IoC (Inversion of Control)

1. Ioc (Inversion of Control : 제어의 역전현상)이란?

   기존에 자바 기반으로 어플리케이션을 개발할 때 자바 객체를 생성하고 서로 간의 의존 관계를 연결시키는 작업에 대한 제어권은 보통 개발되는 어플리케이션에 있었다. 그러나 Servlet, EJB등을 사용하는 경우 Servlet Container, EJB Container에게 제어권이 넘어가서 객체의 생명주기를 Container들이 전담하게 된다. 이처럼 IoC에서 이야기하는 제어권의 역전이란 객체의 생성에서부터 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미한다.

2. IoC의 장/단점

   - 장점

     인터페이스 기반 설계가 가능

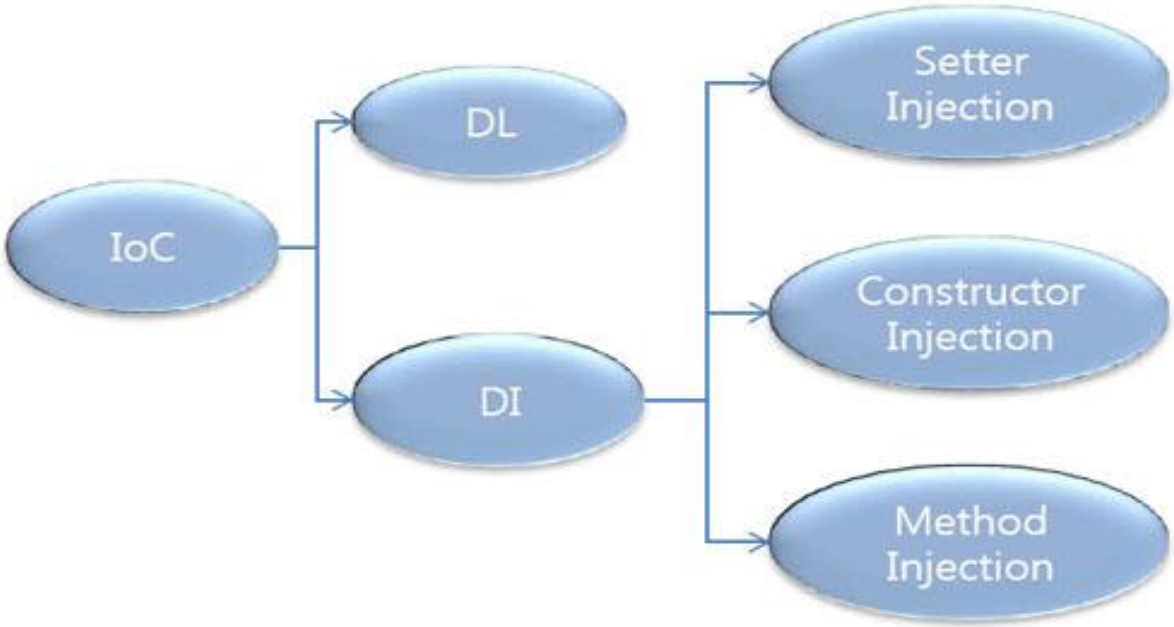
     컴퍼넌트 재사용성 증가

     체계적이고 효율적인 Dependency 관리

   - 단점

     제어구조가 반대로 되어있으므로 이해하기 어려운 코드가 될 수 있음

3. IoC 분류체계



DL (Dependency Lookup)

DL은 저장소에 저장되어 있는 빈(Bean)에 접근하기 위하여 개발자들이 컨테이너에서 제공하는 API를 이용하여 사용하고자 하는 빈(Bean)을 Lookup하는 것을 말한다. 자바 개발자들에게 친숙한 전통적인 접근법이다.
Dependency Full이라고도 불리운다.

DL 예제)

```
ApplicationContext context = new ClassPathXmlApplicationContext("test/dl.xml");
ObjDAO dao = (ObjDAO) context.getBean("svc");
```

DI (Dependency Injection)

DI는 Spring 프레임워크에서 지원하는 IoC의 형태이다.
클래스 사이의 의존관계를 빈 설정 정보를 바탕으로 컨테이너가 자동적으로 연결해주는 것을 말한다.
개발자들은 빈 설정 파일(저장소 관리 파일)에 의존관계가 필요하다는 정보를 추가하면 된다.
Spring 프레임워크는 Setter Injection, Constructor Injection, Method Injection 의 세가지 유형으로 나타난다.

1) Setter Injection
   클래스 사이의 의존관계를 연결시키기 위하여 setter 메소드를 이용하는 방법이다.
   예제)
   - 자바

```
public class ObjSetterTest{
   private ObjDAO dao;
```

```
    public void setObjDAO(ObjDAO dao){
      this.dao = dao;
    }
  }
```

■ xml 파일

```
<bean id="objDao" class="homework1.ObjDAO"/>
<bean id="objSetter" class="homework1.ObjSetterTest">
  <property name="dao" ref=" objDao"/>
</bean>
```

2) Constructor Injection

생성자를 이용하여 의존관계를 연결시킨다.

예제)

■ 자바

```
public class ObjConstructorTest{
  private ObjDAO dao;
  public ObjConstructorTest (ObjDAO dao){
    this.dao = dao;
  }
}
```

■ xml 파일

```
<bean id="objDao" class="homework2.ObjDAO"/>
<bean id="objCons" class="homework2.ObjConstructorTest ">
  <constructor-arg>
    <ref bean="objDao/>
  </constructor-arg>
 </bean>
```

3) Method Injection

Setter Injection과 Constructor Injection이 가지고 있는 한계점을 극복하기 위하여 Spring 프레임워크 1.1버전에서 새롭게 지원하고 있는 DI의 한 종류이다.

Singleton 인스턴스와 Non Singleton 인스턴스의 의존관계를 연결할 필요가 있을 때 사용한다.
즉, 컴퍼넌트의 dependency가 일반 메소드를 통해서 제공되는 방식이다.(거의 사용되지 않음)

예제)

■ 자바

```
public abstract class CarFactory{
   public abstract Car makeBlackCar();
   public abstract Car makeWhiteCar();
}

public class Car{
   prviate String color="";
   public void setColor(String color){
      this.color = color;
   }

   public String toString(){
      return "차색깔 : "+ this.color;
   }
}

public class MainTest{
   public static void main(String[] args) throws Exception{
      ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("homework3/di.xml");
   CarFactory carFactory =
context.getBean("carFactory",CarFactory.class);
   System.out.println("blackCar : "+ carFactory.makeBlackCar());
   System.out.println("whiteCar : "+ carFactory.makeWhiteCar());
```

```
    }
}
```

■ xml

```
<bean id="carFactory" class="homework3.carFactory">
    <lookup-method name=" makeBlackCar" bean="blackCarType"/>
    <lookup-method name=" makeWhiteCar" bean="whiteCarType"/>
</bean>

<bean id="blackCarType" class="homework3.Car"
scope="prototype">
    <property name="color" value="black"/>
</bean>
<bean id="whiteCarType" class="homework3.Car"
scope="prototype">
    <property name="color" value="white"/>
</bean>
```