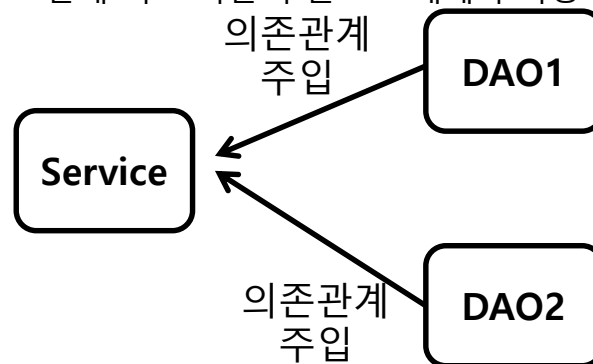


# Dependency Injection (의존성 주입)

# 의존성 주입 (Dependency Injection, DI)

- 의존 관계 주입 (dependency injection)
  - 객체간의 의존관계를 객체 자신이 아닌 외부의 조립기가 수행한다.
  - 제어의 역행 (inversion of control, IoC) 이라는 의미로 사용되었음.
  - Martin Fowler, 2004
    - 제어의 어떠한 부분이 반전되는가라는 질문에 '의존 관계 주입'이라는 용어를 사용
    - 복잡한 어플리케이션은 비즈니스 로직을 수행하기 위해서 두 개 이상의 클래스들이 서로 협업을 하면서 구성됨.
    - 각각의 객체는 협업하고자 하는 객체의 참조를 얻는 것에 책임성이 있음.
    - 이 부분은 **높은 결합도(highly coupling)**와 테스트하기 어려운 코드를 양산함.
  - DI를 통해 시스템에 있는 각 객체를 조정하는 외부 개체가 객체들에게 생성시에 의존관계를 주어짐.
    - 즉, 의존이 객체로 주입됨.
    - 객체가 협업하는 객체의 참조를 어떻게 얻어낼 것인가라는 관점에서 책임성의 역행(inversion of responsibility)임.
  - 느슨한 결합(loose coupling)이 주요 강점
    - 객체는 인터페이스에 의한 의존관계만을 알고 있으며, 이 의존관계는 구현 클래스에 대한 차이를 모르는채 서로 다른 구현으로 대체가 가능



# Spring의 DI 지원

- Spring Container가 DI 조립기를 제공
  - 스프링 설정파일을 통하여 객체간의 의존관계를 설정한다.
  - Spring Container가 제공하는 api를 이용해 객체를 사용한다.

# Spring 설정파일

- Application에서 사용할 Spring 자원들을 설정하는 파일
- Spring container는 설정파일에 설정된 내용을 읽어 Application에서 필요한 기능들을 제공한다.
- XML 기반으로 작성한다.
- Root tag는 <beans> 이다
- 파일명은 상관없다.

예) applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans      xmlns="http://www.springframework.org/schema/beans"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

</beans>
```

# Bean객체 주입 받기 – 설정파일 설정(1/2)

- 주입 할 객체를 설정파일에 설정한다.
  - <bean> : 스프링컨테이너가 관리할 Bean객체를 설정
    - 기본 속성
      - name : 주입 받을 곳에서 호출 할 이름 설정
      - id : 주입 받을 곳에서 호출할 이름 설정 ('/' 값으로 못 가짐)
      - class : 주입할 객체의 클래스
      - factory-method : Singleton 패턴으로 작성된 객체의 factory 메소드 호출 시

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="dao" class="spring.di.model.MemberDAO"/>

</beans>
```

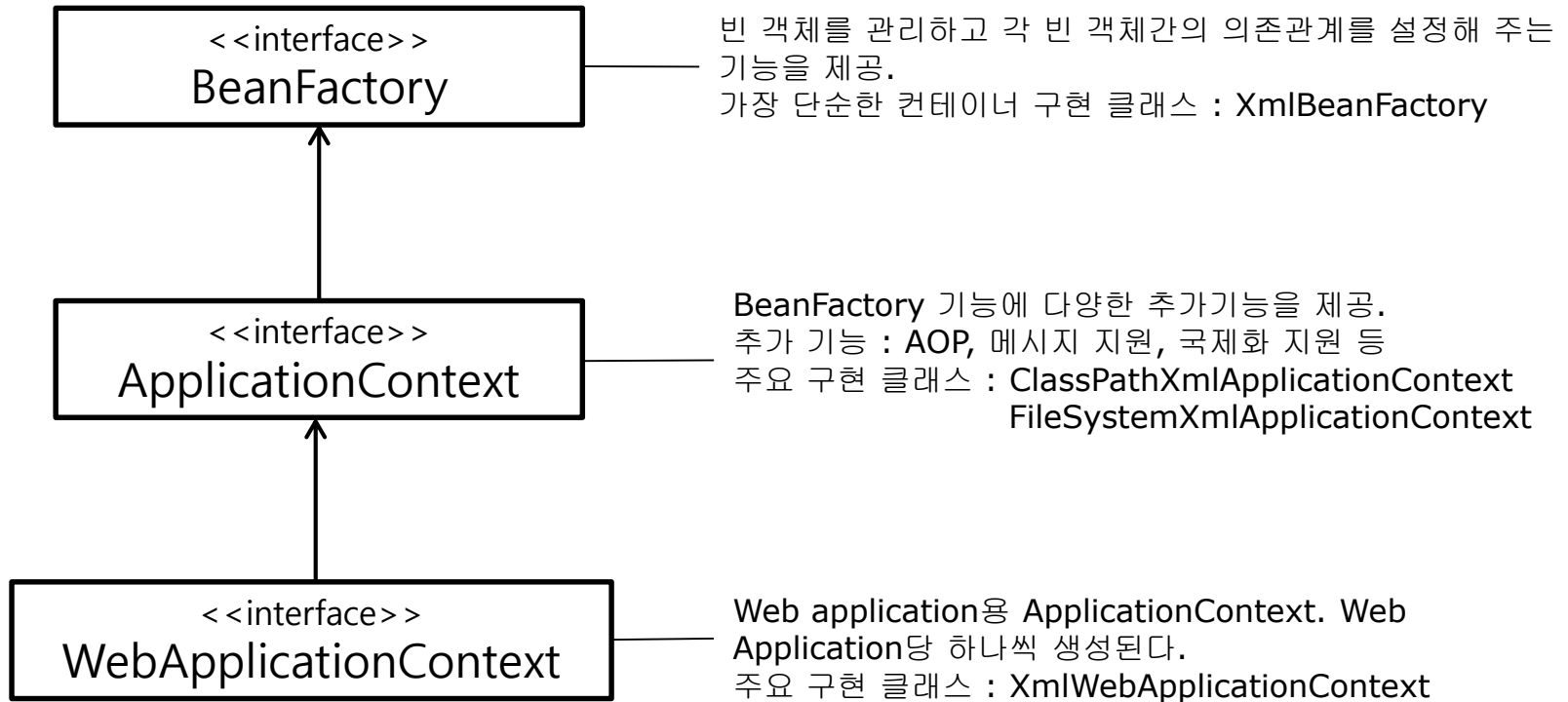
# Bean객체 주입 받기 – 설정 Bean 사용(2/2)

- 설정 파일에 설정한 bean을 Container가 제공하는 주입기 역할 api를 통해 주입 받는다.

```
public static void main(String [] args){  
  
    //설정 파일이 어디 있는지를 저장하는 객체  
    Resource resource = new ClassPathResource("applicationContext.xml");  
  
    //객체를 생성해주는 factory 객체  
    BeanFactory factory = new XmlBeanFactory(resource);  
  
    //설정 파일에 설정한 <bean> 태그의 id/name을 통해 객체를 받아온다.  
    MemberDAO dao = (MemberDAO)factory.getBean("dao");  
  
}
```

# DI관련 주요 클래스 (1/2)

- Spring Container : 객체를 관리하는 컨테이너.
  - 다음 아래의 interface들을 구현한다.



# DI관련 주요 클래스 (2/2)

- Resource 구현 클래스
  - Resource interface : 다양한 종류의 자원을 동일한 방식으로 통일하여 표현할 수 있게 한다.
  - XmlBeanFactory는 객체 생성시 설정 파일의 위치를 알려 줘야 한다. – Resource를 이용
  - 주요 구현 클래스
    - org.springframework.core.io.FileSystemResource  
파일시스템의 특정 파일의 자원을 관리
    - org.springframework.core.io.ClassPathResource  
클래스 패스에 있는 자원을 관리
    - org.springframework.web.context.support.ServletContextResource  
웹 어플리케이션의 **Root 경로를 기준으로** 지정한 경로의 자원을 관리
    - InputStreamResource, PortletContextResource

예)

```
Resource resource = new ClassPathResource("applicationContext.xml");  
BeanFactory factory = new XmlBeanFactory(resource);
```



# 설정을 통한 객체 주입 – Constructor를 이용 (1/4)

- 객체 또는 값을 생성자를 통해 주입 받는다.
- <constructor-arg>
  - <bean>의 하위태그로 설정한 bean 객체 또는 값을 생성자를 통해 주입하도록 설정
  - 설정 방법 : <ref>, <value>와 같은 하위태그를 이용하여 설정, 속성을 이용해 설정
  - 하위태그 이용
    - <ref bean="bean name"/> : 객체를 주입 시
    - <value>값</value> : 문자(String), Primitive data 주입 시
      - type 속성 : 값을 1차로 String으로 처리한다. 값의 타입을 명시해야 하는 경우 사용. ex) <value type="int">10</value>
  - 속성 이용
    - ref="bean 이름"
    - value="값"

# 설정을 통한 객체 주입 – Constructor를 이용 (2/4)

값을 주입 받을 객체

```
package vo;
public class Person{
    private String id,
    private String name,
    private int age;

    public Person(String id){...}           //1번 생성자
    public Person(String id, String name){...} //2번 생성자
    public Person(int age){...}           //3번 생성자
}
```

1번 생성자에 주입 예

```
<bean id="person" class="vo.Person">
    <constructor-arg>
        <value>abcde</value>
    </constructor-arg>
</bean>
또는
<bean id="person" class="vo.Person">
    <constructor-arg value="abc"/>
</bean>
```

# 설정을 통한 객체 주입 – Constructor를 이용 (3/4)

## 2번 생성자에 주입 예

```
<bean id="person" class="vo.Person">  
  <constructor-arg>  
    <value>abcde</value>  
  </constructor-arg>  
  <constructor-arg>  
    <value>Hong Gil Dong</value>  
  </constructor-arg>  
</bean>
```

또는

```
<bean id="person" class="vo.Person">  
  <constructor-arg value="abc"/>  
  <constructor-arg value="Hong Gil Dong"/>  
</bean>
```

## 3번 생성자에 주입 예

```
<bean id="person" class="vo.Person">  
  <constructor-arg>  
    <value type="int">30</value>  
  </constructor-arg>  
</bean>
```

또는

```
<bean id="person" class="vo.Person">  
  <constructor-arg value="abc" type="int"/>>  
  <constructor-arg value="Hong Gil Dong"/>  
</bean>
```

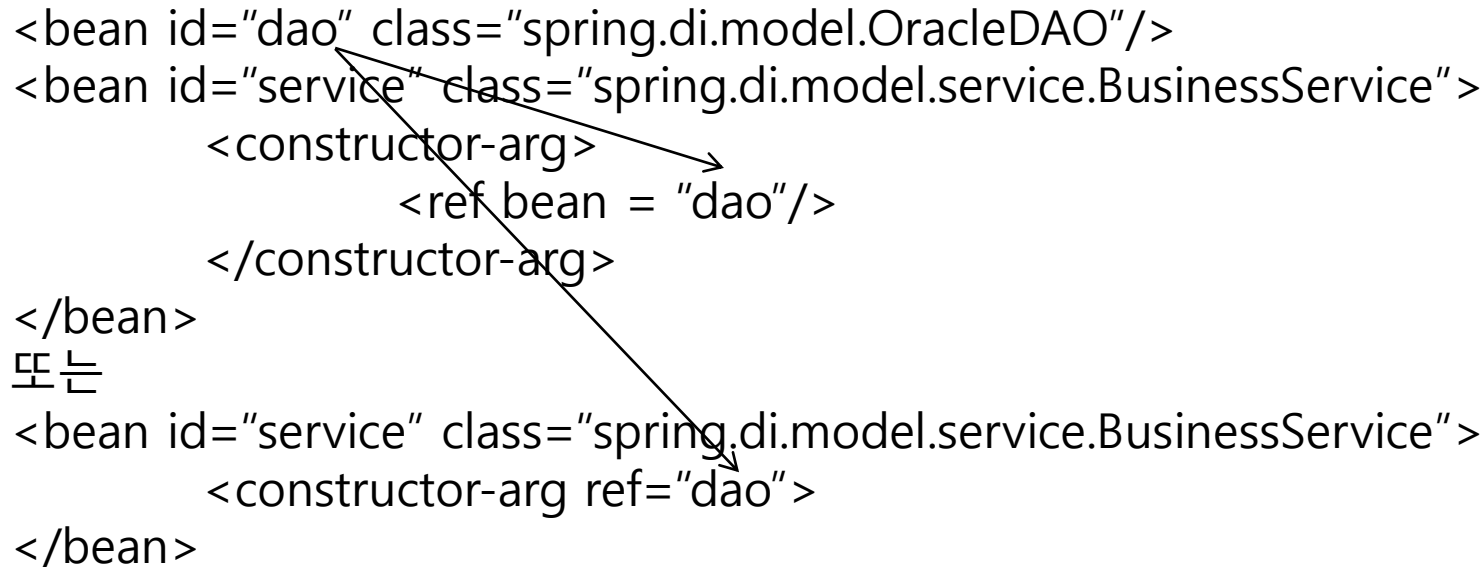
# 설정을 통한 객체 주입 – Constructor를 이용 (4/4)

- bean객체를 주입

값을 주입 받을 객체

```
public class BusinessService{  
    private Dao dao = null;  
    public BusinessService(Dao dao){  
        this.dao = dao;  
    }  
}
```

```
<bean id="dao" class="spring.di.model.OracleDAO"/>  
<bean id="service" class="spring.di.model.service.BusinessService">  
    <constructor-arg>  
        <ref bean = "dao"/>  
    </constructor-arg>  
</bean>  
또는  
<bean id="service" class="spring.di.model.service.BusinessService">  
    <constructor-arg ref="dao">  
</bean>
```



# 설정을 통한 객체 주입 – Property를 이용(1/5)

- property를 통해 객체 또는 값을 주입 받는다-setter 메소드.
  - 주의 : setter를 통해서만 하나의 값만 받을 수 있다.
- <property>
  - <bean>의 하위태그로 설정한 bean 객체 또는 값을 property를 통해 주입하도록 설정
  - 속성 : name – 값을 주입할 property 이름 (setter의 이름)
  - 설정 방법
    - <ref>, <value>와 같은 하위태그를 이용하여 설정
    - 속성을 이용해 설정
    - xml namespace를 이용하여 설정

# 설정을 통한 객체 주입 – Property를 이용(2/5)

## – 하위태그를 이용한 설정

- `<ref bean="bean name"/>` - 객체를 주입 시
- `<value>값</value>` - 문자(String) Primitive data 주입 시
  - type 속성 : 값의 타입을 명시해야 하는 경우 사용.

## – 속성 이용

- `ref="bean 이름"`
- `value="값"`

## – XML Namespace를 이용

- `<beans>` 태그의 스키마설정에 namespace등록
  - `xmlns:p="http://www.springframework.org/schema/p"`
- `<bean>` 태그에 속성으로 설정
  - 기본데이터 주입 : `p:propertyname="value". ex) <bean p:id>`
  - bean 주입 : `p:propertyname-ref="bean_id"`  
ex) `<bean p:dao-ref="dao">`

# 설정을 통한 객체 주입 – Property를 이용 (3/5)

- Primitive Data Type 주입

값을 주입 받을 객체

```
package vo;  
public class Person{  
    private String id,  
    private String name,  
    private int age;  
  
    public void setId(String id) {...}  
    public void setName(String name) {...}  
    public void setAge(int age) {...}
```

```
<bean id="person" class="vo.Person">  
    <property name="name">  
        <value>hong</value>  
    </property>  
    <property name="id" value="abcde"/>  
    <property name="age" value="20"/>  
</bean>
```

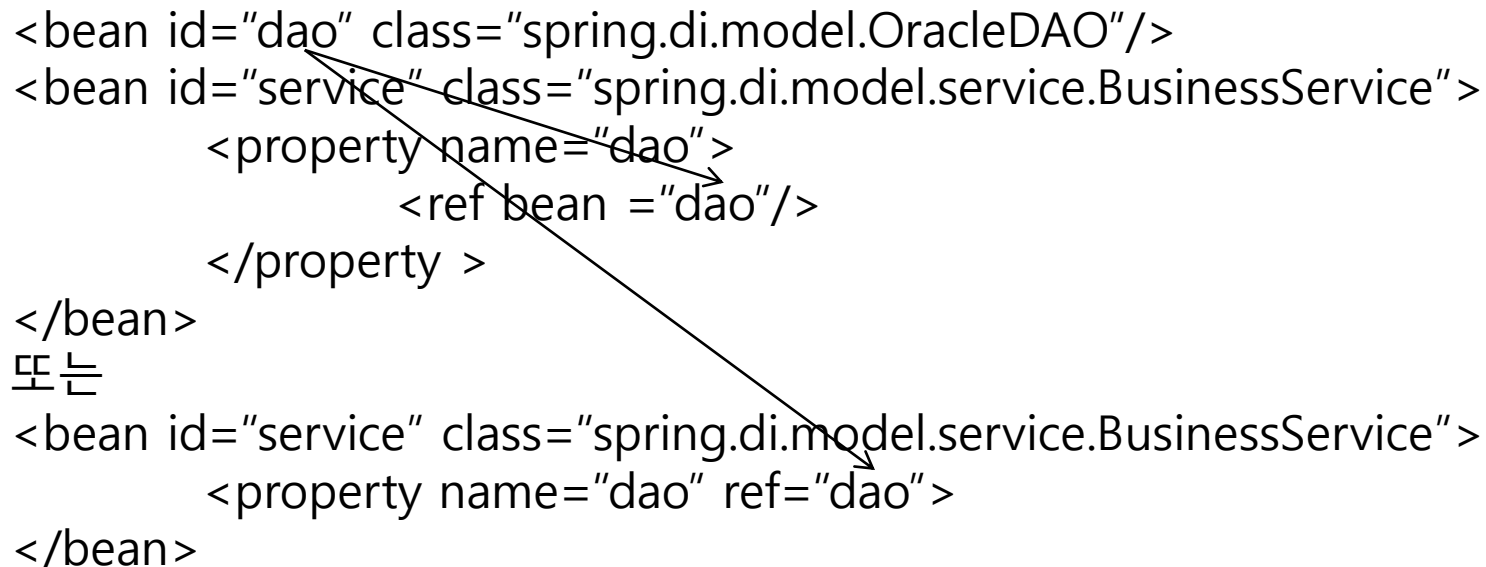
# 설정을 통한 객체 주입 – Property를 이용 (4/5)

- Bean 객체 주입

값을 주입 받을 객체

```
public class BusinessService{  
    private Dao dao = null;  
    public setDao(Dao dao){...}  
}
```

```
<bean id="dao" class="spring.di.model.OracleDAO"/>  
<bean id="service" class="spring.di.model.service.BusinessService">  
    <property name="dao">  
        <ref bean="dao"/>  
    </property>  
</bean>  
또는  
<bean id="service" class="spring.di.model.service.BusinessService">  
    <property name="dao" ref="dao">  
</bean>
```





# 설정을 통한 객체 주입 – Property를 이용 (5/5)

- XML Namespace를 이용한 주입

값을 주입 받을 객체

```
public class BusinessService{
    private Dao dao = null;
    private int waitingTime = 0;
    public setDao(Dao dao){...}
    public setWaitingTime(int wt){...}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"
       xmlns:p="http://www.springframework.org/schema/p" >

    <bean id="dao" class="spring.di.model.OracleDAO"/>
    <bean id="person" class="vo.Person"
          p:waitingTime="20"
          p:dao-ref="dao"/>

</beans>
```

# Collection 객체 주입하기 (1/5)

- <property> 또는 <constructor-arg>의 하위 태그로 Collection 값을 설정하는 태그를 이용해 값 주입 설정
- 설정 태그

| 태그      | Collection종류         | 설 명  |
|---------|----------------------|--|
| <list>  | java.util.List       | List 계열 컬렉션 값 목록 전달                            |
| <set>   | java.util.Set        | Set 계열 컬렉션 값 목록 전달                             |
| <map>   | java.util.map        | Map 계열 컬렉션에 key-value의 값 목록 전달                 |
| <props> | java.util.Properties | Properties에 key(String)-value(String)의 값 목록 전달 |

- Collection에 값을 설정 하는 태그
  - <ref> : <bean>으로 등록된 객체
  - <value> : 기본데이터
  - <bean> : 임의의 bean
  - <list>,<map>,<props>,<set> : 컬렉션
  - <null> : null

# Collection 객체 주입하기 (2/5)

- <list>
  - List 계열 컬렉션이나 배열에 값들을 넣기.
  - <ref>, <value> 태그를 이용해 값 설정
  - <ref bean="bean\_id"/> : bean 객체 list에 추가
  - <value [type="type"]>값</value> : 문자열(String), Primitive 값 list에 추가

```
public void setMyList(List list){...}
```

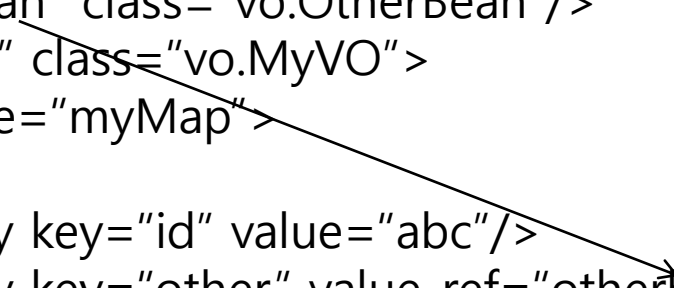
```
<bean id="otherbean" class="vo.OtherBean"/>
<bean id="myBean" class="vo.MyVO">
  <property name="myList">
    <list>
      <value>10</value> ->String으로 저장됨
      <value type="java.lang.Integer">20</value> ->Integer로 저장됨
      <ref bean="otherbean"/>
    </list>
  </property>
</bean>
```

# Collection 객체 주입하기 (3/5)

- <map>
  - Map 계열의 Collection에 객체들을 넣기
    - 속성 : key-type, value-type : key와 value의 타입을 고정시킬경우 사용
  - <entry>를 이용해 key-value를 map에 등록
    - 속성
      - key, key-ref : key 설정
      - value, value-ref : 값 설정

```
public void setMyMap(Map map){...}
```

```
<bean id="otherbean" class="vo.OtherBean"/>
<bean id="myBean" class="vo.MyVO">
  <property name="myMap">
    <map>
      <entry key="id" value="abc"/>
      <entry key="other" value-ref="otherbean"/>
    </map>
  </property>
</bean>
```



# Collection 객체 주입하기 (4/5)

- <props>
  - java.util.Properties 값(문자열)을 넣기
  - <prop>를 이용해 key-value를 properties에 등록
    - 속성
      - key : key값 설정
    - 값은 태그 사이에 넣는다. : <prop key="id">abcde</prop>

```
public void setJdbcProperty (Properties props){...}
```

```
<bean id="myDAO" class="vo.DAO">  
  <property name="jdbcProperty">  
    <props>  
      <prop key="driver">JDBC Driver</prop>  
      <prop key="url">jdbc:url://127.0.0.1/mydb</prop>  
      <prop key="user">dbUser</prop>  
      <prop key="pwd">dbPassword</prop>  
    </props>  
  </property>  
</bean>
```

# Collection 객체 주입하기 (5/5)

- <set>
  - java.util.Set에 객체를 넣기
    - 속성 : value-type : value 타입 설정
  - <value>, <ref>를 이용해 값을 넣는다.

```
public void setMySet(Set props){...}
```

```
<bean id="otherbean" class="vo.OtherBean"/>
<bean id="myBean" class="vo.Bean">
  <property name="mySet">
    <set>
      <value>10</value>
      <value>20</value>
      <ref bean="otherbean"/>
    </set>
  </property>
</bean>
```

# Bean 객체의 생성 단위 (1/2)

- BeanFactory를 통해 Bean을 요청시 객체생성의 범위(단위)를 설정
- <bean> 의 scope 속성을 이용해 설정
  - scope의 값

| 값              |                                 |
|----------------|---------------------------------|
| singleton      | 컨테이너는 하나의 빈 객체만 생성한다. - default |
| prototype      | 빈을 요청할 때 마다 생성한다.               |
| request        | Http 요청마다 빈 객체 생성               |
| session        | HttpSession 마다 빈 객체 생성          |
| global-session | 글로벌 http세션에 대해 빈 객체 생성- 포틀릿 관련  |

- request, session은 WebApplicationContext에서만 적용 가능

# 빈(bean) 생성 제어 (2/2)

- 빈(bean) 범위 지정
  - singleton과 prototype
    - `<bean id="dao" class="dao.OracleDAO" scope="prototype"/>`
    - prototype은 Spring 어플리케이션 컨텍스트에서 `getBean`으로 빈(bean)을 사용시마다 새로운 인스턴스를 생성함.
    - singleton은 Spring 어플리케이션 컨텍스트에서 `getBean`으로 빈(bean)을 사용시 동일한 인스턴스를 생성함.



# 빈(bean) 생성 제어 (3/3)

- Factory 메소드로부터 빈(bean) 생성

```
public class OracleDAO{  
    private OracleDAO() {}  
    private static OracleDAO instance;  
    public static OracleDAO getInstance(){  
        if(instance==null)  
            instance = new OracleDAO();  
        return instance;  
    }  
}
```

*Singleton 클래스는 static factory 메소드를 통해서 인스턴스 생성이 가능하면 단 하나의 인스턴스만을 생성함.*

```
<bean id="dao" class="OracleDAO"  
      factory-method="getInstance"/>
```

- 주 : getBean()으로 호출 시 private 생성자도 호출 하여 객체를 생성한다. 그러므로 위의 상황에서factory 메소드로만 호출 해야 객체를 얻을 수 있는 것은 아니다.