

Spring Framework

- 스프링 DI - 1

■ 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 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
- 컨테이너의 역할
 - A 객체가 필요로 하는 의존 관계에 있는 다른 객체 B 객체를 직접 생성하여 A 객체로 주입(설정)해주는 역할을 담당
- Spring은 객체를 생성하고 객체간의 의존 관계를 자동 설정해주는 조립기 기능을 보유한 컨테이너가 자동으로 제공(주입)
 - 조립기(Assembler) : 객체를 생성하고 객체간의 의존 관계를 자동 설정해 줌

■ 의존

- 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

■ 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

```
MemberDao memberDao = new MemberDao();
```

```
// 의존관계를 생성자를 통해서 주입
```

```
MemberRegisterService svc = new MemberRegisterService(memberDao)
```


■ 의존관계 예제

- 회원 데이터 관련 클래스
 - Member
 - IdPasswordNotMatchingException
 - MemberDao
- 회원 가입 처리 관련 클래스
 - AlreadyExistingMemberException
 - MemberRegisterService
- 암호 변경 관련 클래스
 - MemberNotFoundException
 - ChangePasswordService

```
public class Member {
    private Long id;
    private String email;
    private String password;
    private String name;
    private Date registerDate;

    public Member(String email, String password, String name, Date registerDate) {
        this.email = email;
        this.password = password;
        this.name = name;
        this.registerDate = registerDate;
    }

    void setId(Long id) {this.id = id;}
    public Long getId() {return id;}
    public String getEmail() {return email;}
    public String getPassword() {return password;}
    public String getName() {return name;}
    public Date getRegisterDate() {return registerDate;}
    public void changePassword(String oldPassword, String newPassword) {
        if (!password.equals(oldPassword))
            throw new IdPasswordNotMatchingException();
        this.password = newPassword;
    }
}
```

```
public class IdPasswordNotMatchingException extends RuntimeException {
```

```
}
```

```
public class MemberDao {  
  
    private static long nextId = 0;  
    private Map<String, Member> map = new HashMap<>();  
  
    public Member selectByEmail(String email) {  
        return map.get(email);  
    }  
  
    public void insert(Member member) {  
        member.setId(++nextId);  
        map.put(member.getEmail(), member);  
    }  
  
    public void update(Member member) {  
        map.put(member.getEmail(), member);  
    }  
  
    public Collection<Member> selectAll() {  
        return map.values();  
    }  
  
}
```

```
public class AlreadyExistingMemberException extends RuntimeException {  
  
    public AlreadyExistingMemberException(String message) {  
        super(message);  
    }  
  
}
```

```
public class RegisterRequest {  
    private String email;  
    private String password;  
    private String confirmPassword;  
    private String name;  
  
    public String getEmail() { return email; }  
    public void setEmail(String email) { this.email = email; }  
    public String getPassword() { return password; }  
    public void setPassword(String password) { this.password = password; }  
    public String getConfirmPassword() { return confirmPassword; }  
    public void setConfirmPassword(String confirmPassword) {  
        this.confirmPassword = confirmPassword;  
    }  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
    public boolean isPasswordEqualToConfirmPassword() {  
        return password.equals(confirmPassword);  
    }  
}
```

```
public class ChangePasswordService {

    private MemberDao memberDao;

    public ChangePasswordService(MemberDao memberDao) {
        this.memberDao = memberDao;
    }

    public void changePassword(String email, String oldPwd, String newPwd) {
        Member member = memberDao.selectByEmail(email);
        if (member == null)
            throw new MemberNotFoundException();

        member.changePassword(oldPwd, newPwd);

        memberDao.update(member);
    }
}
```

```
public class MemberRegisterService {

    private MemberDao memberDao;

    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);
    }

}
```



```
public class MemberNotFoundException extends RuntimeException {  
  
}
```

■ 객체조립기

```
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;  
    }  
}
```

■ 객체조립기

```
public class MainForAssembler {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        while (true) {
            System.out.println("명령어를 입력하세요:");
            String command = reader.readLine();
            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();
        }
    }
}
```

■ 객체조립기

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

```
private static void processNewCommand(String[] arg) {
```

```
    if (arg.length != 5) {
```

```
        printHelp();
```

```
        return;
```

```
    }
```

```
    MemberRegisterService regSvc = assembler.getMemberRegisterService();
```

```
    RegisterRequest req = new RegisterRequest();
```

```
    req.setEmail(arg[1]);
```

```
    req.setName(arg[2]);
```

```
    req.setPassword(arg[3]);
```

```
    req.setConfirmPassword(arg[4]);
```

```
    if (!req.isPasswordEqualToConfirmPassword()) {
```

```
        System.out.println("암호와 확인이 일치하지 않습니다.\n");
```

```
        return;
```

```
    }
```

```
    try {
```

```
        regSvc.regist(req);
```

```
        System.out.println("등록했습니다.\n");
```

```
    } catch (AlreadyExistingMemberException e) {
```

```
        System.out.println("이미 존재하는 이메일입니다.\n");
```

```
    }
```

```
}
```

■ 객체조립기

```
private static void processChangeCommand(String[] arg) {
    if (arg.length != 4) {
        printHelp();
        return;
    }
    ChangePasswordService changePwdSvc = assembler.getChangePasswordService();
    try {
        changePwdSvc.changePassword(arg[1], arg[2], arg[3]);
        System.out.println("암호를 변경했습니다.\n");
    } catch (MemberNotFoundException e) {
        System.out.println("존재하지 않는 이메일입니다.\n");
    } catch (IdPasswordNotMatchingException e) {
        System.out.println("이메일과 암호가 일치하지 않습니다.\n");
    }
}

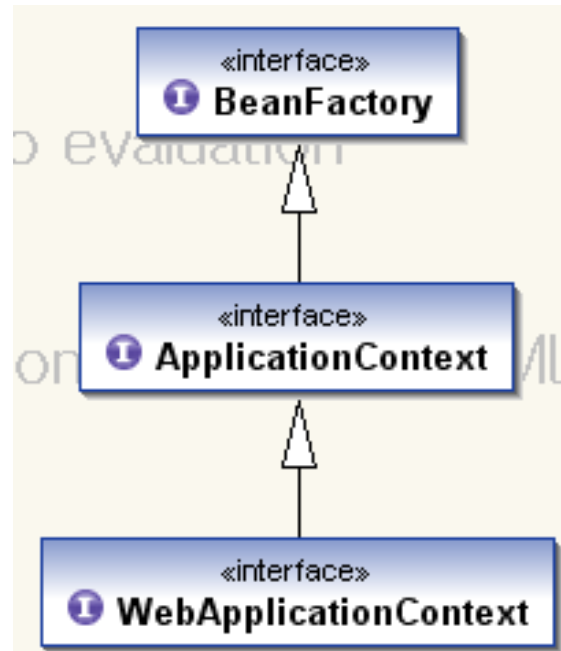
private static void printHelp() {
    System.out.println();
    System.out.println("잘못된 명령입니다. 아래 명령어 사용법을 확인하세요.");
    System.out.println("명령어 사용법:");
    System.out.println("new 이메일 이름 암호 암호확인");
    System.out.println("change 이메일 현재비번 변경비번");
    System.out.println();
}
}
```

■ 스프링의 DI 설정

- **스프링은 앞서 구현한 조립기와 같은 기능을 제공한다.**
- `Assembler` 클래스의 생성자 코드처럼 필요한 객체를 생성하고 의존을 주입해 준다.
- `getMemberRegisterService()` 메서드처럼 **객체를 제공하는 기능도 정의**되어 있음.
- `Assembler` 클래스는 `MemberRegisterService` 나 `MemberDao` 와 같이 특정 클래스 타입의 클래스만 생성할 수 있지만 **스프링은 범용적으로 사용할 수 있는 조립기를 제공한다.**

■ 스프링의 DI 설정

- 스프링은 객체를 관리하는 컨테이너를 제공한다.
 - Spring에선 빈의 생성과 관계 설정, 사용, 제거 등의 기능을 담당 하는 컨테이너를 의미
- 스프링은 컨테이너에 객체를 담아두고, 객체가 필요할 때 컨테이너로 부터 객체를 가져와 사용할 수 있도록 하고 있다.
- BeanFactory 와 ApplicationContext가 컨테이너 역할을 하는 인터페이스가 된다.



■ 스프링의 DI 설정

- BeanFactory 인터페이스

- `org.springframework.beans.factory.BeanFactory`
- 빈 객체를 관리하고 각 빈 객체간의 의존 관계를 설정해주는 기능을 제공해 주는 가장 단순한 컨테이너.
- 빈 팩토리를 상속받고 있는 하위 클래스들로 XML과 같은 외부 설정 파일의 내용을 기반으로 객체 생성을 하게 되면 Spring 컨테이너가 생성되는 것
- 구현 클래스는 `org.springframework.beans.factory.xml.XmlBeanFactory`
- `XmlBeanFactory` 클래스는 외부 자원으로부터 설정 정보를 읽어와 빈 객체를 생성.
- `org.springframework.core.io.Resource` 인터페이스를 사용해서 다양한 종류의 자원을 동일한 방식으로 표현하여 `XmlBeanFactory` 에 정보 전달

■ 스프링의 DI 설정

```
import org.springframework.context.support.GenericXmlApplicationContext;

GenericXmlApplicationContext ctx =
    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 자원으로 부터 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 자원으로 부터 정보를 읽어 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="memberDao" class="dao.MemberDao">  
</bean>
```

```
</beans>
```

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

■ 스프링의 DI 설정 : appCtx.xml

- 생성자가 아닌 static 메서드를 이용해서 객체를 구해야 하는 경우.

싱글톤 패턴이 적용된 클래스의 경우

Spring 컨테이너

호출 → 싱글톤

메서드 → 싱글톤

→ prototype -

→ 요청 시 생성

```
<bean id="memberDao" class="dao.MemberDao"  
  factory-method="getInstance" />
```

메서드 이름

■ 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">
```

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

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

```
  <bean id="changePwdSvc" class="dao.ChangePasswordService">  
    <constructor-arg ref="memberDao" />  
  </bean>
```

```
</beans>
```

■ 스프링의 DI 설정 : MainForSpring.java

```
package main;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

import spring.AlreadyExistingMemberException;
import spring.ChangePasswordService;
import spring.IdPasswordNotMatchingException;
import spring.MemberNotFoundException;
import spring.MemberRegisterService;
import spring.RegisterRequest;

public class MainForSpring {
```

■ 스프링의 DI 설정 : MainForSpring.java

```
private static ApplicationContext ctx = null;
public static void main(String[] args) throws IOException {
    ctx = new GenericXmlApplicationContext("classpath:appCtx.xml");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    while (true) {
        System.out.println("명령어를 입력하세요:");
        String command = reader.readLine();
        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();
    }
}
```

■ 스프링의 DI 설정 : MainForSpring.java

```
private static void processNewCommand(String[] arg) {  
    if (arg.length != 5) {  
        printHelp();  
        return;  
    }  
}
```

MemberRegisterService regSvc =

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

```
RegisterRequest req = new RegisterRequest();  
req.setEmail(arg[1]); req.setName(arg[2]);  
req.setPassword(arg[3]); req.setConfirmPassword(arg[4]);  
if (!req.isPasswordEqualToConfirmPassword()) {  
    System.out.println("암호와 확인이 일치하지 않습니다.\n");  
    return;  
}  
try {  
    regSvc.regist(req);  
    System.out.println("등록했습니다.\n");  
} catch (AlreadyExistingMemberException e) {  
    System.out.println("이미 존재하는 이메일입니다.\n");  
}  
}
```

■ 스프링의 DI 설정 : MainForSpring.java

```
private static void processChangeCommand(String[] arg) {  
    if (arg.length != 4) {  
        printHelp();  
        return;  
    }  
}
```

```
ChangePasswordService changePwdSvc =  
    ctx.getBean("changePwdSvc", ChangePasswordService.class);
```

```
    try {  
        changePwdSvc.changePassword(arg[1], arg[2], arg[3]);  
        System.out.println("암호를 변경했습니다.\n");  
    } catch (MemberNotFoundException e) {  
        System.out.println("존재하지 않는 이메일입니다.\n");  
    } catch (IdPasswordNotMatchingException e) {  
        System.out.println("이메일과 암호가 일치하지 않습니다.\n");  
    }  
}
```


■ 스프링의 DI 설정 : MainForSpring.java

```
private static void printHelp() {  
    System.out.println();  
    System.out.println("잘못된 명령입니다. 아래 명령어 사용법을 확인하세요.");  
    System.out.println("명령어 사용법:");  
    System.out.println("new 이메일 이름 암호 암호확인");  
    System.out.println("change 이메일 현재비번 변경비번");  
    System.out.println();  
}  
}
```

■ 스프링의 DI 설정 : 생성자 방식

- 의존하는 빈 객체를 컨테이너로부터 생성자의 파라미터를 통해서 전달받는 방식
- 클래스를 초기화 할 때 컨테이너로부터 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 name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <constructor-arg ref = "articleDao" />  
</bean>
```

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <constructor-arg>  
        <value>10</value>  
    </constructor-arg>  
</bean>
```

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <constructor-arg value = "10" />  
</bean>
```

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <constructor-arg value = "10" />  
    <constructor-arg ref = "sender" />  
</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.

```
public class WriteArticleServiceImpl implements WriteArticleService {  
  
    private ArticleDao articleDao;  
  
    public void setArticleDao (ArticleDao articleDao) {  
        this.articleDao = articleDao;  
    }  
  
    ...  
}
```

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <property name="articleDao">  
        <ref bean="OracleArticleDao">  
    </property >  
</bean>
```

■ 스프링의 DI 설정 : 프로퍼티 방식

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <property name="retryCount">  
        <value>3</value>  
    </property >  
</bean>
```

```
<bean name= "writeArticleService" class= "service.WriteArticleServiceImpl">  
    <property name="periodTime" value="10" />  
    <property name="sender" ref="smsSender" />  
</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:p="http://www.springframework.org/schema/p"
xmlns:aop="http://www.springframework.org/schema/aop"
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-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd" >

<bean id="regService" class="service.MemberRegService"
    p:dao-ref="memberDao" />

<bean id="memberDao" class="dao.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="monitor" class="service.MemberRegService">  
  <property name="dao">  
    <!-- 임의 빈 객체 -->  
    <bean class="dao.MemberDao"/></bean>  
  </property>  
</bean>
```

■ 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:conf2.xml" />
```

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

```
<bean id="memberPrinter" class="spring.MemberPrinter">
</bean>
```

```
</beans>
```