Spring

春天来了...

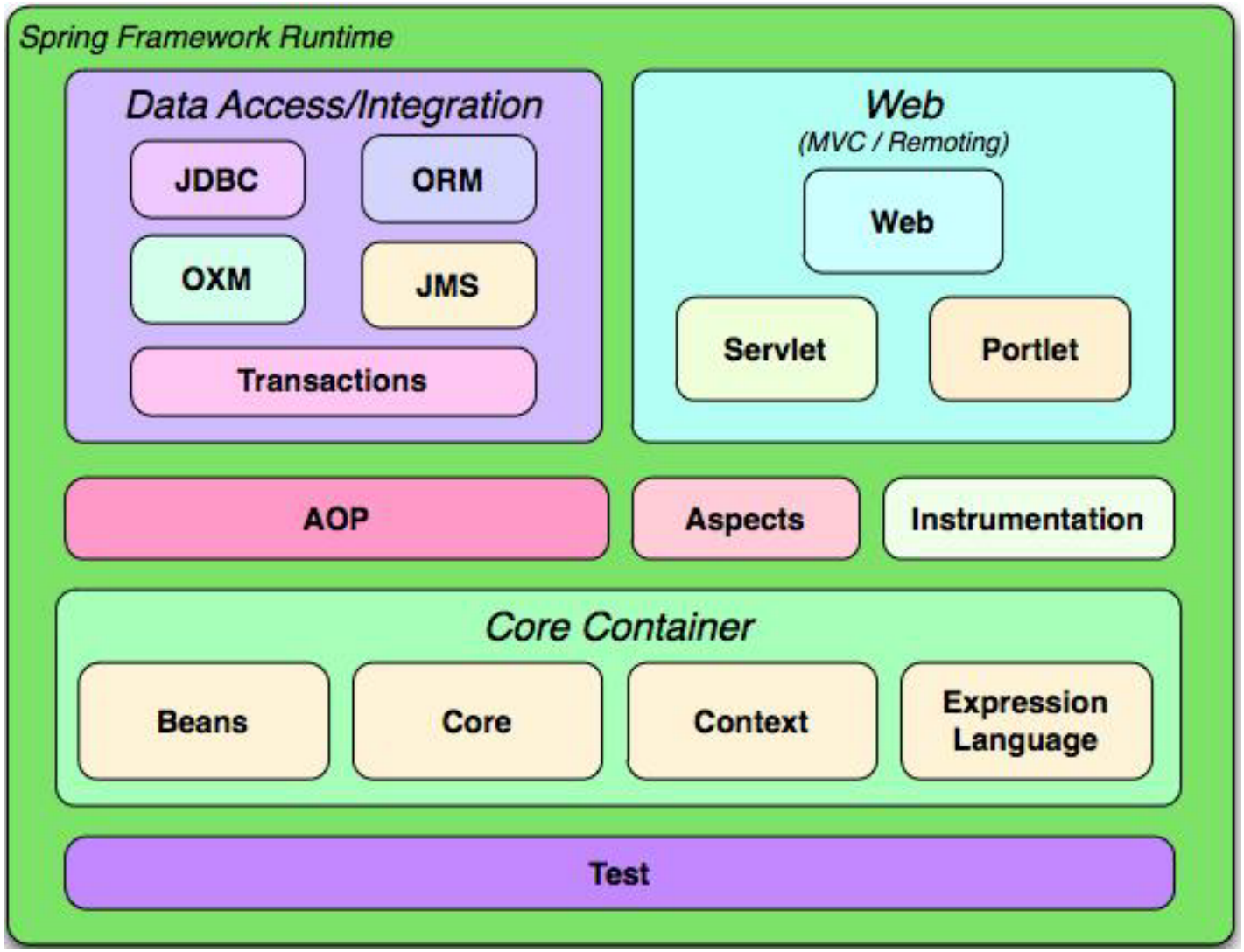
Spring简介

- Spring为企业应用的开发提供一个轻量级的解决方案。该解决方案包括：基于依赖注入的核心机制，基于AOP的声明式事务管理，与多种持久层技术的整合，以及优秀的Web MVC框架等。
- Spring致力于Java EE应用各层的解决方案，而不是仅仅专注于某一层的方案。可以说：Spring是企业应用开发的”一站式”选择，Spring贯穿表现层，业务层，持久层。然而，Spring并不想取代那些已有的框架，而是以高度的开发性与它们无缝整合。

Spring的优点

- 1.低侵入式设计，代码的污染极低。
- 2.独立于各种应用服务器，基于Spring框架的应用，可以真正实现Write Once, Run Anywhere。
- 3.Spring的DI(依赖注入)容器降低了业务对象替换的复杂性，提高了组件之间的解耦。
- 4.Spring的AOP支持允许将一些通用任务如安全，事务，日志等进行集中性处理，从而提供了更好的复用。
- 5.Spring的ORM和DAO提供了与第三方持久层框架的良好整合，并简化了底层的数据库访问。
- 6.Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可自由选用Spring框架的全部或部分。

Spring框架结构图



Spring框架结构图

- 正如前图所示，当我们使用Spring框架时，必须使用Spring Core Container(即Spring容器)，它代表了Spring框架的核心机制，Spring Core Container主要由org.springframework.core，org.springframework.beans和org.springframework.context，org.springframework.expression四个包及其子包组成，主要提供Spring IoC容器支持。

Spring下载安装

- 1. 在Java SE应用中使用Spring
- 2.在Web应用中使用Spring

Spring核心机制：依赖注入(控制反转)

- 示例先：SpringDemo/demo/
- ApplicationContext实例是Spring的核心，它既是一个巨大的工厂，也是一个功能超强的工厂。
- Spring容器根据配置文件信息，负责创建Person实例，并为Person实例设置属性值---这种由Spring容器为对象设置属性的方式被称为控制反转(Inversion of Control，IoC)

Spring核心机制：依赖注入

- 使用依赖注入，不仅可以为Bean注入普通的属性值，还可以注入其他Bean的引用。通过这种依赖注入，Java EE应用中的各种组件不需要以硬编码方式耦合在一起，甚至无须使用工厂模式。
- 依赖注入是目前最优秀的解耦方式。依赖注入让Spring的Bean以配置文件组织在一起，而不是以硬编码的方式耦合在一起。

理解依赖注入(DI)

- 不管是依赖注入，还是控制反转，其含义完全相同：当某个Java实例(调用者)需要另一个Java实例(被调用者)时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。在依赖注入的模式下，创建被调用者的工作不再由调用者来完成，因此称为控制反转；创建被调用者实例的工作通常由Spring容器来完成，然后注入调用者，因此也称为依赖注入。

理解依赖注入(DI)

- 依靠依赖注入，程序运行过程中，如果需要另一个对象协作(调用它的方法，访问它的属性)时，无需在代码中创建被调用者，而是依赖于外部容器的注入。Spring的依赖注入对调用者和被调用者几乎没有任何要求，完全支持对POJO之间依赖关系的管理。

理解依赖注入(DI)

- 依赖注入通常有如下两种：
- 1. 设值注入：IoC容器使用属性的setter方法来注入被依赖的实例。
- 2. 构造注入：IoC容器使用构造器来注入被依赖的实例。

设值注入

- 设值注入是指IoC容器使用属性的setter方法来注入被依赖的实例。这种注入方式简单，直观，因而在Spring的依赖注入里大量使用。
- 示例：SpringDemo/demo1

- 在配置文件中，Spring配置Bean实例通常会指定两个属性：
  - id：指定该Bean的唯一标识，程序通过id属性值来访问该Bean实例。
  - class：指定该Bean的实现类，此处不可再用接口，必须是实现类，Spring容器会使用XML解析器读取该属性值，并利用反射来创建该实现类的实例。

设值注入

- 从上例可以看出Spring管理Bean的灵巧性。Bean与Bean之间的依赖关系放在配置文件里组织，而不是写在代码里。通过配置文件的指定，Spring能精确地为每个Bean注入属性。因此，配置文件里的<bean />元素的class属性值不能是接口，而必须是真正的实现类。
- Spring会自动接管每个<bean />定义里的<property />元素定义，Spring会在调用无参数的构造器创建默认的Bean实例后，调用对应的setter方法为程序注入属性值。<property />定义的属性值将不再由该Bean来主动设置，管理，而是接收Spring的注入。
- 每个Bean的id属性是该Bean的唯一标识，程序通过id属性访问Bean，Bean与Bean的依赖关系也通过id属性关联。

构造注入

- 设值注入是通过setter方法为目标Bean注入依赖关系，另外一种注入方式---在构造实例时，已经为其完成了依赖关系的初始化。这种利用构造器来设置依赖关系的方式，称为构造注入。
- 示例：SpringDemo/demo2

两种注入方式的对比

- 这两种依赖注入的方式，并没有绝对的好坏，只是适应的场景有所不同。相比之下，设值注入具有如下的优点：
- 1.与传统的JavaBean的写法更相似，程序开发人员更容易理解，接受。通过setter方法设定依赖关系显得更加直观，自然。
- 2. 对于复杂的依赖关系，如果采用构造注入，会导致构造器过于臃肿，难以阅读。Spring在创建Bean实例时，需要同时实例化其依赖的全部实例，因而导致性能下降。而使用设值注入，则能避免这些问题。
- 3. 尤其是在某些属性可选的情况下，多参数的构造器更加笨重。

两种注入方式的对比

- 构造注入也不是绝对不如设值注入，在某些特定的场景下，构造注入有如下优势：
- 1.构造注入可以在构造器中决定依赖关系的注入顺序，优先依赖的优先注入。例如：组件中其他依赖关系的注入，常常需要依赖于DataSource的注入。采用构造注入，可以在代码中清晰地决定注入顺序。
- 2.对于依赖关系无需变换的Bean，构造注入更有用处。因为没有setter方法，所有的依赖关系全部在构造器内设定。因此，无需担心后续的代码对依赖关系产生破坏。

两种注入方式的对比

- 依赖关系只能在构造器中设定，则只有组件的创建者才能改变组件的依赖关系。对组件的调用者而言，组件内部的依赖关系完全透明，更符合高内聚的原则。

- 建议采用以设值注入为主，构造注入为辅的注入策略。对于依赖关系无需变化的注入，尽量采用构造注入；而其他的依赖关系的注入，则考虑采用设值注入。

使用Spring容器

- Spring有两个核心接口：BeanFactory和ApplicationContext，其中ApplicationContext是BeanFactory的子接口。它们都可代表Spring容器，Spring容器是生成Bean实例的工厂，并管理容器中的Bean.
- Bean是Spring管理的基本单位，在基于Spring的Java EE应用中，所有的组件都被当成Bean处理，包括数据源，Hibernate的SessionFactory，事务管理器等。

使用Spring容器

- 应用中的所有组件，都处于Spring的管理下，都被Spring以Bean的方式管理，Spring负责创建Bean实例，并管理其生命周期。Spring里的Bean是非常广义的概念，任何的Java对象，Java组件都被当成Bean处理，甚至这些组件并不是标准的JavaBean。
- Bean在Spring容器中运行，无需知道Spring容器的存在，一样可以接受Spring的依赖注入，包括Bean属性的注入，协作者的注入，依赖关系的注入等。

使用Spring容器

- Spring容器最基本的接口就是BeanFactory。BeanFactory负责配置，创建，管理Bean，它有一个子接口ApplicationContext，因此也被称为Spring上下文。Spring容器还负责管理Bean与Bean之间的依赖关系。
- BeanFactory接口包含如下几个基本方法：
  - boolean containsBean(String name)
  - <T> T getBean(Class<T> requiredType)
  - Object getBean(String name)
  - <T> T getBean(String name, Class requiredType)
  - Class<?> getType(String name)

使用Spring容器

- 调用者只需使用getBean()方法即可获得指定Bean的引用，无需关系Bean的实例化过程。即：Bean实例的创建过程完全透明。
- BeanFactory有一个常用的实现类：org.springframework.beans.factory.xml.xmlBeanFactory类。
- ApplicationContext是BeanFactory的子接口，对于大部分Java EE应用而言，使用它作为Spring容器更方便。其常用实现类是FileSystemXmlApplicationContext, ClassPathXmlApplicationContext和AnnotationConfigApplicationContext。

使用Spring容器

- 如果在web应用中使用Spring容器，通常有XmlWebApplicationContext，AnnotationConfigWebApplicationContext两个实现类。

- 创建Spring容器的实例时，必须提供Spring容器管理的Bean的详细配置信息。Spring的配置信息通常采用XML配置文件来设值。

使用Spring容器

- 如果要同时加载多个XML配置文件，则可以采用如下方式：
- ApplicationContext appContext = new ClassPathXmlApplicationContext(new String[] {“bean.xml”, “service.xml”});

使用Spring容器

- 由于ApplicationContext本身就是BeanFactory的子接口，因此ApplicationContext完全可以作为Spring容器使用，而且功能更强。当然，如果有需要，也可以把ApplicationContext实例赋给BeanFactory变量。

- Spring配置文件的根元素是<beans … />，该元素可以接受0个到多个<bean … />子元素，每个<bean … />子元素配置一个Bean实例。

使用ApplicationContext

- 大部分时候，我们都不会使用BeanFactory实例作为Spring容器，而是使用ApplicationContext实例作为容器。

- ApplicationContext允许以声明式方式操作容器，无须手动创建它。可利用如ContextLoader的支持类，在Web应用启动时自动创建ApplicationContext。当然，也可以采用编程的方式创建ApplicationContext。

使用ApplicationContext

- 除了提供BeanFactory所支持的全部功能外，ApplicationContext还有如下额外的功能：
  - ApplicationContext继承MessageSource接口，因此提供国际化支持。
  - 资源访问，比如URL和文件
  - 事件机制
  - 载入多个配置文件
  - 以声明式的方式启动，并创建Spring容器。

使用ApplicationContext

- ApplicationContext包括BeanFactory的全部功能，因此建议优先使用ApplicationContext。除非对于某些内存非常关键的应用，才考虑使用BeanFactory。
- 当系统创建ApplicationContext容器时。默认会预初始化所有的singleton Bean。也就是说，当ApplicationContext容器初始化完成后，容器中所有singleton Bean也实例化完成。
- 这意味着：系统前期创建ApplicationContext时将有较大的系统开销，但一旦ApplicationContext初始化完成，程序后面获取singleton Bean实例时将拥有较好的性能。

Spring容器中的Bean

- 从本质上来看，Spring容器就是一个”超大型”工厂，Spring容器中的Bean就是该工厂的产品。Spring容器能生产哪些产品，则完全取决于开发者在配置文件中的配置。

Spring容器中的Bean

- 对于开发者来说，开发者使用Spring框架所做的主要是两件事：1.开发Bean；2.配置Bean。对于Spring框架来说，它要做的，就是根据配置文件来创建Bean实例，并调用Bean实例的方法完成”依赖注入”---这就是所谓IoC的本质。

Spring配置文件

- <beans…/>元素是Spring配置文件的根元素，该元素可以指定如下属性:
  - default-lazy-init：指定该<beans…/>元素下配置的所有Bean默认的延迟初始化行为。
  - default-autowire：指定该<beans…/>元素下配置的所有Bean默认的自动装配行为。
  - …

Spring配置文件

- 定义Bean时，通常需要指定两个属性:
  - id
  - class
- 除此之外，还可以为<bean…/>元素指定name属性，用于为Bean实例指定别名。

Spring配置文件

- 在默认情况下，当Spring创建ApplicationContext容器时，Spring会自动预初始化容器中所有的singleton实例，如果我们不想让Spring容器预初始化某个singleton Bean，则可为该<bean…/>元素增加lazy-init属性，指定该属性为true，则Spring不会预初始化该Bean实例。

Spring配置文件 --- 容器中Bean的作用域

- 当通过Spring容器创建一个Bean实例时，不仅可以完成Bean实例的实例化，还可以为Bean指定特定的作用域。Spring支持5种作用域：
  - singleton：单例模式，在整个Spring IoC容器中，使用singleton定义的Bean将只有一个实例。
  - prototype：原型模式，每次通过容器的getBean方法获取prototype定义的Bean时，都将产生一个新的Bean实例。
  - request
  - session
  - global session

Spring配置文件 --- 容器中Bean的作用域

- 示例：SpringDemo/demo3

Spring配置文件 --- 配置依赖

- Java应用中各组件的相互调用的实质可以归纳为依赖关系，根据注入方式的不同，Bean依赖注入通常表现为如下两种形式：
  - 属性：通过<property…/>元素配置，对应设值注入
  - 构造器参数：通过<constructor-arg…/>元素设定，对应构造注入

通常情况下，Spring在实例化容器时，会校验BeanFactory中每一个Bean的配置

Spring配置文件 --- 配置依赖

- BeanFactory与ApplicationContext实例化容器中Bean的时机有所不同：前者等到程序需要Bean实例时才创建Bean，而后者在容器创建ApplicationContext实例时，会预初始化容器中的全部Bean。

- 由于Java实例的属性值可以是各种数据类型，除了基本类型值，字符串类型值等，还可以是其他Java实例，也可以是容器中的其他Bean实例，甚至是Java集合，数组等，所以Spring允许通过如下元素为Bean实例的属性指定值：
  - value, ref, bean, list, set, map及pros

Spring配置文件 --- 配置合作者Bean

- 如果需要为Bean设值的属性值是容器中的另一个Bean实例，则应该使用<ref…/>元素。使用<ref…/>元素时可指定如下两个属性：
  - bean
  - local

Spring配置文件 --- 使用自动装配注入合作者Bean

- Spring能自动装配Bean与Bean之间的依赖关系，即无需使用ref显式指定依赖Bean。

- byName规则
- byType规则

Spring配置文件 --- 注入嵌套Bean

Spring配置文件 --- 注入集合值

Spring配置文件 --- 组合属性

Spring的Bean和JavaBean

- Spring容器对Bean没有特殊要求，甚至不要求该Bean像标准的JavaBean --- 必须为每个属性提供对应的getter和setter方法。Spring中的Bean是Java实例，Java组件；而传统的Java应用中的JavaBean通常作为DTO(数据传输对象)，用来封装值对象，在各层之间传递数据。

Spring的Bean和JavaBean

- 虽然Spring对Bean没有特殊要求，但建议Spring中的Bean满足如下要求：
  - 尽量为每个Bean实现类提供无参数的构造器
  - 接受构造注入的Bean，则应提供对应的构造器
  - 接受设值注入的Bean，则应提供对应的setter方法，并不强制要求提供对应的getter方法。

Bean实例的创建方式及依赖配置

- 大多数情况下，BeanFactory直接通过new关键字调用构造器来创建Bean实例，而class属性指定了Bean实例的实现类。因此，<bean…/>元素必须指定Bean实例的class属性，但这不是实例化Bean的唯一方法。
- 创建Bean通常有如下方法：
  - 调用构造器创建Bean实例
  - 调用静态工厂方法创建Bean
  - 调用实例工厂方法创建Bean