Day 06 (2019-01-23)
1. Dependency Injection

(1) 왜 필요한가?

1) 의존관계란?

한 객체가 다른 객체를 생성하거나 다른 객체를 사용할 때(메소드를 호출 할 때) 객체에 의존한다고 한다

```java
package io.crscube.semina.day02;

/**
 * Created by taesu at : 2019-01-18
 * <p>
 * 여기에 FlowController 클래스에 대한 설명을 기술해주세요
 *
 * @author taesu
 * @version 1.0
 * @since 1.0
 */
public class FlowController {
    public void process(){
        FileDataReader fileDataReader = new FileDataReader( path: "resources/test.txt");
        byte[] plainBytes = fileDataReader.read();

        ByteEncryptors encryptors = new ByteEncryptors();
        byte[] encropt = encryptors.encropt(plainBytes);

        new FileDataWriter().write(encropt);
    }
}
```

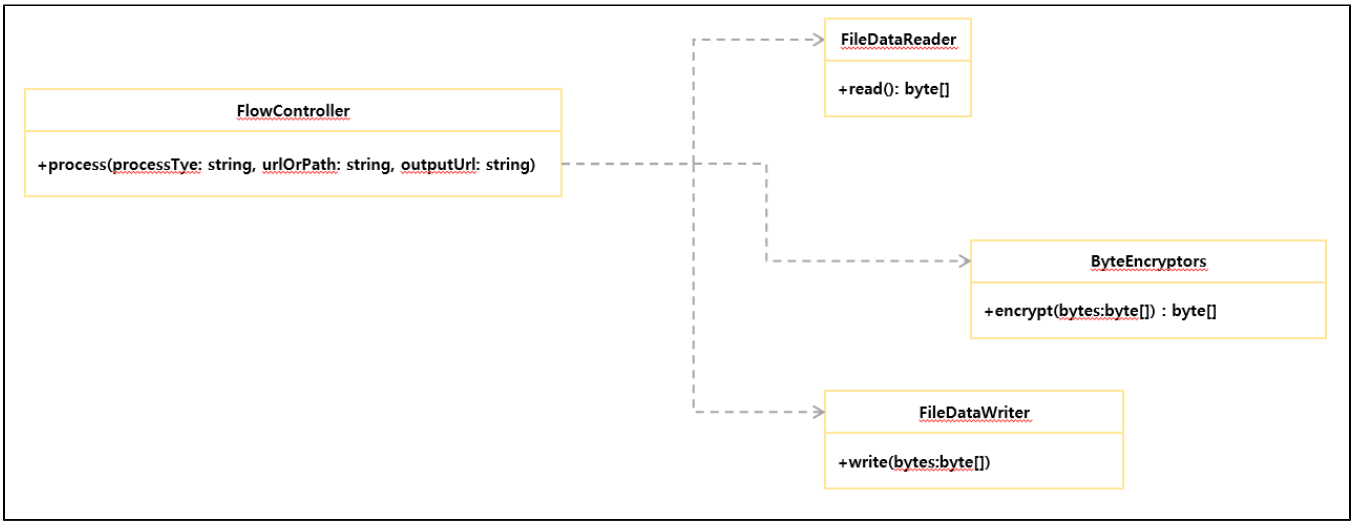
```java
public class ByteEncryptors {
    public byte[] encropt(byte[] plainBytes){
        //복잡한, 고성능 암호화 처리
        return plainBytes;
    }
}
```

FileDataReader.java

```java
public class FileDataReader {
    private String path;

    public FileDataReader(String path) { this.path = path; }

    public byte[] read() { return path.getBytes(); }
}
```

FileDataWriter.java

```java
package io.crscube.semina.day02;

/**
 * Created by taesu at : 2019-01-18
 * <p>
 * 여기에 FileDataWriter 클래스에 대한 설명을 기술해주세요
 *
 * @author taesu
 * @version 1.0
 * @since 1.0
 */
public class FileDataWriter {
    public void write(byte[] bytes){
        //원격지에 write
    }
}
```



FlowController는 FileDataReader, ByteEncryptors, FileDataWriter에 의존하고 있다

2) 인기 폭발 파일 암호화 모듈

〉 파일 암호화 모듈의 성능이 매우 뛰어나 이곳저곳에서 솔루션을 구매하겠다는 업체가 늘어남

〉 요구사항....

- 원격 서버에서 읽은 파일을 암호화 해서 다시 원격 서버로 올려주세요 (프로토콜은 http)

```java
import org.springframework.util.ObjectUtils;

/**
 * Created by taesu at : 2019-01-18
 * <p>
 * 여기에 FlowController 클래스에 대한 설명을 기술해주세요
 *
 * @author taesu
 * @version 1.0
 * @since 1.0
 */
public class FlowController {
    public void process(String processType, String urlOrPath, String outputUrl) {
        byte[] plainBytes;
        if(ObjectUtils.nullSafeEquals(processType, "FILE")) {
            FileDataReader fileDataReader = new FileDataReader(urlOrPath);
            plainBytes = fileDataReader.read();
        } else {
            HttpFileDataReader httpFileDataReader = new HttpFileDataReader(urlOrPath);
            plainBytes = httpFileDataReader.read();
        }


        ByteEncryptors encryptors = new ByteEncryptors();
        byte[] encropt = encryptors.encropt(plainBytes);

        if(ObjectUtils.nullSafeEquals(processType, "FILE")) {
            new FileDataWriter().write(encropt);
        } else {
            new new HttpFileDataWriter().write(outputUrl, encropt);
        }
    }
}
```

```java
public class ByteEncryptors {
    public byte[] encropt(byte[] plainBytes){
        //복잡한, 고성능 암호화 처리
        return plainBytes;
    }
}
```

FileDataReader.java / HttpFileDataReader.java

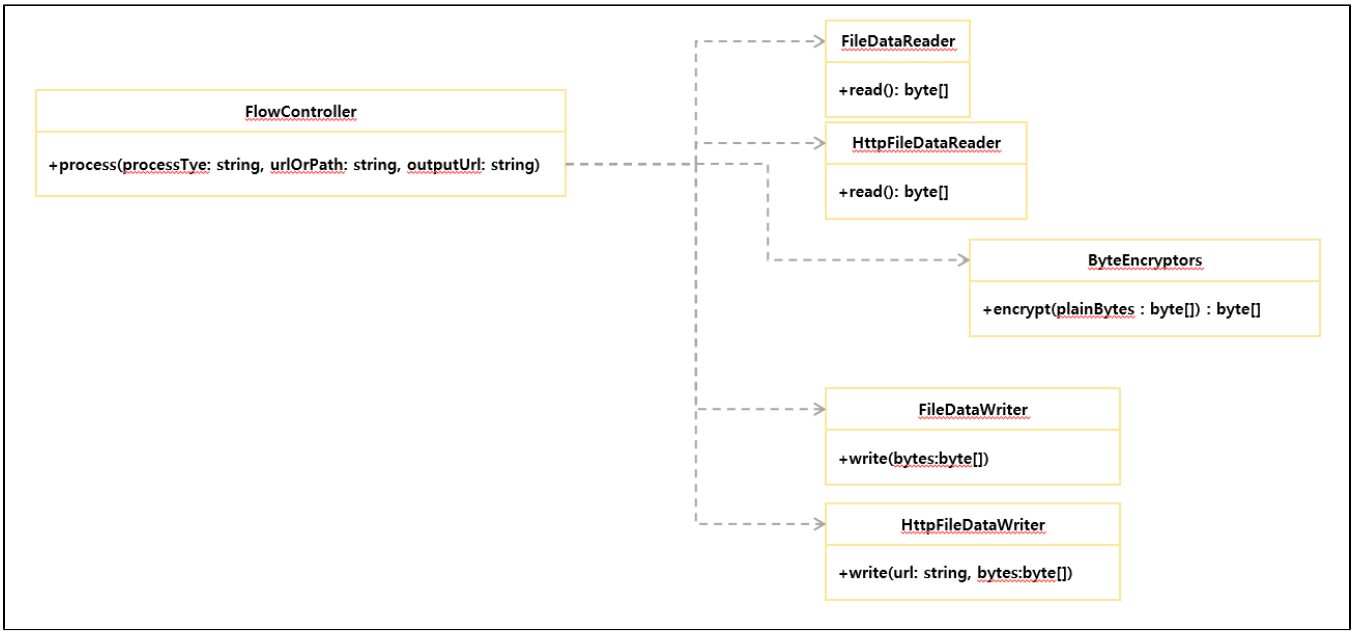
HttpFileDataReader > read()

```java
public HttpFileDataReader(String url) { this.url = url; }

public byte[] read(){
    //전달받은 URL로부터 읽어들이는 코드
    //편의상 getBytes로...
    return url.getBytes();
}
```

FileDataWriter.java / HttpFileDataWriter.java

HttpFileDataWriter > write()

```java
/**
 * Created by taesu at : 2019-01-18
 * <p>
 * 여기에 HttpFileDataWriter 클래스에 대한 설명을 기술해주세요
 *
 * @author taesu
 * @version 1.0
 * @since 1.0
 */
public class HttpFileDataWriter {
    public void write(String url, byte[] bytes){
        //원격지에 파일 업로드
    }
}
```



FlowController가 **의존하는 구현체가 늘어남**(HttpFileDataReader, HttpFileDataWriter)

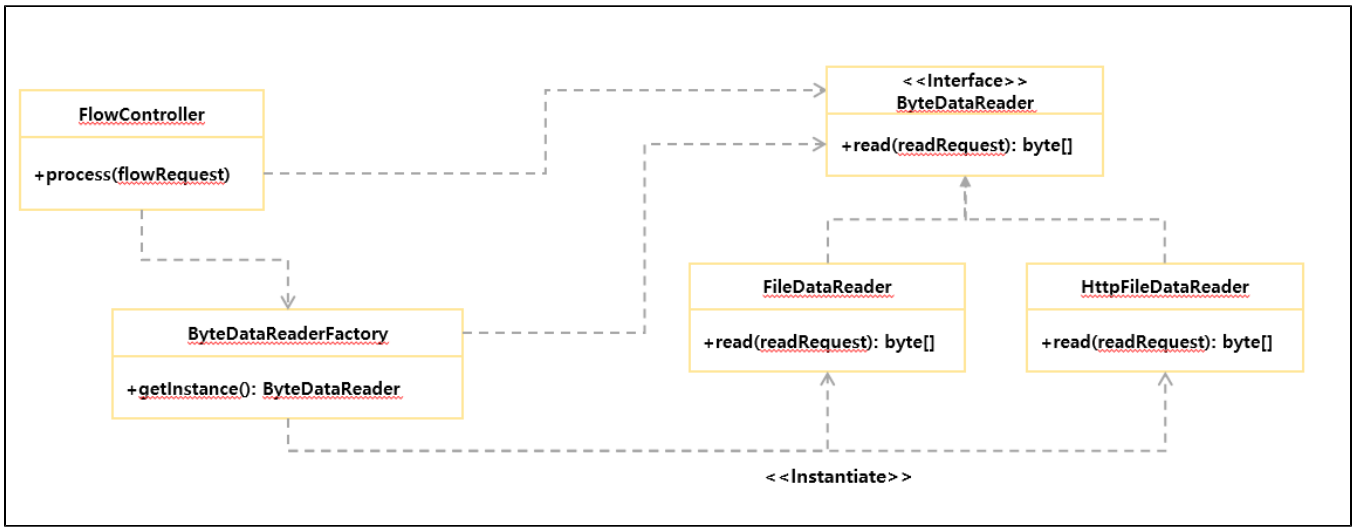
FlowController의 구현에 **if-else 문이 중첩**

만약 요구사항이 늘어난다면?

- Http로 받아서 Local 디렉토리에 저장해주세요
- Local에서 받아서 Http로 업로드 해주세요
- FTP 프로토콜도 지원하게 해주세요
- **늘어나는 if-else문과 버그, 유지보수성 저하**

→ 모든 원인은 의존 대상 객체에 대한 생성을 직접 하기때문 (구현체를 직접 생성하여 직접적으로 의존하기 때문)

※ 이상적인 의존관계 (Reader에 대해서만)



- FlowConroller가 FileDataReader, HttpFileDataReader로부터 필요한 것은 **'어디선가' byte 배열 데이터를 읽어오는 것.**
- 어디선가 byte 배열 데이터를 읽어오는 동작을 추상화 한 ByteDataReader라는 Interface(고수준 모듈)에 의존하고 있음 (요구사항이 자주 변하는 부분을 추상화)
- ByteDataReaderFactory를 통해 의존 대상을 동적으로 주입받고 있음

(2) Spring에서 제공하는 Dependency Injection 방법

Bean에게 주입하는 대상은 반드시 Bean이어야 한다

1) Constructor injection

대상 Bean

```
public class ConstructorInjection {
   private ByteDataReader byteDataReader;

   public ConstructorInjection(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader(){
      return this.byteDataReader;
   }
}
```

XML 방식

```
<bean id="httpFileDataReaderImpl" class="io.crscube.semina.day02.injection.HttpFileDataReaderImpl" />
<bean id="constructorInjection" class="io.crscube.semina.day02.injection.ConstructorInjection">
   <constructor-arg ref="httpFileDataReaderImpl" />
</bean>
```

Spring 4.3부터는 대상 Bean이 하나의 생성자만을 가지는 경우 @Autowired가 필요 없음

Java를 이용한 방식

```
public class ConstructorInjection {
   private ByteDataReader byteDataReader;

   @Autowired
   public ConstructorInjection(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader(){
      return this.byteDataReader;
   }
}
```

2) Setter injection

대상 Bean

```
public class SetterInjection {
   private ByteDataReader byteDataReader;

   public void setByteDataReader(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader() {
      return byteDataReader;
   }
}
```

XML 방식

```
<bean id="httpFileDataReaderImpl" class="io.crscube.semina.day02.injection.HttpFileDataReaderImpl" />
<bean id="setterInjection" class="io.crscube.semina.day02.injection.SetterInjection">
   <property name="byteDataReader" ref="httpFileDataReaderImpl" />
</bean>
```

Java를 이용한 방식

```
public class SetterInjection {
   private ByteDataReader byteDataReader;

   @Autowired
   public void setByteDataReader(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader() {
      return byteDataReader;
   }
}
```

3) Field injection

Java를 이용한 방법만 지원한다

대상 Bean

```
public class FieldInjection {
 @Autowired
// @Resource
// @Inject
   private ByteDataReader byteDataReader;

   public ByteDataReader getByteDataReader() {
      return byteDataReader;
   }
}
```

- @Autowired, @Resource를 사용할 경우 annotation-config가 활성화 되어있어야 injection 된다
- @Inject를 사용할 경우 아래의 dependency 추가가 필요하다(annotation-config는 없어도 된다)

```
<dependency>
   <groupId>javax.inject</groupId>
   <artifactId>javax.inject</artifactId>
   <version>1</version>
</dependency>
```

XML 방식

```
<context:annotation-config />
<bean id="httpFileDataReaderImpl" class="io.crscube.semina.day02.injection.HttpFileDataReaderImpl" />
<bean id="fieldInjection" class="io.crscube.semina.day02.injection.FieldInjection" />
```

※ Autowired, Inject, Resource

앞서 사용한 @Autowired는 @Inject 또는 @Resource로 대체 가능하다

	@Autowired	@Inject	@Resource
범용	스프링 전용	자바에서 지원	자바에서 지원
연결 방식	타입에 맞춰서 연결 → Bean의 class(type)을 우선으로 bean 객체 검색	타입에 맞춰서 연결 → 객체의 class (type)을 우선으로 검색	이름으로 연결 → Bean의 이름(id)을 우선으로 bean 객체 검색
탐색 순서	1. 타입이 같은 bean 객체를 검색한다. @Qualifier가 있을 경우 Qualifier Value까지 같은 객체를 찾는다. 2. 타입이 같은 bean 객체가 2개 이상이면, @Qualifier가 지정한 bean 객체를 찾는다. 3. 타입이 같은 bean 객체가 2개 이상이고, @Qualifier가 지정되어 있지 않으면, 변수명과 id가 같은 bean 객체를 찾는다	@Autowired와 비슷하게 동작	1. @Resource가 지정한 name과 id가 같은 bean 객체를 찾는다. 2. 찾지 못하면, 타입이 같은 bean 객체를 찾는다. 3. 만약 타입이 같은 bean 객체가 2개 이상이면, 변수명과 id가 같은 bean 객체를 찾는다. 4. 그래도 못찾았으면, @Qualifier가 지정한 bean 객체를 찾는다

※ 자동 주입과 명시적 의존 주입 간의 관계

@Autowired, @Inject, @Resource등을 이용하는경우 자동으로 주입을 받을수 있어 편리하지만,
어떤 구현체가 주입 되는지는 qualifier를 통해 직접 살펴보아야 한다

명시적으로 주입하는 것은 XML 방식으로 진행하며 직접적인 설정이 필요하기에 불편하지만,
어떤 구현체가 주입되는지 직접 확인할 수 있다.

→ 보통 개발자의 취향차라고 하지만 Spring의 표준은 Java를 이용한 방법을 표준화 하는 추세이다

※ 어떤 Injection 방법을 사용해야 할까?

Constructor Injection을 권장하는 추세

관련자료

- *아래 1~5 내용에 대한 원문*
- *Spring DI atterns (the-good-the-bad-and-the-ugly)*
- Angular의 Dependency Injection

1. 단일 책임의 원칙

생성자의 인자가 많을 경우 코드량도 많아지고, 의존관계도 많아져 단일 책임의 원칙에 위배됨을 바로 알 수 있다(Setter injection도 동일하다). 그래서 Constructor Injection을 사용함으로써 의존관계, 복잡성을 쉽게 알수 있어 리팩토링의 단초를 제공하게 된다.

2. 테스트 용이성

DI 컨테이너에서 관리되는 클래스는 특정 DI 컨테이너에 의존하지 않고 POJO여야 한다.
DI 컨테이너를 사용하지 않고도 인스턴스화 할 수 있고, 단위 테스트도 가능하며, 다른 DI 프레임 워크로 전환할 수도 있게 된다. (Setter injection도 동일하다)
→ 대부분 테스트에서 사용하는 MockMVC를 이용할 때 mock 된 bean을 주입해야하는데 Field Injection에선 매우 번거로운 작업니다

3. Immutability

Constructor Injection에서는 필드는 final로 선언할 수 있다.
불변 객체가 가능한데 비해 Setter Injection 및 Field Injection은 final는 선언할 수 없기 때문에 객체가 변경 가능한 상태가 된다.

4. 순환 의존성

Constructor Injection에서는 멤버 객체가 순환 의존성을 가질 경우 BeanCurrentlyInCreationException이 발생해서 순환 의존성을 알 수 있게 된다.
→ 순환의존관계가 존재하면 변경의 연쇄작용이 발생할 수 있어 반드시 끊어내는 것이 중요하다

5. 의존성 명시

의존 객체 중 필수는 Constructor Injection을 옵션인 경우는 Setter Injection을 활용할 수 있다.
→ 필수 의존관계와 선택전 의존관계를 명시할 수 있다
→ Field Injection을 통해서도 가능하다
@Autowired(required=false)

2. Component scan

Bean을 구현할 때마다 XML에 추가해야하나?
→ 여전히 벗어나지 못한 XML 지옥

(1) Component scan

Spring Container는 특정 패키지 하위에 약속된 Annotation을 선언한 Class를 찾아 이를 Bean으로 등록하는 기능인 Component scan을 제공한다

약손된 Annotation은 @Component와 이를 상속하는 Annotation등이 있다(@Controller, @RestController, @Service, @Repository 등)

XML 방식

```
<context:component-scan base-package="io.crscube.semina.day02.scan" />
```

대상 Bean

```
package io.crscube.semina.day02.scan;

import org.springframework.stereotype.Component;


@Component
public class ComponentBean {
   private ByteDataReader byteDataReader;

   public ComponentBean(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader() {
      return byteDataReader;
   }
}
```

대상 Bean

```
/******************    ByteDataReader .java ***********************/
package io.crscube.semina.day02.scan;


public interface ByteDataReader {
}

/******************    HttpByteDataReaderImpl .java ***********************/
package io.crscube.semina.day02.scan;

import org.springframework.stereotype.Service;


@Service
public class HttpByteDataReaderImpl implements ByteDataReader {
}
```

(2) @Bean과 @Component

Component scan은 개발자에게 매우 큰 생산성을 부여한다.

그렇다면 @Configuration, @Bean 등을 선언하는 방법은 잊어도 될까?

1) Component scan으로 등록 불가능한 Bean 예제

- `com.fasterxml.jackson.databind.ObjectMapper`
    - `JasonParser`

        - 비즈니스로직에서 JsonParser를 사용할 필요가 있다면?
    - Datasource
        - Spring에서 제공하는 JndiObjectFactoryBean을 통해 JNDI Datasource를 사용하려면?

→ 우리가 직접 개발하지 않은(우리가 컨트롤 불가능한), 외부의 라이브러리 등을 Bean으로 등록해야할 경우
XML 또는 Java configuration을 통해 Bean을 직접 등록하여 사용해야한다

→ 그 외에 우리가 직접 개발한 것들은 모두 Component scan이 가능하다

(3) Java Configuration을 통해 Component scan 설정 해보기

Main

```
package io.crscube.semina.day02.scan;
public class AnnotationConfigurationApplication {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new AnnotationConfigApplicationContext
(ComponentScanByJavaConfiguration.class);
      ComponentBean componentBean = Objects.requireNonNull(applicationContext.getBean("componentBean",
ComponentBean.class));
      System.out.println(componentBean.getByteDataReader());
   }
}
```

Component

```
package io.crscube.semina.day02.scan;

import org.springframework.stereotype.Component;

@Component
public class ComponentBean {
   private ByteDataReader byteDataReader;

   public ComponentBean(ByteDataReader byteDataReader) {
      this.byteDataReader = byteDataReader;
   }

   public ByteDataReader getByteDataReader() {
      return byteDataReader;
   }
}
```

Service interface

```
package io.crscube.semina.day02.scan;

public interface ByteDataReader {
}
```

Service implementation

```
package io.crscube.semina.day02.scan;

import org.springframework.stereotype.Service;

@Service
public class HttpByteDataReaderImpl implements ByteDataReader {
}
```

Java configuration

```
package io.crscube.semina.day02.scan;

public class ComponentScanByJavaConfiguration {
//??????????
}
```

3. Annotation config vs Component scan vs Annotation driven

(1) Annotation Config

Spring-context namespace에 선언된 xml element

spring-context-4.3.xsd

```
<xsd:element name="annotation-config">
   <xsd:annotation>
      <xsd:documentation><![CDATA[
Activates various annotations to be detected in bean classes: Spring's @Required and
@Autowired, as well as JSR 250's @PostConstruct, @PreDestroy and @Resource (if available),
JAX-WS's @WebServiceRef (if available), EJB 3's @EJB (if available), and JPA's
@PersistenceContext and @PersistenceUnit (if available). Alternatively, you may
choose to activate the individual BeanPostProcessors for those annotations.

Note: This tag does not activate processing of Spring's @Transactional or EJB 3's
@TransactionAttribute annotation. Consider the use of the <tx:annotation-driven>
tag for that purpose.

See javadoc for org.springframework.context.annotation.AnnotationConfigApplicationContext
for information on code-based alternatives to bootstrapping annotation-driven support.
      ]]></xsd:documentation>
   </xsd:annotation>
</xsd:element>
```

Bean 생성을 위해 클래스에 선언 된 Annotation에 대한 감지기능을 활성화 한다

Spring의 **@Required (RequiredAnnotationBeanPostProcessor), @Autowired (AutowiredAnnotationBeanPostProcessor)**뿐 아니라
JSR 250의 **@PostConstruct, @PreDestroy, @Resource (CommonAnnotationBeanPostProcessor)**
JAX-WS의 **@WebServiceRef** 및 JPA의 **@PersistanceConetxt, @PersistenceUnit**, EJB 3의 @EJB
그리고 Bean 설정에 대한 내용을 담는 **@Configuration (ConfigurationClassPostProsser)**을 를 탐지

→ 만약 annotation-config 기능이 없다면 위 설정을 모두 선언하여 활성화 시켜주어야 했을 것

해당태그는 Spring의 @Transactional을 활성화 시키지 않으며 〈tx:annotation-driven/〉을 통해 설정 가능하다

(2) Component scan

Spring-context namespace에 선언된 xml element

spring-context-4.3.xsd

```
<xsd:element name="component-scan">
   <xsd:annotation>
      <xsd:documentation><![CDATA[
Scans the classpath for annotated components that will be auto-registered as
Spring beans. By default, the Spring-provided @Component, @Repository, @Service,
@Controller, @RestController, @ControllerAdvice, and @Configuration stereotypes
will be detected.

Note: This tag implies the effects of the 'annotation-config' tag, activating @Required,
@Autowired, @PostConstruct, @PreDestroy, @Resource, @PersistenceContext and @PersistenceUnit
annotations in the component classes, which is usually desired for autodetected components
(without external configuration). Turn off the 'annotation-config' attribute to deactivate
this default behavior, for example in order to use custom BeanPostProcessor definitions
for handling those annotations.

Note: You may use placeholders in package paths, but only resolved against system
properties (analogous to resource paths). A component scan results in new bean definitions
being registered; Spring's PropertySourcesPlaceholderConfigurer will apply to those bean
definitions just like to regular bean definitions, but it won't apply to the component
scan settings themselves.

See javadoc for org.springframework.context.annotation.ComponentScan for information
on code-based alternatives to bootstrapping component-scanning.
      ]]></xsd:documentation>
   </xsd:annotation>
```

Spring의 Bean으로 자동 등록 될 Annotated component를 classpath로부터 scan한다

기본적으로 Spring에서 제공하
는 @Component, @Repository, @Service, @Controller, @RestConroller, @ConrollerAdvice, @Configuration에 해당하는 Stereotype들이 탐지된다

해당태그는 annotation-config 태그를 암시적으로 포함하며, 비활성화 하려면 annotation-config 속성을 명시적으로 해제해야 한다.

spring-context-4.3.xsd

```
<xsd:complexType>
      <xsd:sequence>
         <xsd:element name="include-filter" type="filterType" minOccurs="0" maxOccurs="unbounded">
         </xsd:element>
         <xsd:element name="exclude-filter" type="filterType" minOccurs="0" maxOccurs="unbounded">
         </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="base-package"                type="xsd:string"   use="required">
      </xsd:attribute>
      <xsd:attribute name="resource-pattern"         type="xsd:string">
      </xsd:attribute>
      <xsd:attribute name="use-default-filters" type="xsd:boolean"  default="true">
      </xsd:attribute>
      <xsd:attribute name="annotation-config"         type="xsd:boolean"  default="true">
....
```

annotation-config 명시적으로 비활성화

```
<context:component-scan base-package="io.crscube.semina.day02.scan" annotation-config="false" />
```

(3) Annotation driven

Spring-mvc namespace에 선언된 xml element

→ 반드시 spring-webmvc dependency가 필요하다

spring-mvc-4.3.xsd

```
<xsd:element name="annotation-driven">
   <xsd:annotation>
      <xsd:documentation source="java:org.springframework.web.servlet.mvc.method.annotation.
RequestMappingHandlerAdapter"><![CDATA[
Configures the annotation-driven Spring MVC Controller programming model.
Note that this tag works in Web MVC only, not in Portlet MVC!

See org.springframework.web.servlet.config.annotation.EnableWebMvc javadoc for details
on code-based alternatives to enabling annotation-driven Spring MVC support.
      ]]></xsd:documentation>
   </xsd:annotation>
   <xsd:complexType>
      <xsd:all minOccurs="0">
         <xsd:element name="path-matching" minOccurs="0">
            <xsd:annotation>
               <xsd:documentation><![CDATA[Configures the path matching part of the Spring MVC Controller
programming model.
Like annotation-driven, code-based alternatives are also documented in EnableWebMvc javadoc.
               ]]></xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
               <xsd:attribute name="suffix-pattern" type="xsd:boolean">
                  <xsd:annotation>
                     <xsd:documentation><![CDATA[
Whether to use suffix pattern match (".*") when matching patterns to requests. If enabled
a method mapped to "/users" also matches to "/users.*".
The default value is true.
                     ]]></xsd:documentation>
                  </xsd:annotation>
```

```
               </xsd:attribute>
               <xsd:attribute name="trailing-slash" type="xsd:boolean">
                  <xsd:annotation>
                     <xsd:documentation><![CDATA[
Whether to match to URLs irrespective of the presence of a trailing slash.
If enabled a method mapped to "/users" also matches to "/users/".
The default value is true.
                           ]]></xsd:documentation>
                  </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="registered-suffixes-only" type="xsd:boolean">
                  <xsd:annotation>
                     <xsd:documentation><![CDATA[
Whether suffix pattern matching should work only against path extensions
explicitly registered when you configure content negotiation.
This is generally recommended to reduce ambiguity and to
avoid issues such as when a "." appears in the path for other reasons.
The default value is false.
                     ]]></xsd:documentation>
                  </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="path-helper" type="xsd:string">
                  <xsd:annotation>
                     <xsd:documentation><![CDATA[
The bean name of the UrlPathHelper to use for resolution of lookup paths.
Use this to override the default UrlPathHelper with a custom subclass, or to share common UrlPathHelper
settings across
multiple HandlerMappings and MethodNameResolvers.
                     ]]></xsd:documentation>
                  </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="path-matcher" type="xsd:string">
                  <xsd:annotation>
                     <xsd:documentation><![CDATA[
The bean name of the PathMatcher implementation to use for matching URL paths against registered URL patterns.
Default is AntPathMatcher.
                     ]]></xsd:documentation>
                  </xsd:annotation>
               </xsd:attribute>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="message-converters" minOccurs="0">
            <xsd:annotation>
               <xsd:documentation><![CDATA[
Configures one or more HttpMessageConverter types to use for converting @RequestBody method parameters and
@ResponseBody method return values. Using this configuration element is optional.
HttpMessageConverter registrations provided here will take precedence over HttpMessageConverter types registered
by default. Also see the register-defaults attribute if you want to turn off default registrations entirely.
               ]]></xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:choice maxOccurs="unbounded">
                     <xsd:element ref="beans:bean">
                        <xsd:annotation>
                           <xsd:documentation><![CDATA[
An HttpMessageConverter bean definition.
                           ]]></xsd:documentation>
                        </xsd:annotation>
                     </xsd:element>
                     <xsd:element ref="beans:ref">
                        <xsd:annotation>
                           <xsd:documentation><![CDATA[
A reference to an HttpMessageConverter bean.
                           ]]></xsd:documentation>
                        </xsd:annotation>
                     </xsd:element>
                  </xsd:choice>
               </xsd:sequence>
               <xsd:attribute name="register-defaults" type="xsd:boolean" default="true">
                  <xsd:annotation>
```

```
                    <xsd:documentation><![CDATA[
Whether or not default HttpMessageConverter registrations should be added in addition to the ones provided
within this element.
                    ]]></xsd:documentation>
                 </xsd:annotation>
              </xsd:attribute>
           </xsd:complexType>
        </xsd:element>
        <xsd:element name="argument-resolvers" minOccurs="0">
           <xsd:annotation>
              <xsd:documentation><![CDATA[
Configures HandlerMethodArgumentResolver types to support custom controller method argument types.
Using this option does not override the built-in support for resolving handler method arguments.
To customize the built-in support for argument resolution configure RequestMappingHandlerAdapter directly.
              ]]></xsd:documentation>
           </xsd:annotation>
           <xsd:complexType>
              <xsd:choice minOccurs="1" maxOccurs="unbounded">
                 <xsd:element ref="beans:bean" minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                       <xsd:documentation><![CDATA[
The HandlerMethodArgumentResolver (or WebArgumentResolver for backwards compatibility) bean definition.
                       ]]></xsd:documentation>
                    </xsd:annotation>
                 </xsd:element>
                 <xsd:element ref="beans:ref" minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                       <xsd:documentation><![CDATA[
A reference to a HandlerMethodArgumentResolver bean definition.
                       ]]></xsd:documentation>
                       <xsd:appinfo>
                          <tool:annotation kind="ref">
                             <tool:expected-type type="java:org.springframework.web.method.support.
HandlerMethodArgumentResolver"/>
                          </tool:annotation>
                       </xsd:appinfo>
                    </xsd:annotation>
                 </xsd:element>
              </xsd:choice>
           </xsd:complexType>
        </xsd:element>
        <xsd:element name="return-value-handlers" minOccurs="0">
           <xsd:annotation>
              <xsd:documentation><![CDATA[
Configures HandlerMethodReturnValueHandler types to support custom controller method return value handling.
Using this option does not override the built-in support for handling return values.
To customize the built-in support for handling return values configure RequestMappingHandlerAdapter directly.
              ]]></xsd:documentation>
           </xsd:annotation>
```

Spring MVC 컴포넌트들을 기본설정을 가지고 활성화 하기위해 사용된다

해당태그 선언 없이도 WebMVC를 위한 컴포넌트(@Controller, @RequestMapping) 등은 잘 동작한다

따라서 mvc:annotation-driven은 Spring MVC를 위한 **좀 더 특별한 동작에 관련**되어있다

- 사용자의 요청을 DispathcerServlet이 받은 후 등록된 HandlerMapping, HandlerAdapter에 따라 요청 파라미터를 가공하고, validation 한다
  → Request 파라미터(또는 body)로 넘어온 문자열을 원하는 타입, Date format 또는 파라미터 전체를 객체로 변환
  → Request body에 전달된 JSON 또는 XML 형태의 문자열을 우리가 원하는 객체의 형태로 변환 (**MessageConverter**)
- 사용자의 요청에 따른 응답을 요구하는 방식에 따라 맞춰 내보낸다.
  - JAXB가 클래스패스에 있고 요청에 명시 되어있다면 XML 형태로
  - Jackson이 클래스패스에 있고 요처엥 명시 되어있다면 JSON 형태로