**Spring공부 정리**

전채현

**스프링 설치부터 실행까지**

1. 구글에 2019-06 이클립스 검색해서 들어가면 R Packages가 있다

거기 들어가서 Eclipse IDE for Enterprise Java Developers에서 해당 운영체제 맞게 다운로드한다

eclipse-java-2019-06-R-win32-x86\_64.zip 나는 윈도우라서 이거 다운로드함

2. 그럼 C드라이브나 D드라이브 안에 spring이라는 폴더를 하나 만든다(여기에 스프링 관련 workspace나 설치 파일을 저장하려고)

(바탕화면 말고 C드라이브나 D로 넣어야 하는 이유 : 바탕화면에 다 해버리면 나중에 권한문제가 발생할 수 있고 다운로드하는 데 한글 이름이 길면 다운로드가 안되는 경우도 있다 그래서 그럼)

3. 그리고 폴더 안에 다운받았던 파일을 압축을 풀어준다.

4. 그 다음 workspace-spring이라는 폴더를 하나 더 만들어 준다 ( 스프링 프레임워크를 사용할 때 프로젝트들의 저장 공간 )

5. 그러면 압축이 풀리면 eclipse라는 폴더가 있는데 그 안에 들어가면 eclipse.exe라는 실행파일이 보인다.

6. 그걸 클릭하고 workspace를 방금 만든 workspace-spring으로 지정해 준다

7. 그리고 eclipse를 들어간다

8. 상단 메뉴바에

Window -> General -> Workspace -> Text file encoding -> other : UTF-8 (Apply 클릭)

Window -> General -> Web Browser -> Use external web browser(체크) / External web browsers -> Chrome (Apply 클릭)

Window -> General -> Server -> Runtime Environments -> Add -> Apache Tomcat v8.5 클릭하고 next ->

Tomcat installation directory (현재 톰캣이 설치되어 있는 경로 지정 : 시스템 환경변수에 설정한 경로로 설정 기억안나면 윈도우키 + R -> sysdm.cpl로 들어가서 고급 -> 환경변수로 들어가서 CATALINA\_HOME 경로로 확인)

JRE는 default JRE로 설정하고 Finish

Window -> Web -> CSS Files, HTML Files, JSP Files 세 군데의 안에 Encoding을 맨 위에 UTF-8로 변경하고 Apply클릭

(why?? UTF-8로 맞추냐?? 한글을 사용하는 경우나 다른 사람의 프로젝트를 받아오는 경우에도

인코딩 방식이 동일하지 않으면 인코딩이 깨져서 글자가 깨지는 대참사가 발생한다

그래서 UTF-8로 맞춰두면 한글은 안깨지고 다른 사람의 프로젝트의 한글도 왠만하면 깨지는 걸 막아준다)

-> 이렇게 하면 eclipse 기본 설정은 끝

9. 그리고 Servers 연결해야하는데 기본에 안보이면 Window -> Show view -> Servers 클릭해서 밖으로 빼낸다.(other에 있는 경우도 있다)

그리고 Servers에 뭔가 클릭하고 싶은게 있을텐데 클릭하면 Tomcat v8.5가 설정된다

10. 그러면 이제 스프링을 설치할건데 어디서 하냐??

11. Help -> Eclipse Marketplace(앱스토어, 구글플레이? 같은 앱을 다운 받는 느낌으로 스프링을 사용한다)

12. Eclipse Marketplace에서 sts( spring tools suite )라고 검색하면 Spring Tools 3라고 나올건데 그걸 Install한다

(여기서 개인 컴터나 노트북마다 시간차 생김)

13. 그럼 Confirm Selected Features가 나오는데 Confirm이라는 버튼을 클릭하고 넘어간다

(여기서 confirm하는 것은 spring에서 설치될 것을 확인하라는 것 -> 안에 이런게 들어가는데 승인하실거죠??)

14. 그리고 Review Licenses가 나오는데 I accept the terms of the license agreements를 클릭하고 finish

(그럼 spring이라는 앱이 다운중이다 다른 작업은 하지말자 괜히 꼬인다)

15. 하단에 Install Software 설치중 % 가 보이니까 좀 기다려

16. 중간에 Security Warning이라고 뜨면 Install anyway를 선택하고 계속 진행한다. 그럼 Certificates 가 나오는데 Select All하고 Accept selected하고 넘어간다

17. 그럼 Restart할거냐는 창이 하나 뜬다. 그럼 Restart Now를 누르고 진행한다

18. Restart를 하면 spring이 설치가 된겁니다. 그럼 이제 project를 만들어 봅시다

19. File -> new -> Project -> Spring -> Spring Legacy Project 선택

20. Project name : test4 그리고 Templates 는 맨 밑에 Spring MVC Project를 선택하고 Next 그러면 창이 하나 뜨는데 Yes하고 넘어간다

21. 그러면 top-level- package를 작성하는데 도메인의 역순으로 작성해준다고 생각하면 된다 (www.naver.com 이면 com.naver.www)

여기선 Project name을 test4라고 정했기 때문에 com.itbank.test4라고 지정하겠다 (이렇게 끝에 이름이 맞춰주는 것이 좋다 )

22. 그러면 test4라는 이름의 project 완성

23. 이제 xml파일을 잠깐 수정할거다

여기서 xml이란 ??

xml을 들여다 보면

<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>appServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

이런식으로 작성된게 대부분 일건데 이걸 쉽게 생각합시다

spring에는 크게 pom.xml, web.xml, root-context.xml, servlet-content.xml 4개가 있었죠??

이건 하나하나가 주문서라고 생각합시다. xml안에는 주문한 사항이 적혀있는거죠 ( 프로젝트를 이렇게 저렇게 만들어주세요 )

스프링 프로젝트는 메이븐 (maven) 기반 프로젝트입니다. 기억해요. 메이븐은 외부에서 라이브러리를 관리합니다.

pom.xml( project object model )은 프로젝트에 포함시킬 라이브러리를 관리하는 xml인데 쉽게 생각하면

pom.xml은 외부에서 주소(url)를 가지고 와서 라이브러리를 프로젝트에 포함시키는 주문서라고 생각합시다.

그러면 밑에 버전만 바꿔주면 주문한 내용대로 url을 보내서 이걸 가지고 와죠가 되는겁니다.

이제 그럼 pom.xml을 수정해 보겠습니다

프로젝트를 열어보면 맨 밑에 pom.xml이 있는게 보이죠??

상단에 보면

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/maven-v4\_0\_0.xsd">

이런식으로 url으로 뭔가 가져올거 같은 느낌이 들죠??

그리고 밑에 <java-version>1.6</java-version> -> <java-version>1.8</java-version>로 변경

그리고

<org.springframework-version>3.0.0.RELEASE</org.springframework-version>

->

<org.springframework-version>4.3.30.RELEASE</org.springframework-version>

으로 변경

밑에

<source>1.6</source>

<target>1.6</target>

->

<source>1.8</source>

<target>1.8</target>

로 변경

하면 pom.xml의 변경은 끝

여기서 왜?? 변경하는데 ?? 우리가 지금 사용하는 javac -version은 1.8 입니다. 그래서 버젼을 맞춰준거고

스프링도 1.8을 지원하는 놈이 있겠죠?? 그래서 같이 맞춰준겁니다.

그리고 src를 열어보면 바로 밑에 web.xml이 보이죠 ??

여기서는 version="2.9" -> version="3.1"로 변경

web-app\_2\_9.xsd -> web-app\_3\_1.xsd로 변경해줍니다.

?hy? 3.0버전 이후로 annotation이 사용가능해져서 사용합니다.W

-> 이거하고 alt + f5으로 update해줘야 반영됩니다.

web.xml이 하는 역할은 뭔데 ??

-> 프로젝트가 웹 프로그램으로 동작하기 위한 기본적인 설정을 넣어주는 주문서다.

-> 요청과 응답을 어떻게 받을 것인가 web-app version ( jsp version, el version, tomcat version )

-> 모든 요청을 처리하는 DispatcherServlet(자바 파일은 DS에 의해 호출되는 것) 을 선언

-> 인코딩 필터(모든 요청에 대해서 request.set...을 다 할 수 있는 것), 서블릿 상세 설정 파일 경로 포함

그럼 이제 root-context.xml하고 servlet-content.xml이 남는데 이건

web.xml에 적힌 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

여기서 DispacherServlet의 상세 정보가 담겨 있습니다.

root-context.xml 은 요청과 응답에 상관없이 프로젝트가 참조해야할 객체를 선언하는 주문서다.

servlet-content.xml은 요청과 응답에 관여하는 객체를 선언하는 주문서이다

servlet-content.xml에 가보면 밑에 base-package="com.itbank.test4" 이렇게 있다

음 뭔가 많이 본 형태일 것이다. src/main/java에 들어가면 이런 package가 보이는데

이 package안에 @Controller 가 붙은 class를 참조해라 라는 주문서다

그럼 이제 화면 띄우기

com.itbank.test4 안에 HomeController가 있는데 여기에 내용을 작성하고

WEB-INF -> views -> homs.jsp나 다른 jsp에서 출력할 내용을 작성

여기서 주의!! WEB-INF에 jsp를 바로 실행하면 오류뜬다. 여기서 바로 실행하면 tomcat이 뭔 이상한 경로지 하고 오류를 보낸다

그럼 어떻게 실행하냐 프로젝트에서 바로 Run on Server를 해야한다.

그럼 스프링 설치부터 프로젝트 실행까지 끝!!

===========================================

**Dependency Injection** : 스프링 프레임워크의 핵심 기능

**스프링의 핵심 기능 : Dependency Injection, transaction management**

스프링 : 기업형 응용 프로그램을 보조하기 위한 쉬운 프레임워크

Java EE(Spring) : 분산형, 기업형 응용 프로그램, 개발을 위한 API, 결합력을 낮추는 DI,

wDB Transaction 처리, 로그 처리..

Java SE : 일반적인 로컬 응용 프로그램 개발을 위한 API, 파일IO, console IO, 윈도우 IO, network IO

MVC -> DI <- 느슨한 결합력과 인터페이스, 트랜잭션 -> AOP, 인증과 권한 -> Servlet Filter

코드 수정을 없애고 DI를 위한 설정

B b = new B1(); -> B1() -> 객체 생성과 조립을 위한 외부 설정 (XML, Annotation)

service.setB(b); -> setB(b) -> DI

스프링 : 부품을 조립해주는 녀석들

주문서대로 부품을 생성하고 조립을 해주는 스프링

Ioc (Inversion Of Control) 컨테이너

-> 역순으로 무언가를 구성하는 컨테이너를 말한다

원래는

객체 -> a -> b -> c -> d 이건데

<------------------

만들어지는 건 반대로 xml을 읽어오면서 큰거에서 작은거로 가는 것이다.

-> 그래서 역순으로 객체를 생성하는 컨테이너 Dependency를 포함하고 있어서 그런듯

-> 부품이 어떤 조립관계를 갖는지 확인한다. 작은부품이 만들어지고 큰부품이 만들어지고 결합하는 과정을 반복한다

===========================================

@Controller 는 하나의 객체를 만들어서 안에 참조할 것을 적어주는 것

그래서 xml에 패키지를 읽어오면서 @Controller가 붙은 것을 빈으로 만들어준다 그런데 그 안에 참조할게 있으면 같이 참조해서 작동해주는 것

===========================================

**객체가 담겨있는 공간 : 객체 컨테이너**

GenericXmlApplicationContext ctx = new GenericXmlApplicationContext("classpath:appCtx.xml");

객체 의존관계

Toy toy = new Toy();

컨테이너에 선언된 객체 중 toy 클래스의 객체는 하나뿐이므로, 클래스 정보만 전달해도 객체를 불러올 수 있다

Toy ob1 = ctx.getBean(Toy.class);

만약 하나의 클래스로 선언한 객체가 여러개라면 bean의 id를 명시하여 불러온다

Toy ob2 = ctx.getBean("toy", Toy.class);

객체 컨테이너를 제외한 생성자 호출이 없다

특별히 명시하지 않으면, 모든 객체는 기본적으로 싱글톤이다.

springbean을 이용한다

IOC : Inversion Of Control, 제어 역전

스프링 컨테이너에 선언되어 있는 스프링 빈을 개발자가 받아서 프로그램을 진행한다

객체 관리(선언 및 의존성 관리)를 개발자가 아니라, 스프링이 한다

DI : Dependency Injection

스프링 빈 간의 참조가 필요하다면, (setter 나 생성자 매개변수를 통해 연결할 수 있다)

(스프링 빈 끼리는 자동으로 참조가 가능하다)

Auto DI : 스프링이 타입과 id로 추론하여 의존관계를 연결시켜줄 수 있다

@Autowired : 스프링 빈 중에서 자료형이 맞는 객체를 찾아서 알아서 넣어줘라

===========================================

<?xml version="1.0" encoding="UTF-8"?>

<!-- appCtx : applicationContext -->

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

지금 밑에 있는 것들이 IoC 컨테이너에 담겨있는 것이다.

<bean id="exam" class="teset3.Exam">

<property name="kor" value="30"></property>

<property name="mat" value="40"></property>

<property name="eng" value="50"></property>

</bean>

===========================================

Battery b1 = new Battery();

<bean id="b1" class="teset3.Battery"/>

Toy toy = new Toy();

<bean id="toy" class="teset3.Toy">

<property name="battery" ref="b1"/>

setBattery(Battery b1) 라고 만들어서 battery라는 이름으로 사용가능 한거다 / 아니면 사용불가

setBattery를 호출하는 방법 : name의 이름을 넣을 때 set을 빼고 맨 앞의 문자를 소문자로

받는 값에 따라서 value인지 ref인지 판단해야 한다

받는 값이 참조변수인지 아닌지 판단해서 참조변수이면 ref 아니면 value를 사용

생성자 매개변수를 만드는 방법 :<constructor-arg ref="b1"></constructor-arg>

</bean>

===========================================

**Dependency 객체 생성과 초기화**

Exam 객체 constructor를 이용한 속성 값 설정

Exam exam = new Exam(int num1, int num2, int num3);

<bean id="exam" class="teset3.Exam">

<constructor-arg index="0" value="50"/>

위에 처럼 index로 들어갈 순서를 지정할 수 있다

아니면 name="kor" 처럼 직접 컬럼이름을 지정해줄 수 있다

type도 넣어줄 수 있다

<constructor-arg value="30"/>

<constructor-arg value="70"/>

</bean>

이렇게 접두사를 붙여서 사용할 수 있다 근데 사용하려면 Namespaces에 들어가서 p를 체크해야 한다

<bean id="exam" class="teset3.Exam" p:kor="30" p:eng="40" p:mat="70"/>

<!-- <bean id="exams" class="java.util.ArrayList"> -->

<!-- <constructor-arg> -->

<!-- <list> -->

<!-- <bean class="teset3.Exam" p:kor="30" p:eng="40" p:mat="70"> -->

<!-- <ref bean="exam"> -->

<!-- </list> -->

<!-- </constructor-arg> -->

</beans>

===========================================

객체가 담겨있는 공간이므로, 객체 컨테이너라고 부른다

GenericXmlApplicationContext ctx =

new GenericXmlApplicationContext("classpath:appCtx.xml");

이렇게도 사용가능하다

ApplicationContext context = new GenericXmlApplicationContext();

ApplicatonContext 의 종류

1. ClassPathXmlApplicationContext

2. FileSystemXmlApplicationContext

3. XmlWebApplicationContext

4. AnnotationConfigApplicationContext

컨테이너에 선언된 객체 중 toy 클래스의 객체는 하나뿐이므로, 클래스 정보만 전달해도 객체를 불러올 수 있다

자료형 명으로 꺼내기 Toy의 타입에 맞는 클래스를 가져와줘

Toy ob1 = ctx.getBean(Toy.class);

System.out.println(ob1);

ob1.show();

만약 하나의 클래스로 선언한 객체가 여러개라면 bean의 id를 명시하여 불러올 수 있다

Toy ob2 = ctx.getBean("toy", Toy.class);

객체 컨테이너를 제외한 생성자 호출이 없다

특별히 명시하지 않으면, 모든 객체는 기본적으로 싱글톤이다

springbean을 이용한다

System.out.println(ob2);

===========================================

프로젝트 생성하고 나서는 패키지 이름을 수정해도 된다

============================================

===========================================

스프링 404

1. 주소 맵핑 X

2. 클라이언트에 보내줄 .jsp가 없는 경우

===========================================

405 -> 주소는 있는데 method가 없는 경우

===========================================

**어노테이션**

★ : 자주 사용하는 것

@Controller★

@RequestMapping

@GetMapping★

@PostMapping★

@RequestParam

@ModelAttribute

@PathVariable

매개변수 가능한 자료형

HttpServletRequest request

requeset.setAttribute("name", name);

HttpSession

String

primitive

DTO★

Model★

ModelAndView

반환형 가능한 자료형

void / String ★ / ModelAndView

===========================================

**스프링에서 반복되는 작업을 공통으로 처리할 수 있는 3가지 요소**

1. filter인코딩

2. interceptor로그인

3. aop로깅

dispatcherselvet의 멤버 필드가 model이다. -> 값을 넣기 위해 참조변수를 가지는 것이다.

반환되는 문자열과 보내는 attribute를 같이 넘겨주는 것이 ModelAndView이다.

Dispacther서블릿이 @Controller 를 인식하는 것

===========================================

<context:component-scan base package="com.itbank.test2"/>

지정한 패키지 내에 클래스 중에서 특정 어노테이션이 붙은 (\*\*클래스\*\*\*)를

객체로 생성하고 빈으로 등록해라

특정 어노테이션의 종류

@Controller(Web): MVC에서 컨트롤러, 서블릿에게 요청을 넘겨받음 ( request, response )

@Service(Java): 컨트롤러가 호출할 수 있는 함수를 모아서 가지는 객체, Model이나 Controller에 종속적이지 않은 코드를 작성(단독 실행할 수 있는거)

@Repository(DB): 데이터의 저장소, 즉 DB에 접근하는 객체, 보통 DAO를 가리킨다

@Component: 용도는 명확하지 않으나, 스프링 빈으로 등록하여 사용할 객체

@RestController: 기존 컨트롤러의 기능에서 모든 메서드에 @ResponseBody가 자동으로 적용된다. AJAX용으로 사용

@ControllerAdvice : 지정한 범위에서 발생하는 모든 예외를 집중시켜 처리하는 클래스

[Handler Mapping] => 스프링 빈을 찾는것 : 어노테이션이 @Controller인 것을 스프링 빈이 된 것을 객체로 인식해서 찾아가는 것

@Autowired : 의존성을 자동으로 맞춰준다 컨트롤러안에 @Autowired가 있으면 그거의 bean이 있는지 찾아보고 있으면 자동으로 참조해주는 것이다.

=============================================

**Model vs. ModelAndView 차이점**

-> Model은 데이터만 저장한다. ModelAndView는 데이터와 view page를 같이 저장한다

Spring Model 객체

Controller의 메서드는 Model이라는 타입의 객체를 파라미터로 받을 수 있다.

순수하게 JSP Servlet으로 웹 어플리케이션을 만들 때

보통 request나 session 내장객체에 정보를 담아 jsp에 넘겨주곤 했는데 Spring에서는 Model이라는 녀석을 쓴다.

즉 request.setAttribute() 와 비슷한 역할을 하는 것.

편리한 점은 개발자는 직접 model을 생성할 필요는 없다. 다만 파라미터로 선언만 해주면 스프링이 알아서 만들어준다.

스프링 MVC의 Controller는 기본적으로 Java Beans 규칙에 맞는 객체는 자동으로 화면에 전달해준다.

Java Beans의 규칙에 맞는다는 것은 단순히 생성자가 없거나 빈 생성자를 가지며,

getter/setter를 가진 클래스의 객체들을 의미한다.

전달될 때는 클래스명의 앞글자를 소문자로 처리하여 전달한다

Controller 메서드의 리턴타입

String : jsp를 이용하는 경우 jsp파일의 이름을 나타냄

void : 호출하는 URL과 동일한 이름의 jsp를 나타냄

VO, DTO타입 : 주로 json타입의 데이터를 만들어서 반환하는 용도

ResponseEntity : reponse할때 HTTP헤더 정보와 내용을 가공

Model, ModelAndView : Model로 데이터를 반환하거나 화면까지 지정

HttpHeaders : 응답에 내용없이 HTTP헤더 메세지만 전달하는 용도로 사용

===========================================

@RequestParam : 단일 파라미터를 전달 받을 때 사용하는 어노테이션

-> GET방식으로 넘어온 URI의 queryString을 받기에 적절하다

http://localhost:8080/reservation/api/reservations?reservationEmail=test@naver.com

이런 주소가 있으면 value에 reservationEmail를 넣으면 파라미터의 값을 가져올 수 있다.

@RequestParam(value = "reservationEmail",required = false) String reservationEmail)

value에는 넘어오는 파라미터의 name값을 적으면 되고

( value인데 name 값을 받는다. 근데 name 속성도 있다.)

required = false는 필수는 아니지만 @RequestParam의 required의 기본값은 true 여서

적지 않으면 선언한 파라미터가 반드시 있어야 한다. 없으면 400 에러 발생.

===========================================

@PathVariable : URL 경로에 변수를 넣어주는거에요

http://localhost:8080/reservation/api/reservations/2

-> 이런방식으로 전달받을 경우 사용한다

@PutMapping(path = "/reservations/{reservationId}")

public Reservations cancleReservations(@PathVariable(name = "reservationId") Integer reservationInfoId) {

return reservationService.cancelReservationById(reservationInfoId);

}

===========================================

@ModelAttribute

- 클라이언트가 전송하는 multipart/form-data 형태의 HTTP Body 내용과 HTTP 파라미터의 값들을 생성자나 Setter를 통해 주입하기 위해 사용된다

- @ModelAttribute에는 매핑시키는 파라미터의 타입이 객체의 타입과 일치하는지를 포함한 다양한 검증(Validiation) 작업이 추가적으로 진행된다.

===========================================

커맨드 객체(Command Object)

- HttpServletRequest 를 통해 들어온 요청 파라미터들을 setter 메서드를 이용하여 객체에 정의되어있는 속성에 바인딩이 되는 객체를 의미합니다.

- 커맨드 객체는 보통 DTO 를 의미하며, HttpServletRequest 로 받아오는 요청 파라미터의 key 값과 동일한 이름의 속성들과 setter 메서드를 가지고 있어야 합니다.

===========================================

@ExceptionHandler

-> @Controller, @RestController 가 적용된 Bean내에서 발생하는 예외를 잡아서 하나의 메서드에서 처리해주는 기능

@ExceptionHandler(NullPointerException.class)

public ModelAndView nullex(NullPointerException e){

ModelAndView mav = new ModelAndView();

mav.setViewName("alert");

mav.addObject("msg", e.getMessage());

return mav;

}

-> @ExceptionHandler 라는 어노테이션을 쓰고 인자로 캐치하고 싶은 예외클래스를 등록해주면 끝

@ExceptionHandler({Exception1.class, Exception2.class})

-> 이런식으로 두개 이상 등록 가능

@ExceptionHandler 주의사항

1. @Controller, @RestController 에만 적용가능하다 (@Service 안에 있으면 안됨)

2. return 타입이 자유롭다

3. @ExceptionHandler를 등록한 @Controller에서만 적용가능하다. 다른 @Controller에서 NPE가 발생하더라도 못잡는다.

===========================================

@ControllerAdvice

@ExceptionHandler가 하나의 클래스에 대한 것이라면, @ControllerAdvice는 모든 @Cotroller 즉, 전역에서 발생할 수 있는 예외를 잡아 처리해주는 어노테이션이다

@RestControllerAdvice

public class MyAdvice {

@ExceptionHandler(NullPointerException.class)

public String npe() {

return "npe";

}

}

===========================================

IoC 컨테이너 : 객체를 생성하고 객체를 조립해주는 공간

===========================================

**Annotation 으로 설정할 때의 장점**

-> XML로 설정할 떄의 모듈 변경 방법

<bean id="b" class="data.B2">

<bean id="console" class="service.S">

<property name="b" ref="b"/>

</bean>

->> 객체의 변경할 내용이 있을 때마다 업데이트를 해줘야 한다

-> Annotation으로 설정할 때의 모듈 변경 방법

폴더를 열었을 때 @가 붙은 어노테이션을 발견하면 객체로 만든다.

객체를 바꿔버리면 설정이 바뀌어 버리는 것이다. -> 그래서 손쉽게 업데이트를 할 수 있다.

===========================================

**DI 지원을 위한 스프링 어노테이션**

-> @Autowired 어노테이션

xml에서 @Autowired를 찾아달라고 하는 방법

Namespace에서 context를 추가하고 <context:annotation-config/> 를 입력해주면 된다.

-> config가 annotation 이 있다는 것이다. 그래서 그 안을 들여다 봐라. 컨테이너에 객체를 올려두고 객체안에 어노테이션이 있는지 확인한다.

-> 이름을 맞추는 방법으로 자동으로 찾아가는 방식도 있다

-> <context:annotation-config/>으로 읽어서 만들어낸 컨테이너에서 public void setExam(Exam exam)에서 Exam타입이 있는지 자동으로 확인

-> 의존성을 자동으로 맞춰준다 컨트롤러안에 @Autowired가 있으면 그거의 bean이 있는지 찾아보고 있으면 자동으로 참조해주는 것이다.

-> 이름이 같지 않으면 바인딩이 안될 것이다. 그때 사용하는 방법 : @qualifier()

-> @Qualifier("exam1")으로 하면 exam1에 해당되는 이름을 찾아서 매개변수와 연결해달라

===========================================

**@Autowired injection 방법 3가지**

1. setter injection

2. constructor injection

3. 기본 생성자에도 가능 -> 다만 생성자가 없는 상태에서 @Autowired를 사용하면 오류가 발생한다

@Autowired(required = false) => 객체가 없어도 일단 진행한다

===========================================

**pom.xml에 추가하는 라이브러리 정리**

1. xerces (version : 2.12.1)

-> XML 경로 상에 한글이 포함되어 예외를 발생시키지 않고 동작하게 하는 라이브러리

2. spring-jdbc (version : 4.3.30)

-> 스프링에서 DB접속을 지원하는 라이브러리

3. c3p0 (version : 0.9.5.5)

-> Connection Pool

4. ojdbc 8 (version : 21.3.0.0)

-> 오라클데이터 베이스에 접근하기 위한 라이브러리

5. mybatis (version : 3.5.7)

-> 영속성 프레임워크 / jdbcTemplate보다 DB에 접근을 쉽게 해주는 친구

-> myBatis는 Spring과 전혀 상관없는 독립적인 프레임워크

-> DAO객체와 SQL문을 mapping해주는 영속성 프레임워크

===========================================

mybatis 특징

1. jdbc만 사용했을 떄보다 소스코드가 깔끔해진다

2. java코드와 sql을 분리한다 (\*\*\*\*)

3. DAO와 JDBC 사에에 위치하여 서로를 맵핑해준다.

mybatis 동작 과정

1. mybatis config 파일을 읽어 SqlSessionFactoryBuilder 객체를 생성

(DB설정 정보, mapper파일 등록, typeAlias 설정 등등)

2. SqlSessionFactoryBuilder 객체를 이용해 SqlSessionFactory 객체를 생성

(SqlSessionFactoryBuilder는 단순히 SqlSessionFactory 객체를 생성해주기 위한 용도)

3. CURD 처리가 들어오면 SqlSessionFactory로 SqlSession 객체를 생성

4. SqlSession 객체를 이용해 DB요청을 한 후, 결과값을 받아옵니다.

===========================================

6. mybatis-spring

-> mybatis를 spring framework에 녹여내 좀 더 쉽게 사용하고자하는 연동 모듈

-> SqlSessionFactoryBean과 SqlSession을 spring framework의 bean으로 등록

-> SqlSession을 빈으로 등록할 때 class 속성을 보시면 SqlSessionTemplate라고 되어있다. -> SqlSession의 구현체

7. spring-orm

orm (Object Relation Mapping)

O : Object객체 ( Object Oriented Programming )

R : Relation관계 ( Relation DBMS ) <- 테이블 안에 있는 데이터가 Relation

M : Mapping맵핑 ( 자동으로 맵핑해준다 )

- 자바객체와 데이터베이스 테이블간의 맵핑처리

- 객체는 객체대로 설계

- 관계형 데이터베이스는 관계형 데이터베이스대로 설계

- ORM 프레임워크가 중간에서 맵핑

- 대중적인 언어에는 대부분 ORM기술이 존재

- 데이터베이스에 저장된 데이터와 객체를 맵핑하는 것은 프로그래밍 전략

- 데이터 생성, 데이터 조작 및 데이터 엑세스를 단순화 함

-> mybatis가 만들어낸 public class DAO implement test~ 이걸 객체로 만듬

-> 이걸 proxy라고 한다 / 중간다리 역할

-> 테이블 안에 있는 것을 자동으로 맵핑해준다

8. HikariCP

-> c3p0를 대체하는 역할 ( 커넥션풀 대체 역할 )

-> 자바버젼에 따라서 사용하는게 다르다 ( java 8 은 4.0.3을 사용하는 것이 바람직하다 )

===========================================

**root-context.xml 안에 들어가는 내용**

<!-- Connection Pool 이용한 DB접근 정보 : c3p0 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<property name="driverClass" value="oracle.jdbc.driver.OracleDriver"/>

<property name="jdbcUrl" value="jdbc:oracle:thin:@221.164.9.200:50000:xe"/>

<property name="user" value="c##itbank"/>

<property name="password" value="it"/>

<property name="maxPoolSize" value="100"/>

<property name="maxIdleTime" value="600"/>

<property name="idleConnectionTestPeriod" value="300"/>

</bean>

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

</bean>

<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="close">

<constructor-arg ref="sqlSessionFactory"/>

</bean>

<context:component-scan base-package="com.itbank.service"/>

<!-- DB에 접근할 mybatis-spring scan선언 -->

<mybatis-spring:scan base-package="com.itbank.model"/>

**---------------------여기서 부터는 HikariCP를 사용하는 방법-----------------------------**

-> HikariCP는 커넥션풀을 대체하기 때문에 기존에 사용하던 c3p0와는 다르게 DB정보를 작성해야 한다

-> Connection을 관리해주는 dataSource는 hikariCP로 처리하고, DB에 접속 및 쿼리문 수행, 객체-관계 맵핑은 myBatis 로 처리한다

<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />

<property name="jdbcUrl" value="jdbc:oracle:thin:@192.168.1.100:1521:xe" />

<property name="username" value="c##itbank" />

<property name="password" value="it" />

</bean>

<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">

<constructor-arg ref="hikariConfig" />

</bean>

<!-- 실질적으로 template를 설정해주는 곳 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<!--

myBatis에서 xml형식의 mapper 파일을 작성할 때 사용하는 설정 전역설정

src/main/resources -> config -> mybatis-config.xml을 참조하세요

-->

<property name="configLocation" value="classpath:config/mybatis-config.xml"/>

<!-- mapper 파일의 경로를 리스트 형식으로 작성 -->

<property name="mapperLocations">

<list>

<!--

classpath 는 src/main/resource를 말한다 여기에 있는 것이 실제로 있어야 작동할 수 있다

각각의 mapper파일이 DAO와 1:1 대응된다

-->

<value>classpath:config/sqlmap-test.xml</value>

<value>classpath:config/sqlmap-member.xml</value>

<value>classpath:config/sqlmap-login.xml</value>

</list>

</property>

</bean>

<!-- sqlSessionTemplate는 sqlSessionFactory의 생성자를 참조하여 생성되는 것이다 -->

<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="close">

<constructor-arg ref="sqlSessionFactory"/>

</bean>

<context:component-scan base-package="com.itbank.service"/>

<context:component-scan base-package="com.itbank.member"/>

<!-- DB에 접근할 mybatis-spring scan선언 -->

<!-- myBatis는 interface일 때만 등록한다 -->

<!-- 즉, 클래스를 인터페이스로 만들었으면 mybatis-spring으로 스캔할 수 있다 -->

<mybatis-spring:scan base-package="com.itbank.model"/>

===========================================

**mybatis-config.xml 안에 내용 설명**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration PUBLIC

"-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

--> 위에는 ctrl + space로 만들어서 해당 내용을 불러와야 태그를 사용할 수 있다

<configuration>

<settings>

<!-- 캐쉬 활성화 X, 자동키 X, user\_name -> userName : 맵핑때문에 언더바를 카멜케이스로 바꾸겠다 -->

<setting name="cacheEnabled" value="false"/>

<setting name="useGeneratedKeys" value="false"/>

<setting name="mapUnderscoreToCamelCase" value="false"/>

</settings>

<!-- type== 자료형, 자료형 이름이 너무 길어서 별명으로 부르면 길이를 짧게 사용할 수 있다 -->

<typeAliases>

<typeAlias type="com.itbank.member.MemberDTO" alias="member"/>

</typeAliases>

</configuration>

===========================================

**sqlmap-member.xml 안에 있는 내용**

각각의 mapper가 DAO와 연동되어야 하기 때문에 namespace에 DAO의 이름을객체 형식으로 작성

TestDAO testDAO = new TestDAO(); 라서 namespace가 testDAO가 되는 것이다

namespace는 내가 참조할 DAO를 적어주면 된다

<mapper namespace="MemberDAO">

<select id="itbank" resultType="com.itbank.member.MemberDTO"> / Alias 별명으로 이름을 줄인 것

<select id="itbank" resultType="member">

select \* from member

</select>

<select id="selectOrder" resultType="member" parameterType="String">

select \* from member order by ${order}

</select>

<insert id="insert" parameterType="member">

insert into member values (

#{userid}, #{userpw}, #{username}, #{email}, #{gender}

)

</insert>

</mapper>

===========================================

**sqlmap-login.xml 안에 있는 내용**

<mapper namespace="com.itbank.model.LoginDAO">

<!-- LoginDAO안에 login함수라는 뜻 / 원래는 statement에 넣어줘야 되는데 여기서는 interface로 했으니까 LoginDAO 안에 login 이라는 함수를 받는 것-->

<select id="login" resultType="member" parameterType="member"><!-- namespace : 패키지, 인터페이스DAO -->

select \* from member where userid= #{userid} and userpw = #{userpw}<!-- id : 인터페이스의 메서드이름 -->

</select>

</mapper>

===========================================

**sqlmap-test.xml 안에 있는 내용**

namespace는 내가 참조할 DAO를 적어주면 된다

<mapper namespace="MemberTestDAO">

sst.selectOne(statement), statement가 id에 맵핑

select 이므로 반환하는 데이터의 자료형을 명시해야 처리가 가능하다 / resultType=기본자료형

id를 함수이름으로 사용하면 mapper 자동주입

<select id="test1" resultType="string">

select banner from v$version

</select>

<select id="now" resultType="java.sql.Date">

select sysdate from dual

</select>

</mapper>

===========================================

**iBatis**

- iBatis는 SQL에 기반한 데이터베이스와 자바, 닷넷(.NET), 루비(Ruby) 등을 연결시켜 주는 역할을 하는 영속성 프레임워크(Persistence Framework)

- 연결은 프로그램의 소스코드에서 SQL 문장을 분리하여 별도의 XML 파일로 저장하고 이 둘을 서로 연결시켜주는 방식으로 작동

- iBatis는 사용자가 SQL 문장을 만들면 그에 적합한 객체모델을 생성하는 방식으로 작동

- 모든 SQL을 XML로 작성하고 SQL문을 사용하는 DAO클래스를 설계해서 SQL호출하는 방식

- 팀 전원이 아파치 소프트웨어 재단에서 구글 코드로 이전하면서 중단됨

- 새로운 프레임워크 MyBatis로 변경

장점

- 인터페이스와 애노테이션을 통해서 SQL문 설정하고 처리할 수 있음

- XML만을 이용해서 SQL문을 설정

- DAO에서는 xml찾아서 실행하는 코드 작성하는 방식

- SQL수정 유지보수적합하나 복잡성 증가

- 동적 Query 지원

- ORM 지원

===========================================

여기서 DAO는 interface로 만들어서 느슨한 결합력을 유지한다

===========================================

**HikariCP**

-> Datebase connection pool 관리

-> connection pool을 관리하는게 중요한 이유 : 성능에 큰 영향을 주기 때문이다.

-> connection 을 맺는 과정은 상당히 복잡하고 컴퓨터의 자원을 많이 소모하는 작업

-> HikariCP는 미리 정해놓은 만큼의 connection을 pool에 담아 두고 요청이 들어오면 Thread가 connection을 요청 -> HikariCP는 pool 안에 있는 connection 연결

============================================

**파일 업로드하기**

mvn에서 라이브러리 가져오기

commons-fileupload (version -> 1.4)

commons-io (versrion -> 2.11.0)

============================================

**@Component: 용도는 명확하지 않으나, 스프링 빈으로 등록하여 사용할 객체**

<context:component-scan>

-> 어플리케이션 컨텍스트에 등록된 빈들의 어노테이션들이 적용될 수 있게 한다.

-> 어노테이션이 설정된 새로운 빈들을 찾는 스캔도 할 수 있게 한다

특화된 @Component 어노테이션 -> @Controller, @Service, @Repository 이렇게 사용하면 이름을 특화시킬 수 있다.

->@Controller, @Service, @Repository 이렇게 사용하면 객체를 만들고자 하는 클래스가 어떤 역할을 하는지 명확하게 알 수 있다.

============================================

**Java Configuration**

<context:component-scan> -> 자바 클래스로 변경하는 방법 -> @ComponentScan("com.itbank.member") , @Configuration

<bean id="exam" class="com.itbank.member"> -> @Bean public Exam exam() {return new NewlecExam();}

-> id가 메서드의 이름이 된다

Application Context 생성하기

ApplicationContext context = new AnnotationConfigApplicationContext(NewlecAppConfig.class);

ApplicatonContext 의 종류

1. ClassPathXmlApplicationContext

2. FileSystemXmlApplicationContext

3. XmlWebApplicationContext

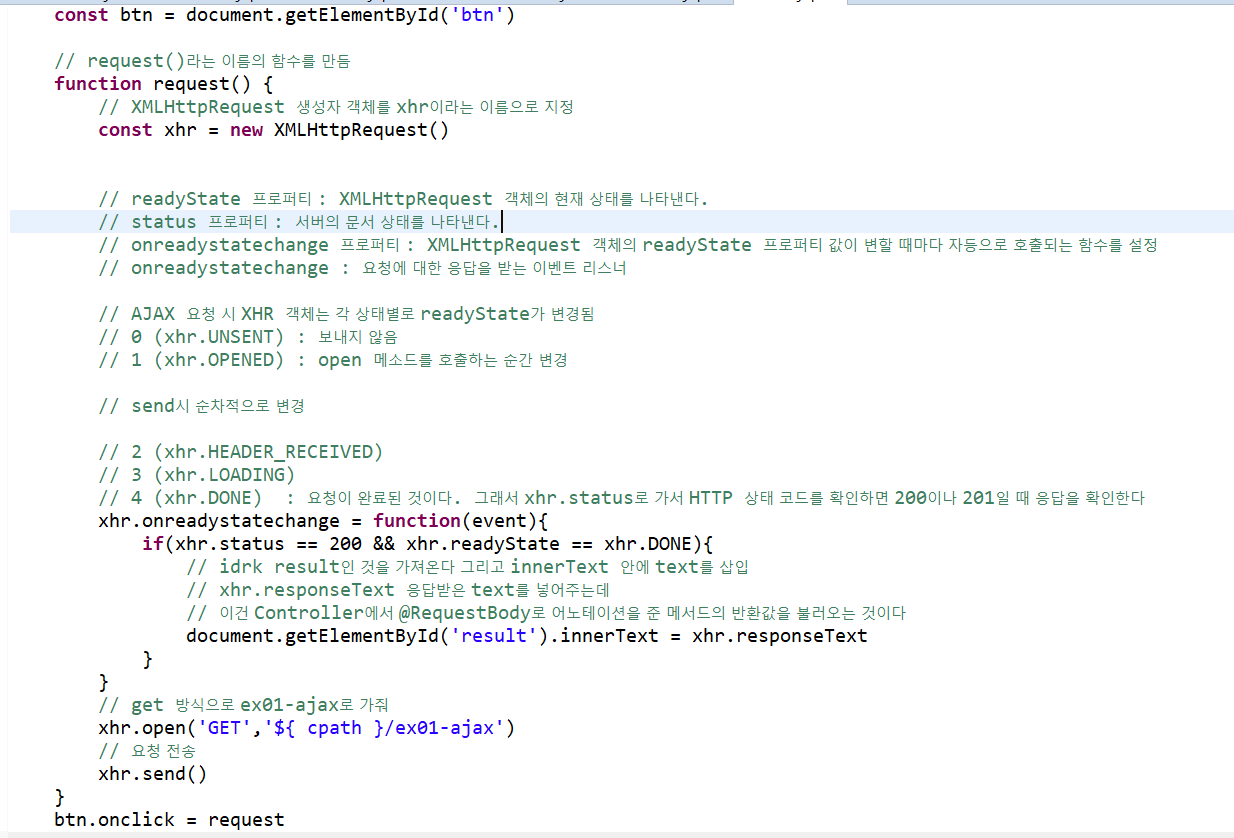
4. AnnotationConfigApplicationContext

============================================

**Ajax 정리**



**그림 1.xhr정리\_1**



**그림2. xhr정리\_2**

============================================

**Ajax**

-> XHR을 사용하여 비동기 처리를 하는 기술 AJAX(Asynchronous Javasrcipt And XML)

**fetch**

-> axios와 달리 브라우저에 내장되어있어서 라이브러리를 사용하지 않고 비동기통신을 할 수 있다.

-> fetch()함수는 첫번째 인자로 URL, 두번쨰 인자로 옵션 객체를 받고, Promise 타입의 객체를 반환한다.

-> 반환된 객체는 API호출이 성공했을 경우에는 응답객체를 resolve하고, 실패했을 경우에는 예외객체를 reject한다

-> 옵션 객체에는 HTTP 방식(method : get / post), HTTP 요청 헤더(headers), HTTP 요청 전문 (body) 등을 설정 가능

-> headers에는 요청헤더를 설정할 수 있고 body는 본문내용을 설정한다.

-> fetch는 get방식의 통신이 디폴트로 작동하고있어 따로 작성해주지 않아도 된다. get방식은 데이터를 가져올때 쓰는 방식이다.

-> post방식은 서버에 데이터를 생성한다면 데이터 요청전문을 포함한다. put,delete방식은 데이터 수정, 삭제를하기위해서 사용한다.

**Promise 타입**

-> 서버에서 받아온 데이터를 화면에 표시할 때 사용한다

-> .then은 순서대로 실행되면 실행된 값을 다음 .then에서 value로 값을 받아 실행한다

**클라이언트와 서버의 비동기 통신**

클라이언트에서 서버로 통신하는 메시지를 요청(request)메시지라고 하며,

서버에서 클라이언트로 통신하는 메시지를 응답(response) 메시지라고 한다.

웹에서 화면전환(새로고침)없이 이루어지는 동작들은 대부분 비동기 통신으로 이루어진다.

비동기통신을 하기 위해서는 클라이언트에서 서버로 요청 메시지를 보낼 때, 본문에 데이터를 담아서 보내야 하고,

서버에서 클라이언트로 응답을 보낼때에도 본문에 데이터를 담아서 보내야 한다.

이때 본문이 body이다

즉, 요청 본문은 requestBody, 응답 본문은 responseBody를 담아서 보내야 한다

이때 본문에 담기는 데이터 형식은 여러가지 형태가 있겠지만 가장 대표적인 것이 JSON이다

즉, 비동기식 클라이언트-서버 통신을 위해 JSON형식의 데이터를 주고 받는 것이다

**JSON?**

JavaScript Object Notation

key와 value 한쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷이다

http 통신 시 데이터를 주고 받을 때 사용하는 데이터 포맷 중 하나이다

비동기 통신을 위해, 넓게는 xml을 대체하는 주요 데이터 포맷이다

작성 순서

0. 라이브러리 추가 -> jackson-databind를 추가해준다

1. Controller에 ajax를 할 jsp파일의 주소를 맵핑한다

@GetMapping("/ex01")

public void ex01() {}

2. ex01.jsp를 생성하고 내부에 값을 받아올 준비를 한다

-> 버튼을 누르면 해당 값이 나오거나, 입력을 받아서 값을 불러오는 거나

3. ex01.jsp 하단에 스크립트 태그를 열어서 ajax구문을 작성한다

-> fetch(url, opt)으로 객체를 넘겨서 응답객체를 반환받는다

4. fetch의 첫번째 인자인 url이 가리키는 주소를 Controller에서 맵핑해준다

@GetMapping("/ajax1")

@ResponseBody

-> 서버에서 클라이언트로 응답 데이터를 전송하기 위해 @ResponseBody를 사용하여 자바 객체를

-> HTTP 응답 본문의 객체로 변환하여 클라이언트로 전송한다

5. 주소맵핑한 함수에 DB에 접근하여 값을 불러와서 값을 반환해준다

-> 반환된 값은 객체로 변환하여 클라이언트로 전송된다

6. 그럼 화면은 그대로인데 ajax기술을 이용해서 비동기통신을 해서

-> 특정 event나 submit이 발생할 경우 작성해둔 ajax 구문을 통해서

-> fetch로 주소와 옵션을 넘겨주면 응답 객체를 반환받아서 그걸 rendering해서

-> 화면에 바로 출력해주는 것이다