

**Spring, springCloud, springBoot和springMVC的区别**

spring mvc < spring < springboot

1、Spring

Spring是一个开源容器框架，可以接管web层，业务层，dao层，持久层的组件，并且可以配置各种bean,和维护bean与bean之间的关系。其核心就是控制反转(IOC),和面向切面(AOP),简单的说就是一个分层的轻量级开源框架。

2、SpringMVC

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。SpringMVC是一种web层mvc框架，用于替代servlet（处理|响应请求，获取表单参数，表单校验等。SpringMVC是一个MVC的开源框架，SpringMVC=struts2+spring，springMVC就相当于是Struts2加上Spring的整合。

3、SpringBoot

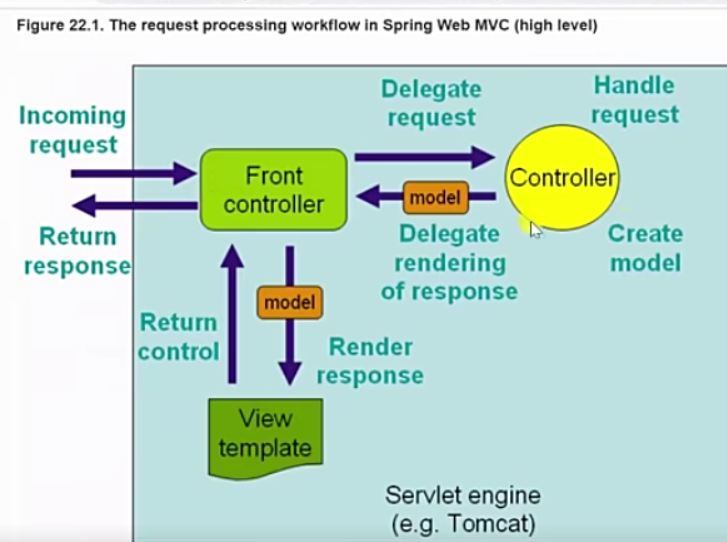
Springboot是一个微服务框架，延续了spring框架的核心思想IOC和AOP，简化了应用的开发和部署。Spring Boot是为了简化Spring应用的创建、运行、调试、部署等而出现的，使用它可以做到专注于Spring应用的开发，而无需过多关注XML的配置。提供了一堆依赖打包，并已经按照使用习惯解决了依赖问题--->习惯大于约定。

1. Spring cloud大部分的功能插件都是基于springBoot去实现的，springCloud关注于全局的微服务整合和管理，将多个springBoot单体微服务进行整合以及管理；  springCloud依赖于springBoot开发，而springBoot可以独立开发

**DispatcherServlet**

基于Servlet WebApplicationContext容器管理controller，viewResolver，handleMapping(映射路径的组件，把路径和controller相匹配)

Front controller是DispatcherServlet



**MVC框架**

MVC指model view controller=模型-视图-控制器，是一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，把业务逻辑聚集到一个组件里，在改进和个性化定制界面和用户交互的时候，不用重写业务逻辑。主要是分三层的一种架构，主要采用封装（分层）的思想来分割页面和数据库访问的框架，降低了两者的耦合度，同时通过Controller的控制，也使系统更加的灵活，使我们的代码更健壮，复用性更强

1.model是用于处理数据逻辑的部分。通常模型对象负责在数据库中存取数据。

2.view是应用程序中处理数据显示的部分。通常视图是依据数据模型创建的。

3.controller是应用程序中处理数据交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送。

2、mvc的**优点**

（1）多个视图共享一个模型，大大提高代码的可重用性。

（2）三个模块相互独立，改变其中一个不会影响其他两，所以依据这种设计模式能构建良好的松耦合性的组件。

（3）控制器提高了应用程序的灵活性和可控制性。控制器可以连接不同的模型和视图满足用户需求，(这样控制器可以为构造应用程序提高强有力的手段)

规避了JSP与Servlet各自的短板，Servlet只负责业务逻辑而不会通过out.append()动态生成HTML代码；JSP中也不会充斥着大量的业务代码。这大大提高了代码的可读性和可维护性。

3、mvc的**缺点**

（1）增加了系统结构和实现的复杂性。

对于简单页面，严格遵循mvc，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

（2）视图与控制器过于紧密的连接。

视图与控制器是相互分离，但确实联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。

（3）视图对模型数据的低效率访问。

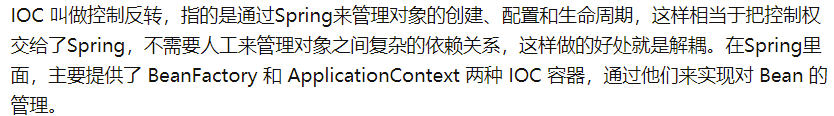
依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

**传统的Servlet**：用Java写的服务端程序，有独立于平台和协议的特性，可以响应任何类型的请求，但大多情况下只用来扩展基于HTTP协议的Web服务器，主要功能是交互式地浏览和生成数据，生成动态的Web内容

Servlet是一个接口或者是实现Servlet接口的类叫Servlet

**传统开发模式Servlet(不用IOC)的弊端**：分成3层架构：把视图控制，业务逻辑和数据库分离出来单独形成一个类—易于复用和维护。代码分成controller, service, DAO：一个DAO被多个Service对象使用，一个Service对象被多个Controller用---有逻辑复用，没有做到资源复用。多个controller创建多个重复的service，多个service创建多个重复的DOA，可以用单例模式，但每个类都是单例就会重复很多代码，而且变化的代价太大，一个service变了要改很多个controller。本质原因：组件的调用方参与了组件的创建和配置工作。所以用容器来创建和配置，比如Tomcat是Servlet的容器，它创建和配置好了Servlet. 我们用IOC容器来管理组件

**IOC(控制反转)**：对象的创建和配置的控制权从调用方移给容器。把对象给IOC管理，调用方不用再负责组件的创建，要用组件时候直接获取Bean就行---调用方只要声明依赖项，所需要的Bean就自动配置完了，就像在调用方外部注入一个依赖项给她使用—这种方式是**依赖注入**

****

**OOP的局限性：**有重复代码的时候就封装出来复用，但只能复用核心业务逻辑，而不能复用辅助逻辑，比如日志记录，性能统计，事务管理, OOP是自上而下的，但辅助逻辑是横向贯穿所有方法的。这些辅助逻辑是切面，可以用代理模式，但切面逻辑的难点不是不修改原有业务，而是对所有业务生效。要增强一个业务类就要新建一个代理类，切面多了就会导致代理类嵌套。AOP可以解决，他是OOP的补充，让我们能够不修改原有代码，就能让切面逻辑在所有业务逻辑里生效。



**IOC和AOP好处**：

IOC解决：1.创建重复对象，造成资源浪费

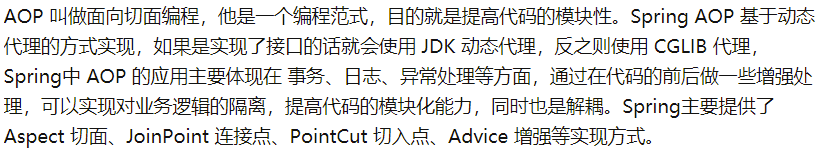
2.更换实现类要改很多地方

3.创建和配置组件工作繁杂，而且给组件的调用方会带来很多不便

AOP解决：切面逻辑编写繁琐，有多少个业务方法就要编几次

**Spring AOP(面向切面编程) -- 提高代码模块性**

利用一种横切技术，剖解开封装的对象内部，然后把那些影响多个类的公共行为封装到一个可重用模块，也就是切面。也就是把那些与业务无关，但是被业务模块共用的逻辑或责任封装起来来减少系统的重复代码，降低模块间的耦合度，增加之后的可操作性和可维护性。



切点(Pointcut):指定一个通知将被引发的一系列连接点的集合。AOP框架让开发者能指定切入点：比如用正则表达式

横切关注点的使用场景：权限，缓存，懒加载，调式，同步

AOP代表的是一个横向的关系，如果说“对象”是一个空心的圆柱体，其中封装的是对象的属性和行为；那么面向方面编程的方法，就仿佛一把利刃，将这些空心圆柱体剖开，以获得其内部的消息。而剖开的切面，也就是所谓的“方面”了。然后它又以巧夺天功的妙手将这些剖开的切面复原，不留痕迹。

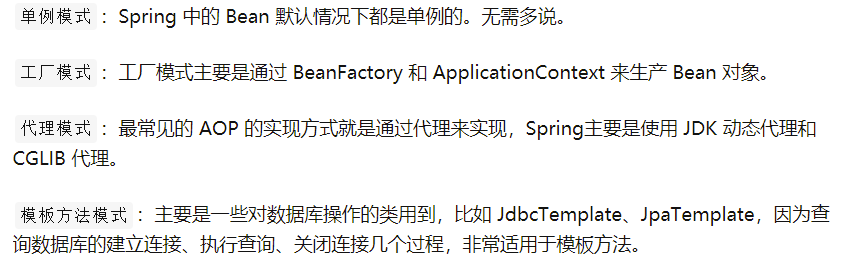
AOP把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。

横切关注点的一个特点是，他们经常发生在核心关注点的多个地方，而各处都基本相似。比如权限认证、日志、事务处理。Aop 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。正如Avanade公司的高级方案构架师Adam Magee所说，AOP的核心思想就是“将应用程序中的商业逻辑同对其提供支持的通用服务进行分离。”

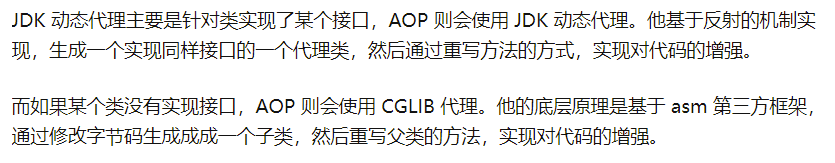
框架编写流程：1.导包 2.写配置 3.测试

**面试题**

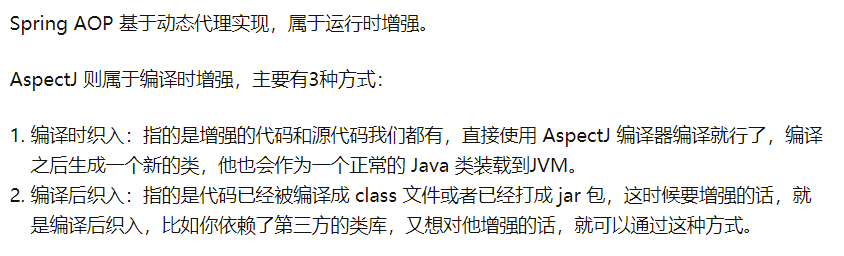
1. **设计模式**

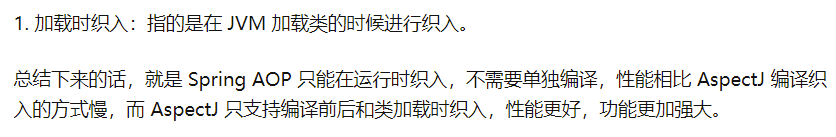
****

1. **JDK动态代理和CGLIB代理**

****

1. **Spring AOP和AspectJ AOP**

****

****

1. **BeanFactory和FactoryBean**

区别：BeanFactory是个Factory，也就是IOC容器或对象工厂，是ApplicationContext的父类。

FactoryBean是个Bean。所有的Bean都是由BeanFactory(也就是IOC容器)来进行管理的。但FactoryBean不是简单的Bean，而是一个能生产或者修饰对象生成的工厂Bean, 它的实现和设计模式里工厂模式和修饰器模式类似

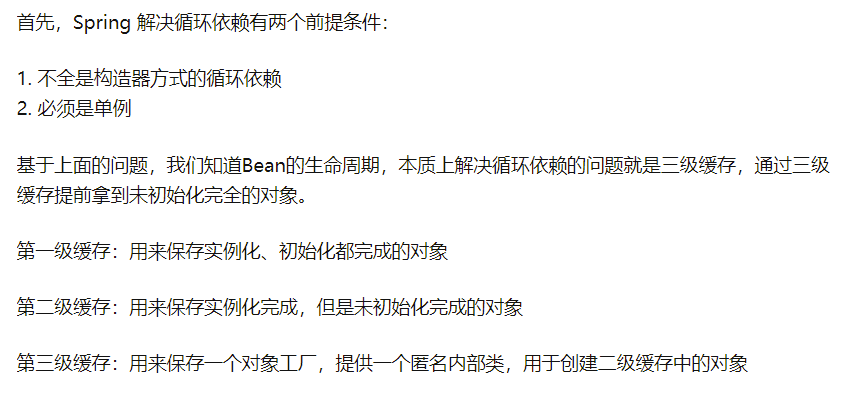
bean：1.bean是对象，一个或者多个不限定2.bean由IoC管理 3.应用程序由一个个bean构成

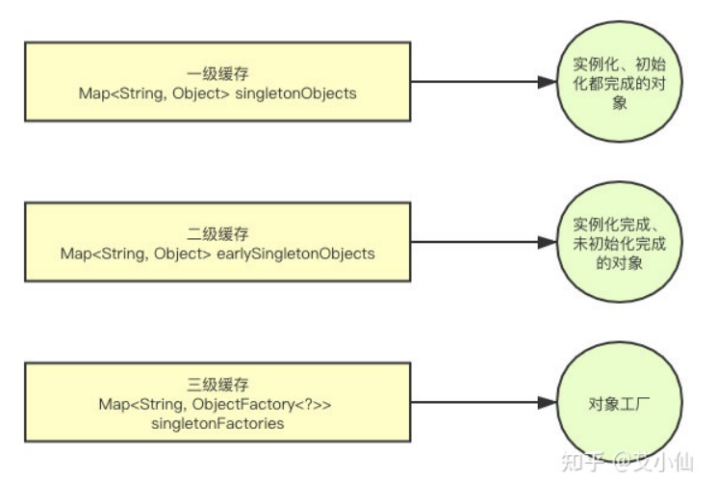


1. **Bean的生命周期**

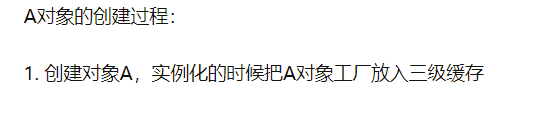


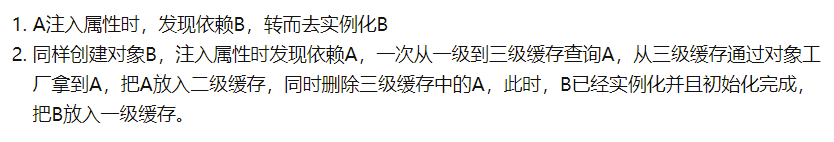
1. **Spring解决循环依赖**

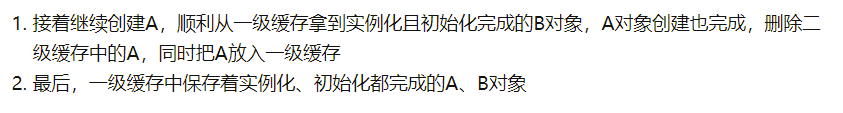


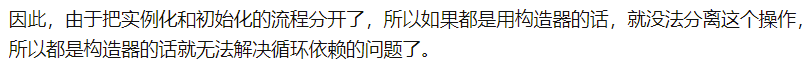


具体过程例子：

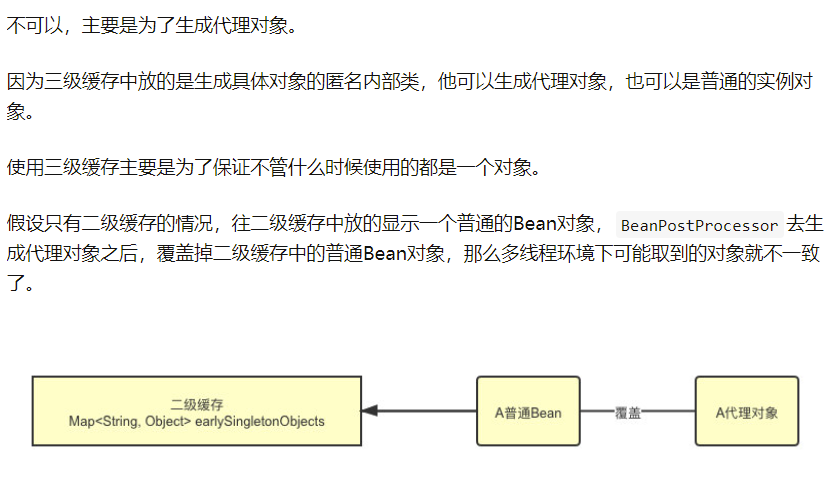




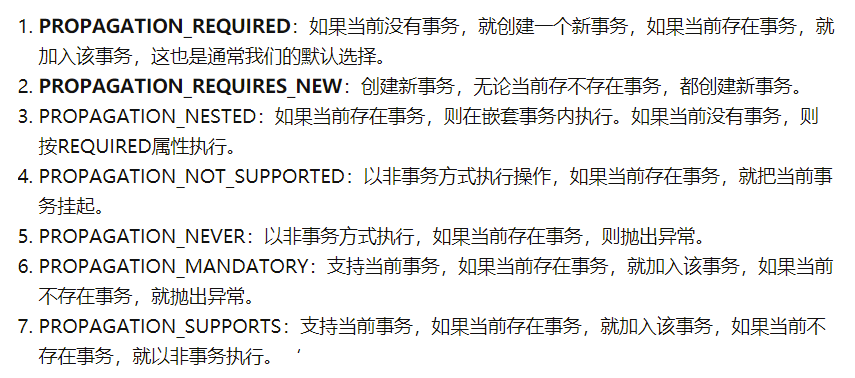




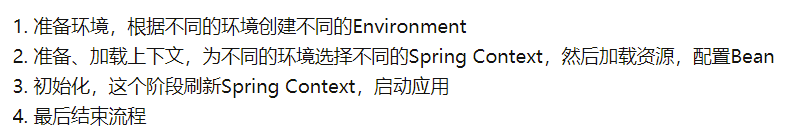
1. **为什么要三级缓存，二级不行吗**

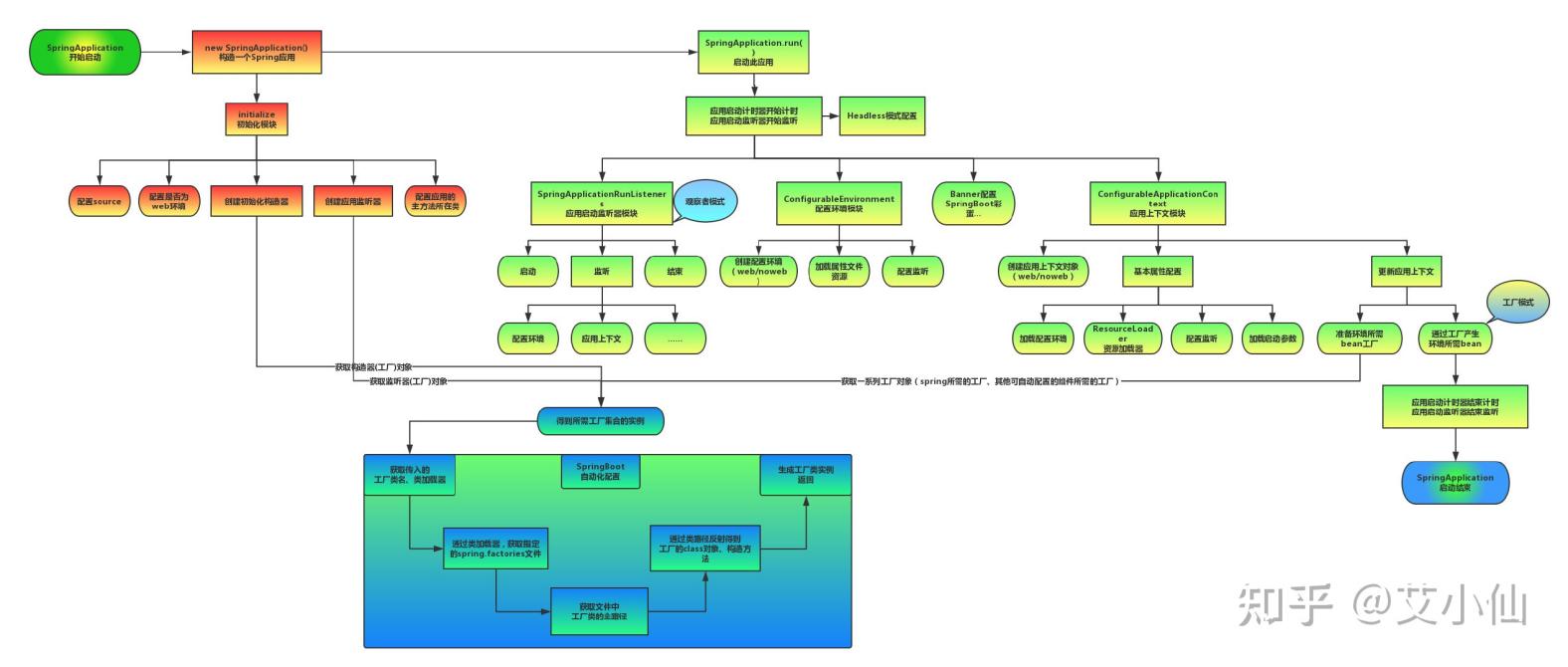


1. **事务传播机制**



1. **SpringBoot启动流程**





MyBatis--基于java的持久层框架

核心组件：1.sqlSessionFactory：创建sqlSession的工厂类

2.sqlSession：核心组件，用于向数据库执行SQL

3.主配置文件：XML配置文件，对底层进行配置

4.Mapper接口：就是DAO接口

5.Mapper映射器：用于写SQL +将SQL和实体类映射的组件 用XML, 注解都可

前三个不用写

**Mybatis**: 支持定制化 SQL、存储过程和对象关系映射(ORM—就是把数据库表和实体类以及实体类的属性对应起来，让我们可以操作实体类就实现操作数据库表), 内部封装了jdbc(java数据库连接)细节，让开发者只要关注sql语句本身，不用花精力去处理加载驱动、创建连接、创建statement等繁杂的过程，主要设计目的就是让我们对执行SQL语句时对输入输出的数据管理更加方便，能方便地写出SQL和方便地获取SQL的执行结果。

**具体过程**：mybatis通过xml或注解的方式将要执行的各种statement配置起来，并通过java对象和statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回

**分三层**：  
1、API接口层：提供给外部使用的接口API，开发人员通过这些本地API来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。  
2、数据处理层：负责具体的SQL查找、SQL解析、SQL执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。  
3、基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑

自定义mybastis：

Mybatis用代理dao的方式实现增删改查：1.创建代理对象，在代理对象中调用selectList

步骤：1.读配置文件2.创建sqlsessionFactory工厂(mybatis用了创建者模式，传in就行，创建工厂细节被隐藏)3.使用工厂生产sqlsession对象(用了工厂模式—解耦，降低类之间的依赖关系，因为让工厂去生产对象) 4.使用sqlsession创建DAO接口的代理对象(用了代理模式—可以不修改源码的基础上对已有方法增强) 5.使用代理对象执行方法

用1和5就行，但有234更灵活

SelectList方法执行—要提供连接信息+映射信息(1.SQL语句2.封装结果的实体类全限定类名—封装成一个对象):

1.根据配置文件创建connection对象 2.获取预处理对象(要SQL语句) 3.执行查询 4.遍历结果集用于封装，把每个结果加到一个list里(因为实体类的属性和表的列名一样，所以可以把列名看成是实体类的属性名称，就可以用反射的方式来根据名称获取每个属性，并把值赋进去) 5.返回list