Spring

前奏

框架是什么?

框架就是一些类和接口的集合，通过这些类和接口协调来完成一系列的程序实现。

优点:

减少重复的代码的编写、

让代码的结构更加清晰、耦合度更低，开发后期维护方便.

减少BUG的产生

1 Spring概述

Spring 是最受欢迎的企业级 Java 应用程序开发框架，数以百万的来自世界各地的开发人员使用 Spring 框架来创建性能好、易于测试、可重用的代码。

Spring 框架是一个开源的 Java 平台，它最初是由 Rod Johnson 编写的，并且于 2003 年 6 月首次在 Apache 2.0 许可下发布。

Spring 是轻量级的框架，其基础版本只有 2 MB 左右的大小。

Spring 框架的核心特性是可以用于开发任何 Java 应用程序，但是在 Java EE 平台上构建 web 应用程序是需要扩展的。 Spring 框架的目标是使 J2EE 开发变得更容易使用，通过启用基于 POJO 编程模型来促进良好的编程实践。

1.1 Spring介绍

1.Spring是什么?

Spring是一个开源的轻量级的Java开发框架

2.Spring有什么作用?

简化应用程序的开发。

3.简化应用程序开发体现在哪些方面?

①IOC容器

② AOP

1.2 Spring简介

Spring框架由Rod Johnson开发，2004年发布了Spring框架的第一版。Spring是一个从实际开发中抽取出来的框架，因此它完成了大量开发中的通用步骤，留给开发者的仅仅是与特定应用相关的部分，从而大大提高了企业应用的开发效率。

Spring总结起来优点如下：

一、低侵入式设计，代码的污染极低。

二、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的承诺。

三、Spring的IoC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。

四、Spring的AOP支持允许将一些通用任务如安全、事务、日志等进行集中式管理，从而提供了更好的复用。

五、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。

六、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框架的部分或全部。

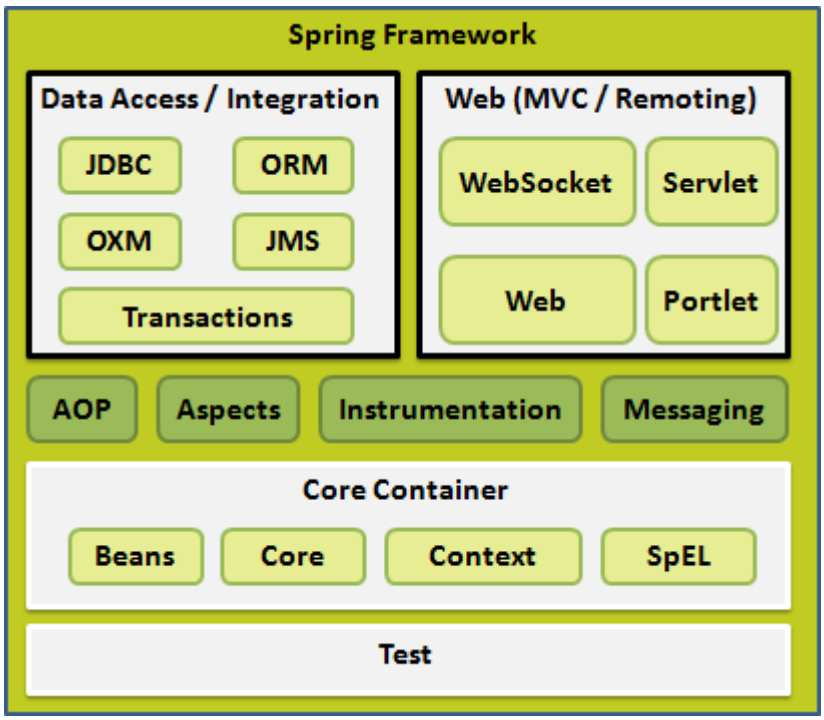
Spring 是一个开源框架

Spring 是一个 IOC(DI) 和 AOP 容器框架

2 Spring 体系结构

Spring 有可能成为所有企业应用程序的一站式服务点，然而，Spring 是模块化的，允许你挑选和选择适用于你的模块，不必要把剩余部分也引入。下面的部分对在 Spring 框架中所有可用的模块给出了详细的介绍。

Spring 框架提供约 20 个模块，可以根据应用程序的要求来使用。



2.1 Spring模块讲解

核心容器由**spring-core，spring-beans，spring-context，spring-context-support和spring-expression**（SpEL，Spring表达式语言，Spring Expression Language）等模块组成，它们的细节如下：

- **spring-core**模块提供了框架的基本组成部分，包括 IoC 和依赖注入功能。
- **spring-beans** 模块提供 BeanFactory，工厂模式的微妙实现，它移除了编码式单例的需要，并且可以把配置和依赖从实际编码逻辑中解耦。
- **context**模块建立在由**core**和 **beans** 模块的基础上建立起来的，它以一种类似于JNDI注册的方式访问对象。Context模块继承自Bean模块，并且添加了国际化（比如，使用资源束）、事件传播、资源加载和透明地创建

上下文（比如，通过Servelet容器）等功能。Context模块也支持Java EE的功能，比如EJB、JMX和远程调用等。**ApplicationContext**接口是Context模块的焦点。**spring-context-support**提供了对第三方库集成到Spring上下文的支持，比如缓存（EhCache, Guava, JCache）、邮件（JavaMail）、调度（CommonJ, Quartz）、模板引擎（FreeMarker, JasperReports, Velocity）等。

- **spring-expression**模块提供了强大的表达式语言，用于在运行时查询和操作对象图。它是JSP2.1规范中定义的统一表达式语言的扩展，支持set和get属性值、属性赋值、方法调用、访问数组集合及索引的内容、逻辑算术运算、命名变量、通过名字从Spring IoC容器检索对象，还支持列表的投影、选择以及聚合等。

2.1.1 数据访问/集成

数据访问/集成层包括 JDBC，ORM，OXM，JMS 和事务处理模块，它们的细节如下：

（注：JDBC=Java Data Base Connectivity，ORM=Object Relational Mapping，OXM=Object XML Mapping，JMS=Java Message Service）

- **JDBC** 模块提供了JDBC抽象层，它消除了冗长的JDBC编码和对数据库供应商特定错误代码的解析。
- **ORM** 模块提供了对流行的对象关系映射API的集成，包括JPA、JDO和Hibernate等。通过此模块可以让这些ORM框架和spring的其它功能整合，比如前面提及的事务管理。
- **OXM** 模块提供了对OXM实现的支持，比如JAXB、Castor、XML Beans、JiBX、XStream等。
- **JMS** 模块包含生产（produce）和消费（consume）消息的功能。从Spring 4.1开始，集成了spring-messaging模块。。
- **事务**模块为实现特殊接口类及所有的 POJO 支持编程式和声明式事务管理。（注：编程式事务需要自己写beginTransaction()、commit()、rollback()等事务管理方法，声明式事务是通过注解或配置由spring自动处理，编程式事务粒度更细）

2.1.2 Web

Web 层由 Web，Web-MVC，Web-Socket 和 Web-Portlet 组成，它们的细节如下：

- **Web** 模块提供面向web的基本功能和面向web的应用上下文，比如多部分（multipart）文件上传功能、使用Servlet监听器初始化IoC容器等。它还包括HTTP客户端以及Spring远程调用中与web相关的部分。。
- **Web-MVC** 模块为web应用提供了模型视图控制（MVC）和REST Web服务的实现。Spring的MVC框架可以使领域模型代码和web表单完全地分离，且可以与Spring框架的其它所有功能进行集成。
- **Web-Socket** 模块为 WebSocket-based 提供了支持，而且在 web 应用程序中提供了客户端和服务器端之间通信的两种方式。
- **Web-Portlet** 模块提供了用于Portlet环境的MVC实现，并反映了spring-webmvc模块的功能。

2.1.3 其他

还有其他一些重要的模块，像 AOP，Aspects，Instrumentation，Web 和测试模块，它们的细节如下：

- **AOP** 模块提供了面向方面的编程实现，允许你定义方法拦截器和切入点对代码进行干净地解耦，从而使实现功能的代码彻底的解耦出来。使用源码级的元数据，可以用类似于.Net属性的方式合并行为信息到代码中。
- **Aspects** 模块提供了与 **AspectJ** 的集成，这是一个功能强大且成熟的面向切面编程（AOP）框架。
- **Instrumentation** 模块在一定的应用服务器中提供了类 instrumentation 的支持和类加载器的实现。
- **Messaging** 模块为 STOMP 提供了支持作为在应用程序中 WebSocket 子协议的使用。它也支持一个注解编程模型，它是为了选路和处理来自 WebSocket 客户端的 STOMP 信息。
- **测试**模块支持对具有 JUnit 或 TestNG 框架的 Spring 组件的测试。

3 Spring常用特性

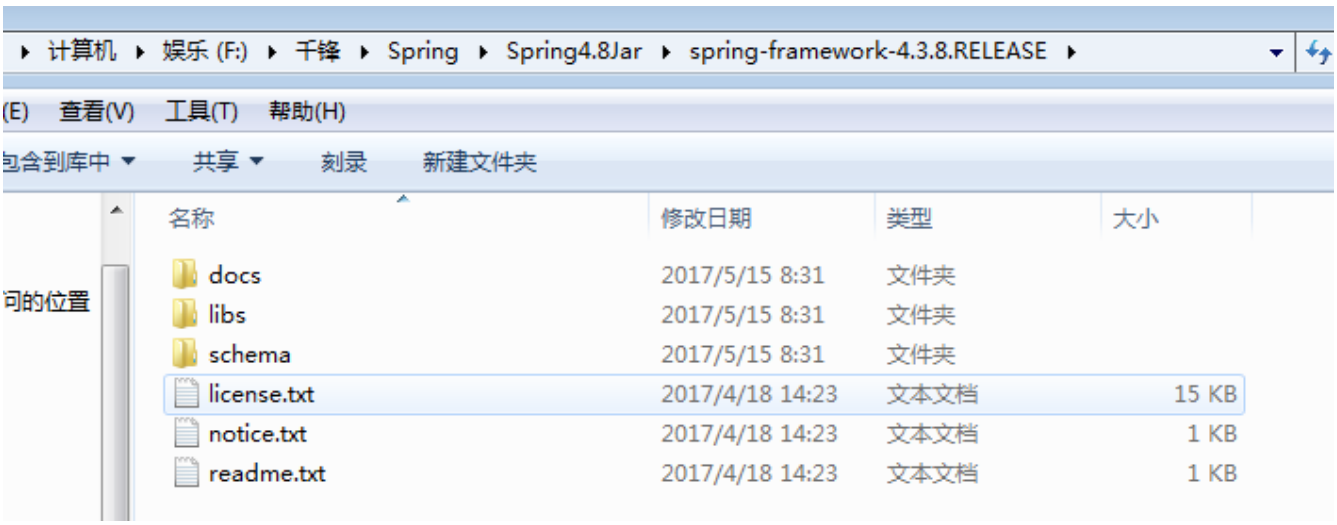
利用Spring来创建对象（JavaBean工厂） 利用Spring构建业务逻辑层 管理依赖关系 适应需求变更 利用Spring创建数据访问对象（DAO） 利用Spring进行事务处理、日志操作

4 Spring安装与下载

1 下载并解压

http://repo.spring.io/release/org/springframework/spring/

下载zip压缩包: spring-framework-4.0.6.RELEASE-dist.zip 并解压。



2 创建普通Java Project

eclipse or

3 将相应的jar包加入类路径

spring.jar

当然，这些jar包对于一个web项目还是不够的，仍需要一些其他的jar包，如commons-lang-*.jar,commons-fileupload-*.jar等等。

使用Spring框架开发最少需要:

commons-logging.jar spring-aop.jar spring-core.jar spring-context.jar spring-beans.jar spring-expression.jar spring-web.jar spring-tx.jar aopalliance.jar aspectjrt.jar aspectjweaver.jar

4创建applicationContext.xml 配置文件

在**src**目录下创建文件 applicationContext.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
```

```
        xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/aop
         http://www.springframework.org/schema/aop/spring-aop.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd">

</beans>
```

xmlns可到此目录引用 命名空间（或通过直接URL方式）： Spring4.8Jar/spring-framework-4.3.8.RELEASE/docs/spring-framework-reference/html/xsd-configuration.html

5 Spring IOC

Spring两大核心技术：

控制反转（IOC）和面向切面编程（AOP）的轻量级的容器，为软件开发提供全方位支持的应用程序框架。

容器是符合某种规范能够提供一系列服务的管理器，开发人员可以利用容器所提供的服务来方便地实现某些特殊的功能。 所谓的“重量级”容器是指那些完全遵守J2EE的规范，提供规范中所有的服务。 “轻量级”容器的也是遵守J2EE的规范，但其中的服务可以自由配置。

Spring IOC简介：

```
控制反转（IOC）  （Inversion of Control）
依赖注入（DI）  （Dependency Injection）
    当某个角色（比如创建一个JAVA实例，调用者）需要另一个角色（另一个JAVA实例，被调用者）的协助时，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。但在Spring里，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。

      Spring核心容器是整个应用中的超级大工厂，所有的JAVA对象都交给Spring容器管理–这些JAVA对象被统称为Spring容器中的bean
```

控制反转示例

applicationContext.xml

在 Spring 的 IOC 容器里配置 Bean

在 xml 文件中通过 bean 节点来配置 bean

```
  <bean id="user" class="com.qfjy.bean.User"></bean>
```

id：bean 的名称。

在 IOC 容器中必须是唯一的 id 可以指定多个名字，名字之间可用逗号、分号、或空格分隔

每个bean都会指定class属性，class属性指明了Bean的来源，即Bean的实际路径，注意，这里要写出完整的包名+类名(全限定名)。

java类

```
public class DemoTest {
    public static void main(String[] args) {
        /**
         * 调用方创建对象
         */
        User user=new User();
        user.setId(123);
        user.setName("qfjy");
        System.out.println(user);
        /**
         * Spring IOC容器管理对象创建
         */
        //创建Spring IOC容器
        ApplicationContext ctx=
                new ClassPathXmlApplicationContext("applicationContext.xml");
       //在容器中取出对应bean实例
        User user1= (User) ctx.getBean("user");
        System.out.println(user1);
    }
}
```

应用本身不负责依赖对象的创建和维护，而是由外部容器来负责。

这样控制权就由应用转移到外部容器，控制权的转移就是所谓的反转。

Bean其它描述

在Bean中有一个id属性及class属性，这个id唯一标识了该Bean。在配置文件中，不能有重复的Bean的id， 因为在代码中通过BeanFactory或ApplicationContext来获取Bean的实例时，都要用它来作为唯一索引。

Bean的id属性 为Bean指定别名可以用name属性来完成，如果需要为Bean指定多个别名， 可以在name属性中使用逗号（,）、分号（;）或空格来分隔多个别名，在程序中可以通过任意一个别名访问该Bean实例。 例如，id为“HelloWorld”的Bean，其别名为：

```
 <bean id="user" name="u;u1;u2" class="com.qfjy.bean.User"></bean>
```

则在程序中可以利用“u”、“u1”、“u2”任意一个来获取Bean的实例

以下三种方法均可完成Bean实例的获取

```
         //根据名称在容器中取出对应bean实例
        User u= (User) ctx.getBean("u");
        User u1= (User) ctx.getBean("u1");
        User u2= (User) ctx.getBean("u2");
```

BeanFactory

简介：

1. Spring容器最基本的接口就是BeanFactor。BeanFactory负责配置、创建、管理Bean，他有一个子接口：ApplicationContext，因此也称之为Spring上下文。Spring容器负责管理Bean与Bean之间的依赖关系。
2. BeanFactory：是IOC容器的核心接口， 它定义了IOC的基本功能，我们看到它主要定义了getBean方法。getBean方法是IOC容器获取bean对象和引发依赖注入的起点。方法的功能是返回特定的名称的Bean。
3. BeanFactory 是初始化 Bean 和调用它们生命周期方法的“吃苦耐劳者”。注意，BeanFactory 只能管理单例（Singleton）Bean 的生命周期。它不能管理原型(prototype,非单例)Bean 的生命周期。这是因为原型 Bean 实例被创建之后便被传给了客户端,容器失去了对它们的引用。
4. BeanFactory有着庞大的继承、实现体系，有众多的子接口、实现类。

BeanFactory接口包含以下几个基本方法：

Ø Boolean containBean(String name):判断Spring容器是否包含id为name的Bean实例。

Ø getBean(Class requiredTypr):获取Spring容器中属于requiredType类型的唯一的Bean实例。

Ø Object getBean(String name)：返回Sprin容器中id为name的Bean实例。

BeanFactorty接口提供了配置框架及基本功能，但是无法支持spring的aop功能和web应用。而ApplicationContext接口作为BeanFactory的派生，因而提供BeanFactory所有的功能。而且ApplicationContext还在功能上做了扩展，相较于BeanFactorty，ApplicationContext还提供了以下的功能：

（1）MessageSource, 提供国际化的消息访问
（2）资源访问，如URL和文件
（3）事件传播特性，即支持aop特性 （4）载入多个（有继承关系）上下文 ，使得每一个上下文都专注于一个特定的层次，比如应用的web层

ApplicationContext：是IOC容器另一个重要接口， 它继承了BeanFactory的基本功能， 同时也继承了容器的高级功能，如：MessageSource（国际化资源接口）、ResourceLoader（资源加载接口ApplicationEventPublisher（应用事件发布接口）等。

二者区别

1. BeanFactroy采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化，这样，我们就不能发现一些存在的Spring的配置问题。而ApplicationContext则相反，它是在容器启动时，一次性创建了所有的Bean。ApplicationContext 唯一的不足是占用内存空间。当应用程序配置Bean较多时，程序启动较慢。
2. BeanFactory负责读取bean配置文档，管理bean的加载，实例化，维护bean之间的依赖关系，负责bean的声明周期。ApplicationContext除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能(国际化..资源访问..etc..)
3. 常用的获取ApplicationContext的方式：FileSystemXmlApplicationContext：从文件系统或者url指定的xml配置文件创建，参数为配置文件名或文件名数组，有**相对路径**与**绝对路径**。

```
ApplicationContext factory=new
FileSystemXmlApplicationContext("src/applicationContext.xml");

ApplicationContext factory=new
FileSystemXmlApplicationContext("D:/java/eclipse/workSpace/1-
spring/src/applicationContext.xml");
```

4.ClassPathXmlApplicationContext：从classpath的xml配置文件创建，可以从jar包中读取配置文件。
ClassPathXmlApplicationContext 编译路径总有几种方式：

```
ApplicationContext factory = new
ClassPathXmlApplicationContext("classpath:applicationContext.xml");

ApplicationContext factory = new ClassPathXmlApplicationContext("applicationContext.xml");
```

依赖注入示例

Spring 支持 3 种依赖注入的方式

属性注入 构造器注入 工厂方法注入(很少使用，不推荐)

属性注入

Set注入 最简单的注入方式(简单、直观、Spring大量使用)

```
   <bean id="user" name="u;u1;u2" class="com.qfjy.bean.User">
      <!--Set注入 -->
       <property name="id" value="1"></property>
       <property name="name" value="qfjy"></property>
   </bean>
```

构造器注入

是指带有参数的构造函数注入(完成程序初始化动作)

```
 <bean id="user"  class="com.qfjy.bean.User">
       <!--构造器注入，需添加对应的构造方法 -->
      <constructor-arg name="id" value="99"></constructor-arg>
       <constructor-arg name="name" value="qfjy"></constructor-arg>
   </bean>
```

1、根据参数的名字传值：（推荐用法）

```
<constructor-arg name="id" value="99"></constructor-arg>
```

2、直接传值。这种方法也是根据顺序排的，所以一旦调换位置的话，就会出现bug

```
<bean id="user"  class="com.qfjy.bean.User">
       <!--构造器注入，需添加对应的构造方法 -->
      <constructor-arg  value="99"></constructor-arg>
       <constructor-arg  value="qfjy"></constructor-arg>
   </bean>
```

3、根据索引赋值 index，索引都是以0开头

```
   <bean id="user"  class="com.qfjy.bean.User">
      <constructor-arg index="0"  value="99"></constructor-arg>
      <constructor-arg index="1"  value="qfjy"></constructor-arg>
   </bean>
```

注入其它类型

JavaBean对象

```
  <bean id="user"  class="com.qfjy.bean.User">
       <property name="name" value="qfjy"></property>
        <property name="role" ref="role"></property>
    </bean>

    <bean id="role" class="com.qfjy.bean.Role">
        <property name="rname" value="管理员"></property>
    </bean>
```

内部Bean

当 Bean 实例仅仅给一个特定的属性使用时, 可以将其声明为内部 Bean.

内部 Bean 声明直接包含在 **或** 元素里, 不需要设置任何 id 或 name 属性 内部 Bean 不能使用在任何其他地方

```
   <bean id="user"  class="com.qfjy.bean.User">
       <property name="name" value="qfjy"></property>
        <!-- 内部bean 特点：不能被外部bean访问-->
        <property name="role" >
            <bean  class="com.qfjy.bean.Role">
                <property name="rname" value="管理员"></property>
            </bean>
        </property>
    </bean>
```

集合类型List

对list集合的注入。如果类中有list类型的属性，在为其依赖注入值的时候就需要在配置文件中的元素下应用其子元素。

```
<property name="infos">
            <list>
                <value>喜欢学习java</value>
                <value>唱歌听音乐</value>
            </list>
</property>
```

集合类型注入bean

```
<property name="roles">
            <list>
                <!--引用外部bean -->
                <ref bean="role"></ref>
                <!--内部bean -->
                <bean class="com.qfjy.bean.Role">
                    <property name="rid" value="1"></property>
                    <property name="rname" value="管理员"></property>
                </bean>
            </list>
        </property>
```

集合类型Map

```
<property name="map">
            <map key-type="java.lang.String" value-type="com.qfjy.bean.Role">
                <entry key="r1" value-ref="role"></entry>
                <entry key="r2" >
                    <bean class="com.qfjy.bean.Role">
                        <property name="rid" value="1"></property>
                        <property name="rname" value="管理员"></property>
                    </bean>
                </entry>
            </map>
        </property>
```

6 Bean作用域

创建一个bean定义，其实质是用该bean定义对应的类来创建实例的对象。

默认情况下, Spring 只为每个在 IOC 容器里声明的 Bean 创建唯一一个实例, 整个 IOC 容器范围内都能共享该实例: 所有后续的 getBean() 调用和 Bean 引用都将返回这个唯一的 Bean 实例. 该作用域被称为 **singleton**, 它是所有 Bean 的默认作用域.

类别	说明
singleton	在Spring IoC容器中仅存在一个Bean实例，Bean以单例方式存在，默认值
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()
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Session使用不同Bean，仅适用于WebApplicationContext 环境
globalSession	一般用于Portlet应用环境，该作用域仅适用于WebApplicationContext 环境

五种作用域中，request、session和global session三种作用域仅在基于web的应用中使用（不必关心你所采用的是什么web应用框架），只能用在基于web的Spring ApplicationContext环境。

备注：Portlet是基于Java的Web组件，由Portlet容器管理，并由容器处理请求，生产动态内容。

7 Bean生命周期

Spring IOC 容器管理 Bean 的生命周期, Spring 允许在 Bean 生命周期的特定点执行定制的任务.

Spring IOC 容器对 Bean 的生命周期进行管理的过程:

1通过构造器或工厂方法创建 Bean 实例 2为 Bean 的属性设置值和对其他 Bean 的引用 3调用 Bean 的初始化方法 4Bean 可以使用了 5当容器关闭时 close(), 调用 Bean 的销毁方法 在 Bean 的声明里设置 init-method 和 destroy-method 属性, 为 Bean 指定初始化和销毁方法.

```
<bean id="user"  class="com.qfjy.bean.User" init-method="init" destroy-method="destroy">
```

```
  public void init(){
      System.out.println("----初始化操作---");
  }
  public  void destroy(){
      System.out.println("----结束等操作");
  }
```

注解方式： init-method:@PostConstruct destroy-method:@PreDestroy

8 Bean 后置处理器

Bean 后置处理器允许在调用初始化方法前后对 Bean 进行额外的处理。 Bean 后置处理器对 IOC 容器里的所有 Bean 实例逐一处理, 而非单一实例. 其典型应用是: 检查 Bean 属性的正确性或根据特定的标准更改 Bean 的属性.(**一个好的框架必备的特性至少得有开闭原则，可扩展性**)

一、对Bean 后置处理器而言, 需要实现**BeanFactoryPostProcessor** 接口. 在初始化方法被调用前后, Spring 将把每个 Bean 实例分别传递给上述接口的以下两个方法:

BeanPostProcessor接口的源码，它定义了两个方法，一个在bean初始化之前，一个在bean初始化之后

```
public class UserBeanPostProcessor implements BeanPostProcessor {
    /**
     *
     * @param o bean对象
     * @param s bean 名称
     * @return
     * @throws BeansException
     */
    @Override
    public Object postProcessBeforeInitialization(Object o, String s) throws BeansException
{
        System.out.println("-->"+o);
        System.out.println("-->"+s);
        return o;
    }

    @Override
    public Object postProcessAfterInitialization(Object o, String s) throws BeansException {
        System.out.println(o);
        System.out.println(s);
        return o;
    }
}
```

9 循环依赖

什么是循环依赖?

循环依赖其实就是循环引用，也就是两个或者两个以上的bean互相持有对方，最终形成闭环。比如A依赖于B，B依赖于A，

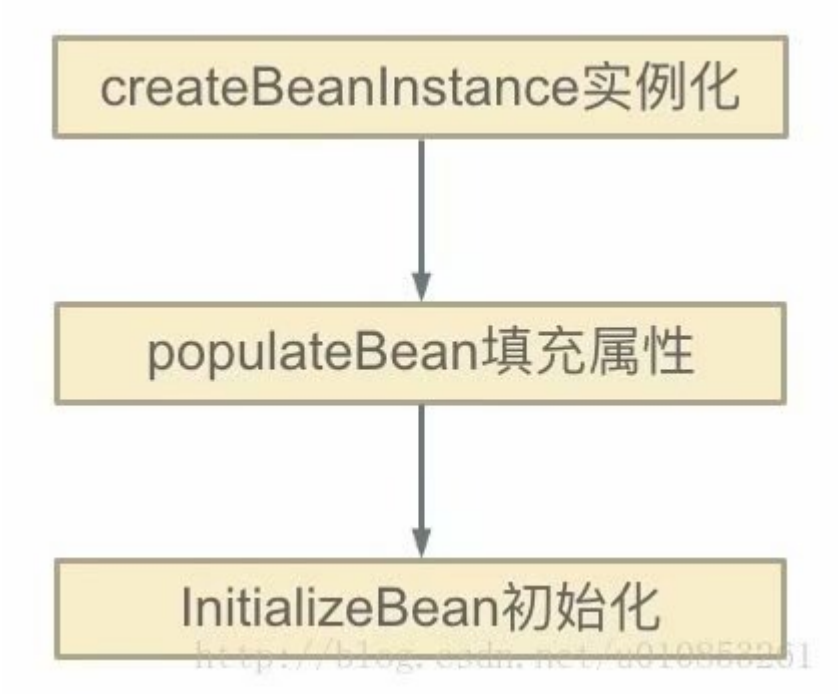
注意，这里不是函数的循环调用，是对象的相互依赖关系。循环调用其实就是一个死循环，除非有终结条件。

Spring中循环依赖场景有：

（1）构造器的循环依赖 （2）field属性的循环依赖 其中，构造器的循环依赖问题无法解决，只能抛出BeanCurrentlyInCreationException异常，在解决属性循环依赖时，spring采用的是提前暴露对象的方法。

Spring怎么解决循环依赖

Spring的循环依赖的理论依据基于Java的引用传递，当获得对象的引用时，对象的属性是可以延后设置的。（但是构造器必须是在获取引用之前）



（1）createBeanInstance：实例化，其实也就是调用对象的构造方法实例化对象

（2）populateBean：填充属性，这一步主要是多bean的依赖属性进行填充

（3）initializeBean：调用spring xml中的init 方法。

从上面单例bean的初始化可以知道：循环依赖主要发生在第一、二步，也就是构造器循环依赖和field循环依赖。那么我们要解决循环引用也应该从初始化过程着手，对于单例来说，在Spring容器整个生命周期内，有且只有一个对象，所以很容易想到这个对象应该存在Cache中，Spring为了解决单例的循环依赖问题，使用了**三级缓存**。

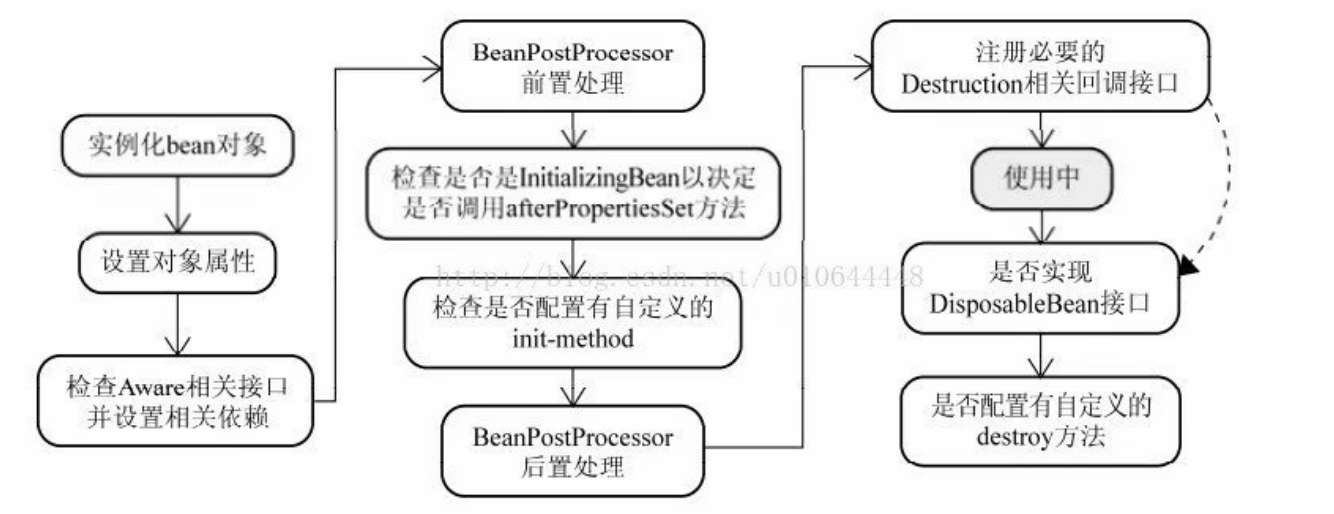
这三级缓存分别指：

singletonFactories： 单例对象工厂的cache

earlySingletonObjects：提前暴光的单例对象的Cache

singletonObjects：单例对象的cache

基于setter属性的循环依赖



Spring先是用构造实例化Bean对象，创建成功后，Spring会通过以下代码提前将对象暴露出来，此时的对象A还没有完成属性注入，属于早期对象，此时Spring会将这个实例化结束的对象放到一个Map中，并且Spring提供了获取这个未设置属性的实例化对象引用的方法。 结合我们的实例来看，当Spring实例化了A、B、C后，紧接着会去设置对象的属性，此时A依赖B，就会去Map中取出存在里面的单例B对象，以此类推，不会出来循环的问题

代码: applicationContext.xml

A依赖于B B依赖于C

```
<bean id="a" class="com.qfjy.bean.A">
        <property name="name" value="aaaa"></property>

        <property name="b" ref="b"></property>
    </bean>

    <bean id="b" class="com.qfjy.bean.B">
        <property name="name" value="bbbb"></property>
        <property name="a" ref="a"></property>
    </bean>
```

控制台打印:

```
A.....
B.....
B name.....
A name.....
```

基于构造器的循环依赖

```
<bean id="a" class="com.qfjy.bean.A">
    <property name="name" value="aaaa"></property>

    <constructor-arg index="0" value="aaa"></constructor-arg>
    <constructor-arg index="1" ref="b"></constructor-arg>
</bean>
```

Spring容器会将每一个正在创建的Bean 标识符放在一个“当前创建Bean池”中，Bean标识符在创建过程中将一直保持在这个池中，因此如果在创建Bean过程中发现自己已经在“当前创建Bean池”里时将抛出BeanCurrentlyInCreationException异常表示循环依赖；在池中的Bean都是未初始化完的，所以会依赖错误，（初始化完的Bean会从池中移除）

总结：

不要使用基于构造函数的依赖注入，可以通过以下方式解决：

1.在字段上使用@Autowired注解，让Spring决定在合适的时机注入

2.用基于setter方法的依赖注入。