# 项目知识学习

## Java Spring Beans生命周期

Spring容器可以管理singleton作用域Bean的生命周期，在此作用域下，Spring能够精确地知道该Bean何时被创建，何时初始化完成，以及何时被销毁。

而对于prototype作用域的Bean，Spring只负责创建，当容器创建了Bean 的实例后，Bean的实例就交给客户端代码管理，Spring容器将不再跟踪其生命周期。每次客户端请求prototype作用域的Bean时，Spring容器都会创建一个新的实例，并且不会管那些被配置成prototype作用域的Bean的生命周期。

了解Spring生命周期的意义就在于，可以利用Bean在其存活期间的指定时刻完成一些相关操作。这种时刻可能有很多，但一般情况下，会在Bean被初始化后和被销毁前执行一些相关操作。

在Spring中，Bean的生命周期是一个很复杂的执行过程，我们可以利用 Spring提供的方法定制Bean的创建过程。当一个Bean被加载到Spring容器时，它就具有了生命，而Spring容器在保证一个Bean能够使用之前，会进行很多工作。Spring容器中Bean的生命周期流程如图1.1所示。

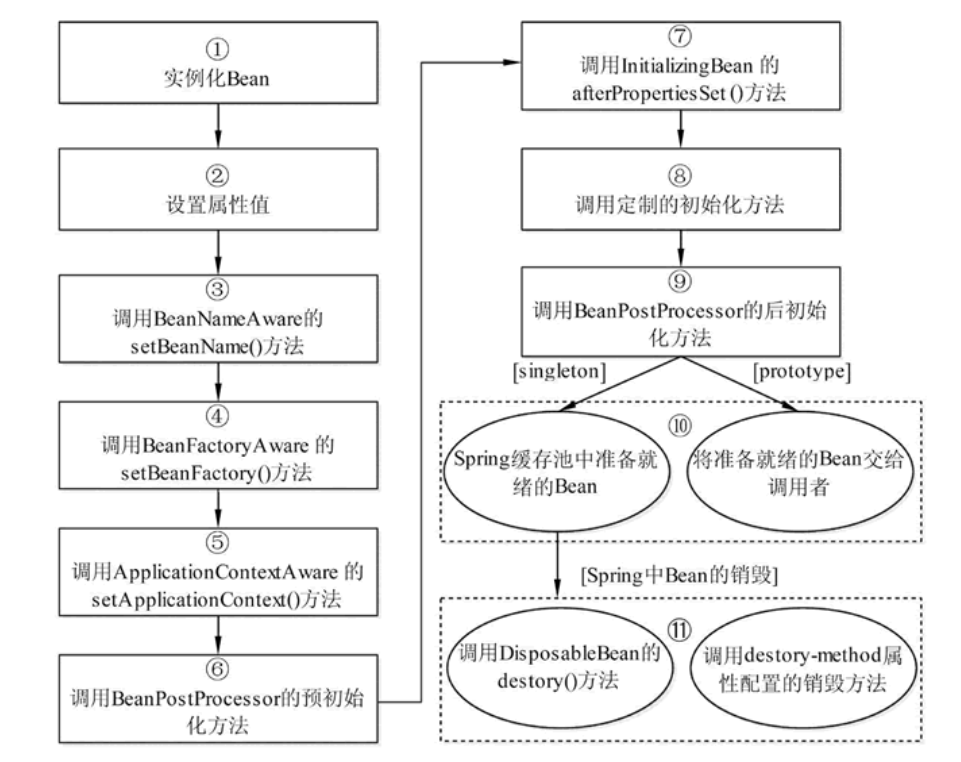


图1.1 Beans生命周期

### 1.1.1 Beans初始化和销毁处理（对应图1.1中7、8和11）

Beans 的初始化和销毁处理有三种方式：

1. 在XML配置文件中对Beans显式指定初始化和销毁方法名称（或者通过注解），并且指定后必须在对应的类实现方法。否则会抛出异常：无法初始化Beans;
2. 在Beans对应的类中实现org.springframework.Beans.factory.InitializingBean接口的afterProperties()方法和org.springframework.Beans.factory.DisposableBean接口的destroy()方法；
3. 在XML文件指定所有Beans默认初始化和销毁方法名称，但是默认方法的实现不是必须的(不同于对Beans显式指定的必须实现)。

如果这三种方法全部都实现了，那么执行顺序为：

首先检查Beans是否实现了接口的方法，如果实现了，先执行接口的方法；之后Spring IOC检查XML文件是否对Beans指定了相应的方法，如果实现了，执行相应的方法，并且覆盖XML文件指定的默认方法(如果有的话)；最后，如果对Beans没有指定相应方法则执行默认方法。

Java文件：



XML文件配置：



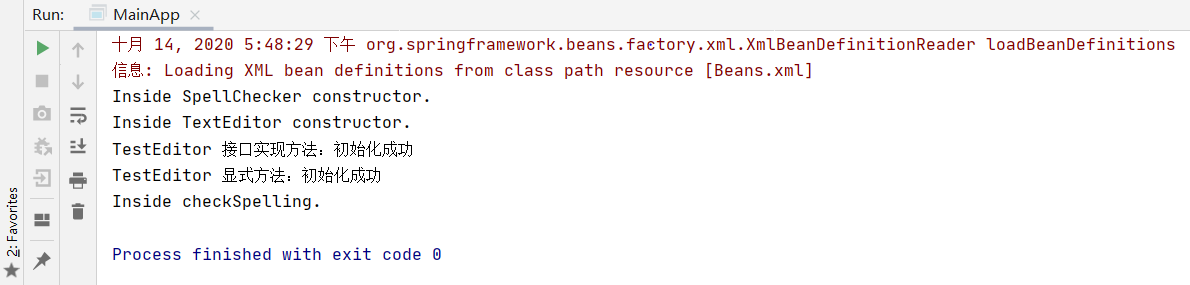


图1.2 生命周期回调执行结果

### 1.1.2 Beans后置处理器（对应图1.1中6，9）

BeanPostProcessor接口定义回调方法，可以实现该方法来提供自己的实例化逻辑，依赖解析逻辑等。也可以在Spring容器通过插入一个或多个 BeanPostProcessor的实现来完成实例化，配置和初始化一个Bean之后实现一些自定义逻辑回调方法。

BeanPostProcessor可以对Bean（或对象）实例进行操作，这意味着Spring IoC容器实例化一个Bean实例，然后BeanPostProcessor接口进行它们的工作。

ApplicationContext会自动检测由BeanPostProcessor接口的实现定义的 Bean,注册这些Bean为后置处理器，然后通过在容器中创建Bean,在适当的时候调用它。

函数原型：

@Override

public Object postProcessBeforeInitialization(Object o, String s) throws BeansException {

System.*out*.println("BeforeInitialization"+s);

return o;

}

@Override

public Object postProcessAfterInitialization(Object o, String s) throws BeansException {

System.*out*.println("AfterInitialization"+s);

return o;

}

注意：Spring AOP就是通过postProcessBeforeInitialization实现的。

实现后置处理器，并且在Beans.xml中注册。

Java文件：用来重写后置处理器，其他文件与1.1.1相同。



XML文件：



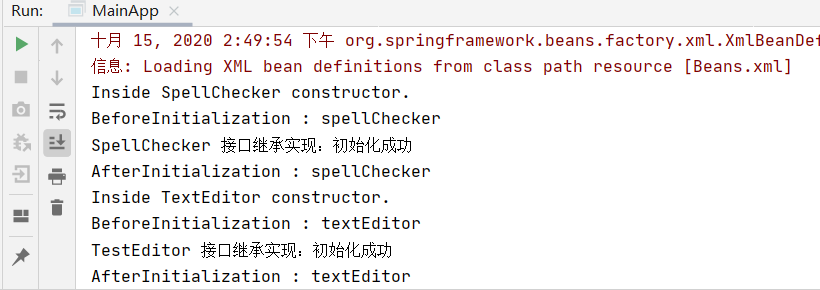


图1.3 重写后置处理器执行结果

### 1.1.3 Beans的singleton模式和prototype模式（对应图1.1中10）

1. singleton

单例模式，使用singleton定义的Bean在Spring容器中只有一个实例，这也是Bean默认的作用域。

1. prototype

原型模式，每次通过Spring容器获取prototype定义的Bean时，容器都将创建一个新的Bean实例。

在Beans的生命周期中，如果在<Bean>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<Bean>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.

Java文件：



单例模式XML文件配置：



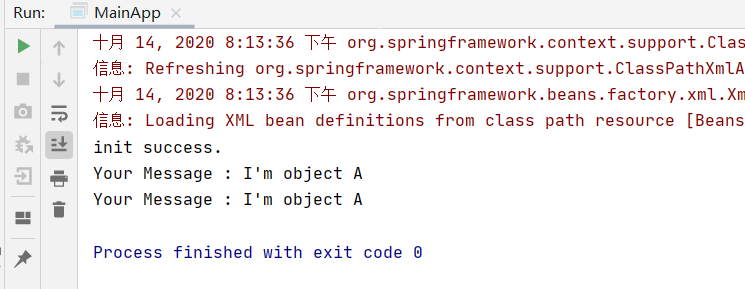


图1.4 单例模式执行结果

非单例模式XML文件配置：



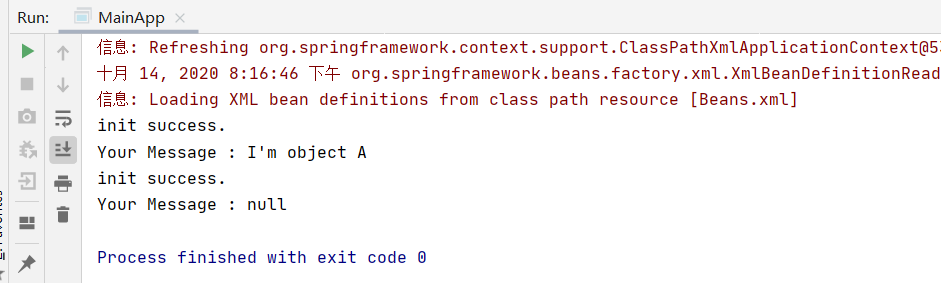


图1.5 非单例模式执行结果

由以上结果可知：

Singleton是单例类型，就是在创建起容器时就同时自动创建了一个Bean的对象，不管你是否使用，他都存在了，每次获取到的对象都是同一个对象。注意，Singleton作用域是Spring中的缺省作用域。

Prototype是原型类型，它在我们创建容器的时候并没有实例化，而是当我们获取Bean的时候才会去创建一个对象，而且我们每次获取到的对象都不是同一个对象。

### 1.1.4 Spring Aware接口（对应3、4、5）

Spring中提供了一些以Aware结尾的接口，实现了Aware接口的Bean在被初始化后，可以获取相应的资源（如Bean的名称）。通过Aware接口可以对相应的资源进行操作，为Spring的扩展提供了方便的入口。

下面展示了在HelloWorld.java中通过实现setApplicationContextAware接口获取当前IOC容器上下文。

Java代码与XML文件与1.1.3相似。

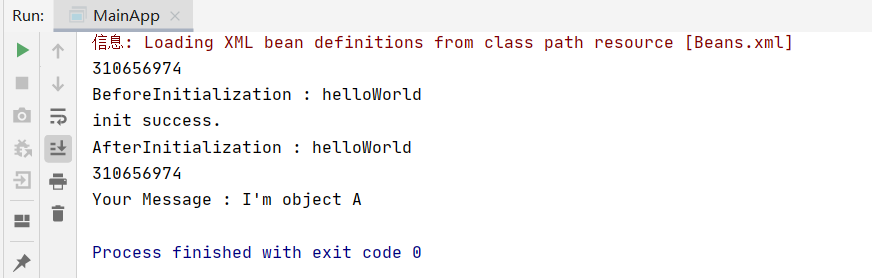


图1.6 获取当前Bean哈希码

结果表明在实现的Aware接口内Bean的哈希码和Main中获得的Bean哈希码相同，说明实现的Aware接口可以获取当前IOC容器上下文。

### 1.1.5 待解决的问题

#### 1.1.5.1 Spring销毁Bean的时候destroy调用

当Beans被销毁时，如果声明了需要调用回调方法destroy时，destroy会在Beans销毁后被调用，但实际上destroy并没有被调用（Bean的范围已经声明为singleton，即由IOC容器管理其生命周期）。

可能原因：应该跟Spring垃圾回收机制有关。

#### 1.1.5.2 Bean到底何时被创建？

Bean在生命周期什么时候才能被使用？在1.1.4创建实现Aware接口的方法中，利用getbean()方法成功获取了当前bean的哈希码。但是如果把bean的范围改为非单例模式，则控制台会在实现Aware接口的方法中抛出异常信息：循环依赖。

分析原理：由于单例模式下一旦IOC容器创建，则IOC容器会自动实例化所有单例beans,所以在Aware接口中调用getbean()获取当前bean本身时，调用成功。但是如果在多例模式下，在Aware接口调用getbean()时，此时该bean本身正处在创建过程中，在这个时候再次调用getbean()，会再次创建这个bean，产生自身循环依赖。

经过测试，删除单例模式下主函数的getbean()，观察IOC容器创建时实例化单例beans的行为，发现在bean的生命周期到第9步之后，bean才能够被使用。

# 2 Tutorial About GNN : Theme I 学习

学习中…

# 3 Network Representation Learning with Rich Text Information

## 3.1 论文写作背景及目的

表示学习已经在很多项目任务中表现出了它的功效，比如图像识别或文本采集。网络表示学习旨在对于每个节点的进行矢量表示，这种方法慢慢被认为是网络分析里很重要的一部分。大多数网络学习方法都通过探讨网络结构来学习。事实上，网络节点也包含了很多信息（比如文本信息），但是却没有在典型的表示学习方法中很好的利用起来。本文证明深度游走（DeepWalk）等同于矩阵分解（MF）,并提出了text-associated DeepWalk（TADW）。TADW将文本特征的节点在MF的基础上合并进网络表示学习中。最后文章用多标签分类检验TADW的有效性。

目前很多图表示学习工作只考虑了图结构，但是在现实之中，图的节点包含了大量信息，例如维基百科的知识图谱的节点包含大量文本信息。有一种简单的方法就是单独的学习图结构和节点信息再把得到的表示结果拼接起来，但是这样忽略了节点信息和图结构之间存在密切的联系，往往效果很差。而DeepWalk在随机游走的过程中又很难处理节点信息。

文章主要有两个方面的工作：

1、证明DeepWalk与MF的等同性；2、通过MF与节点信息的结合创造性的提出了TADW表示学习模型。

## 3.2 相关的工作

表示学习广泛运用于知识表示学习，计算机视觉，NLP和图表示学习之中，但是在此之前的工作都没有很好的处理节点信息的问题。有一些方法例如NetPLSA用作主题模型的建模，这些方法结合了网络和网络节点的关系。本文把这些方法作为baseline进行对比。

DeepWalk通过随机游走生成节点序列，运用word2vec的思想，通过skip-gram模型训练最大化中心节点的上下文节点出现概率，从而得到节点的低维表示。

## 3.3 主要内容

### 3.3.1 问题建模

1. 图的定义：.

结点：.

给定一个图G，模型目标是把每一个图结点都映射到维度为k的向量中，其中.

1. DeepWalk转化为MF的模型分析

给定图,DeepWalk的实质是把矩阵分解为和.如图3.1所示：



图3.1 DeepWalk矩阵分解

其中中每个元素表示从节点随机游走t步到节点的概率对数，W矩阵k维列向量就是所有节点v的低维表示，H矩阵k维列向量就是所有节点v作为上下文时的低维表示。

1. TADW模型

给定图,TADW实质是把分解为,,.最后把W矩阵k维列向量和矩阵k维列向量拼接起来就是最后的节点v的2k维向量表示。T矩阵是代表节点的文本信息。如下图3.2：

**

图3.2 TADW矩阵分解

### 3.3.2 DeepWalk与MF等价证明

#### 3.3.2.1 基本概念定义

D：随机游走节点对，D中的每一个元素是,其中v是中心节点，c是上下文节点。V是中心点合集，是上下文节点合集。通常情况下V=.

对于节点对,定义是节点对出现次数，那么有：

分别代表v,c在D中出现次数。

矩阵是DeepWalk中embedding得到的中心节点向量矩阵，是上下文向量矩阵，对应于skip-gram模型中隐层矩阵和Softmax输出层矩阵。如下图3.3：

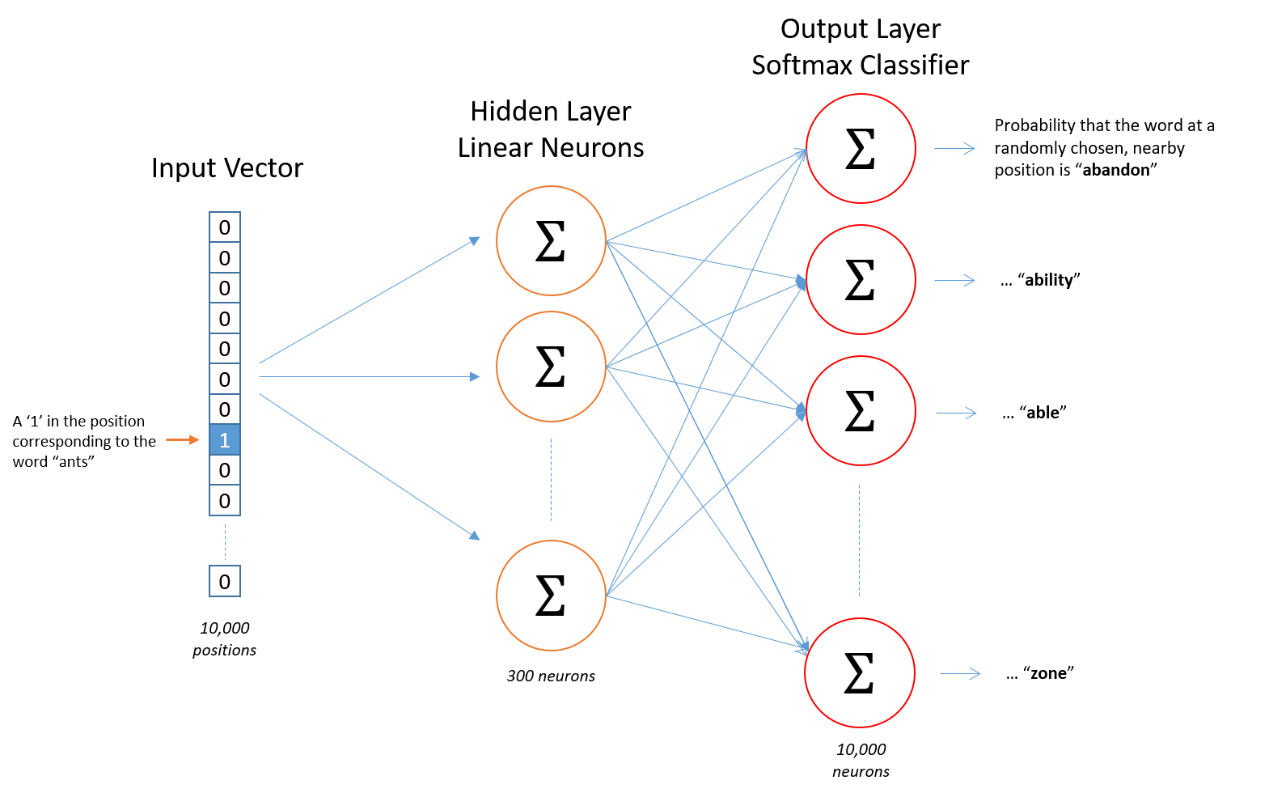
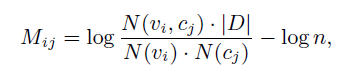


图3.3 DeepWalk中skip-gram模型

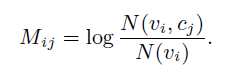
目标是找到一个矩阵使得.

#### 3.3.2.2 证明过程

首先说明了skip-gram with negative sampling(SGNS)的实质是将矩阵M分解（证明过程已经由Levy and Goldberg,2014完成），证明前提是得到的向量维度k足够大。给定M矩阵中各元素值定义：

 （3.1）

其中n是negative sampling关于每组的负采样数目。类似的，现在本文给定skip-gram with Softmax的M矩阵定义：

 （3.2）

接下来按照DeepWalk的采样方式，讨论,以及对于每次采样,是这次采样的中心节点，是上下文节点，把加入D中，条件是,t是采样窗口大小。所以节点作为中心点将会在D中记录2t次。

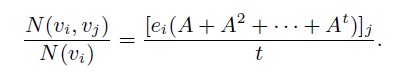
显然是代表v在D中出现的频率，实际上就是v的PageRank值。同时定义代表在以t为大小的窗口内出现次数。下面计算：

* 计算方法：

定义PageRank转移矩阵A,.是节点i的出度。

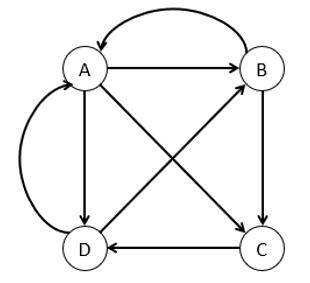
定义为节点i的初始向量，为one-hot行向量形式。

假定从i节点开始Random-Walk,那么的第j维数据表示从节点i到节点j的概率。所以以此类推的第j维数据表示经过t步从i到j的概率。所以是节点j出现在节点i右侧窗口的可能次数。给出计算公式：



（3.3）

Example：



,,,

结论：证明了DeepWalk本质上就是M矩阵的计算。并且给出了M的计算公式。

### 3.3.3 TADW实现

### 3.3.3.1 低秩矩阵分解

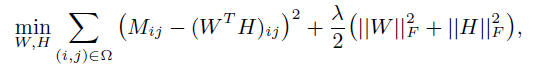
目的：是指用来近似矩阵M，那么的元素就可以用于估计M中对应不可见位置的元素值，而可以看成是M的分解，所以称为矩阵分解。

前提：矩阵M的秩要远远小于其阶数。

限制：约束最优化问题是NP-hard问题。

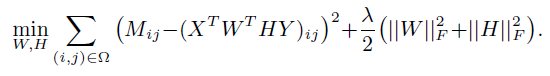
解决方法：用损失函数作为训练使得结果逼近M。

本文给出损失函数定义如下：

（3.4）

其中后面一项为正则化项。

但是式（3.4）没有把矩阵M的特征信息包含在内，于是文中提到了一种称作inductive matrix completion的矩阵补全方法，可以把表示矩阵特征的向量整合到损失函数的计算中。

（3.5）

其中和中的表示M矩阵中第i个特征单元的特征。

### 3.3.3.2 TADW模型的提出

受（3.5）式inductive matrix completion方法的启发，在一个图中，给定其节点特征矩阵,其中T的第i行列向量表示节点的特征向量。

由于直接计算M矩阵的时间复杂度为（计算公式见（3.3）,当t非常大时）。参考在DeepWalk中，为了减少训练复杂度，DeepWalk采用了基于采样的随机游走，当DeepWalk获得的游走序列越多时，准确度越高，性能越低。所以在TADW中，计算M矩阵采用了一种折中的方法，令,.且现实中的网络多为稀疏网络，在时，M的计算复杂度为,矩阵分解方面，选择对M矩阵而不是进行分解，原因是矩阵分解的训练复杂度与M矩阵中非零元素成正相关，如果矩阵为密集矩阵，甚至可以直接对A矩阵（PageRank转移矩阵）进行分解。

受（3.5）式的启发，给出TADW损失函数：

（3.6）

通过训练得到W和作为节点的低维表示向量，由于两个都能作为节点的向量表示，文中把两个k维向量连接起来作为节点的表示向量。