spring笔记

1：

core ioc orm框架 mvc dao（jdbc事务）

web context（application context）

1，Spring Core

Core模块是Spring的核心类库，Spring的所有功能都依赖于该类库，Core主要实现IOC功能，Sprign的所有功能都是借助IOC实现的。

2，AOP

AOP模块是Spring的AOP库，提供了AOP（拦截器）机制，并提供常用的拦截器，供用户自定义和配置。

3，ORM

Spring 的ORM模块提供对常用的ORM框架的管理和辅助支持，Spring支持常用的Hibernate，ibtas，jdao等框架的支持，Spring本身并不对ORM进行实现，仅对常见的ORM框架进行封装，并对其进行管理

4，DAO模块

Spring 提供对JDBC的支持，对JDBC进行封装，允许JDBC使用Spring资源，并能统一管理JDBC事物，并不对JDBC进行实现。（执行sql语句）

5，WEB模块

WEB模块提供对常见框架如Struts1，WEBWORK（Struts 2），JSF的支持，Spring能够管理这些框架，将Spring的资源注入给框架，也能在这些框架的前后插入拦截器。

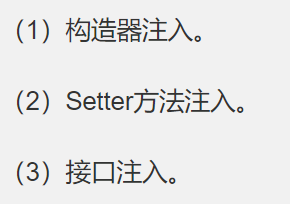
6，Context模块

Context模块提供框架式的Bean访问方式，其他程序可以通过Context访问Spring的Bean资源，相当于资源注入。

7，MVC模块

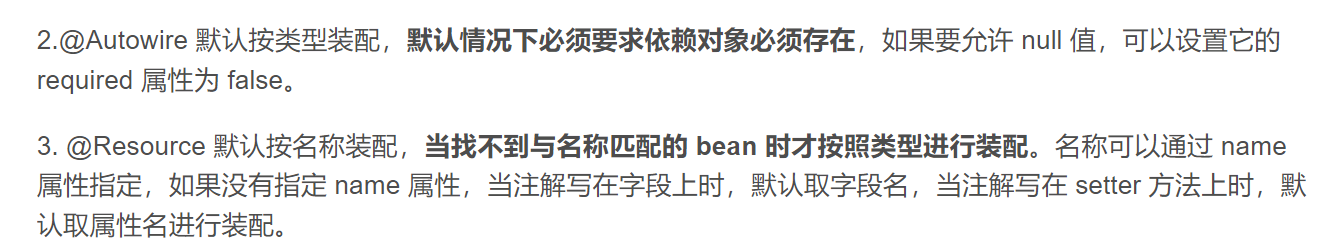
WEB MVC模块为Spring提供了一套轻量级的MVC实现，在Spring的开发中，我们既可以用Struts也可以用Spring自己的MVC框架，相对于Struts，Spring自己的MVC框架更加简洁和方便。

2：依赖注入的方法



构造器注入 setter注入

基于注解的注入 autowire和resource



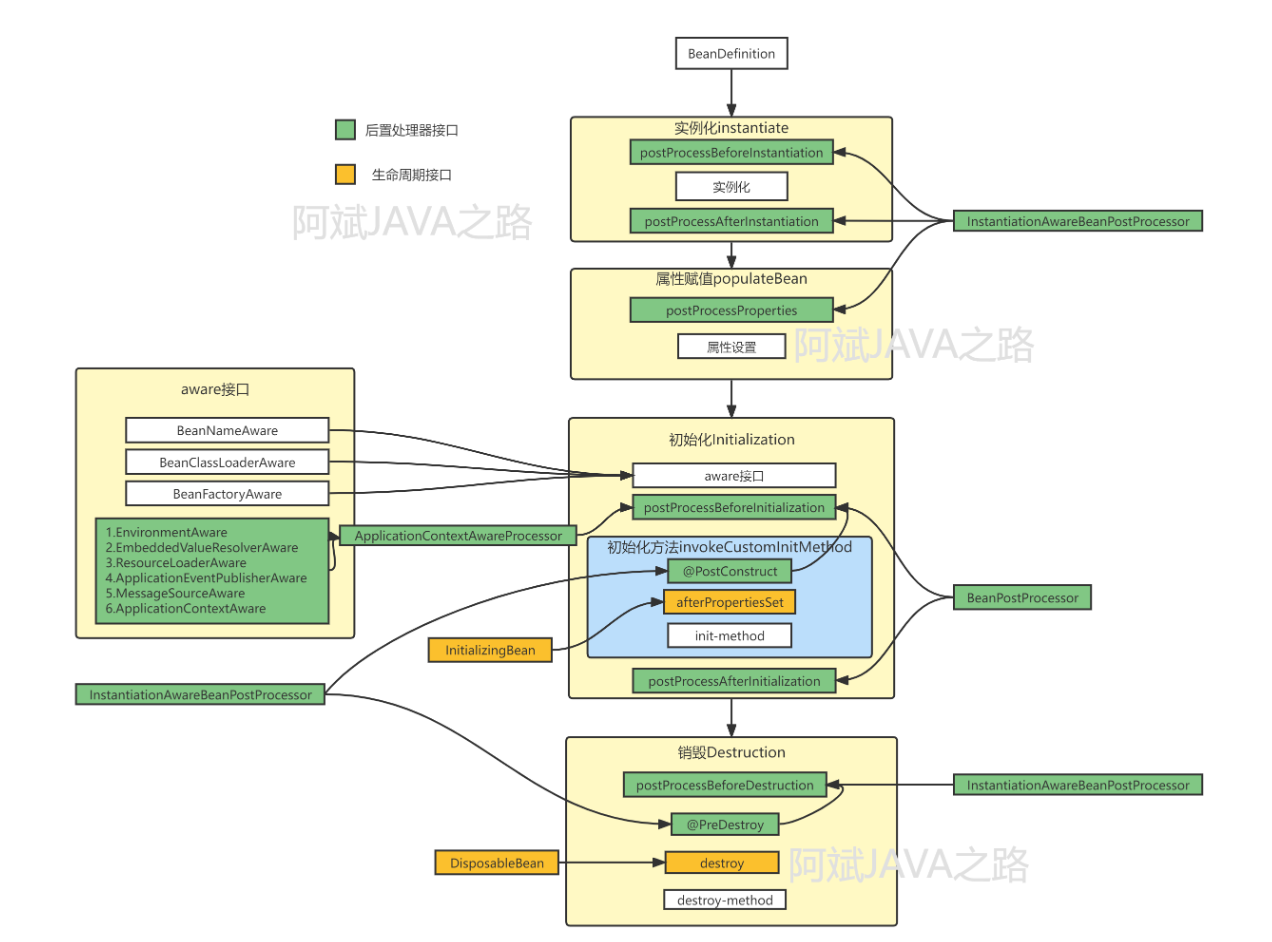
3：beanfactory 和applicationcontext的区别

前者是最顶层的接口， 就很简单的getbean定义了一个规范，

applicationcontext在实现的过程中 还实现了 单例缓存， 配置xml加载 解析 获得beandefinationMap 的功能； createBean（） 执行beanfactory beanpost init destroy 等一系列的方法；

是正经提供给开发者使用的；

4：



1、根据配置情况调用Bean构造方法或工厂方法实例化 Bean。

2、利用**依赖注入完成Bean中所有属性值的配置**注入。

3、如果Bean 实现了BeanNameAware 接口，则 Spring调用Bean的setBeanName()方法传入当前Bean的id值。

4、如果Bean实现了BeanFactoryAware 接口，则 Spring 调用setBeanFactory()方法传入当前工厂实例的引用。

5、如果Bean 实现了ApplicationContextAware 接口，则 Spring调用setApplicationContext()方法传入当前ApplicationContext 实例的引用。

1. 如果BeanPostProcessor 和Bean关联，则 Spring将调用该接口的预初始化方法postProcessBeforelnitialzation()对 Bean进行加工操作，此处非常重要，Spring的AOP就是利用它实现的。
2. 如果Bean实现了InitializingBean接口，则 Spring将调用afterPropertiesSet()方法。

实例化->生成对象->属性填充后会进行afterPropertiesSet方法

1. 如果在配置文件中通过 init-method属性指定了初始化方法，则调用该初始化方法。

bean里面自己的方法；

1. 如果BeanPostProcessor和 Bean关联，则 Spring将调用该接口的初始化方法、 postProcessAfterlntialization()。此时，Bean已经可以被应用系统使用了。

10、如果在 中指定了该Bean的作用范围为scope=“singleton”，则将该Bean放入Spring IOC的缓存池中，将触发Spring对该Bean 的生命周期管理;如果在中指定了该Bean的作用范围为scope=“prototype”，则将该Bean交给调用者，调用者管理该Bean的生命周期，Spring不再管理该 Bean。

11、如果Bean实现了DisposableBean接口，则 Spring 会调用destory()方法将Spring 中的 Bean销毁; 如果在配置文件中通过destory-method属性指定了Bean的销毁方法，则 Spring将调用该方法。

bean的生命周期：

1：解析classpath里面的beanid beanname 类名，加载出来 beanDefination的map

按理说应该还有个beanFactoryPostProcessor 我见过的

2：

进行实例化； 完成依赖注入， 对beandef里面存的 beanref引用 ，依次调用，这个阶段会出现循环依赖的问题；

3：

判断bean上 是否实现了各种 aware接口，如果有 需要执行相应的代码， 使得最终实例化出来的bean可以调用接口的方法 比如 beanfactoryaware applicationcontextaware之类的；

4：beanpostprocessor

对于满足条件的bean执行相应的方法

和aop的实现有关系， aop的切片 里面以表达式的形式定义了切片的描述，spring在这里，就对这里进行了代理的的替换， 完成aop方法的代理注入

cglib和jdk自身的；

如果bean实现了InitializingBean

实例化->生成对象->属性填充后会进行afterPropertiesSet方法 此处还会执行这个方法

5：bean自身的Initmethod

6：beanpostprocessorafterinnitialation

6.5 根据此时是单例还是多例， 单例是要放入到缓存map里面的，多例就不需要了；

7 :使用的disposablebean的方法 调用 destroy（）方法；

12：

spring支持的事务管理

主要是编程事务 和生命的事务

有多种事务传播的策略：



不被管理 数据库不支持； 不是pub 或者final的方法；

13：

事务的传播机制

事务的传播性一般用在**事务嵌套**的场景，比如一个**事务方法**里面调用了**另外一个事务方法**，那么两个方法是各自作为独立的方法提交还是内层的事务合并到外层的事务一起提交，这就是需要事务传播机制的配置来确定怎么样执行。

常用的事务传播机制如下：

属性 描述

PROPAGATION\_REQUIRED Spring默认的传播机制，能满足绝大部分业务需求，如果外层有事务，则当前事务加入到外层事务，一块提交，一块回滚。如果外层没有事务，新建一个事务执行， 随主人；

PROPAGATION\_REQUES\_NEW 该事务传播机制是每次都会新开启一个事务，同时把外层事务挂起，当当前事务执行完毕，恢复上层事务的执行。如果外层没有事务，执行当前新开启的事务即可 自己新开事务；

PROPAGATION\_SUPPORT 如果外层有事务，则加入外层事务，如果外层没有事务，则直接使用非事务方式执行。

PROPAGATION\_NOT\_SUPPORT 该传播机制不支持事务，如果**外层存在事务则挂起**，执行完当前代码，则恢复外层事务，无论是否异常都不会回滚当前的代码

PROPAGATION\_NEVER 该传播机制不支持外层事务，即如果外层有事务就抛出异常

PROPAGATION\_MANDATORY 与NEVER相反，如果外层没有事务，则抛出异常

PROPAGATION\_NESTED 该传播机制的特点是可以保存状态保存点，当前事务回滚到某一个点，从而避免所有的嵌套事务都回滚，即各自回滚各自的，如果子事务没有把异常吃掉，基本还是会引起全部回滚的。

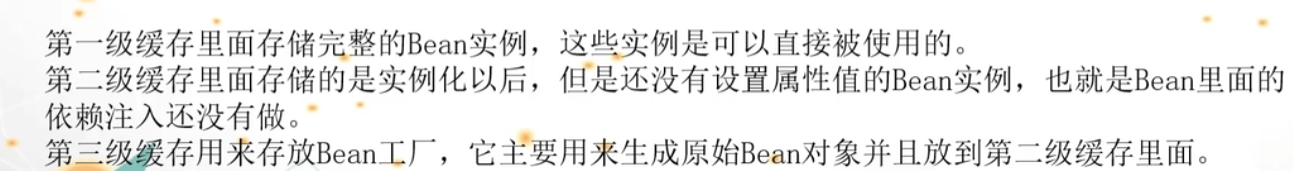
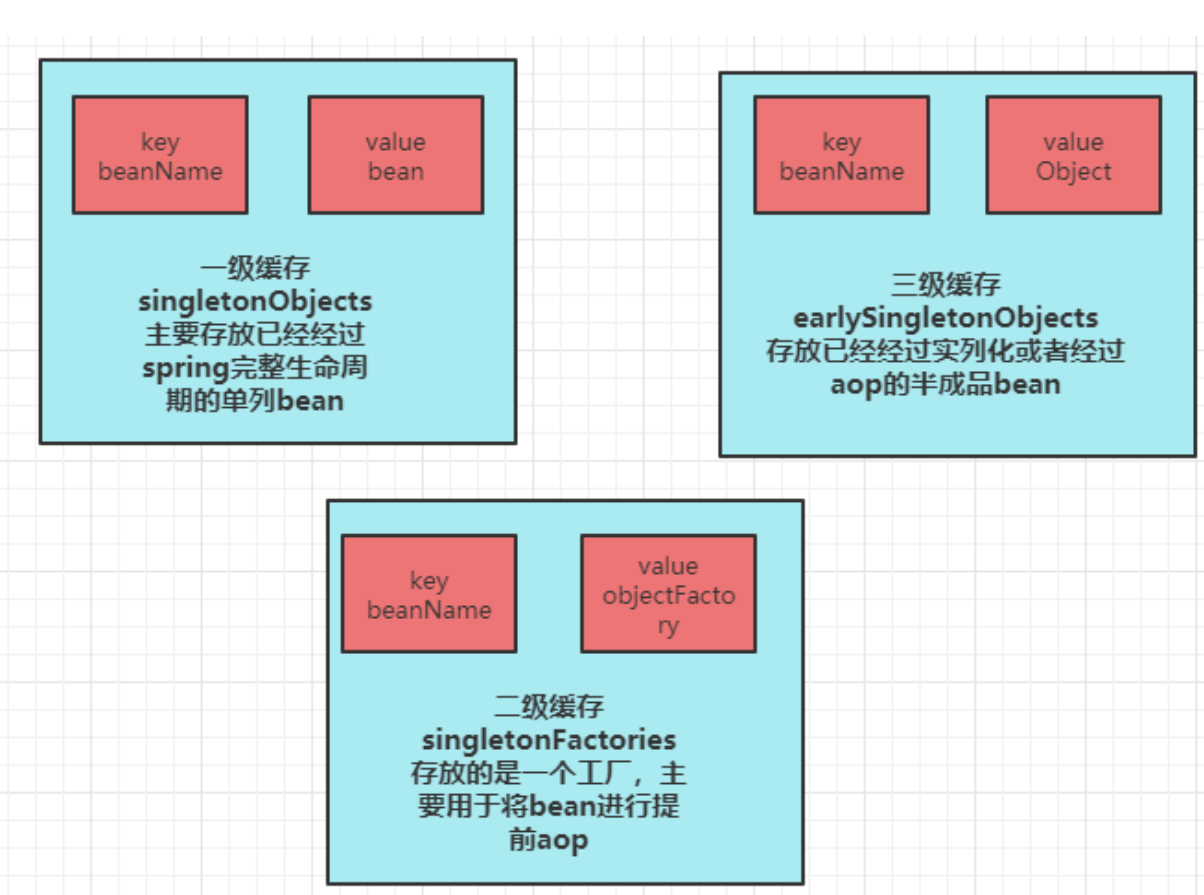
外部有事务 内部可以自己开新的事务 可以不支持事务 可以延续外部的事务；

外部没有实物 自己不开事务 或者开启新的事务；

大概这么有8种吧。

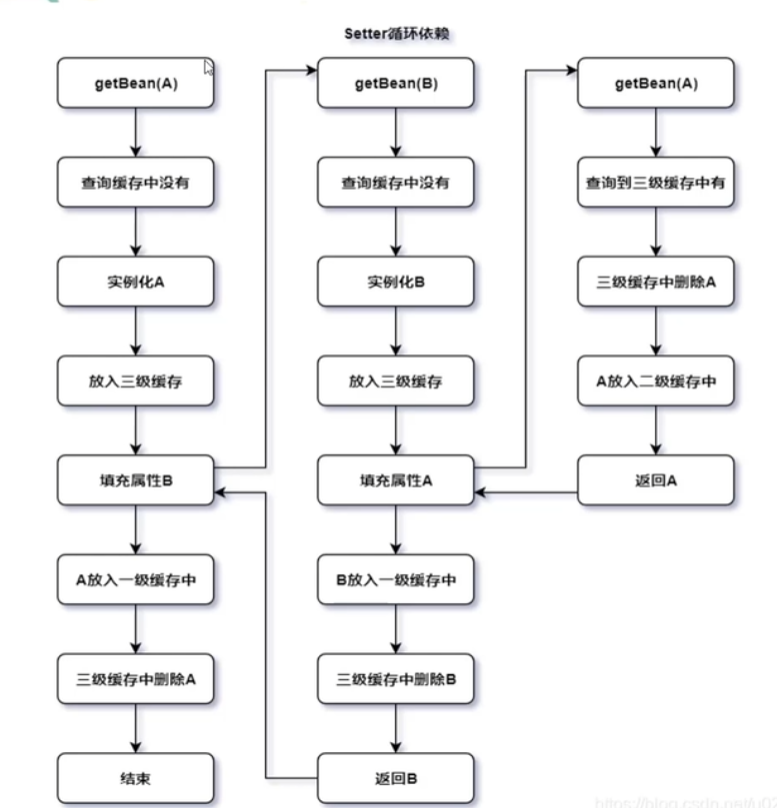
14：

如何解决循环依赖



二级是尚未实例化完全的 三级是 beanFactory

spring中创建的bean有三级缓存；



第一层是 singletonObjects

二级缓存是 用于提前aop的

三级是 经过实例化的半成品， 发成循环依赖的时候， 会检测

newA()的时候 在 singletonFactories 放入一个半成品的a B。setA的时候，就从这个缓存中取出来；

b走完了以后 b进入1级缓存， a 再继续自己的流程 ，此时 B已经存在了；

再增加一个 singletonFactories 缓存

在依赖注入前，即 a.setB() 以及 b.setA() 将 a 及 b 的半成品对象（未完成依赖注入和初始化）放入此缓存

执行依赖注入时，先看看 singletonFactories 缓存中是否有半成品的对象，如果有拿来注入，顺利走完流程

对于上面的图

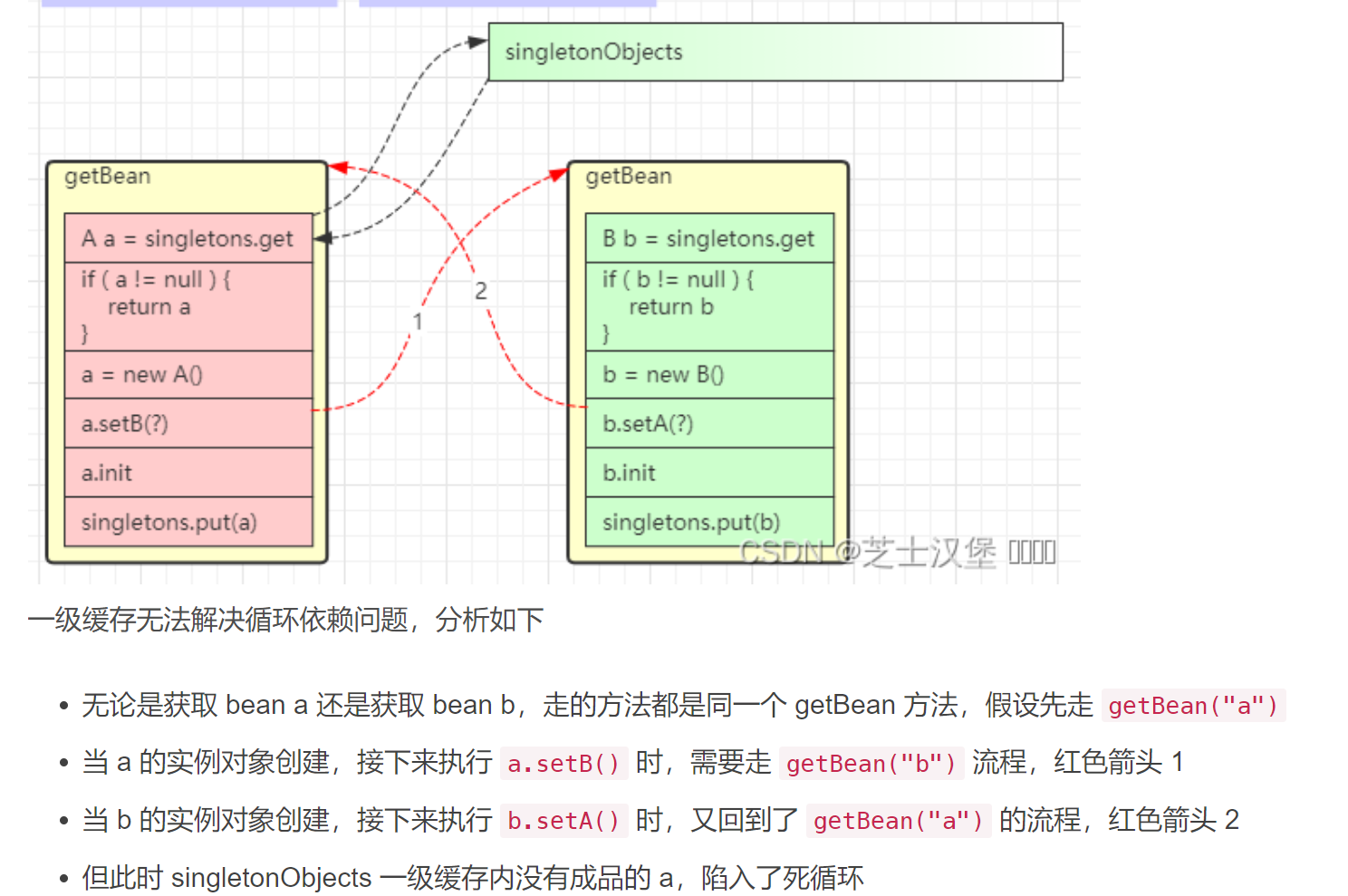
a = new A() 执行之后就会把这个半成品的 a 放入 singletonFactories 缓存，即 factories.put(a)

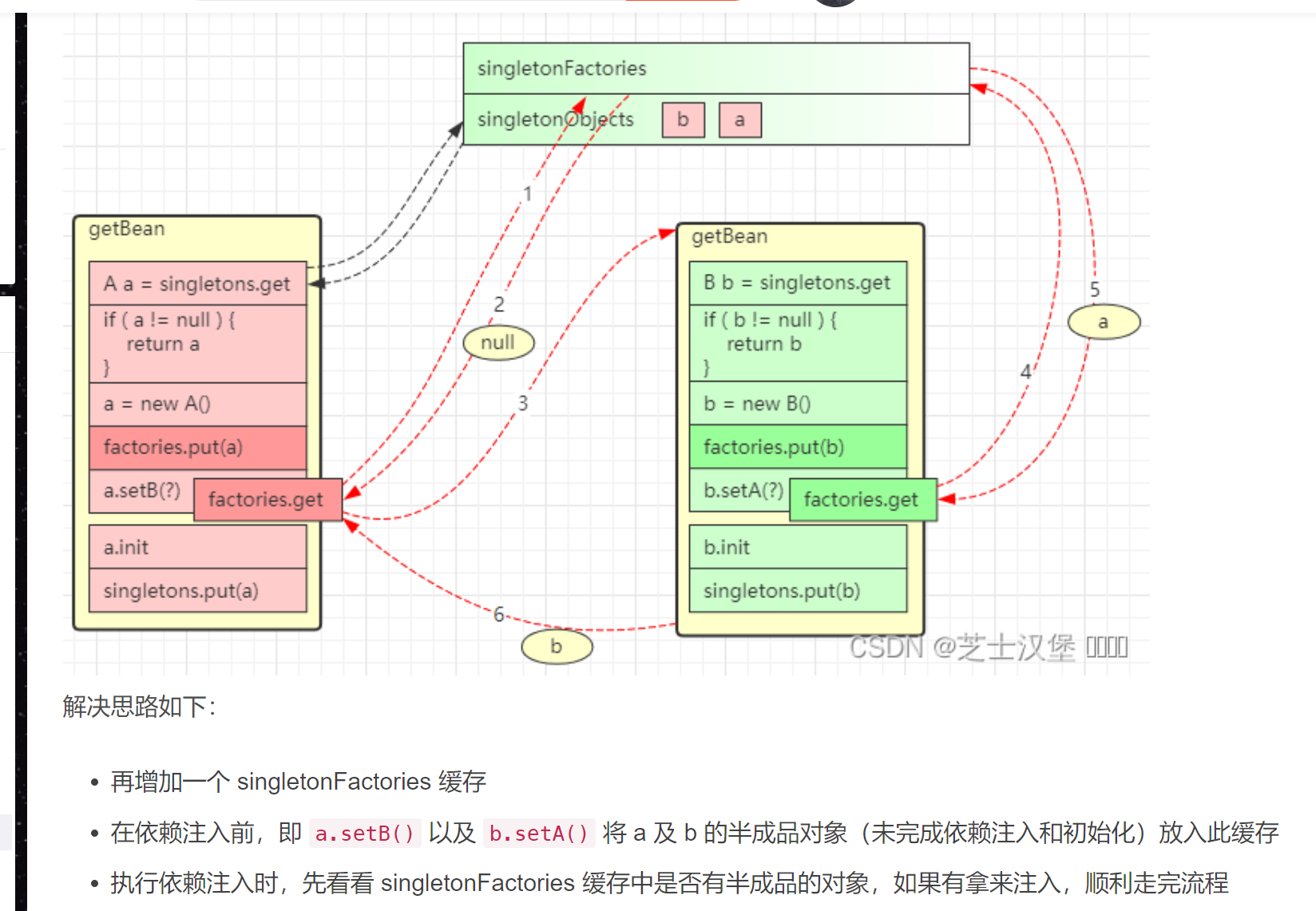
接下来执行 a.setB()，走入 getBean("b") 流程，红色箭头 3

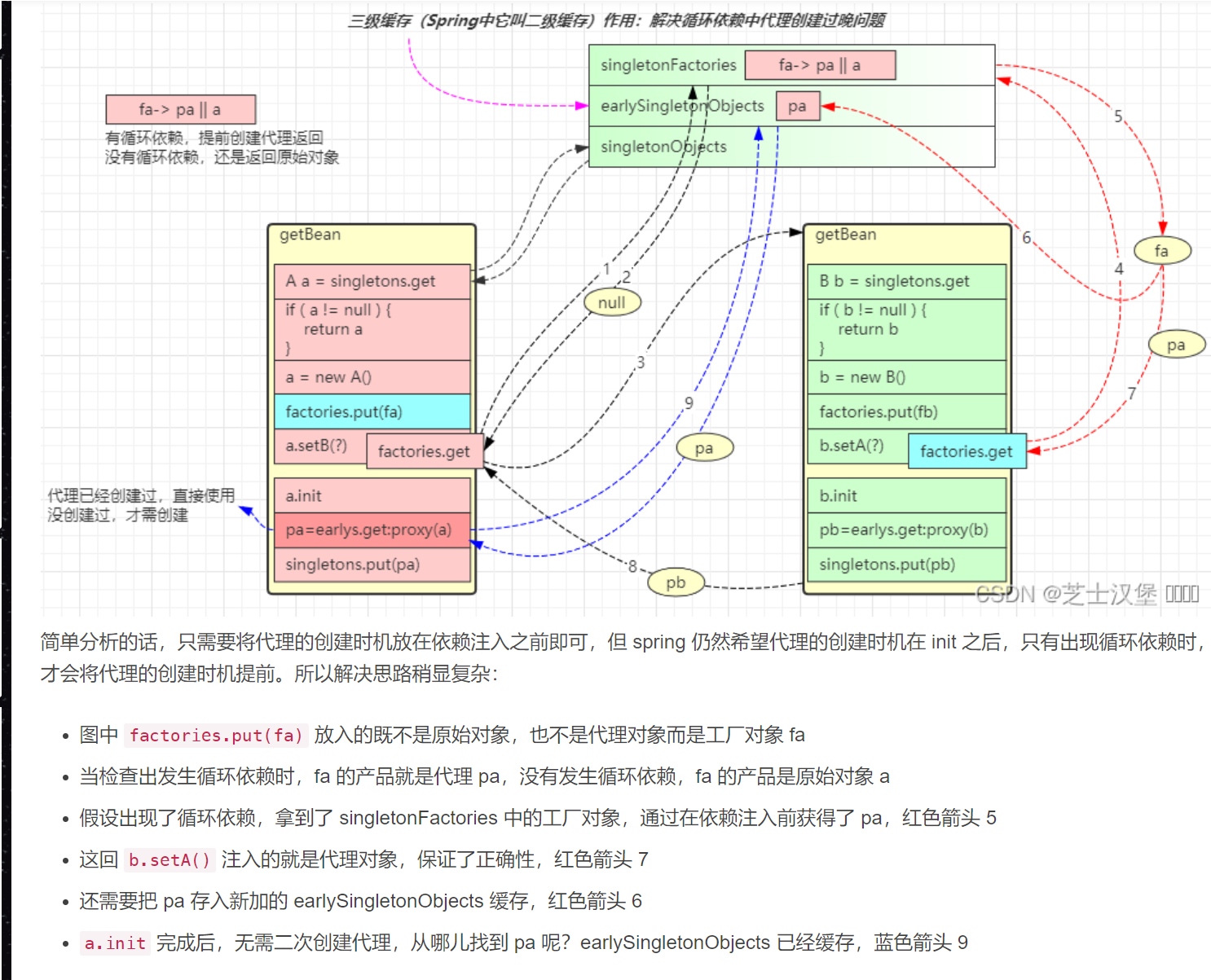
这回再执行到 b.setA() 时，需要一个 a 对象，有没有呢？有！

factories.get() 在 singletonFactories 缓存中就可以找到，红色箭头 4 和 5

b 的流程能够顺利走完，将 b 成品放入 singletonObject 一级缓存，返回到 a 的依赖注入流程，红色箭头 6

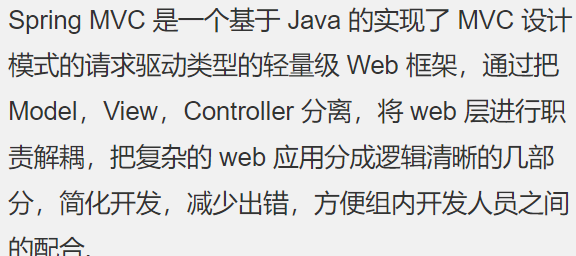


放入singletonFactories里面 ，是未初始化的对象， 就可以解决循环依赖的问题；



ealsobjes

springmvc笔记

1：

**到了dispatcherservlet 根据handlermapping找到对应的handler， 处理完后返回MAV对象，mav交给view解析器， 生成页面对象和model对象给前端，将返回的对象交给view进行数据的填充，之后就可以返回前端了**

2： 请求首先到了servlet层，根据请求调用java Bean，将显示结果交给JSP

3：请求流程：



首先回答tomcat 的请求流程 生命周期；

执行前；当一个请求发来时先进服务器（Tomcat）,在服务器中会有拦截器，过滤器啊，等这些功能走完之后，才真正的进入了框架中。

1.用户发来一个请求，首先进入的是前端控制器DispatcherServlet

2.前端控制器将（DispacherServlet）用户发来的请求发送给处理器映射器（HandlerMapping）

3.处理器映射器根据前端控制器发来的用户的请求找到对应符合的控制器（Handler）,并且将其封装成处理器执行链，返回给前端控制器。

4.处理器适配器接收到来自前端控制器的执行链后，找到对应执行此执行链的处理器适配器（HandlerAdapter）来调用的具体的控制器（就是说其对应的方法或者逻辑）

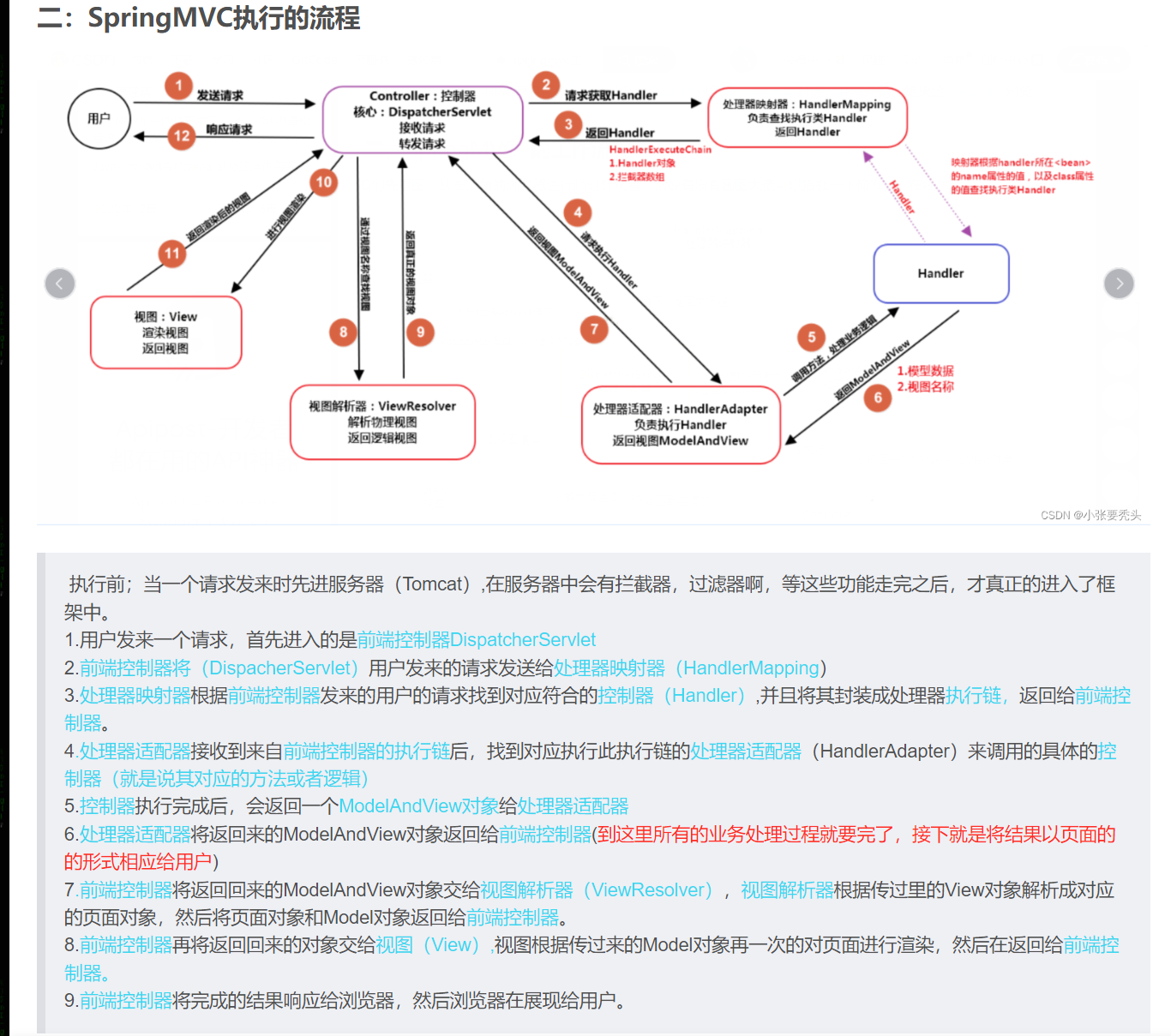
5.控制器执行完成后，会返回一个ModelAndView对象给处理器适配器

6.处理器适配器将返回来的ModelAndView对象返回给**前端控制器**(到这里所有的业务处理过程就要完了，接下就是将结果以页面的的形式相应给用户)

7.前端控制器将返回回来的ModelAndView对象交给**视图解析器**（ViewResolver），视图解析器根据传过里的View对象解析成对应的页面对象，然后将页面对象和Model对象返回给前端控制器。

8.前端控制器再将返回回来的对象交给视图（View）,视图根据传过来的Model对象再一次的对页面进行渲染，然后在返回给前端控制器。

9.前端控制器将完成的结果响应给浏览器，然后浏览器在展现给用户。



4：springmvc提供全局处理器来处理异常；

5：

加一个HttpServletResponse就可以了；



springboot笔记

1: 核心配置类：



2：