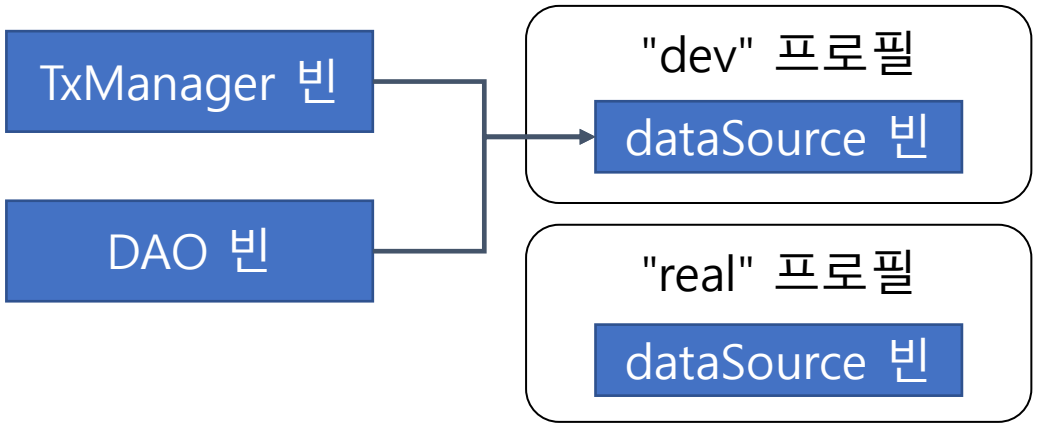
14강 프로필과 프로퍼티 파일

프로필

- 개발이 진행되는 동안에는 실제 서비스를 목적으로 운영중인 DB를 이용할 수 없다.
- 그래서 실제 서비스가 진행되는 웹서버와 DB서버는 별도로 두고 개발이 진행될 때 사용하는 웹서버와 DB서버를 사용하는 경우가 흔하다
- 그러나 이럴 경우 서비스 환경에 맞는 JDBC 연결 정보를 활용해야 하는데 단순히 설정파일을 변경하고 배포하는 방법은 번거롭고 실수할 가능성이 있다.

- 처음부터 실 서비스용, 개발환경 용 설정을 따로 구분해서 그때 그때 변경해가면서 적절한 설정을 사용하도록 하면 되는데 스프링에서 제공하는 이런 기능을 프로필이라고 한다.



프로필 - xml에서 프로필 사용하기

- XML설정에서 프로필을 사용하려면 <beans>태그의 profile 속성을 이용해서 프로필 이름을 정해주면 된다.
- (개발환경 - spring-db-dev.xml)

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.1.xsd"
        profile="dev">

    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
        <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:XE"/>
        <property name="user" value="green"/>
        <property name="password" value="1234"/>
        <property name="maxPoolSize" value="30"/>
    </bean>

</beans>
```

프로필 - xml에서 프로필 사용하기

- 프로필이 서로 다른 xml에서 같은 이름의 빈을 사용하고 있어도 아무런 문제가 없다.
- (배포 환경 - spring-db-real.xml)

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.1.xsd"
        profile="real">

    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
        <property name="jdbcUrl" value="jdbc:oracle:thin:@db.interstander.com:41521:XE"/>
        <property name="user" value="greenJSP"/>
        <property name="password" value="jsp1234"/>
        <property name="maxPoolSize" value="30"/>
    </bean>
</beans>
```

프로필 - xml에서 프로필 사용하기

- 다만 프로필 선택시 설정 정보를 불러오기전 사용할 프로필을 선택해야한다.(setActiveProfile())
- 그런 다음 설정 정보를 불러와서(load())
- 컨테이너를 초기화 해야한다.(refresh())

- 위 순서를 지키지 않으면 설정이 지정되지 않은 상태에서 빈을 불러와서 예외가 발생하게 된다.

- 또한 두개 이상의 프로필을 불러오는 경우는 다음과 같이 ,로 구분해서 불러오면 된다.

```
context.getEnvironment().setActiveProfiles("dev","oracle");
```

프로필 - xml에서 프로필 사용하기

- Main01.java에 테스트 코드를 작성한다.(개발용,실 배포용 DB서버에는 테이블이 존재해야한다.)

```
public class Main01 {
    public static void main(String[] args) {
        GenericXmlApplicationContext ctx=
                new GenericXmlApplicationContext();
        ctx.getEnvironment().setActiveProfiles("real");
        ctx.load("classpath:spring-member.xml",
                "classpath:spring-ds-dev.xml",
                "classpath:spring-ds-real.xml");
        ctx.refresh();//재 초기화

        MemberDao dao = ctx.getBean("memberDao",MemberDao.class);

        for(Member m:dao.selectAll()) {
            System.out.println(m.getName());
        }

        ctx.close();
    }
```

프로필 - xml에서 프로필 사용하기

- 프로필 별로 별도의 파일을 만들어서 사용해도 되지만 중첩 <beans>태그를 두어 하나의 파일에 두개 이상의 프로필을 넣어서 사용할 수 있다.

```
<beans profile="dev">
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="user" value="greenJSP" />
    <property name="password" value="jsp1234" />
    </bean>
</beans>

<beans profile="real">
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@db.interstander.com:41521:xe"/>
    <property name="user" value="greenJSP" />
    <property name="password" value="jsp1234" />
    </bean>
</beans>
```

프로필 - Java에서 프로필 사용하기

- 자바 설정파일에서 프로필을 지정하려면 @Profile 애노테이션을 이용하면 된다.

(DbDevConfig.java)

```java
@Configuration
@Profile("dev")
public class DbDevConfig {

    @Bean
    public DataSource dataSource() {
        ComboPooledDataSource ds = new ComboPooledDataSource();

        try {
            ds.setDriverClass("oracle.jdbc.OracleDriver");
        }catch(PropertyVetoException e) {
            throw new RuntimeException(e);
        }

        ds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
        ds.setUser("green");
        ds.setPassword("1234");

        return ds;
    }
```

프로필 - Java에서 프로필 사용하기

- 자바 설정파일에서 프로필을 지정하려면 @Profile 애노테이션을 이용하면 된다.

(DbRealConfig.java)

```java
@Configuration
@Profile("real")
public class DbRealConfig {

    @Bean
    public DataSource dataSource() {
        ComboPooledDataSource ds = new ComboPooledDataSource();

        try {
            ds.setDriverClass("oracle.jdbc.OracleDriver");
        }catch(PropertyVetoException e) {
            throw new RuntimeException(e);
        }

        ds.setJdbcUrl("jdbc:oracle:thin:@db.interstander.com:41521:xe");
        ds.setUser("greenJSP");
        ds.setPassword("jsp1234");

        return ds;
    }
```

프로필 - xml에서 프로필 사용하기

- Main02.java에 테스트 코드를 작성한다.(개발용,실 배포용 DB서버에는 테이블이 존재해야한다.)

```
public class Main01 {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx=
                new AnnotationConfigApplicationContext();

        ctx.getEnvironment().setActiveProfiles("dev");
        ctx.register(MemberConfig.class,DbDevConfig.class,DbRealConfig.class);

        ctx.refresh();

        MemberDao dao = ctx.getBean("memberDao",MemberDao.class);

        for(Member m:dao.selectAll()) {
            System.out.println(m.getName());
        }

        ctx.close();
    }
```

프로필 - Java에서 프로필 사용하기

- xml에서 중첩 <beans>태그를 활용하여 하나의 xml파일에 프로필 설정을 모은 것 처럼 java도 중첩 클래스를 이용해서 여러 프로필 설정을 하나의 파일에 모아서 사용할 수 있다 (다만 중첩클래스의 static이어야 한다)

```java
@Configuration
@Profile("dev")
public static class DataSourceDev{
    @Bean
    public DataSource dataSource() {
        ComboPooledDataSource ds = new ComboPooledDataSource();
        try {
            ds.setDriverClass("oracle.jdbc.OracleDriver");
        }catch(PropertyVetoException e) {
            throw new RuntimeException(e);
        }
        ds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
        ds.setUser("green");
        ds.setPassword("1234");
        return ds;
    }
}

@Configuration
@Profile("real")
public static class DataSourceReal{
    @Bean
    public DataSource dataSource() {
        ComboPooledDataSource ds = new ComboPooledDataSource();
        try {
            ds.setDriverClass("oracle.jdbc.OracleDriver");
        }catch(PropertyVetoException e) {
            throw new RuntimeException(e);
        }
        ds.setJdbcUrl("jdbc:oracle:thin:@db.interstander.com:41521:xe");
        ds.setUser("greenJSP");
        ds.setPassword("jsp1234");
        return ds;
    }
}
```

프로필 - 다수의 프로필 사용하기

- 하나의 프로필에 두개 이상의 설정이름을 담을 수 있다.

```
<beans profile="dev,test">
    <bean id="dataSource" class="com.m
```

```
@Configuration
@Profile("dev,test")
public static class DataSourceDual{
```

- 또한 !연산을 이용해서 해당 프로필이 활성화되지 않는 경우에만 사용됨을 나타낼 수도 있다.

```
<beans profile="!real">
    <bean id="dataSource" class="co
```

```
@Configuration
@Profile("!real")
public static class DataSourceNot{
```

프로필 - 웹 애플리케이션에서 프로필 설정하기

- 웹 애플리케이션도 시스템 프로퍼티(spring.profiles.active)나 환경변수를 사용해서 사용할 프로필을 선택할 수 있다.(XML설정이나 JAVA 설정이나 똑같이 적용하면 된다.)

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>spring.profiles.active</param-name>
        <param-value>real</param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:spring-mvc.xml
            classpath:spring-member.xml
```

프로퍼티 파일을 이용한 프로퍼티 설정

- 스프링은 외부 프로퍼티 파일을 이용해서 스프링 빈을 설정하는 방법을 제공한다.

- 예를 들어 db.properties파일이 있다고 가정한다.

```
db.driver =oracle.jdbc.OracleDriver
db.jdbcUrl = jdbc:oracle:thin:@localhost:1521:XE
db.user=green
db.password=1234
```

프로퍼티 파일을 이용한 프로퍼티 설정

- XML에서의 프로퍼티 설정 => context 모듈을 추가해야한다.

=> <context:property-placeholder>사용

```
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:property-placeholder location="classpath:message/db.properties"/>

    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <property name="driverClass" value="${db.driver}"/>
        <property name="jdbcUrl" value="${db.jdbcUrl}"/>
        <property name="user" value="${db.user}" />
        <property name="password" value="${db.password}" />
    </bean>
```

프로퍼티 파일을 이용한 프로퍼티 설정

- java에서의 프로퍼티 설정

=> PropertySourcesPlaceholderConfigurer과 @Value 애노테이션을 사용

단 PropertySourcesPlaceholderConfigurer 클래스가 특수 목적용이므로 statice 메서드로 만들어야한다.

프로퍼티 파일을 이용한 프로퍼티 설정

- java에서의 프로퍼티 설정

```java
@Configuration
public class DbPropertyConfig {
    @Value("${db.driver}")
    private String driver;
    @Value("${db.jdbcUrl}")
    private String jdbcUrl;
    @Value("${db.user}")
    private String user;
    @Value("${db.password}")
    private String passwrod;

  @Bean
    public static PropertySourcesPlaceholderConfigurer properties() {
        PropertySourcesPlaceholderConfigurer configurer =
                new PropertySourcesPlaceholderConfigurer();
        configurer.setLocation(new ClassPathResource("message/db.properties"));
        return configurer;
    }

    @Bean
    public DataSource dataSource() {
        ComboPooledDataSource ds = new ComboPooledDataSource();

        try {
            ds.setDriverClass(driver);
        }catch(PropertyVetoException e) {
            throw new RuntimeException(e);
        }

        ds.setJdbcUrl(jdbcUrl);
        ds.setUser(user);
        ds.setPassword(passwrod);

        return ds;
    }
```