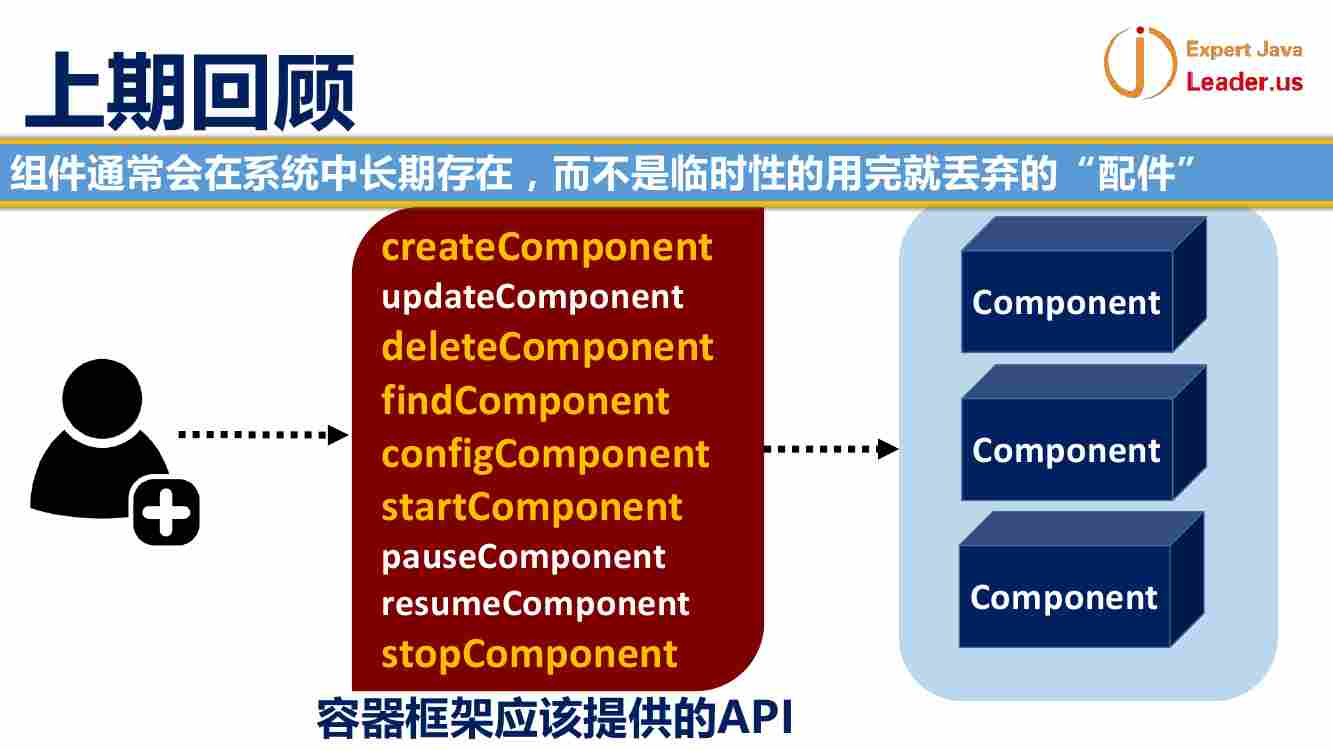


Spring如何定义一个组件



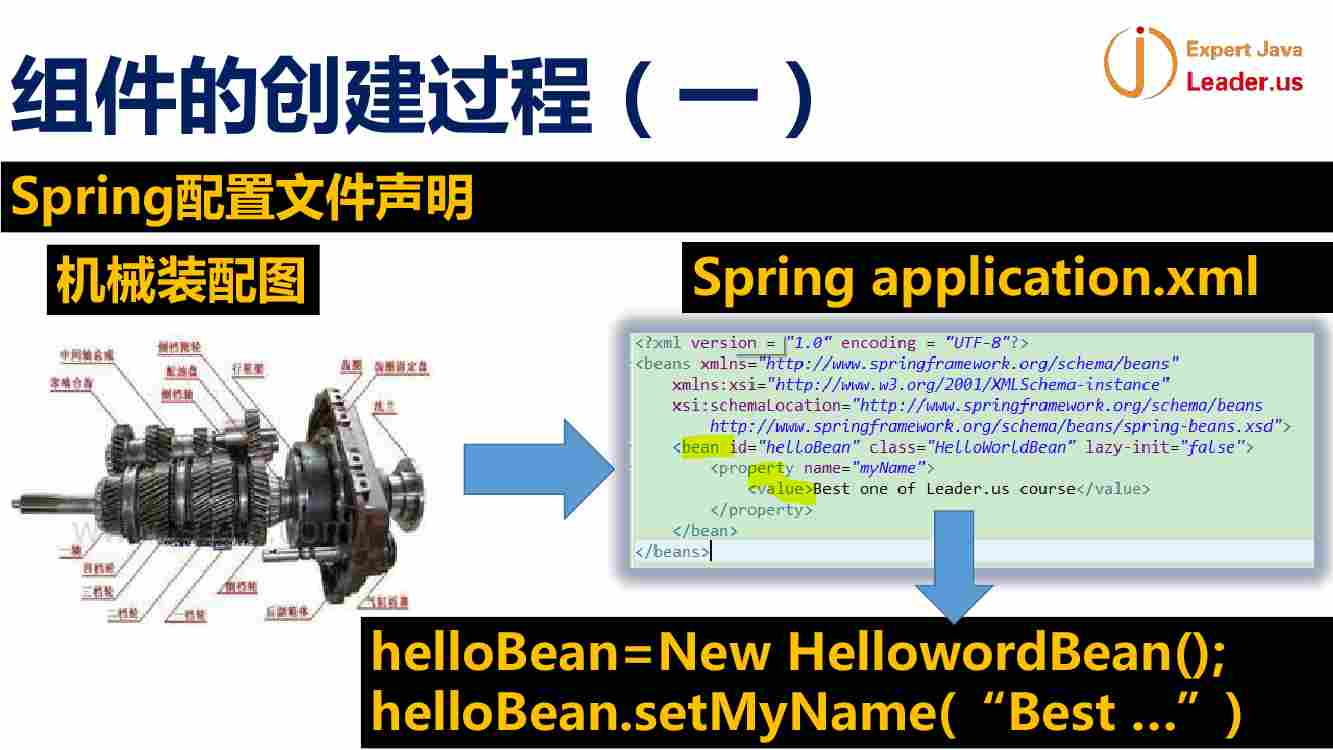
组件是在一个系统中 **长期存在 可以装配的对象**

另外 容器负责组件创建 管理 装配

容器定义API 然后通过容器定义的API 来和组件交互



【这个是如何创建组件 不是 容器 但是 创建组件是容器干的活啊 之前不是说了么。。。 看看怎么】



之前是XML来进行配置

对比机器 复杂机器有装配图 什么组件在什么位置 对应到spring中的就是 application.xml ----- spring中的组件 叫<bean>

上面这个过程定义了helloBean这个组件 --- 也就是spring扫描这个定义之后 立即创建 【默认的lazy-init属性是false ===== 注意 学习单例模式的时候 单例模式说过 如果某一个对象创建非常复杂的时候 为了节省时间 所以 需要才去创建 所以 默认情况都是单例的 所以。。。这个设计的属性是这个意思】

装配了myName的属性

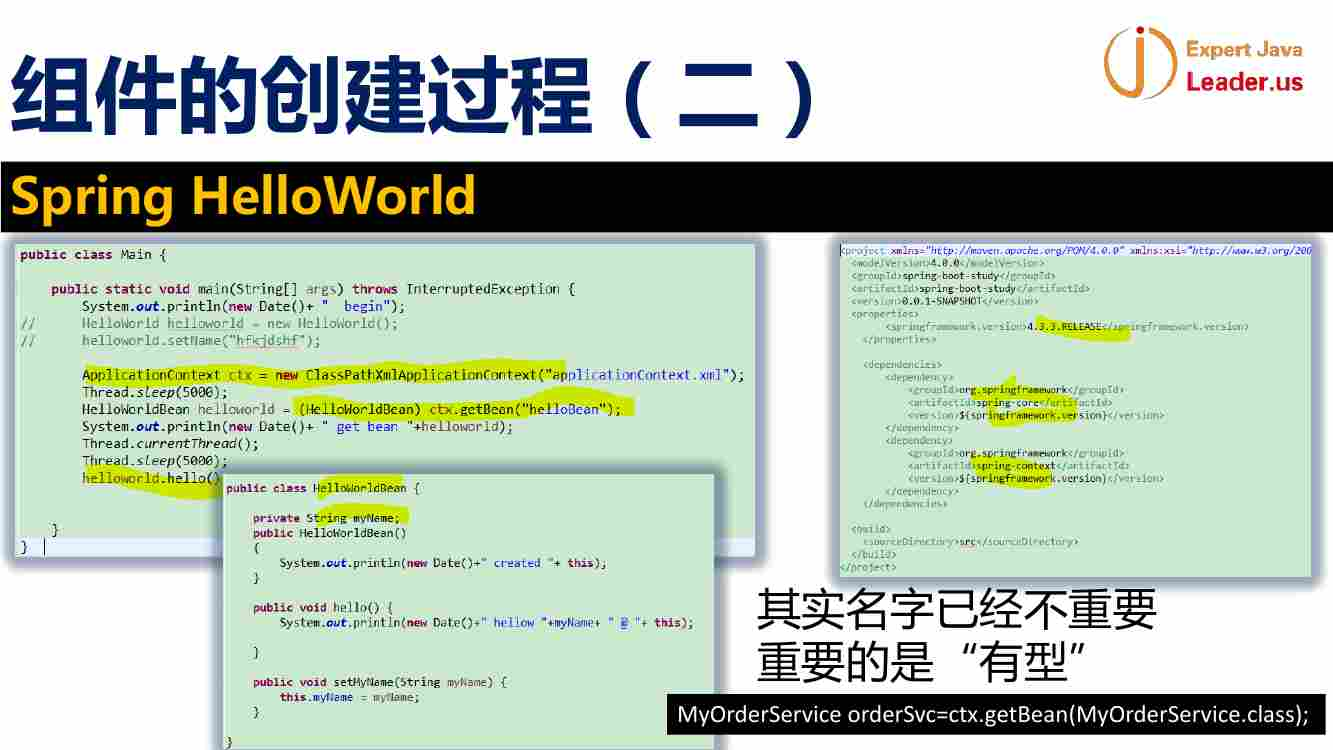
这个XML和代码差不多 没有什么简化

优点在于 我可以不修改代码的情况下 重新进行装配

属性依赖很多 当发生变化 架构重构 这个时候 组件的依赖关系 都会打乱 ---修改代码会非常混乱

但是 修改xml文件 就容易的很多

【所以 我们client 也就是使用人员 为Spring Container提供了创建组件 装配组件的客户化定制 但是 真的创建这些组件 完全是Spring Container来搞定的 不是我们client自己去new】



这个创建过程 依赖的package是spring-core和spring-context

我们用的是4.3

Spring最核心的两个包就是这个

这个过程 实际上是

ApplicationContext ctx = new ClassPathXmlApplicationContext(“applicationContext.xml”);

…

这个ctx最后不用的时候 可以关闭掉【注意 spring学习的时候 或者从计科文那本书上知道 我们ApplicationContext不就是一个更高层的接口么 继承了BeanFactory, ThemeSource, EventListener等等接口的一个Spring容器么 所以 你看看 我们的客户端在new这个容器的时候 这个过程就可以来创建需要立即创建的对象 然后放到容器中供我们调用】

也就是Spring的ApplicationCOntext就是容器 所有的Bean都在这里面 ----- 所有组件bean的创建都记录在了ApplicationContext中

Spring中组件对象不用继承任何接口 ------- 这是Spring最最成功的地方

和以前的老的J2EE框架来相比 以前的老的框架 需要这个bean来继承或者实现自己的接口 非常。。。 这需要理解很多【这其实就可以使用门面模式来理解 Spring让我们的客户端更加易用】 ---- 老的j2ee api增加了学习复杂度 并且有入侵

并且不用学习spring的接口 就可以把spring用好

这是spring最初的用法 并且是xml方式



【ppt写的很好】

不会强迫你怎么样 重构容易

Spring本身是考虑怎么让程序开发更容易 让代码更容易 更简洁

兼容并收 把很多j2ee很多开源框架纳入进来

如果使用了Apache Avalon或者j2ee容器 就是上面的的示例代码片段

你会发现 你实现这样的接口 并且正确理解这个组件体系 先后配置 要把这个本身搞懂 才能正确使用 非常麻烦

之前 j2ee容器是一个黑箱子 如果有bug 你无法证明 你一点办法没有 但是spring全部透明



Xml以前是没有问题的 但是 后来证明是错误的 这是因为

Xml问题 本身是个非常结构化 规范化的语言 ---- xml必须严格规范 schema 有严格的格式 有很多多余的东西 冗余特别多 非常严格 对人本身没有多大价值 但是 对于计算机可以严格遵守 自动化遵守

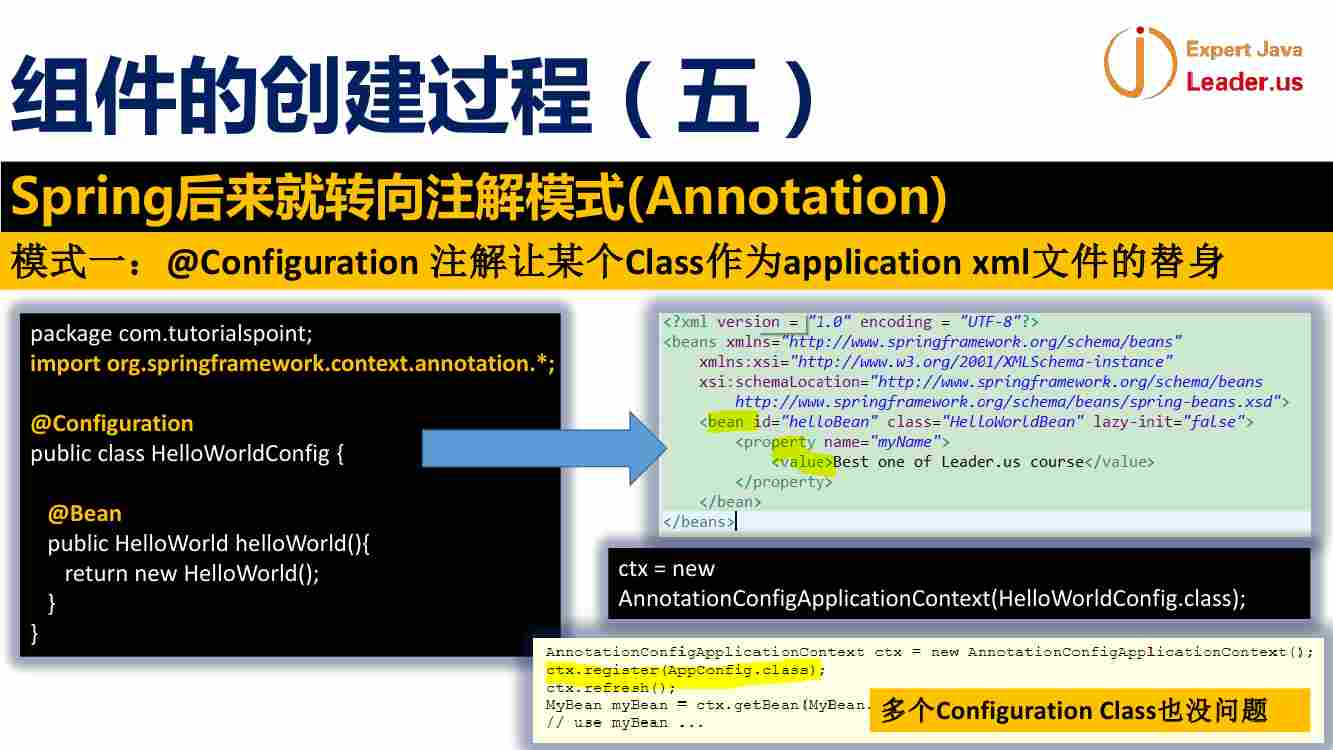
读写xml是非常反人类的 ---- 对程序员更是 所以 有了那句话 “**Designed for Computer use, not for human**”

所以 你搜索 xml错误 会搜错出来各种错误 所以 mycat会遇到各种xml错误 严格意义不适合程序手写

后来转向了annotation jdk也支持

Annotation就是替代xml

Annotation 是程序的一部分 对程序友好 ---- 因为程序就是写这个的



之前在application.xml中写的 全部在java文件中写

这里面还是没有任何的接口继承

这@Configuration 变成了xml的等价体

@Bean就替换了<Bean>

**这样在annotation中 不会出现拼错的问题（因为类文件是存在编译的）**

\*\*Xml的一个优势就是装配 ---- 在annotation中 是没有写装配的变更和依赖的 ---而是通过了一种自动的方式更加强大化

##**一个xml对应一个Configuration Java Bean** ---- 所有的组件 还是在同一个Configuration java bean中的 还是有xml以前的优势的

程序中的时候 配置还是一起的 还是有优势的

从xml到annotation似乎又丧失了xml的优点 ---- xml具有灵活的改动

但是 实际上 成本上一样的 实际上xml的改动和annotation是一样的

程序中 我们的配置还是在一起的 @Configuration 里面还是有很多@Bean

这样 我们的启动方式就变了

定义完成之后 使用了 ctx = new AnnotationConfigApplicationContext(HelloWorldConfig.class);

用这个容器 把class放进去

还可以加载多个class ---- 还可以使用

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(); 【空参数】

Ctx.register(AppConfig1.class);

Ctx.register(AppConfig2.class);

….

多个注册完成之后

最后 要调用 一次

Ctx.refresh(); ----- 这个过程新的bean加入进来 可能堆老的bean产生影响 所以使用refresh之后 就可以自动化做完某些事

做一些背后的操作 自动化的操作

一般不会是多个Configuration 除非是测试 生产情况就是一个

大部分采用注解的方式 然后由一个Configuration标注的class去驱动

这个是一个主流

很多老的系统都朝着注解的方向去构建



不论是注解 还是xml

在xml配置里面 有个 <context:component-scan> 【理解了容器 和 组件 就知道了 ApplicationContext 和 <context： 这个context实际上就是容器的意思 冒号后面的component就是组件的意思 这里面就是容器和组件的意思 ----- 这样我们了解了容器 组件 我们解读一下这个标签的含义

<context:>

两方面理解context为何是Container ----- ApplicationContext这个接口的继承关系知道 这个就是一个容器 负责创建组件 所以标签中的<context:>这个就是容器的行为 ---- 容器要做什么呢？ Component-scan 也就是 扫描base-package的值的包和子包中标记了特定注解的java bean并纳入到容器 ---- 什么叫纳入到容器？不就是由容器来创建 装配 并最后存放在容器中的Java bean么 ----- spring容器中的Bean就是容器中的组件

】

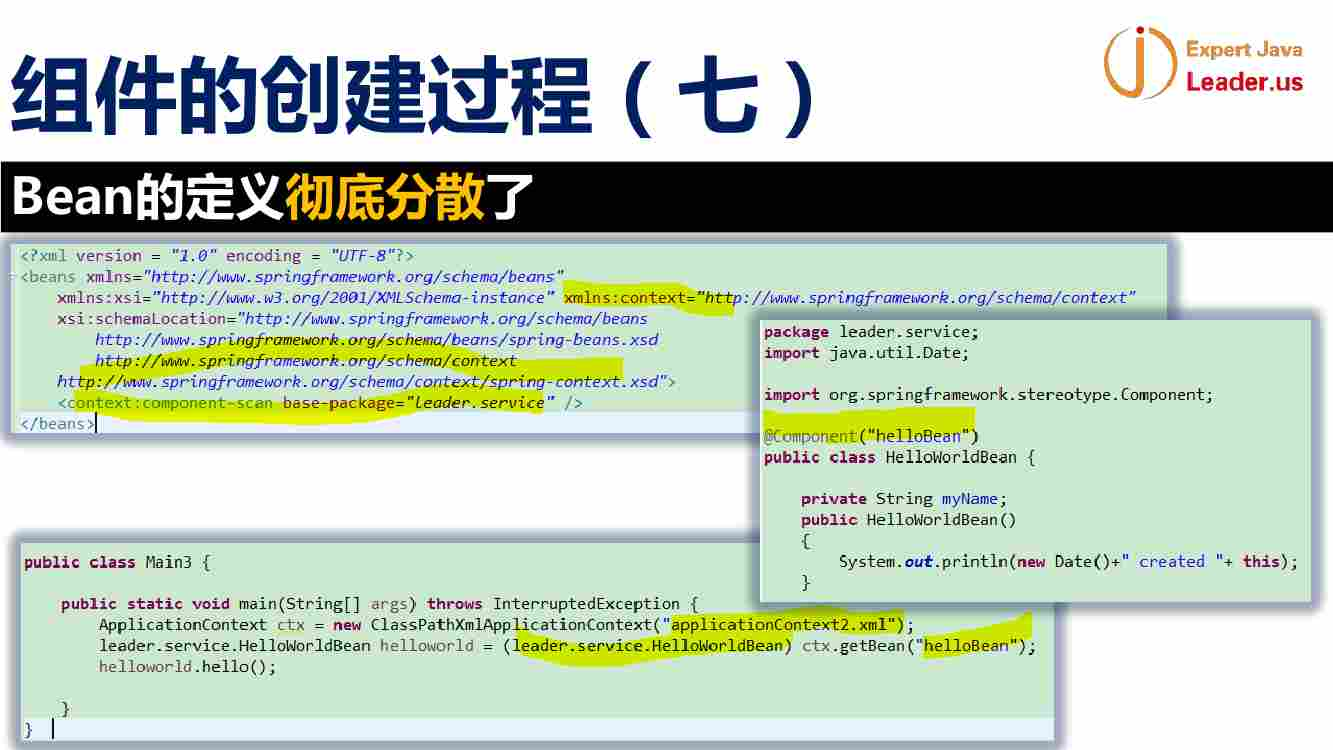
**Component scanning大法：**

无论是使用注解 还是使用xml 最重要的还是 <context:component-scan> 这个

这个就不用集中定义 而是各个地方定义好之后 由这个地方进行集中控制

最重要的就是 使用注解标记好java bean之后 使用这个

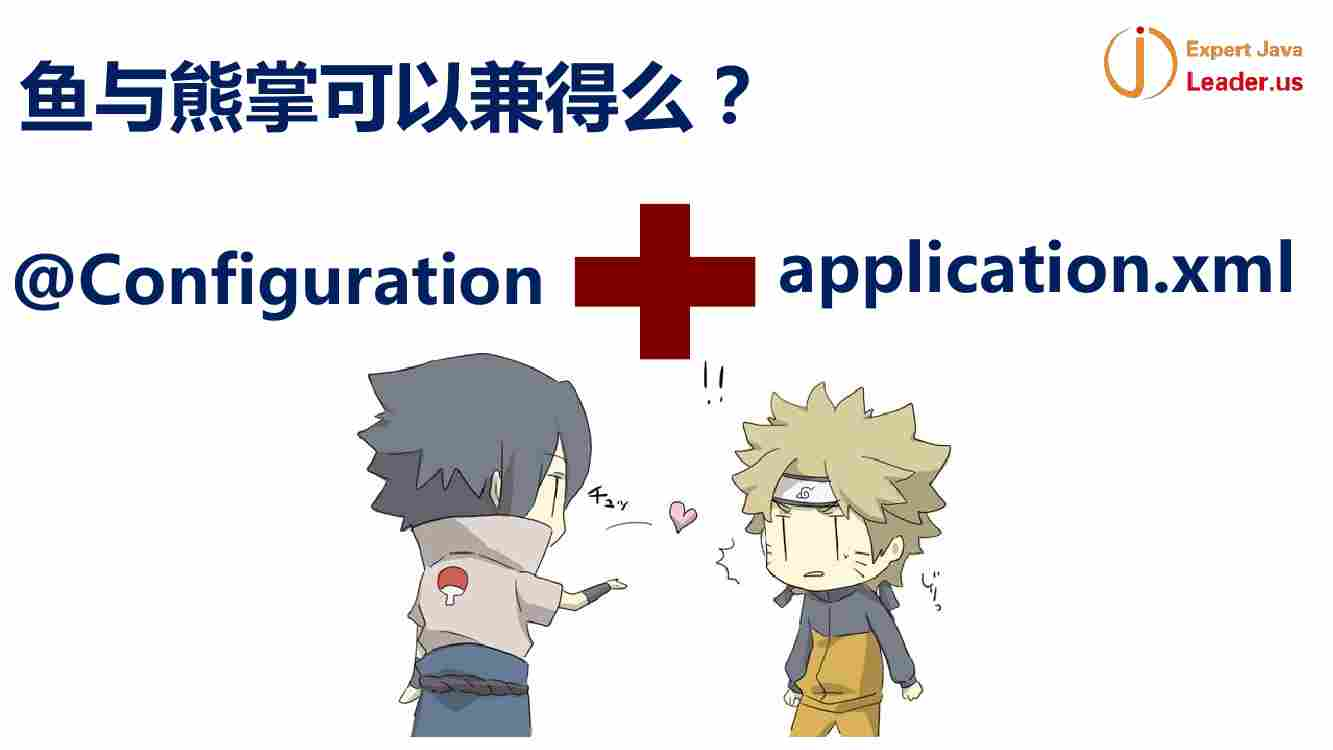
**这个component-scan大法 是在spring3.1中体现出来的**



有了component-Scan这个之后 我们就是要在Java Bean中直接添加@Component就好 然后配置在<context:component-scan base-package=””>中的这个值就能把所有标记为响应注解的component纳入到我们的spring容器中  
 这个就表明这个是Bean 直接加载它 这样 我们的xml文件就是不变了【这个里面我认为 实际上是不变的xml文件 和 变化的annotation进行配合】

客户端的使用方式还是没有改变的

这种情况 xml是没有价值的 原来重要的定义bean的地方没用了



Spring熟悉的人 xml很习惯 但是 新人喜欢annotation 这两个能和平共处？

实际上 @configuration + application.xml是可以和平共处的



使用两个注解 @Configuration + @ImportResource 【一个就是纯纯的注解 另一个注解来标记xml文件】

Spring的核心代码中 还有注入其他的Configuration的做法

这样混搭xml和注解 一起来做

一部分是xml 另一部分annotation

Spring注解主流 辅助部分 使用xml定义

------- 这里面提出的问题就是 如何在xml中开启注解@Configuration？【老师让回去试试】

可以xml启动 然后 引出@configuration

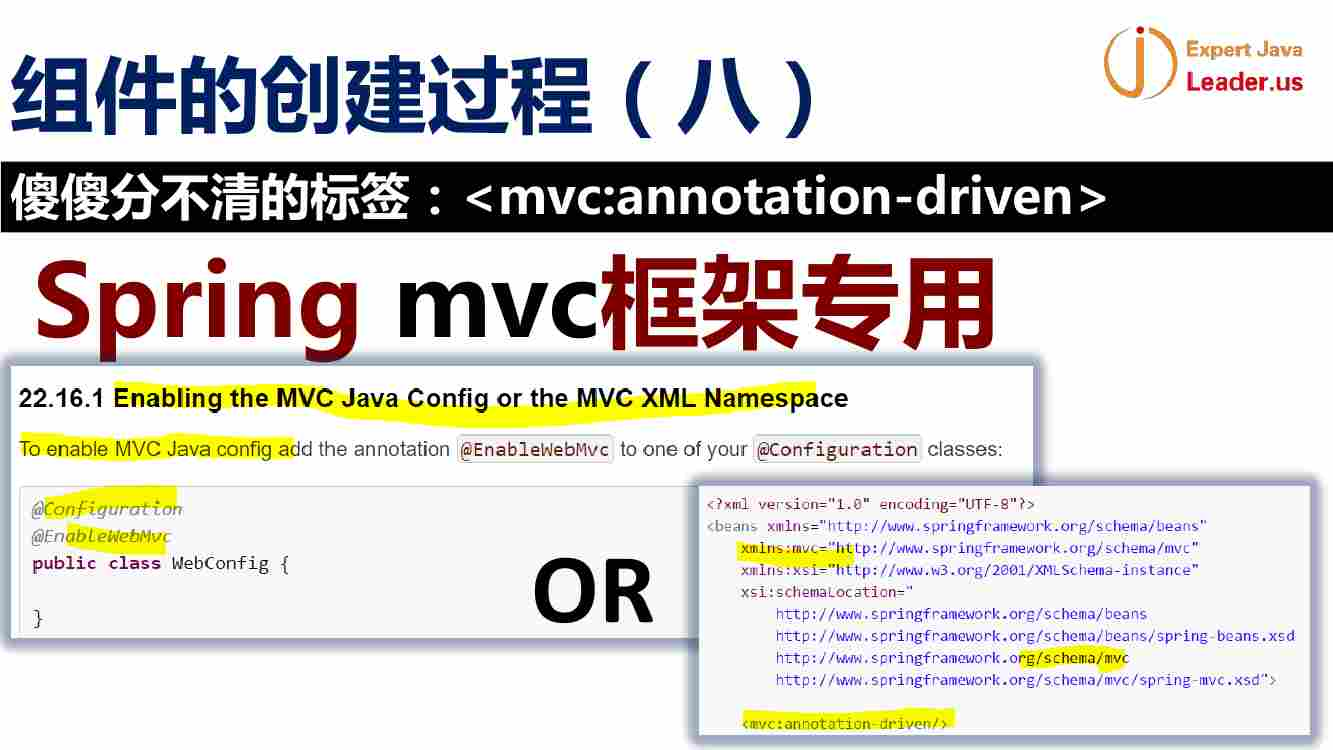
这种混搭模式更加适合在程序的迁移的时候 --- 从xml迁移到annotation 可以采用这种混搭模式 一部分使用xml 一部分迁移的使用注解



注解的方式很强大 xml中很多都可以被注解替代 spring2以后就强化这个

刚才的<context:component-scan>在注解也是存在的 就是@ComponentScan(“xxxxx”) 非常强大 如果你是新的项目 完全可以直接使用注解来完成整个项目

鼓励的方式 官方的方式 就是使用注解来替换xml



Spring中非常麻烦的地方就是提供了很多很多注解 增加了很多类 分不清做啥

<mvc:annotation-driven> 整个是mvc专用的

也就是说 整个配置等价于 @EnableMvc整个注解



Web开发的时候 有很多问题

数据绑定问题 就是前端传递过来的数据 如何转成java bean

参数校验 格式转换问题 虽然没有什么技术含量 但是 天天要做 所以 spring都提供了支持

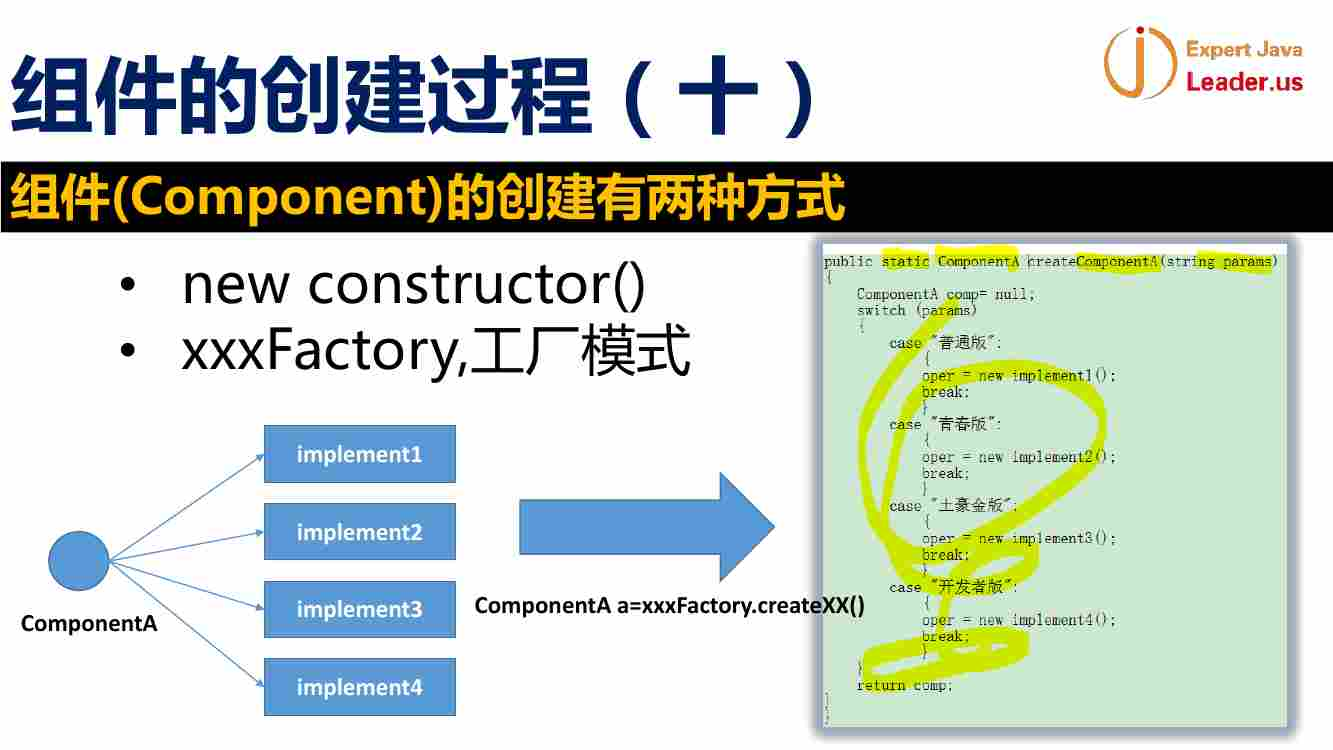
Ppt上面就列举了各种各样的注解的支持web开发【直接看pdf就好】

【可以看到 老师在做ppt的过程中 最大的特点 不是 说我的PPT真的很漂亮 而是非常恰如其分的在合适的位置 加上了合适的动画小人配上一定的话 非常经典的逗乐pdf】

也就json 和string都是支持的

看一遍这个ppt上面写的

Spring的好处 就是学会之后 会提高你的工作效率 实用的工作技能



工厂模式 需要被掌握

基本上就是一个接口 有很多种实现 然后 需要其中的一种 这个构造过程很麻烦 这个过程可以使用Factor.createXxxx()创建 把一个对象的过程封装到内部 ----然后 随意可以替换 这个时候 不影响你的客户端的使用

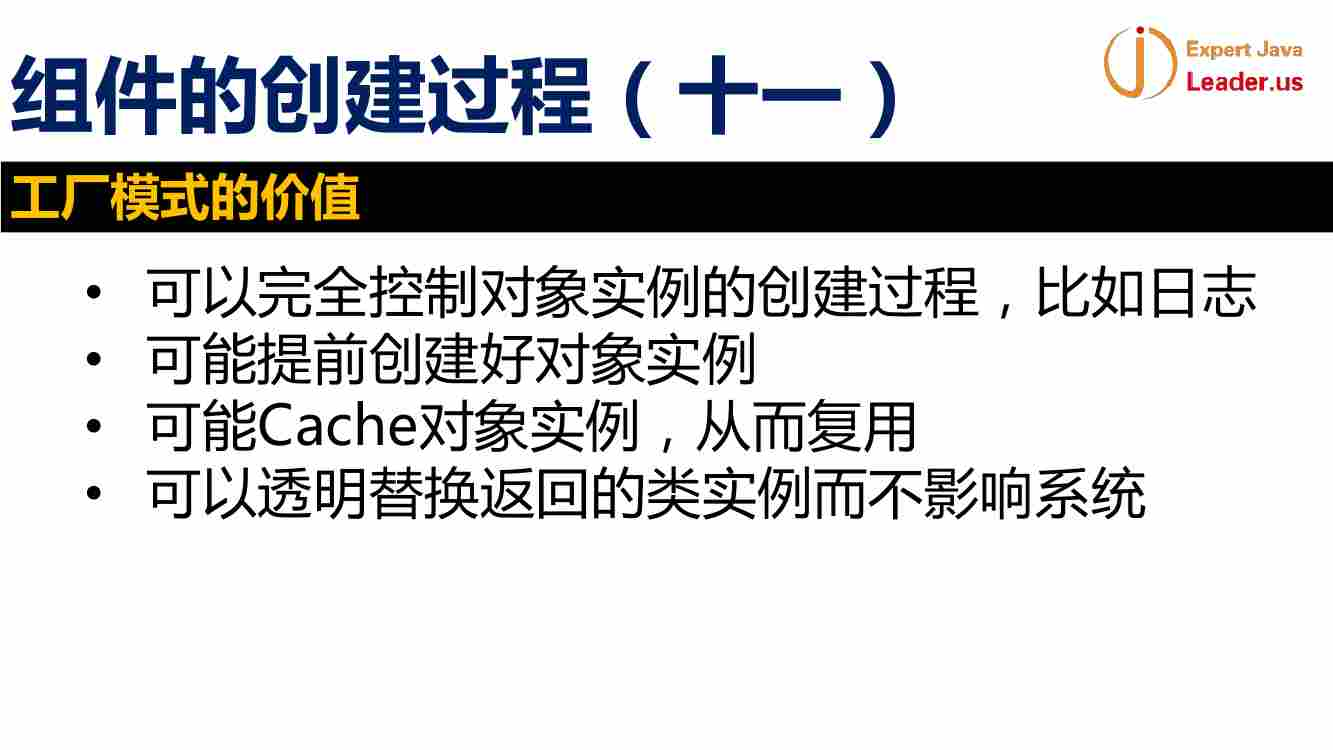
举个例子 【这个例子里面就是简单工厂】

通过参数的方式 来决定到底是使用哪一种实现

这样构造细节局限在内部 使用者不是知道这个细节 但是使用者可以通过参数来

Spring中在Factory模式下还要装配一些数据 这些数据才能使用 ---- 所以 可以理解整个Spring就是一个factory【就是这样啊 Application继承的一个元老级别的父接口 **就是 BeanFactory**】

建议大家 找找工厂模式的类图 正确掌握工厂模式的类图 优缺点 这是最常用的模式



第三点中的 可能Cache对象实例 ---- 从而复用

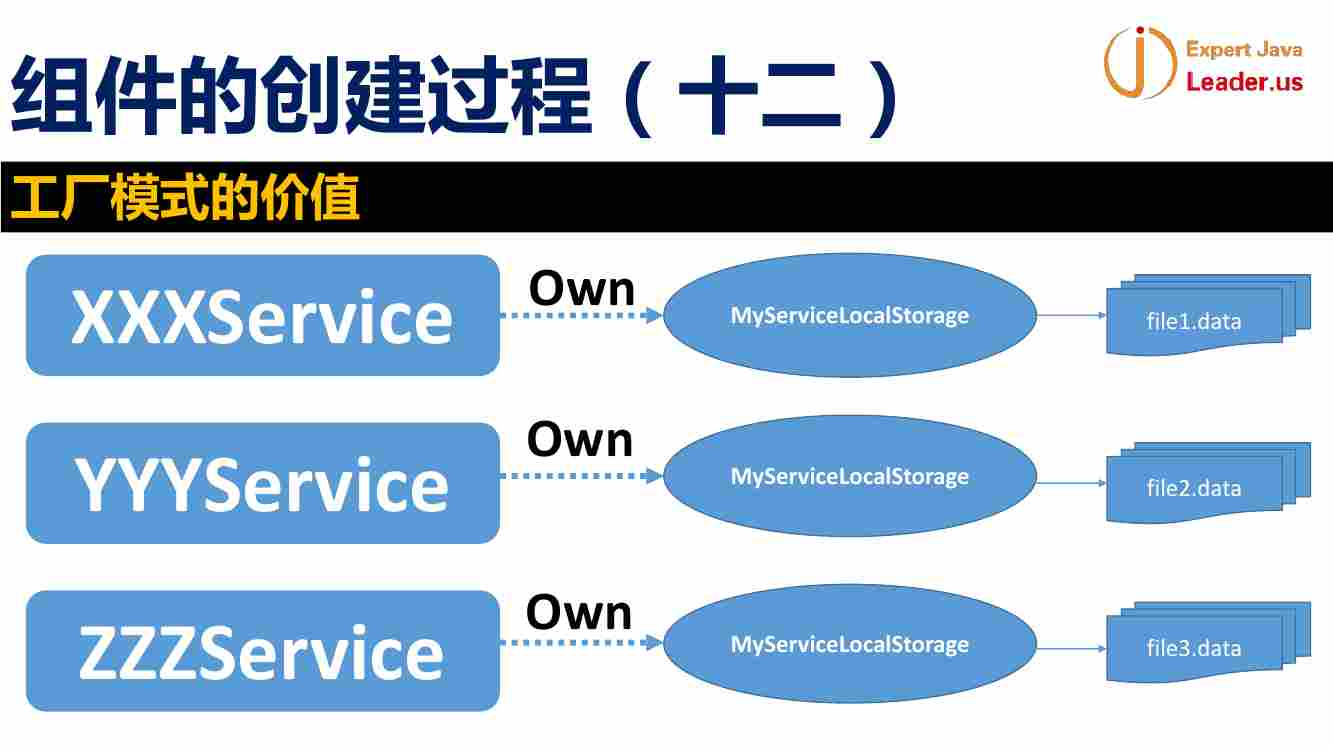
整个复用有多个级别的

ThreadLocal复用

对象池级别的复用

最后一个的“可以透明替换返回的类实例 而不影响系统”原来某个类是不适合的 新版本替换掉 整个系统感知不到

所以 Cache + 透明替换 是非常有价值的【替换整个 回去自己理解理解】

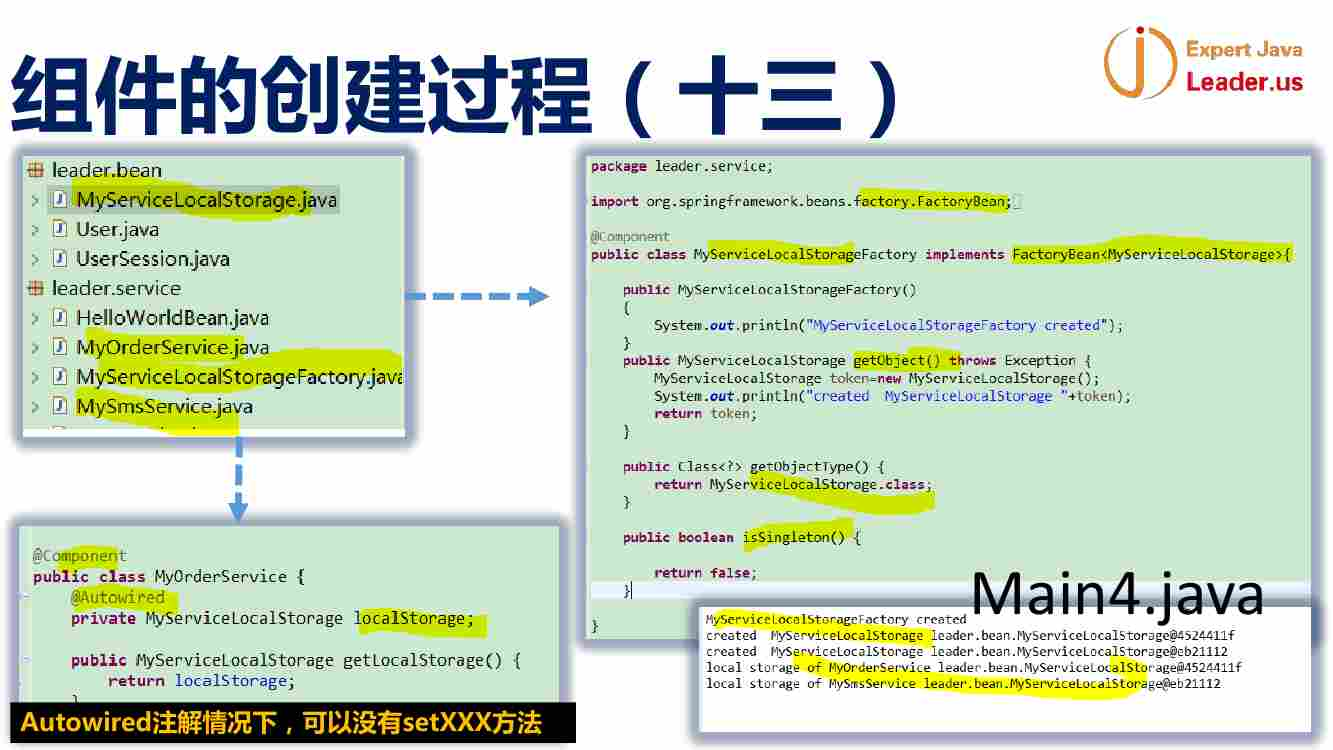


看下这个例子

三个service 每个service都由自己的storage 每个storage里面有自己装配文件 【这个说明的就是 我的每一个XXXService YYYService ZZZService持有的localStorage一定不是同一个对象 是各自的 所以 一定不是单例的】

【这里想使用更为灵活的FactoryBean来生成这个MyServiceLocalStorage】

需要三个不同的Storage bean实例。。。。 这个时候 可以使用Factory工厂

[]

GetObject返回一个对象

IsSingleton返回true 对应prototype 所以 这个spring中的工厂factoryBean既可以创建单例的 也可以创建多例的

这个Storage是@Component 就会被MyOrderService的@Autowired装配上

Spring知道某个Factory创建什么Bean

【这块在Spring源码分析中给出的是 xml配置 不灵活 希望能给出在代码中的灵活配置 ----- 这个就是工厂组件

那么如何装配 是否是单例 等等 都在FactoryBean这个接口中定义好了

Interface FactoryBean<T>{ //泛型T表示这个工厂Bean要创建的对象的类型 ----- Bean在Spring容器中就是组件的意思 所以 这个时候 通过Bean就应该获得这个工厂生产的产品

Public T getObject() throws Exception;

Public boolean isSingleton();

public Class<?> getType();

}

这样 通过这个工厂类 就可以生成这个工厂对应的产品T

====== getObject()不是获取这个工厂实例本身 而是获取这个工厂生产的产品 -------- 实际上 我认为这就是**工厂方法设计模式**

public class Car {

private int maxSpeed;

private String brand;

private double price;

//setter getter

}

Public class CarFactoryBean implements FactoryBean<Car>{

Private String carInfo;

Public Car getObject() throws Exception {

Car car = new Car();

String [] infos = carInfo.split(“,”);

Car.setBrand(infos[0]);

Car.setMapspeed(Integer.valueof(infos[1]));

Car.setPrice(Double.valueOf(infos[2]));

Return car;

}

Public Class<Car> getObjectType(){

Return Car.class;

}

Public boolean isSingleton(){

Return false;

}

Public String getCarInfo(){

Return this.carInfo;

}

Public void setCarInfo(String carInfo){

this.carInfo = carInfo;

}

Public String getCarInfo(){

Return this.carInfo;

}

}

这样 原本 我们在XML中配置的car的配置如下：

<bean id=”car” class=”com.test.factoryBean.CarFactoryBean” carInfo=”超级跑车,400,200000” />

这样 我不用嵌套配置 就很方便的配置出来这个FactoryBean需要的实例的配置

否则 我就需要配置成如下：

<bean id=”car” class=”com.test.factoryBean.CarFactoryBean”>  
 <property name=”maxSpeed” value=”400” />

<property name=”price” value=” 200000” />

<property name=” brand” value=” 超级跑车” />

</bean>

这个显然没有上面的配置灵活 ----- 这个就是使用工厂模式的好处

这个例子里面 显然使用的是注解的方式 @AutoWired直接进行装配

FactoryBean这个接口中的getObject就相当于工厂方法模式中的接口 getProduct()方法

从上面的例子可以看出：

最先是MyServiceLocalStorageFactory先创建 然后是 getObject被调用 也就是对应的产品被创建 ---- 最后是 引用了这个LocalStorageService的MyOrderService和MySmsService就会被创建出来

】

可以看到 这个LocalStorage的hashCode不一样 就是创建了多个



延迟加载的用途 ----- 我们开发一个系统非常的大

测试的时候 或者启动的时候 仅仅需要创建部分 有些资源不具备

要测这部分 没关系

所以 lazy-init 就是这个bean被访问的时候 才会被创建 否则不会创建 ----- 这个就是单例模式的延迟加载

这个default-lazy-init 属性使用@Lazy标注出来

这个是全局Lazy的意思

----- 老师给出了一个问题 就是我们的FactoryBean是不能被Lazy的 ----- 思考为什么？

这时候Bean被创建的两种方式 ---- 程序的代码有getBean 或者这个Bean被依赖

所以 如果设置了default-lazy-init为true的时候，如果不具备代码中显示调用了getBean或者Bean被依赖 这个bean 组价永远不会被创建！！！！！ ----- 所以 有时候 有些功能不正确的时候，可以看看 是否是这个bean组件的default-lazy-init被设置为了true ---- 陷阱



组件的作用域

组件全局唯一的 单例模式 定义一个就是一个 被其他引用 大家share同一个

Prototype ----- 每次调用的时候 就创建一个新的 new一个新的实例

这个实际上用于有状态的组件 ----- 不会被shared的

前面的LocalStorage使用了FactoryBean来做 和这个prototy一致

记录有状态的

Web应用中

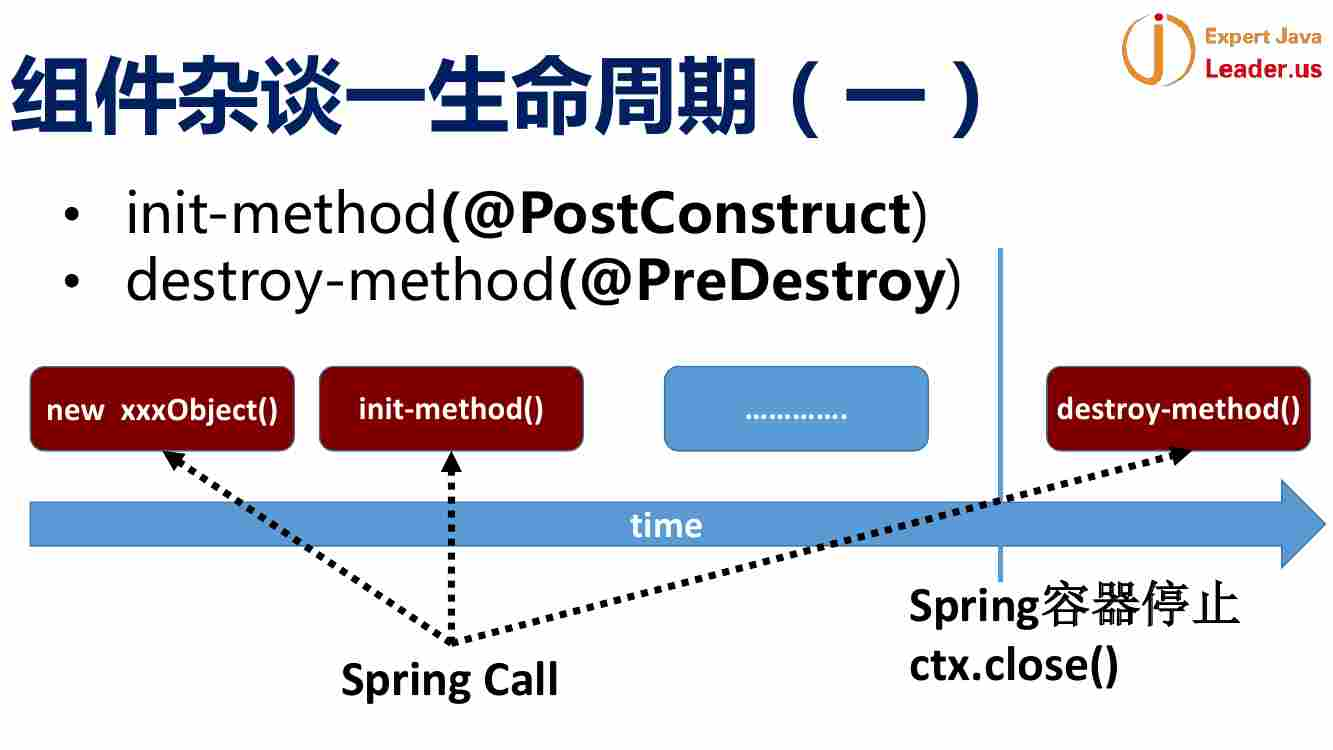
Portlet现在已经过时了 所以 这个global sesssion就不再使用了

session 就是一个用户单独的会话中注册出来一个单独的bean组件 ----- 购物车做成一个bean 就是适合在用户的一个session中创建的一个bean

request是每次请求的时候 生成一个java bean

每次点击页面 多次请求 就会有多个对象创建出来

------ 这个生成的太多 严格来说都不算组建了



组件的生命周期 组件先创建出来 new出来

然后初始化 Init-method =-====== @PostConstruct

然后使用这个组件

最后销毁 destroy-method =========@PreDestroy ----- 当容器SpringContext关闭停止ctx.close()的时候 都会调用对应的destroy-method

【看到没 创建 初始化 和 销毁的三个红框 写的都是Spring Call ------ 也就是应验了上节课的 组件的创建 初始化 和销毁 都由容器来保和维护 ----- 但是 唯一不同的就是篮筐 就是 我的组件的使用 是直接暴露给客户端调用的！！！！！！

**所以 容器为client使用组件提供了非常便利的条件**

客户端**使用**组件Component

**剩下的Component的管理 客户端不需要去操心**

**容器负责对Component的新建 初始化 和 销毁**

**实际上 可以看成是门面模式 而已 Spring的ApplicationCOntext为Client提供了一个统一的ApplicationContext的界面**

】

Init如果抛出异常 这个bean就会被通知不可用 这个时候整个容器会有问题

---- 如果 你要求你的bean初始化发生异常 你的工程不可用 就要在init发生异常的时候 抛出异常 此时工程停止

否则 init异常 也要求工程能运行 这个时候 你就只能打印日志 不能抛出异常

那么 我们可以用注解的方式 init-method 对应的就是 @PostConstruct ----- 表示我的初始化 是在构造方法之后运行 正符合Spring容器管理component的时间顺序

@PreDestroy 就是在我的对象销毁之前 我要做清理 所以 是Prexxxx

这两个注解是Java中的标准注解



如果有Java标准注解 就用Java标准的 没有 采用Spring的

Java中的标准注解和Spring是有一定的差距 后面说

【PPT中的服务器加载Servlet ---- 表示 我认为是ClassLoader加载这个类的字节码】

PPT上面的解释：

@PreConstruct和@PreDestroy 是Java EE5（EE不是SE）中的Jsr-250进入的 ----- 但是 这些非j2ee的jsr-250的注解被加入到J2**se6**中（Generated PostConstruct PreDestroy Resource---- 表示依赖关系 等价于Spring中的AutoWired一个文件 Resources --- 依赖多个文件） ----- ee和se本来是独立的 但是 这里面定义重合了 这样jdk6中 就可以用到了这几个和组件声明有关的注解

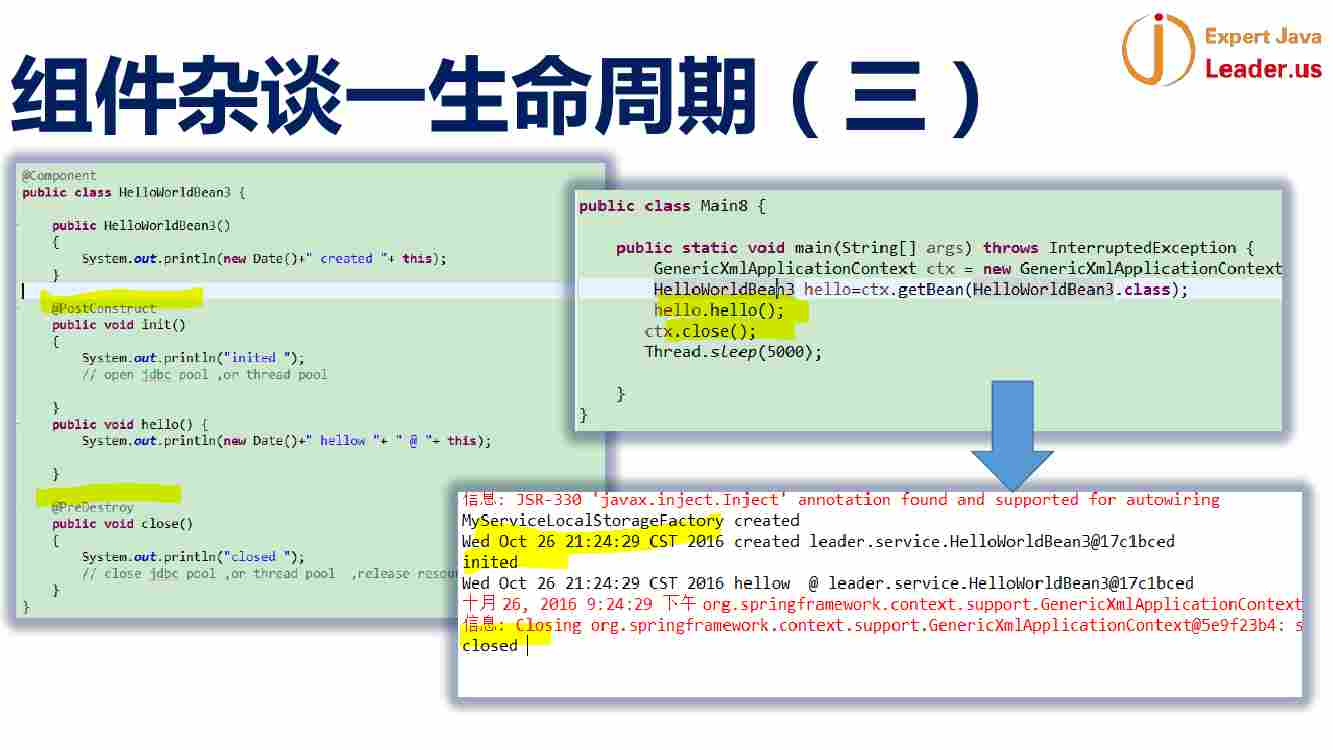
Spring也要遵守JSR-250的规范 ----- 所以 也支持 @PreDestroy, @PostConstruct, @Resource这些注解 ---- spring可以用这5个注解 这5个注解在spring中有等价的注解

在ee中 上面的五个注解 可以用在定义Servlet ---- Servlet需要实现接口 ---- 现在可以用这些注解

现在Servlet的开发已经不用了

但是 要知道 在j2se 6中 可以使用之这五个和组件相关的注解

在Spring中 同样可以使用这个5个相关的注解（只是spring中有对应的替代）



有一个例子

我们使用上面的两个注解

@PostConstruct 和 @PreDestroy 这两个注解 定义完这个组件的时候

【实际上 我们使用这两个注解 + @Component

首先@Component就表示这个应该是Spring容器中的一个组件

通过PostConstruct和PreDestroy标示了 我这个组件的生命周期哪个阶段对应的哪个方法 实际上这个组件的生命周期就被客户换出来了

Main方法中 直接使用

】

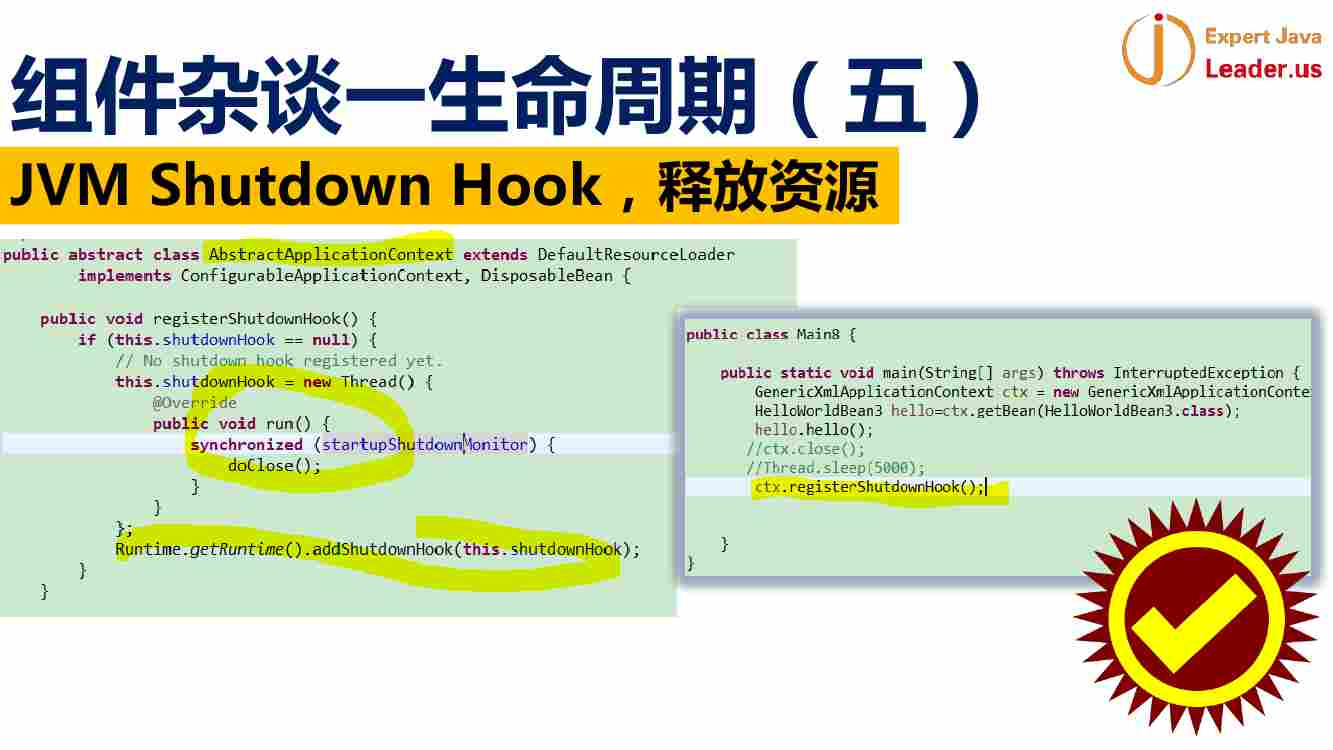
打印的就是 创建 初始化 调用 和 销毁

当你在使用这个方法之前 会调用init 保证组件装配的好 当spring停止的时候 对应的资源释放



加了@Scope(“prototype”) 之后 每次调用这个组件的时候 都会实例化一个新的组件 两个不同的bean

打印可以看到 不同的两个对象的hashCode



Spring还有一个重要的知识点 就是使用了JVM中的Shutdown hook -----释放资源【hook不就是需要回调的钩子方法么】

如果 用kill xxx 的时候 JVM收到shutdown信号 可以告诉spring释放资源

Spring在AbstractApplicationContext中有一个方法 registerShutdownHook方法

这个shutdownHook实际上是Thread ----这个代码里面 run中核心就是调用SpringContext的doClose();

因为run方法本身不是同步的 所以 这个时候 必须使用同步代码块 来关闭这个SpringContext

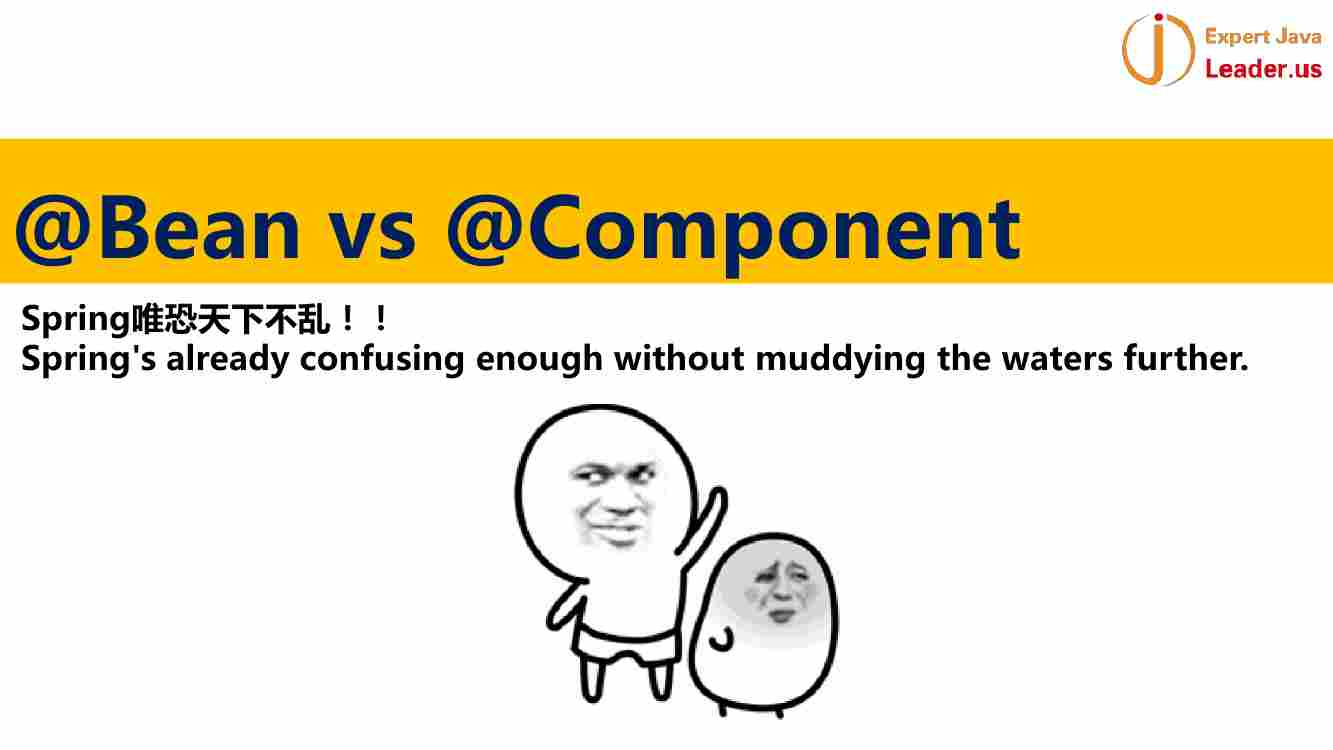
这个进程就创建完了

然后 就调用了Runtime.getRuntime().addShutdownHook(this.shutdownHook); 【学习一下Runtime的方法】

注册上【注册 实际上就是观察者模式】

自己可以写这种程序 然后写一个ShutDownHook

所以后面的main方法调用的时候 就调用了 ctx.registerShutdownHook(); 这样就注册了一个shutdownHook(老师没有说的很清楚)

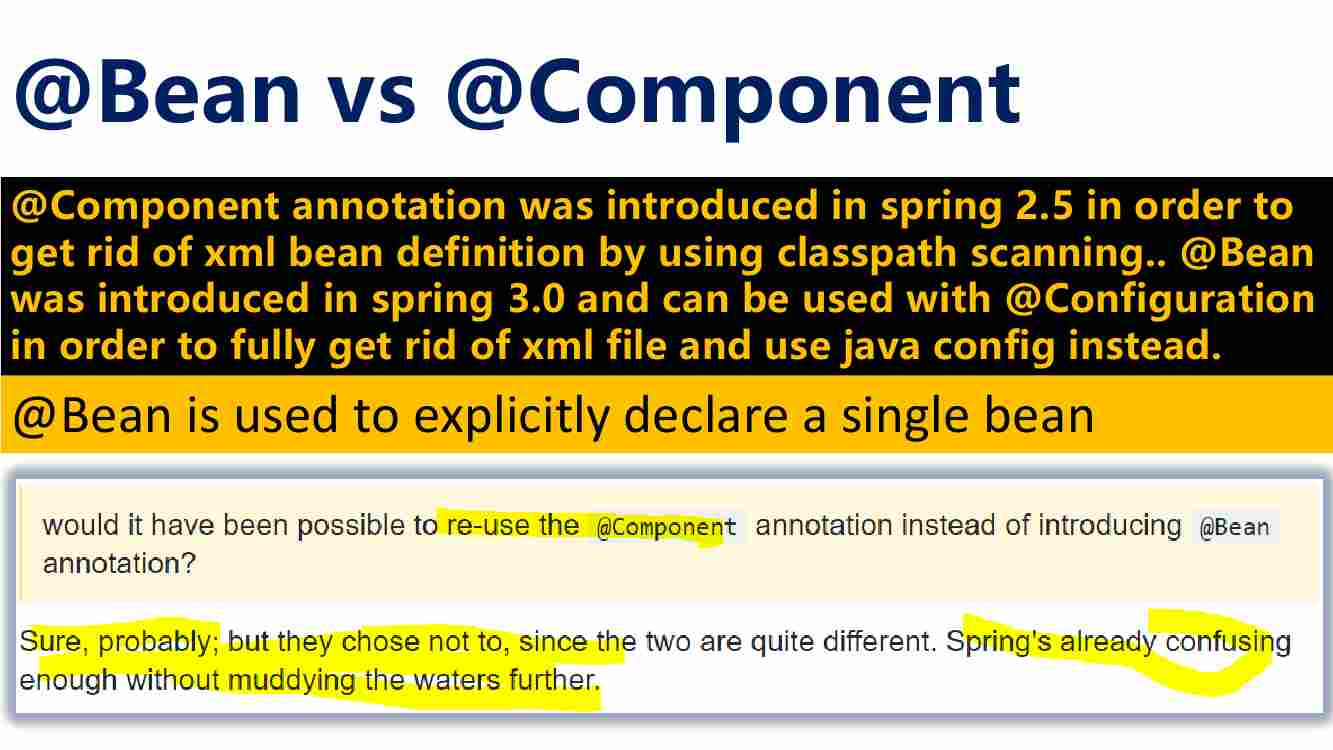


我们开始的时候 配置都是Bean 每个组件都是Bean

但是 Bean的注解是以后产生的 之前都是Component

----- spring这个技术已经难以精通性的 掌握 这个过程一直加新东西 很少有人能把这个跟进去

【是啊 Component本身就是组件 Bean在Xml也表示组件 哪个？ 语义上】



Bean是在Component之后引入的 [2.5有了@Bean就想把XML彻底干掉](mailto:2.5有了@Bean就想把XML彻底干掉) ------- 2.0开始引入注解 2.5希望 全部用classpath和context:component-scan 来做

有一个问题

Xml中有一些bean是不被装配的 只是声明这是一个Bean，被其他所引用 自身不会被Spring autowired装配

但是Component就不是这样 不仅定义这是一个Bean ---- spring还要为这个component装配所有的依赖

Component即作为bean的承担着 一被别人依赖 还要依赖别人 就可能引入死循环 用好了不会

Spring为了避免这个

Bean不会管自身的依赖 不能说这个bean 依赖 只能说创建出来一个bean 被别的Bean依赖

这个就是 Single Bean ---- spring不给装配 这是Bean和Component的区别【可以继续查查】



Spring中的事件 可能对我们有关系 也可能没有关系

这个跟我们有关系

如何获取一个Context对象？

得到ApplicationCOntext 意味着 可以从Context中获取任何的bean

不用特别住强制继承 或者实现什么接口 仅仅使用@EventListener来注册一个事件 一旦这个事件触发 就会获取这个Context

【个人了解一下这个注解 就是 ：Listener就是观察者 观察到某一个时间发生 也就是Subject的发出新的内容的时候 通知观察者 这样观察者调用自己的回调方法 setApplicationContext(ContextRefreshEvent eve) ------- 这个ContextRefreshEvent就是Subject发回来的数据 】

【了解spring中的事件 估计是观察者模式 ----- 之前我分析过这个】

【可以了解一下张龙讲的Swing编程的中的观察者模式】

上面的代码 就能够通过静态的方法获取Spring容器中的一个bean

【看一下这个ContextRefreshEvent 容器刷新事件 ----- 容器刷新了 那容器不就是变化了了么 这个时候 subject会发出一个数据给监听者 监听者可以通过这个数据获取到想要的ApplicationContext】

这个工具类可以解决很多问题 【试试】



加载也是一个分厂复杂的过程 Spring在这里面做了很多的工作。。。。