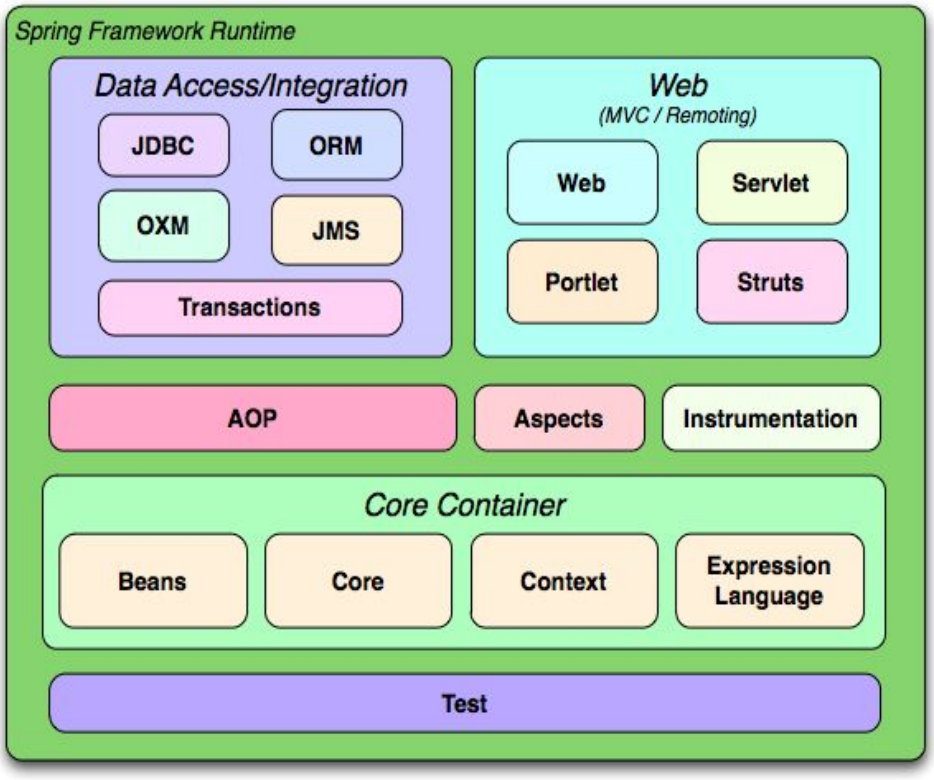
1월 28일 < Spring>

프레임워크와 라이브러리스프링 프레임워크의 탄생 배경
IOC 컨테이너== 스프링 컨테이너 == 컨테이너
IOC 컨테이너 생성- sts3 플러그인 설치
IOC 컨테이너 생성- 스프링 설정파일(스프링 빈 생성) 작성
IOC 컨테이너 생성- scope 테스트(singleton, prototype)
IOC 컨테이너 생성- collection 객체 생성
IOC 컨테이너 생성- property-placeholder
IOC 컨테이너 생성 - bean scan



1. 프레임워크와 라이브러리

1) Framework 와 library 차이 이해

- 라이브러리

프로그램 호출에 의한 제어, 정해진대로 사용, 원본 라이브러리 재정의 가능

beanUtils - 파라미터를 바인딩 하고싶은 클래스 정보를 인자로 준다

- 개발자가 제어한다

- 프레임워크

- inversion of control : 프로그램의 흐름이 프레임 워크에 의해 제어

- Extensibility : 기능 확장 가능

==> 프레임워크 : 전체적인 틀 (톱니바퀴 - 사이즈는 정해져 있으나 문양같은건 고를 수 있다)

2) 탄생배경

스프링 이전

- C/C++ 기반의 웹 프로그래밍 -> 서버 벤더에 종속된 개발환경

  -> 서버에따라 구현방법이 다르다 -> 개발자의 불만

- 처음 자바의 용도 (java beans spec 자바 표준 규약 참고)

- 과거 JAVA는 GUI component를 개발하기 위한 목적이 강했음

- 자바빈이란, 재사용가능한 software component

  ⇒ 시간이 지남에 따라 사람들은 server 구현용도로 java의 가능성에 주목

- J2 EE : 표준 설계 / EJB 기능이 불편(SQL에서 ORDER BY를 사용 못함)

- J2EE Development without EJB -> SPRING FRAME WORK

프레임 워크를 왜 써야할까?

- 비기능을 구현하지 않고도 사용 가능 : 비기능은 계속 반복이 되는데, 만들기는 어렵다

- 프레임워크는 다양하지만, 그 중 스프링은 대중적이다

- 참고) https://www.slideshare.net/kthcorp/a3-15041875 (봄날은 간다)

- 봄날은 간다의 요점: 안정적이고 좋으나 사용자한테 빠른 응답을 내거나 엄청많은 트랜잭션을 처리할 때에는 항상 스프링이 답은 아니다(점점 달라지므로 참고하면 좋다)

< 정리>

※ 프레임워크를 사용하는 이유

- 개발자는 각자 실력차이의 폭이 큰 직업군.
- 개발자의 구성에 따라 프로젝트의 결과물이 차이가 큼.
- 회사는 프레임워크를 사용함으로써 일정한 품질을 보장함.
- 개발자는 정해진 틀에서 부품을 끼워맞추기에 개발시간을 단축 할 수 있음.
- 시스템의 기능 외에 필요한 비기능 요소들 제공받을 수 있다.

(기능을 이용해서 쓰려면 비기능 요소를 사용 -> 반복적으로 비기능 요소를 구현할 필요가 없다)

- 비기능 요소는 반복적으로 사용되며 고정적 (DB connection 관리, transaction 관리)
- 기능은 유연적 (인사 시스템의 사원 정보 저장, 조회, 변경)
- 비기능 요소들을 직접 구현할 자신이 없다

- 비즈니스 로직 구현에 집중할 수 있다

⇒ 구현 / 유지보수의 편이, 개발자들의 역량 획일화

획일화 는 장점이자 단점 (차별성이 없을 수 있다) * 차별성을 위해서는 sql을 잘 짜거나, 비기능 요소들을 직접 구현

=> 개발자들이 비즈니스 로직에만 집중할 수 있도록 기반 환경을 제공

※ 스프링 프레임워크를 사용하는 이유

1.복잡함에 반기가 들어서 만들어진 프레임워크

- EJB에 비해 가볍기 때문에 엔터프라이즈급의 시스템을 더 빠른 시간에 작성 가능.

2. 프로젝트의 전체 구조를 설계할때 유용한 프레임워크

- 기존의 프레임워크들은 웹, 또는 하드웨어, 데이터베이스등 전문적인 영역만 지원하는 경우가 많은데, 스프링은 어느 한 분야에 집중하기 보다, 전체를 설계하는 용도로 사용

3. 다른 프레임워크의 포용

- 스프링은 전체 구조에 집중했기에 특정영역의 프레임워크와 공존하는 방식으로 사용 가능

※ 장점

1) 개발자는 비즈니스 로직에만 집중해서 코드를 개발할 수 있음

2) 각 프로젝트마다 다른 관심사를 적용할때 코드의 수정을 최소화 시킬수 있음

3) 원하는 관심사의 유지보수가 수월한 코드를 구성 할 수 있음.

단점 : 정해진 틀대로 코드를 작성

※ 스프링의 주요 특징

- 프레임 워크 뜻 : 기능을 미리 클래스나 인터페이스 등으로 만들어 제공하는 반제품

Spring 프레임워크

⇒ POJO로 구성된 애플리케이션 + POJO가 어떻게 관계를 맺고 동작할 것인가를 담은 설계정보

1. POJO기반의 구성

POJO는 특정 규약이나 환경에 종속적이지 않으며 개발자가 비즈니스 로직에 집중 할 수 있게 해준다.

- POJO(Plain Old Java Object) 의 구성만으로 가능하도록 제작(따로 프레임워크의 사용을 위해 공부할 필요가 없음).

2. 의존성 주입(DI)을 통한 객체간의 관계 구성

- 의존성이란 : 어떤 객체 A가 다른 객체 B의 도움을 받아야만 일이 처리가능하다면,
A는 B에 의존적이다.

java에서는 인터페이스를 이용하여 이런 의존적인 객체의 관계를 유연하게 처리하도록 함.

- 의존성 주입이란, 의존적인 객체를 직접생성하거나 제어하는 것이 아니라,

특정 객체에 필요한 객체를 외부에서 결정해서 연결시키는 것.

주입방법

1. Setter Injection

2. Constructor Injection

3. 어노테이션 사용 (+ Method Injection)

3. AOP지원

AOP(Aspect oriented programming): 기존의 절차지향이나 객체지향에서 기능의 모듈화는 프로그램을 복잡하게만들며, 수정 및 유지보수가 힘들게 함.

공통된 부분등, 특정 부분들(핵심 관심사) 로 나누어 관리함으로써,프로그램을 모듈화 하는 방식. (즉, 반복적인 코드를 줄이고 핵심 비즈니스 로직에 집중하도록 지원)

4. 편리한 MVC 구조

5. WAS에 독립적인 개발환경

6. 트랜잭션의 지원

※ 트랜잭션이란 : 데이터 베이스에서 한꺼번에 수행되어야 될 연산. (예= atm)

스프링은 이를 어노테이션이나 xml로 설정 가능

IOC 컨테이너 이해

IOC(Inversion of Control)

1. 개념

-IOC(제어권의 역전)이란,

객체의 생성, 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미

- Component dependency resolution

의존관계 결정 , 설정(configuration) 및 생명주기 (lifecycle)를 해결하기 위한 디자인 패턴

2. IoC의 컨테이너

스프링 프레임워크도 객체에 대한 생성 및 생명주기를 관리할 수 있는 기능을 제공하고 있음.

즉, IoC 컨테이너 기능을 제공한다.

- IoC 컨테이너는 객체의 생성을 책임지고, 의존성을 관리한다.
- POJO의 생성, 초기화, 서비스, 소멸에 대한 권한을 가진다.
- 개발자들이 직접 POJO를 생성할 수 있지만 <u>컨테이너에게 맡긴다.</u>

⇒ 객체를 생성하는 설명서와 함께 제어권을 다른 무엇인가에 위임 :

객체를 관리하는 컨테이너 (Spring IOC 컨테이너)

3. IoC의 분류

DL(Dependency Lookup) 과 DI(Dependency Injection)

- DL : 저장소에 저장되어 있는 Bean에 접근하기 위해 컨테이너가 제공하는 API를 이용하여 Bean을 Lookup 하는 것

⇒ 컨테이너에 설정된 빈을 참조 (잘 사용하지 않는다) - applicationContext객체로부터 service 객체 참조

- DI : 각 클래스간의 의존관계를 빈 설정(Bean Definition) 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것 (DL 사용시 컨테이너 종속이 증가하여, 주로 DI를 사용함.)

⇒ 컨테이너가 빈 설정 정보를 바탕으로 의존성을 고려하여 객체(bean)를 주입 - xml, 어노테이션 설정

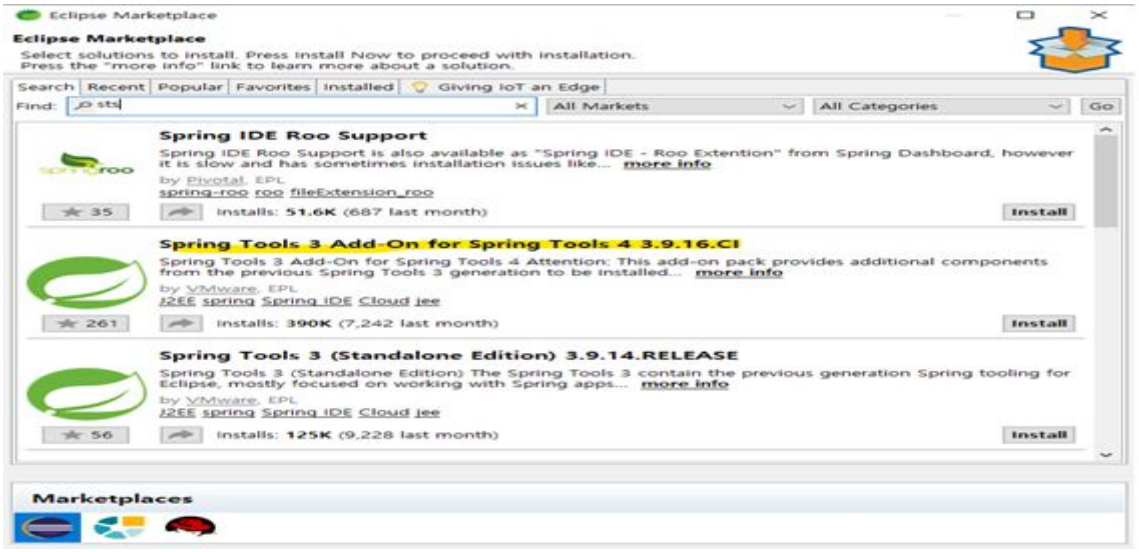
* 등록한 객체를 전부 관리 : 선언적 트랜잭션이 가능하다 -> 관리를 해주는 spring Container

- 기존에서는 servletContainer ==> spring Container 로 관리를 넘겨준다

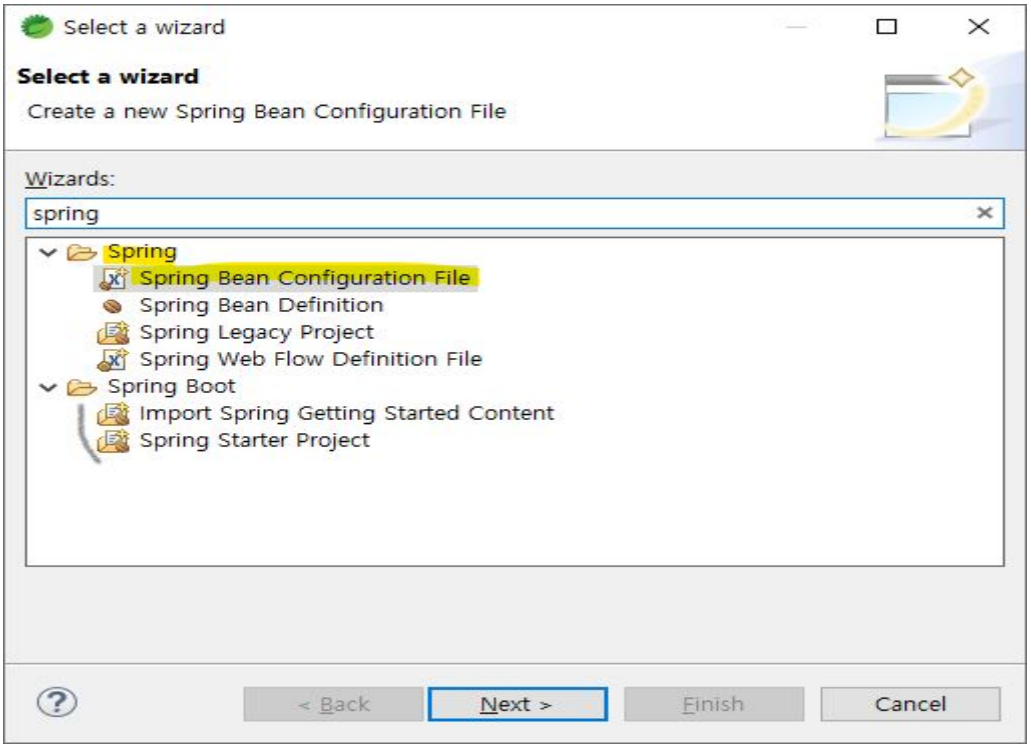
4. IOC 컨테이너 생성

1) STS3 플러그인 설치

- marketPlace 가서 설치: 오래된 spring 개발환경도 관리 가능, 템플릿을 제공해줘서 xml을 편하게 만들 수 있다 (스프링 설정파일-xml)



2) Spring Bean Configuration File 생성(xml)



```java
public static void main(String[] args) {
    // 1. 스프링 설정 파일을 이용하여 스프링 컨테이너를 생성
        // 1) 스프링 설정파일 생성 - 마켓플레이스에서 설치 후 (kr/or/ddit/ioc/ioc.xml)생성
        // 2) 스프링 컨테이너 타입 : ApplicationContext
    //ApplicationContext extend EncironmentCapable  - interface가 interface를 상속 받을때는 extend

    ApplicationContext context = new ClassPathXmlApplicationContext("classpath:/kr/or/ddit/ioc/ioc.xml");
```

3) 스프링 설정파일 이용하여 스프링 컨테이너 생성 (DL)

- 설정파일을 두가지 형태로 기술 - 설정파일 : ioc.xml

1. 물리적인 파일의 경로 - d:\\upload file :

2. 클래스 path - class classpath:

3-1) ApplicationContext - 경로 설정하기

```java
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:/kr/or/ddit/ioc/ioc.xml");
```

3-2) iocMain.java

```java
public static void main(String[] args) {
    // 1. 스프링 설정 파일을 이용하여 스프링 컨테이너를 생성
        // 1) 스프링 설정파일 생성 - 마켓플레이스에서 설치 후 (kr/or/ddit/ioc/ioc.xml)생성
        // 2) 스프링 컨테이너 타입 : ApplicationContext
    //ApplicationContext extend EncironmentCapable  - interface가 interface를 상속 받을때는 extend

    ApplicationContext context = new ClassPathXmlApplicationContext("classpath:/kr/or/ddit/ioc/ioc.xml");

    // 2. 스프링 컨테이너에게 만들어진 스프링 빈을 요청 (스프링 빈- 객체)
    // DL(Dependency Lookup) : 스프링 컨테이너에게 스프링 빈을 요청하는 과정

    // 1) ioc에 만든 bean 가져오기
    //UserDao userDao = new UserDaoImpl(); - 원래
    UserDao userDao = (UserDao) context.getBean("userDao");


    // 3. 스프링 컨테이너에서 관리되고 있는 빈이 잘 만들어 졌는지 확인
    UserVo userVo = userDao.getUser("brown");
```

3-3) userVo

```java
public class UserVo {
    String userid;
    String usernm;

    public UserVo() {}

    // 인자가 있는 생성자 만들기
    public UserVo(String userid, String usernm) {
        setUserid(userid);
```

- toString을 해줘야 값이 나온다(선언 안하면 주소값이 나온다)
- 인자가 있는 생성자를 만들어줘야한다
- getter, setter 생성

3-4) userDao - interface

```
public interface UserDao {

    // 사용자 아이디로 사용자 조회
    // 인자가 사용자 아이디, 반환타입이 userVo
    UserVo getUser(String userid);

}
```

3-5) userDaoImpl

```
public class UserDaoImpl implements UserDao{

    @Override
    public UserVo getUser(String userid) {
        // 원래는 데이터베이스에서 조회를 해야하나, 개발 초기단계라 설정이 완료되지 않음
        // 현재, 확인하려고 하는 기능은 스프링 컨테이너에 초점을 맞추기 위해 new 연산자를 통해 생성한 vo 객체를 반환

//      UserVo user = new UserVo("brown", "브라운"); // 인자가 있는 생성자 만들기
        return new UserVo("brown", "브라운");
    }
```

3-6) userService - interface

```
public interface UserService {

    // 사용자 정보를 조회하는 로직 - 아이디로 받아서 vo를 반환
    UserVo getUserVo(String userid);


}
```

3-7) userServiceImpl

```
public class UserServiceImpl implements UserService{

    // spring 형태를 dao를 주입하는 방식
    // 1. setter 메소드 이용
    // 2. 클래스를 만들때 생성자를 통해 주입을 받을 수 있다

    private UserDao userDao;

    @Override
    public UserVo getUserVo(String userid) {
        return userDao.getUser(userid);
    }
```

4) dao, service를 context를 통해 스프링 컨테이너에게 스프링 빈을 요청

```java
public class IocMain {

    private static final Logger logger = LoggerFactory.getLogger(IocMain.class);

    public static void main(String[] args) {
        // 1. 스프링 설정 파일을 이용하여 스프링 컨테이너를 생성
            // 1) 스프링 설정파일 생성 - 마켓플레이스에서 설치 후 (kr/or/ddit/ioc/ioc.xml)생성
            // 2) 스프링 컨테이너 타입 : ApplicationContext
        //ApplicationContext extend EncironmentCapable  - interface가 interface를 상속 받을때는 extend

        ApplicationContext context = new ClassPathXmlApplicationContext("classpath:/kr/or/ddit/ioc/ioc.xml");

        // 2. 스프링 컨테이너에게 만들어진 스프링 빈을 요청 (스프링 빈- 객체)
        // DL(Dependency Lookup) : 스프링 컨테이너에게 스프링 빈을 요청하는 과정

        // 1) ioc에 만든 bean 가져오기
        //UserDao userDao = new UserDaoImpl(); - 원래
        UserDao userDao = (UserDao) context.getBean("userDao");

        // 3. 스프링 컨테이너에서 관리되고 있는 빈이 잘 만들어 졌는지 확인
        UserVo userVo = userDao.getUser("brown");

        logger.debug(" dao - userVo : {}", userVo);

        // 스프링 컨테이너로부터 userService 스프링 빈을 DL을 통해 얻어오고
        // getUser 메소드를 call, 반환된 값(userVo)을 logger를 통해 출력

        UserService userService = (UserService) context.getBean("userService");
        UserVo vo = userService.getUserVo("brown");

        logger.debug(" service - userVo :{}", vo);
```

- 직접적으로는 스프링 컨테이너에 의해 관리가 되지 않는다(어노테이션이 없음)

⇒ 이렇게 하고 실행 할 경우결과 (new 연산자가 없어 생성되지 않음 - null 에러)

5) service에 property 해주기

4-1) setter 생성 (serviceImpl 에 설정)

4-2) ioc.xml에 <property name=" " ref =" " > 설정

```
     userService - 다른 스프링 빈을 주입을 받아서 만들어졌다
    UserService userService = new UserServiceImpl();
    userService.setUserDao(userDao);

-->

<bean id="userService" class="kr.or.ddit.user.service.UserServiceImpl">
    <property name="userDao" ref="userDao"/>
</bean>
```

- property 는 filed, setter를 의미
- value : 문자열 , 숫자 / ref : 다른 스프링 빈, 주입받고 싶은 스프링 빈 이름(id)을 기술

```
435 [main] DEBUG kr.or.ddit.ioc.IocMain -  dao - userVo : UserVo [userid=brown, usernm=브라운]
 in thread "main" java.lang.NullPointerException: Cannot invoke "kr.or.ddit.user.repoitory.UserDao.getUser(String)" because "this.userDao" is nul
t kr.or.ddit.user.service.UserServiceImpl.getUserVo(UserServiceImpl.java:16)
t kr.or.ddit.ioc.IocMain.main(IocMain.java:40)
```

테스트 코드 만들기

- 기존 테스트 코드에서 dao를 생성한것을 application context로 변환
- junit 은 eclipse , maven에 자동으로 내장되어 있다

  ⇒ main 메소드가 없어도 실행되는 이유

프링 환경에서 junit 코드를 실행 ⇒ junit 코드도 스프링 빈으로 등록

1) 환경설정

```
// 스프링 환경에서 junit 코드를 실행 ⇒ junit 코드도 스프링 빈으로 등록
@RunWith(SpringJUnit4ClassRunner.class)
public class UserDaoTest {
```

2) contextConfiguration

```
// 기본값은 설정파일 위치를 지정하게 되어있다
@ContextConfiguration("classpath:/kr/or/ddit/ioc/ioc.xml")
```

- iocMain은 spring에서 관리해주는것이 아님 → 가져오려면 xml에 설정해주면 된다

test 코드 작성

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:/kr/or/ddit/ioc/Ioc.xml")
public class UserDaoTest {

    @Resource(name="userDao")
    private UserDao userDao;


    @Test
    public void getUserTest() {
        /***Given***/
        String userid = "brown";

        /***When***/
        UserVo userVo = userDao.getUser(userid);

        /***Then***/
        assertEquals("브라운", userVo.getUsernm());

    }
}
```

인자가 있는 생성자를 만들었으므로 에러 ⇒ 기본 생성자를 만들어 줘야 한다

- SCOPE

  bean의 속성 (jsp와 다르다), singleton(default - 설정을 안하면 default로 설정된다) , prototype이 존재 - 현재까지 사용한것은 id, class

- 디자인 패턴 - singleton : 특정 클래스의 인스턴스가 메모리상에서 하나만 존재하도록 하는 패턴 (생성과 관련된 패턴)
- Spring scope - singleton : id 당 하나의 객체를 만든다, bean 엘리멘트를 선언시 중복된 class를 사용 가능

  → 동일한 클래스의 객체가 스프링 컨테이너 상에서는 여러개 존재하는 것이 가능., 스프링 컨테이너에서 빈의 식별 단위는 id

```
@Resource(name="userService")
private UserService userService;

@Resource(name="userService")
private UserService userService2;

@Resource(name="userServiceCons")
private UserService userServiceCons;

// userServiceCons 스프링 빈이 정상적으로 생성 되었는지 테스트
@Test
public void userServiceConsTest() {
    /***Given***/


    /***When***/


    /***Then***/
    assertNotNull(userServiceCons);
}

@Test
public void beanScopeTest() {
    /***Given***/


    /***When***/

    /***Then***/
// 디자인 패턴의 singletone 개념으로 보면 두개의 객체는 한개의 클래스로 부터 나왔으므로 동일해야 한다
    assertEquals(userService, userServiceCons);
    // ==> 실행하면 id당 관리를 하므로 동일하지 않다 (디자인 패턴이 아니라 spring이므로 )

    만약 동일한 스프링 빈을 주입받았으므로 userService, userService는 같은 객체다
    assertEquals(userService, userService2);
}
```

collection

```
<!--  list, set, map 컬렉션 객체를 스프링 빈으로 등록  : xml로 해당 객체를 만들 수 있는지 확인 -->
<!--    property를 사용하면 setter 로 주입을 하겠다는 의미  -->
<bean id="collectionBean" class="kr.or.ddit.ioc.CollectionBean">
    <property name="list">
        <list>
            <value>brown</value>
            <value>sally</value>
            <value>cony</value>
        </list>
    </property>
    <property name="map"></property>
    <property name="set"></property>
</bean>
```

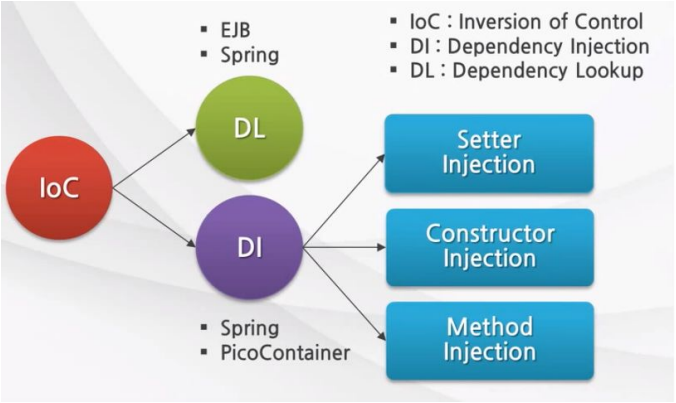
리스트, map, set을 여기서 지정해 줄 수 이있다

- map 은 키와 value를 entry 에 저장한다

- ref : 스프링 빈을 주입할 때 사용하는 속성, 해당 속성에 들어오는 문자열은 스프링 빈의 이름(id)이어야 한다

cf) 공통적으로 만들어야 하는 리스트일 경우. 클래스를 만들기는 애매하면 xml-bean에 설정 실제 코드를 구현하지 않아도 만들 수 있다
⇒ spring에서 collection 객체를 만들어 내는것이 가능하다

- xml에 설계했을때의 치명적인 단점
    - 없는 class를 명확하게 보여주지만 오타가 생겨도 컴파일 해주지 않아 실행하기 전에 모른다 -> 자바의 경우는 컴파일이 가능
    - 공공기관에서는 많이 사용 (편한건 자바지만 xml로 사용)
        - xml 과 java 로 하는것 - 스프링부트 두가지 공부해야된다



<참고>

db 연결하기

1) db 연결 vo 생성 / properties에 연결할 정보 등록하기
2) xml에 설정

```
<!--    properties 파일의 값을 스프링 빈의 값으로 주입하기
        db와 연결을 하려면 일단 properties의 값을 가져와야 한다
        - context를 설정해줘야 한다 (namespace에서)
        -
-->
    <context:property-placeholder location = "classpath:/kr/or/ddit/config/dbinfo.properties"/>
    <bean id="dbConfig" class="kr.or.ddit.ioc.DBConfig">
            <property name="driverClassName" value="${}"/>
            <property name="url" />
            <property name="username" />
            <property name="password" />
    </bean>
```

* 지금까지 쓴 방식은 기본적인 방식

- 이렇게 빈을 설정하면 테이블이 많을 경우 많이 생성해줘야하는 문제가 있으므로, 회사에서 보편적으로 사용하는 방식 은 **component-scan 을 사용**하여 빈을 만든다

  → bean 엘레멘트를 대체할 수 있는 방법

  - 기존 : 스프링 빈을 하나씩 선언 <bean ... * 선언하려고 하는 갯수만큼 >

  - 개선 : component-scan 이용

  웹에서 자주 사용하는 모듈(레이얼) - controller, service, repository(dao)

  1. 구현 클래스에 어노테이션을 적용 (@Controller, @Service, @Repository)

  2. 엘레멘트를 xml에 선언 ⇒ 특정 패키지 밑 등록된 클래스를 스캔

     - @Controller, @Service, @Repository 등 어노테이션이 붙은 클래스들을 스프링 빈으로 등록

componentScanTest 사용

1) xml 에 componnent-sacn을 설정

```
<!-- 어디서부터 scan할건지 지정
    ==> base-package : kr.or.ddit 패키지 하위의 모든 클래스를 대상으로 스캔
                       여러개의 베이스 패키지 설정 가능
                       base-package = "kr.or.ddit.user, kr.or.ddit.login"

        기본 스캔 대상이 되는 annotation : @Controller, @Service, @Repository, @Component

        annotation == 주석

-->
<context:component-scan base-package="kr.or.ddit"/>
```

2) DAO에서 respository 생성

⇒ dao에서 주입을 해주기 위해 가장 많이 쓰는 것은 어노테이션 (추가로 생성자, setter 가 있다)

```
@Repository("userDao")
public class UserDaoImpl implements UserDao{

    @Override
    public UserVo getUser(String userid) {
        // 원래는 데이터베이스에서 조회를 해야하나, 개발 초기단계라 설정이 완료되지 않음
        // 현재, 확인하려고 하는 기능은 스프링 컨테이너에 초점을 맞추기 위해 new 연산자를 통해 생성한 vo 객체

        // UserVo user = new UserVo("brown", "브라운"); // 인자가 있는 생성자 만들기
        return new UserVo("brown", "브라운");
    }
```

3) Service에 @Service 를 생성, resource를 통해 사용할 dao 가져오기

⇒ service에는 주로 service 어노테이션 사용

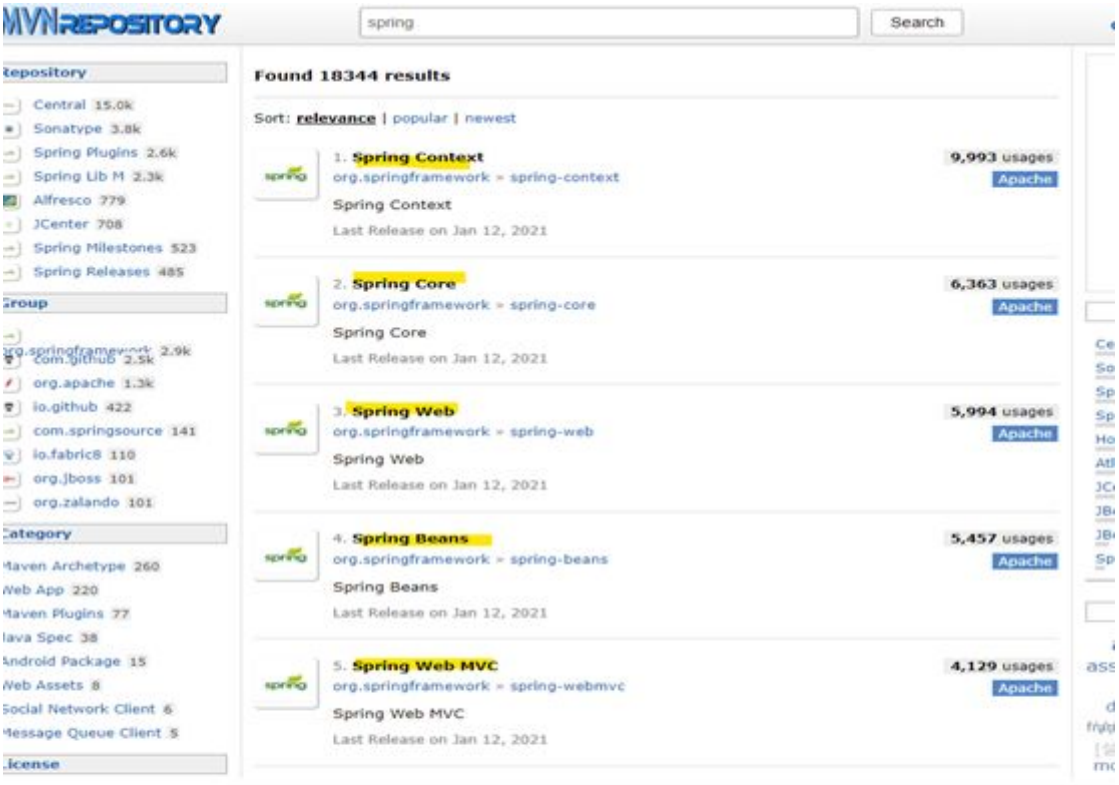
+ dao Resource 로 변경해주기

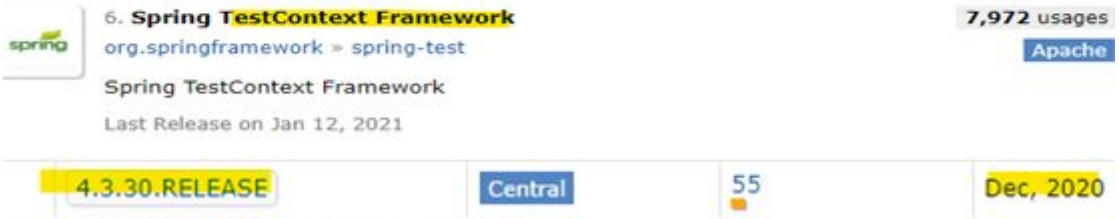
```
@Service
public class UserServiceImpl implements UserService{

    // spring 형태를 dao를 주입하는 방식
    // 1. setter 메소드 이용
    // 2. 클래스를 만들때 생성자를 통해 주입을 받을 수 있다

//  private UserDao userDao;
    // repository 나 클래스 명에서 가져오기 (xml)
    @Resource(name="userDao")
    private UserDao userDao;

    public UserServiceImpl() {  }

    public UserServiceImpl(UserDao userDao) {
        this.userDao = userDao;
    }

    @Override
    public UserVo getUser(String userid) {
        return userDao.getUser(userid);
    }
```

4) Test 해보기 (test 할 때 runwith, contextConfigure, 사용할 resource name 맞게 적어주기)

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:/kr/or/ddit/ioc/component-scan.xml")
public class ComponentScanTest {

    // @Repository 어노테이션을 적용한 userDaoImpl 스프링 빈이 정상적으로 스프링 컨테이너에 등록 되었는지 확인
    @Resource(name = "userDao")
    private UserDao userDao;

    @Resource(name = "userServiceImpl")
    private UserService userService;

    @Test
    public void userDaoImplSpringBeantTest() {
        assertNotNull(userDao);

        UserVo userVo = userDao.getUser("brown");
        assertEquals("브라운", userVo.getUsernm());

    }

    // userServiceImpl 스프링 빈이 정상적으로 컨테이너에 등록 되었는지 확인
    @Test
    public void userServiceImplSptringBeantTest() {
        assertNotNull(userService);

        UserVo userVo = userService.getUser("brown");
        assertEquals("브라운", userVo.getUsernm());
    }
```

- Maven Project 생성시 Spring jar 설정 (6가지) + junit, jstl, logback (pom.xml에 설정)





버전: 4.3.30 사용 - 스프링 부트일때에는 5버전 사용
+ jdbc 까지 받으면 된다

slf4 : simple log back