2、Spring核心容器

2.1 Spring IoC 概述

IoC（控制反转）是Spring框架的基础，也是Spring框架的核心理念。

控制反转（Inversion of Control，IoC）是一个比较抽象的概念，是Spring框架的核心，用来消减计算机程序的耦合问题。依赖注入（Dependency Injection，DI）是IoC的另外一种说法，只是从不同的角度，描述相同的概念。

当某个Java对象需要调用另一个Java对象时，在传统编程模式下，调用者通常会采用“new 被调用者”的代码方式来创建对象，这种方式会增加调用者与被调用者之间的耦合性，不利于代码后期维护和升级。当Spring出现之后，对象的实例化不再由调用者来创建，而是由Spring容器来创建。Spring容器会负责控制对象之间的依赖关系，而不是由调用者的程序代码直接控制，这样控制权由调用者转移到了Spring容器，控制权发生了反转，这就是Spring 的控制反转。

从Spring容器角度来看，Spring容器负责将被依赖对象赋值给调用者的成员变量，相当于为调用者注入它所依赖的实例，这就是Spring的依赖注入。控制反转是一种通过描述（在Spring中可以是XML或注解）并通过第三方去产生或获取特定对象的方式。在***Spring中实现控制反转的是IoC容器，其实现方法是依赖注入***。

Spring的主要功能是通过其核心容器实现的。Spring容器会负责控制程序之间的关系，而不是由程序代码直接控制。Spring为我们提供了两种方式去构建核心容器，分别为BeanFactory和ApplicationContext，本节将对这两种核心容器进行简单介绍。

2.1.1 Spring容器的构建

2.1.1.1 通过BeanFactory构建Spring容器

> 使用BeanFactory加载Spring配置文件的方法在实际开发中并不多见，同学们了解即可。

BeanFactory由org.springframework.beans.factory.BeanFactory接口定义，它提供了完整的IoC服务支持，是一个管理Bean的工厂，主要负责实例化各种Bean。

创建BeanFactory实例时，需要提供XML文件的绝对路径。

```
BeanFactory beanFactory = new XmlBeanFactory(new
FileSystemResource("F://aaa/ch1/src/beans.xml"))
```

2.1.1.2 通过ApplicationContext构建Spring容器

通过ApplicationContext实例化Spring容器的方法有以下三种：

2.1.1.2.1 通过ClassPathXmlApplicationContext创建

ClassPathXmlApplicationContext将从类路径classPath目录（src根目录）寻找指定的XML配置文件

```
ApplicationContext ac = new ClassPathXmlApplicationContext("beans.xml")
```

2.1.1.2.2 通过FileSystemXmlApplicationContext创建

FileSystemXmlApplicationContext将从指定文件的绝对路径中寻找XML配置文件，找到并装载完成ApplicationContext的实例化工作。

```
ApplicationContext ac = new
FileSystemXmlApplicationContext("F://aaa/ch1/src/beans.xml")
```

> 采用绝对路径的加载方式将导致程序的灵活性变差，一般不推荐使用。

2.1.1.2.3 通过Web服务器实例化ApplicationContext容器

在做Web应用开发时，Web服务器实例化ApplicationContext容器时，一般使用基于org.springframework.web.context.ContextLoaderListener的实现方式（需要将spring-web-4.3.6.RELEASE.jar复制到WEB-INF/lib目录中），此方法只需在web.xml中添加如下代码：

```
<context-param>
    <!-- 加载src目录下的applicationContext.xml文件 -->
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:applicationContext.xml
    </param-value>
</context-param>
  <!-- 指定以ContextLoaderListener方式启动Spring容器 -->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

> 通常在Java应用开发中，常用ClassPathXmlApplicationContext来实例化Spring容器，而在Java Web开发中，通常采用Web服务器来实例化Spring容器。

> BeanFactory和ApplicationContext两种都是通过XML配置文件加载Bean 的。二者的主要区别时：如果bean的某个属性没有注入，只用BeanFactory加载后，在调用getBean()方法时会抛出异常，而ApplicationContext则在初始化时进行自检，这样有利于检查所有依赖是否注入。在实际开发中，通常使用ApplicationContext的方式构建Spring容器，Bean Factory很少使用。

2.1.2 依赖注入

2.1.2.1 依赖注入概念

依赖注入与控制反转其实是一个概念，只不过是从不同的角度来对同一个概念进行描述。

2.1.2.2 依赖注入的实现方式

在Spring中实现IoC容器的方法是依赖注入，依赖注入的作用是在使用Spring框架创建对象时，动态地将其所依赖的对象（如属性值）注入Bean组件中。Spring框架的依赖注入通常有两种实现方式：一种是构造方法注入，另一种是属性setter方法注入。

2.1.2.2.1 通过setter方法注入

> 此方法是Spring框架主流的注入方式。

指Spring容器使用setter方法注入被依赖的实例，通过调用午餐构造器实例化Bean后，调用该Bean的setter方法，即可实现基于setter方法的依赖注入。

第一步：创建dao层接口，所在包为：cn.edu.abtu.dao

```
public interface ITestDao{
    public void sayHello();
}
```

第二步：创建dao接口的实现类，所在包为：cn.edu.abtu.dao.impl

```
public class TestDaoImpl implements ITestService{
    @Override
    public void sayHello(){
        System.out.println("Hello, SSM!!!!");
    }
}
```

第三步：创建service层接口，所在包为：cn.edu.abtu.service

```
public interface ITestService{
    public void sayHello();
}
```

第四步：创建service层接口的实现类，所在包为：cn.edu.abtu.service.impl

```
public class TestServiceImpl implements ITestService{
    private TestDao testDao;
    public void setTestDao(TestDao testDao){
        this.testDao = testDao;
    }
    //省略getter方法
    @Override
    public void sayHello(){
        testDao.sayHello();
    }
}
```

第五步：将TestServiceImpl类和TestDaoImpl类托管给Spring容器，让Spring容器为其创建对象，同时调用TestServiceImpl类的setter方法完成依赖注入。在applicationContext.xml中添加如下代码

```
<bean id="testService" class="cn.edu.abtu.service.impl.TestServiceImpl">
    <!-- setter方法注入通过property子元素将testDao实例注入给TestServiceImpl类 -->
    <property name="testDao" ref="testDao"/>
</bean>
<bean id="testDao" class="cn.edu.abtu.dao.impl.TestDaoImpl">

</bean>
```

第六步：测试代码

```
ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
TestService testService = (TestService)ac.getBean("testService");
testService.sayHello();
```

2.1.2.2.2 通过构造方法注入

该方法指Spring容器通过构造方法注入被依赖的实例。基于构造方法的依赖注入通过调用带参数的构造方法来实现，每一个参数都代表这一个依赖。具体步骤如下：

第一步：创建dao层接口

略。此步与上面方法步骤一样

第二步：创建dao层接口的实现类

略。此步与上面方法步骤一样

第三步：创建service接口

略。此步与上面方法步骤一样

第四步：创建service接口的实现类

> 请同学们注意此处的注入方式和setter属性注入方式的区别

```
public class TestServiceImpl implements ITestService{
    private TestDao testDao;
    //构造方法，用于实现依赖注入的接口对象
    public TestServiceImpl(TestDao testDao){
        super();
        this.testDao = testDao;
    }

    @Override
    public void sayHello(){
        testDao.sayHello();
    }
}
```

第五步：在applicationContext.xml中配置TestServiceImpl和TestDaoImpl

```
<bean id="testService" class="cn.edu.abtu.service.impl.TestServiceImpl">
    <!-- 将testDao实例通过构造方法注入到TestServiceImpl中 -->
    <constructor-arg index="0" ref="testDao"/>
</bean>
<!-- 将TestDaoImpl配置给Spring容器，让Spring容器为其创建实例 -->
<bean id="testDao" class="cn.edu.abtu.dao.impl.TestDaoImpl">

</bean>
```

第六步：测试代码

略。此步与上面方法步骤一样

2.2 Spring Bean

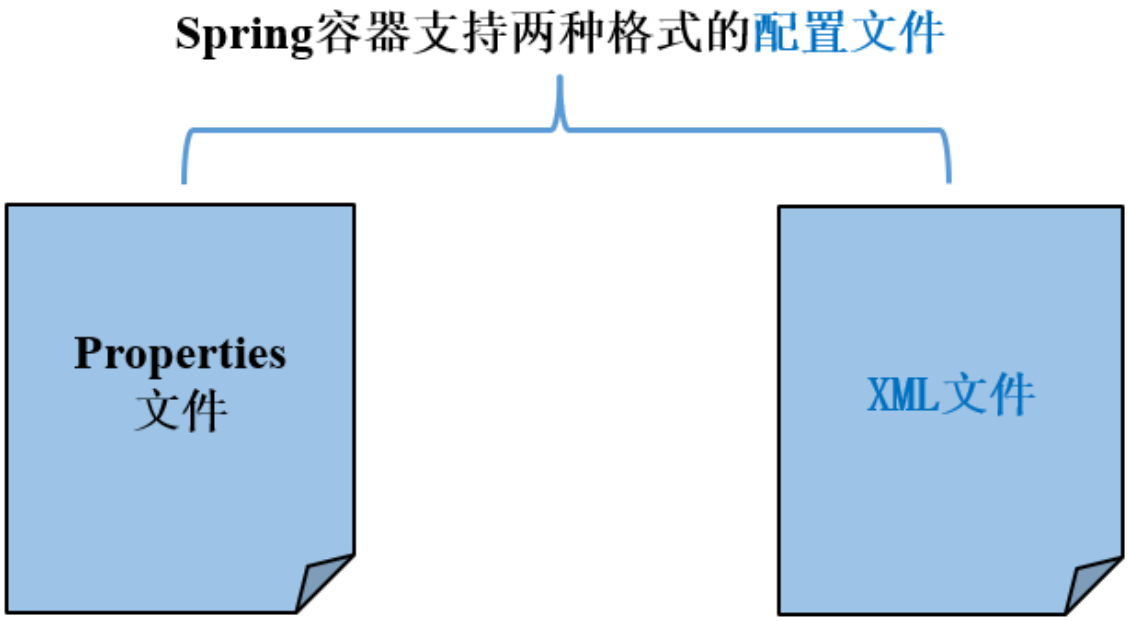
> 本章主要介绍Spring Bean的配置、实例化、作用域、生命周期以及装配方式等内容，要求大家掌握。

2.2.1 Bean的配置

如果把Spring容器看做一个大型工厂，则Spring容器中的Bean就是该工厂的产品。要想使用这个工厂生产和管理Bean，就需要在配置文件中告诉它需要哪些Bean，以及需要使用何种方式将这些Bean装配到一起。

> Bean的本质就是Java中的类，而Spring中的Bean其实就是对实体类的引用，来生产Java类对象，从而实现生产和管理Bean

在实际开发中，最常使用的是XML文件格式的配置方式，这种配置方式是通过XML文件来注册并管理Bean之间的依赖关系。



2.2.1.1 Bean的属性解释

XML配置文件的根元素是，中包含了多个子元素，每一个子元素定义了一个Bean，并描述了该Bean如何被装配到Spring容器中。关于元素的常用属性如下表所示：

属性或子元素名称	描述
id	是一个 Bean 的唯一标识符，Spring 容器对 Bean 的配置、管理都通过该属性来完成。
name	Spring 容器同样可以通过此属性对容器中的 Bean 进行配置和管理，name 属性中可以为 Bean 指定多个名称，每个名称之间用逗号或分号隔开。
class	该属性指定了 Bean 的具体实现类，它必须是一个完整的类名，使用类的全限定名。
scope	用来设定 Bean 实例的作用域，其属性值有：singleton（单例）、prototype（原型）、request、session、global Session、application 和 websocket。其默认值为 singleton。
...	...

2.2.1.2 Bean的子元素解释

Bean的子元素	子元素的描述
`<constructor-arg>`	该元素是元素的子标签，使用构造方法注入，指定构造方法的参数。该元素的index属性指定参数的序号，ref指定对BeanFactory中其它Bean的引用关系，type属性指定参数的类型，value指定参数的常量值
`<property>`	该元素是元素的子标签，用于设置一个属性。该元素的name属性指定Bean实例中相应的属性名称。value属性指定属性值，ref属性指定对BeanFactory中其它Bean 的引用关系
`<list>`	标签的子标签，用于封装List或者数组类型的依赖注入
`<map>`	标签的子标签，用于封装Map类型的依赖注入
`<set>`	标签的子标签，用于封装Set类型的依赖注入
`<entry>`	标签的子标签，用于设置一个键值对

2.2.1.3 Bean的配置代码

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans xmlns="http://www.springframework.org/schema/beans"
    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.xsd">
            <bean id="bean1" class="com.itheima.Bean1" />
            <bean name="bean2" class="com.itheima.Bean2" />
  </beans>
```

2.2.2 Bean的实例化

在面向对象的程序中，想要使用某个对象，就需要先实例化这个对象。同样，在Spring中，要想使用容器中的Bean，也需要实例化Bean。实例化Bean有三种方式，分别为构造器实例化、静态工厂方式实例化和实例工厂方式实例化。

> 最常用的是构造器实例化

2.2.2.1 构造方法实例化

在Spring框架中，Spring容器可以调用Bean对应类中无参数构造方法来实例化Bean，这种方式称为构造方法实例化。

第一步：创建Bean类

```
public class BeanClassTest{
    private String msg;
    public BeanClassTest(){
        msg = "通过构造方法实例化Bean"
    }
}
```

第二步：在applicationContext.xml中配置Bean

```
<bean id="beanClassTest" class="cn.edu.abtu.conins.BeanClassTest"/>
```

第三步：测试

```
ApplicationContext ac = new ClassPathXmlApplicationContext("beans.xml");
ac.getBean("beanClassTest")
```

2.2.2.2 静态工厂实例化

> 此种方法同学们稍做了解即可

参考教材2.2.2

2.2.2.3 实例工厂实例化

> 此种方法同学们稍做了解即可

参考教材2.2.3

2.3 Bean 的作用域

Spring容器中定义了7种作用域，由Bean的scope属性指定：

作用域名称	说明
singleton（单例）	使用 singleton 定义的 Bean 在 Spring 容器中将只有一个实例，也就是说，无论有多少个 Bean 引用它，始终将指向同一个对象。这也是 Spring 容器默认的作用域。
prototype（原型）	每次通过 Spring 容器获取的 prototype 定义的 Bean 时，容器都将创建一个新的 Bean 实例。
request	在一次 HTTP 请求中，容器会返回该 Bean 的同一个实例。对不同的 HTTP 请求则会产生一个新的 Bean，而且该 Bean 仅在当前 HTTP Request 内有效。
session	在一次 HTTP Session 中，容器会返回该 Bean 的同一个实例。对不同的 HTTP 请求则会产生一个新的 Bean，而且该 Bean 仅在当前 HTTP Session 内有效。
globalSession	在一个全局的 HTTP Session 中，容器会返回该 Bean 的同一个实例。仅在使用 portlet 上下文时有效。
application	为每个 ServletContext 对象创建一个实例。仅在 Web 相关的 ApplicationContext 中生效。
websocket	为每个 websocket 对象创建一个实例。仅在 Web 相关的 ApplicationContext 中生效。

> 在上表7种作用域中，singleton和prototype是最常用的两种作用域

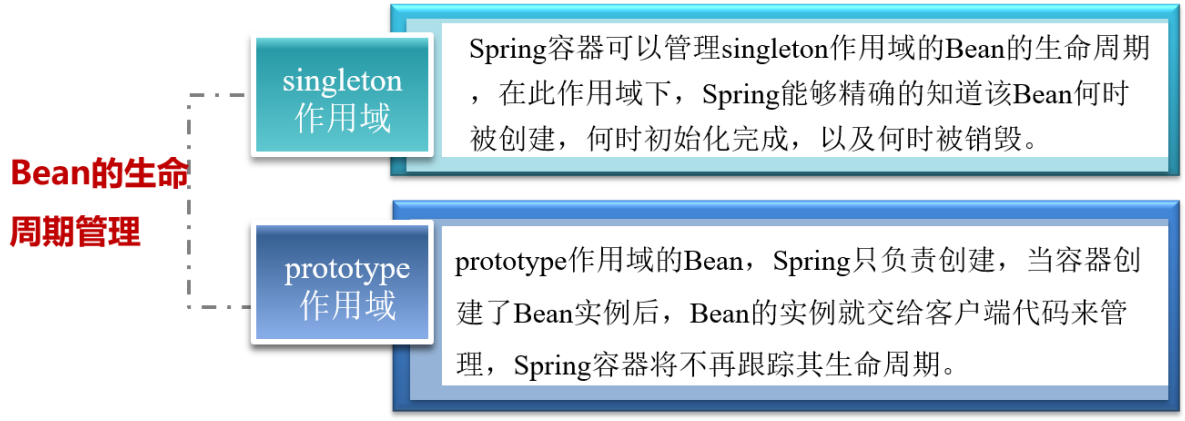
> singleton是Spring容器默认的作用域，当Bean的作用域为singleton时，Spring容器就只会存在一个共享的Bean实例。singleton作用域对于无会话状态的Bean（如Dao 组件、Service组件）来说，是最理想的选择。

> 在Spring配置文件中，可以使用元素的scope属性，将Bean的作用域定义成singleton。

```
<bean id="scope" class="com.itheima.scope.Scope" scope="singleton"/>
```

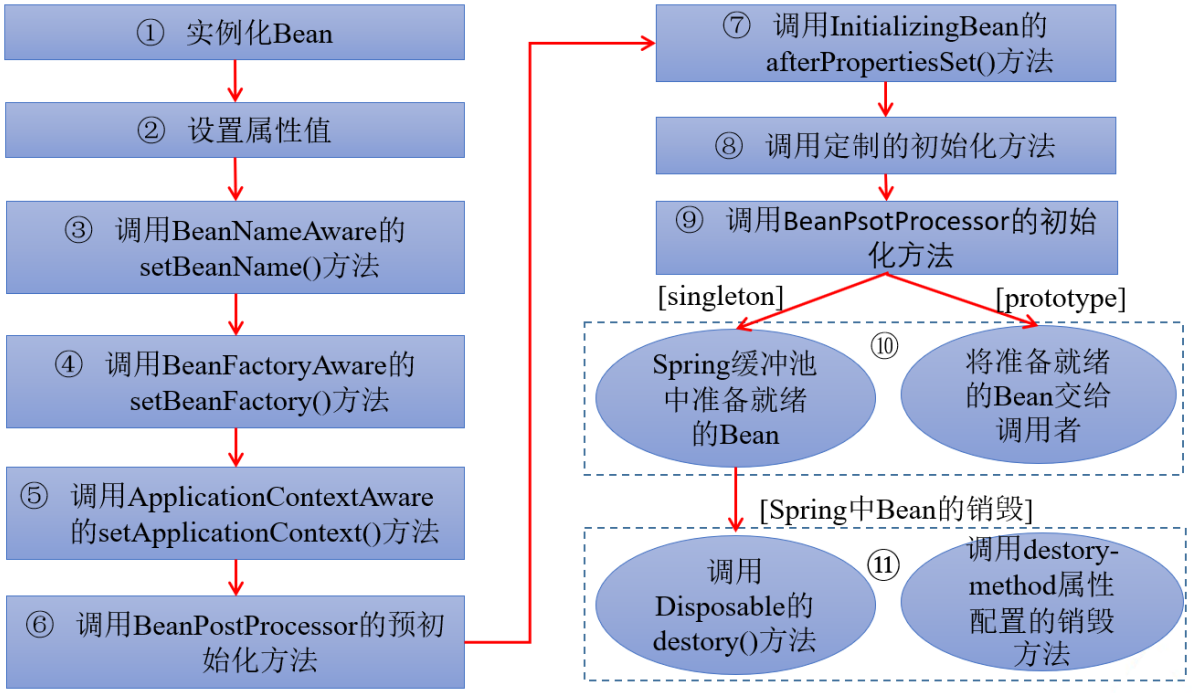
2.4 Bean的生命周期

了解Spring中Bean的生命周期的意义就在于，可以利用Bean在其存活期间的特定时刻完成一些相关操作。这种时刻可能有很多，但一般情况下，常会在Bean的postinitiation(初始化后)和predestruction（销毁前）执行一些相关操作。



2.4.1 Bean 的生命周期流程

Spring容器中Bean的生命周期流程如下图所示



具体步骤：

1. 根据Bean的配置情况，实例化一个Bean。
2. 根据Spring上下文对实例化的Bean进行依赖注入，即对Bean的属性进行初始化。
3. 如果Bean实现了BeanNameAware接口，将调用它实现的setBeanName(String beanId)方法，此处参数传递的是Spring配置文件中Bean的ID。
4. 如果Bean实现了BeanFactoryAware接口，将调用它实现的setBeanFactory()方法，此处参数传递的是当前Spring工厂实例的引用。
5. 如果Bean实现了ApplicationContextAware接口，将调用它实现的setApplicationContext(ApplicationContext)方法，此处参数传递的是Spring上下文实例的引用。
6. 如果Bean关联了BeanPostProcessor接口，将调用预初始化方法postProcessBeforeInitialization(Object obj, String s)对Bean进行操作。
7. 如果Bean实现了InitializingBean接口，将调用afterPropertiesSet()方法。
8. 如果Bean在Spring配置文件中配置了init-method属性，将自动调用其配置的初始化方法。
9. 如果Bean关联了BeanPostProcessor接口，将调用postProcessAfterInitialization(Object obj, String s)方法，由于是在Bean初始化结束时调用After方法，也可用于内存或缓存技术。
   以上工作（1至9）完成以后就可以使用该Bean，由于该Bean的作用域是singleton，所以调用的是同一个Bean实例。
10. 当Bean不再需要时，将经过销毁阶段，如果Bean实现了DisposableBean接口，将调用其实现的destroy方法将Spring中的Bean销毁。
11. 如果在配置文件中通过destroy-method属性指定了Bean的销毁方法，将调用其配置的销毁方法进行销毁。

> 在Spring中，通过实现特定的接口或者通过元素的属性设置可以对Bean的生命周期过程产生影响。开发者可以随意配置元素的属性，但不建议过多地使用Bean实现接口，因为这样将会使代码和Spring耦合度比较高

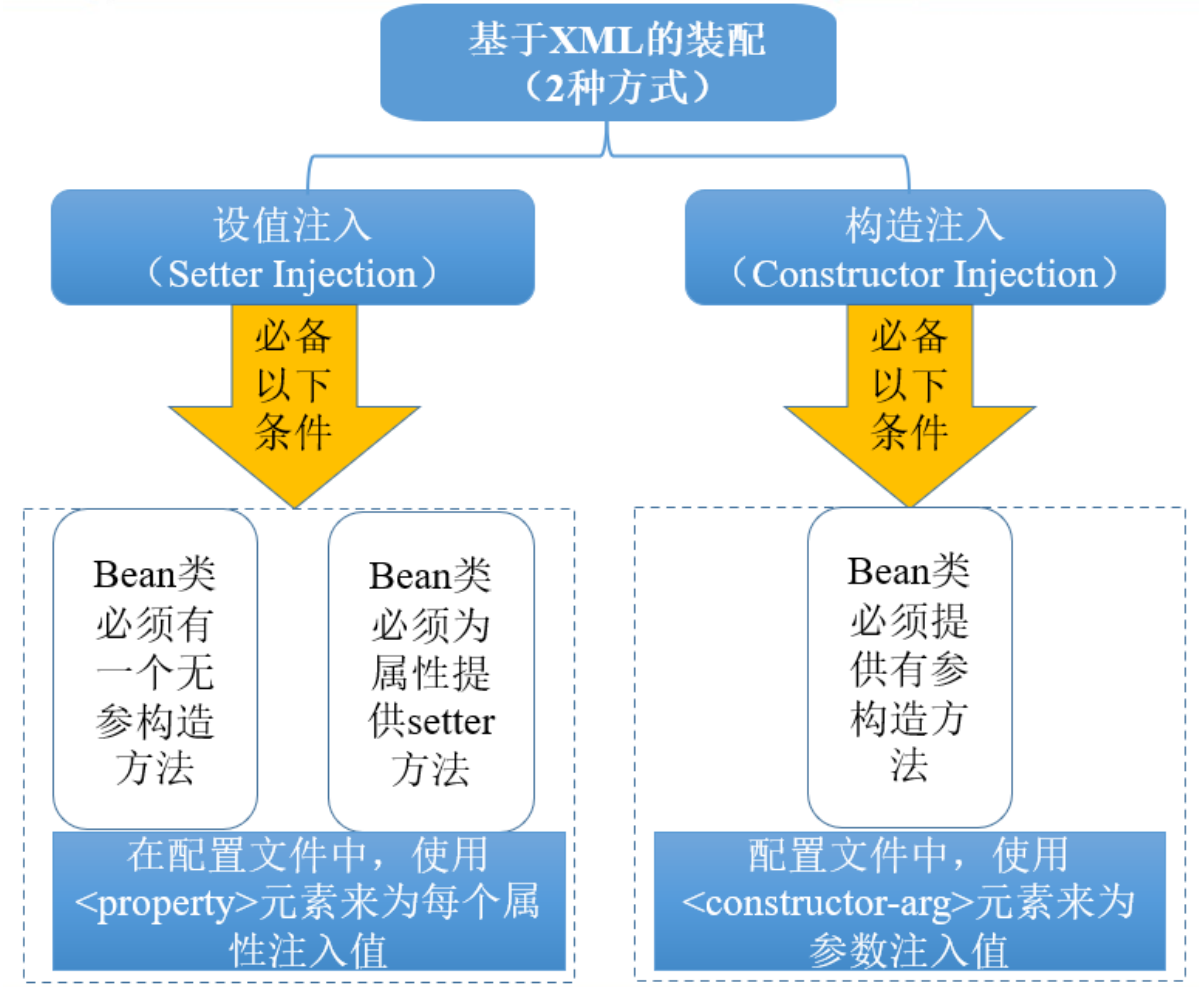
2.5 Bean的装配

Bean的装配可以理解为依赖关系注入，Bean的装配方式即Bean依赖注入的方式。Spring容器支持多种形式的Bean的装配方式，如***基于XML的装配***、***基于注解（Annotation）的装配***和***自动装配***，本节将主要讲解这三种装配方式的使用

> 其中最常用的是基于注解的装配，同学们在初学时可以采用基于XML的装配方式

2.5.1 基于XML的配置的Bean装配

> 在使用构造方法注入Bean时，Bean的实现类需要提供一个带参构造，在使用属性的setter方法注入时，Bean的实现类必须提供一个无参构造



2.5.1.1 基于XML配置的Bean装配案例

第一步：Bean类的实现

```
public User(String username, Integer password, List<String> list) {
    super();
        this.username = username;
        this.password = password;
    this.list = list;
      }
      public User() { super();}
      ......
      //省略属性setter方法
```

第二步：配置Bean

```
      <bean id="user1" class="com.itheima.assemble.User">
         <constructor-arg index="0" value="tom" />
        ...
      </bean>
      <bean id="user2" class="com.itheima.assemble.User">
       <property name="username" value="张三"  />
        ...
      </bean>
```

第三步：测试

```
    ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("beans.xml");
    System.out.println(applicationContext.getBean("user1"));
    System.out.println(applicationContext.getBean("user2"));
```

2.5.2 基于注解的Bean装配

> 在Spring框架中，尽管使用XML配置文件可以很简单装配Bean，但是如果应用中有大量的Bean需要装配，可能会导致XML配置文件过于臃肿，给后续的维护和升级带来一定的困难。为此，Spring提供了对Annotation（注解）技术的全面支持。

2.5.2.1 注解解释

注解名称	描述
@Component	用于描述Spring中的Bean，它是一个泛化的概念，可以作用在任何层次上，仅仅表示一个组件
@Repository	用于将数据访问层（DAO）的类标识为Spring中的Bean，即注解数据访问层Bean，与@Component功能相同
@Service	用于将业务层（Service）的类标识为Spring中的Bean，功能与@Component功能相同
@Controller	用于将控制层（Controller）的类标识为Spring中的Bean，功能与@Component功能相同
@Autowired	用于对Bean的属性变量、属性的setter方法及构造方法进行标注，配合对应的注解处理器完成Bean的自动配置工作
@Resource	其作用与Autowired一样。@Resource中有两个重要属性：name和type。Spring将name属性解析为Bean实例名称，type属性解析为Bean实例类型
@Qualifier	与@Autowired注解配合使用，会将默认的按Bean类型装配修改为按Bean的实例名称装配，Bean的实例名称由@Qualifier注解的参数指定

> @Resource注解默认名称来装配注入的，只有找不到与名称匹配的Bean时才会按照类型来装配注入。
>
> @Autowired默认按照Bean的类型进行装配，如果想按照Bean的名称来装配，则需要和@Qualifier注解一起使用。Bean的实例名称由@Qualifier的参数来指定。

> 上面几个注解中，虽然@Repository、@Service和 @Controller等注解的功能与@Component()相同，但为了使标注类的用途更加清晰（层次化），在实际开发中推荐使用@Repository标注数据访问层（DAO层）、使用@Service标注业务逻辑层（Service层）以及使用@Controller标注控制器层（控制层）

2.5.3 自动装配

> 此种装配方式同学稍做了解即可。