Spring Framework
- 스프링 DI

CONTENTS

- 의존
- DI를 통한 의존 처리
- 스프링의 DI 설정
- 두 개 이상의 설정파일 사용하기

의존

- **스프링의 주요 기능중의 하나는 DI 패턴을 지원하는 것이다.**
  - **스프링 컨테이너는 설정 파일을 기반으로 빈 객체를 저장하고 있으며, 각 객체간의 의존 관계를 관리해 준다.**
  - DI 개념을 적용한 프로그램들은 이 개념을 적용하지 않은 프로그램들에 비해 **설계가 쉽고, 추후 이미 개발되어 있는 프로그램에 변경 사항이 발생했을 경우라 하더라도 변경 내용 적용이 용이하므로 확장성이 매우 좋음**
  - 각 객체간의 의존 관계와 객체들의 생명주기를 Spring을 기반으로 간편하게 개발 및 유지 보수 하는 메커니즘을 제공받는 것

의존

- **의존성 주입**
  - Spring Framework가 지원하는 핵심 기능
  - 객체 사이의 의존 관계가 객체 자신이 아닌 **외부(조립기)에 의해 설정됨**
  - **외부 조립기는 xml 설정 파일을 기반으로 의존 설정을 함.**
  - Spring은 객체를 생성하고 객체간의 의존 관계를 자동 설정해주는 **조립기 기능**을 보유한 컨테이너가 자동으로 제공(주입)
    - 조립기(Assembler) : 객체를 생성하고 객체간의 의존 관계를 자동 설정해 줌
  - **컨테이너의 역할**
    - **A 객체가 필요로 하는 의존 관계에 있는 다른 객체 B 객체를 직접 생성하여 A 객체로 주입(설정)해주는 역할을 담당**

의존관계

```
public class MemberRegisterService {

        // 의존관계!! → 의존 객체를 직접 생성
        private MemberDao memberDao = new MemberDao();

        public void regist(RegisterRequest req) {
                Member member = memberDao.selectByEmail(req.getEmail());
                if (member != null) {
                        throw new AlreadyExistingMemberException("dup email "+req.getEmail());
                }
                Member newMember = new Member(
                                req.getEmail(), req.getPassword(), req.getName(),
                                new Date());
                memberDao.insert(newMember);
        }
}
```

DI를 통한 의존 처리

```
public class MemberRegisterService {
        private MemberDao memberDao;

        // 의존 관계를 생성자를 통해 주입한다.
        public MemberRegisterService(MemberDao memberDao) {
                this.memberDao = memberDao;
        }

        public void regist(RegisterRequest req) {
                Member member = memberDao.selectByEmail(req.getEmail());
                if (member != null) {
                        throw new AlreadyExistingMemberException("dup email "+req.getEmail());
                }
                Member newMember = new Member(
                        req.getEmail(), req.getPassword(), req.getName(), new Date());
                memberDao.insert(newMember);
        }
}
```

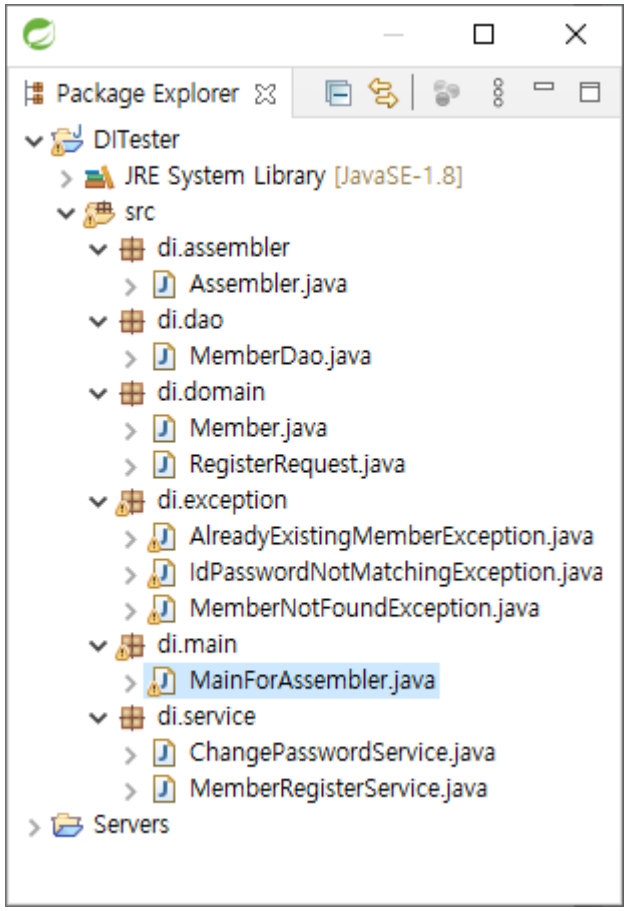
DI를 통한 의존 처리

컨테이너 조립기가 하는일

```
//1. 의존관계를 생성자를 통해서 주입
MemberDao memberDao = new MemberDao();

//2. 의존관계를 생성자를 통해서 주입
MemberRegisterService svc = new MemberRegisterService(memberDao)
```

의존관계 예제

- 회원 데이터 관련 클래스
  - Member
  - RegisterRequest
  - IdPasswordNotMatchingException
  - MemberDao
- 회원 가입 처리 관련 클래스
  - AlreadyExistingMemberException
  - MemberRegisterService
- 암호 변경 관련 클래스
  - MemberNotFoundException
  - ChangePasswordService
- 프로그램 실행
  - MainForAssembler
  - Assembler



의존관계 예제

```
public class Assembler {

        private MemberDao memberDao;
        private MemberRegisterService regSvc;
        private ChangePasswordService pwdSvc;

        public Assembler() {
                memberDao = new MemberDao();
                regSvc = new MemberRegisterService(memberDao);
                pwdSvc = new ChangePasswordService(memberDao);
        }

        public MemberDao getMemberDao() {
                return memberDao;
        }

        public MemberRegisterService getMemberRegisterService() {
                return regSvc;
        }

        public ChangePasswordService getChangePasswordService() {
                return pwdSvc;
        }
}
```

조립기는 객체를 **보관**합니다.

조립기는 객체를 **생성**합니다.

조립기는 객체를 **제공**합니다.

조립기는 객체를 **제공**합니다.

조립기는 객체를 **제공**합니다.

의존관계 예제

```
public class MainForAssembler {

        public static void main(String[] args) {
                Scanner reader = new Scanner(System.in);
                while (true) {
                        System.out.println("명령어를 입력하세요:");
                        String command = reader.nextLine();
                        if (command.equalsIgnoreCase("exit")) {
                                System.out.println("종료합니다.");
                                break;
                        }
                        if (command.startsWith("new ")) {
                                        processNewCommand(command.split(" "));
                                continue;
                        } else if (command.startsWith("change ")) {
                                processChangeCommand(command.split(" "));
                                continue;
                        }
                        printHelp();
                }

        }
```

의존관계 예제

조립기 객체 생성

```
private static Assembler assembler = new Assembler();

private static void processNewCommand(String[] arg) {
        …
        MemberRegisterService regSvc = assembler.getMemberRegisterService();
        …
}

private static void processChangeCommand(String[] arg) {
        …
        ChangePasswordService changePwdSvc = assembler.getChangePasswordService();
        …
}

private static void printHelp() {
        …
}

}
```

조립기가 MemberRegisterService 객체를 제공

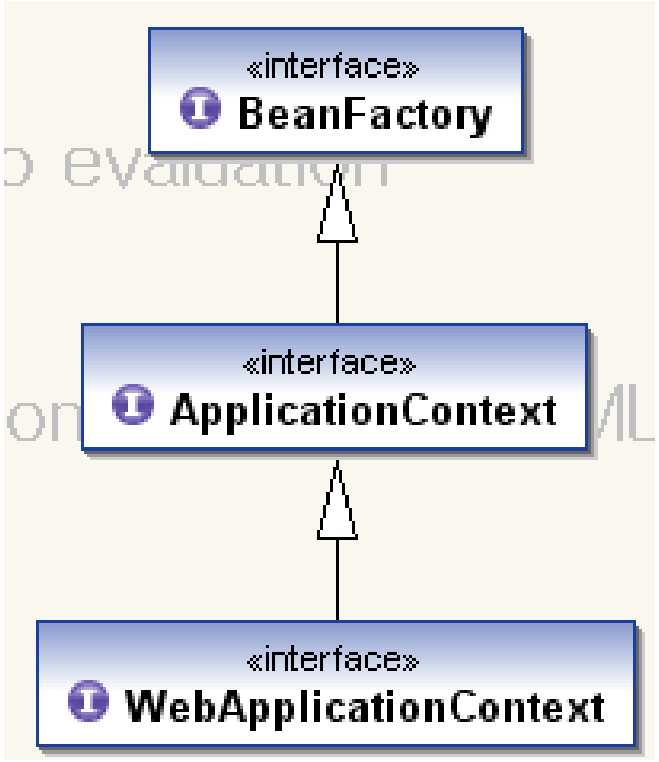
조립기가 ChangePasswordService 객체를 제공

스프링의 DI 설정

- **스프링은 앞서 구현한 조립기와 같은 기능을 제공한다.**
  - Assembler 클래스의 생성자 코드처럼 필요한 **객체를 생성하고 의존을 주입**해 준다.
  - getMemberRegisterService() 메서드처럼 **객체를 제공하는 기능**도 정의되어 있음.
  - Assembler 클래스는 MemberRegisterService나 MemberDao와 같이 특정 클래스 타입의 클래스만 생성할 수 있지만 **스프링은 범용적으로 사용할 수 있는 조립기를 제공**한다.
  - 이러한 조립기를 컨테이너라 한다.

스프링의 DI 설정

- **스프링은 객체를 관리하는 컨테이너를 제공한다.**
  - Spring에선 빈(객체)의 생성과 관계 설정, 사용, 제거 등의 기능을 담당 하는 컨테이너를 제공
  - 스프링은 컨테이너에 객체를 담아두고, 객체가 필요할 때 컨테이너로 부터 객체를 가져와 사용할 수 있도록 하고 있다.
  - BeanFactory 와 ApplicationContext가 컨테이너 역할을 하는 인터페이스이다.



스프링의 DI 설정

- **BeanFactory 인터페이스**
  - **org.springframework.beans.factory.BeanFactory**
  - 빈 객체를 관리하고 각 빈 객체간의 의존 관계를 설정해주는 기능을 하는 가장 단순한 컨테이너.
  - 구현 클래스는 org.springframework.beans.factory.xml.XmlBeanFactory
    - 빈 팩토리를 상속받고 있는 하위 클래스로 **XML과 같은 외부 설정 파일의 내용을 기반으로 객체 생성을 하게 되면 Spring 컨테이너가 생성된다.**
    - XmlBeanFactory 클래스는 **외부 자원으로부터 설정 정보를 읽어와 빈 객체를 생성**.

스프링의 DI 설정

```
import org.springframework.context.support.GenericXmlApplicationContext;
GenericXmlApplicationContext ctx = new
GenericXmlApplicationContext("classpath:applicationContext.xml");
ctx.getBean("xml에 명시된 빈의 id", 이용할 클래스명.class);
```

클래스	설명
org.springframework.core.io.FileSystemResource	파일시스템의 특정 파일로부터 정보를 읽어 온다.
org.springframework.core.io.InputStreamResource	InputStream으로 부터 정보를 읽어 온다.
org.springframework.core.io.ClassPathResource	클래스패스에 있는 자원으로부터 정보를 읽어 온다.
org.springframework.core.io.UrlResource	특정 Url로 부터 정보를 읽어 온다.
org.springframework.web.context.support.ServletContextResource	웹 어플리케이션의 루트 디렉토리를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 익어 온다.
org.springframework.context.support.GenericXmlApplicationContext	XML에서 빈의 의존관계 정보를 이용하는 IoC/DI 작업에는 GenericXmlApplicationContext를 사용

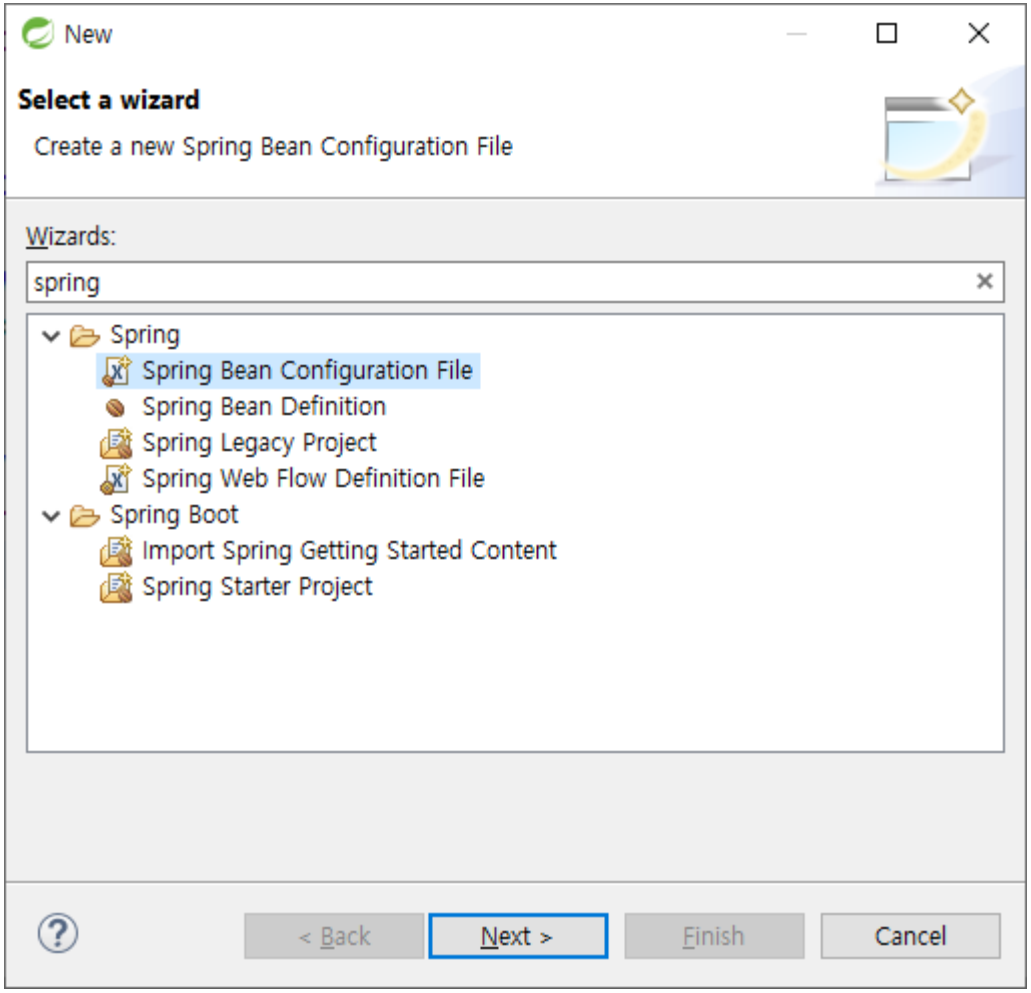
스프링의 DI 설정 : appCtx.xml

- 컨테이너에 저장될 빈 객체와 각 빈 객체간의 연관 관계는 XML 파일을 통해서 설정.
- 스프링은 어노테이션을 이용한 설정도 지원.

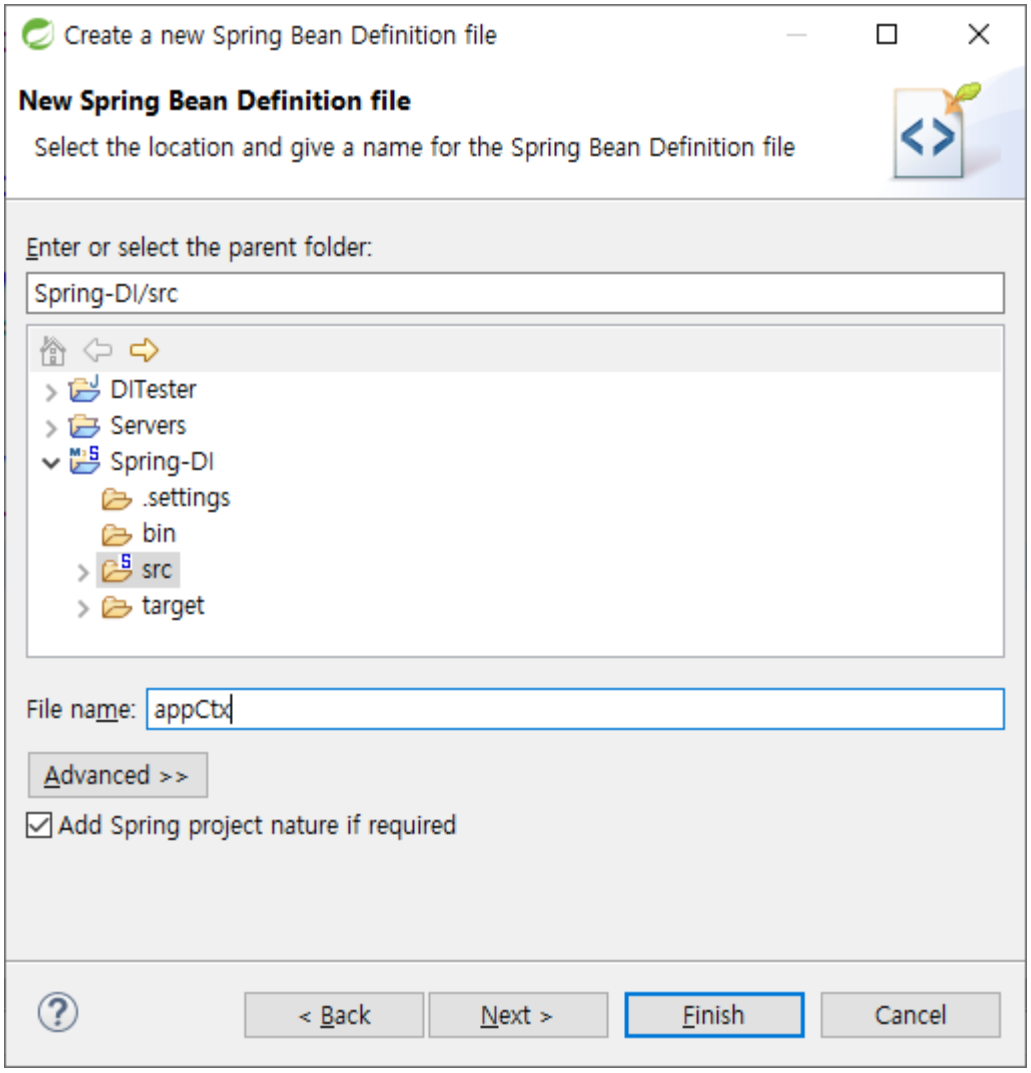
```
<?xml version="1.0" encoding="UTF-8"?>

<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-3.0.xsd">

<bean id="articleDao" class="dao.MySQLArticleDao">
</bean>

</beans>
```

```
<bean name="articleDao" class="dao.MySQLArticleDao">
</bean>
```

스프링의 DI 설정 : appCtx.xml



스프링의 DI 설정 : appCtx.xml



스프링의 DI 설정 : appCtx.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.xsd">

                                                    스프링 컨테이너에 등록해서 사용할 객체 등록
        <bean id="memberDao" class="di.dao.MemberDao">
        </bean>

        <bean id="memberregSvc" class="di.service.MemberRegisterService">
                    <constructor-arg>
                                <ref bean="memberDao"/>
                    </constructor-arg>
        </bean>                                     스프링 컨테이너에 등록해서 사용할 객체 등록

        <bean id="memberPwSvc" class="di.service.ChangePasswordService">
                    <constructor-arg ref="memberDao"/>
        </bean>
                                                    스프링 컨테이너에 등록해서 사용할 객체 등록

</beans>
```

스프링의 DI 설정 : MainForSpring.java

```
public class MainForSpring {

        private static GenericXmlApplicationContext ctx = null;

        public static void main(String[] args) {
                ctx = new GenericXmlApplicationContext("classpath:appCtx.xml");
                …
        }
        private static void processNewCommand(String[] arg) {
                …
        MemberRegisterService regSvc = ctx.getBean("memberregSvc", MemberRegisterService.class);
                …
        }
        private static void processChangeCommand(String[] arg) {
                …
        ChangePasswordService changePwdSvc = ctx.getBean("memberPwSvc", ChangePasswordService.class);
                …
        }

        private static void printHelp() {
                …
        }
}
```

조립기 객체 생성

조립기 객체 생성

조립기 객체 생성

조립기 객체 생성

스프링의 DI 설정 : 생성자 방식

- 의존하는 빈 객체를 컨테이너로부터 생성자의 파라미터를 통해서 전달받는 방식
- 클래스를 초기화 할 때 컨테이너로 부터 의존 관계에 있는 특정 리소스인 빈 객체를 생성자를 통해서 할당 받는 방법

  - **<constructor-arg> 태그를 이용하여 의존하는 객체를 전달.**

```
<bean id="memberDao" class="spring.MemberDao">
</bean>
<bean id="memberRegSvc" class="spring.MemberRegisterService">
        <constructor-arg ref="memberDao" />
</bean>
```

스프링의 DI 설정 : 생성자 방식

```
<bean id="memberregSvc" class="di.service.MemberRegisterService">
        <constructor-arg>
                <ref bean="memberDao"/>
        </constructor-arg>
</bean>
```

```
<bean id="memberPwSvc" class="di.service.ChangePasswordService">
        <constructor-arg ref="memberDao"/>
</bean>
```

스프링의 DI 설정 : 프로퍼티 방식

- 프로퍼티 설정 방식은 setXXXX() 형태의 설정 메서드를 사용해서 필요한 객체와 값을 전달 받는다.
- Set 뒤에는 프러퍼티(변수) 이름의 첫 글자를 대문자로 치환한 이름을 사용한다.

```
// 이 서비스 클래스는 서비스 클래스가 완성 되어 있지 않더라도 코드 완성에 문제가 없다, 정상적으로 컴파일이 된다.
public class ChangePasswordService2 {

        // interface 타입의 참조변수 생성
        private Dao dao;

        // 하위 클래스 타입의 객체를 주입 받을 수 있는 객체
        public void setMemberDao(Dao dao) {
                this.dao = dao;
        }
}
```

```
<bean id="memberregSvc" class="di.service.MemberRegisterService2">
        <property name="dao">
                <ref bean="memberDao"/>
        </property>
</bean>
```

스프링의 DI 설정 : 프로퍼티 방식

```
<bean id="memberregSvc" class="di.service.MemberRegisterService2">
        <property name="dao">
                <ref bean="memberDao"/>
        </property>
</bean>
```

```
<bean id="memberPwSvc" class="di.service.ChangePasswordService2">
        <property name="dao" ref="memberDao"/>
</bean>
```

스프링의 DI 설정 : XML 네임스페이스를 이용한 프로퍼티 설정

```
<?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:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="memberDao" class="di.dao.MemberDao"></bean>

<bean id="memberregSvc" class="di.service.MemberRegisterService"
        p:dao-ref="memberDao"/>

<bean id="memberPwSvc" class="di.service.ChangePasswordService"
        p:dao-ref="memberDao"/>

</beans>
```

의존관계 설정 : 임의 빈 객체 전달

- 식별자를 각지 않는 빈 객체를 생성해서 전달할 수도 있다.
- <constructor-arg> 태그나 <property> 태그에 <bean> 태그를 중첩해서 사용하면 된다.

```
<bean id="memberDao" class="di.dao.MemberDao"></bean>

<bean id="memberregSvc" class="di.service.MemberRegisterService2">
        <property name="dao">
                <bean class="di.dao.MemberDao"/>
        </property>
</bean>

<bean id="memberPwSvc" class="di.service.ChangePasswordService2">
        <property name="dao" ref="memberDao"/>
</bean>


</beans>
```

두 개 이상의 설정파일 사용하기

```
<?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.xsd">

    <import resource="classpath:config1.xml"/>
    <import resource="classpath:config2.xml"/>

</beans>
```

두 개 이상의 설정파일 사용하기

```
public class MainForSpring2 {

        private static GenericXmlApplicationContext ctx = null;

        public static void main(String[] args) {

            String[] xmlConfigPath = {"classpath:config1.xml", "classpath:config2.xml"};
            ctx = new GenericXmlApplicationContext(xmlConfigPath);
```

의존관계 자동 설정

@Autowired 애노테이션을 이용한 의존 자동 주입

@Resource 애노테이션을 이용한 자동 의존 주입

자동 주입과 명시적 의존 주입 간의 관계

의존 관계 자동 설정

- **의존하는 빈 객체의 타입이나 이름을 이용하여 의존객체를 자동으로 설정할 수 있는 기능**
- **autowire속성을 이용**
- **자동 설정과 직접 설정의 혼합도 가능**
- **세 가지 방식**
  - **byName :** 프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정
  - **byType :** 프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정
  - **constructor :** 생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달

의존 관계 자동 설정

- **byName**
  - 프로퍼티 이름과 동일한 이름을 갖는 빈 객체를 프로퍼티 값으로 설정

```
// interface 타입의 참조변수 생성
private Dao dao;

// 하위 클래스 타입의 객체를 주입 받을 수 있는 객체
public void setDao(Dao dao) {
        this.dao = dao;
}
```

```
<bean id="dao" class="di.dao.MemberDao"/>

<bean id="memberregSvc" class="di.service.MemberRegisterService2" autowire="byName"/>

<bean id="memberPwSvc" class="di.service.ChangePasswordService2" autowire="byType"/>
```

빈 객체 범위

- **기본적으로 컨테이너에 한 개의 빈 객체를 생성**
- scope 속성을 이용 범위 설정

범위	설명
singleton	컨테이너에 한 개의 빈 객체만 생성한다.(기본값)
prototype	빈을 요청할 때마다 빈 객체를 생성한다.
request	HTTP 요청마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)
session	HTTP 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)

빈 객체 범위

```
MemberRegisterService2 regSvc1 = ctx.getBean("memberregSvc", MemberRegisterService2.class);
MemberRegisterService2 regSvc2 = ctx.getBean("memberregSvc", MemberRegisterService2.class);

System.out.println("------------------------------------------------");
System.out.println("bean scope 을 prototype으로 설정");
System.out.println("regSvc1 == regSvc2 --> " + (regSvc1 == regSvc2));
System.out.println("------------------------------------------------");

ChangePasswordService2 changePwdSvc1 = ctx.getBean("memberPwSvc", ChangePasswordService2.class);
ChangePasswordService2 changePwdSvc2 = ctx.getBean("memberPwSvc", ChangePasswordService2.class);

System.out.println("bean scope 을 singleton으로 설정");
System.out.println("changePwdSvc1 == changePwdSvc2 --> " + (changePwdSvc1 == changePwdSvc2));
System.out.println("------------------------------------------------");
```

```
…

<bean id="dao" class="di.dao.MemberDao"/>

<bean id="memberregSvc" class="di.service.MemberRegisterService2" autowire="byName"
scope="prototype"/>

<bean id="memberPwSvc" class="di.service.ChangePasswordService2" autowire="byType"
scope="singleton"/>

…
```

빈 객체 범위

```
…
GreetingService bean = (GreetingService)factory.getBean("greeting");
GreetingService bean2 = (GreetingService)factory.getBean("greeting");
System.out.println(bean == bean2); // false
…
```

```
…
<bean id="greeting" class="beanscope.GreetingServiceImpl"  scope="prototype">
        <property name="greeting">
                <value>Hello</value>
        </property>
</bean>
…
```

애노테이션 기반 설정

- **애노테이션이란?**
  - JDK5 버전부터 추가된 것으로 메타데이터를 XML등의 문서에 설정하는 것이 아니라 소스 코드에 "@애노테이션"의 형태로 표현하며 **클래스, 필드, 메소드의 선언부에 적용 할 수 있는 특정 기능이 부여된 표현법**

- **애노테이션 사용 이유**
  - 프레임워크들이 활성화 되고 **애플리케이션 규모가 커질수록 XML 환경 설정은 복잡**해지는데, 이러한 어려움을 개선시키기 위하여 자바 파일에 애노테이션을 적용해서 코드를 작성함으로써 개발자가 설정 파일에 작업하게 될 때 발생시키는 오류의 발생 빈도를 낮춰주기도 함

애노테이션 기반 설정

- **애노테이션 문법**
  - **@애노테이션"[애노테이션]"의 형태로 표현**하며 클래스, 필드, 메소드의 선언부에 적용 할 수 있고 특정 기능이 부여된 표현법입니다

```
 9 @Component
10 public class PersonServiceImpl implements PersonService{
11     private String message;
12
13     @Resource(name="person1")
14     private Person person1;
15
16     @Resource(name="person2")
17     private Person person2;
```

애노테이션 기반 설정

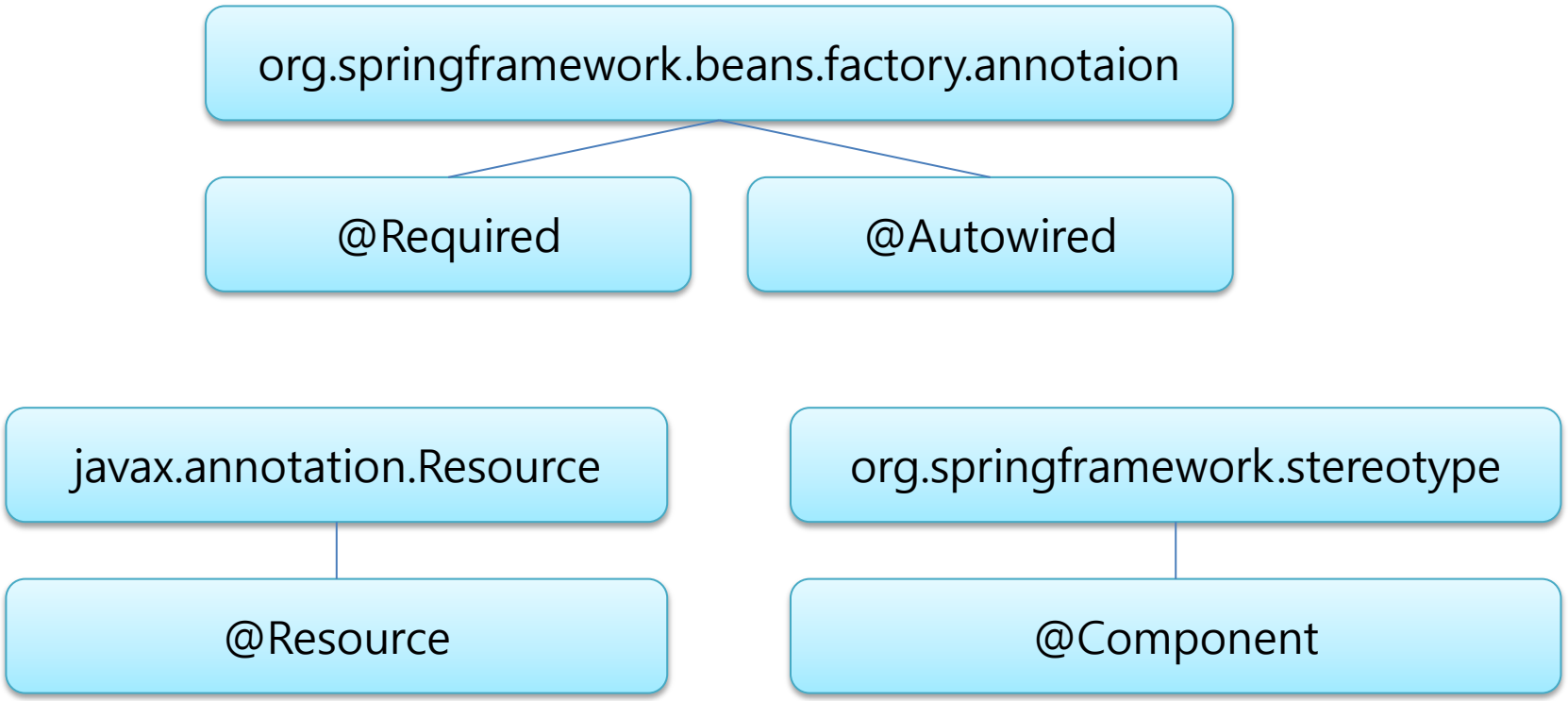
- **주석과 애노테이션의 차이점**
  - //과, @ 은 개발자가 보기에는 똑같이 실행되지 않는 코드 같지만 실제로 //은 컴파일러가 실행은 안 해도 **@은 컴파일러가 실행되기 전에 애노테이션으로 설정한 내용대로 코드가 작성되었는지 확인하기 위해 실행을 함**
  - "@Override" 애노테이션이 선언된 메소드를 Eclipse 툴에서 재정의 할 경우 재정의 문법에 위배된 코드로 개발이 되면 문법 오류 표현을 해 주기 때문에 빠른 시간에 문제점에 대한 해결.
  - **개발자가 환경 설정에서의 실수를 했을 경우에도 수정해주는 역할을 하기도 하고, 메타 정보를 선언하게 되는 XML 설정 파일이 점점 복잡해지지 않도록 자바 코드 안에서 설정하므로 개발자의 도움말이 되는 역할을 함**
  - Java 에서 이미 정의되어 있는 애노테이션
    - @Override - 메소드가 오버라이드 됐는지 검증.부모 클래스 또는 구현해야할 인터페이스에서 해당 메소드를 찾을 수 없다면 컴파일 오류.
    - @Deprecated - 메소드를 사용하지 말도록 유도. 만약 사용한다면 컴파일 경고.
    - @SuppressWarnings - 컴파일 경고를 무시.
    - @SafeVarargs - 제너릭 같은 가변인자 매개변수를 사용할 때 경고를 무시.(자바7 이상)
    - @FunctionalInterface - 람다 함수등을 위한 인터페이스를 지정. 메소드가 없거나 두개 이상 되면 컴파일 오류.(자바 8이상)

애노테이션 기반 설정

- Spring2.5 버전 부터는 애노테이션을 이용하여 빈과 관련된 정보를 설정할 수 있게 되었으며, 복잡한 XML 문서 생성과 관리에 따른 수고를 덜어주어 개발 속도를 향상 시킬 수 있음

애노테이션	설 명
@Required	setter주입방식을 이용한 애노테이션, 필수 프로퍼티를 명시할 때 사용
@Autowired	타입을 기준으로(byType) 빈을 찾아 주입하는 애노테이션
@Resource	이름을 기준으로(byname) 빈을 찾아 주입하는 애노테이션
@Component	빈 스캐닝 기능을 이용한 애노테이션

애노테이션 기반 설정


org.springframework.beans.factory.annotaion
@Required
@Autowired
javax.annotation.Resource
@Resource
org.springframework.stereotype
@Component

@Autowired 애노테이션을 이용한 의존 자동 주입

- 자동 주입 대상에 @Autowire 애노테이션 사용
- XML 설정에 <context:annotation-config /> 설정 추가

```
public class MemberRegisterService3 {

        @Autowired
        private Dao dao;

        public void regist(RegisterRequest req) throws Exception {
                …
        }
}
```

@Autowired 애노테이션을 이용한 의존 자동 주입

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        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.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context.xsd">

        <context:annotation-config />

        <bean id="dao" class="di.dao.MemberDao">
        </bean>

        <bean id="memberregSvc" class="di.service.MemberRegisterService">
        </bean>

 </beans>
```

@Qualifier 애노테이션을 이용한 의존 객체 선택

- **@Qualifier 애노테이션은 사용할 의존 객체를 선택 할 수 있도록 해준다.**
- **@Qualifier 애노테이션을 사용하려면 아래 두 가지 설정이 필요**
  - 설정에서 빈의 한정자 설정
  - @Autowired 애노테이션이 적용된 주입 대상에 @Qualifier 애노테이션을 설정
    이때 @Qualifier 애노테이션의 값으로 앞서 설정한 한정자 사용

@Qualifier 애노테이션을 이용한 의존 객체 선택

```
<bean id="printer1" class="spring.MemberPrinter">
        <qualifier value="sysout" />
</bean>

<bean id="printer2" class="spring.MemberPrinter" />

<bean id="infoPrinter" class="spring.MemberInfoPrinter">
</bean>
```

```
@Autowired
@Qualifier("sysout")
public void setPrinter(MemberPrinter printer) {
        System.out.println("setPrinter: " + printer);
        this.printer = printer;
}
```

@Autowired 의 필수 여부 지정

- **@Autowired(required=false)**

```
public class MemberRegisterService {

        @Autowired(required=false)
        private MemberDao memberDao;

        public MemberRegisterService() {
        }

        ....
```

@Autowired 애노테이션의 적용 순서

1. **타입이 같은 빈 객체를 검색한다.**

   한 개면 그 객체를 사용한다.

   @Qualifier가 명시되어 있을 경우,

   @Qualifier와 같은 값을 갖는 빈 객체이어야 한다.

2. 타입이 같은 빈 객체가 두 개 이상 존재하면,

   @Qualifer로 지정한 빈 객체를 찾는다.

   존재하면 그 객체를 사용한다.

3. 타입 같은 빈 객체가 두 개 이상 존재하고,

   @Qualifier가 없을 경우,

   이름이 같은 빈 객체를 찾는다.

   존재하면, 그 객체를 사용한다.

@Resource 애노테이션을 이용한 자동 의존 주입

- **@Resource 애노테이션은 빈의 이름을 이용해서 주입할 객체를 검색**
  - **@Resource 애노테이션을 사용하려면 두 가지 설정이 필요**
    - **자동 주입 대상에 @Resource 애노테이션 사용**
    - **XML설정에 <context:annotation-config /> 설정 추가**

```
public class MemberRegisterService {
        @Resource(name="memberDao")
        private MemberDao memberDao;

        public MemberRegisterService() {
        }
        ....
```

@Resource 애노테이션의 적용순서

1. **name 속성에 지정한 빈 객체를 찾는다.**
   존재하면 해당 객체를 주입할 객체로 사용한다
2. name 속성이 없을 경우,
   동일한 타입을 갖는 빈 객체를 찾는다.
   존재하면 해당 객체를 주입할 객체로 사용한다.
3. name 속성이 없고,
   동일한 타입을 갖는 빈 객체가 두 개 이상일 경우,
   같은 이름을 가진 빈 객체를 찾는다.
   존재하면 해당 객체를 주입할 객체로 사용한다.
4. name 속성이 없고,
   동일한 타입을 갖는 빈 객체가 두 개 이상이고
   같은 이름을 가진 빈 객체가 없을 경우,
   @Qualifer를 이용해서 주입할 빈 객체를 찾는다.

자동 주입과 명시적 의존 주입 간의 관계

- 자동 주입과 명시적 의존 주입 설정이 함께 사용되는 경우 명시적인 의존 주입 설정이 우선한다.

자바코드 설정 기초

자바 코드 설정과 자동 주입

두 개 이상 클래스를 이용한 자동 설정

자바 코드 설정과 XML 설정의 혼합

자바코드 설정 기초

- XML 설정 없이도 자바 코드를 이용해서 생성할 빈 객체와 각 빈간의 의존 관계 설정
- 하나의 클래스 안에 여러 개의 빈을 정의 할 수도 있고, 클래스 자체가 자동인식 빈의 대상이 되기 때문에 XML에 명시적으로 등록하지 않아도 됨
- **사용되는 어노테이션**
  - **@Configuration**
    - **빈 설정 메타 정보를 담고 있는 자바 코드에 선언**
  - **@Bean**
    - **클래스 내의 메소드를 정의 할 수 있음**
    - **자바 설정인 경우에도 싱글톤!!**

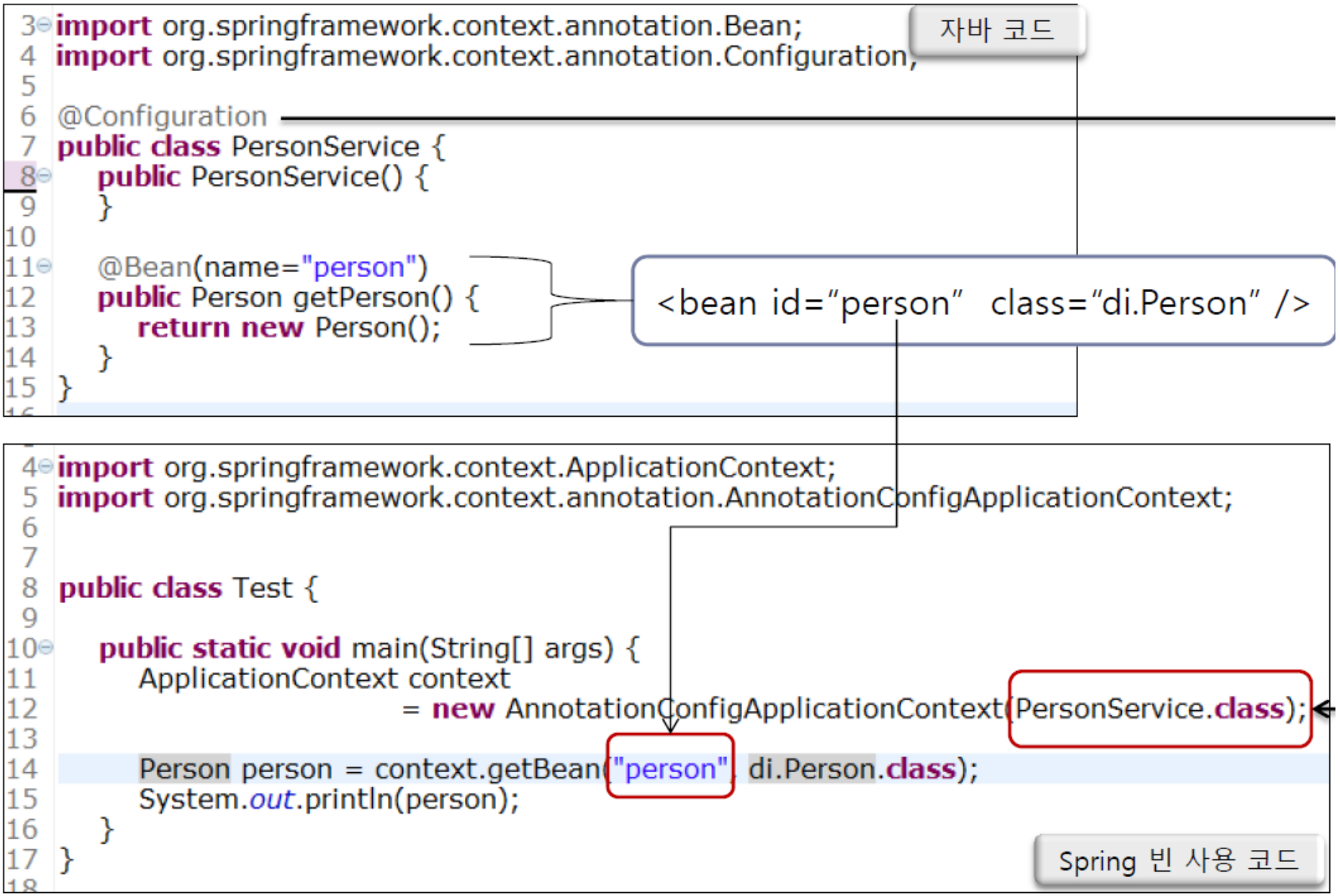
자바코드 설정 기초

- **Spring 컨테이너에 새로운 빈 객체 제공**

@Configuration	**@Bean**
빈 설정 메타 정보를 담고 있는 클래스가 됨	1. 새로운 빈 객체를 제공할 때 사용 2. 적용된 메서드의 이름을 빈의 식별 값으로 사용 3. name 속성을 사용하여 새로운 빈 이름 적용 가능 4. @Scope(value="prototype") 으로 범위 설정 가능

```
ApplicationContext context = new AnnotationConfigApplicationContext(클래스명.class)
```

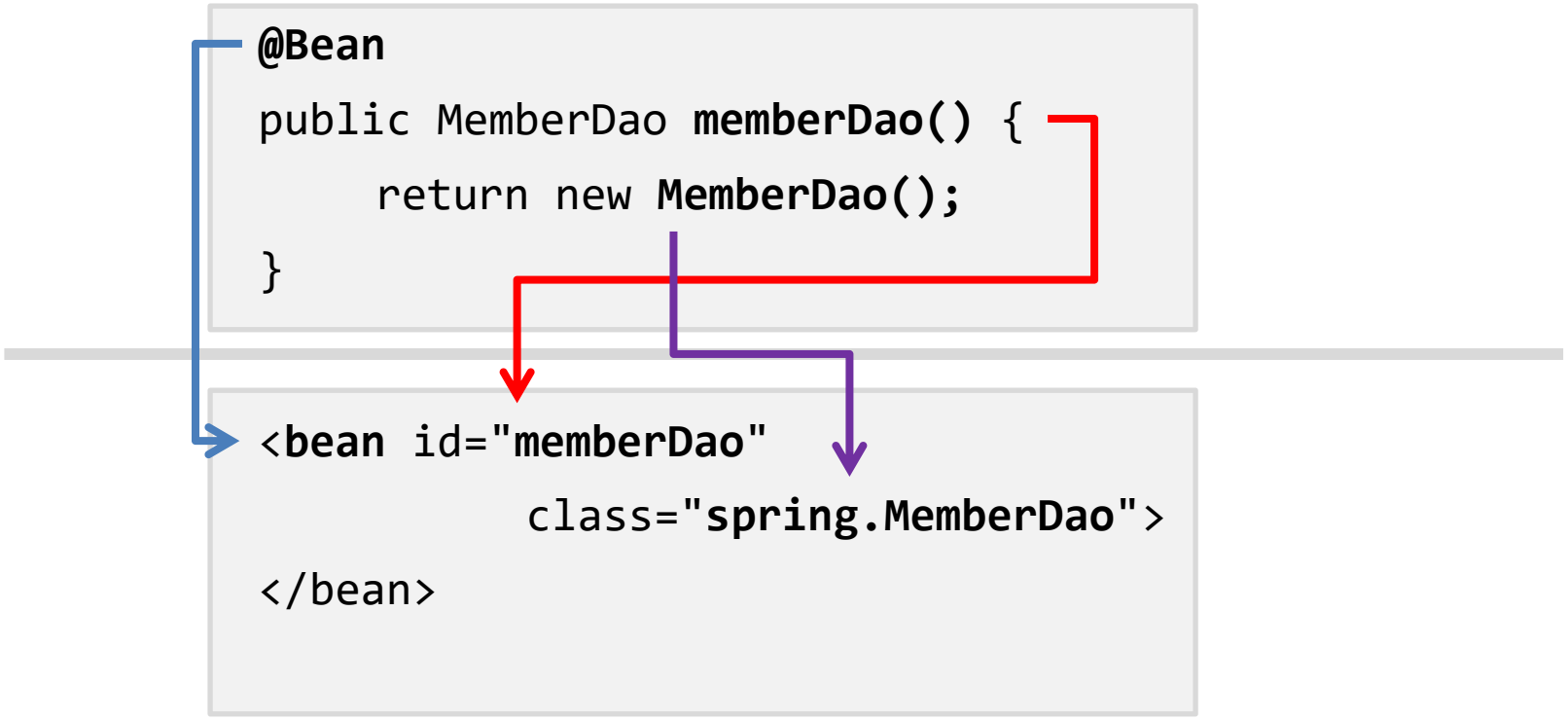
자바코드 설정 기초




```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class PersonService {
    public PersonService() {
    }

    @Bean(name="person")
    public Person getPerson() {
        return new Person();
    }
}
```

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;


public class Test {

    public static void main(String[] args) {
        ApplicationContext context
                        = new AnnotationConfigApplicationContext(PersonService.class);

        Person person = context.getBean("person", di.Person.class);
        System.out.println(person);
    }
}
```

자바코드 설정 기초


@Bean
public MemberDao memberDao() {
return new MemberDao();
}
<bean id="memberDao"
class="spring.MemberDao">
</bean>

자바코드 설정 기초

- **의존관계 주입**
  - 다른 빈 객체에 대한 의존 객체를 주입할 때에는 주입할 빈 객체에 해당하는 메서드를 호출해서 해당 빈 객체를 구함.

```
@Bean
public MemberDao memberDao() {
        return new MemberDao();
}
@Bean
public MemberRegisterService memberRegSvc() {
        // 다른 빈 객체를 의존으로 주입할 경우
        // 해당 빈 객체의 메서드를 호출해서 의존 객체를 구한다.
        return new MemberRegisterService(memberDao());
}
```

자바코드 설정 기초

- **AnnotationConfigApplicationContext를 이용한 @Configuration 클래스 사용**

```
AbstractApplicationContext context =
                            new AnnotationConfigApplicationContext(JavaConfig.class);

…

MemberRegisterService service =
                    context.getBean("memberRegSvc", MemberRegisterService.class);
```

자바코드 설정 기초

```
package di.config;

@Configuration
public class JavaConfig {

    @Bean
    public MemberDao memberDao() {
        return new MemberDao();
    }

    @Bean
    public MemberRegisterService memberRegSvc() {
        return new MemberRegisterService(memberDao());
    }

    @Bean
    public ChangePasswordService memberPwSvc() {
        return new ChangePasswordService(memberDao());
    }

}
```

자바코드 설정 기초

```
public class MainForSpring6 {

	public static void main(String[] args) throws AlreadyExistingMemberException {

		ApplicationContext ctx = new AnnotationConfigApplicationContext(JavaConfig.class);

		RegisterRequest regReq = new RegisterRequest();
		regReq.setEmail("ryuyj@nate.com");
		regReq.setName("유영진");
		regReq.setPassword("1234");
		regReq.setConfirmPassword("1234");

		MemberRegisterService regSvc = ctx.getBean("memberRegSvc", MemberRegisterService.class);
		regSvc.regist(regReq);

		MemberDao dao = ctx.getBean("memberDao", MemberDao.class);
		Member member  = dao.selectByEmail("ryuyj@nate.com");

		System.out.println(member.getName());
	}

}
```

@Scope 어노테이션을 이용한 범위 설정

- 범위 설정

```
@Bean
@Scope(value = "prototype")
public MemberDao memberDao() {
          return new MemberDao();
}
```

@Qualifier 한정자 지정

- 두 개 이상의 빈 객체가 매칭이 되는 경우 @Qualifier 를 써서 연관될 수 있는 빈을 한정 지을 수 있다.

```
@Bean
@Qualifier("main")
public Recorder recorder() {
          return new Recorder();
}
```

자바 코드 설정과 자동 주입

- 자바설정을 사용하는 경우에는 별도의 설정을 하지 않아도 애노테이션을 이용한 자동 주입 기능이 활성화 된다.
- 자바 설정의 경우 **자동 주입은 생성자 타입의 자동 주입은 적용되지 않고,**
  필드나 메서드 형태로만 가능하다.

두 개 이상 클래스를 이용한 자동 설정

```
@Configuration
public class ConfigPart1 {

        @Bean
        public MemberDao memberDao() {
                return new MemberDao();
        }

        @Bean
        public MemberRegisterService memberRegSvc() {
                return new MemberRegisterService(memberDao());
        }

}
```

두 개 이상 클래스를 이용한 자동 설정

```
@Configuration
public class ConfigPart2 {


          @Autowired
          private MemberDao memberDao;


        @Bean
        public ChangePasswordService memberPwSvc() {
                  return new ChangePasswordService(memberDao());
        }
}
```

두 개 이상 클래스를 이용한 자동 설정

```
public class MainTwoConfs {

        public static void main(String[] args) {

                ApplicationContext ctx =
        new AnnotationConfigApplicationContext(ConfigPart1.class, ConfigPart2.class);

                MemberRegisterService regSvc =
                        ctx.getBean("memberRegSvc", MemberRegisterService.class);

                MemberInfoPrinter infoPrinter =
                        ctx.getBean("infoPrinter", MemberInfoPrinter.class);

                RegisterRequest regReq = new RegisterRequest();
                regReq.setEmail("ryuyj@nate.com");
                regReq.setName("유영진");
                regReq.setPassword("1234");
                regReq.setConfirmPassword("1234");
                regSvc.regist(regReq);

        …
        }
}
```

두 개 이상 클래스를 이용한 자동 설정 : @Import

```
@Configuration
@Import(ConfigPartSub.class)
public class ConfigPartMain {

        @Bean
        public MemberDao memberDao() {
                return new MemberDao();
        }

        @Bean
        public MemberRegisterService memberRegSvc() {
                return new MemberRegisterService(memberDao());
        }

}
```

자바 코드 설정과 XML 설정의 혼합

- **자바 설정에서 XML 설정 임포트하기**

```
@Configuration
@ImportResource("classpath:sub-conf.xml")
public class JavaMainConf {

        @Autowired
        private MemberDao memberDao;

        @Bean
        public MemberRegisterService memberRegSvc() {
                return new MemberRegisterService(memberDao);
        }
}
```

자바 코드 설정과 XML 설정의 혼합

```
<?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.xsd">

         <bean id="memberDao" class="spring.MemberDao" />

 </beans>
```

자바 코드 설정과 XML 설정의 혼합

- **XML 설정에서 자바 설정 임포트하기**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:context="http://www.springframework.org/schema/context"
          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.xsd
                    http://www.springframework.org/schema/context
                    http://www.springframework.org/schema/context/spring-context.xsd">

          <context:annotation-config />

          <bean class="config.JavaSubConf" />

          <bean id="memberDao" class="spring.MemberDao" />
          <bean id="infoPrinter" class="spring.MemberInfoPrinter">
                    <property name="memberDao" ref="memberDao" />
                    <property name="printer" ref="memberPrinter" />
          </bean>
 </beans>
```

자바 코드 설정과 XML 설정의 혼합

```
@Configuration
public class JavaSubConf {

	@Autowired
	private MemberDao memberDao;

	@Bean
	public MemberRegisterService memberRegSvc() {
		return new MemberRegisterService(memberDao);
	}

}
```