어노테이션 기반 설정

어노테이션 기반 설정

❖ @Required 어노테이션을 이용한 필수 프로퍼티 검사

- @Required 어노테이션
  - org.springframework.beans.factory.annotation 패키지에 정의
  - 필수 프로퍼티 명시
    - 프로퍼티 설정 메서드에 붙임

```
import org.springframework.beans.factory.annotation.Required;

public class Cameras {
    private int number;

    @Required
    public void setNumber(int number) {
        this.number = number;
    }
}
```

어노테이션 기반 설정

❖ @Required 어노테이션을 이용한 필수 프로퍼티 검사

- ○ @Required 어노테이션
  - ▪ 설정 파일에 RequiredAnnonationBeanPostProcessor를 빈으로 등록(필수)

```
<bean
   class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />

<bean id="camera1" class="com.lecture.spring.homecontrol.Camera">
   <property name="number" value="1" />
</bean>
```

필수 프로퍼티이므로 설정하지 않으면 예외 발생

어노테이션 기반 설정

❖ @Required 어노테이션을 이용한 필수 프로퍼티 검사

- <context:annotation-config> 태그
  - @Required 어노테이션 적용
  - RequiredAnnonationBeanPostProcessor 빈 등록 생략 가능

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:annotation-config/>
    ...
</beans>
```

어노테이션 기반 설정

❖ **<context:annotation-config> 태그가 자동으로 등록해 주는 빈**

- @Required 처리
- @Autowired 처리
- @PostConstruct, @PreDestroy 처리
- @Configuratioon 처리

→ <context:annotation-config> 태그를 사용하여 설정 파일 단순화

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired
  - org.springframework.beans.factory.annotation 패키지에 정의
  - 의존 관계를 자동으로 설정
  - byType 방식으로 의존 관계 설정
  - 생성자, 필드, 메서드의 세 곳에 적용 가능

```
import org.springframework.beans.factory.annotation.Autowired;

public class MonitorViewer implements Viewer {

    private DisplayStrategy displayStrategy;

    @Autowired
    public void setDisplayStrategy(DisplayStrategy displayStrategy) {
        this.displayStrategy = displayStrategy;
    }
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired
  - 일반 메서드에도 적용 가능

```
import org.springframework.beans.factory.annotation.Autowired;

public class HomeController {

    private AlarmDevice alarmDevice;
    private Viewer viewer;
    ...

    @Autowired
    public void prepare(AlarmDevice alarmDevice, Viewer viewer) {
        this.alarmDevice = alarmDevice;
        this.viewer = viewer;
    }
    ...
}
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired
  - 멤버 필드에 적용

```
public class MonitorViewer implements Viewer {

    @Autowired
    private DisplayStrategy displayStrategy;

    ...
}
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired
  - 배열에 적용
    - 해당 타입의 모든 빈 객체를 배열로 전달

```
public class HomeController {
    ...
    private infraredRaySensor[] sensors;

    @Autowired
    public void setSensors(InfraredRaySensor[] sensors) {
        this.sensors = sensors;
    }
}
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired
  - 제너릭이 적용된 컬렉션 타입 전달

```
public class HomeController {
    ...
    private List<InfraredRaySensor> sensors;

    @Autowired
    public void setSensors(List<InfraredRaySensor> sensors) {
        this.sensors = sensors;
    }
}
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Autowired 어노테이션 적용 프로퍼티의 필수 여부 지정
  - 기본적으로 의존 객체가 등록되지 않았다면 예외 발생
  - required 속성으로 필수 여부 지정 (true/false)

```
public class HomeController {
    ...
    private List<InfraredRaySensor> sensors;

    @Autowired(required=false)
    public void setSensors(List<InfraredRaySensor> sensors) {
        this.sensors = sensors;
    }
}
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- ○ @Qualifier 어노테이션을 이용한 자동 설정 제한
  - ▪ @Autowired 어노테이션은 byType 방식으로 자동 설정
    - 두 개 이상의 같은 타입 객체가 있는 경우 예외 발생
    - @Qualifier로 이름 지정하여 해결

```
public class HomeController {
    ...
    @Autowired
    @Qualifier("main")
    private Recorder recorder;
    ...
}
```

설정하고자 하는 빈 객체의 <qualifier> 태그 값 지정

```
<bean id="recorder" class="com.lecture.spring.homecontrol.Recorder">
    <qualifier value="main" />
</bean>
```

어노테이션 기반 설정

❖ @Autowired 어노테이션을 이용한 자동 설정

- @Qualifier 어노테이션을 이용한 자동 설정 제한
  - 메서드가 2개 이상의 파라미터를 가지는 경우
    - 파라미터 앞에 기술

```
@Autowired
public void prepare(AlarmDevice alarmDevice,
                    @Qualifier("main") Viewer viewer) {
    this.alarmDevice = alarmDevice;
    this.viewer = viewer;
}
```

어노테이션 기반 설정

❖ @Resource 어노테이션을 이용한 프로퍼티 설정

- @Resource
  - javax.annotation 패키지에서 정의
  - 의존하는 빈 객체를 전달할 때 사용
    - name 속성에 자동으로 연결한 빈 객체의 이름(식별자)를 지정

```
import javax.annotation.Resource;

public class HomeController {

    @Resource(name = "camera1")
    private Camera camera1;

    private Camera camera4;

    @Resource(name = "camera4")
    public void setCamera4(Camera camera4) {
        this.camera4 = camera4;
    }
}
```

지정한 camera1이 등록되어 있지 않다면 NoSuchBeanDefinitionException 예외 발생

어노테이션 기반 설정

❖ @PostConstruct 어노테이션 및 @PreDestroy 어노테이션과 라이프 사이클

- @PostConstruct 어노테이션 및 @PreDestroy 어노테이션
  - javax.annotation 패키지에서 정의
  - 라이프 사이클의 초기화 및 제거 과정을 제공

- @PostConstruct
  - 의존하는 객체를 설정한 이후에 초기화 작업을 수행할 메서드에 적용
  - 스프링 설정파일의 init-method 속성 명시 메시드 호출 시점

- @PreDestory
  - 컨테이너에서 객체를 제거하기 전에 호출
  - 스프링 설정파일의 destroy-method 속성 명시 메시드 호출 시점

어노테이션 기반 설정

❖ @PostConstruct 어노테이션 및 @PreDestroy 어노테이션과 라이프 사이클

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class HomeController {
    @PostConstruct
    public void init() {
        // 초기화 작업
        ...
    }

    @PreDestroy
    public void close() {
        // 자원 반환 등 종료 처리
        …
    }
}
```

빈 객체 스캔

❖ 어노테이션을 이용한 자동 스캔

- 자동 스캔
  - 클래스 패스에 위치한 클래스를 검색하여 특정한 어노테이션이 붙은 클래스를 빈으로 자동 등록하는 기능
  - XML 설정 파일에 빈 정보 추가할 필요 없어짐
  - @Controller, @Component, @Service 등

```
import org.springframework.stereotype.Component;
...
@Component
public class HomeController {

    @Resource(name = "camera1")
    private Camera camera1;
    ...
    @Autowired
    public void prepare(AlarmDevice alarmDevice, Viewer viewer) {
        this.alarmDevice = alarmDevice;
        this.viewer = viewer;
    }
    ...
}
```

빈 객체 스캔

❖ 어노테이션을 이용한 자동 스캔

- @Component 어노테이션 적용
  - 스프링 설정 파일에서 <context:component-scan> 태그 추가
    - base-package 속성에 지정한 패키지에서 @Component 어노테이션이 적용된 클래스를 검색하여 빈으로 등록

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="com.lecture.spring.homenetwork"/>
    ...
</beans>
```

빈 객체 스캔

❖ 자동 검색된 빈의 이름과 범위

- 자동 검색 빈의 이름
  - 클래스의 (첫 글자를 소문자로 바꾼)이름을 빈 이름으로 사용
  - HomeController --> homeController

```
@Component
public class HomeController {
    ...
}
```

```
ApplicationContext context = ... ;
HomeController controller =
(SystemMonitor)context.getBean("homeController");
```

  - 특정 이름 명시
    - 어노테이션 속성에 빈의 이름 지정

```
@Component("homeControl")
public class HomeController {
    ...
}
```

빈 객체 스캔

❖ 자동 검색된 빈의 이름과 범위

- 자동 검색 빈의 범위 설정
  - 디폴트는 singleton
  - 범위 지정 : @Scope 어노테이션으로 지정

```
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("prototype")
public class HomeController {

    ...
}
```

빈 객체 스캔

❖ 자동 검색된 빈의 이름과 범위

- 자동 검색 빈의 범위 설정
  - <aop:scoped-proxy> 태그와 동일한 프록시 객체를 생성하고자 하면
    - proxMode 속성값으로 ScopedProxyMode 열거값을 할당

```
import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode ;
import org.springframework.stereotype.Component;

@Component
@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET_CLASS)
public class HomeController {

   ...
}
```

  - ScopedProxyMode 열거값
    - NO – 프록시를 생성하지 않음
    - INTERFACES – 인터페이스에 대한 프록시 생성
    - TARGET_CLASS – 클래스에 대해 프록시 생성
    - DEFAULT – 기본값. NO와 동일

빈 객체 스캔

❖ 자동 검색된 빈의 이름과 범위

- ㅇ 자동 검색 빈의 범위 설정
  - ▪ <context:component-scan> 태그의 scoped-proxy 속성
    - 기본적으로 프록시 객체를 생성할지 여부 지정
    - no, interfaces, targetClass 값 중 하나

```
<context:component-scan base-package="com.lecture.spring.homenetwork"
      scoped-proxy="no">
```

빈 객체 스캔

❖ 스캔 대상 클래스 범위 지정하기

- <context:include-filter>
  - 자동 스캔 대상에 포함시킬 클래스 명시
- <context:exclude-filter>
  - 자동 스캔 대상에 포함시키지 않을 클래스 명시

```
<context:component-scan base-package="spring.chap04">
    <context:include-filter type="regex"
                            expression=".*HibernateRepository"/>
    <context:exclude-filter type="aspectj"
                            expression="..*IBatisRepository"/>
</context:component-scan>
```

빈 객체 스캔

❖ 스캔 대상 클래스 범위 지정하기

- 않을 클래스 명시

type 속성	설명
annotation	클래스에 지정한 어노테이션이 적용됐는지의 여부. expression 속성에는 org.example.SomeAnnotation과 같이 어노테이션 이름을 입력한다.
assingable	클래스에 지정한 타입으로 할당 가능한지의 여부. expression 속성에는 org.example.SomeClass와 같이 타입 이름을 입력한다.
regex	클래스 이름이 정규 표현식에 매칭되는 지의 여부. expression 속성에는 org\.example\.Default*와 같이 정규표현식을 입력한다.
aspectj	클래스 이름이 AspectJ 표현식에 매칭되는 지의 여부. expression 속성에는 rog.example..*Service+와 같이 정규표현식을 입력한다.