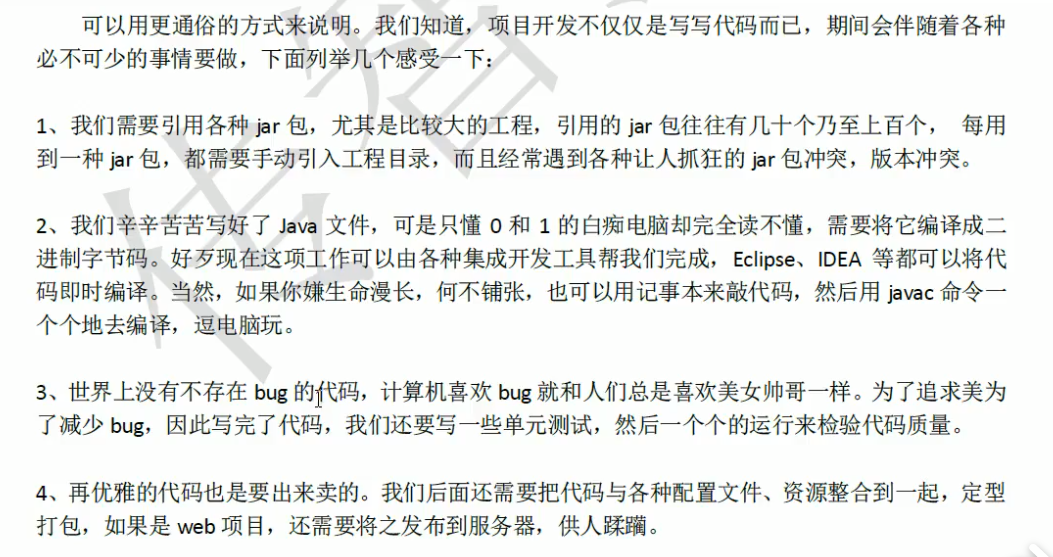
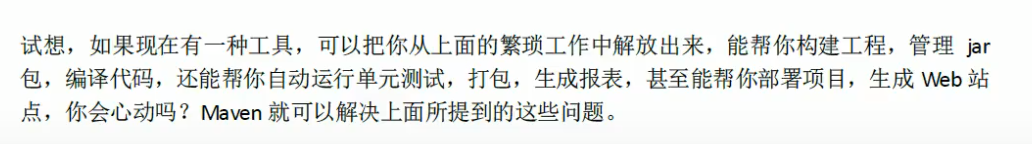
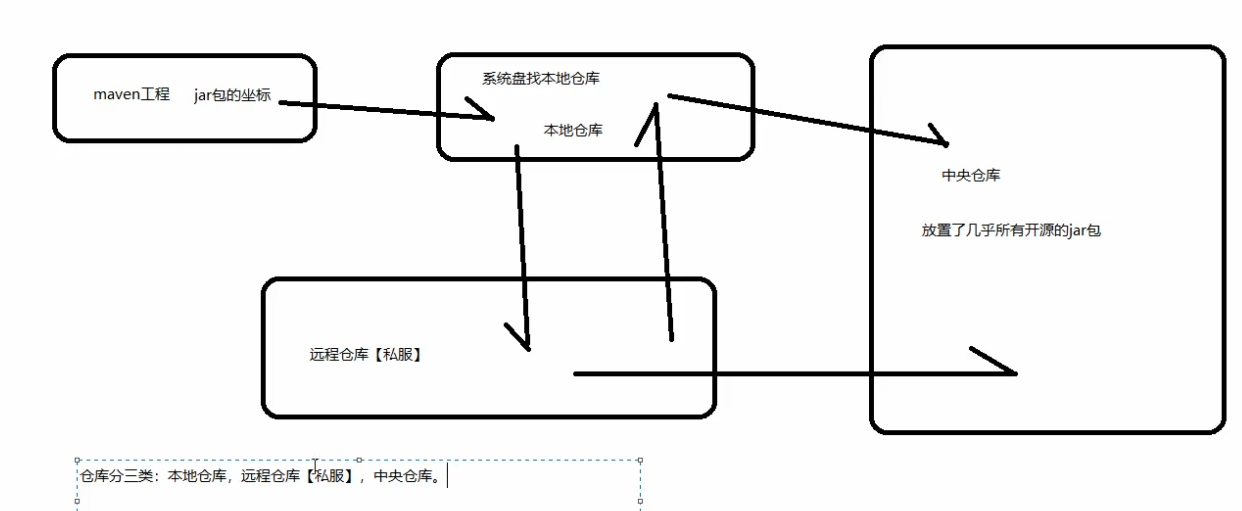
#maven

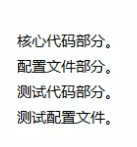




Maven仓库的依赖关系：

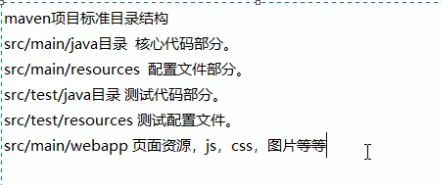


Maven标准目录结构

一个工程由四部分：

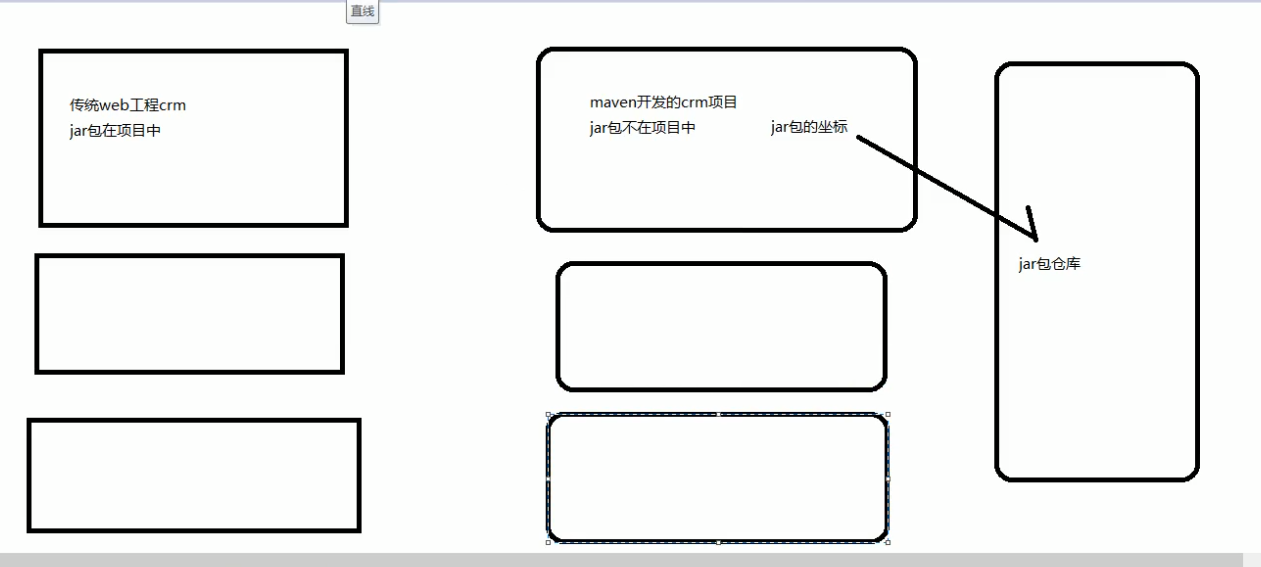
传统的项目结构，就一个src文件夹，四部分代码都放进去，这布盒里

Maven项目结构：



#依赖管理（maven对jar包的管理过程）

Maven如何节省空间、

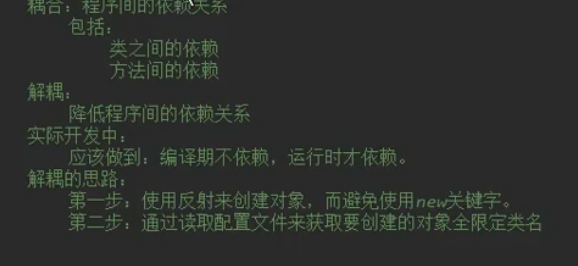


不仅节省空间 也实现了代码重用

#程序的耦合



注册驱动降低耦合的例子



Day01这个简单的模拟账户项目，若按传统的方式来写：

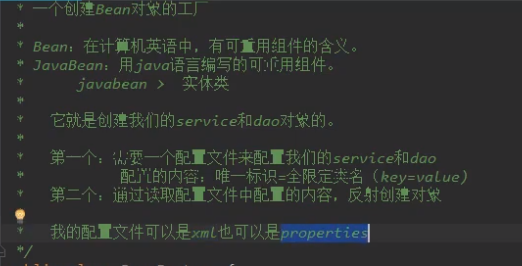
Ui层调用业务层，要new一个业务层对象：

业务层调用dao层，要new一个dao对象：

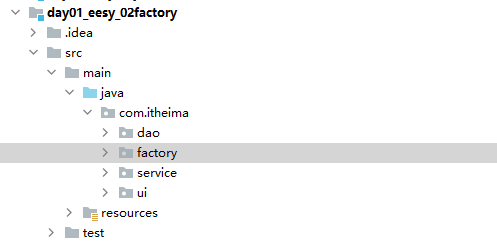
好嘛，这耦合关系简直灾难。为了避免这个情况（一堆new），我们引入：

工厂模式

创建一个bean对象工厂：



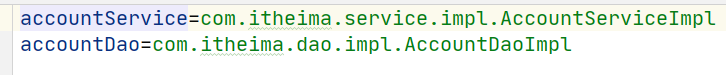
也就是说项目结构变成了这样：



核心代码中多了factory这一“层”

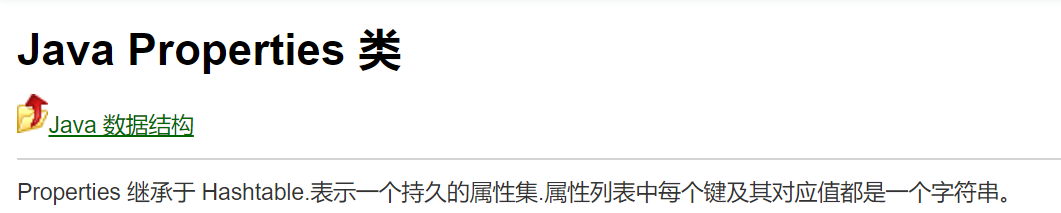
前面学习maven结构时我们知道：配置文件应该写在

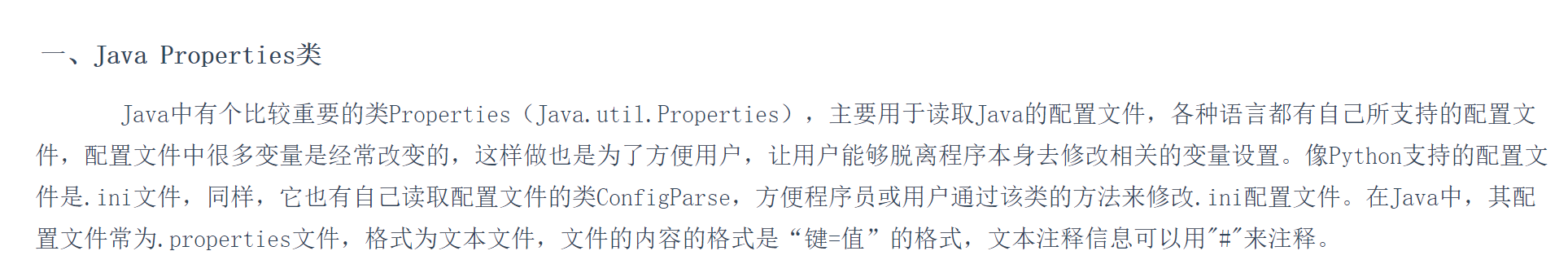
配置文件中要写的东西很简单：就是给我们要用到的bean设定一个标识：



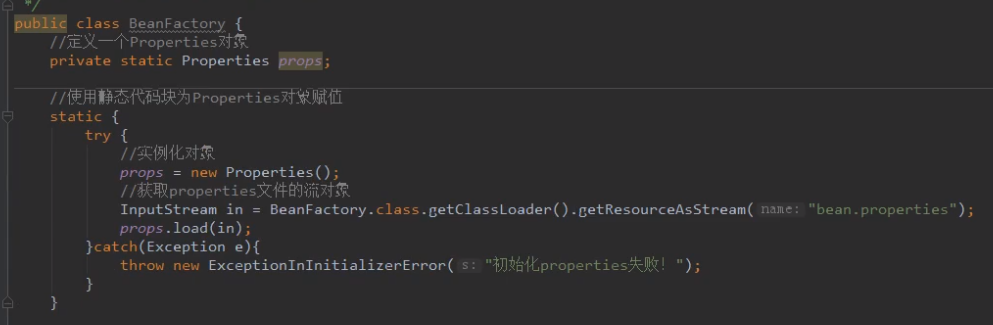
然后我们回到factory.java中，建立工厂。

在建立工厂之前，我们首先学习一个java类：Properties类。





不难发现，在java中想使用自己写的配置文件，需要借助这个类。具体而言：



接着我们需要写一个getbean（）方法来从工厂中获取bean对象。

需注意，这个方法的返回值应当为object，因为你的工厂中有不少bean，你也不知道返回的是啥类型，写为object更具通用性。



然后我们就可以在对应的层里使用它来创建对象啦：





这里其实有一个悬念，getBean（）中，通过反射创建的对象bean应当是一个String类型，getBean返回的是一个String->Object类型，但却可以直接强转为IAccounetDao？！

悬念解决：其实上面代码的备注就写错了，应该是这样：

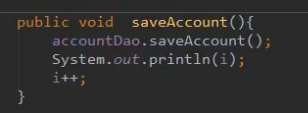


需要注意，ui层在调用service层时，每次都是新建一个service的，



as是多例对象.

我们可以验证：（i是定义在类里面、方法外面的变量）

在IAcounnt中的方法加上i++，在ui层中循环调用五次工厂建立五次service层，发现都输出1，也就是说每次调用的都是新建的service！

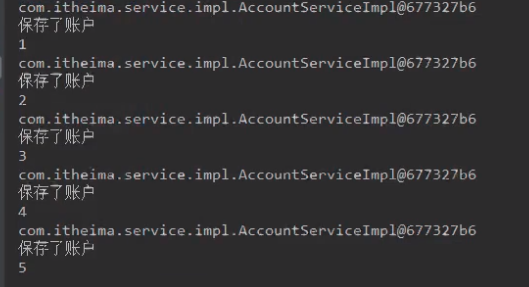
我们希望改进算法使得service层的对象是单例对象

因此在factory中我们定义一个beans用来存放每次new出来的bean，若干个bean就可以整体一起存放成为单例对象。



Bean的获取也很简单了：





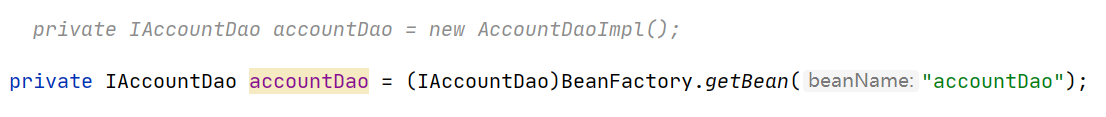
变成了单例对象！

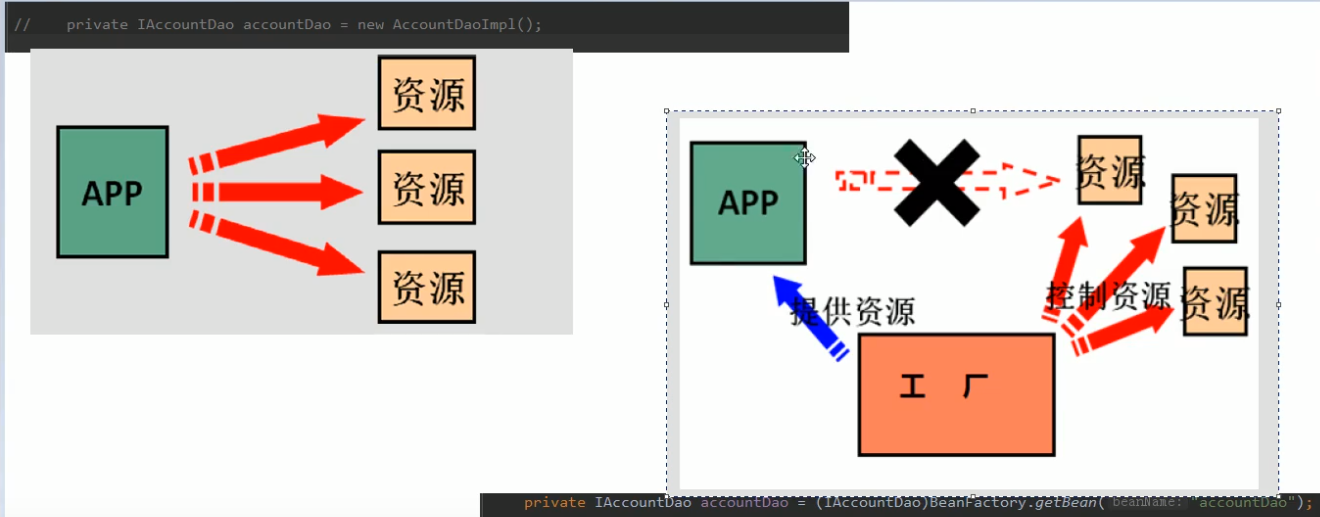
当然了，上述情况是我们把变量i定义在了方法外，如果你把i放在方法内，还是五个1.

#IOC控制反转

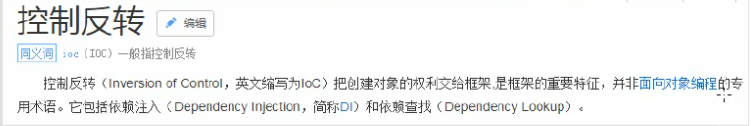
工厂模式大概了解了，那么什么是ioc呢？

我们看看service中两种创建dao对象的方式：





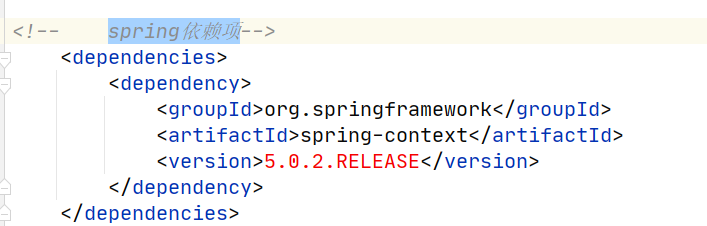
概念



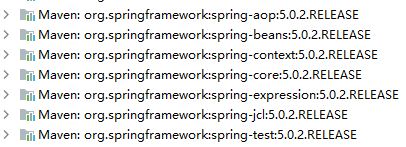
在这个例子中，factory是我们自己编写的，然而有了spring之后，factory不用我们自己编写，spring就可以替我们搞定。

#spring的配置

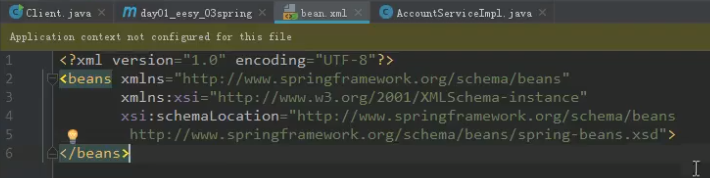
新建一个maven工程，首先要在配置文件中设置spring的相关依赖如下：



然后就会发现工程的依赖项External Libraries里多了这些：

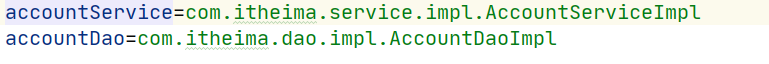


接着我们需要在source文件夹中导入约束（是一个xml文件），自己新建bean.xml。约束代码当然不死自己写了，上spring文档点开core，搜索xmlns，就得到了xml格式的约束。



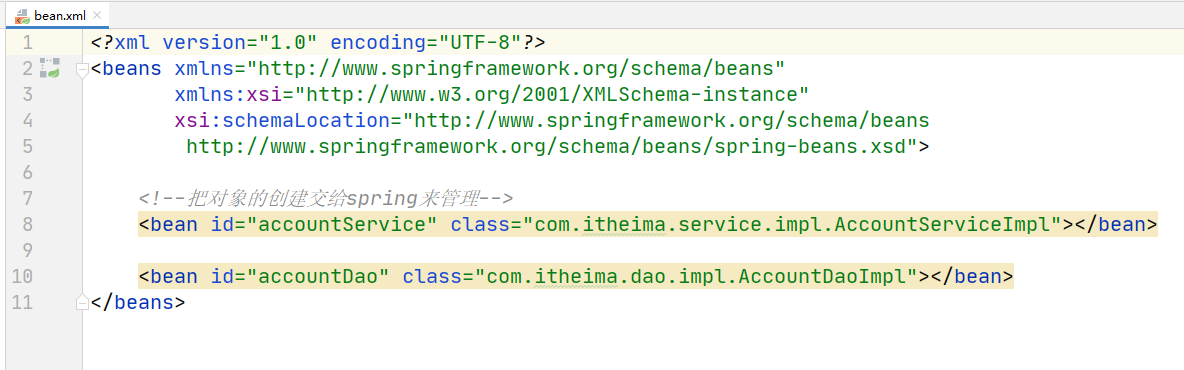
第三件事，需要把创建对象交给spring来管理，也就是像例子一中那样建立工厂。

例子一中我们自己写了bean.properties配置文件来给bean起了一个映射名字：

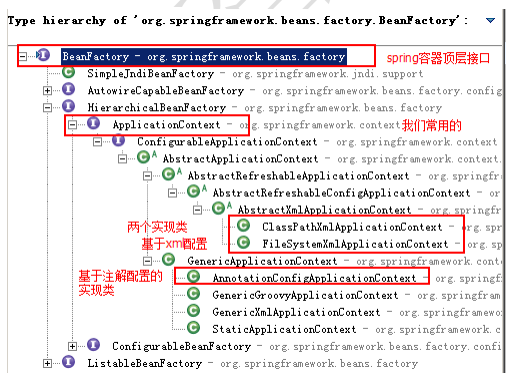


在这里，利用spring做相同的事情（建立 映射名字—路径 的键值对）

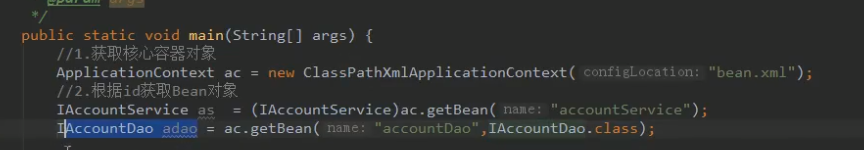
代码如下：（事实上这不是唯一的方法，下面会提到）



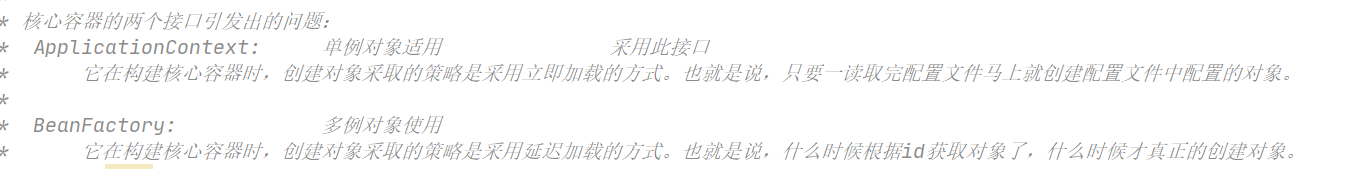
Spring自带的工厂结构如下：



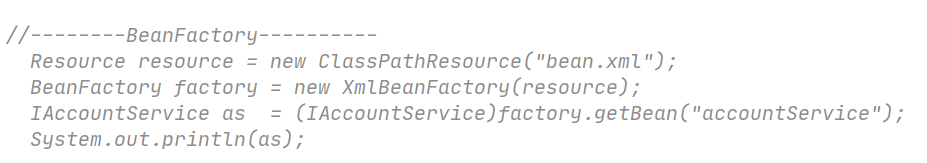
.xml配置文件写好后，就可以在三层中使用工厂啦：



事实上，使用spring的工厂有两种方式：

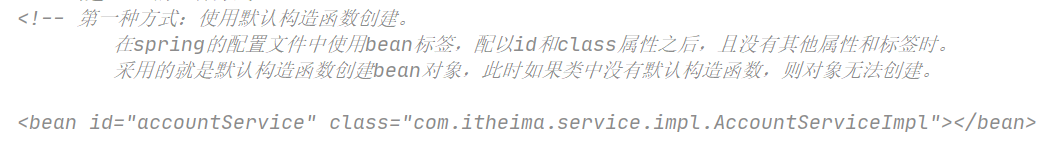


Beanfactory的使用方法（早已过时！）：



#spring对bean的一些管理细节

Spring创建bean的三种方法：



你要用的那个类必须有默认的构造函数

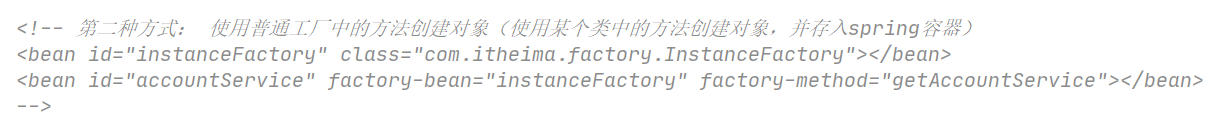
但如果咱调用别人的类，也不知道它有无默认构造函数也无法修改，咋办呢

别人写的jar包一般类似这样：



里面有很多get方法，调用不同的get方法就可以得到不同的类。

Spring配置文件这样写：

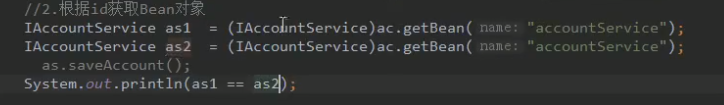


在三层里怎么调用，和第一种生成方法相同！（获取核心容器、根据id获取bean对象）

还有一种情况，工厂是静态的。连new一个工厂对象的过程也省略了，如下：



用工厂getbean多次获得同一对象：



在ui层中通过工厂getbean（）我们可以发现：



构造函数只走了一次，自然获得的两个对象完全相同！

这就引出了一个问题：bean的作用范围（常用的是前两个

）：

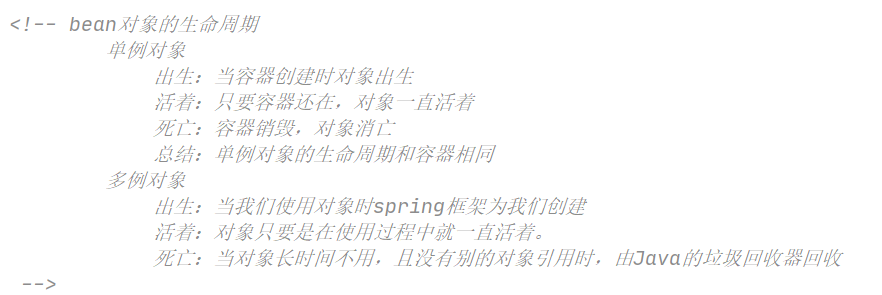


验证：把bean写成这样

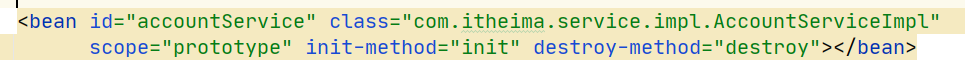


再测试上面的例子，构造函数跑两次，返回false！

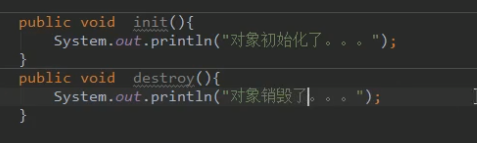
Bean对象的生命周期



为了验证，我们将bean标签修改为：

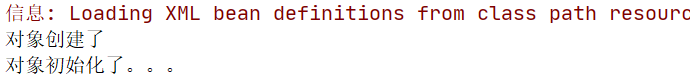


并且在对应对象中加上：



这样一来，getbean时自动执行那两个函数。

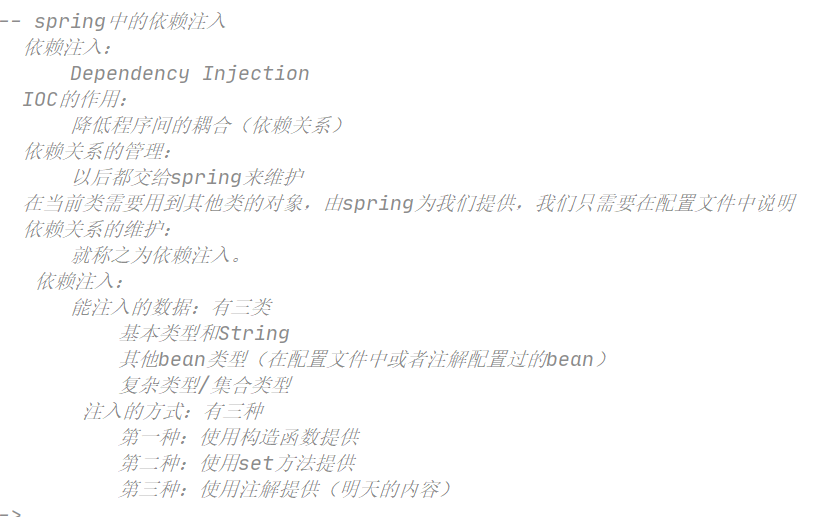
打印发现：



没有销毁！？

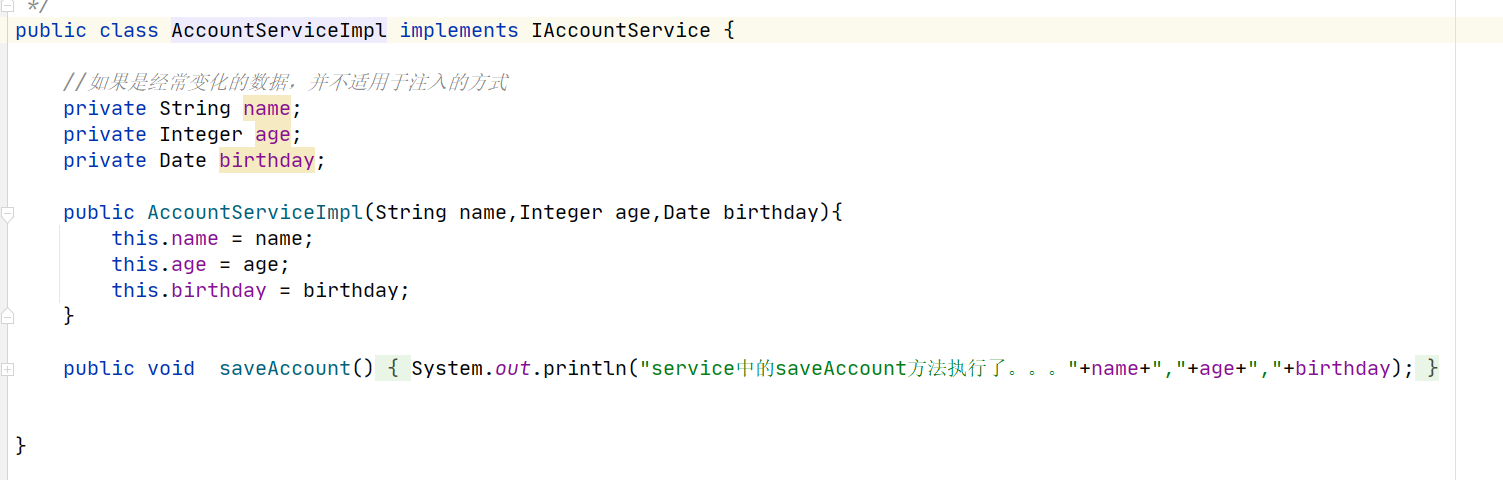
因为main一结束，所有内存都自动释放了，bean还没来得及destroy已经被释放掉了。

#依赖注入（DI：Dependency Injection）



第一种：使用构造函数

首先我们在service中：

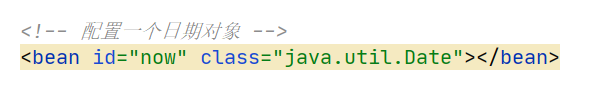


（此时他没有默认构造函数了！）

在bean.xml中：



最常用的肯定是name（可以唯一定位一个参数），其它的知道就可。

Birthday属性是一个date类型，不是基础类型或String，直接value输入肯定是不行的，这就要用到ref。now是我们自己配置的date类型的一个对象：

用bean这么一配置，就会有一个名为now的date类型对象被存放在spring核心容器中。

第二种：使用set方法

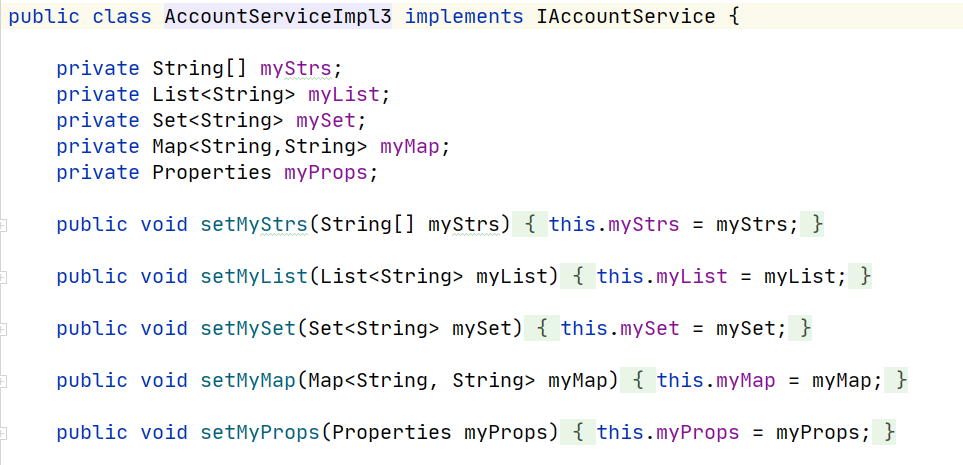
Service中：



Bean.xml中



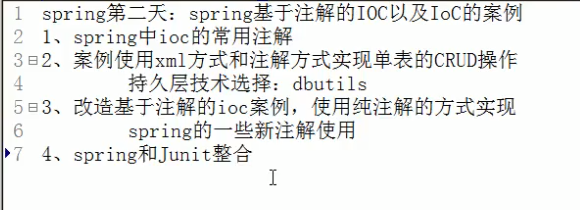
第三种：注入集合数据







#基于注解的ioc



首先要明白，注解和xml做的事情是一样的，就是配置。只是用不同的方法做同一件事。

