Spring是什么？

Spring就是要简化开发

轻量级

零配置编程，API使用简单

面向Bean

只需要编写非常普通的Bean

松耦合

充分利用AOP思想

万能胶

与主流框架无缝集成

设计模式

将Java中经典的设计模式运行的彻底

Spring采用了四个关键策略

1.基于POJO的轻量级和最小侵入性编程；

侵入性：代码的嵌套的使用；

独立开发合并运行减少侵入式；

2.通过依赖注入和面向接口松耦合；

依赖注入：以成员变量的形式把这个类的公共的代码引入像Spring的注入的方式有get，set方式， 构造方法注入；

3.基于切面和惯性进行声明式编程；

切面：AOP，使用的目的也是为了解耦，在Spring出现之前AOP只是一种思想，出现之后，使它变成一种现实，真正实现了代码与代码不相互嵌套，侵入，最终合在一起运行；典型的事务的管理；

4.通过切面和模板减少样版式代码；

像Spring jdbc, Spring mvc里面使用很多模板的概念，解决数据处理的逻辑；

像这些都是为了简化开发的；

2.2.面向Bean

Spring 是面向Bean的编程BOP，Bean在Spring中才是真正的主角，spring习惯叫做BOP编程

Spring里面有一个核心的IOC容器，主要是用来存储Bean的，保证Bean之间的依赖关系，Spring

提供IOC容器通过配置文件或者注解的方式来管理对象之间的依赖关系；

控制反转（其中最常见的方式叫做依赖注入 Dependency Injection DI）,还有一种方式叫“依赖查找”

Dependency Lookup DL，它在C++，Java，PHP以及.NET中运用。在最早的Spring是包含有依赖注入方法和依赖查询的，但是因为依赖查询使用频率过低，不久就被Spring移除了，所以在Spring中

控制反转也被称作依赖注入，它的基本概念是：不创建对象，但是描述创建它们的方式，在代码中不直接与对象和服务连接，但在配置文件中描述哪一个组件需要哪一项服务。容器（在Spring框架中是IOC容器） 负责将这些联系在一起。

【@Autowite A a //Spring 初始化，实例化，现在没有new()也实例化了，也就是说这个控制权交给spring了，控制反转，之前new()来对象由一个变量来保存，spring创建的对象是有Spring容器来保存IOC容器，用来存储bean对象；web容器是用来装Servlet的

IOC最终的目的是实现依赖注入】

在典型的IOC场景中容器创建了所有对象，并设置必要的属性将它们连接在一起，决定什么时间调用方法

2.3依赖注入

Spring 设计的核心 org.springframework.beans包（架构核心是org.springframework.core包），它设计目的是与JavaBean组件一起使用。这个包通常不是由用户直接使用，而是由服务器将其用作其他多数功能的底层中介，下一个最高级抽象是BeanFactory接口，它是工厂设计模式的实现，允许通过名称创建和检索对象。BeanFactory 也可以管理对象之间的关系。

BeanFacory支持两种对象模型，其一：单例：模型提供了具有特定名称的对象共享实例，可以在查询时对其进行检索。Signgleton是默认的也是最常见的对象模型。对于无状态服务对象很理想。其二：原型: 模型确保每次检索都会创建单独的对象，在每个用户都需要自己的对象时，原型模型最适合。

bean工厂的概念是Spring作为IOC容器的基础。IOC则将处理事情的责任从应用程序代码转移到框架。

//实现依赖注入

@Autowired Interface A a ; //自动把它的实现类注入进来

@Resource("aaa") A b ;// IOC容器中类的id为aaa对象自动注入到这里（可区分父子类）

@Autowired A a; //根据类型自动注入

依赖链所以的对象，ioc容器里面初始化：实例化是按照先后顺序进行的；

Spring的注入方式

1.setter

2.构造方式

3.强制赋值 @Autowired private Interface A a ; //也可以自动把它的实现类注入进来

2.4面向切面

面向切面编程，即AOP，是一种编程思想，它允许程序员对横切关注点或横切典型的职责分界线的行为（例如日志和事务管理）进行模块化。AOP的核心构造是方便将哪些影响多个类的行为封装到可重用的模块中。

Aop和IOC是补充性的技术，它们都运用模块化方式解决企业应用程序开发中的复杂问题。在典型

的面向对象开发方式中，可能要将日志记录语句放在所有方法和Java类中才能实现日志功能。在AOP方式中，可以反过来将日志服务模块化，并以声明的方式将它们应用到需要日志的组件上，当然，优势就是Java类不需要知道日志服务的存在，也不需要考虑相关的代码，所以，用Spring AOP编写的用用程序代码是松耦合的；

AOP一定要达到无缝合并，否则就不叫AOP，AOP核心切面即规则目的解耦；

AOP核心思想： 实现AOP编程，先把一个整体拆开，分别开发，等到发布的时候，再组装到一起运行；拆开是要按照一定的规则，这个规则就是切面，一个切面就是一个规则，面向切面编程就是要怎么去定义这个规则；

总结AOP思想：解耦

AOP的功能完全集成到了Spring事务管理、日志和其他各种特性的上下文中。

事务： 规则，先开启一个事务，执行提交事务，事务回滚，关闭事务，这种有规律的东西，就可以

认为它就是一个固定的规则，这时候，我们就单独把这个具有一定规律的规则，单独分离出来，作为一个独立的模块；

Authentication 权限认证

Logging 日志

Lazy Loading 懒加载

Context Process 上下文处理

Error Handler 错误跟踪（异常捕获机制）

Cache 缓存

2.5常用的设计模式

代理模式：1、事情必须做，而自己有没有时间做或者不想做；2、持有被代理对象的引用。

静态代理、动态代理

JDK动态代理：

被代理对象一定要实现接口是因为在进行字节码重组的时候，对象进行转换，要拿到这个方法进行转换所以要实现接口，不实现这个接口，就没法知道这个对象是干什么的

//原理：

//1.拿到被代理对象的引用，然后获取它的接口

//2.JDK代理重新生成一个类，同时实现我们给的代理对象所实现的接口

//3.把被代理对象的引用也拿到了

//4.重新动态生成一个class字节码

//5.然后编译

CGLib动态代理：

cglib.jar(全称Code Generation Library 代码生成库)

asm.jar(全称assembly,装配)

特点：1、执行者、被代理人 2.对于被代理人来说，这件事情是一定要做的，但是自己不想做，找代理

cglib实现方式它不是通过接口去获取而是通过自动生成一个类，来继承被代理的类，它子类的引用指向父类；

JDK的动态代理是通过接口来进行强制转换的；

生成以后的代理，可以强制转换为接口

CGLib的动态代理是通过自动生成一个类，继承被代理对象的类，这样就可以强制转换为被代理对象

（也就是自己的那个类） 子类的引用赋值给父类

3：需要获取到被代理的人个人资料。

满足代理模式应用场景三个必要要求,穷取法

1.两个角色，执行者、被代理对象

2.注重过程，必须要做，被代理对象不想做

3.执行者必须拿到被代理对象的个人资料（执行者持有被代理对象的引用）

总结 ：代理模式归根到最底层是做了 字节码重组

可以在每一个方法调用之前加一些代码，在方法调用之后再加一些代码

AOP:事务代理（声明式事务，哪个方法需要加事务），日志监听

service方法

开启一个事务（open）

事务的执行是由我们自己的代码完成的

监听到是否有异常，可能需要根据异常的类型来决定这个事务是否要回滚还是继续提交（commit/rollback）

事务要关闭(close)

代理模式关心的过程；

工厂模式

简单工厂，共产方法、抽象工厂

特点：

隐藏复杂的逻辑处理过程，只关心结果

Spring中的工厂模式主要是通过BeanFactory用来生产各种Bean，

如：单例的Bean，被代理过的Bean，最原始的Bean（原型），List类型的Bean，作用域不同的Bean

最终达到的效果是解耦

工厂模式关心的是结果

单例模式：

常用单例模式写法

穷举法： 配置文件：如果不是单例（针对于某一种功能的配置，两个配置文件中内容是一样，则有一个是浪费的，如果是不一样的，我们就不知道以哪一个为准了）

直接上级领导：对于你来说，如果有多个领导，你到底听谁的。

是什么？为什么要有单例模式？ 单例具体怎么实现？

特点：

1.保证从系统启动到系统停止，全过程只会产生一个实例。是什么

2.当我们再应用中遇到功能性冲突的时候，需要使用单例模式。为什么

单例模式的七种写法：

单例模式

一个类模板，在整个系统运行过程中，只允许产生一个实例（有且只有一个）new

解决一个并发访问的时候线程安全问题保证单例的技术方案有很多种：

饿汉式（一般不会存在线程安全的问题）、懒汉式（经常存在线程安全的问题）、注册登记式、枚举式、序列化和反序列化保证单例

饿汉式

在实例使用之前，不管用不用，都先new出来，避免了线程安全问题；

懒汉式

默认加载的时候不需要实例化,在需要用到这个实例时候才实例化;

延时加载

注册登记式

每使用一次,都往一个固定的容器中去注册并且将使用过的对象进行缓存,下次

去取对象的时候,就直接从缓存中取值,以保证每次获取的都是同一个对象;

枚举式就是注册登记式的一种；

IOC中的单例模式,就是典型的注册登记式单例

序列化和反序列化

保证单例：重写readResolve();

public static final LazyFour getInstance(){

/\*方法中的逻辑，是要在用户调用的时候才开始执行的

方法中实现逻辑需要分配内存也是在调用时才分配的，分配的内存空间不会被直接的分配\*/

return LazyHolder.INSTANCE;

}

static int a = 1;

/\*不管该class又没有实例化，static静态块总会在classLoader执行完后，就加载完毕\*/

static {

//因为静态块中的内容，只能访问静态属性和静态方法

//只要是静态方法或者属性，直接用Class的名字就能点出来

LazyFour.a = 2;

//JVM 内存中的静态去，这一块的内容是公共的

}

/\*\*

\* 我们所写的所有代码，在java的反射机制面前，都是

\* 裸奔的，反射机制是可以拿到private 修饰的内容的，我们

\* 可以理解成即使加上private也不靠谱（按正常套路出牌，貌似可以）

\*/

//类装在到JVM过程

//从上往下（必须先声明后使用）

//先属性，后方法

//先静态，后动态

委派模式：1、类似于中介的功能（委托机制）；2、只有被委托人的引用。

两个角色 受托人，委托人（社会上是平等关系）

公司里面：项目经理，普通员工

项目经理：主要职责是安排任务

普通员工：执行任务

跟代理模式相比委派模式关心的是结果；

特点：

1、类似于中介的功能（委托机制）；

2、持有被委托人的引用；

3、不关心过程，只关心结果。

为什么要委派模式：主要目的就是隐藏具体的实现的逻辑；

Spring中，IOC容器中，有一个Register的东西（为了告诉容器，在类被初始化的过程中，需要做很多不同的逻辑处理，需要实现多个任务执行者，分别实现各自的能供）

工厂模式是保证结果的多样性，对于用户来说只有一种方法；

策略模式：1、过程不同，但结果一样。

举例： spring JDBC RowMapper

JDBC 驱动标准

原型模式：1、过程相同，但结果不一样，或者叫模块模式（提高开发效率）

原型模式

DTO、VO、POJO、Entity

DTO和VO之间存在一些属性名称、类型都相同，数据库表查询出来的对象会赋值给DTO

DTO通常不直接传值给MVC中的Model；

把DTO的值赋值给VO

再把VO中的值传输到View中去；

复制过程：把每一个DTO中每一个属性的值赋值给VO中的每一个属性的值，属性名称相同，属性类型相同

apache会提供BeanUtil复制功能利用反射去实现的（原型模式）

反射的性能比较低一般会利用Clone克隆

spring 有一个scope="prototype"原型模式，把对象中配置的依赖的关系，在每次使用对象之前都会创建一个新的对象，并且会将依赖关系完整的赋值给这个新创建的对象；

spring默认是单例模式；

举例：克隆

//当这里有上百个属性时赋值是该怎么处理？

//用循环，用反射，确实可以，但是反射性能并不高

//字节码复制，用原型模式来解决

//能够直接拷贝其实际内容的数据类型/只支持9种，八大基本数据类型+String 这种是浅拷贝意思是像

//List这样的类型的数据是克隆对象种的list和原对象的list是同一个对象

//深度克隆

public Object deepClone(){

ByteArrayOutputStream bos = null;

ObjectOutputStream oos =null;

ByteArrayInputStream bis = null;

ObjectInputStream ois = null;

try{

//序列化

bos = new ByteArrayOutputStream();

oos = new ObjectOutputStream(bos);

oos.writeObject(this);

//反序列化

bis = new ByteArrayInputStream(bos.toByteArray());

ois = new ObjectInputStream(bis);

TheGreatestSage copy = (TheGreatestSage) ois.readObject();

copy.birthday = new Date();

return copy;

}catch (Exception e){

e.printStackTrace();

return null;

} finally {

if(null != bos){

try {

bos.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if(null != bis){

try {

bis.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if(null != ois){

try {

ois.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if(null != oos){

try {

oos.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

ORM框架用原型模式比较多：如： Fast JSON（apache CommonUtil包中） 将j记录行转换为java对象；

非常复杂的Java之间的数据备份，get，set，用反射是比较麻烦的；

只要实现了序列化，就可以进行数据的拷贝，它是不走构造方法直接字节流操作

模板模式：

执行流程一样，但中间有些步骤不同。

举例：

喝茶过程实际上是比较麻烦的，机器产生，饮料冲泡机流程：

1.烧开水

2.准备一个杯子，（放啥？）

3.用水冲泡

4.添加铺料（放啥？）

固定执行流程，就是一个模板

springJDBC就是一个模板模式，JDBC也是有一定的流程的，JDBC是java中的一种规范，规范制定好了，各个数据库厂商自己去实现

加载驱动类 DriverManager

建立连接

创建语句集（标准语句集，预处理语句集）（语句集不确定，Mysql,Oracle,SQL Server,Access）

执行语句集

结果集 ResultSet

ORM（是映射为一个Map,还是List，或者是我们自己定义的类，，，）框架对结果集处理

Spring 思想 应用场景（特点） 一句话归纳

AOP 1. Aspect Oriented Programming（面向切面编程） 2. 找出多个类中有一定规律的代码，开发时拆开，运行时再合并。 3.面向切面编程，即面向规则编程。 解耦，专人做专事

OOP 1. Object Oriented Programming（面向对象编程） 2.归纳总结生活中一切事物。 封装、继承、多态

BOP 1. Bean Oriented Programming （面向Bean编程） 2. 面向 Bean （普通的java类）设计程序 一切从Bean开始

IOC 1. Inversion of Control （控制反转） 2.将 new 对象的动作交给 Spring 管理，并由 Spring 保存已创建的对象（IOC容器） 转交控制权（控制反转）

DI/DL 1. Dependency Injection（依赖注入） / Dependency Loopup（依赖查找） 2. 依赖注入、依赖查找，Spring 不仅保存自己创建的对象，而且保存对象与对象之间的关系。 3. 注入即赋值， 主要三种方式构造方法、set方法、直接赋值。 先理清关系再赋

设计模式 应用场景（特点） 一句话归纳

代理模式 Proxy 1.两个参与角色： 执行者、被代理者人。 2.对于被代理人来说，这件事是一定要做的，但是我自己又不想做或者没有时间做，找代理。 3. 代理人需要获取到被代理的个人资料(持有被代理人的引用) 办事要求人，所以找代理

工厂模式 Factory 1. 对调用者来说隐藏了复杂的逻辑处理过程， 调用者只关心执行结果。 2.对工厂来说要对结果负责，保证生产出符合规范的产品。 只对结果负责，不要三无产品。

单例模式 Singleton 1. 保证从系统启动到系统终止， 全过程只会产生一个实例。 2.当我们在应用中遇到功能性冲突时，需要使用单例模式 保证独一无二

委派模式 Delegate 1.两个参与角色，委托人和被委托人。 2.委托人和被委托人在权利上完全平等（即实现同一个接口） 3. 委托人持有被委托人的引用。 4. 不关心过程， 只关心结果。 干活是你的(普通员工) ，功劳是我的（项目经理）

策略模式 Strategy 1. 最终执行最终结果固定的。 2. 执行过程和执行逻辑不一样 我行我素，达到目的就行

原型模式 Prototype 1.首先有一个原型。 2. 数据内容相同，但对象实例不同(完全两个个体) 拔一根猴毛，吹出千万个。

模板模式 Template 1. 执行流程固定，但中间有些步骤有细微差别（运行时才确定） 2. 可实现批量生产 流程标准化，原料自己加

三、回首Spring架构设计

3.1系统架构

Spring总共大约由20个模块，由1300多个不同的文件构成。而这些组件被分别整合在核心容器（Core Container）、AOP（Aspect Oriented Programming）和设备支持（Instrmentation）、数据访问及集成（Data Access/Integration）、Web、报文发送（Messaging）、Test，6个模块集合中。

以下是SPring 4 的系统架构图。

Jdbc Template 模板模式

ORM rowMapper策略模式

Transaction 事务管理 代理模式

Websocket 协议， ws

WEB mvc

portlet 页面 jsp，

AOP config

Beans BOP

spring核心实现逻辑 Core

前后端分离，动静态分离+Nginx把静态资源和动态资源完全分离，后端只输出数据，HTML负责渲染；

组成Spring框架的每个模块集合或者模块都可以单独存在，也可以一个或者多个模块联合实现。每个模块的组成和工程如下：

1.核心容器：由spring-beans、spring-core、spring-context和spring-expression（Spring Expression Language ，SpEL）四个模块组成。

spring-beans和spring-core模块是Spring框架的核心模块，包括了控制反转(IOC，Inversion of Control)和依赖注入（Dependency Injection,DI），

BeanFactory接口是Spring框架中的核心接口，它是工厂模式的具体的体现。BeanFactory使用控制反转对应用程序的配置和依赖性规范与实际的应用程序代码进行了分离。

但BeanFactory 容器实例化后并不会自动实例化Bean，只有当Bean被使用时，BeanFactory 容器才会对该Bean进行实例化与依赖关系的装配；

Spring-context 模块构架于核心模块之上，它扩展了BeanFacotry，为它添加了Bean生命周期控制、框架事件体系以及资源加载透明化等功能。此外该模块还提供了许多业务级支持，如邮件访问、远程访问、任务调度等，ApplicationContext时该模块的核心接口，它是BeanFactory的超类，与BeanFactory不同，ApplicationContext容器实例化后会自动对所有的单实例Bean进行实例化与依赖关系的装配，使之处于待用状态；

spring-expression 模块是统一表达式语言（unified EL）的扩展模块，可以查询、管理运行中的对象，同时也方便的可以调用对象方法、操作数组、集合等。它的语言类似传统EL，但提供了额外的功能，最出色的要数函数调用和简单字符串的模板函数。这种语言的特性是基于Spring产品的需求而设计，它可以非常方便地同Spring IOC进行交互。

2.AOP和设备支持：由spring-aop、spring-aspects和spring-instrumentation 3个模块组成。

spring-aop 是Spring的另一个核心模块，是AOP主要的实现模块。作为继OOP后，对程序员影响最大

的编程思想之一，AOP极大地开拓了人们对于编程的思路。在Spring中，它是以JVM的动态代理技术为

基础，然后设计出了一系列的AOP模切实现，比如前置通知，返回通知，异常通知，同时，Pointcut

接口来匹配切入点，可以使用现有的切入点来设计横切面，也可以扩展相关方法根据需求进行切入。

spring-aspects 模块集成自AspectJ框架，主要是为Spring Aop提供多种AOP实现方法。

###

spring-instrumentation模块是基于JAVA SE中的“jaav.lang.instrument”进行设计的，应该算是

AOP的一个支援模块，主要作用是在JVM启用时，生成一个代理类，程序员通过代理类在运行时修改类的字节，从而改变一个类的功能，实现AOP的功能，在分类里，我把它分在了AOP模块下；

3.数据访问及集成：由spring-jdbc、spring-tx、spring-orm（对象关系映射）、spring-jms和spring-oxm5个模块组成

spring-jdbc模块是Spring提供的JDBC抽象框架的主要实现模块，用于简化Spring JDBC。主要是提供

JDBC模块方式、关系数据库对象化方式、SimpleJdbc方式、事务管理来简化JDBC编程，主要实现类是

JdbcTemplate、SimpleJdbcTemplate以及NameParameterJdbcTemplate.

spring-tx模块是Spring JDBC事务控制实现模块，使用Spring框架，它对事务做了很好的封装，通过它的AOP配置，可以灵活的配置在任何一层：但是在很多的需求和应用，直接使用JDBC事务控制还是有其优势的。其实，事务是以业务逻辑为基础的：一个完整的业务应该对应业务层里的一个方法；如果业务操作失败，则整个事务回滚；所以，事务控制是绝对应该放在业务层的；但是持久层的设计则应该

遵循一个很重要的原则：保证操作的原子性，即持久层里的每个方法都应该是不可以分割的。所以，在使用Spring JDBC事务控制时，应该注意其特殊性。

spring-orm模块是ORM框架支持的模块，主要集成Hibernate,Java Persistence API(JPA)和 Java

Data Objects(JDO)用于资源管理、数据访问对象（DAO）的实现和事务策略。

spring-jms模块（Java Messaging Service）能够发送和接受消息，自Spring Framework 4.1以后，它还提供了对spring-messaging模块的支持；

spring-oxm模块主要提供了一个抽象层以支持OXM（OXM是Object-to-XML-Mapping的缩写，它是一个O/M-mapper，将java对象映射成XML数据，或者将XML数据映射成JAVA对象） ,例如：JAXB,Castor,XMLBeans,JiBX和XStream等。

4.Web:由spring-web、spring-webmvc、spring-websocket和spring-webmvc-portlet 4个模块组成。

spring-web模块为Spring提供了最基础web支持，主要建立于核心容器之上，通过Servlet或者Listeners来初始化Ioc容器，也包含一些于WEB相关的支持；

spring-webmvc模块众所周知是一个web-Servlet模块，实现了Spring MVC(model-view-controller)的Web应用。

spring-websocket模块主要是与Web前端的全双工通讯的协议；

spring-webmvc-portlet模块是知名的Web-Portlets模块(Portlets在Web门户上管理和显示的可插拨的用户界面组件，Portlet产生可以聚合到门户页面中标记语言代码的片段 如 HTML XML等)

5.报文发送：即spring-messaging模块；

spring-messaging是Spring 4新加入的一个模块，主要职责是为Spring 框架集成一些基础的报文传送应用；

6.Test：即Spring-test模块

spring-test 一般和Junit 混合使用，模块主要为测试提供支持的，毕竟在不需要发布（程序）到你的应用服务器或者连接到其他企业设施的情况下能够执行一些集成测试或者其他测试对于任何企业都是非常重要的。

3.2依赖关系

该图是Sping 3.2.x的包结构，可以从中清楚看出Spring 各个模块之间的依赖关系

从图中可以看出，IOC的实现包spring-beans和 AOP 的实现包 spring-aop 是整个框架的基础，而 spring-core 则是整个框架的核心，基础的功能都在这三个包中。

在此基础之上，spring-context 提供上下文环境，为各个模块提供粘合作用。

在spring-context 基础之上提供了spring-tx和spring-orm包，而web部分的功能，都是要依赖spring-web来实现的。

由于struts框架自身的Bug一直都没有修复，以及SpringMVC 已经足够强大，所以在最新的spring4中已经移除了 spring-struts 模块。

如果你想加入spring源码的学习，建议从spring-core入手，其次是spring-beans和spring-aop，随后是spring-context，再其次是spring-tx和spring-orm，最后是spring-web和其他部分