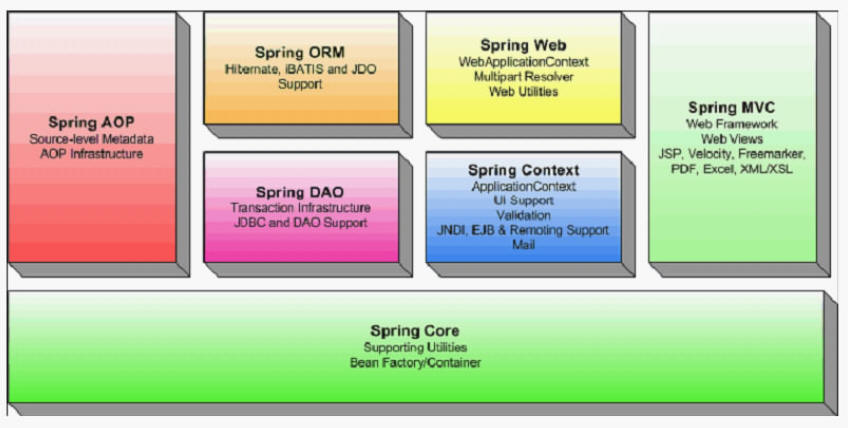
# Spring是什么

Spring就是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。    

* **核心容器：**核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转（IOC） 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。
* **Spring Context：**Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。
* **Spring AOP：**通过配置管理特性，Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。
* **Spring DAO：**JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。
* **Spring ORM：**Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。
* **Spring Web 模块：**Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Jakarta Struts 的集成。Web 模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。
* **Spring MVC 框架：**MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

## Ioc （控制反转）

 “控制反转”的概念所在：控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。

不创建依赖对象，但是描述创建它们的方式。在代码中不直接与对象和服务连接，但在配置文件中描述哪一个组件需要哪一项服务。容器负责将这些联系在一起。

Ioc设计模式实现了“面向接口编程，而不是实现”的原则。

## AOP （面向切面的编程）

         就是将具体的通用的应用从业务逻辑中分离出来，各自做各自专业的事情。之后通过外部容器，将他们组合起来。

# IoC

## BeanFactory 和 ApplicationContent

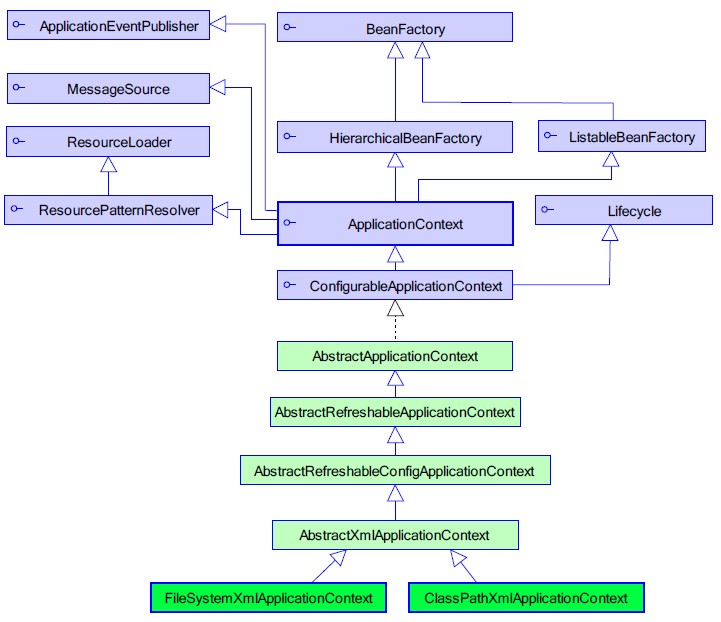
### BeanFactory

是Spring 框架最核心的接口，它提供了高级IoC 的配置机制。

### ApplicationContext

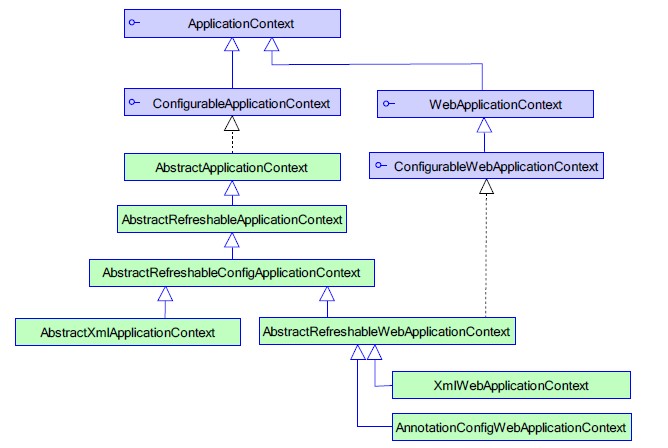
ApplicationContext由BeanFactory 派生而来，提供了更多面向实际应用的功能。

ApplicationContext的主要实现类是ClassPathXmlApplicationContext 和FileSystemXmlApplicationContext，前者默认从类路径加载配置文件，后者默认从文件系统中装载配置文件。



### WebApplicationContext

是专门为Web 应用准备的。



Spring提供了两种方式用于初始化WebApplicationContext方法，ServletContext监听器、自启动Servlet。

配置Spring配置文件路径:

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/applicationContext.xml,/WEB-INF/beanConfig.xml</param-value>

         </context-param>

监听器方式(org.springframework.web.context.ContextLoaderListener)

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

<listener>

自启动Servlet方式(org.springframework.web.context.ContextLoaderServlet)

<servlet>

<servlet-name>springContextLoaderServlet</servlet-name>

<servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

### 获取Bean工厂对象：

1. ApplicationContext ac = new FileSystemXmlApplicationContext("applicationContext.xml");
2. 通过Spring提供的工具类获取ApplicationContext对象

 ApplicationContext ac1 = WebApplicationContextUtils.getRequiredWebApplicationContext(ServletContext sc);  
         ApplicationContext ac2 = WebApplicationContextUtils.getWebApplicationContext(ServletContext sc);

1. 继承自抽象类ApplicationObjectSupport或WebApplicationObjectSupport
2. 实现接口ApplicationContextAware

### 获取Bean对象：

UserService userService = (UserService)beanFactory.getBean("userService");

## Bean装配

  <bean id="userDao" class="springdemo.dao.UserDaoImpl"></bean>

<bean id="userService" class="springdemo.service.UserServiceImpl" >

<property name="userDao" ref="userDao"></property>

</bean>

### Bean元素

value：它的是基本数据类型的

ref：是引用别一个Bean它的属性有二个Bean和local

bean：这里的Bean是指嵌套Bean，它只对外围Bean有效

list Set Map Props：是对Bean实现的集合属性

### 依赖注入的实现

**构造函数注入**

在构造函数注入中，我们通过调用类的构造函数，将接口实现类通过构造函数变量传入由于是构造函数，就必须注入，没有选择性。

**属性注入**

属性注入可以有选择地通过Setter 方法完成调用类所需依赖的注入，更加灵活方便。

**接口注入**

将调用类所有依赖注入的方法抽取到一个接口中，调用类通过实现该接口提供相应的注入方法。为了采取接口注入的方式，必须先声明一个接口，由于通过接口注入需要额外声明一个接口，增加了类的数目，而且它的效果和属性注入并无本质区别。

### 自动装配

**byName**

从Spring环境中获取目标对象时，目标对象中的属性会根据名称在整个Spring环境中查找<bean>标签的id属性值。如果有相同的，那么获取这个对象，实现关联。

 整个Spring环境：表示所有的spring配置文件中查找，那么id不能有重复的。

**byType**

从Spring环境中获取目标对象时，目标对象中的属性会根据类型在整个spring环境中查找<bean>标签的class属性值。如果有相同的，那么获取这个对象，实现关联。

缺点：如果存在多个相同类型的bean对象，会出错。如果属性为单一类型的数据，那么查找到多个关联对象会发生错误。

如果属性为数组或集合(泛型)类型，那么查找到多个关联对象不会发生异常。

**constructor**

使用构造方法完成对象注入，其实也是根据构造方法的参数类型进行对象查找，相当于采用byType的方式。

**autodetect**

自动选择：如果对象没有无参数的构造方法，那么自动选择constructor的自动装配方式进行构造注入。如果对象含有无参数的构造方法，那么自动选择byType的自动装配方式进行setter注入。

**none**

不支持自动装配功能

### 作用域

**singleton**

在spring IOC容器中仅存在一个Bean实例,Bean以单实例的方式存在.

**prototype**

每次从容器中调用Bean时,都返回一个新的实例,即每次调用getBean()时,相当于执行new XxxBean()的操作.

**request**

每次HTTP请求都会创建一个新的Bean,该作用域仅适用于webApplicationContext环境.

**session**

同一个HTTP session共享一个Bean,不同HTTP session使用不同的Bean,该作用域仅适用于webApplicationContext环境.

**globalSession**

同一个全局session共享一个Bean,一般用于portlet应用环境,该作用域仅适用于webApplicationContext环境.

         <bean id="" class="" **scope**="singleton"/>

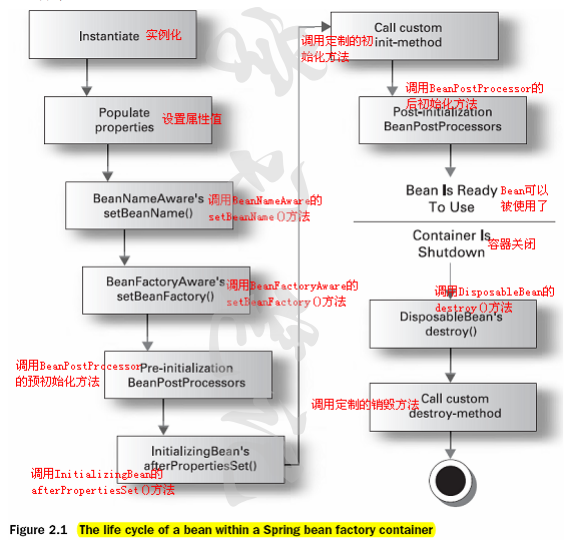
          scope 默认值是singleton

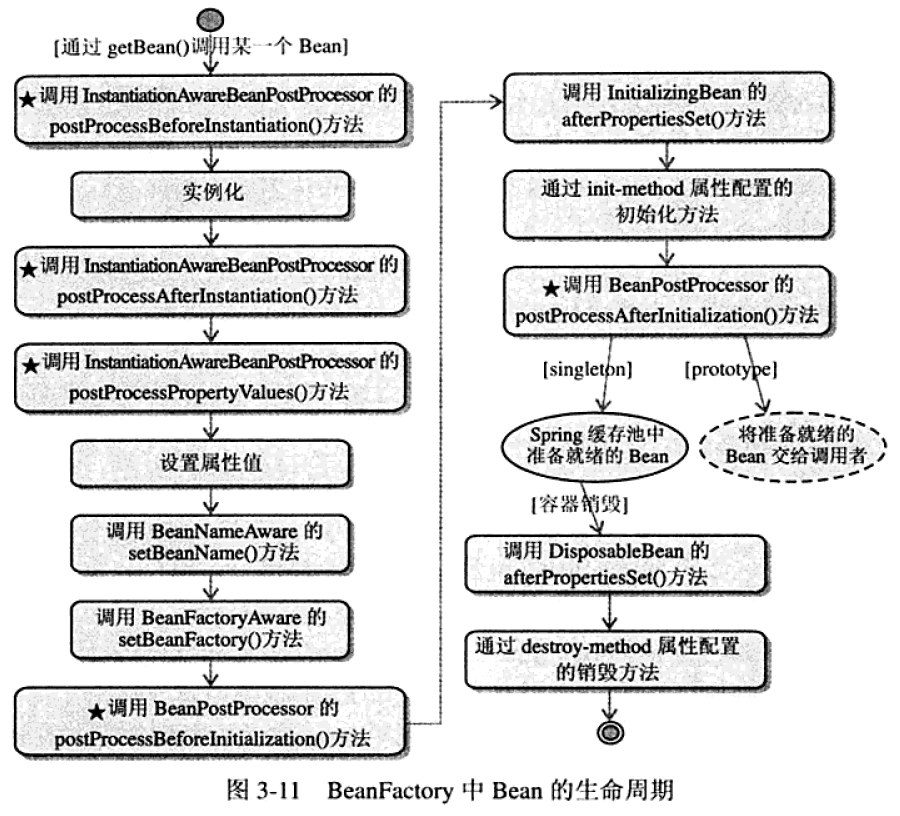
### 抽象Bean（模板）和子Bean

<bean id="chineseTemplate" class="xxx" **abstract="true"**></bean>

        <bean id="shanghai" parent="chineseTemplate"></bean>

## Bean生命周期





## 后处理器事件

### Bean后处理器

实现BeanPostProcessor接口

### 容器后处理器：

实现BeanFactoryPostProcessor接口

### 两个常用的容器后处理：

**PropertyPlaceholderConfigurer后处理器**

负责读取properties属性文件里的属性值，并将这些属性值设置成Spring配置文件的元数据。

**PropertyOverrideConfigurer后处理器**

属性文件指定的信息可以直接覆盖Spring配置文件中的元数据

# AOP

    1、Aop概念

    2、用法和配置

    3、Aop实现原理

          代理模式