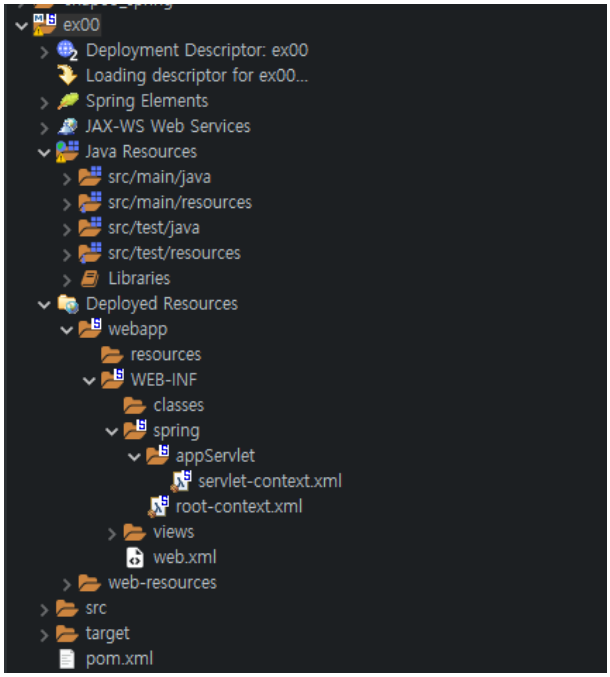
Spring Project 구성요소

생성일	@2022년 8월 9일 오후 7:41
태그	

1. 스프링 프로젝트 생성



pom.xml

-. Maven이 사용하는 pom.xml

web.xml

-. Tomcat의 web.xml

root-context.xml

-. 스프링 설정파일

servlet-context.xml

-. 웹과 관련된 스프링 설정 파일

2. 스프링 주요 특징

(1) POJO 기반의 구성

-. 별도의 API등을 사용하지 않는 POJO (Plane Old Java Object)의 구성만으로 가능하도록 제작

-. 즉, 특정 라이브러리등에 종속적이지 않다는 것을 의미한다.

(2) 의존성 주입 (DI)

-. 의존성 (Dependency) : 하나의 객체가 다른 객체 없이 제대로 된 역할을 할 수 없다.

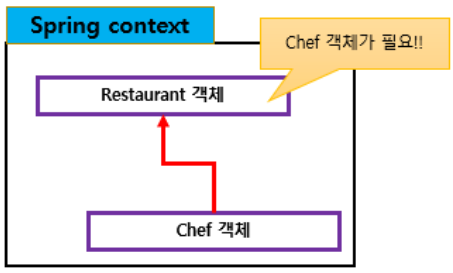
-. 주입 (Injection) : 외부에서 밀어 넣는것

-. 즉 의존성 주입이란, **"외부에서" 어떤 객체가 필요하다고 하는 객체를 직접 "밀어주는 것"** 이다.

-. Bean : 스프링에서 ApplicationContext에서 관리하는 객체를 의미한다.

-. 이러한 빈과 빈의 관계를 XML 설정 / 어노테이션 설정 / Java 설정 파일로 의존관계를 처리할 수 있다.

3. 의존성 주입 테스트



Restaurant 객체는 Chef객체가 필요한 상황이다.

1. **Restaurant과 Chef객체가 Spring에서 관리를 할 객체임을 표시하여야 한다. 이 때 어노테이션 @Component를 클래스위에 붙여주어야 한다.**
2. **Restaurant는 Chef객체가 필요한 상황이므로, 이를 표시해주어야 하는데, 이 때 사용하는 방법은 (1) setter 메소드를 이용한 주입방식 / (2) 생성자를 이용한 주입방식이 있다.**

```
@Component
@Data
public class Restaurant {

  @Setter(onMethod_= @Autowired)
  private Chef chef;
}
```

```
@Component
@Data
public class Chef {

}
```

3. **앞서 생성한 Spring에서 관리를 할 객체를 찾아주어야 한다. 이러한 설정은 root-context.xml에 설정한다**.

-. **context:component-scan 태그**에 base-package 속성으로 빈으로 사용할 객체가 들어있는 패키지명을 쓴다.

-. 패키지 이하 자바파일에서 빈 객체들을 찾는다는 의미이다.

-. 즉, 스프링은 어노테이션 @Component 존재하는 클래스의 인스턴스를 생성하게 된다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd">

  <!-- Root Context: defines shared resources visible to all other web components -->

  <context:component-scan base-package="org.zerock.myapp"></context:component-scan>

</beans>
```

4. **Restaurent객체는 Chef 객체가 필요하다는 어노테이션 @Autowired 설정이 있으므로, 스프링은 Chef 객체를 Restaurant 객체에 주입한다.**

4. 테스트 코드 작성

-. Junit5를 이용한 테스트 코드 작성

```
import static org.junit.jupiter.api.Assertions.assertNotNull;

import java.util.concurrent.TimeUnit;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.TestInstance.Lifecycle;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.Timeout;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;

import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.extern.log4j.Log4j2;

// TEST 코드 작성 시, 필요한 어노테이션
@Log4j2         // Log출력을 위한 Log4j2
@NoArgsConstructor    // 기본 Default 생성자

@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations= {
//    필요한 스프링 설정파일을 등록해줌. 이때, file: 이 사용되는데,
//    이 file: 의 의미는 "프로젝트 폴더와 같음
    "file:src/main/webapp/WEB-INF/spring/root-context.xml"
})


@TestInstance(Lifecycle.PER_CLASS)    // Instance를 Class단위로 생성
```

```
@TestMethodOrder(OrderAnnotation.class)   // 테스트 순서 작성
public class Ex00_Tests {

  // 현재 이 Test에는 Restaurant 객체가 필요하다.
  // 이를 위해, Setter를 통해 객체를 주입받는다
  @Setter(onMethod_ = {@Autowired})
  private Restaurant restaurant;

  @Test
  @Order(1)
  @DisplayName("1. testExist")
  @Timeout(value=3, unit=TimeUnit.SECONDS)
  public void testExist() {
    log.trace("testExise() invoked.");

    assertNotNull(restaurant);
    log.info("retaurant : {}", restaurant);
    log.info("restaurant.getChef()", restaurant.getChef());

  } // testExist

} // end class
```

5. Spring관련 어노테이션

@Component

-. 해당 클래스가 스프링에서 객체로 만들어 관리하는 대상임을 명시하는 어노테이션이다.

-. 즉, 클래스에 해당 어노테이션이 적용되면, 스프링에서 관리하는 Bean 객체가 된다는 의미이다.

-. 빈 객체는 Spring의 root-context.xml에서 설정한다.

@Autowired

-. 스프링에, 자신이 특정한 객체에 의존한다는 의미로, 자신에게 해당 타입의 객체(빈)을 주입하라는 의미이다.

-. 위 Restaurant객체에서, Chef 객체가 필요하다고 Setter를 통해, 쓰이는 것을 볼 수 있다.

```
@Component
@Data
public class Restaurant {

  @Setter(onMethod_= @Autowired)
  private Chef chef;
}
```

6. 의존성 주입하는 방법 2가지 (Setter / 생성자)

1. Setter 메소드를 통한 의존성 주입방법

-. 위 Restaurant 빈에서는 @Setter(onMethod_=@Autowired)를 통해, 의존성 주입을 하였다.

2. 생성자를 통한 의존성 주입방법

-. @Autowired를 사용하지 않고, 생성자를 통해, 의존성을 주입할 수 있다.

```
@Component
@Data
public class Restaurant {

  private Chef chef;

  public Restaurant (Chef chef) {
    this.chef = chef;
  } // Constructor
} // end class
```