제3장 스프링 프레임워크 기본

- Spring의 역할
- 빈(bean) 객체 담기
- 빈(bean) 생성
- 빈(bean) 속성으로 주입
- 빈(bean) 생성 제어

Spring의 역할

- 특정 비즈니스 목표를 위해서 여러 객체들이 같이 작동함
- 서로를 알고 수행할 작업을 위해서 통신함
- Spring에서 객체들은 자신들의 작업을 하는데 있어서 필요한 다른 객체를 찾거나 생성할 책임이 없음
- 대신 컨테이너에 의해서 협업하는 객체에 대한 참조가 주어짐
- 객체간의 연관(association)을 만드는 행위가 DI의 기본이며 wiring이라고 함

빈(bean) 객체 담기 (1/8)

- Spring 컨테이너
  - Spring 프레임워크의 핵심
  - 애플리케이션을 구성하는 컴포넌트를 관리하기 위해서 의존 역행(DI)을 사용
    - 협업하는 컴포넌트 간 연관 생성 포함
    - 객체들이 깔끔하고 이해하기 쉬우며, 재사용성이 높고, 단위 테스트가 쉬움
  - 두 가지 유형의 컨테이너 구현
    - Bean factory (org.springframework.beans.factory.BeanFactory 인터페이스)
    - Application context (org.springframework.context.ApplicationContext 인터페이스)

빈(bean) 객체 담기 (2/8)

- BeanFactory
  - Factory Method 설계 패턴으로 구현
    - 빈(bean) 객체에 대한 생성과 제공을 담당
    - 단일 유형의 객체를 생성하는 것이 아니라, 여러 유형의 빈(bean)을 생성하고 제공하는 일반적인 목적의 factory
  - 초기화시 협업하는 객체들 간의 연관(association) 생성이 가능
    - 빈(bean) 자체와 빈(bean)의 클라이언트에게 설정에 대한 부담을 덜어줌
    - 객체를 제공할 때 제공되는 객체들은 완전하게 설정이 되며, 협업하는 객체에 대해서도 알고 있으며, 사용이 가능한 상태로 만들어짐
  - 빈(bean)의 생명주기 관여
    - 초기화와 소멸 메소드에 별도의 기능 추가가 가능
  - 생성시 org.springframework.work.core.io.Resource 객체를 생성자에게 넘겨줌
    - BeanFactory factory = new XmlBeanFactory(new FileSystemResource("c:/beans.xml"));
  - 빈(bean)들은 bean factory 로 "지연되게" (lazily) 로딩됨
    - 빈(bean) 정의는 바로 로딩되지만, 빈(bean)들은 필요 시점까지 초기화되지 않음
  - BeanFactory에서 빈(bean)을 초기화시키고 속성을 세팅
    - MyBean myBean = (MyBean)factory.getBean("myBean");
    - Spring 컨테이너 내의 빈(bean)의 생명주기가 시작

빈(bean) 객체 담기 (3/8)

- Resource 구현 객체

Resource 구현	목적
org.springframework.core.io.ByteArrayResource	내용이 byte 의 배열인 resource를 정의
org.springframework.core.io.ClassPathResource	클래스패스로부터 검색할 resource를 정의
org.springframework.core.io.DescriptiveResource	실질적인 읽기 가능한 resource가 아니지만 resource description을 담고있는 resource 정의
org.springframework.core.io.FileSystemResource	파일 시스템으로부터 검색할 resource를 정의
org.springframework.core.io.InputStreamResource	입력 스트림으로부터 검색할 resource를 정의
org.springframework.web.portlet.context.PortletContextResource	포틀릿 컨텍스트에서 가용한 resource를 정의
org.springframework.web.context.support.ServletContextResource	서블릿 컨텍스트에서 가용한 resource를 정의
org.springframework.core.io.UrlResource	주어진 URL으로부터 검색할 resource를 정의

빈(bean) 객체 담기 (4/8)

- Application Context
  - BeanFactory와 모든 면에서 유사
    - 빈(bean) 정의 로딩, 빈(bean) 엮음(wire), 요청에 따른 빈(bean) 제공
  - 메시지의 국제화 지원을 포함하여 텍스트 메시지를 분석하는 수단 제공
  - 이미지와 같은 파일 resource를 로딩하는 일반적인 방법 제공
  - 리스너로 등록된 빈(bean)에게 이벤트 발생 가능
  - 거의 모든 애플리케이션에서는 BeanFactory보다 더 선호
    - BeanFactory는 모바일 장치와 같이 resource가 거의 없는 환경에서 주로 사용
- 세가지 주요 사용 객체

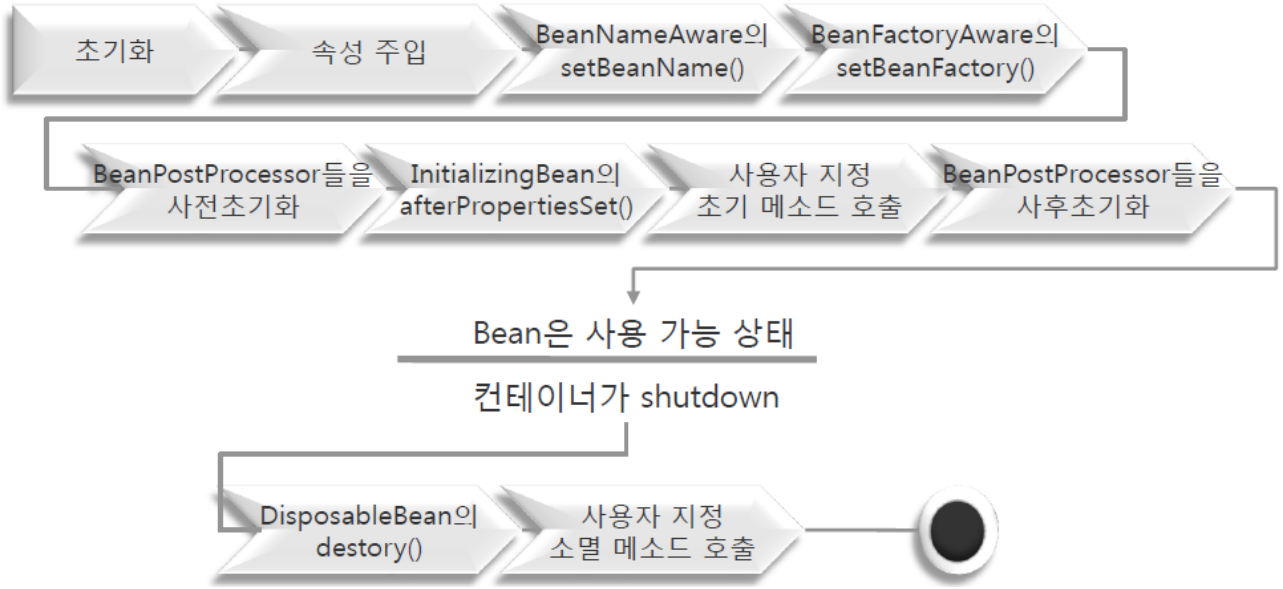
ClassPathXmlApplicationContext	컨텍스트 정의 파일을 클래스패스 reosurce로 취급하면서 클래스패스 상의 XML 파일로부터 컨텍스트 정의를 로딩
FileSystemXmlApplicationContext	파일 시스템의 XML 파일로부터 컨텍스트 정의를 로딩
XmlWebApplicationContext	웹 어플리케이션 내에 포함된 XML 파일로부터 컨텍스트 정의를 로딩

빈(bean) 객체 담기 (5/8)

- Application Context
  - 사용 예제
    - ApplicationContext context = new FileSystemXmlApplicationContext("c:/foo.xml");
    - ApplicationContext context = new ClassPathXmlApplicationContext("foo.xml");
  - getBean() 메소드를 통해서 빈(bean)을 제공
    - ApplicationContext 인터페이스는 BeanFactory 인터페이스를 확장
  - Singleton 빈(bean)에 대한 로딩 차이
    - BeanFactory는 모든 빈(bean)에 대해서 getBean() 메소드가 호출될 때까지 빈(bean) 생성을 지연시켜서 로딩
    - ApplicationContext는 모든 singleton 빈을 context 기동 시 미리 로딩
    - 필요할 때 즉시 사용 가능한 상태로 유지 – 애플리케이션이 빈(bean)이 생성될 때까지 대기할 필요가 없음

빈(bean) 객체 담기 (6/8)

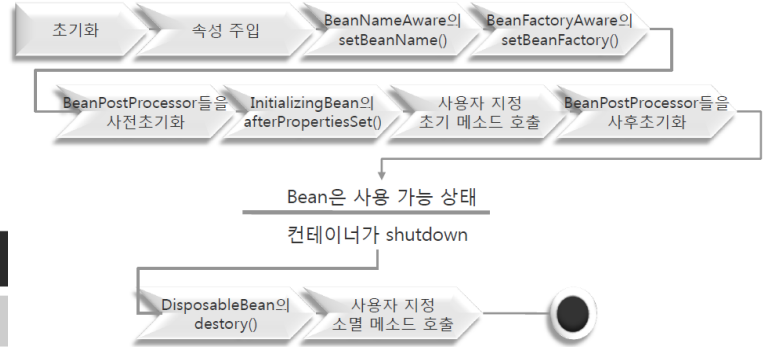
- 빈(bean) 의 생명
  - 일반적인 자바 어플리케이션
    - new 키워드를 사용해서 빈(bean)을 초기화시키면 사용 가능한 상태가 됨
    - 빈(bean)이 더 이상 사용되지 않으면, garbage collection의 처리 대상이 됨
- BeanFactory 내에서의 Spring 빈(bean)의 생명주기



빈(bean) 객체 담기 (7/8)

- 빈(bean) 의 생명
  - BeanFactory 내에서의 Spring 빈(bean)의 생명주기

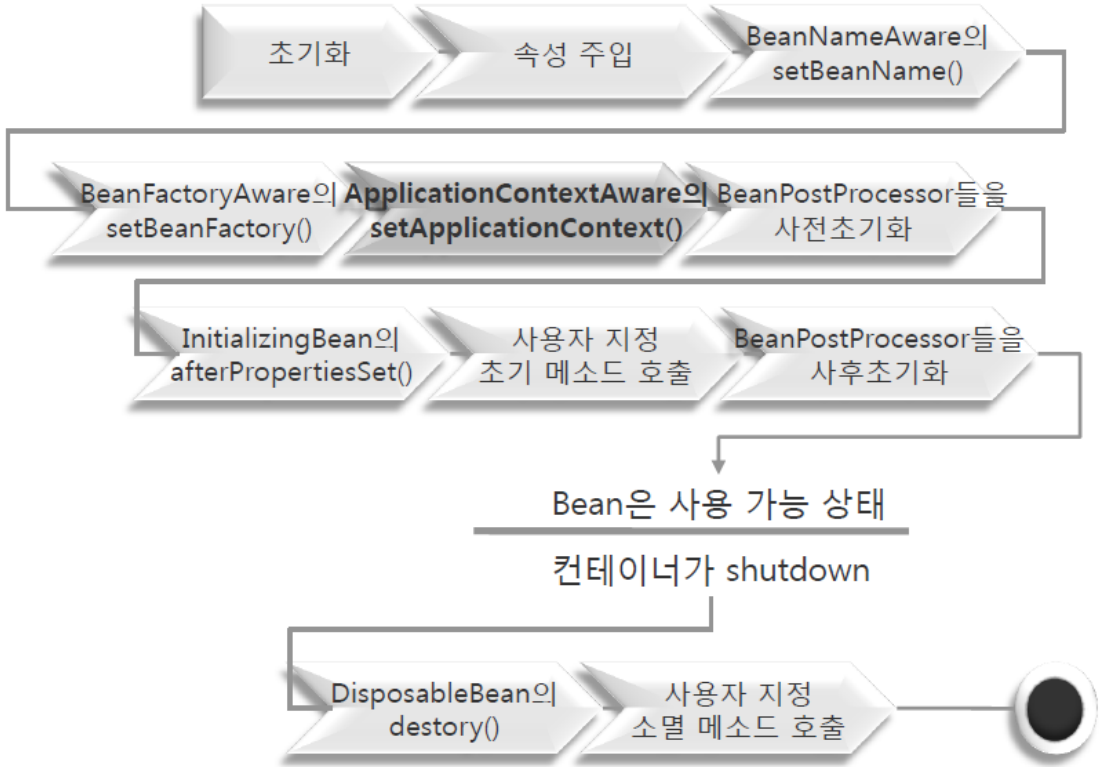
단계	설명
1. 초기화	Spring이 빈(bean)을 초기화시킴.
2. 속성 주입	Spring이 빈(bean)의 속성들을 주입함.
3. 빈(bean) 이름 세팅	빈(bean)이 BeanNameAware를 구현하고 있으면, Spring은 빈(bean)의 ID를 setBeanName() 로 넘김.
4. BeanFactory 세팅	빈(bean)이 BeanFactoryAware를 구현하고 있으면, Spring은 빈(bean) factory를 setBeanFactory() 로 넘김.
5. 사후처리 (초기화전)	BeanPostProcessor 들이 있는 경우, Spring은 이들의 postProcessBeforeInitialization() 메소드 호출
6. 빈(bean) 초기화	빈(bean)이 InitializingBean을 구현하고 있으면, afterPropertiesSet() 메소드 호출됨. 빈(bean)이 사용자 지정 초기 메소드가 선언되어 있으며, 특정 초기화 메소드가 호출됨.
7. 사후처리 (초기화후)	BeanPostProcessor 들이 있는 경우, 이들의 postProcessAfterInitialization() 메소드 호출
8. 빈(bean) 사용 대기	이 시점에 빈(bean)은 어플리케이션에 의해서 사용가능한 상태로 대기하며 더 이상 필요 없을 때까지 bean factory 내에 남아 있음.
9. 빈(bean) 소멸	빈(bean)이 DisposableBean을 구현하고 있으며, destroy() 메소드가 호출됨. 빈(bean)이 사용자 지정 destroy-메소드가 선언되어 있으면, 특정 메소드가 호출됨.

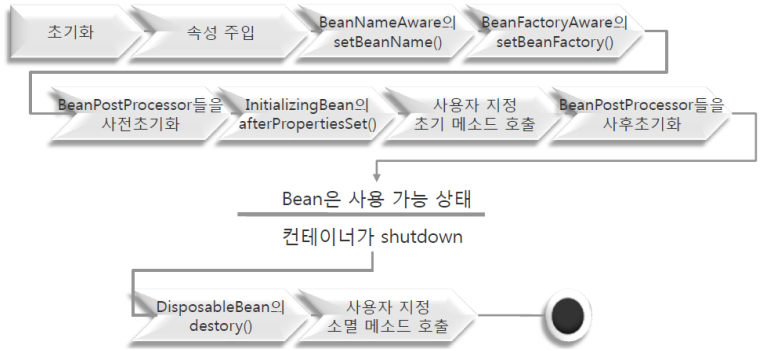


BeanFactory 내에서의 Spring 빈(bean)의 생명주기

빈(bean) 객체 담기 (8/8)

- 빈(bean) 의 생명
  - ApplicationContext 내에서의 Spring 빈(bean)의 생명주기





* BeanFactory 내에서의 Spring 빈(bean)의 생명주기

빈(bean) 생성 (1/6)

- 빈(bean) 선언
  - 인터페이스

```
public interface Performer {
        void perform() throws PerformanceException;
}
```

  - 단순한 빈(bean)

```
public class Juggler implements Performer {
        private int beanBags = 3;
        public Juggler() {}
        public Juggler(int beanBags) {
                this.beanBags = beanBags;
        }

        @Override
        public void perform() throws PerformanceException {
                System.out.println("JUGGLING " + beanBags + " BEANBAGS");
        }
}
```

빈(bean) 생성 (2/6)

- 빈(bean) 선언
  - Spring 설정 파일

    <bean id="duke" class="kr.kim.example.performer.Juggler"/>

    new kr.kim.example.performer.Juggler();
    - 생성자를 private이나 protected 등으로 변경
  - Spring application context 로딩

    ApplicationContext ctx = new ClassPathXmlApplicationContext("kr/…/spring-idol.xml");

    Performer performer = (Performer)ctx.getBean("duke");

    performer.perform();
  - 생성자를 통한 주입

    <bean id="duke" class="kr.kim. example.performer.Juggler">

    <constructor-arg value="15"/>

    </bean>

빈(bean) 생성 (3/6)

- 빈(bean) 선언
  - 생성자에 객체 참조 주입

```
public class PoeticJuggler extends Juggler {
      private Poem poem;
                  public PoeticJuggler(Poem poem) {
                  super();
                  this.poem = poem;
      }
      public PoeticJuggler(int beanBags, Poem poem) {
                  super(beanBags);
                  this.poem = poem;
      }

      @Override
      public void perform() throws PerformanceException {
                  super.perform();
                  System.out.println("WHILE RECITING...");
                  poem.recite();
      }
}
```

다른 객체를 파라미터로 가지는 생성자

빈(bean) 생성 (4/6)

- 빈(bean) 선언
  - 참조 클래스

```
public interface Poem {
        void recite();
}
```

```
public class Sonet29 implements Poem {
        private static String[] LINES = {
                "When, in disgrace with fortune and men's eyes,",
                "I all alone beweep my outcast state",
                "That then I scorn to change my state with kings."
        };
        public Sonet29() {}

        @Override
        public void recite() {
                for (String line : LINES) {
                        System.out.println(line);
                }
        }
}
```

빈(bean) 생성 (5/6)

- 빈(**bean**) 선언
  - Spring 설정파일

```
<bean id="sonet29" class="kr.kim.example.performer.Sonet29"/>
<bean id="duke" class="kr.kim.example.performer.PoeticJuggler">
        <constructor-arg value="15"/>
        <constructor-arg ref="sonet29"/>
</bean>
```

// 파라미터의 순서가 맞아야 됨

빈(bean) 생성 (6/6)

- 생성자 주입 **(constructor-injection)**
  - 강한 의존관계를 맺음
    - 의존관계 없이 초기화될 수 없음
  - 생성자를 통해 의존관계가 세팅 되기 때문에
    - 부수적인 셋터가 필요 없어서 코드 라인 수가 줄어듬
    - 의존 관계의 속성을 읽기만 가능한 (immutable) 상태로 만들어서 변경을 방지할 수 있음
- 셋터 주입 **(setter-injection)**
  - 여러 유형의 생성자가 나타날 수 있기 때문에 어떤 생성자를 사용할지에 대해서 사용하는 입장에서 헷갈릴 수 있음
  - 비슷한 유형의 파라미터가 두 개 이상이 존재 시 각각의 파라미터가 어떤 목적으로 있는지 결정하기 어려움
  - 생성자 주입은 상속관계에 있어서 상위의 private 속성을 접근하기 위해서 super() 를 사용해야 함

빈(bean) 속성으로 주입 (1/11)

- 예제

```
public interface Instrument {
        void play();
}
```

```
public class Instrumentalist implements Performer {
    public Instrumentalist() { }
    @Override
    public void perform() throws PerformanceException {
        System.out.print("Playing " + song + " : ");
        instrument.play();
    }
    private String song;
    public void setSong(String song) { this.song = song; }
    private Instrument instrument;
    public void setInstrument(Instrument instrument) {
        this.instrument = instrument;
    }
}
```

- <bean id="kenny" class="kr.kim.example.performer.Instrumentalist"/>

빈(bean) 속성으로 주입 (2/11)

- 단순한 값 주입

```
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist">
        <property name="song" value="Jingle Bell"/>
</bean>
```

```
Instumentalist kenny = new Instumentalist();
kenny.setSong("Jingle Bells");
```

```
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist">
        <property name="song" value="Jingle Bell"/>
        <property name="age" value="37"/>
</bean>
```

//String 뿐만 아니라 숫자형, Boolean도 지정이 가능

빈(bean) 속성으로 주입 (3/11)

- 다른 객체 참조

```
public class Saxophone implements Instrument {
    public Saxophone() {}

    @Override
    public void play() {
            System.out.println("TOOT TOOT TOOT");
    }
}
```

```
public class Piano implements Instrument {
    public Piano() {}

    @Override
    public void play() {
            System.out.println("PLINK PLINK PLINK");
    }
}
```

```
<bean id="saxophone" class="kr.kim.example.performer.Saxophone"/>
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist">
        <property name="song" value="Jingle Bell"/>
        <property name="instrument" ref="saxophone"/>
</bean>
```

//다른 객체에 대한 참조

```
<bean id="piano" class="kr.kim.example.performer.Piano"/>
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist">
        <property name="song" value="Jingle Bell"/>
        <property name="instrument" ref="piano"/>
</bean>
```

빈(bean) 속성으로 주입 (4/11)

- 내부 **빈(inner bean)** 주입
  - 내부 클래스(inner class)와 마찬가지로 특정 빈(bean) 범위 내에서만 정의되는 빈(bean)

```
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist">
        <property name="song" value="Jingle Bell"/>
        <property name="instrument”>
            <bean class="kr.kim.example.performer.Saxophone"/>
        <property/>
</bean>
```

```
<bean id=“duke" class="kr.kim.example.performer.PoeticJuggler">
        <constructor-arg value=“15"/>
        <constructor-arg>
            <bean class="kr.kim.example.performer.Sonnet29"/>
        </constructor-arg>
</bean>
```

//내부 빈(inner bean)

- 내부 빈(inner bean)은 id 속성을 가지지 않음
  - 재사용이 불가능
  - 한번만 주입을 사용하고 다른 빈에서 참조할 수 없음

빈(bean) 속성으로 주입 (5/11)

- Collection 엮음 (**wiring**)
  - Collection 값에 대한 설정으로 Spring은 4가지 유형을 제공

Collection 요소	사용 목적
<list>	중복을 허용하는 list 값을 엮음 (wiring)
<set>	중복을 허용하지 않는 set 값을 엮음 (wiring)
<map>	name과 value가 특정 타입이 아닌 name-value 쌍의 collection 값을 엮음 (wiring)
<props>	name과 value가 String 타입인 name-value 쌍의 collection 값을 엮음(wiring)

빈(bean) 속성으로 주입 (6/11)

- 예제

```
public class OneManBand implements Performer {
    public OneManBand() {}
    @Override
    public void perform() throws PerformanceException {
        for (Instrument instrument : instruments) {
            instrument.play();
        }
    }
    private Collection<Instrument> instruments;
    public void setInstruments(Collection<Instrument> instruments) {
        this.instruments = instruments;
    }
}
```

//Collection 값에 대한 주입

빈(bean) 속성으로 주입 (7/11)

- **List 와 Array**

```
<bean id="hank" class="kr.kim.spring.example.OneManBand">
    <property name="instruments">
            <list>
                    <ref bean="guitar"/>
                    <ref bean="cymbal"/>
                    <ref bean="harmonica"/>
            </list>
    </property>
</bean>
```

- <list> 태그
  - <ref> 이외에도 <value>, <bean>, <null/> 등이 사용될 수 있음
  - 또 다른 <list>를 사용할 수도 있음
- java.util.Collection 혹은 배열에서 사용 가능
  - java.util.List<Instrument> insturments;
  - Instrument[] instruments;

빈(bean) 속성으로 주입 (8/11)

- **Set**

```
<bean id="hank" class="kr.kim.example.performer.OneManBand">
    <property name="instruments">
            <set>
                    <ref bean="guitar"/>
                    <ref bean="cymbal"/>
                    <ref bean="harmonica"/>
                    <ref bean="harmonica"/>
            </set>
    </property>
</bean>
```

- 위의 예제에서 harmonica는 중복되는 값으로 set의 중복을 허용하지 않는 성질로 인해서 나머지 하나는 무시하고 하나만 set 값에 담김

빈(bean) 속성으로 주입 (9/11)

- **Map**

```
public class OneManBand implements Performer {
    public OneManBand() {}
    @Override
    public void perform() throws PerformanceException {
            for (String key : instruments.keySet()) {
                System.out.println(key + " : ");
                instruments.get(key).play();
            }
    }

    private Map<String, Instrument> instruments;
    public void setInstruments(Map<String, Instrument> instruments) {
            this.instruments = instruments;
    }
}
```

속성	목적
key	Map 의 key값으로 String인 경우
key-ref	Map 의 key값으로 Spring 컨텍스트의 빈(bean) 참조
value	Map 의 value값으로 String인 경우
value-ref	Map 의 value값으로 Spring 컨텍스트의 빈(bean) 참조

```
<bean id="hank"  class="kr.kim.example.performer.OneManBand">
    <property name="instruments">
            <map>
                <entry key="GUITAR" value-ref="guitar"/>
                <entry key="CYMBAL" value-ref="cymbal"/>
                <entry key="HARMONICA" value-ref="harmonica"/>
            </map>
    </property>
</bean>
```

빈(bean) 속성으로 주입 (10/11)

- **Properties**

<property>	빈(bean) 클래스 속성으로 값을 주입시 사용
<props>	java.util.Properties 타입의 collection 값을 정의시 사용
<prop>	<props> collection의 각각의 값 정의시 사용

```
public class OneManBand implements Performer {
    public OneManBand() {}
    @Override
    public void perform() throws PerformanceException {
            for (Iterator iter = instruments.keySet().iterator(); iter.hasNext(); ) {
                    String key = (String)iter.next();
                    System.out.println(key + " : " + instruments.getProperty(key));
            }
    }

    private Properties instruments;
    public void setInstruments(Properties instruments) {
            this.instruments = instruments;
    }
}
```

```
<bean id="hank"  class="kr.kim.example.performer.OneManBand">
    <property name="instruments">
            <props>
                <prop key="GUITAR">STRUM STRUM STRUM</prop>
                <prop key="CYMBAL">CRASH CRASH CRASH</prop>
                <prop key="HARMONICA">HUM HUM HUM</prop>
            </props>
    </property>
</bean>
```

빈(bean) 속성으로 주입 (11/11)

- null 엮음 (**wiring**)
  - 속성에 대해서 명시적으로 null 값을 세팅시
    - <property name="someNotNullProperty"><null/></property>
  - 자동 엮음(autowiring) 을 오버라이딩시 속성에 null 값을 세팅

빈(bean) 생성 제어 (1/5)

- 빈(bean) 범위 지정
  - Singleton 빈(bean)
    - 기본적으로 모든 Spring 빈(bean)들은 singleton 임
    - 컨테이너가 bean을 제공시 빈(bean)의 인스턴스는 항상 동일함
  - 5가지 빈(bean) 범위

범위	내용
singleton	Spring 컨터이너당 단일 인스턴스에 대한 빈(bean) 정의를 지정 (기본값)
prototype	빈(bean)이 몇번이고(사용할 때마다) 초기화
request	HTTP 요청에 빈(bean) 정의를 지정. 웹에서 Spring 컨텍스트를 사용하는 경우 (Spring MVC와 같은).
session	HTTP 세션에 빈(bean) 정의를 지정. 웹에서 Spring 컨텍스트를 사용하는 경우 (Spring MVC와 같은).
global-session	전역 HTTP 세션에 빈(bean) 정의를 지정. 포틀릿 컨텍스트에서 사용되는 경우

빈(bean) 생성 제어 (2/5)

- 빈(**bean**) 범위 지정
  - Singleton과 prototype
    - <bean id="saxophone" class="…Saxophone" **scope="prototype"**/>
    - prototype은 Spring 애플리케이션 컨텍스트에서 getBean으로 빈(bean)을 사용시 새로운 인스턴스를 생성함
    - singleton은 Spring 애플리케이션 컨텍스트에서 getBean 으로 빈(bean)을 사용시 동일한 인스턴스를 생성함
    - Spring 설정 XML 파일이 여러 개라 하더라도 singleton은 항상 동일한 인스턴스임

빈(bean) 생성 제어 (3/5)

- Factory 메소드로부터 빈(**bean**) 생성

```
public class Stage {
    private Stage() {}
    private static class StageSingletonHolder {
            static Stage instance = new Stage();
    }
    public static Stage getInstance() {
            return StageSingletonHolder.instance;
    }
}
```

/* **Singleton** 클래스는 **static factory** 메소드를 통해서
인스턴스 생성이 가능하며 단 하나의 인스턴스만을 생성함 */

- <bean id="theStage" class="…Stage" **factory-method="getInstance"**/>

빈(bean) 생성 제어 (4/5)

- 빈 **(bean)** 초기화와 소멸
  - 빈(bean) 생성시점과 소멸시점에 특별한 행위를 넣고 싶은 경우에 사용

```
public class Instrumentalist implements Performer {
…
    public void tuneInstrument() {
            instrument.tune();
    }
…
    public void cleanInstrument() {
            instrument.clean();
    }
}
```

```
<bean id="kenny" class="kr.kim.example.performer.Instrumentalist“
            init-method=“tuneInstrument“ destroy-method=“cleanInstrument">
    <property name="song" value="Jingle Bell"/>
    <property name="instrument” ref=“saxophone“/>
</bean>
```

```
<?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-2.0.xsd"
        default-init-method=“tuneInstrument”
        default-destroy-method=“cleanInstrument”>
        …
</beans>
```

빈(bean) 생성 제어 (5/5)

- 빈 **(bean)** 초기화와 소멸
  - InitializingBean과 DisposableBean

```
public class Instrumentalist implements Performer, InitializingBean, DisposableBean {
…
    public void afterPropertiesSet() throws Exception {
            instrument.tune();
    }
…
    public void destroy() throws Exception {
            instrument.clean();
    }
}
```

  - Spring 설정 XML에 별도로 지정하지 않더라도 자동으로 초기화와 소멸 메소드가 실행됨
  - 애플리케이션이 Spring 의 API에 종속됨
    - Spring 컨테이너 내에서 특별하게 사용되는 프레임워크 bean을 개발시 사용