05강 빈 라이프사이클과 범위

컨테이너의 초기화 및 종료

- 스프링 컨테이너는 초기화와 종료라는 라이프사이클을 가진다.

- 컨테이너의 초기화 => 빈 객체의 생성과 의존 객체 주입 및 초기화
- 컨테이너의 종료 => 빈 객체의 소멸

```
ApplicationContext ctx= new GenericXmlApplicationContext("classpath:appCtx.xml");
```

컨테이너 초기화

```
Greeter g = ctx.getBean("greeter", Greeter.class);
```

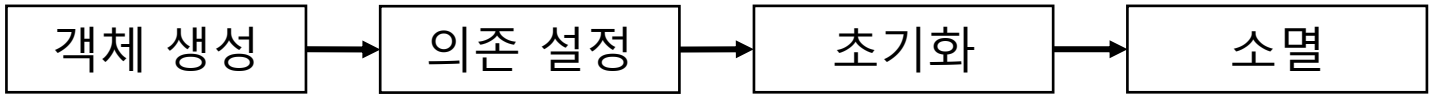
컨테이너 사용

```
ctx.close();
```

컨테이너 종료

빈 객체의 라이프 사이클

- 스프링 컨테이너는 빈 객체의 라이프 사이클을 관리한다.



- 스프링 컨테이너가 초기화 할 때
  - 빈 객체 생성
  - 의존 설정
  - 빈 객체의 초기화 수행

- 스프링 컨테이너가 종료 될 때
  - 빈 객체 소멸

빈 객체의 라이프 사이클

- 빈 객체가 생성되고 소멸하기 위해서 빈 객체의 지정한 메서드를 호출하는데 그 내용을 스프링에서는 다음 두 인터페이스에 정의하고 있다.

빈 생성

```
public interface InitializingBean{
    void afterPropertiesSet() throws Exception;
}
```

빈 소멸

```
public interface DisposableBean{
    void destroy() throws Exception;
}
```

빈 객체의 라이프 사이클

- 빈 객체가 생성되고 소멸하기 위해서 빈 객체의 지정한 메서드를 호출하는데 그 내용을 스프링에서는 다음 두 인터페이스에 정의하고 있다.

```java
public class Client implements InitializingBean, DisposableBean{
    //   빈 생성시 사용할 메소드, 빈 소멸시 사용할 메소드
    private String host;

    public void setHost(String host) {
        this.host = host;
        System.out.println("Client.setHost() 실행");
    }

    public void send() {
        System.out.println("Client.send() to "+host);
    }

    @Override
    public void destroy() throws Exception {
        //빈이 소멸할때 자동으로 실행할 메소드
        System.out.println("Client.destroy() 실행");
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        //  빈 생성시 초기화할때 자동으로 실행할 메소드
        System.out.println("Client.afterPropertiesSet() 실행");
    }

}
```

```xml
<bean id="client" class="spring.bean.Client">
    <property name="host" value="서버"/>
</bean>
```

```java
public class Main01 {

    public static void main(String[] args) {
        GenericXmlApplicationContext ctx =
                new GenericXmlApplicationContext("classpath:ctxConf.xml");
        // 스프링 컨테이너 초기화 => 빈 객체 생성->빈 객체 의존 주입 -> 빈 객체 초기화

        Client client = ctx.getBean("client",Client.class);
        // 만들어진 빈 객체를 사용하기 위해서 불러오는 과정

        client.send();
        // 빈 객체를 사용함

        ctx.close();
        // 컨테이너 종료 =>  내부에 생성된 빈 객체가 소멸
    }
```

빈 객체의 라이프 사이클

- 상황에 따라 위 두 인터페이스를 사용할 수 없을 때가 존재한다.
- 이런 경우 스프링 설정에서 직접 메서드를 지정해 줄 수 있다.
  - 공통 클래스 (Client2)

```
public class Client2 {
    private String host;

    public void setHost(String host) {
        this.host = host;
        System.out.println("Client2.setHost() 실행");
    }

    public void send() {
        System.out.println("Client2.send() to "+host);
    }

    public void connect() throws Exception{// 빈 객체 초기화시 실행할 메소드
        System.out.println("Client2.connect() 실행");
    }

    public void close() throws Exception{ // 빈 객체 소멸시 실행할 메소드
        System.out.println("Clients.close() 실행");
    }
```

빈 객체의 라이프 사이클

- 상황에 따라 위 두 인터페이스를 사용할 수 없을 때가 존재한다.
- 이런 경우 스프링 설정에서 직접 메서드를 지정해 줄 수 있다.
  - XML 설정 예시

```
    <bean id="client2" class="spring.bean.Client2"
        init-method="connect"       destroy-method="close">
<!--      빈 객체를 초기화시 실행할 메소드 ,    빈 객체를 소멸시 실행할 메소드    -->
        <property name="host" value="서버2"/>
    </bean>
```

빈 객체의 라이프 사이클

- 상황에 따라 위 두 인터페이스를 사용할 수 없을 때가 존재한다.
- 이런 경우 스프링 설정에서 직접 메서드를 지정해 줄 수 있다.
  - Java 설정 예시

```
public class JavaConfig {

    @Bean(initMethod="connect",destroyMethod="close")
    public Client2 client2() {
        Client2 client2 = new Client2();
        client2.setHost("자바서버");
        return client2;
    }
}
```

빈 객체의 라이프 사이클

- 상황에 따라 위 두 인터페이스를 사용할 수 없을 때가 존재한다.
- 이런 경우 스프링 설정에서 직접 메서드를 지정해 줄 수 있다.
  - main

```
public static void main(String[] args) {
    useXML();
    System.out.println("========================================");
    useJava();
}
```

```
private static void useXML() {
    System.out.println("=====  XML 설정 파일 =====");
    GenericXmlApplicationContext ctx =
            new GenericXmlApplicationContext("classpath:ctxConf2.xml");
    //스프링 컨테이너 초기화 => 빈 객체 생성 -> 빈 객체 의존 주입 -> 빈 객체 초기화
    //                                                  초기화시 사용되는 메소드 => xml설정파일에서 결정
    //                                                  init-method="사용할 메소드이름"
    Client2 client2 = ctx.getBean("client2",Client2.class);

    client2.send();

    ctx.close();
// 스프링 컨테이너 종료 => 내장된 빈 객체 소멸
    //                  소멸될때 사용되는 메소드 => xml설정 파일에서 결정
    //                  destroy-method="사용할 메소드 이름"

}
```

빈 객체의 라이프 사이클

- 상황에 따라 위 두 인터페이스를 사용할 수 없을 때가 존재한다.
- 이런 경우 스프링 설정에서 직접 메서드를 지정해 줄 수 있다.
  - main

```
public static void main(String[] args) {
    useXML();
    System.out.println("========================================");
    useJava();
}
```

```
private static void useJava() {
    System.out.println("=====  Java 설정 파일 =====");
    AnnotationConfigApplicationContext ctx =
            new AnnotationConfigApplicationContext(JavaConfig.class);
    //스프링 컨테이너 초기화 => 빈 객체 생성 -> 빈 객체 의존 주입 -> 빈 객체 초기화
    //                                                  초기화시 사용되는 메소드 => xml설정파일에서 결정
    //                                                  init-method="사용할 메소드이름"
    Client2 client2 = ctx.getBean("client2",Client2.class);

    client2.send();

    ctx.close();
// 스프링 컨테이너 종료 => 내장된 빈 객체 소멸
    //                  소멸될때 사용되는 메소드 => xml설정 파일에서 결정
    //                  destroy-method="사용할 메소드 이름"

}
```

객체의 범위

- 앞서 살펴본 바로 스프링 컨테이너의 빈 객체는 동일한 이름의 경우 싱글톤 범위를 갖는다고 배웠다.

- 사용 빈도가 낮지만 예외적으로 빈 객체를 프로토 타입으로 설정할 수 있는데 이 경우 일반적인 라이프 사이클을 따르지 않는다.

  -> 스프링 컨테이너에 의해 빈 객체를 생성하고 의존 설정후 초기화까지 수행하지만 컨테이너가 종료한다고 소멸 메서드를 실행하지는 않는다. => 직접 소멸 처리를 해야한다.

- 빈 객체의 범위를 프로토타입으로 지정하면 매번 빈 객체를 구할 때마다 새로운 객체를 생성해 낸다.

객체의 범위

- 프로토타입 범위를 지정하는 예제 3 - main

```
public static void main(String[] args) {
    useXML();
    System.out.println("========================================");
    useJava();
}
```

객체의 범위

- 프로토타입 범위를 지정하는 예제 3 - xml

```xml
<bean id="client" class="spring.bean.Client" scope="prototype">
    <property name="host" value="프로토서버"/>
</bean>
```

```java
private static void useXML() {
    GenericXmlApplicationContext ctx =
            new GenericXmlApplicationContext("classpath:ctxConfPrototype.xml");

    Client client1 = ctx.getBean("client",Client.class);
    Client client2 = ctx.getBean("client",Client.class);

    System.out.println("XML 설정파일을 이용한 프로토타입 예제");
    System.out.println("client1==client2 => "+(client1==client2));

}
```

객체의 범위

- 프로토타입 범위를 지정하는 예제 3 - xml

```java
@Configuration
public class JavaConfigPrototype {

    @Bean
    @Scope("prototype")
    public Client client() {
        Client client = new Client();
        client.setHost("프로토서버2");
        return client;
    }
}
```

```java
private static void useJava() {
    AnnotationConfigApplicationContext ctx =
            new AnnotationConfigApplicationContext(JavaConfigPrototype.class);

    Client client1 = ctx.getBean("client",Client.class);
    Client client2 = ctx.getBean("client",Client.class);

    System.out.println("Java 설정파일을 이용한 프로토타입 예제");
    System.out.println("client1==client2 => "+(client1==client2));
}
```