第三讲要点

一、注意要点

1. 引用外库的时候，从网站上复制粘贴
2. 如果再配置文件中显示为版本号为红色，是因为本地没有的原因，点击Maven的reload按钮，下载到本地

二、Spring IOC

要遵守开闭原则，即对修改关闭，但是对扩展开放

那些可能会被更改的地方，不要写死，而是以调用接口的方法，在接口层面依赖，做到松耦合

IOC，可以帮你实现对象注入

每一个对象称之为Beans，虽然俩个类之间没有直接依赖关系，但是在创建时，IOC可以用依赖注入的方法来实现Beans的关联。

三、通过配置文件注入

配置文件注入的方法调用是从函数名来确定的，和字段名没有关系

Eg：

```
felid BookDao A;
setA (BookDao b){
    this A = b ;
}
```

```
<bean id = "bookDao" class = "xxx" />
<bean id = "bookService" class = "xxx">
    <property name = "a" ref="bookDao">
</bean>
```

在配置文件中，根据"a"来找到类里面的setA()函数,不做大小写的区分，而 ref 根据id来注入

Eg:

```
//创建IoC容器
ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
//获取bean（根据bean配置id获取）
BookService bookService = (BookService) ctx.getBean("bookService");
bookService.save();
```

依赖于接口，而不是直接依赖于类，通过配置文件的装配，实现依赖

这样要用到外部的内库或类时，可以直接注入（Spring的粘合度很好的原因）

四、生命周期

scope

singleton (默认)	**prototype**
只创建单例	不同线程创建不同的对象，在getBean函数调用时才创建对象
适用无状态的对象，即运行中无修改的值	适用于状态变化的对象，保证线程安全不冲突

五、四种方法创建对象

```
<!--方式一：构造方法实例化bean-->
<bean id="bookDao" class="com.itheima.dao.impl.BookDaoImpl"/>
<!--方式二：使用静态工厂实例化bean-->
<bean id="orderDao" class="com.itheima.factory.OrderDaoFactory" factory-
method="getOrderDao"/>
<!--方式三：使用实例工厂实例化bean-->
<bean id="userFactory" class="com.itheima.factory.UserDaoFactory"/>
<bean id="userDao" factory-method="getUserDao"factory-bean="userFactory"/>
<!--方式四：使用FactoryBean实例化bean-->
<bean id="userDao2" class="com.itheima.factory.UserDaoFactoryBean"/>
```

1. 方法一，直接通过调用该类的构造函数

   如果构造函数有参数时，也可以通过配置文件来输入

   这里建议使用 ***index*** 来传参，因为name在编译时不会保留，所以容易出错

   ```
   <!--标准书写-->
    <bean id="bookDao" class="com.itheima.dao.impl.BookDaoImpl">
        <!--根据构造方法参数名称注入-->
        <constructor-arg name="connectionNum" value="10"/>
        <constructor-arg name="databaseName" value="mysql"/>
    </bean>
    <bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>

    <bean id="bookService" class="com.itheima.service.impl.BookServiceImpl">
        <constructor-arg name="userDao" ref="userDao"/>
        <constructor-arg name="bookDao" ref="bookDao"/>
    </bean>

    <!--解决形参名称的问题，与形参名不耦合--!>
    <bean id="bookDao" class="com.itheima.dao.impl.BookDaoImpl">
        <!--根据构造方法参数类型注入--!>
        <constructor-arg type="int" value="10"/>
        <constructor-arg type="java.lang.String" value="mysql"/>
   ```

```
</bean>
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>

<bean id="bookService" class="com.itheima.service.impl.BookServiceImpl">
    <constructor-arg name="userDao" ref="userDao"/>
    <constructor-arg name="bookDao" ref="bookDao"/>
</bean>

<!--解决参数类型重复问题，使用位置解决参数匹配-->
<bean id="bookDao" class="com.itheima.dao.impl.BookDaoImpl">
    <!--根据构造方法参数位置注入-->
    <constructor-arg index="0" value="mysql"/>
    <constructor-arg index="1" value="100"/>
</bean>
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>
```

2. 方法二，调用工厂中的静态函数，把返回值放入Bean中

3. 方法三，先实例化工厂类，再通过实例调用创建对象的函数

4. 方法四，继承FactoryBean的接口，代替原始工厂中创建对象的方

```
public class UserDaoFactoryBean implements FactoryBean<UserDao> {
    //代替原始实例工厂中创建对象的方法
    public UserDao getObject() throws Exception {
        return new UserDaoImpl();
    }
    public Class<?> getObjectType() {
        return UserDao.class;
    }
}
```

[PAT] 这里注意以下什么时候用构造函数，什么时候用注入。在是必须的属性时，调用构造，如果是可有可无的属性，用属性注入来做。

六、初始化和销毁

可以在类中些init和destroy函数，分别在创建后，关闭前调用

destroy函数对于singleton性质的变量时在整个程序close后，对于prototype是单个bean结束后调用

```
<!--init-method：设置bean初始化生命周期回调函数-->
<!--destroy-method：设置bean销毁生命周期回调函数，仅适用于单例对象-->
<bean id="bookDao" class="com.itheima.dao.impl.BookDaoImpl" init-method="init"
destroy-method="destory"/>
```

也可以继承接口，直接重写初始化和销毁的接口(InitializingBean,DisposableBean)，这种就不需要在配置文件中说明

```
public class BookServiceImpl implements BookService, InitializingBean,
DisposableBean {
    private BookDao bookDao;

    public void setBookDao(BookDao bookDao) {
        System.out.println("set .....");
        this.bookDao = bookDao;
    }

    @Override
    public void destroy() throws Exception {
        System.out.println("service destroy");
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("service init");
    }
}
```

七、集合

如果传的参数是集合，用相应的属性来做

```
<!--数组注入-->
<property name="array">
    <array>
        <value>100</value>
        <value>200</value>
    </array>
</property>

<!--map集合注入-->
<property name="map">
    <map>
        <entry key="country" value="china"/>
        <entry key="province" value="henan"/>
                <entry key="city" value="kaifeng"/>
    </map>
</property>

<!--Properties注入-->
<property name="properties">
    <props>
        <prop key="country">china</prop>
        <prop key="province">henan</prop>
        <prop key="city">kaifeng</prop>
```

```
    </props>
</property>

<!--其他类似的-->
<list>  </list>
<set>  </set>
```

如果是对象数组，则用`<ref></ref>`来引入对象

八、Autowired

自动编织，在属性上加上 `autowire`

1. byType

   根据类型找set方法中参数所匹配类型的bean

   如果找不到就不注入，但是如果有俩个ref的bean，就会报错

2. byName

   根据id俩=来找，更加准确

九、getBean

可以通过getBean来获取对象

1. 通过id

   返回的的是一个object，因此需要强制转型

   ```
   BookDao bookDao = (BookDao) ctx.getBean("bookDao")
   ```

2. 通过id和类型

   指定返回值的类型，泛型

   ```
   BookDao bookDao = ctx.getBean("bookDao" ,BookDao.class)
   ```

3. 通过类型

   必须保证只有一个

   ```
   BookDao bookDao = ctx.getBean(BookDao.class)
   ```

十、Spring Annotations

加上注解，自动识别

先创建一个Spring配置类，说明要扫描的范,在该类前加上`@Configurationn`（类）的注解，并在`@ComponentScan({" "})`（类）里填入希望扫描的范围

在再实体类上加上`@Component`（类）的注解，可以是用`@Autowired`来自动根据类型注入，也可以是在`@Component(" ")`（类）的括号内指定id和name

最后在AnnotationConfigApplicationContext加载Spring配置类初始化Spring容器

其他的一些注解

`@Scope("singleton")`（类） 生命周期

`@PostConstruct`（函数） 初始化

`@PreDestroy`（函数） 销毁前

`@Bean(" ")`（函数） 用工厂方法创建Bean,等同于使用实例工厂实例化bean(方法三)

`@Autowired` 可以标注在字段，属性，构造函数上，其中**私有的**也可以标注，缺省情况是byType，如果要byName要在注解后面写括号说明

`@Properties({ })`（类） 可以通过属性文件中的键值对来配置，例如：在实体类的某一字段上标注属性`@Value("${name}")`，则编译时会在属性文件中寻找name的值，注入，适合于***需要可配置的常量***