# Spring 3.2




## ----依赖注入概论

# 传统组建调用方式

Programming Your Future

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
{
    A1010Services services=new A1010Services(this.createdto(request));
    Map ins=services.findById();
    if(ins!=null)
    {
        request.setAttribute("ins", ins);
    }
    .....
}
```

# 缺陷分析



- 1 **无法真正动态加载**
  - 无法动态替换，替换时必须修改代码，即使应用的是工厂，依然存在代码修改的情况，因为我们必须重新设计工厂。

- 2 **实例无法共享，**
  - 多组件无法共享同一实例，因为实例的生命周期是依赖于调用者组件，二者同生共死，即使应用工厂，也是每个实例只为一个客户端使用。

- 3 **测试困难**
  - 因为必须包含一个组件的实例，如果实例包含了数据库引用的化，无法离开数据库运行环境。

# 使用对象必须创建吗



- **对象创建方式**
  - Servlet环节，为了使用一个对象，我们曾经采用如下方式:StudentServices services=new StudentServices ();在应用设计模式的思想进行分析以后，我们明白，过去的做法，造成了创建者和使用者责任的无法区分，系统的耦合度过高。
- **于是我们应用工厂进行解耦**
  - 程序世界进入半现代社会，但是，虽然使用者不再new一个对象了，可是，也不过仅仅是把对象的创建过程封装到了工厂中而已，我们毕竟还是new了一个对象。耦合并没有完全解除。
- **为了使用对象，我们必须要new吗？耦合就没有办法完全解除吗？我们一直在期待！**

# 不new你怎么办



- **我们可以采用IoC(控制反转)。**
- • 控制反转：Inversion of Control
- 让组件之间的依赖关系通过抽象(接口或抽象类的变量)来建立,在组件运行期间,将组件依赖的实际对象,注入(填充)进来,并由组件内部包含的抽象变量来引用,从而,就可以解藕了.

# 春天的来临



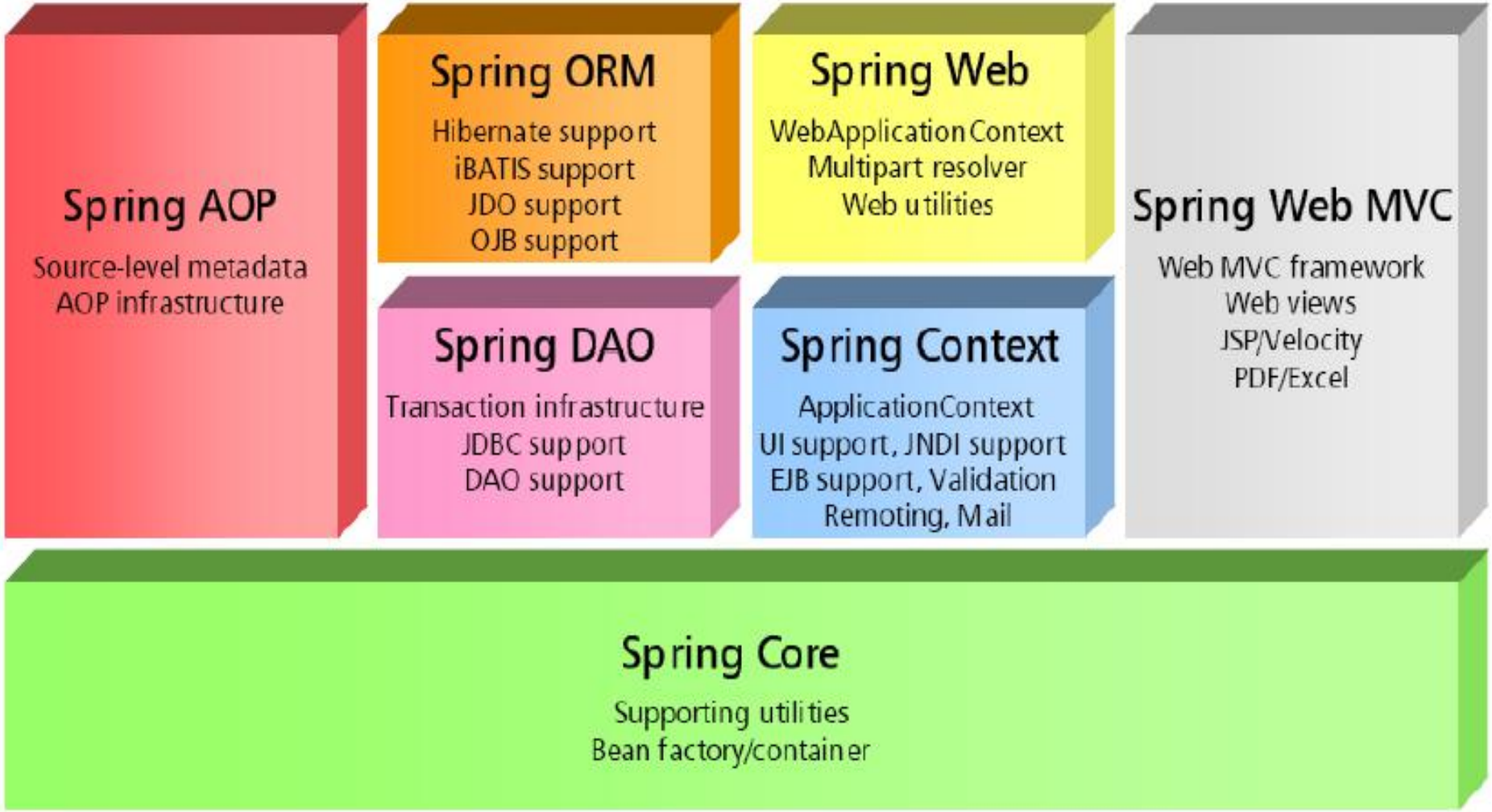
- 2002最后一场雪以后，Rod Johnson结合自己多年的开发经验，撰写了《Expert One-on-One J2EE设计与开发》一书，书中对正统J2EE架构的臃肿、低效提出质疑，并阐述新的技术实现以及新的思维模式。引发了人们对正统J2EE的反思。这本书真正地改变了Java世界。翌年春天，基于该书的源代码，形成了一个新的技术架构，这就是Spring。 Spring的出现，使得正统J2EE架构一统天下的局面被打破。基于SSH(Struts+ Spring +Hibernate)的J2EE架构也逐渐得到人们的认可，并且在大型的项目架构中也逐渐开始应用。
- 这本书也体现了作者对技术的态度：技术的选择应该基于实证或是自身的经验，而不是任何形式的偶像崇拜或者门户之见。

# Spring简介



- **Spring是一个开源的J2EE框架。它作为一个优秀的轻量级的企业应用开发框架，可以大大简化企业应用开发的复杂性，能够创建出松耦合、易测试、易扩展、易维护的Java应用系统，就象春风一样，吹拂着Java大地。**
- 特点
  - 轻量级：从大小和系统开支上说Spring都算是轻量级的。
  - 反向控制：Spring提倡使用反向控制(IoC)来实现松耦合。
  - 面向切面：Spring对面向切面编程(AOP)提供了强大的支持，通过将系统服 务（如监控和事务管理）从业务逻辑中分离出来，实现了内聚开发。
  - 容器：Spring是一个容器，能够装配对象并管理对象的生命周期。
  - 框架：Spring实现了把简单的组件组合装配成一个复杂系统的功能。
- **Spring的官方网址：www.springframework.org**

# Spring架构组成


Spring AOP
Source-level metadata
AOP infrastructure
Spring ORM
Hibernate support
iBATIS support
JDO support
OJB support
Spring Web
WebApplicationContext
Multipart resolver
Web utilities
Spring Web MVC
Web MVC framework
Web views
JSP/Velocity
PDF/Excel
Spring DAO
Transaction infrastructure
JDBC support
DAO support
Spring Context
ApplicationContext
UI support, JNDI support
EJB support, Validation
Remoting, Mail
Spring Core
Supporting utilities
Bean factory/container

# Spring架构组成（一）

- 核心容器：
  - 核心容器提供Spring框架的基本功能，为Spring提供了基础服务支持，核心容器的主要组件就是BeanFactory，BeanFactory是所有基于Spring框架系统的核心，通过BeanFactory，Spring使用工厂模式来实现IoC，将应用程序的配置和依赖关系与实际的应用程序分离开来.　Spring之所以称为容器，就是由于BeanFactory的自动装备和注入。

# Spring架构组成（二）

- **Application Context（上下文）**
  - Spring 上下文是由一个配置文件，向 Spring 框架提供上下文信息。 **BeanFactory使spring成为容器，上下文模块使Spring成为框架**，这个模块对BeanFactory进行了扩展，添加了对I18N，系统生命周期事件以及验证的支持。 这个模块提供了许多企业级服务，例如邮件服务，JNDI访问，EJB集成，远程调用以及定时服务，并且支持与模板框架的集成。

# Spring架构组成（三）

- Spring AOP：
  - AOP面向切面编程。该模块将AOP 编程功能集成到了 Spring 框架中。这样，凡是 Spring 框架管理的任对象都可以很容易地支持 AOP。该模块为应用程序中基于 Spring 管理的对象提供了事务管理服务。
  - 这个模块由于使用了 AOP Alliance的API，所以，可以和其他AOP框架互通，
  - AOP Alliance是一个开源项目 目的是促进AOP的使用，并且通过定义一套通用的接口和组件来确保不同的AOP之间达到互通性。

# Spring架构组成（四）



- **Spring DAO：**
  - **这个模块封装了数据库连接的创建，语句对象生成，结果集处理，连接关闭等操作，而且重构了所有数据库系统的异常信息，用户不再需要处理数据库异常了。 在这个模块中，利用了Spring的AOP模块完成了为系统中对象提供事务管理的服务。**

# Spring架构组成（五）

Programming Your Future

- **Spring ORM**：
  - Spring 没有实现自己的ORM方案，而是为当前主流的ORM框架预留了整合接口，**包括hibernate，JDO。** 所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。

# Spring架构组成（六）

- **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架构组成（七）

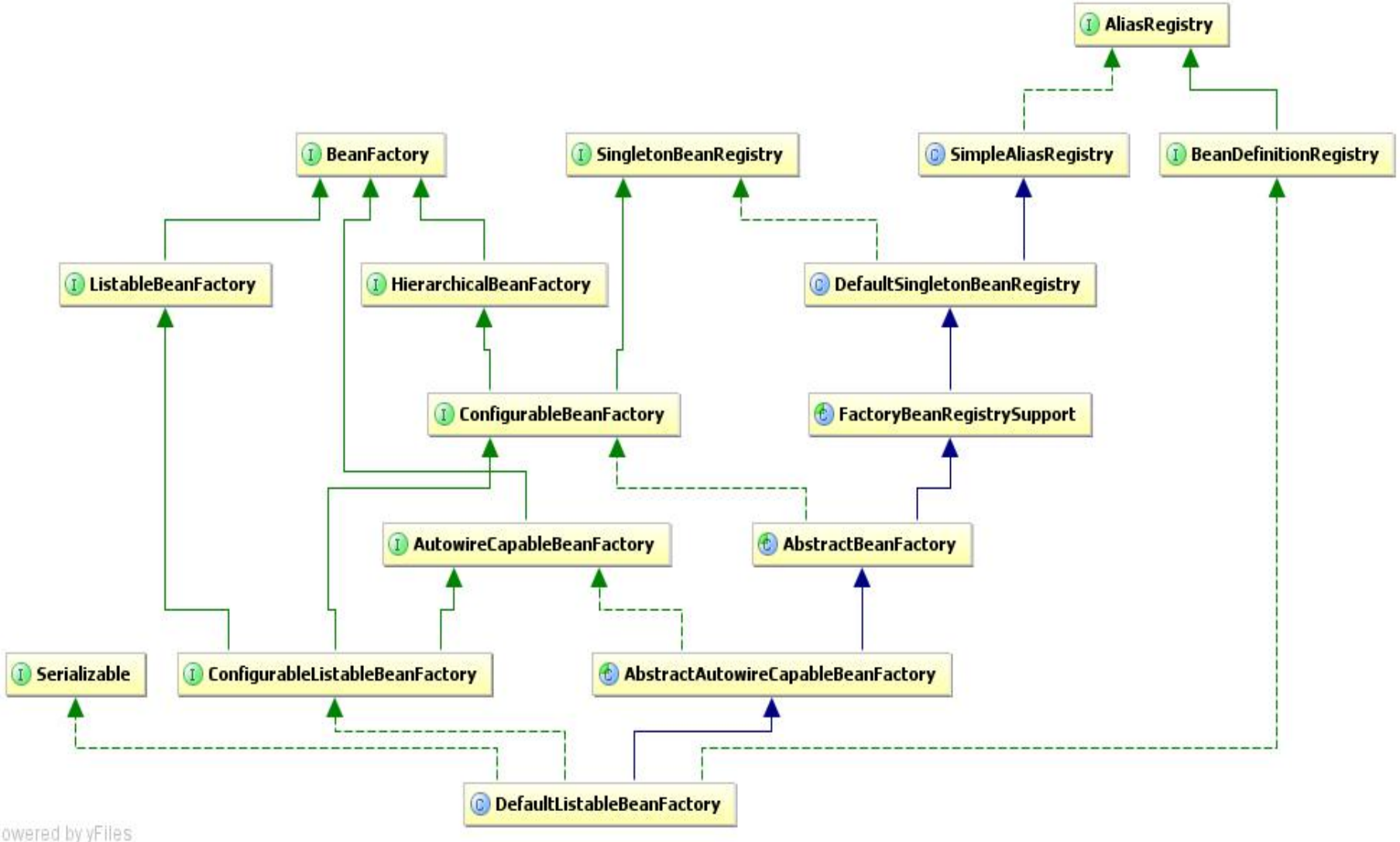
- **Spring MVC 框架**：MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

# BeanFactory树



- BeanFactory是Spring中采用工厂设计模式的一个核心接口，该接口的实现类负责创建和分发Bean。
- 在Spring中由几种BeanFactory的实现类，其中最常用的是：
- XmlBeanFactory
- DefaultListableBeanFactory

# BeanFactory树--类图



# BeanFactory树---实现方式1



```
BeanFactory factory=
    new XmlBeanFactory(new ClassPathResource("Beans.xml"));
Person p= factory.getBean("person",Person.class);
System.out.println(p.getPname());
```

# BeanFactory树---实现方式2(new)

Programming Your Future

- DefaultListableBeanFactory factory
- = new DefaultListableBeanFactory();
- XmlBeanDefinitionReader reader
- = new XmlBeanDefinitionReader(factory);
- //加载资源文件
- reader.loadBeanDefinitions(new ClassPathResource("Beans.xml"));
- //获取 Bean
- Person p=factory.getBean("person", Person.class);
- System.out.println(p.getPname());

# Spring的程序的运行流程



- 从上面的例子中我们可以看出Spring程序工作的一般流程如下：
  - 1：初始化Spring的Bean工厂
  - 2：通过Bean工厂获得Bean的实例
  - 3：调用Bean实例方法，完成业务处理
  - 4：销毁Spring的Bean工厂(Spring负责)！

# Bean的初始化过程



- Spring的bean初始化过程一共有14个步骤,并且严格按顺序执行.这其中,只有第一,第二和13个步骤是必要的,其他步骤,都需要在bean中引入Spring的接口,所以,一般而言,不需要使用.
- 必要步骤
  - 1.根据xml中对bean的定义,创建一个bena的实例,并传入必要的构造方法参数.(创建bean)
  - 2.根据XML中对bean的定义完成依赖注入(装配Bean)
  - 13.如果为bean定义了init-method方法,调用改方法完成初始化工作.
    在配置文件中为bean指定init-method属性,属性名为要调用的方法名称.
    名称不限定,可以私有.（初始化Bean)
- 注意:
  - Spring下,bean是先装配后初始化的.
    - 因为初始化过程可能用到注入的实际对象

# 关于Bean的疑问

- 提问？
  - Spring的工作流程是围绕怎样得到Bean和怎么使用Bean来展开的，即使BeanFactory的创建也可以看作是Bean使用过程中的附属产品，那么，Spring中的Bean到底是什么东西？

# 再论Bean



- JavaBean原始形态是一种符合特定规范的java对象，有无参构造器，通过get/set提供外界对私有属性的访问，是最简单最基本的java组件，即可被用作数据承载的载体即VO，也可以用来执行业务逻辑。
- 随着编程思维以及现实环境的发展， JavaBean的规范在实际应用中日趋模糊，日渐与POJO融合。
- 在Spring中，大力推荐使用的JavaBean，实际上就是POJO，凡是可以被实例化或是可以被JNDI获得的对象，都可以被Spring容器管理，都是Bean

# ApplicationContext



- 虽然BeanFactory是Spring中很重要的一个概念， 但是在大多数情况下，我们并不是直接应用它，而是应用BeanFactory的子接口ApplicationContext接口。
-

# 实现类的应用



- ApplicationContext有很多实现类，但只需掌握常用的三个：
- ClassPathXmlApplicationContext：
  - 从类路径下的XML文件中载入上下文定义信息
- FileSystemXmlApplicationContext：
  - 从文件系统路径下的XML文件中载入上下文定义信息
- XmlWebApplicationContext：
  - 通过ContextLoaderListener从内部导入context文件

# ApplicationContext应用示例



- 从文件系统路径下导入配置文件
  - FileSystemXmlApplicationContext实例
- 从类路径下导入配置文件
  - **ClassPathXmlApplicationContext实例**

# ApplicationContext与BeanFactory



- **BeanFactory(延迟加载)**
  - 采用延迟加载Bean，直到第一次使用getBean()方法获取Bean实例时，才会创建Bean。
- **ApplicationContext(即时加载)**
  - ApplicationContext在自身被实例化时一次完成所有Bean的创建。
- 区别
  - 在服务启动时ApplicationContext就会校验XML文件的正确性，不会产生运行时bean装配错误。
  - BeanFactory在服务启动时，不会校验XML文件的正确性，获取bean时，如果装配错误会，马上就会产生异常

# 再论软件组件调用



- 一如我们此前讨论传统组件调用方式存在很多缺陷。控制权在应用程序本身，程序的流程完全由开发者自己控制。在这种情况下，如果系统中有大量组件,并且其生命周期和依赖关系由组件自己来维护,就会大大增加系统的复杂程度,而且会造成组件之间的紧耦合,不利于系统的维护和测试。

# 再论软件组件调用（下）



- 经过上面的分析我们可以发现，问题的核心归结为:如何组装大量的Bean，使之相互配合完成复杂的工作，即便于维护又有利于测试。以上问题的产生,就在于bean的依赖关系管理不恰当。
- 解决之道就是应用IoC,那么什么是IoC那。

# 再论IoC



- **IoC的直译是控制反转。**
- 在IoC模式下，控制权限从应用程序转移到了IoC容器中。组件不是由应用程序负责创建和配置，而是由IoC容器负责。
- 使用IoC的情况下,对象是被动地接收依赖类而不是主动地查找,
  - - 对象不是从容器中查找他的依赖类，而是容器在实例化对象时，主动地将他所依赖（一般依赖关系，对象包含）的对象注入给他。
- 应用程序只需要直接使用已经创建并且配置好的组件即可，而不必自己负责创建和配置。
- 在实际的工作中人们发现使用IoC来表述这种机制,并不是很准确甚至有些晦涩,于是引发了另外一个概念,DI(依赖注入)
- 注意:
  - 把什么转移了,控制权到底是什么关系?

# 依赖注入（上）

Programming Your Future

- 目前人们认为使用依赖注入（DI）来阐述IoC更为合适。而事实上早在2004年初，Martin Fowler在他的站点谈论控制反转时曾提问：“*问题在于，它们转变的是什么方面的控制？*”。 Fowler建议重命名该原则（或至少给它一个更加明确的名称），并开始使用“*依赖注入*”这个术语。

# 解析IoC和DI（上）



- IoC解析
- IoC(控制反转)是Spring容器的内核,AOP以及声明式事务都是在此之上完成的. IoC包括两层含义1.控制;2反转.关键的问题在于他把什么东西的控制给反转了.
  - 控制指组件之间的调用关系由程序自身控制(从而造成了紧耦合).
  - 反转是说,将这种调用关系的控制权从程序中异除,并交给第三方管理.
- 也就是组件的调用关系由程序自身管理转变为第三方管理.是组件依赖关系的控制权进行了反转
- 对于上述机制,应用IoC来描述并不准确,而且有些晦涩.

# 解析IoC和DI（下）



- DI解析
- 之所以会产生组件调用,是为了获取被调用组件的功能,调用者将自己应该做的事情,委派给了被调用的对象.也就是说,调用者要完成自身的任务,必须依赖被调用的对象.这种关系实际上就是一般依赖关系(通俗点说,就是一个组件内部包含另外一个组件的实例,把该自己干的事交给自己包含的组件去完成),因此IoC所表述的机制,实际上就是将调用者对接口实现类的依赖关系,从程序中移除,转交第三方管理实例,并且,由第三方在运行期间将调用者依赖的具体类填充进来.也就是说组件之间的依赖关系,是在程序运行期间由第三方来管理的.这个就是依赖注入的概念(DI).基于上述分析,DI比IoC更准确.
- 实际上就是将调用者为完成功能所依赖的实现类,在程序运行期间,由容器自动填充给调用者.这个就是依赖注入的核心思想.在依赖注入的应用中,组件并不关心被注入的对象是谁,只关心这个对象能完成的功能.也就是这个对象是那个接口的具体类实例.

# 简单的约定

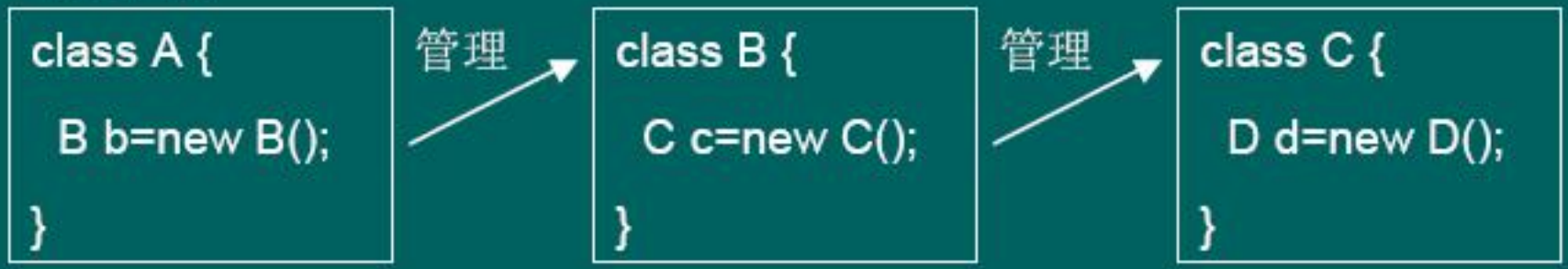
Programming Your Future

- 我们在讲课时一律采用“依赖注入”这个概念，而提到容器时，我们采用“IoC容器”一词，因为他更加流行。

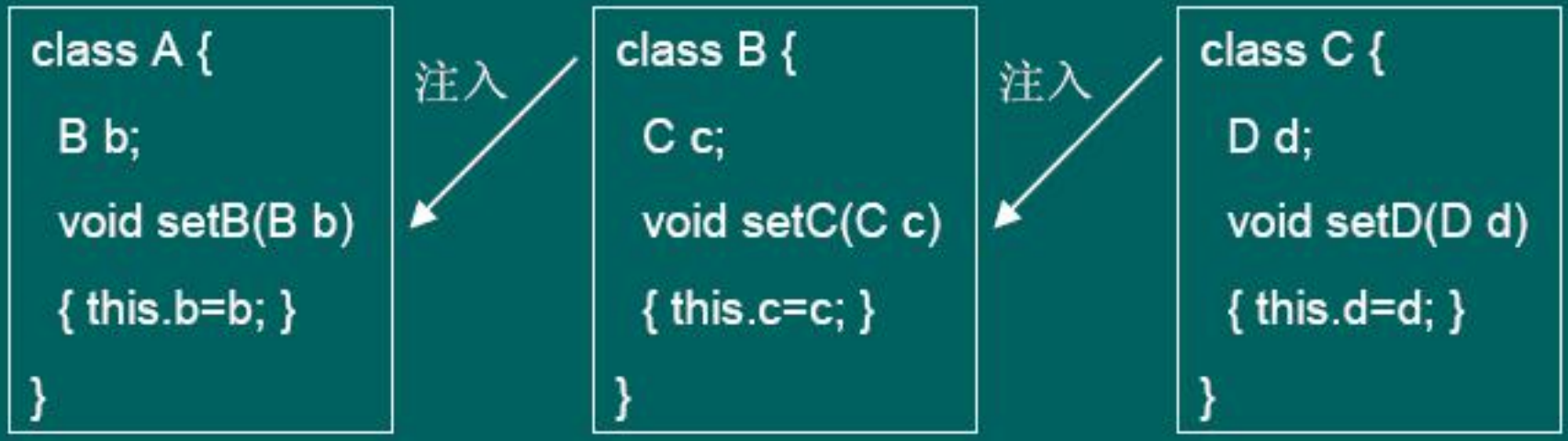
# 依赖方式比较

Programming Your Future

- 不使用IoC：




- 使用IoC



# 依赖注入的三种方式

Programming Your Future

- 构造子注入
  - 构造注入指的就是在接受注入的类中定义一个构造方法，并在参数中定义需要注入的元素。
- 设置属性注入
  - Set注入指的就是在接受注入的类中定义一个Set方法，并在参数中定义需要注入的元素。
- 接口注入---接口注入存在侵入性
  - 接口注入指的就是在接口中定义要注入的信息，并通过接口完成注入。

# 基本类型注入

- 基本数据类型的注入在配置文件中直接注入即可，步需要数据类型，IoC容器会自动的根据bean中属性的类型，完成类型转换，并赋值。
- 注入语法如下
- <bean id="classes" class="com.qhit.impl.Classes">
- <property name="sid" value="1"/>
- <property name="sname" value="貂禅"/>
- </bean>
- 其中id表示bean在整个工程中的唯一标识，class表示该标识的引用类
  - <property name="sname">
- <value>貂禅</value>
- </property>

# Null的注入

- 如果某个Bean中某个属性没有值,我们应该注入null
- 语法如下
  - <property name="state">
  - <null/>
  - </property>

# 引用类型的注入

- 如果我们注入的不是基本属性,而是引用类型,即对于委派关系进行注入,我们需要:
- 用<ref> 代替<value>
- <bean id="classes" class="com.qhit.bean.Classes">
- <property name="cid" value="1"/>
- <property name="cname" value="天山"/>
- </bean>
- <bean id="student" class="com.qhit.bean.Student">
- <property name="sid" value="1"/>
- <property name="sname" value="杨云骢"/>
- <property name="state" value="1"/>
- <property name="classes" ref="classes"/>
- </bean>

# 集合类型注入-set

- Set中基本数据类型的注入
- 等同于普通类中基本数据类型的注入, 但是set会自动过滤重复数据.
- 语法如下

```
<property name="students">
    <set>
      <value>A</value>
      <value>B</value>
      <value>C</value>
      <value>B</value>
    </set>
</property>
```

# Set中引用类型的注入

- 如果属性承载的也是对bean 引用,在set内部等同于对象的引用:语法如下
- <property name="students">
- <set>
- <ref bean="stu1"/>
- <ref bean="stu2"/>
- <ref bean="stu3"/>
- <ref bean="stu4"/>
- </set>
- </property>

# Map的注入

```
<property name="homes">
  <!-- map映射关键子 -->
  <map>
    <!-- map中的一个实体,以及该元素实体字 -->
    <entry key="R1">
      <!-- map中该实体的值 -->
      <value>优秀班级</value>
    </entry>
    <entry key="R2">
      <!-- map中该实体的值是一个引用类型 -->
      <ref bean="stu1"/>
    </entry>
  </map>
</property>
```

# List注入

```
<property name="home">
  <list>
    <value>黑风洞</value>
    <value>花果山</value>
    <value>灵山</value>
  </list>
</property>
```

# 资源文件的注入

Programming Your Future

```
<property name="info">
  <props>
    <prop key="DS">java:Mysql</prop>
    <prop key="uname"> root </prop>
    <prop key="pwd"></prop>
  </props>
 </property>
```

# 构造子注入

- 构造子注入的优点是,构造函数的参数是对象初始化是强制注入的,这样,可以保证类在创建时就被正确初始化了:

```
<bean id="business" class="com.qhit.bean.Business">
        <!-- 按参数顺序添加 -->
        <constructor-arg ref="stu1"/>
        <constructor-arg ref="classes"/>
</bean>
```

# 多构造子注入实现细节

Programming Your Future

- 存在多构造子是,容易引起IoC容器的混乱,需要明确告诉容器,程序中使用的是那个构造子.语法如下:

```
<bean id="b2" class="com.qhit.business.ioc.Business2">
    <constructor-arg index="0" value="2" type="int"></constructor-arg>
    <constructor-arg index="1" value="1" type="int"></constructor-arg>
</bean>
```

- 属性含义:
  - index :第几个参数,从零开始
  - type:参数类型

# 接口注入



- 由于接口注入的繁琐与侵入性过高Spring没有提供对接口注入的实现.

# Bean的作用域



- 在Spring2.0开始,为bean定义了五种类型的作用域,用于指定bean的生命周期.
- 配置问题属性为scope.作用域类别如下:
  - singleton (单例模式): 每次getBean返回相同的实例
  - prototype (原型模式): 每次创建返回新的实例
  - request :
  - session :
  - globalSession :
- 技巧:对于单例和原型模式可以通过构造子输出方式验证.

# 再论注入方式

Programming Your Future

- 在Spring中提供了属性注入(设置属性注入)和构造子注入两种方式,到底该应用那种方式那? Spring推荐优先使用属性注入,因为属性注入简单明了,从属性名称中,我们可以清除看到依赖关系.而构造子注入在存在多构造子时容易引起不清晰.
- 但是,到底该用何种方式,一切取决于实际需要.

# 谢谢！