1. **스프링 프레임워크**

:: 동적인 웹 사이트를 개발하기 위한 여러가지 서비스를 제공

- 특징
- ‘경량 컨테이너’(크기와 부하의 측면)로서 자바 객체를 직접 관리
  : 각 객체 생성, 소멸과 같은 라이프 사이클을 관리하며 스프링으로부터 필요한 객체를 얻어올 수 있음
- 의존성 주입(DI: Dependency Injection)
  : 각 계층이나 서비스들 간에 의존성이 존재할 경우 프레임워크가 연결해줌
- 컨테이너
  : 애플리케이션 객체의 생명주기와 설정을 포함하고 관리
  ex. iBatis, myBatis 등 완성도 높은 데이터베이스 처리 라이브러리와 연결할 수 있는 인터페이스 제공
- 모델 - 뷰 - 컨트롤러 패턴
  : 웹 프로그래밍 개발 시 표준적 방식인 Spring MVC패턴을 사용한다.
  - DispatcherServlet이 Controller 역할을 담당하여 각종 요청을 적절한 서비스에 분산시켜줌.
  - 이를 각 서비스들이 처리하여 결과를 생성.
  - 그 결과는 다양한 형식의 view 서비스로 화면에 표시

- 선생님 설명

- 기존의 방법: servlet형태로 controller를 생성해, 해당 컨트롤러에서 모든 요청을 받아 command handling 작업
  ex. controller?cmd=00
- spring: 이미 해당 역할을 하는 servlet(controller)이 구축되어 있음
  해당 controller를 상속받은 java 코드(dispatcher servlet)를 생성해서 컨트롤러 역할 하기.
  url 자체가 cmd가 됨. 해당 servlet이 url 종류별로 확인을하고 해당 url에 등록된 함수를 호출.

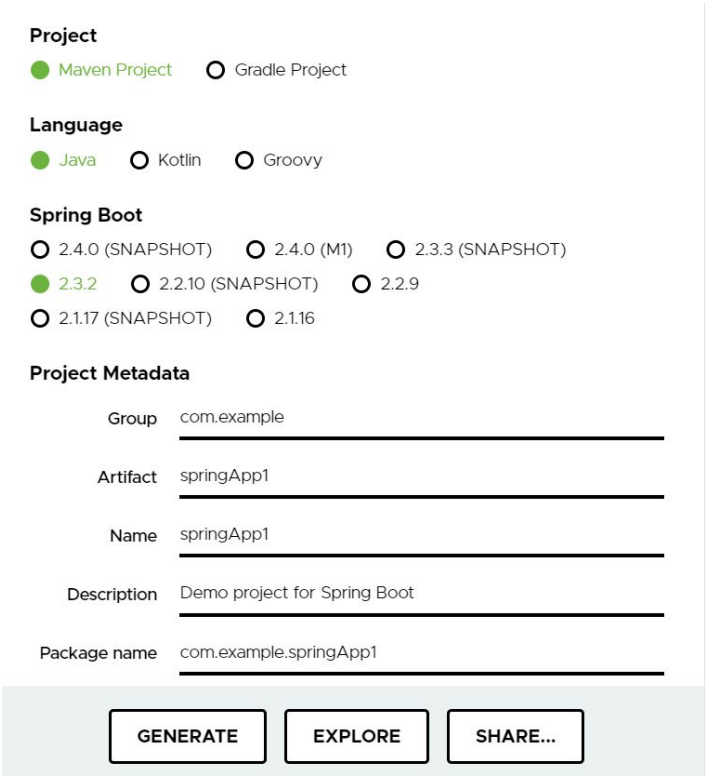
2. **스프링부트란?**

: 스프링 프레임워크를 사용하는 프로젝트를 간편하게 설정할 수 있는 스프링 프레임워크의 서브 프로젝트

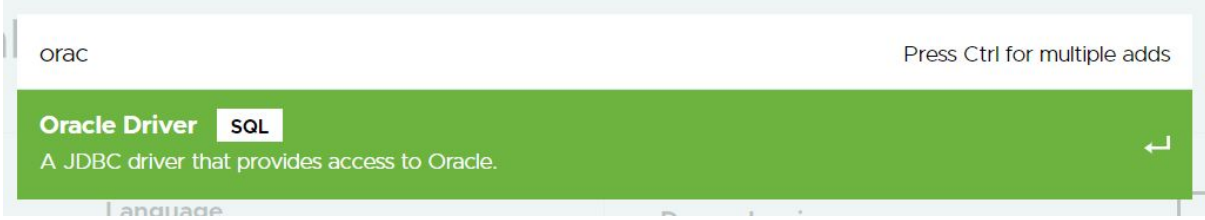
- 특징
- 까다로운 스프링의 설정방법을 간소화
- 프로젝트 환경 구축에서의 비기능적 기능을 기본 제공
  (ex. 내장형 서버, 시큐리티, 측정, 상태 점검, 외부 설정)

3. **스프링부트 시작**

1) 스프링 이니셜라이저 :: springboot 프로젝트 쉽게 생성하는 tool
   - 사이트 접속 :: https://start.spring.io/

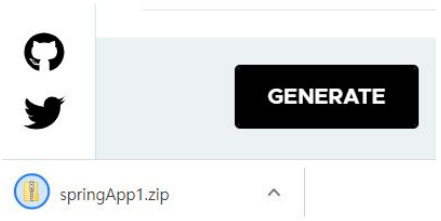


- project
  maven: 파일 버전을 관리
- language
- artifact : 원하는 어플리케이션 이름
- packaging
  jar: 일반
  war: 웹
- oracle db

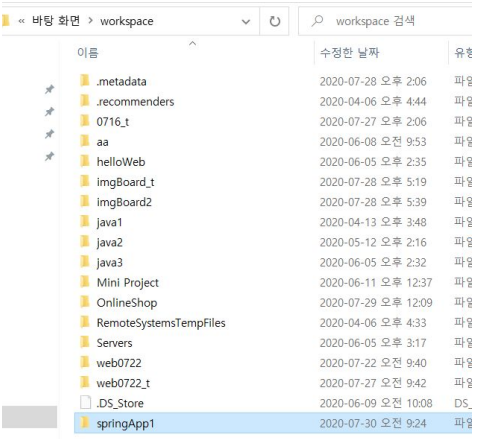


- mybatis

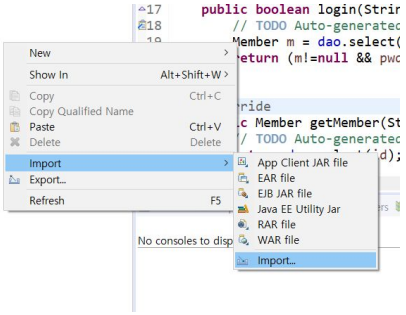
- generate => 압축파일로 생성



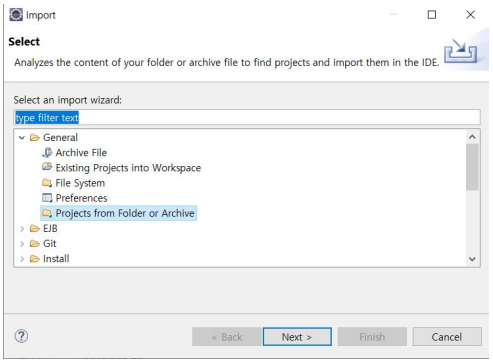
- eclipse workspace에 압축해제 폴더 붙여넣기



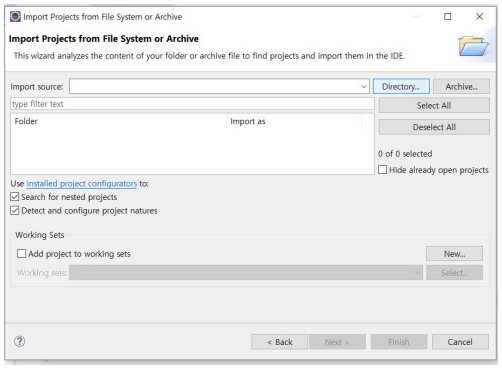
2) eclipse로 import
   a. import -> import



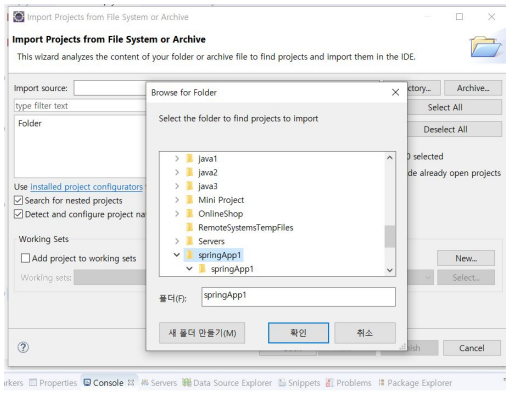
   b. import => projects from folder or archive



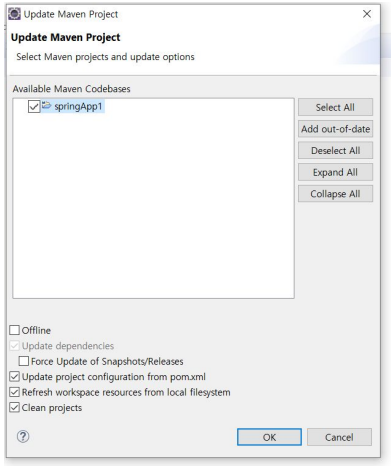
   c. directory



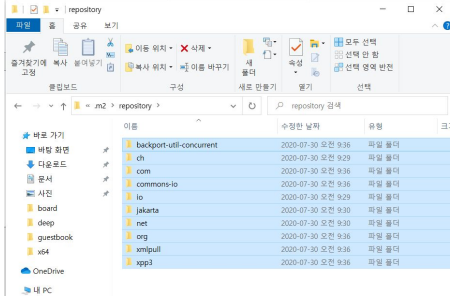
d. 해당 폴더 선택



- 하자마자 pom.xml에 에러가 나면, maven이 업데이트가 안되었을 가능성 maven필요한 jar 파일들을 설치시킨다.



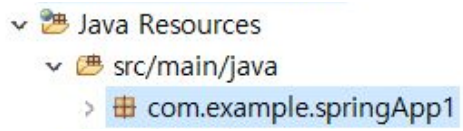
- cf1. 설정파일들이 저장된 위치 C:\Users\Playdata\.m2\repository



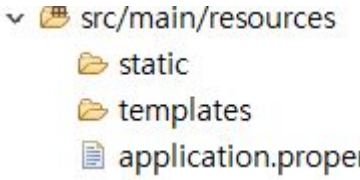
- cf2. 의존성 추가하고 싶을 때는 스프링 init 사이트 가서 추가된 코드 라인만 복사해서 넣으면 된다.

4. 프로젝트 구조

a. 자바 코드::src/main/java의 com.example.springApp



b. src/main/resources



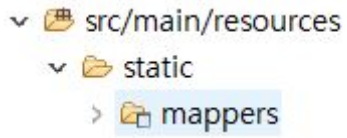
- static: html, javascript, css 등의 웹 정적 페이지 저장
- template
- application properties : spring app에서 필요한 여러 설정값.
  ex. db의 id나 pwd, driver 값…

● <application.properties>
spring.mvc.view.prefix=/WEB-INF/views/ #(view페이지에 대한 접두사)자동으로 앞에 붙여줌
spring.mvc.view.suffix=.jsp #(view페이지에 대한 접미사)자동으로 뒤에 붙여줌

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver # 드라이버이름
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/xe # localhost에 필요한 ip
spring.datasource.username=hr
spring.datasource.password=hr

mybatis.mapper-locations: static/mappers/*Mapper.xml #mapper 파일의 위치

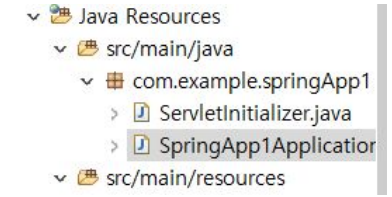
- cf. mybatis? 데이터베이스 연동 프레임워크
  daoimpl(db작업을 구현한 클래스) 구현 필요 없음
- cf2. Mapper.xml: mybatis 사용시 어떤 쿼리 실행, 검색 결과를 어떤 dto에 담을 것인지를 xml형태로 작성한 파일
- mappers 폴더 static 폴더 안에 넣어주기(끌어오지 말고 복사 붙여넣기!)



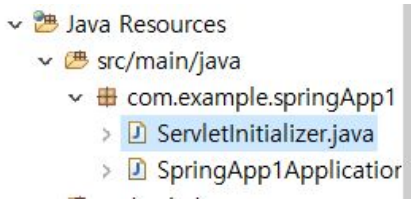
● <pom.xml> ⇒ 아래 코드 dependencies태그 닫기 전 아무곳이나 붙여넣기

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>
```

c. 메인



d. 서블릿 이니셜라이저: 서블릿 초기화 역할



5. 코드

1) 컨트롤러 (일반 자바 클래스로 생성하여 @Controller 어노테이션을 붙임)

```java
@Controller
public class TestController {
//요청 url 별로 실행할 코드 작성

    //의존성 자동 주입: 타입이나 이름이 동일한 bean을 찾아서 넣어줌
    @Autowired
    private MemberService service;

    @RequestMapping("/")
    public String home(){   //@GetMapping("/edit") @PostMapping("/edit")
        return "index";     //String: 뷰 페이지 경로 (view/index.jsp)
    }

    //get방식으로 '/edit' 받는 경우
    @GetMapping("/edit")
    public void editForm(){//반환값이 없는 경우: url과 동일한 이름의 view page로 간다

    }
    //post방식으로 '/edit' 받는 경우
    @PostMapping("/edit")
    public String edit(@ModelAttribute("m") Member m){
        return "editResult";
    }

    // "/list" 받는 경우
    @RequestMapping("/list")
    public ModelAndView list(){//ModelAndView: 모델과 뷰 넣을수 있는 객체
        //뷰 페이지 경로 setting(1: 생성자 param에 viewpage 이름 넣기, 2:mav.setViewName("viewpagename"));
        ModelAndView mav = new ModelAndView("list");

        ArrayList<String> list = new ArrayList<String>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        list.add("ddd");
        mav.addObject("list",list);//(전달할 데이터 이름, 값)
        return mav;
    }
```

```java
    // "/member/list" 받는 경우
    @GetMapping("/member/list")
    public ModelAndView modelList(){
        ModelAndView mav = new ModelAndView("list2");//list2라는 이름의 viewpage 생성자에 설정

        ArrayList<Member> list = service.getMembers();
        mav.addObject("m",list);//m이라는 이름으로 list를 참조할 수 있게 전송한다.
        return mav;
    }
```

//의존성 주입 : 타입으로 설정된 클래스로 jump해서 객체 생성.

- 어노테이션 종류

- @Controller: 해당 클래스를 컨트롤러 클래스로 사용하겠다는 표시

```
import org.springframework.stereotype.Controller;

@Controller
```

- @RequestMapping: url을 컨트롤러의 메서드와 맵핑(get과 post 모두 취급)

```
@RequestMapping("/")
public String home(){
    return "index";
}
```

- @GetMapping과 @PostMapping : 각 방식의 역할이 다를 경우

```
@GetMapping("/edit")

@PostMapping("/edit")
```

- @service : 서비스
- @Autowired: 의존성 자동주입(선생님 노트)
  서비스 객체, dao(mapper) 객체 생성등에 사용
  - mapper(daoimpl) 객체 생성

```
@Service
public class MemberService {
    @Autowired
    private MemberMapper mapper;
```

  - service 객체 생성

```
@Controller
public class MemberController {
    @Autowired
    private MemberService service;

    @GetMapping("/join")
```

  기존의 service 클래스=> dao 객체가 필요함. 생성자에서 만들어줬었음. (의존성 높아서 좋지 않음)
  유지보수를 위해서 직접 구현하지 않고 setter나 생성자를 사용해서 의존성 주입해줌.
  *spring에서는 new로 직접 dao나 service를 생성하지 않고, container가 만들어줌. 과거에는 application.xml을 두고 bean 태그에 해당 객체들을 등록했었음.
  autowired는 TYPE이나 NAME이 같은 객체를 넣어줌.
  **결국 @ 어노테이션이 있으면 타고들어가서 읽어옴. 마지막으로는 mapper.xml

2) [모델] com.example.springApp1.model.join 패키지

- Member (dto)
- MemberMapper 인터페이스(Dao)

```java
package com.example.springApp1.model.join;

import java.util.ArrayList;

import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface MemberMapper {
    void addMember(Member m);
    Member getMember(String id);
    ArrayList<Member> getMembers();
    void editMember(Member m);
    void delMember(String id);
}
```

- MemberService

```java
package com.example.springApp1.model.join;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MemberService {
    @Autowired
    private MemberMapper mapper;

    public void addMember(Member m){
        mapper.addMember(m);
    }
    public  Member getMember(String id){
        return mapper.getMember(id);
    }
    public ArrayList<Member> getMembers(){
        return mapper.getMembers();
    }
    public void editMember(Member m){
        mapper.editMember(m);
    }
    public void delMember(String id){
        mapper.delMember(id);
    }
}
```

3) MemberMappers.xml
: daoImpl을 자동으로 구현하기 위해 myBatis에게 제공하는 파일. mapping 정보가 담김.

- 용어
    - xml: 사용자 정의 태그로 value를 맵핑
    - namespace: dao 인터페이스 풀네임(패키지명+이름)
    - id: 메소드 이름
    - resultType: 반환할 값의 타입 지정(기본 타입)
    - parameterType: 메서드가 받을 파라미터 타입(기본 타입, 클래스 타입 다 가능)
    - resultMap: 반환할 값의 타입 지정(사용자 정의 타입)
- 수정
  현재 MemberMapper 인터페이스 위치로 namespace값 변경하기

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.springApp1.model.join.MemberMapper">
    <resultMap type="com.example.springApp1.model.join.Member" id="member">
        <result property="id" column="id" />
        <result property="pwd" column="pwd" />
        <result property="name" column="name" />
        <result property="email" column="email" />
    </resultMap>
    <insert id="addMember" parameterType="com.example.springApp1.model.join.Member">
        insert into member values(#{id}, #{pwd}, #{name}, #{email})
    </insert>
    <select id="getMember" resultMap="member" parameterType="String">
        select * from member where id=#{id}
    </select>
    <select id="getMembers" resultMap="member">
        select * from member
    </select>
    <update id="editMember" parameterType="com.example.springApp1.model.join.Member">
        update member set pwd=#{pwd}, email=#{email} where id=#{id}
    </update>
    <delete id="delMember" parameterType="String">
        delete from member where id=#{id}
    </delete>
</mapper>
```

4) [뷰] /src/main/webapp/WEB-INF/views에 view 파일들 생성

- 주의

- view 페이지 jsp 파일 다이렉트로 실행하면 에러
- spring 사용 안할 경우, web-inf 밑에 views 페이지 넣으면 실행 안됨!!!

a. edit.jsp(post 방식으로 요청)
input 태그 안에 각 line의 name이 dto 클래스 변수명과 일치해야 함

```jsp
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<h3>edit form</h3>
<form action="${pageContext.request.contextPath }/edit" method="post">
id:<input type="text" name="id"><br>
pwd:<input type="text" name="pwd"><br>
name:<input type="text" name="name"><br>
email:<input type="text" name="email"><br>
<input type="submit" value="edit"><br>
</form>
</body>
</html>
```

b. editResult.jsp

- 해당 클래스 이름으로 접근 가능

```
@PostMapping("/edit")
public String edit(Member m){

</head>
<body>
<h3>edit result</h3>
${member.id } / ${member.pwd } / ${member.name } / ${member.email }
</body>
</html>
```

- 설정한 이름으로 접근 가능

```
@PostMapping("/edit")
public String edit(@ModelAttribute("m") Member m){

<h3>edit result</h3>
${m.id } / ${m.pwd } / ${m.name } / ${m.email }
</body>
```

```jsp
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<h3>edit result</h3>
${m.id } / ${m.pwd } / ${m.name } / ${m.email }
</body>
</html>
```

c. list.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<h3>list</h3>
<c:forEach var="s" items="${list }">
<li>${s }</li>
</c:forEach>
</body>
</html>
```

d. list2.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<h3>list2</h3>
<c:forEach var="s" items="${m }">
<li>${s.id } / ${s.pwd }/${s.name }/${s.email }</li>
</c:forEach></body>
</html>
```

5) 연습: 회원가입
get방식 join form 띄우기
post방식 join완료