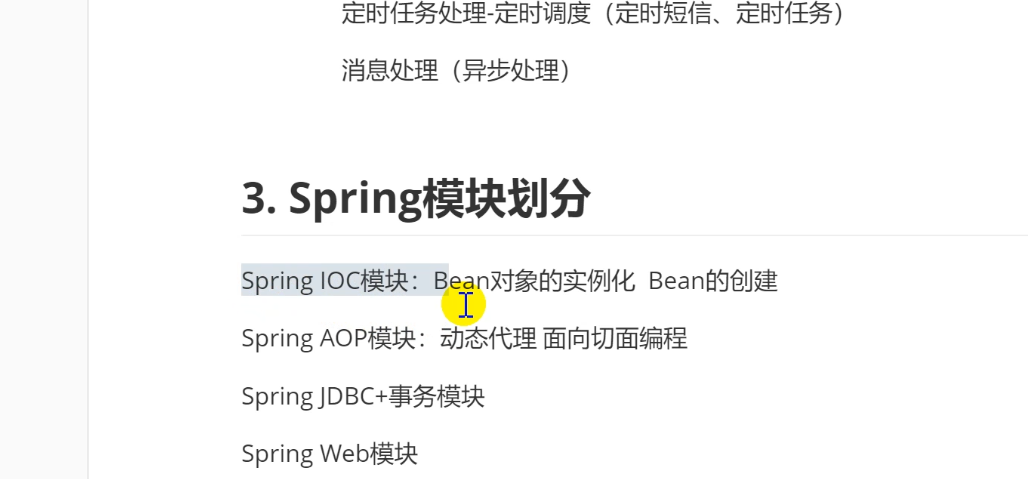


Aop aspect oriented programming



# Spring ioc 容器实现



m

Maven spring pom依赖



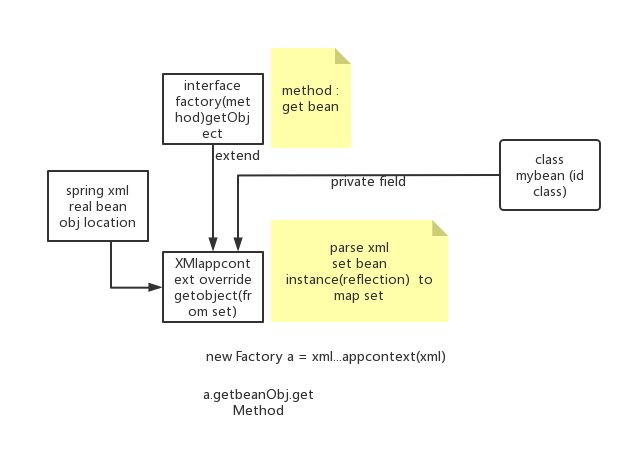
继承接口

模拟spring ioc 实例化bean 对象

创建 Beanfactory interface 写一个get object 方法

实现beanfactory 接口





## 核心就是反射

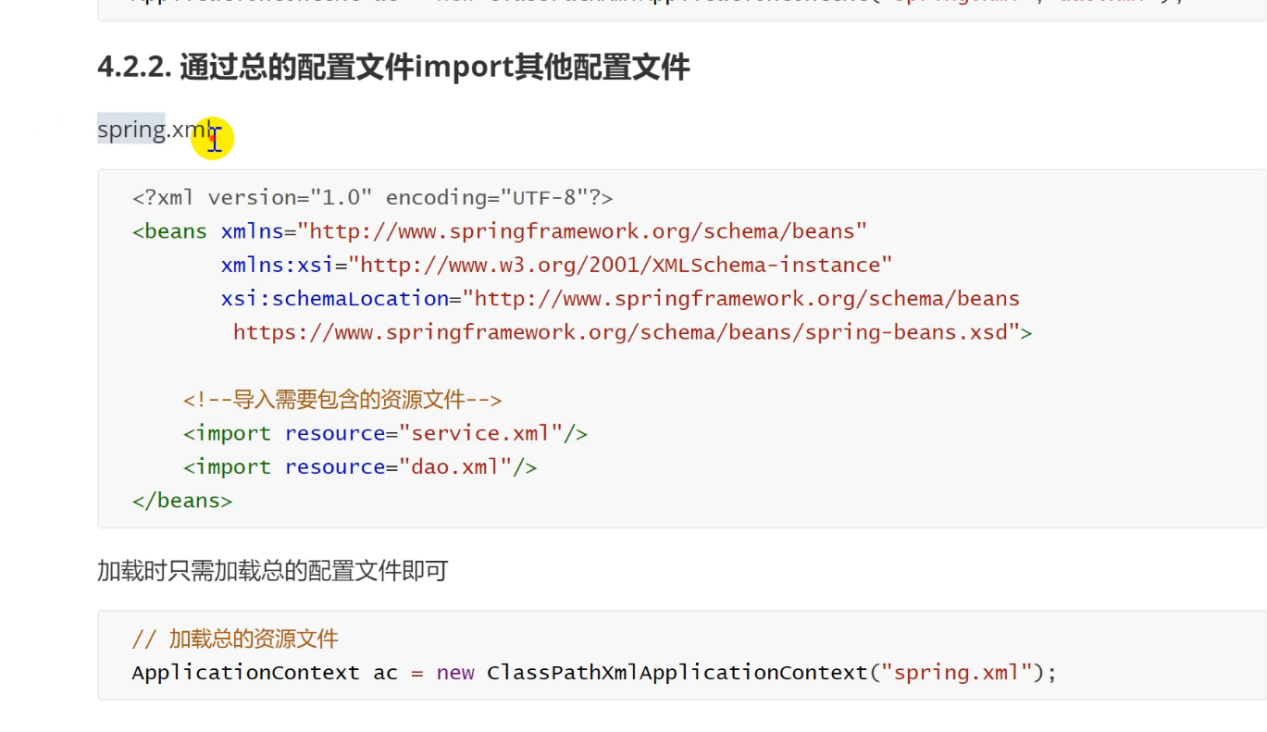


 Java中XPath的应用 XPath是W3C的一个标准。它最主要的目的是为了在XML1.0或XML1.1文档节点树中定位节点所设计。

dom4j是一个Java的XML API，是jdom的升级品，用来读写XML文件的。dom4j是一个十分优秀的JavaXML API

Bean.xml 文件要放在resource 文件夹下

Ioc 实例化的对象都是单例

通过总配置文件导入其他xml 文件

# Spring ioc 对象实例化



将xml 中的class 指向bean 工厂





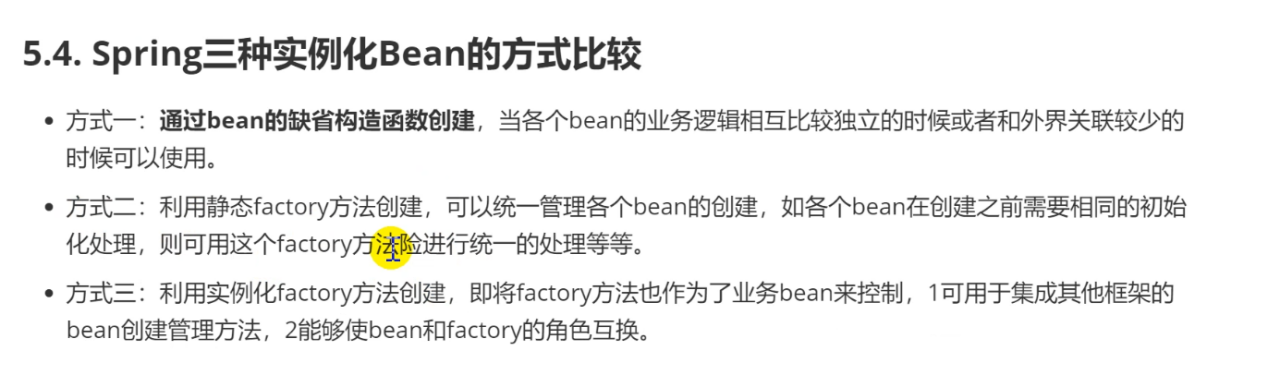
## 静态工厂实例化



优点;在创建实例对象前可以做其他事情；在工厂方法中添加。

## 实例化工厂实例化



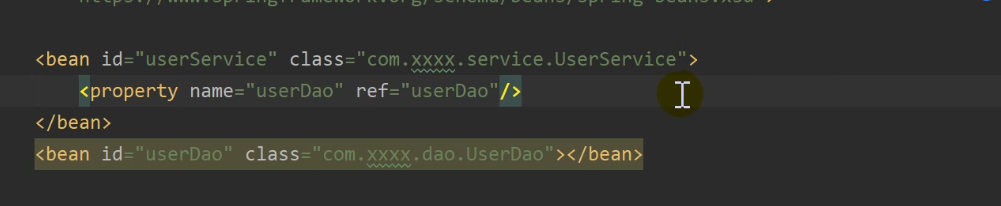


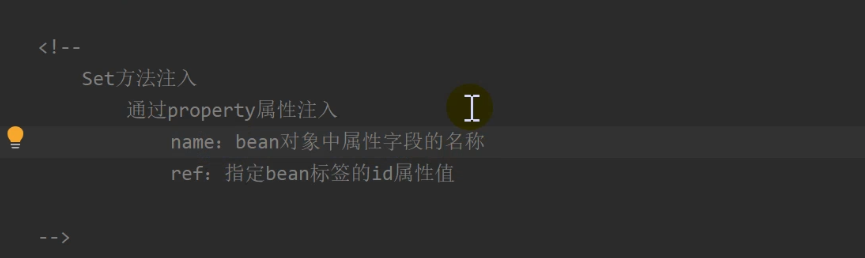
# Spring IOc 注入

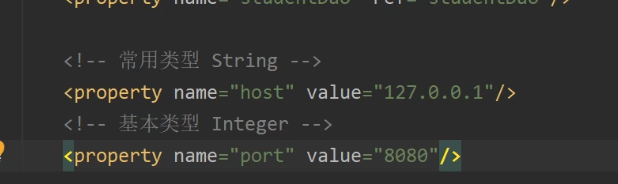


## 注入指的是一个bean 的对象在另一个bean 类中维护并使用

UserService 中想要创建并维护一个userdao 对象 需要在spring xml 文件中配置



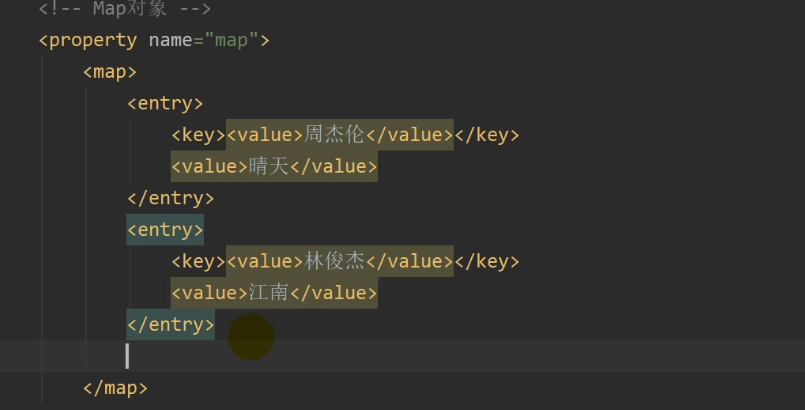


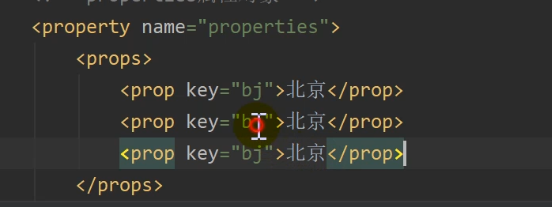


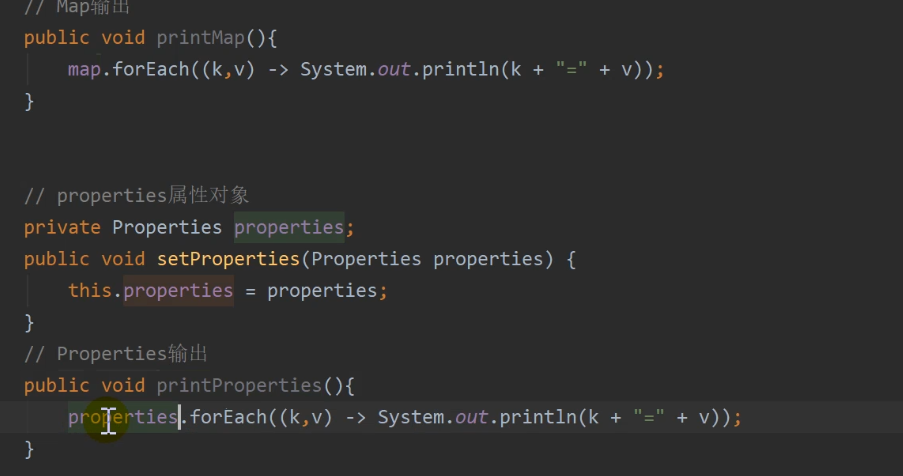




不管什么类型 都需要写上set 方法

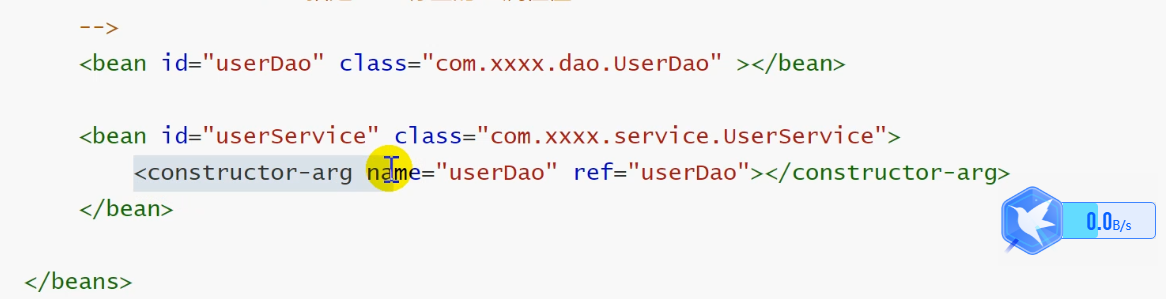


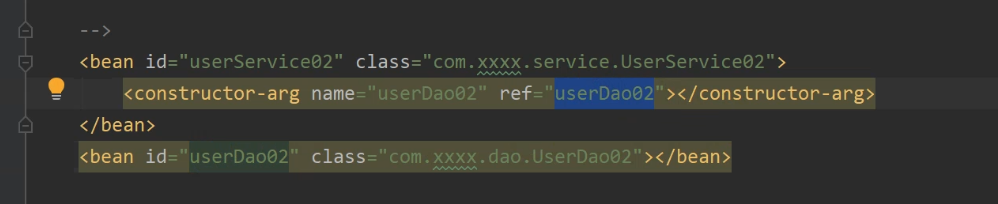


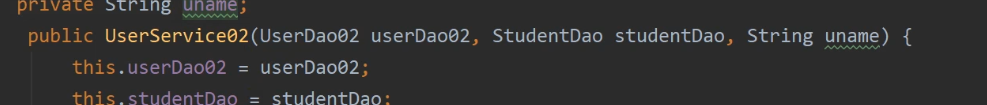


## 通过构造器注入

通过父bean 的构造器传入 bean 各种类型









# 循环依赖问题 （构造器注入产生）

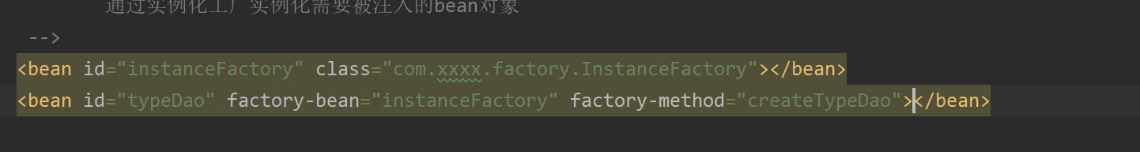
a inject b b inject a waiting each other to be instanced

解决： 使用set 注入

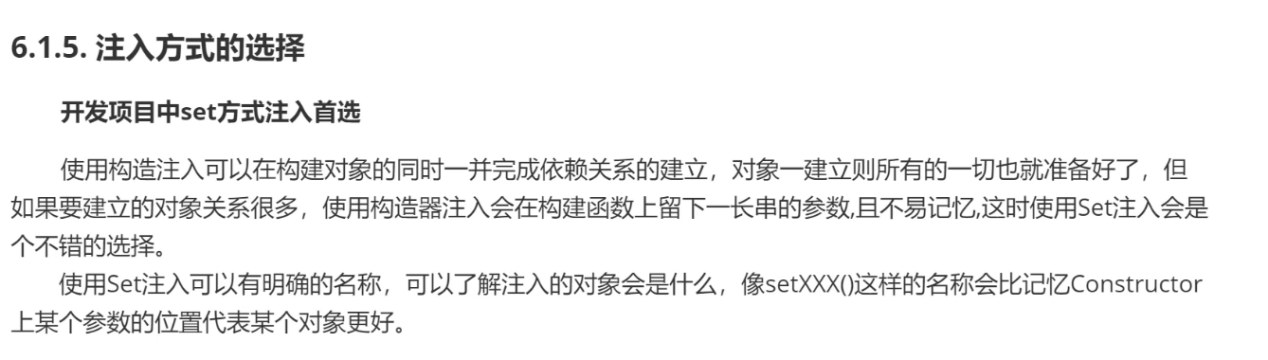
# Static factory injection



INSTANCE FACTORY INJECTION



注入方式选择

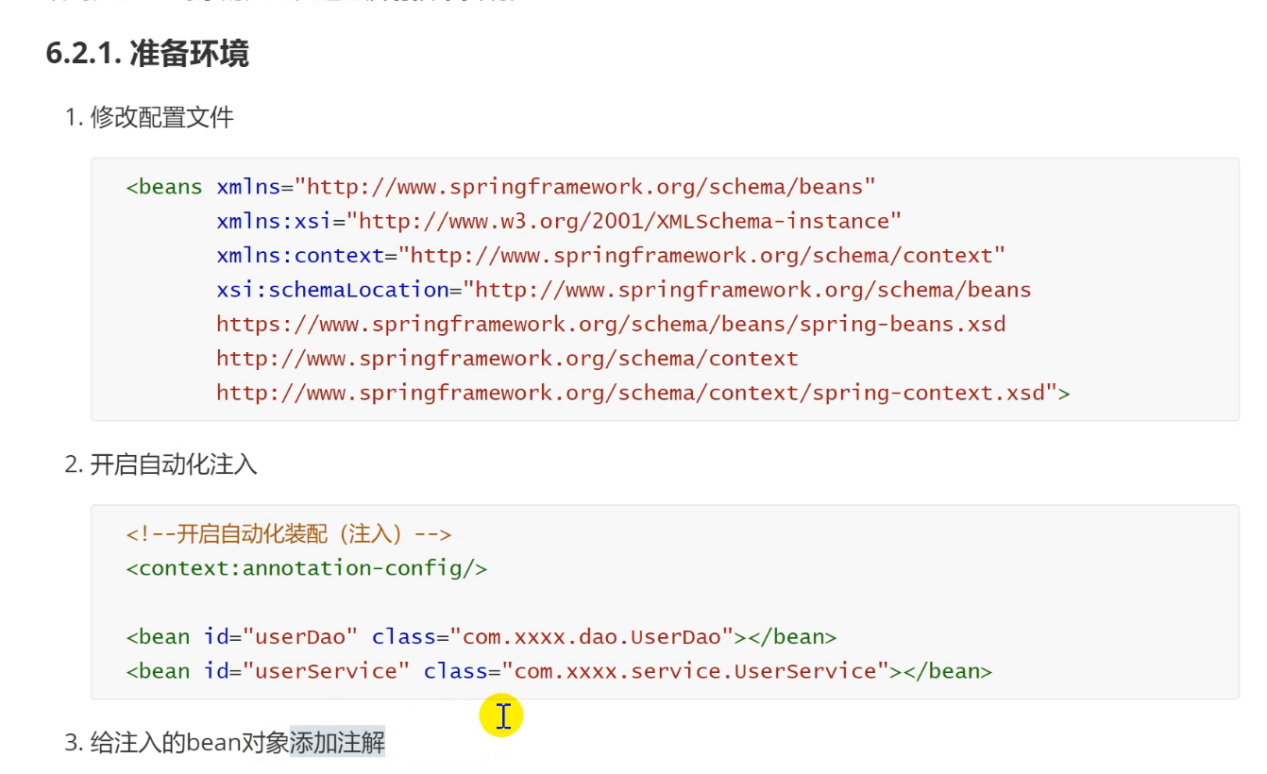


命名空间方式注入



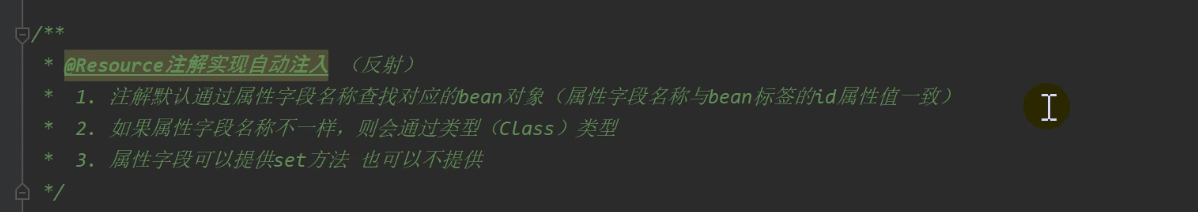


# Ioc自动注入



在对应的属性上加注解 就可以自动注入

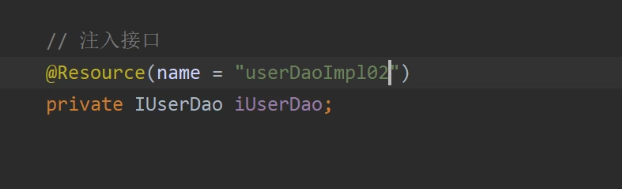




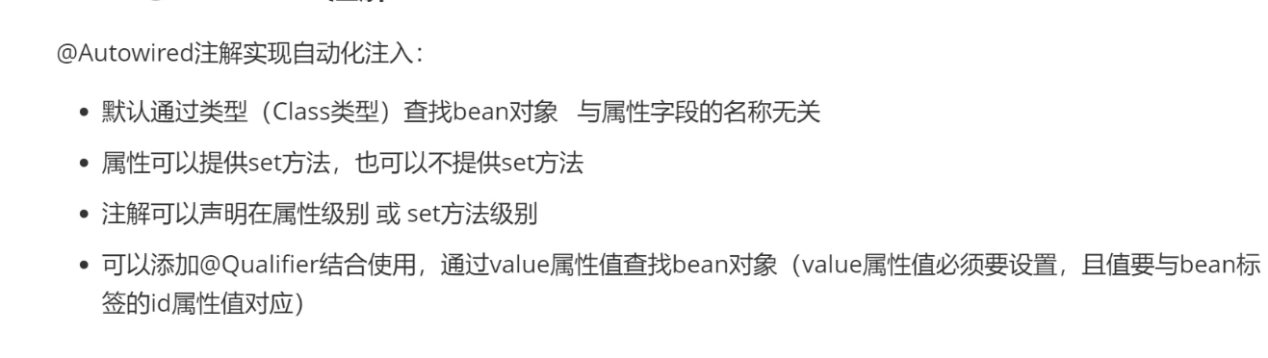


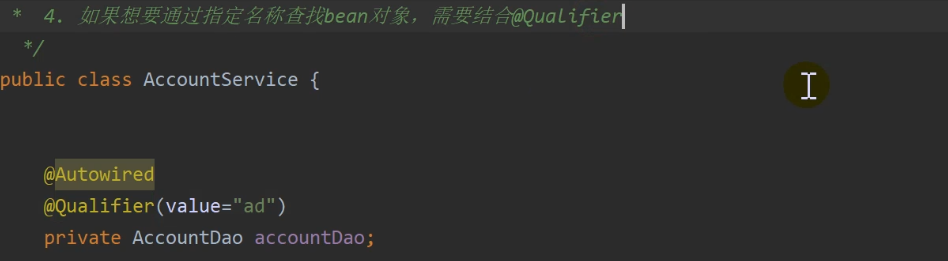


注入接口 要写明接口的实现类

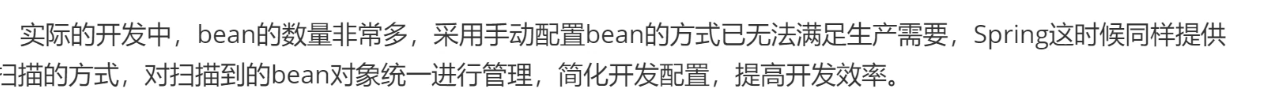




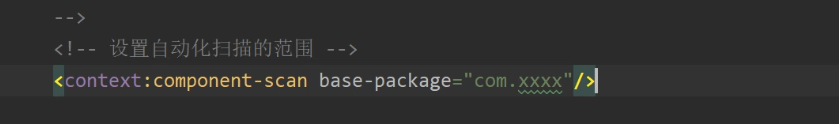




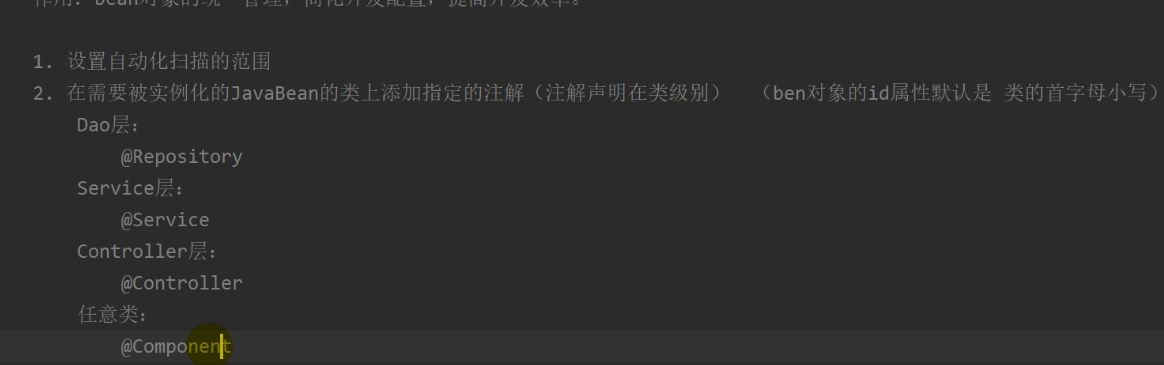
Ioc扫描器







Controller service repository 三层注解 层层注入 处理业务





# Spring 模拟用户登录

MessageModel （满足条件设值）

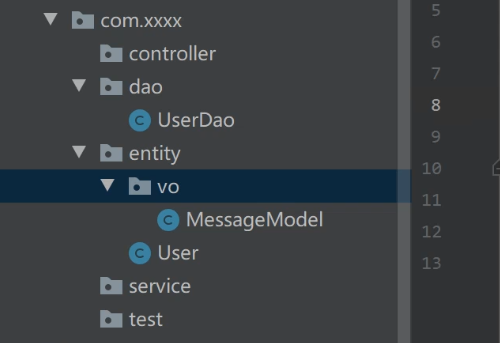
Entity user 数据库中赋值 不需要bean 容器处理

Userdao 查出 返回user 对象 需要bean

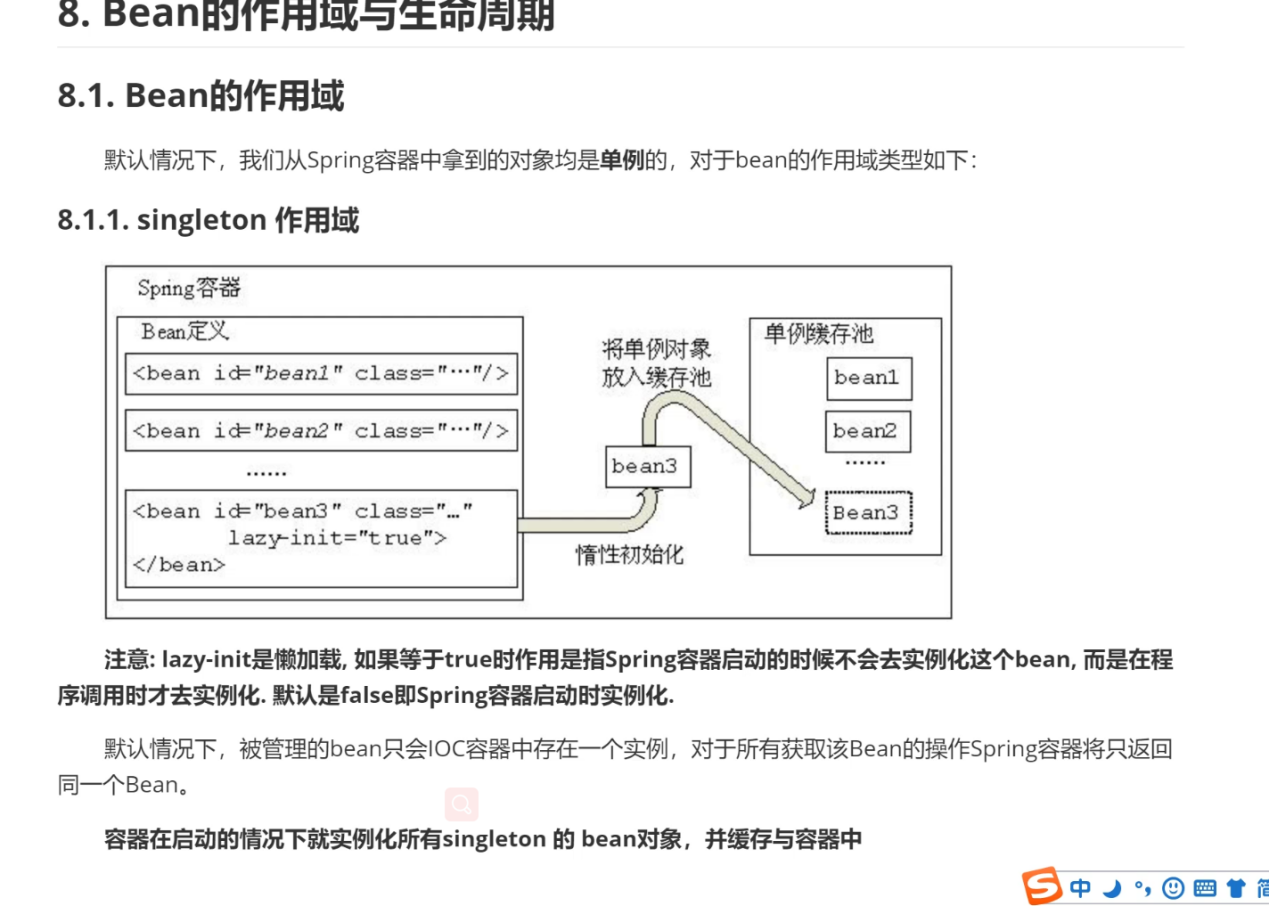
Service 需要bean 返回MessageModel 对象

controller 需要bean

Userdao repository



# Bean 单例模式的作用域



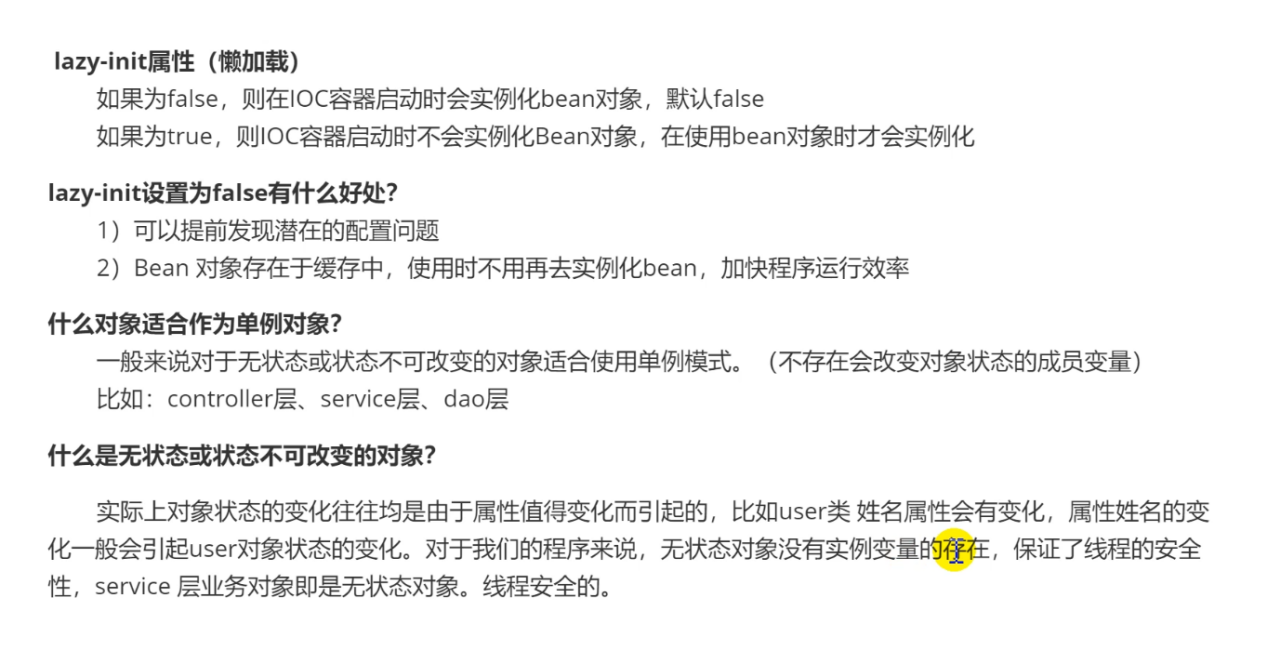
# Bean懒加载

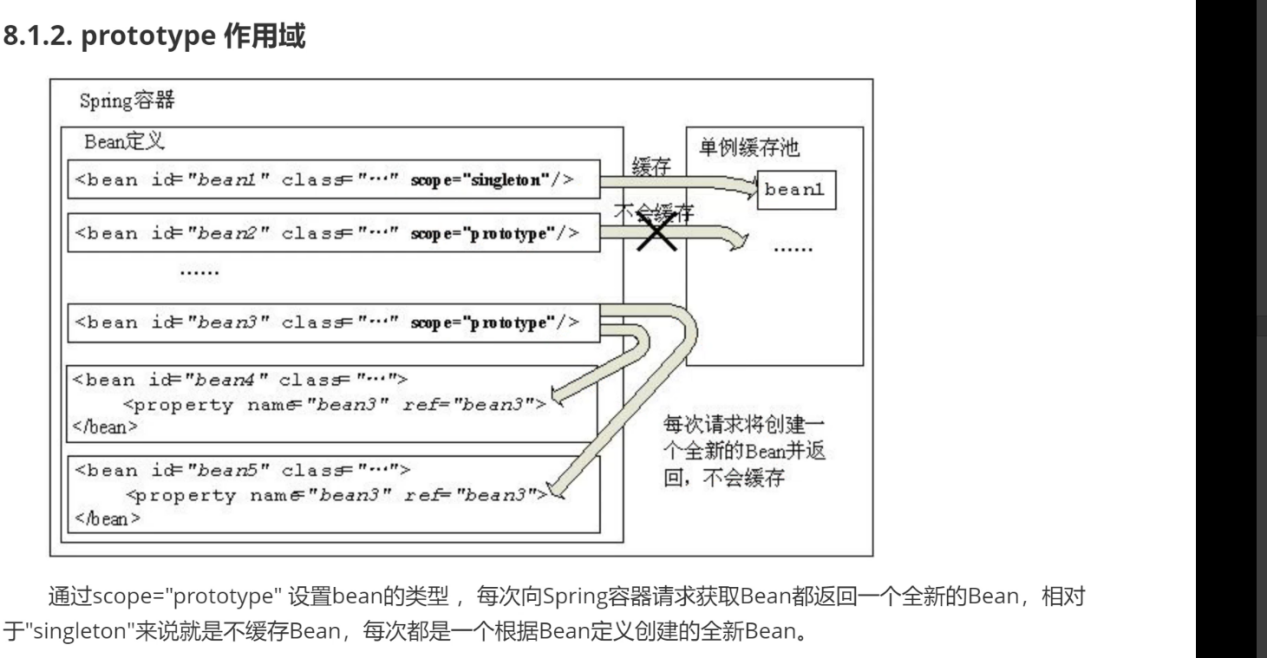
# 

# 适合实例化 bean（无状态）

Servlet user 对象 会有不同的用户 使用，每次实例化，单读赋值。 所以不能单例 不适合ioc bean





Bean scope 属性

# Bean 生命周期



### **Bean的生命周期**

对比已经学过的servlet 生命周期（容器启动装载并实例化servlet类，初始化servlet，调用service方法，销毁servlet）。

同样对于Spring容器管理的bean也存在生命周期的概念

在Spring中，Bean的生命周期包括Bean的定义、初始化、使用和销毁4个阶段

#### **Bean的定义**

在Spring中，通常是通过配置文档的方式来定义Bean的。

在一个配置文档中，可以定义多个Bean。

#### **Bean 的初始化**

默认在IOC容器加载时，实例化对象。

Spring bean 初始化有两种方式：

方式一：在配置文档中通过指定 init-method 属性来完成。

public class RoleService {  
    // 定义初始化时需要被调用的方法  
    public void init() {  
        System.out.println("RoleService init...");  
   }  
}

<!-- 通过init-method属性指定方法 -->  
<bean id="roleService" class="com.xxxx.service.RoleService" init-method="init"></bean>

方式二： 实现 org.springframework.beans.factory.InitializingBean 接口。

public class RoleService implements InitializingBean {  
​  
    @Override  
    public void afterPropertiesSet() throws Exception {  
        System.out.println("RoleService init...");  
   }  
}

<bean id="roleService" class="com.xxxx.service.RoleService" ></bean>

Bean对象实例化过程是在Spring容器初始化时被实例化的，但也不是不可改变的，可以通过 lazy-init="true" 属性延迟bean对象的初始化操作，此时再调用getBean 方法时才会进行bean的初始化操作

#### **Bean 的使用**

方式一：使用 BeanFactory

// 得到Spring的上下文环境  
BeanFactory factory = new ClassPathXmlApplicationContext("spring.xml");  
RoleService roleService = (RoleService) factory.getBean("roleService");

方式二：使用 ApplicationContext

// 得到Spring的上下文环境  
ApplicationContext ac = new ClassPathXmlApplicationContext("spring.xml");  
RoleService roleService = (RoleService) ac.getBean("roleService");

#### **Bean的销毁**

实现销毁方式(Spring容器会维护bean对象的管理，可以指定bean对象的销毁所要执行的方法)。

步骤一：实现销毁方式(Spring容器会维护bean对象的管理，可以指定bean对象的销毁所要执行的方法)

<bean id="roleService" class="com.xxxx.service.RoleService" destroy-method="destroy"></bean>

\*\*步骤二：\*\*通过 AbstractApplicationContext 对象，调用其close方法实现bean的销毁过程

AbstractApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");  
ctx.close();

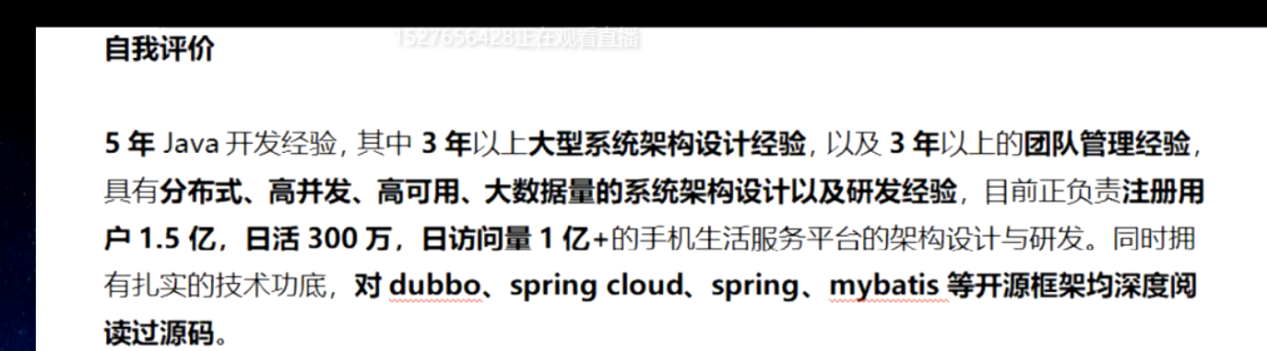
# 什么是控制反转？

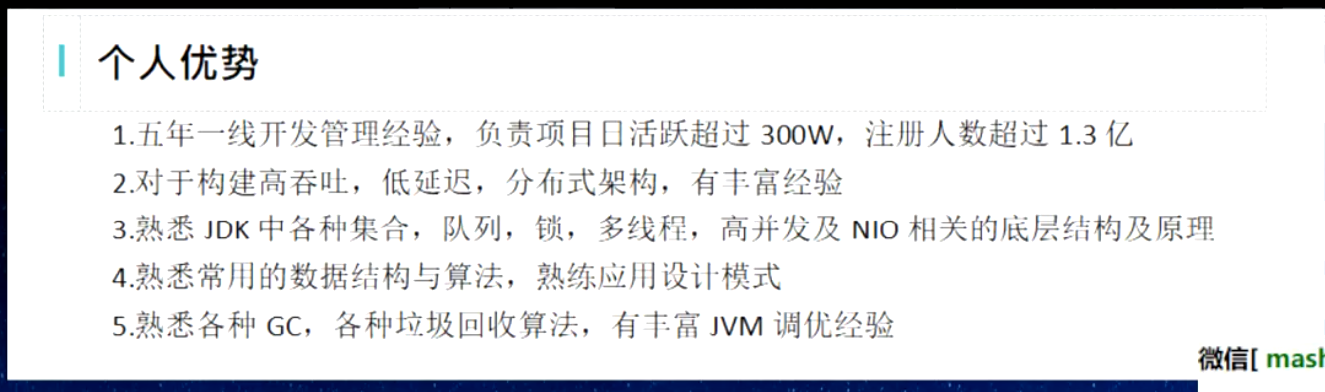
作用： 降低耦合度 modularity extensive

It means **giving the control of creating and instantiating the spring beans to the Spring IOC container** and the only work the developer does is configuring the beans in the spring xml file.

The term is related to, but different from, the [dependency inversion principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle" \o "Dependency inversion principle), which concerns itself with [decoupling dependencies](https://en.wikipedia.org/wiki/Dependency_(computer_science)" \o "Dependency (computer science)) between [high-level and low-level](https://en.wikipedia.org/wiki/High-_and_low-level" \o "High- and low-level) [layers](https://en.wikipedia.org/wiki/Abstraction_layer" \o "Abstraction layer) through shared [abstractions](https://en.wikipedia.org/wiki/Abstraction_(computing)" \o "Abstraction (computing)). The general concept is also related to [event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming" \o "Event-driven programming) in that it is often implemented using IoC so that the custom code is commonly only concerned with the handling of events, whereas the [event loop](https://en.wikipedia.org/wiki/Event_loop" \o "Event loop) and dispatch of events/messages is handled by the framework or the runtime environment.







真实写项目 删除记录都是假删

变为不可用

# Spring

数据源 数据库连接池

Jdbcmanager 事务管理 通过sql connection 里面的 进行管理