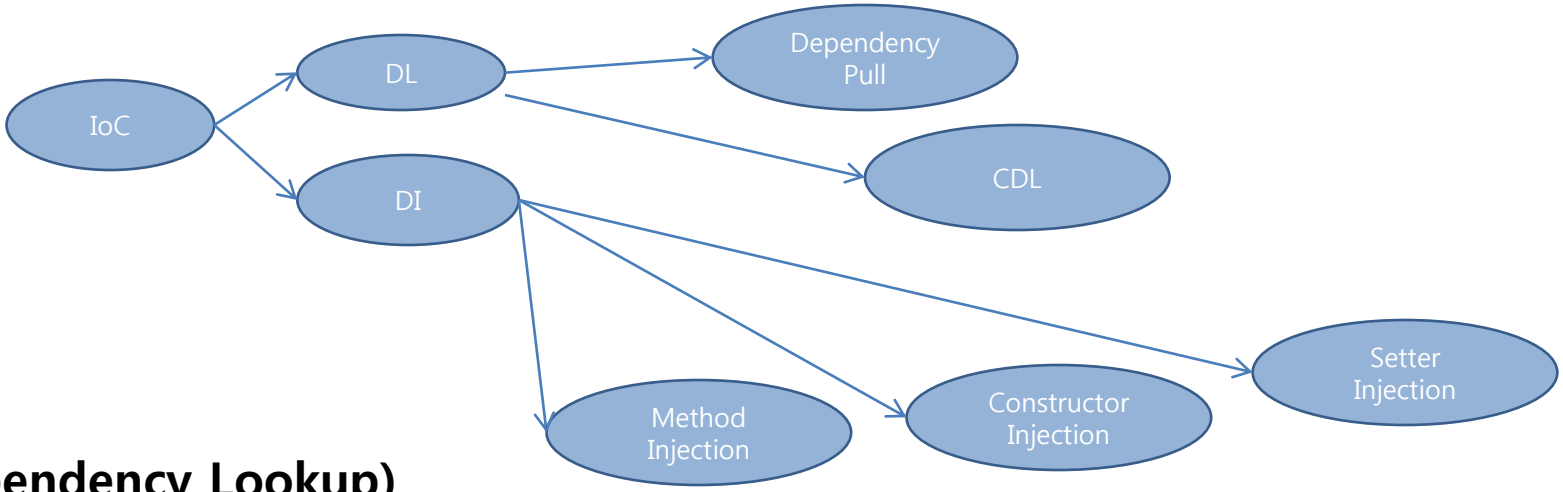
Spring IoC(DI, DL)

오라클자바커뮤니티

(ojc.asia, ojcedu.com)

IoC 컨테이너 분류체계



- **DL(Dependency Lookup)**

  모든 IoC 컨테이너는 각 컨테이너에서 관리해야 하는 객체들을 관리하기 위한 별도의 저장소를 가진다.Bean에 접근하기 위하여 컨테이너에서 제공하는 API를 이용하여 사용하고자 하는 Bean을 Lookup 하는 것으로 컨테이너 API와 의존관계를 많이 가지면 가질수록 어플리케이션 컨테이너에 종속되는 단점이 있다.

- **DI(Dependency Injection)**

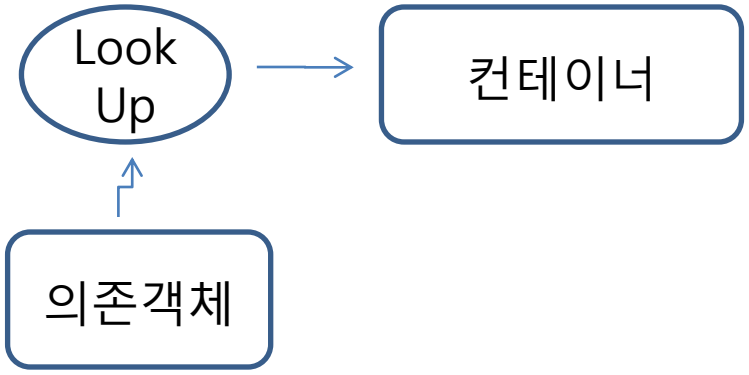
  각 계층 사이, 각 class 사이에 필요로 하는 의존관계가 있다면 이를 스프링 컨테이너가 자동 적으로 연결시켜 주는 것으로 각 class 사이의 의존관계를 Bean 설정 정보 또는 어노테이션을 바탕으로 컨테이너가 자동적으로 연결해 주는 것이다.

의존성 풀(Dependency Pull)

IoC 타입중 가장 익숙한 타입으로 필요할 때 마다 레지스트리에서 의존성을 가지고 온다. EJB의 경우 JNDI API를 통해 EJB 컴포넌트를 룩업 한다.

컨텍스트화된 의존성 룩업(Contextualized Dependency Lookup)

레지스트리가 아니라 리소스를 관리하는 컨테이너를 대상으로 룩업을 수행하며 보통 정해진 시점에 룩업이 수행된다.



컨테이너는 내부 WAS(톰캣, JBOSS등)나 스프링프레임워크에서 제공한다.

Injection vs Lookup

어떤 IoC 방식을 사용할지는 별로 어렵지 않다. 스프링의 경우 초기 빈 Lookup을 제외하면 의존성이 항상 Injection 형태의 IoC를 사용하게 된다. 대부분의 환경에서 DI를 사용해서 모든 객체를 연결할 수는 없으며 초기 컴포넌트에 접근 시 DL을 사용해야 하는 경우가 많다. 예를 들어 일반 자바APP에서는 main에서 스프링 컨테이너를 부트스트랩하고 ApplicationContext 인터페이스를 통해 의존객체를 가져와야 한다. 즉 스프링에서는 DI를 사용할 수 있으면 사용하고 그렇지 못한 경우라면 DL을 사용하면 된다.

의존성 풀 방식의 코드는 레지스트리에 대한 참조를 가지고 있어야 하고 레지스트리와 연동해서 의존성 객체를 가지고 와야 한다. 또한 CDL을 사용시 특정 인터페이스를 구현해야 하고 모든 의존성을 직접 가지고 와야 한다. 하지만 의존성 주입(Dependency Injection)을 사용하면 적절한 생성자, 메소드, 세터 등을 통해 의존성을 주입 받기만 하면 된다.

IoC 컨테이너 분류체계

- **DI(dependency Injection)**

  -Setter Injection : class 사이의 의존관계를 연결시키기 위해 setter 메소드를 이용하는 방법.

  -Constructor Injection : class 사이의 의존관계를 연결시키기 위해 생성자를 이용하는 방법.

  -Method Injection : Method Injection은 Setter Injection과 Constructor Injection이 가지고 있는 한계점을 극복하기 위하여 지원하고 있는 DI의 한 종류이다. 어떤 메소드의 실행을 다른 메소드로 대체한다든지 또는 메소드의 리턴형을 추상클래스로 지정한 후 필요에 따라 추상클래스를 상속받은 임의의 객체를 리턴 하도록 구성할 수 있다.

Setter Injection vs Constructor Injection

생성자 주입은 컴포넌트를 사용하기 전에 의존 해야하는 클래스의 인스턴스를 가지고 있어야 할 때 유용하며 세터주입은 부모 컴포넌트의 새 인스턴스를 생성하지 않고 동적으로 각기 다른 구현을 사용해 의존성을 대체할 수 있다. 즉 의존성을 인터페이스로 선언할 수 있다는 점이 장점이며 덜 강압적인 주입이라 할 수 있다.

기본 생성자만 있는 클래스에 생성자 주입을 정의하면 비 IoC 환경에서 해당 클래스를 사용하는 모든 클래스에 영향을 주게 된다. 하지만 세터 주입의 경우 다른 클래스가 이 클래스와 상호작용 하는데 있어 영향을 덜 주게 된다.

일반적으로 세터 주입을 이용하는 것이 좋다. 이 방식은 비 IoC 설정에서 코드를 사용하는데 최소한의 영향만을 주기 때문이다. 생성자 주입은 컴포넌트로 항상 의존성이 전달되게 할 때 적합하다.

Setter Injection(XML 기반)

Emp.java

```
package edu.ojc.setter1;

interface Emp {
 public abstract void gotoOffice();
 public abstract void getoffWork();
}
```

Programmer.java

```
package edu.ojc.setter1;

class Programmer implements Emp {
public void gotoOffice() {
        System.out.println("프로그래머 출근 합니다.");
}
public void  getoffWork() {
        System.out.println("프로그래머 퇴근 합니다.");
}
}
```

Setter Injection(XML 기반)

Designer.java

```
package edu.ojc.setter1;


class Designer implements Emp {
public void gotoOffice() {
	System.out.println("디자이너 출근 합니다.");
}
public void  getoffWork() {
	System.out.println("디자이너 퇴근 합니다.");
}
}
```

Setter Injection(XML 기반)

Develope.java

```
package edu.ojc.setter1;


public class Develope {
	Emp emp;


	public void setEmp(Emp emp) {
		this.emp = emp;
	}
	void coding() {
	emp.gotoOffice();
		System.out.println("개발합니다...");
		emp.getoffWork();
	}
}
```

- **Setter Injection(XML 기반)**
  - *Src/main/resources/ojc1.xml – p 네임스페이스를 이용한 세터주입*

```
<?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="programmer" class="edu.ojc.setter1.Programmer" />
<bean id="designer" class="edu.ojc.setter1.Designer" />
<bean id="develope" class="edu.ojc.setter1.Develope" p:emp-
ref="programmer" />
<!--
//아래처럼 Property 태그를 이용해도 됩니다.
<bean id="develope" class="edu.ojc.setter1.Develope">
<property  name="emp“><ref bean="programmer"/></property>
</bean>
<bean id="develope" class="edu.ojc.setter1.Develope">
<property name="emp" ref="programmer"/> </bean>
 -->
</beans>
```

Setter Injection(XML 기반)

TestMain.java

```
package edu.ojc.setter1;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlAppli
cationContext;

public class TestMain {
public static void main(String[] args) {
 ApplicationContext context =
              new
ClassPathXmlApplicationContext("ojc1.xml");

      Develope dev =
(Develop)context.getBean("develope");
      dev.coding();
}
}
```

Setter Injection(Annotation 기반)

Emp.java

```
package edu.ojc.setter1;

interface Emp {
 public abstract void gotoOffice();
 public abstract void getoffWork();
}
```

Programmer.java

```
package edu.ojc.setter1;
 import org.springframework.stereotype.Component;
@Service
class Programmer implements Emp {
public void gotoOffice() {
       System.out.println("프로그래머 출근 합니다.");
}
public void  getoffWork() {
       System.out.println("프로그래머 퇴근 합니다.");
}}
```

Setter Injection(Annotation 기반)

Designer.java

```
package edu.ojc.setter1;
import org.springframework.stereotype.Component;

@Service
class Designer implements Emp {
public void gotoOffice() {
	System.out.println("디자이너 출근 합니다.");
}
public void  getoffWork() {
	System.out.println("디자이너 퇴근 합니다.");
}
}
```

Setter Injection(Annotation 기반)

Develope.java

```
package edu.ojc.setter1;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
@Service
public class Develope {
        Emp emp;
        @Autowired
        @Qualifier("programmer")
        public void setEmp(Emp emp) {
                this.emp = emp;
        }
        void coding() {
        emp.gotoOffice();
                System.out.println("개발합니다...");
                emp.getoffWork();
        }
```

Setter Injection(Annotation 기반)

src/main/resources/ojc2.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"
xmlns:context="http://www.springframework.org/schema/context"
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-4.1.xsd">
        <context:component-scan base-package="edu.ojc.setter2"/>
</beans>
```

Setter Injection(Annotation 기반)

TestMain.java

```
package edu.ojc.setter1;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlAppli
cationContext;

public class TestMain {
public static void main(String[] args) {
 ApplicationContext context =
              new
ClassPathXmlApplicationContext("ojc1.xml");

      Develope dev =
(Develop)context.getBean("develope");
      dev.coding();
}
}
```

Constructor Injection(XML 기반)

Emp.java

```
package edu.ojc.constructor1;

interface Emp {
 public abstract void gotoOffice();
 public abstract void getoffWork();
}
```

Programmer.java

```
package edu.ojc.constructor1;

class Programmer implements Emp {
public void gotoOffice() {
       System.out.println("프로그래머 출근 합니다.");
}
public void  getoffWork() {
       System.out.println("프로그래머 퇴근 합니다.");
}
}
```

Constructor Injection(XML 기반)

Designer.java

```
package edu.ojc.constructor1;

class Designer implements Emp {
public void gotoOffice() {
	System.out.println("디자이너 출근 합니다.");
}
public void  getoffWork() {
	System.out.println("디자이너 퇴근 합니다.");
}
}
```

Constructor Injection(XML 기반)

Develope.java

```
package edu.ojc.constructor1;


public class Develope {
	Emp emp;

	 public Develope(Emp emp) {
		this.emp = emp;
	}

	void coding() {
	emp.gotoOffice();
		System.out.println("개발합니다...");
		emp.getoffWork();
	}
}
```

Constructor Injection(XML 기반)

◆ *src/main/resources/ojc3.xml – c 네임스페이스를 이용한 생성자주입*

```
<?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:c="http://www.springframework.org/schema/c"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="programmer" class="edu.ojc.constructor1.Programmer" />
<bean id="designer" class="edu.ojc.constructor1.Designer" />
<bean id="develope" class="edu.ojc.constructor1.Develope" c:emp-
ref="programmer" />
<!--
<bean id="develope" class="edu.ojc.constructor1.Develope">
        <constructor-arg ref="programmer"/>
</bean>
-->
</beans>
```

Constructor Injection(XML 기반)

TestMain.java

```
package edu.ojc.constructor1;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplication
Context;

public class TestMain {
public static void main(String[] args) {
 ApplicationContext context =
            new ClassPathXmlApplicationContext("ojc3.xml");

      Develope dev = (Develope)context.getBean("develope");
      dev.coding();
}
}
```

Constructor Injection(Annotation 기반)

Emp.java

```
package edu.ojc.constructor2;

interface Emp {
 public abstract void gotoOffice();
 public abstract void getoffWork();
}
```

Programmer.java

```
package edu.ojc.constructor2;
import org.springframework.stereotype.Service;
 @Service
class Programmer implements Emp {
public void gotoOffice() {
        System.out.println("프로그래머 출근 합니다.");
}
public void  getoffWork() {
        System.out.println("프로그래머 퇴근 합니다.");
}
}
```

Constructor Injection(Annotation 기반)

Designer.java

```
package edu.ojc.constructor2;

import org.springframework.stereotype.Service;

@Service
class Designer implements Emp {
public void gotoOffice() {
        System.out.println("디자이너 출근 합니다.");
}
public void  getoffWork() {
        System.out.println("디자이너 퇴근 합니다.");
}
}
```

Constructor Injection(Annotation 기반)

Develope.java

```
package edu.ojc.constructor2;
 import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
@Service
public class Develope {
	Emp emp;
	@Autowired
	public Develope(@Qualifier(value="programmer") Emp emp)
{
		this.emp = emp;
	}

	void coding() {
	emp.gotoOffice();
		System.out.println("개발합니다...");
		emp.getoffWork();
	}
```

Constructor Injection(Annotation 기반)

src/main/resources/ojc4.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"
xmlns:context="http://www.springframework.org/schema/context"
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-4.1.xsd">
   <context:component-scan base-package="edu.ojc.constructor2"/>
</beans>
```

Constructor Injection(Annotation 기반)

TestMain.java

```
package edu.ojc.constructor2;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationC
ontext;

public class TestMain {
public static void main(String[] args) {
 ApplicationContext context =
            new ClassPathXmlApplicationContext("ojc4.xml");

      Develope dev = (Develop)context.getBean("develope");
      dev.coding();
}
}
```

Method Injection

Method 정의 전체에 Bean을 주입하는 것

◆ *Method Replace(메소드 대체)*

기존 메소드를 런타임 중에 새로운 구현으로 대체

◆ *Getter Injection(게터 주입)*

기존 메소드를 런 타임중에 spring Context로 부터 특정한 빈을 반환하는 새로운 구현으로 대체. Method 주입의 특별한 경우로 Method가 특정 타입의 빈을 반환하도록 선언해 두고(보통 abstract) 런타임 중 실제 반환되는 빈은 Spring Context에서 오도록 하는 방법

Method Injection(Method Replace)

Emp.java

```
package ojc;

public interface Emp {
        public String work();
}
```

Programmer.java

```
package ojc;

public class Programmer implements Emp {
        public String  work() {
                return "Programmer Working...";
        }
}
```

Method Injection(Method Replace)

Designer.java

```
package ojc;

import java.lang.reflect.Method;

import org.springframework.beans.factory.support.MethodReplacer;
import org.springframework.stereotype.Service;

public class Designer implements MethodReplacer {
        public Object reimplement(Object target, Method method, Object[] args)
throws Throwable {
        return "Designer Working...";
    }
}
```

Method Injection(Method Replace)

MethodReplaceTest.java

```
package ojc;

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

public class MethodReplaceTest {
        Emp programmer;
        public void setProgrammer(Programmer programmer) {
                this.programmer = programmer;
        }

        public static void main(String[] args) {
                ApplicationContext context = new
ClassPathXmlApplicationContext("lookup.xml");
         MethodReplaceTest test = (MethodReplaceTest)
context.getBean("methodReplaceTest");
                //Programmer의 work()가 아닌 대체자인 Designer의 work()가 실행됨
                System.out.println(test.programmer.work());
        }
}
```

Method Injection(Method Replace)

lookup.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"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.1.xsd">
  <bean id="programmer" class="ojc.Programmer">
    <replaced-method name="work" replacer="designer" />
  </bean>

  <bean id="designer" class="ojc.Designer" />

  <bean id="methodReplaceTest" class="ojc.MethodReplaceTest">
        <property name="programmer" ref="programmer"/>
  </bean>
</beans>
```

- **Method Injection**
  - ***Getter Injection***
    [Emp.java]

```
package emp;
public abstract class Emp {
public Emp() {}
  public void work() {
    getEmp().work(); //주입받은 객체의 getEmp() 메소드를 사용
  }
  //getEmp를 주입, Emp가 주입된다.
  //Run-Time중에 xml 파일로 부터 빈을 주입받는다.
  public abstract Emp getEmp() ;
}
```

- **Method Injection**
  - ***Getter Injection***
    [Programmer.java]

```
package emp;

public class Programmer extends Emp{
    public Emp getEmp() {
    return this;
    }

    public void work() {
    System.out.println("프로그래머가 개발을 합니다.");
    }
}
```

DI(Dependency Injection)

- **Method Injection**
  - ***Getter Injection***
    [Designer.java]

```
package emp;

public class Designer extends Emp{
    public Emp getEmp() {
    return this;
    }

    public void work() {
    System.out.println("디자이너가 디자인을 합니다.");
    }
}
```

Method Injection

Getter Injection [src/main/resources/emp.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/schem
a/beans http://www.springframework.org/schema/beans/spring-
beans.xsd">
       <bean id="myemp" class="emp.Programmer" />
       <!-- <bean id="myemp" class="emp.Designer" /> -->
       <!-- 아래 lookup-method는 메소드를 런타입중에 새로운 구
현으로 대체 Getter주입은 Setter Injection의
                역 으로 생각. getEmp()를 호출할 때마다 다른
myemp가 반환 그러나 Setter Injectioin은 단 한번만 주입 주입되는
                메소드명이 반드시 getXXX로 시작할 필요는 없다. --
>
       <bean id="emp" class="emp.Emp">
              <lookup-method name="getEmp" bean="myemp" />
       </bean>
</beans>
```