Spring 4화

Bean Factory(Bean 부분)

Spring(3화) 목차

- Bean 이란
- Bean 속성 테스트
  -

Bean?

단순히 스프링 IoC 컨테이너에서 존재하는 인스턴트

- Bean 이라 부르는 이유 : 커피의 주원료가 콩이라서
- Java : Oak라고 불리어 졌으나, 이미 다른 곳에서 쓰고 있어 가장 많이 소비되어지는 커피가 자바커피라서, 많이 사용되어지는 언어가 되라는 의미에서 지어짐.
- 그럼 Jar(Java Archive)는?

Jar!!!

커피 콩을 담아두는 용기

SIZE

133ml
362ml
798ml
1200ml

Spring Bean이란?

- 특징
  - Class가 객체화 되어 IoC 컨테이너에 들어가 있는 객체 인스턴스
  - Scope를 지니며, Scope별로 life cycle을 가진다.
  - 이름을 가지며 해당 이름으로 Bean 식별한다.
  - IoC 컨테이너에 의해 의존성을 주입받을수 있다.
- 속성
  - Class : Class의 지성
  - Name : 구별할 수 있는 이름
  - Scope : Bean Life cycle 지정
  - constructor arguments : Bean 생성시 주입될 값
  - properties : 객체에 주입될 값
  - autowiring mode, lazy-initialization mode, initialization method, destruction method

Spring Bean이란?

- 특징
  - Class가 객체화 되어 IoC 컨테이너에 들어가 있는 객체 인스턴스
  - Scope를 지니며, Scope별로 life cycle을 가진다.
  - 이름을 가지며 해당 이름으로 Bean 식별한다.
  - IoC 컨테이너에 의해 의존성을 주입받을수 있다.
- 속성
  - Class : Class의 지성
  - Name : 구별할 수 있는 이름
  - Scope : Bean Life cycle 지정
  - constructor arguments : Bean 생성시 주입될 값
  - properties : 객체에 주입될 값
  - autowiring mode, lazy-initialization mode, initialization method, destruction method

Spring Bean이란?

- Xml이나 Annotation으로 정의된 Bean은 BeanFactory의 BeanDefinitons에 들어가 있다.
  - org.springframework.beans.factory.config.BeanDefinition
  - 해당 메소드를 확인하면 Bean의 정의가 어떤식으로 저장되고 불리어 지는지 확인 가능하다.
  - 기본 정의시 지정된 값들

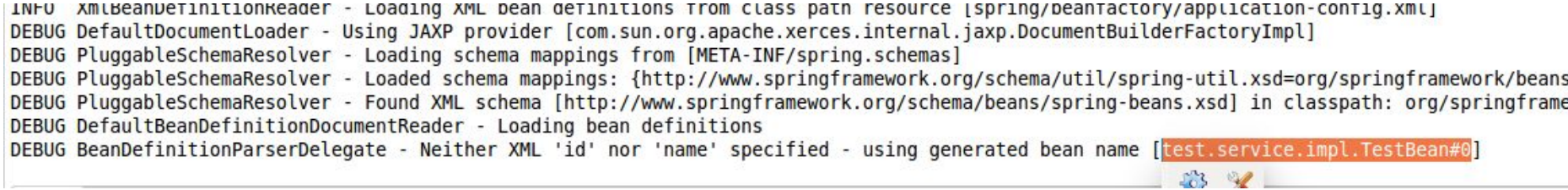
```
@Test
public void DefaultBeanTest(){
    ClassPathXmlApplicationContext ac = new ClassPathXmlApplicationContext("spring/beanfactory/application-

    ConfigurableListableBeanFactory dlbf = ac.getBeanFactory();
    BeanDefinition bd = dlbf.getBeanDefinition("testBean");
    logger.debug("getConstructorArgumentValues : " + String.valueOf(bd.getConstructorArgumentValues().getAr
    logger.debug("getScope : " + String.valueOf(bd.getScope()));
    logger.debug("getRole : " + String.valueOf(bd.getRole()));
    logger.debug("getBeanClassName : " + String.valueOf(bd.getBeanClassName()));
    logger.debug("getDescription : " + String.valueOf(bd.getDescription()));
    logger.debug("getFactoryBeanName : " + String.valueOf(bd.getFactoryBeanName()));
    logger.debug("isLazyInit : " + String.valueOf(bd.isLazyInit()));
    logger.debug("isAutowireCandidate : " + String.valueOf(bd.isAutowireCandidate()));
    logger.debug("isPrototype : " + String.valueOf(bd.isPrototype()));
    logger.debug("isSingleton : " + String.valueOf(bd.isSingleton()));
```

```
)EBUG BeanFactoryTest - getScope :
)EBUG BeanFactoryTest - getRole : 0
)EBUG BeanFactoryTest - getBeanClassName : test.service.impl.TestBean
)EBUG BeanFactoryTest - getDescription : null
)EBUG BeanFactoryTest - getFactoryBeanName : null
)EBUG BeanFactoryTest - isLazyInit : false
)EBUG BeanFactoryTest - isAutowireCandidate : true
)EBUG BeanFactoryTest - isPrototype : false
)EBUG BeanFactoryTest - isSingleton : true
```

Spring Bean 테스트

- class속성만 있다면?
  - <bean class = "test.service.impl.TestBean" />
  - BeanDefinitionReaderUtils.generateBeanName
  - test.service.impl.TestBean#0



Spring Bean Name 테스트

- 이름이 같다면?
  - <bean name="testSameName" class = "test.service.impl.TestBean" />
  - <bean name="testSameName" class = "test.service.impl.TestBean" />
  - 에러가 발생한다.(BeanDefinitionParserDelegate.checkNameUniqueness)
  - 그렇다면 Name은 같고 Id가 다르다면?
  - <bean id = "testSameName2" name="testSameName" class = "test.service.impl.TestBean" />
  - <bean id = "testSameName1" name="testSameName" class = "test.service.impl.TestBean" />
  - 에러가 발생한다.
- 그럼 Id와 Name차이는 멀까?
  - 사용 하는 것은 같다. 둘다 객체를 구분해주는 식별자다.
  - id는 Bean당 하나, Name 여러개 가능, 하지만 중복이 되면 안된다.

Spring Bean Scope란?

- Bean이 생성되어지는 범위, 생존 주기?
  - singleton(default) : Context당 한개
  - prototype : getBean이 될때 마다 생성
  - request(웹기반) : 요청당 생성
  - session(웹기반) : 세션당 생성
  - global session(웹기반) : 포틀렛 기반이라 아마 사용안할것 같다. 써본적이 없다.
  - SimpleThreadScope : Tread단위로 생성하게 한다.

Spring Bean Scope 테스트

```
<bean id = "testBeanSingleton" class = "test.service.impl.TestBean" scope="singleton"/>
<bean id = "testBeanPrototype" class = "test.service.impl.TestBean" scope="prototype"/>
<bean id = "testBeanThreadtype" class = "test.service.impl.TestBean" scope="thread">
       <aop:scoped-proxy/>
</bean>
```

```
final ConfigurableListableBeanFactory clbf = ac.getBeanFactory();
logger.debug(clbf.getBean("testBeanSingleton").toString());
logger.debug(clbf.getBean("testBeanSingleton").toString());

logger.debug(clbf.getBean("testBeanPrototype").toString());
logger.debug(clbf.getBean("testBeanPrototype").toString());

logger.debug(clbf.getBean("testBeanThreadtype").toString());
logger.debug(clbf.getBean("testBeanThreadtype").toString());
```

Spring Bean Scope 주의할점

- 주의할점
  - singleton을 제외한 scope를 무분별하게 사용한다면 heapMemory가 동이난다.
  - protoType만으로 가능한지 한번더 고려해본다.
  - prototype을 제외한 request이하의 request나 session attribute에서 사용되어 지기 때문에 해당 부분들의 덩치가 예상하지 못하게 커질수 있다.

Spring Bean FactoryMethod

- Bean을 FactoryMethod를 지정하여 생성할수 있다.

```java
public class TestBean {
    static Logger logger = LoggerFactory.getLogger(TestBean.class);

    private String name;

    public static TestBean getTestBean(){
        logger.debug("TestBeanFactory");
        return new TestBean();
    }
```

```xml
<bean id = "testBeanSingletonByFactoryBean" class = "test.service.impl.TestBean" scope="singleton" factory-method="getTestBean"/>
<bean id = "testBeanPrototypeByFactoryBean" class = "test.service.impl.TestBean" scope="prototype" factory-method="getTestBean"/>
```

```java
ConfigurableListableBeanFactory clbf = ac.getBeanFactory();
clbf.getBean("testBeanSingletonByFactoryBean");
clbf.getBean("testBeanSingletonByFactoryBean");

clbf.getBean("testBeanPrototypeByFactoryBean");
clbf.getBean("testBeanPrototypeByFactoryBean");
```