 브라우져 프로그램이 해독해서 화면에 view로 가독성 있게 띄워줌.

사용자가 뭘 클릭했을 때 (어떤 요청을 보냈을 때) 그 요청과 그 정보들을 가지고,

어디로 보내고 그걸 바탕으로 뭘 만들고, 뭘 어디로 보내고, 어디서 뭐를 받고, 그런 전체적인

흐름을 항상 염두.

/ get / post

+) post는 폼 통해서 주거나 받을 수밖에 없음. localhost:8888/web2/test1 이렇게 부른 다는 건, post

/경로 세 가지 생각해야.

+) 사용자가 보는 경로(우리가 가상으로 붙인 이름, 주소).

http://localhost:8888/web2/test1 이라고 해놓으면 test1이라는 이름을 가진 메소드가 실행되는 거지.

+) 자바에서는 http://localhost:8888/web2/

html에서는 http://localhost:8888/web2/test1

+) 그냥 RequestMapping ( value = "여기에") 들어갈 주소는, 클라이언트 입장에서 주소.

클라이언트가 요청할 때 입력하는 주소.

그리고 그 주소로 요청이 오면 실행할 녀석이 바로 그 아래 메소드. 즉,

사용자가 그 메소드를 실행하려면 브라우저 주소창에 뭐라고 쳐야 하는지, 주소.

한편, 클라이언트가 그 주소를 입력해서 그 메소드가 실행되면, 그 메소드가 뱉어내는

return 값은, 서버 입장. 그래서 어떤 파일을 클라이언트한테 보내줄지. 서버 컴퓨터에서

그 파일이 어디에 있는지, 그 경로. 서버 컴퓨터에서 해당 파일의 파일 경로.

/views 밑에는 서브 폴더 얼마만큼이든 만들어도 됨.

+) views 밑에 test 라는 폴더에 testview3.jsp를 만들었으면 value = "/test3" 이렇게 만들고 (이러면 http://localhost:8888/web2/test3만 입력해도 접속 가능) return 값은 test/testview3 이렇게 (앞과 뒤에 자동으로 prefix와 suffix 붙어서 실행 됨)

//20190104

/새로운 기능을 가지는 컨트롤러 만들려면?

+) 일단 global.sesoc.web3는 그대로 두고, 그 아래에 . 더 붙여서 패키지명 만듦.

global.sesoc.web3.controller 이런 식으로. 이름 막 붙이면 작동 안 됨.

/ @Controller 는 미리 다 읽어 놔. http://localhost:8888/web3/test5 라고 치면(요청을 보내면),

미리 읽어 놓은 Controller 들의 @RequestMapping 중, 요청값(value)이 test5인 애를 찾아서

해당 메소드를 실행함. 그 메소드의 리턴값이 주는 jsp를 응답으로 보내 줌. 화면에 띄워 줌.

/for 구문이나 while 구문 등, 함수(function) 구문 { } 안에서 선언한 변수는 그 함수 밖에서는

못 씀. 다른 함수 입장에서 보자면 없는 거나 마찬가지. 함수의 구문 { } 밖에서

선언한 거(메소드에서 선언한 거)는 다른 모든 함수 구문 { } 안에서도 쓸 수 있음. 마찬가지로,

메소드 구문 { } 안에서 선언한 거는 그 메소드 밖에서는 쓸 수 없음.

다른 메소드 입장에서 보면 그 메소드 안에 있는 변수는 없는 거나 마찬가지.

하지만 메소드 구문 { } 밖에서 선언한 거(클래스에서 선언한 것)는 다른 모든 메소드 구문 { }

안에서도 쓸 수 있음.

여기서 한 단계 더 나아가 보자. 클래스 구문 { } 안에서 선언한 변수는, 그 클래스 밖에서는

쓸 수 없음. 다른 클래스 입장에서 보면 없는 거나 마찬가지. 지금까지대로라면, 클래스 밖에서

선언한 변수를 다른 모든 클래스들이 쓸 수 있어야 하는 건데, 클래스 밖은 없거든. 따라서

클래스가 다른 클래스에 있는 변수에 접근하고 싶을 때 쓰는 게 호출이지. 그리고 그 호출을

public private static 등을 이용해서 접근 권한을 통제하는 거고

/RequestMapping 에 있는 value가 서로 겹치는 메소드가 하나라도 있다? 그러면 전체가 실행이

안 됨. 왜냐면 실행하기도 전에 미리 다 읽어오니까. 그 과정에서 뭔가 이상한 점이 발견되니까

걍 드러 눕는 거지.

+) 따라서 이건 어떻게 방지하냐? 위에서 주소들을 따로 string 변수에 담아서 정의해 주고,

밑에 value 에서는 그 String의 식별자만 넣어 주는 거지. 위에서는 이러한 주소를 담아 놓은

String 들의 정의들을 모아 놓으면 한 눈에 볼 수 있음.

/서버에서 클라이언트로 보내고 싶은 말. 자바에서 연산하고 실행해 봐야, 그 메소드 벗어나면

그 변수와 정보들은 사라져버리지. 이걸 보내고 싶을 때? 자바에서 jsp로 보내는 말

+) 메소드를 보면, 패러미터로 주는 게 없는데 받아 오면서 시작해? 이런 건 전부

Spring에서 주는 거

+) 값을 주고받을 때 Model 변수 필요. 이것도 Spring에서 주는 거

+) Model은 원래 자바에서 map 처럼 사용하기. 부를 이름과, 넣는 정보. 항상 이 두 요소를

한 쌍으로 하여 정보를 추가해 줌.

+)jsp 파일이랑 html이랑 뭐가 다르지? 겉으로 보기엔 똑같은 것 같은데? jsp는 사실은 java 파일.

jsp - java server page

+) 모델로 보내 준 정보들. 그 정보에 붙여 놨던 이름. 이 이름을 ${이름} 이렇게 불러 줌.

자바에서 model.addAttribute("name", "hello"); 이렇게 보내 줬으면

jsp 에서는 ${name} 이라고 치면 그 자리에 애초에 hello 라고 있었던 것처럼 됨.

+) ${ } 이런 기능을 el 이라고 함

+)서버에서는 처리, 클라이언트는 결과만, 소스보기 해도 ${name} 이런 흔적들은 안 보임

+)만약 ${fake} 이런 식으로 값이 없는 애를 호출하면 그냥 투명인간 취급. 애초에 없던 애 취급.

흔적도 없음

/ ${ } 사용 법

+)단일 정보 변수들. 예를 들어 단순히 int, double, boolean, String 등은 그냥 그 변수의

호출자로 불러주면 됨. ${name} 이렇게.

+) 근데 한 변수에 여러 개의 정보가 들어 있는 객체의 경우, 예를 들어 List , ArrayList ,

HashMap, Set 등은? 예를 들어 java에서 model.addAttribute("map", hm); 이러면

hm이라는 이름을 붙여 놓은 HashMap 에 정보들을 넣어 놓고, jsp에서 ${map) 이러면

hm에 있는 정보들 한 번에 다 출력? 그렇게 되긴 하는데 모양이 이쁘지 않아.

student5 의 kor 점수, eng 점수, mat 점수 등 따로 출력하고 싶으면?

${hm.e1} 이러면 hm.get(e1); 으로 실행. ${student5.kor} 이러면 student5.getKor(); 자동 실행.

/jsp 파일의 정체

+)jsp는 디자이너가 디자인 해놓고 개발자가 프로그래밍 쪽만 중간중간에 끼워 넣는 식

원래는 자바 프로그램. <br> 이런 태그들은 그대로 브라우저로 보내주는 거일 뿐.

System.out.print( ); 에 넣은 것처럼.

+)우리가 jsp에 코딩 짜면, 그거에 맞는, 원래대로 제대로 java 언어로 짜 놓은 java 파일이

자동으로 만들어짐. workspace 안에 metadata에 저장됨. 그거 열어 보면 java 언어 그대로 있음.

C:\Java\workspaceforspring\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost

/반대로 클라이언트가 서버로 정보를 보낼 때

+) form 에 있는 action 이 그 경로. submit 누르면 action에 있는 경로를 요청함.

그와 동시에 form에 있는 정보들을 그 경로로 보내 줌. 정보들을 가지고 감.

단, name이 있어야만 함. 없으면 안 가져감. 서버로 가져갈 정보가 아니라면 name 도 필요 없음.

+) form 에서 넘어오는 건 사실은 전부 다 String 임. 근데 spring 에서 알아서 int나 다른 걸로

변환해 주는 거임.

+) jsp 에서 form 의 method 가 get /post 그리고 java의 RequestMapping 에서 method 가

서로 짝이 맞아야 함.

+) jsp 에서 각 태그의 name = " " 이랑, RequestMapping에 해당하는 method가

받아 오는 parameter의 변수명이랑 맞아야 함.

/ redirect는 요청 정보가 끊어진다, 주소 표시줄이 바뀐다

forward는 요청 정보가 유지 된다, 주소 표시줄도 바뀐다

. -누가 요청했는지, IP뭔지, ... 뭐 이런 저런 정보

말하자면 트랙킹, 어디 페이지에서 어디로 가고, 거기서 어디로 가고, 그런 흔적? 정보?

어디서 어디로 온 건지 경로가 유지. 이 전의 페이지에서 받아온 정보가 계속 유지 됨.

예를 들어 설문 조사할 때 여러 페이지에 걸쳐서 정보 입력할 때나, 회원가입할 때, 이 전의

페이지에서, 실명 인증하고, 이름, 주소 입력해놓고 다음 페이지에서 아이디, 비번, 메일, 이런 거

입력 받을라 하니까, 그 사이에 자꾸 정보가 초기화 돼 봐. 개빡치지. 또는 설문조사 항목이 한

500개 돼가지고, 페이지가 여러 페이지야. 그럼 이 전 페이지에서 응답한 정보가 담아서

설문조사 전체가 끝날 때까지 가져가면서 유지돼야 하잖아. 또는 장바구니 담아가면서 여러 개의

상품을 한 번에 구매할 때. 이런 경우들은, 이 전 페이지에서 받은 정보들을 다른 페이지로

이동하더라도 계속 가져가고 싶을 때, 이런 기술 중 하나가 포워딩. 근데 정보를 잃지 않고

다른 페이지까지 끌고가는 기술이, 포워딩만 있는 건 아니고, 그 크기에 따라서,

세션이거나, 쿠키거나, 파일까지 갈 수도.

+) 반대로 이 전의 기록들을 다 지워버리고 초기화 하는 경우가 redirect.

예를 들어 로그 아웃. 이거는 그 전에 누가 입력해놓은 정보들이 그대로 남아있다면

보안상 문제도 있고, 메모리 문제도 있지. 또는 회원가입이 완료되고,

본 메인 페이지로 돌아가질 때. 이런 경우들이 redirect

+) 정보 유지 기술이 포워딩, 세션, 쿠키, 파일, 등 단위로 가지고 다님

+)"redirect:/경로" 이러면 됨