**目前很多公司的架构，从Struts2迁移到了SpringMVC。你有想过为什么不使用Servlet+JSP来构建Java web项目，而是采用SpringMVC呢？**

**既然这样，我们从源头说起。Struts2的源头其实也是Servlet。Servlet的作用是接收浏览器传给服务端的请求（request），并将服务端处理完的响应（response）返回给用户的浏览器，浏览器和服务端之间通过http协议进行沟通，其过程是浏览器根据用户的选择将相关信息按http协议报文的规范组装请求的http报文，报文通过网络传输到指定的服务器，服务器通过特定的web容器接收这个报文信息，例如：tomcat，jetty，jboss这样的web容器，web容器会将http报文解析出来，如果是用户请求，最终解析出来的报文信息会用一个request对象存储起来，服务端使用这个request做完相应的处理后，服务端程序将结果信息封装到response对象里，然后将response对象交给web容器，web容器则把这个response对象转变为http协议的报文，并将报文回传给浏览器，浏览器最后解析这个响应报文，将最终结果展示给用户。**

**Web容器创造了servlet接口，servlet接口就是开发人员自己实现业务逻辑的地方，程序员开发servlet就好比做填空题，而填空题的语境或者说上下文提示就是由request和response对象，但是javaEE规范里的servlet接口很简单，就三个方法init，service和destory，但是这个接口太笼统，所以规范里还提供了一个HttpServlet类，这个类根据http请求类型提供了doGet，doPost等方法，servlet接口最大的特点就是根据http协议的特点进行定义，因此做servlet开发时候如果使用者对http协议特点不是特别熟悉，都会碰到或多或少令人迷惑的问题，特别是碰到一些复杂特殊的请求时候：例如文件上传，返回特殊的文件格式到浏览器，这时候使用servlet开发就不是很方便了，servlet开发还有个问题可能大家常常被忽视，就是请求的数据的类型转化，http协议传输都是文本形式，到了web容器解析后也是文本类型，如果碰到货币，数字，日期这样的类型需要我们根据实际情况进行转化，如果页面传送的信息非常多，我们就不得不做大量类型转化，这种工作没有什么技术含量，是个体力活而且很容易导致程序错误。同时java的企业开发都是围绕javabean进行，类型转化好的数据还要封装到对应的javabean里，这种转来转去的事情对于项目开发绝对不是什么好事情，所以古老的struts1为这种问题找到了一种解决方案，就是定义了一个DTO对象（数据传输对象），专门负责做这样的事情，不过到了struts2，整个替代servlet的action本身就是一个javabean。**

**Java的企业开发一个技术特点就是使用javabean进行的，struts2的特点之一就是它替代servlet的操作类就是一个典型的javabean，首先struts2框架将页面传输的数据进行类型转化和封装后将请求信息封装到了这个javabean的属性里，这样我们开发web程序时候就省去了烦心的类型转化和封装的问题，前面我讲到传统的servlet是根据http协议进行定义的，它会按你请求方式（post还是get方式）来处理用户的请求，但是对于一名程序开发人员而言，一个请求，具体到一个url，其实对于服务端而言就是服务端对外提供的一个功能，或者说是服务端对外的一个动作，如果我们使用servlet开发程序我们就得把http的动作转化为具体的业务动作，这就让程序开发变得繁琐，增强了开发的难度，所以struts2替代servlet的javabean就屏蔽了servlet里http的请求方式和具体业务动作转化的问题，javabean里的每一个方法都可以和每一个url请求一一对应，这必然减轻了开发的难度问题。**

**Servlet另一个作用就是构造response对象，让页面获得正确的响应，其实现代的浏览器是一个多媒体工具，文字，图片，视屏等等东西都可以在浏览器里显示，资源的不同就会导致http响应报文的差别，如果我们使用servlet开发就要根据资源的不同在java程序里用硬编码的形式处理，这样的程序很难复用，而且如果程序员对某种资源的处理理解不到位，就会导致问题的出现，struts2通过配置文件的形式将这样的逻辑从java程序里剥离出来，使用配置的方式进行统一管理，这个做法和spring的AOP方式类似，这样就让结果处理方式更加统一，更加利于管理，同时也提升了程序的健壮性以及降低了开发的难度。**

**Servlet在MVC开发模式里就是其中C层即控制层，控制层就像俄罗斯的双头鹰（一个头向东看一个头向西看）一样，一个头向M层模型层看，一个头向V层视图层看，模型层也是用java编写的，控制层也属于服务端语言开发，所以M层和C层的沟通没有天然的障碍，但是和V层视图层就不一样了，这是一个跨语言的沟通，对于浏览器，它只懂得html，javascript和css，浏览器是理解不了java这种语言的东西，但是要让服务端的东西能被浏览器理解接受，我们就必须得把服务端的响应信息放到页面里，因此就需要一个技术把java的信息转化到html页面里，这就是javaEE规范里提供了jsp技术，jsp其实是一种服务端技术而非客户端技术，不过它看起来似乎更像html技术，最早的jsp开发里都是直接将java代码写到页面里，这种坏处谁都知道，之后javaEE规范提供了自定义标签技术，使用一种类似html标签的方式来解析java代码，struts2框架提供了一整套完整的自定义标签技术，这似乎听起来不算啥，但是它的作用非凡，因为自定义标签之所以叫自定义就是每个人都可以自己来定义，如果没有一个规范必然产生混乱，而且一套完善的自定义标签是个系统工程，一套完整的自定义标签相当于我们在自己定义一套新的开发语言，做程序的人听到这个一定就会明白开发一套完整的自定义标签的工作量和开发难度都是难以想象的，而且自定义标签都是和控制层紧密相连，其难度又会增加一个维度，所以struts2提供的自定义标签对于业务开发带来的将是质的飞越。**

**Servlet里还有两个重要的技术：监听器和过滤器，对于监听器在web开发里使用的场景比较少，都是一些十分特别的情况才会使用，大部分web开发里可以忽略它的使用，我们用的最多的监听器可能就是对ServletContext创建和销毁的监听器，ServletContext是整个web应用的全局对象，它和Web应用的生命周期绑定在一起，因此使用这个监听器对Web应用的全局信息进行初始化和销毁操作，例如spring容器的初始化操作。比较有意思的是过滤器，在struts2里有个拦截器，它们的作用相同都是用来拦截请求的，因为拦截器是struts2的特有功能，在struts2里使用拦截器自然比使用过滤器更顺手，其实拦截器所用的技术比过滤器更加先进，因为拦截器使用了反射技术，因此拦截器拦截的面更大，控制请求的能力更强，它能完成的任务也会更加的丰富多彩。**

**在我第一次接触struts2时候，有人告诉我struts设计的一个目的就是想屏蔽在控制层里操作request和response对象，因为这两个http协议的儿子会造成web开发里思路的混乱，但是我在实际开发里却经常不自觉的使用这两个对象。而且本人做前端开发非常喜欢使用ajax，使用ajax技术时候我就很讨厌struts2的自定义标签，我更加喜欢在页面里用javascript技术处理各种信息，最终struts2在我眼里就是一个servlet的变体，因此曾经有段时间我常常在想是不是可以抛弃struts2，直接用servlet，因为struts2里用到了太多反射机制，特别是使用注解做配置（注解是用反射实现的），在java里反射的执行效率是非常低的，直接使用servlet一定能提升web应用的执行效率。其实这个倒很难做到，因为当时我没法在servlet里灵活的运用spring技术。**

**^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^**

 说完Servlet+jsp技术到Struts2技术的过渡。接下来谈谈Spring。

　　spring技术可以说是java企业开发里最重要的技术，不过真的理解spring的作用和意义还真是一件麻烦的事情，很多人对spring理解其实都是停留在使用阶段（例如：声明式事务很好用等等），当今的spring技术生态环境里可谓是蔚为壮观，spring已经包罗万象，它的内容之多完全不亚于它的本源java语言了，而spring这么大的框都是建立在ioc和aop技术之上，只有深入理解了这两个技术我们才能明白为什么spring这个框能装的下那么多东西了。

　　首先是ioc，ioc技术第一个解释叫做控制反转，它还有个解释就是依赖注入，这两个名字很难从字面理解，但是当你理解它的原理后就会发现它们的描述是何等准确。Ioc技术的本质就是构建对象的技术换句话说就是将一个类实例化成对象的技术，在java里实例化类通过new关键字进行的，每次new一个类都会产生一个新的实例对象，这么做视乎很浪费，有时这种浪费还挺危险，因为在程序开发时候我们常常只需要某个类永远只能产生一个的实例对象这个时候就得使用单例模式，此外在设计模式里还可以通过工厂方式产生对象，使用过spring的人看到上面的文字就知道了，spring里bean的定义就和上面的内容一一对应，scope属性single产生单例对象，prototype产生新对象，bean还可以通过工厂方式产生对象，可以说spring的bean就是制造对象的工具。面向对象编程里对象相当于显示生活中的一个实体，例如我们有个对象作用是完成打猎的操作，那么打猎这个对象内部包含两个辅助对象：人和枪，只有人和枪赋予了打猎这个对象，那么打猎对象才能完成打猎的操作，但是构建一个人和枪的对象并不是看起来那么简单，这里以枪为例，要创造一把枪我们需要金属，需要机床，需要子弹，而机床和子弹又是两个新对象，这些对象一个个相互嵌套相互关联，大伙试想下如果我们在java代码里构建一个枪的对象那是何其的复杂，假如我们要构造的不是简单的枪对象而是更加复杂的航空母舰，那么构造这个对象的成本之高是让人难以想象的，怎么来消除这种对象相互嵌套相互依赖的关系了？spring提供了一种方式，这种方式就是spring提供一个容器，我们在xml文件里定义各个对象的依赖关系，由容器完成对象的构建，当我们java代码里需要使用某个实例的时候就可以从容器里获取，那么对象的构建操作就被spring容器接管，所以它被称为控制反转，控制反转的意思就是本来属于java程序里构建对象的功能交由容器接管，依赖注入就是当程序要使用某个对象时候，容器会把它注入到程序里，这就叫做依赖注入。在java开发里我们想使用某个类提供的功能，有两种方式，一种就是构造一个新的类，新的类继承该类，另一种方式则是将某个类定义在新类里，那么两个类之间就建立一种关联关系，spring的ioc容器就是实现了这种关联关系（记住不是继承关系哦），那么某个类要被赋予到新类有哪些办法了？一般只有两种：一种就是通过构造函数，一种就是通过setXXX方式，这也是spring容器使用到了两种标准的注入方式。

　　不管是上面说的继承方式，还是关联方式其实都是增强目标对象能力的开发手段，在设计模式里有一种代理模式，代理模式将继承模式和关联模式结合在一起使用，代理模式就是继承模式和关联模式的综合体，不过这个综合体的作用倒不是解决对象注入的问题，而是为具体操作对象找到一个保姆或者是秘书，这就和小说里的二号首长一样，这个二号首长对外代表了具体的实例对象，实例对象的入口和出口都是通过这个二号首长，因为具体的实例对象是一号首长，一号首长是要干大事的，所以一些事务性，重复性的工作例如泡茶，安排车子，这样的工作是不用劳烦一号首长的大驾，而是二号首长帮忙解决的，这就是aop的思想，aop解决程序开发里事务性，和核心业务无关的问题，但这些问题对于业务场景的实现是很有必要的，在实际开发里aop也是节省代码的一种方式。

　　Spring的核心技术的作用本质就是一个沟通机制，spring总是尽全力的让沟通的双方信息畅通，同时降低双方的沟通成本，在现实机构里一个善于沟通的人肯定是该公司的领导，很会沟通的领导能调动起各种资源的积极性，善于沟通的领导就会做到海纳百川，让各种不同人追随他，所以当今的spring就是一个大框，什么都可以往里装。**Spring很像银行，它不能直接创造物质财富，但是一切资源都要通过它进行流通，**它能控制经济发展的走向，回到程序的世界，spring的作用是被标榜为程序之间的解耦，spring能降低不同模块之间的耦合度，原因就是在程序开发里不同模块之间信息的沟通是通过对象传递完成的，而对象能否顺利传递就是要合理的构建好对象，而管理好对象的构建方式就能管理好对象传递，这就是spring给系统架构设计带来的好处。

**^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^ ^\_^**

 说到Spring, Spring的事务你懂吗？

**什么是事务？为什么事务要管理？什么是Spring事务？**事务就是对一系列的数据库操作(比如插入多条数据)进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。开发中为了避免这种情况一般都会进行事务管理。在JDBC中，是通过Connection对象进行事务管理的，默认是自动提交事务，可以手工将自动提交关闭，通过commit方法进行提交，rollback方法进行回滚，如果不提交，则数据不会真正的插入到数据库中。Hibernate中则是通过Transaction进行事务管理，处理方法与JDBC中类似。Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管理，可以通过Spring的注入完成此功能。

**我通俗的理解如下：spring只是控制数据库的事务提交和回滚，借助于java的反射机制，在事务控制的方法（通常是service层的方法）前后获取事务开启session，然后执行你的数据操作，如果你的方法内有异常被抛出，spring会捕获异常并回滚你在这个方法内所有的数据操作，如果成功则提交所有的数据，最后spring会帮你关闭需要关闭的东西。所以spring想要做的是，要程序员专注于写逻辑，不需要关心数据库何时开启和关闭连接。**

**再说的通俗点儿：事务，对于一件事，对了就提交，错了就回滚，什么时候回滚，都是事务要做的事情。具体的操作由spring 配置来管理(同时你也可以脱离框架，自己写事务管理方法)。**

**使用Spring事务的优点？**

[复制代码](javascript:void(0);)

　　在SSH框假中Spring充当了管理容器的角色。我们都知道Hibernate用来做持久层，因为它将JDBC做了一个良好的封装，程序员在与数据库进行交互时可以不用书写大量的SQL语句。Struts是用来做应用层的，他它负责调用业务逻辑serivce层。所以SSH框架的流程大致是：Jsp页面----Struts------Service（业务逻辑处理类）---Hibernate（左到右）。struts负责控制Service（业务逻辑处理类），从而控制了Service的生命周期，这样层与层之间的依赖很强，属于耦合。这时，使用spring框架就起到了控制Action对象（Strus中的）和Service类的作用，两者之间的关系就松散了，Spring的Ioc机制（控制反转和依赖注入）正是用在此处。

　　Spring的Ioc（控制反转和依赖注入）

控制反转：就是由容器控制程序之间的（依赖）关系，而非传统实现中，由程序代码直接操控

依赖注入：组件之间的依赖关系由容器在运行期决定 ，由容器动态的将某种依赖关系注入到组件之中。

从上面我们不难看出：从头到尾Action仅仅是充当了Service的控制工具，这些具体的业务方法是怎样实现的，他根本就不会管，也不会问，他只要知道这些业务实现类所提供的方法接口就可以了。而在以往单独使用Struts框架的时候，所有的业务方法类的生命周期，甚至是一些业务流程都是由Action来控制的。层与层之间耦合性太紧密了，既降低了数据访问的效率又使业务逻辑看起来很复杂，代码量也很多。Spring容器控制所有Action对象和业务逻辑类的生命周期，由于上层不再控制下层的生命周期，层与层之间实现了完全脱耦，使程序运行起来效率更高，维护起来也方便。

　　使用Spring的第二个好处（AOP应用）：

事务的处理：

在以往的JDBCTemplate中事务提交成功，异常处理都是通过Try/Catch 来完成，而在Spring中。Spring容器集成了TransactionTemplate，她封装了所有对事务处理的功能，包括异常时事务回滚，操作成功时数据提交等复杂业务功能。这都是由Spring容器来管理，大大减少了程序员的代码量，也对事务有了很好的管理控制。Hibernate中也有对事务的管理，hibernate中事务管理是通过SessionFactory创建和维护Session来完成。而Spring对SessionFactory配置也进行了整合，不需要在通过hibernate.cfg.xml来对SessionaFactory进行设定。这样的话就可以很好的利用Sping对事务管理强大功能。避免了每次对数据操作都要现获得Session实例来启动事务/提交/回滚事务还有繁琐的Try/Catch操作。这些也就是Spring中的AOP（面向切面编程）机制很好的应用。一方面使开发业务逻辑更清晰、专业分工更加容易进行。另一方面就是应用Spirng AOP隔离降低了程序的耦合性使我们可以在不同的应用中将各个切面结合起来使用大大提高了代码重用度。有利于代码重用，特别是Dao代码的重用。事务往往和业务规则紧密关联。当业务逻辑发生改变，意味着dao的大幅度改动。系统规模达到一定程度，修改风险相当大。Spring的好处是不更改现有的dao，仅需对现有的service bean进行配置就达到事务效果了。同时，把事务统一在service层，系统结构更清晰。

为什么说风险风大？  
Spring对于事务的配置有两种方式：第一种，使用xml形式，第二种，使用注解的形式。  
基于XMl方式： 优点：可以在后期维护的时候适当的调整事务管理模式，并且只要遵循一定的命名规范，可以让程序员不必关心事务。  
　　　　　　 缺点：系统越庞大，xml配置就越大。  
基于注解方式：优点：配置比较方便，程序员只要在service层代码设置即可以实现。不需要知道系统需要多少个bean，交给容器来注入就好了。  
　　　　　　 缺点：当你要修改或删除一个bean的时候，你无法确定到底有多少个其他的bean依赖于这个bean。(解决方法：需要有严格的开发文档，在修改实现时尽可能继续遵守相应的接口避免使其他依赖于此的bean不可用)

[复制代码](javascript:void(0);)

　　在我们用SSH开发项目的时候，我们一般都是将事务设置在Service层 那么当我们调用Service层的一个方法的时候它能够保证我们的这个方法中执行的所有的对数据库的更新操作保持在一个事务中，在事务层里面调用的这些方法要么全部成功，要么全部失败。那么事务的传播特性也是从这里说起的。 如果你在你的Service层的这个方法中，除了调用了Dao层的方法之外，还调用了本类的其他的Service方法，那么在调用其他的 Service方法的时候，这个事务是怎么规定的呢，我必须保证我在我方法里掉用的这个方法与我本身的方法处在同一个事务中，否则如何保证事物的一致性。事务的传播特性就是解决这个问题的，“事务是会传播的”在Spring中有针对传播特性的多种配置我们大多数情况下只用其中的一种:PROPGATION\_REQUIRED：这个配置项的意思是说当我调用service层的方法的时候开启一个事务(具体调用那一层的方法开始创建事务，要看你的aop的配置),那么在调用这个service层里面的其他的方法的时候,如果当前方法产生了事务就用当前方法产生的事务，否则就创建一个新的事务。这个工作是由Spring来帮助我们完成的。 以前没有Spring帮助我们完成事务的时候我们必须自己手动的控制事务，例如当我们项目中仅仅使用hibernate，而没有集成进 spring的时候，我们在一个service层中调用其他的业务逻辑方法，为了保证事物必须也要把当前的hibernate session传递到下一个方法中，或者采用ThreadLocal的方法，将session传递给下一个方法，其实都是一个目的。现在这个工作由 spring来帮助我们完成，就可以让我们更加的专注于我们的业务逻辑。而不用去关心事务的问题。默认情况下当发生RuntimeException的情况下，事务才会回滚，所以要注意一下。如果你在程序发生错误的情况下，有自己的异常处理机制定义自己的Exception，必须从RuntimeException类继承，这样事务才会回滚！

　　补充文字来说一下上面的关于Spring的一个问题。基于xml和基于注解，当然了它们都有优缺点。我们通俗的说，是这样的。先来回顾一下传统上是如何配置 Bean 并完成 Bean 之间依赖关系的建立。下面是 3 个类，它们分别是 Office、Car 和 Boss，这 3 个类需要在 Spring 容器中配置为 Bean。

[复制代码](javascript:void(0);)

// Office.java  
public class Office {

private String officeNo =”001”;

//省略 get/setter

@Override

public String toString() {

return "officeNo:" + officeNo;

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// Car.java  
public class Car {

private String brand;

private double price;

// 省略 get/setter

@Override

public String toString() {

return "brand:" + brand + "," + "price:" + price;

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// Boss.java  
public class Boss {

private Car car;

private Office office;

// 省略 get/setter

@Override

public String toString() {

return "car:" + car + "\n" + "office:" + office;

}

}

[复制代码](javascript:void(0);)

**我们在 Spring 容器中将 Office 和 Car 声明为 Bean，并注入到 Boss Bean 中：下面是使用传统 XML 完成这个工作的配置文件 beans.xml：**

[复制代码](javascript:void(0);)

// **bean.xml将以上三个类配置成bean。**  
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="boss" class="com.baobaotao.Boss">

<property name="car" ref="car"/>

<property name="office" ref="office" />

</bean>

<bean id="office" class="com.baobaotao.Office">

<property name="officeNo" value="002"/>

</bean>

<bean id="car" class="com.baobaotao.Car" scope="singleton">

<property name="brand" value=" 红旗 CA72"/>

<property name="price" value="2000"/>

</bean>

</beans>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// **当我们运行这段代码时，控制台将正确打印出boss的信息。这说明 Spring 容器已经正确完成了 Bean 创建和装配的工作。**  
import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AnnoIoCTest {

public static void main(String[] args) {

String[] locations = {"beans.xml"};

ApplicationContext ctx =

new ClassPathXmlApplicationContext(locations);

Boss boss = (Boss) ctx.getBean("boss");

System.out.println(boss);

}

}

[复制代码](javascript:void(0);)

**我们知道 Spring 2.5 中引入了 @Autowired 注释，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。来看一下使用@Autowired 进行成员变量自动注入的代码：**

[复制代码](javascript:void(0);)

**// Autowired是自动装配的意思**  
import org.springframework.beans.factory.annotation.Autowired;

public class Boss {

@Autowired

private Car car;

@Autowired

private Office office;

}  
  
**// Spring 通过一个 BeanPostProcessor 对 @Autowired 进行解析，所以要让@Autowired 起作用必须事先在 Spring 容器中声明 AutowiredAnnotationBeanPostProcessor Bean。  
// 让系统认识@Autowired，让@Autowired注释工作起来。**

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- 该 BeanPostProcessor 将自动起作用，对标注 @Autowired 的 Bean 进行自动注入 -->

<bean class="org.springframework.beans.factory.annotation.

AutowiredAnnotationBeanPostProcessor"/>

<!-- 移除 boss Bean 的属性注入配置的信息 -->

<bean id="boss" class="com.baobaotao.Boss"/> **// 这里面没配置哦**

<bean id="office" class="com.baobaotao.Office">

<property name="officeNo" value="001"/>

</bean>

<bean id="car" class="com.baobaotao.Car" scope="singleton">

<property name="brand" value=" 红旗 CA72"/>

<property name="price" value="2000"/>

</bean>

</beans>

[复制代码](javascript:void(0);)

**这样，当 Spring 容器启动时，AutowiredAnnotationBeanPostProcessor 将扫描 Spring 容器中所有 Bean，当发现 Bean 中拥有@Autowired 注释时就找到和其匹配（默认按类型匹配）的 Bean，并注入到对应的地方中去。按照上面的配置，Spring 将直接采用 Java 反射机制对 Boss 中的**

**car 和 office 这两个私有成员变量进行自动注入。所以对成员变量使用@Autowired 后，您大可将它们的 setter 方法（setCar() 和 setOffice()）从 Boss 中删除。当然，您也可以通过 @Autowired 对方法或构造函数进行标注，来看下面的代码：**

[复制代码](javascript:void(0);)

public class Boss {

private Car car;

private Office office;

@Autowired

public void setCar(Car car) {

this.car = car;

}

@Autowired

public void setOffice(Office office) {

this.office = office;

}

}

[复制代码](javascript:void(0);)

**这时，@Autowired 将查找被标注的方法的入参类型的 Bean，并调用方法自动注入这些 Bean。而下面的使用方法则对构造函数进行标注：**

　Spring IOC三种注入方式（接口注入、setter注入、构造器注入）

public class Boss {

private Car car;

private Office office;

@Autowired

public Boss(Car car ,Office office){

this.car = car;

this.office = office ;

}

}

**// 由于 Boss() 构造函数有两个入参，分别是 car 和 office，@Autowired 将分别寻找和它们类型匹配的 Bean，将它们作为Boss(Car car ,Office office) 的入参来创建 Boss Bean。**  
  
**在默认情况下使用 @Autowired 注释进行自动注入时，Spring 容器中匹配的候选 Bean 数目必须有且仅有一个。当找不到一个匹配的 Bean 时，Spring 容器将抛出BeanCreationException 异常，并指出必须至少拥有一个匹配的 Bean。我们可以来做一个实验：**

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd ">

<bean class="org.springframework.beans.factory.annotation.

AutowiredAnnotationBeanPostProcessor"/>

<bean id="boss" class="com.baobaotao.Boss"/>

<!-- 将 office Bean 注释掉 -->

<!-- <bean id="office" class="com.baobaotao.Office"> **//然后你想，当多个bean之间互相依赖的时候，是不是维护起来很麻烦啊。**

<property name="officeNo" value="001"/>

</bean>-->

<bean id="car" class="com.baobaotao.Car" scope="singleton">

<property name="brand" value=" 红旗 CA72"/>

<property name="price" value="2000"/>

</bean>

</beans>

**由于 office Bean 被注释掉了，所以 Spring 容器中将没有类型为 Office 的 Bean 了，而 Boss 的office 属性标注了 @Autowired，当启动 Spring 容器时，异常就产生了。当不能确定 Spring 容器中一定拥有某个类的 Bean 时，可以在需要自动注入该类 Bean 的地方可以使用 @Autowired(required = false)，这等于告诉 Spring：在找不到匹配 Bean 时也不报错。来看一下具体的例子：**

[复制代码](javascript:void(0);)

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Required;

public class Boss {

private Car car;

private Office office;

@Autowired

public void setCar(Car car) {

this.car = car;

}

@Autowired(required = false)

public void setOffice(Office office) {

this.office = office;

}

}

[复制代码](javascript:void(0);)

**当然，一般情况下，使用 @Autowired 的地方都是需要注入 Bean 的，使用了自动注入而又允许不注入的情况一般仅会在开发期或测试期碰到（如为了快速启动 Spring 容器，仅引入一些模块的 Spring 配置文件），所以@Autowired(required = false) 会很少用到。和找不到一个类型匹配 Bean 相反的一个错误是：如果 Spring 容器中拥有多个候选 Bean，Spring 容器在启动时也会抛出 BeanCreationException 异常。来看下面的例子：**

[复制代码](javascript:void(0);)

**// 在 beans.xml 中配置两个 Office 类型的 Bean**  
<bean id="office" class="com.baobaotao.Office">

<property name="officeNo" value="001"/>

</bean>

<bean id="office2" class="com.baobaotao.Office">

<property name="officeNo" value="001"/>

</bean>

[复制代码](javascript:void(0);)

**我们在 Spring 容器中配置了两个类型为 Office 类型的 Bean，当对 Boss 的 office 成员变量进行自动注入时，Spring 容器将无法确定到底要用哪一个 Bean，因此异常发生了。Spring 允许我们通过 @Qualifier 注释指定注入 Bean 的名称，这样歧义就消除了，可以通过下面的方法解决异常：**

@Autowired

public void setOffice(@Qualifier("office")Office office) {

this.office = office;

}

**@Qualifier("office") 中的 office 是 Bean 的名称，所以 @Autowired 和@Qualifier 结合使用时，自动注入的策略就从 byType 转变成 byName 了。@Autowired 可以对成员变量、方法以及构造函数进行注释，而@Qualifier 的标注对象是成员变量、方法入参、构造函数入参。正是由于注释对象的不同，所以 Spring 不将 @Autowired 和@Qualifier 统一成一个注释类。下面是对成员变量和构造函数入参进行注释的代码：对成员变量进行注释：**

[复制代码](javascript:void(0);)

public class Boss {

@Autowired

private Car car;

@Autowired

@Qualifier("office")

private Office office;

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

**我们面试中提到Spring框架经常被问到的一块儿是Spring的事务，它优于struts2，优于hibernate。那么我们现在就详细的说说这个东西。**

// 事务的特性：原子性、一致性、隔离性、持久性。  
  
常常问的：1. Spring事务的传播特性; 2. Spring事务的隔离机制。

**我们先来探讨一下数据库事务的隔离级别：**

[复制代码](javascript:void(0);)

事务的（ACID）特性是由关系数据库管理系统（RDBMS，数据库系统）来实现的。数据库管理系统采用**日志**来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。  
  
**幻读** 是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，比如这种修改涉及到表中的“全部数据行”。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入“一行新数据”。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样.一般解决幻读的方法是增加范围锁RangeS，锁定检锁范围为只读，这样就避免了幻读。  
  
**脏读** 就是指当一个[事务](http://baike.baidu.com/view/121511.htm)正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是[脏数据](http://baike.baidu.com/view/1445322.htm)，依据脏数据所做的操作可能是不正确的。

　数据库系统有四个**隔离级别**。对数据库使用何种隔离级别要审慎分析，因为维护一个最高的隔离级别虽然会防止数据的出错，但是却导致了并行度的损失，以及导致死锁出现的可能性增加。然而，降低隔离级别，却会引起一些难以发现的bug。

　　1. 序列化 Serializable, 添加范围锁（比如表锁，页锁等，关于range lock，我也没有很深入的研究），直到transaction A结束。以此阻止其它trasaction B对此范围内的insert，update等操作。幻读，脏读，不可重复读等问题都不会发生。

　　2. 可重复读 repeatable read, 对于读出的记录，添加共享锁直到transaction A结束。其它transaction B对这个记录的试图修改会一直等待直到trasaction A结束。**InnoDB**默认级别**。**可能发生的问题：当执行一个范围查询时，可能会发生幻读。

　　3. 提交读 read commited, 在trasaction A中读取数据时对记录添加共享锁，但读取结束立即释放。其它transaction B对这个记录的试图修改会一直等待直到A中的读取过程结束，而不需要整个trasaction A的结束。所以，在trasaction A的不同阶段对同一记录的读取结果可能是不同的。

       可能发生的问题：不可重复读。

　　4. 未提交读 read uncommited, 不添加共享锁。所以其它trasaction B可以在trasaction A对记录的读取过程中修改同一记录，可能会导致A读取的数据是一个被破坏的或者说不完整不正确的数据。另外，在trasaction A中可以读取到trasaction B（未提交）中修改的数据。比如trasaction 　　　　B对R记录修改了，但未提交。此时，在Trasaction A中读取R记录，读出的是被B修改过的数据。可能发生的问题：脏读。

[复制代码](javascript:void(0);)

 那么Spring的事务隔离机制和Spring事务的传播属性呢，Spring的隔离级别我们知道目的是为了防止幻读和脏读。Spring的事务传播属性似乎好难理解。。接下来，先解释下什么叫做事务的传播属性：

[复制代码](javascript:void(0);)

我们都知道事务的概念，那么事务的传播特性是什么呢？(先着重介绍传播特性的概念，关于传播特性的相关配置稍后再介绍)

**背景**：当我们用SSH开发项目的时候，我们一般都是将事务设置在Service层 那么当我们调用Service层的一个方法的时候它能够保证我们的这个方法中执行的所有的对数据库的更新操作保持在一个事务中，在事务层里面调用的这些方法要么全部成功，要么全部失败。那么事务的传播特性也是从这里说起的。

**场景**：如果你在你的Service层的这个方法中，除了调用了Dao层的方法之外，还调用了本类的其他的Service方法，那么在调用其他的Service方法的时候，这个事务是怎么规定的呢，我必须保证我在我方法里掉用的这个方法与我本身的方法处在同一个事务中，否则如何保证事物的一致性。事务的传播特性就是解决这个问题的，“事务是会传播的”在Spring中有针对传播特性的多种配置我们大多数情况下只用其中的1种:PROPGATION\_REQUIRED：这个配置项的意思是说当我调用service层的方法的时候开启一个事务(具体调用那一层的方法开始创建事务，要看你的aop的配置),那么在调用这个service层里面的其他的方法的时候,如果当前方法产生了事务就用当前方法产生的事务，否则就创建一个新的事务。这个工作使由Spring来帮助我们完成的。

以前没有Spring帮助我们完成事务的时候我们必须自己手动的控制事务，例如当我们项目中仅仅使用hibernate，而没有集成进spring的时候，我们在一个service层中调用其他的业务逻辑方法，为了保证事物必须也要把当前的hibernate session传递到下一个方法中，或者采用ThreadLocal的方法，将session传递给下一个方法，其实都是一个目的。现在这个工作由spring来帮助我们完成，就可以让我们更加的专注于我们的业务逻辑。而不用去关心事务的问题。默认情况下当发生RuntimeException的情况下，事务才会回滚，所以要注意一下如果你在程序发生错误的情况下，有自己的异常处理机制定义自己的Exception，必须从RuntimeException类继承这样事务才会回滚！

[复制代码](javascript:void(0);)

**好了，知道了事务的传播特性之后，我们看看Spring提供的六种事务传播Propagation特性：**仔细感觉一下，六中Spring事务传播机制 **对称美**

1. Propagation required, 如果当前没有事务，就新建一个事务。

2. Propagation supports, 如果当前没有事务，就以非事务方式执行。

3. Propagation mandatory, 如果当前没有事务，就抛出异常。

4. Propagation requires new, 如果当前存在事务，挂起当前事务，新建事务。

5. Propagation not supported, 如果当前存在事务，把当前事务挂起，以非事务方式执行。

6. Propagation never, 如果当前存在事务，抛出异常。以非事务方式执行。

关于六种事务传播机制的具体应用场景，参考： [Spring事务传播机制博客](http://blog.csdn.net/it_man/article/details/5074371)

 了解了Spring事务的传播特性，再来看看Spring事务的五种隔离级别isolation level。

[复制代码](javascript:void(0);)

// 先再来回顾前面说到的三个概念

脏读: 指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

不可重复读: 指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

幻觉读: 指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// 再来看Spring事务的隔离级别isolation level：

1. isolation default： 这是一个默认的隔离级别，使用数据库默认的事务隔离级别.(一般情况下，使用这种)。另外四个与JDBC的隔离级别相对应.

2. isolation read uncommited： 这个是事务最低的隔离级别，它允许令外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。(trasaction B可以在trasaction A对记录的读取过程中修改同一记录)  
　　 产生脏读、产生不重复读、产生幻读。

3. isolation read commited： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据.(trasaction B提交后trasaction A才能读取)  
　　 避免脏读、会产生不重复读、会产生幻读。

4. isolation repeatable read： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。  
　　 避免脏读、避免不重复读、会产生幻读。

5. isolation serializable.这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。 除了防止脏读，不可重复读外，还避免了幻读。  
　　 避免脏读、避免不重复读、避免幻读。

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

　　换种口吻说，声明式事务。声明式事务是Spring提供的对程序事务管理的方式之一。Sping的声明式事务，就是在配置文件中采用配置的方式对事务进行管理。Spring中的AOP即，是完成事务管理工作的。

总结一下我们出几道面试题整合一下Spring3的知识。

1. Spring的工作原理：

1.客户端请求提交到DispatcherServlet

2. 由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller

3. DispatcherServlet将请求提交到Controller

4. Controller调用业务逻辑处理后，返回ModelAndView

5. DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图

6. 视图负责将结果显示到客户端

 2. 为什么要用Spring？

[复制代码](javascript:void(0);)

1、Spring能很好的与各大框架进行集成

2、创建对象时，如果我们不用spring。需要用工厂模式来创建，这个spring相当于工厂模式已经帮我们做了创建对象的功能(IOC、依赖注入)。

3、在用Hibernate的时候，如果不用spring每次都要写事务的提交代码，有了spring可以通过AOP帮助我们管理事务。

4、面向切面编程（AOP）在要记录日志的时候添加一条记录后需要在数据里同时添加一条添加成功了或失败的记录，那么就可以用Spring的Aop来处理，虽然不用Aop也能做但是不用Spring的Aop就会写很多重复的代码。  
  
AOP 让开发人员可以创建非行为性的关注点，称为横切关注点，并将它们插入到应用程序代码中。使用 AOP 后，公共服务 （比 如日志、持久性、事务等）就可以分解成方面并应用到域对象上，同时不会增加域对象的对象模型的复杂性。IOC 允许创建一个可以构造对象的应用环境，然后向这些对象传递它们的协作对象。正如单词 倒置 所表明的，IOC 就像反 过来的 JNDI。没有使用一堆抽象工厂、服务定位器、单元素（singleton）和直接构造（straight construction），每一个对象都是用 其协作对象构造的。因此是由容器管理协作对象（collaborator）。Spring即使一个AOP框架，也是一IOC容器。 Spring 最好的地方是它有助于您替换对象。有了 Spring，只要用 JavaBean 属性和配置文件加入依赖性（协作对象）。然后可以很容易地在需要时替换具有类似接口的协作对象。

[复制代码](javascript:void(0);)

 3. 请你谈谈SSH的整合？

[复制代码](javascript:void(0);)

请你谈谈SSH整合

SSH：Struts（表示层）+Hibernate（持久层）+Spring（业务层）

a、Struts

Struts是一个表示层框架，主要作用是界面展示，接收请求，分发请求。

b、Hibernate

Hibernate是一个持久层框架，它只负责与关系数据库的操作。

c、Spring

Spring是一个业务层框架，是一个整合的框架，能够很好地黏合表示层与持久层。

[复制代码](javascript:void(0);)

4. 介绍一下Spring的事务管理？

介绍一下Spring的事务管理

事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。开发中为了避免这种情况一般都会进行事务管理。Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管理，可以通过Spring的注入来完成此功能。

 5. 什么是依赖注入，依赖注入的作用是什么？

什么是依赖注入，依赖注入的作用是什么?

IOC是一种思想，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试. 有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

依赖注入的作用：减少类间耦合度，避免用new来创建对象。

 6.什么是aop，aop的作用是什么？

什么是AOP，AOP的作用是什么?

AOP，面向切面编程，就是把可重用的功能提取出来，然后将这些通用功能在合适的时候织入到应用程序中，比如事务管理、权限控制、日志记录、性能统计等。

AOP的作用

AOP并没有帮助我们解决任何新的问题，它只是提供了一种更好的办法，能够用更少的工作量来解决现有的一些问题，使得系统更加健壮，可维护性更好。

7. **Spring中的BeanFactory与ApplicationContext的作用有哪些?**

1、BeanFactory负责读取bean的配置文件，管理bean的加载、实例化，维护bean之间的依赖关系，负责bean的生命周期。

2、ApplicationContext除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能：

a. 国际化支持

b. 资源访问

c. 事件传递

8.  Hibernate的工作原理？

[复制代码](javascript:void(0);)

Hibernate工作原理及为什么要用？

原理：

1.读取并解析配置文件

2.读取并解析映射信息，创建SessionFactory

3.打开Sesssion

4.创建事务Transation

5.持久化操作

6.提交事务

7.关闭Session

8.关闭SesstionFactory

[复制代码](javascript:void(0);)

9. 为什么要用hibernate？

// 为什么要用：

1. 对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。

2. Hibernate是一个基于JDBC的主流持久化框架，是一个优秀的ORM实现。他很大程度的简化DAO层的编码工作

3. hibernate使用Java反射机制，而不是字节码增强程序来实现透明性。

4. hibernate的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多的各种复杂关系。

 10. Hibernate如何延迟加载？

Hibernate是如何延迟加载?

1. Hibernate2延迟加载实现：a)实体对象 b)集合（Collection）

2. Hibernate3 提供了属性的延迟加载功能

当Hibernate在查询数据的时候，数据并没有存在与内存中，当程序真正对数据的操作时，对象才存在与内存中，就实现了延迟加载，他节省了服务器的内存开销，从而提高了服务器的性能。

11. Hibernate怎么样实现类之间的关系？

Hibernate中怎样实现类之间的关系?(如：一对多、多对多的关系)

类与类之间的关系主要体现在表与表之间的关系进行操作，它们都是对对象进行操作，我们程序中把所有的表与类都映射在一起，它们通过配置文件中的many-to-one、one-to-many、many-to-many、

 12. 说下hibernate的缓存机制。

[复制代码](javascript:void(0);)

详细说下hibernate的缓存机制：  
  
  
 // 为什么要用hibernate缓存：

Hibernate是一个持久层框架，经常访问物理数据库。为了降低应用程序对物理数据源访问的频次，从而提高应用程序的运行性能。缓存内的数据是对物理数据源中的数据的复制，应用程序在运行时从缓存读写数据，在特定的时刻或事件会同步缓存和物理数据源的数据。

 // Hibernate的缓存原理是怎么样的？

Hibernate一级缓存又称为“Session的缓存”。Session的缓存是事务范围的缓存（Session对象的生命周期通常对应一个数据库事务或者一个应用事务）。

Hibernate二级缓存又称为“SessionFactory的缓存”。由于SessionFactory对象的生命周期和应用程序的整个过程对应，因此Hibernate二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，默认下SessionFactory不会启用这个插件。

   // 既然二级缓存是进程级别的缓存，那么它适合缓存什么类型的数据呢？

　　　　　　什么样的数据适合存放到第二级缓存中？ 　　  
　　　　　　　　　　1) 很少被修改的数据 　　  
　　　　　　　　　　2) 不是很重要的数据，允许出现偶尔并发的数据 　　  
　　　　　　　　　　3) 不会被并发访问的数据 　　  
　　　　　　　　　　4) 常量数据 　　  
　　不适合存放到第二级缓存的数据？ 　　  
　　　　　　　　　　1) 经常被修改的数据 　　  
　　　　　　　　　　2) 绝对不允许出现并发访问的数据，如财务数据，绝对不允许出现并发 　　  
　　　　　　　　　　3) 与其他应用共享的数据。

　　// Session的延迟加载

　　Session的延迟加载实现要解决两个问题：正常关闭连接和确保请求中访问的是同一个session。Hibernate session就是java.sql.Connection的一层高级封装，一个session对应了一个Connection。http请求结束后正确的关闭session（过滤器实现了session的正常关闭）；延迟加载必须保证是同一个session（session**绑定**在ThreadLocal）。

　　// Hibernate查找对象, 如何应用缓存？

当Hibernate根据ID访问数据对象的时候，首先从Session一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；如果都查不到，再查询数据库，把结果按照ID放入到缓存删除、更新、增加数据的时候，同时更新缓存。

[复制代码](javascript:void(0);)

 接下来说说Spring事务的配置：

　　Spring 如果没有特殊说明，一般指是跟数据存储有关的数据操作事务操作；对于数据持久操作的事务配置，一般有三个对象，**数据源(dataSouce)，事务管理器(transactionManager)，以及事务代理机制**；Spring 提供了多种的底层数据源实现，以及多种类型的事务管理器；所有的管理器都基于 Platform Transaction Manager 接口实现各自的事务策略；Spring 事务管理采用 AOP 切面代理技术实现，AOP 用于分隔关注点，保证事务的原子性，采用一定的技术 把该关注点 (weaving) 织入到 待完善的关注点上，实现单独组件无法实现的功能，以解决面向对象编程在某些方式下难于实现的操作，更好的支持面向对象的开关原则(扩展开放，修改关闭)。

**对于三部分:dataSource、transactionManager、事务代理机制。无论哪种配置方式，一般变化的都是代理机制部分。DataSource、TransactionManager这两部分只是会根据数据访问方式有所变化。比如使用Hibernate进行数据访问时，DataSource实际为SessionFactory，TransactionManager的实现为HibernateTransactionManager**。

　　那么我们知道了事务有五种配置方式和三个对象，接下来说说它的层面：Spring的事务到底该给Dao配置还是给Service配置呢？Spring的事务为业务逻辑进行事务管理，保证业务逻辑上数据的原子性。事务根据项目性质来细分：事务可以设置到三个层面(dao层、service层和web层)，第一：web层事务，这一般是针对那些安全性要求较高的系统来说的。例如电子商务网站。粒度小，一般系统用不着这么细。第二：service层事务，这是一常见的事务划分， 将事务设置在业务逻辑上，只要业务逻辑出错或异常就事务回滚。粒度较小，一般推荐这种方式。第三：数据持久层数据务，也就是常说的数据库事务。这种事务在安全性方面要求低。就是给一个简单的增删改之类的操作增加事务操作，粒度大。

　　Spring声明式事务让我们从复杂的事务处理中得到解脱。使得我们再也无需要去处理获得连接、关闭连接、事务提交和回滚等这些操作。再也无需要我们在与事务相关的方法中处理大量的try…catch…finally代码。   
我们在使用Spring声明式事务时，有一个非常重要的概念就是事务属性。事务属性通常由事务的传播行为，事务的隔离级别，事务的超时值和事务只读标志组成。我们在进行事务划分时，需要进行事务定义，也就是配置事务的属性。给Service层配置事务，因为一个Service层方法操作可以关联到多个DAO的操作。在Service层执行这些Dao操作，多DAO操作有失败全部回滚，成功则全部提交。事务分为业务事务和系统事务，业务事务也就是业务逻辑上操作的一致性，系统事务自然就是指真正的数据库事务，Spring配置事务的是为了什么进行管理，当然是为业务逻辑进行事务管理，保证业务逻辑上数据的原子性；Dao层是什么，数据访问层，是不应该包含业务逻辑的，这就是和Service层的不同；Service层就是业务逻辑层，事务的管理就是为Service层上的保证。

[复制代码](javascript:void(0);)

package com.bluesky.spring.dao;

import java.util.List;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import org.springframework.stereotype.Component;

import com.bluesky.spring.domain.User;

**@Transactional**  // 看代码(关于事务的配置)

@Component("userDao")

public class UserDaoImpl extends HibernateDaoSupport implements UserDao {

public List<User> listUsers() {

return this.getSession().createQuery("from User").list();

}

}   
  
// 在DAO层上加了注解 @Transactional ，这种申明式事务方式，让我们从复杂的事务处理中得到解脱，使得我们再也无需要去处理获得连接、关闭连接、事务提交和回滚等这些操作。再也无需要我们在与事务相关的方法中处理大量的try…catch…finally代码。

[复制代码](javascript:void(0);)

接下来，我们看看事务的定义

[复制代码](javascript:void(0);)

// TransactionDefinition

public interface TransactionDefinition {

int getPropagationBehavior(); //方法 返回事务的传播行为，是否由一个活动事务来决定一个事务的调用。

int getTimeout(); //方法 返回事务必须在多少秒内执行

boolean isReadOnly(); //方法返回事务是否是只读的

　　　 int getIsolationLevel(); // 返回事务的隔离级别

}   
  
说说事务的隔离级别：在**TransactionDefinition**接口中定义了五个不同的事务隔离级别，**ISOLATION\_DEFAULT**这是一个默认的隔离级别，使用数据库默认的事务隔离级别.另外四个与JDBC的隔离级别相对应。**ISOLATION\_READ\_UNCOMMITTED** 这是事务最低的隔离级别，它充许别外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。

[复制代码](javascript:void(0);)

例如:

Mary的原工资为1000,财务人员将Mary的工资改为了8000，但未提交事务

Java代码

1. Connection con1 = getConnection();

2. con.setAutoCommit(false);

3. update employee set salary = 8000 where empId ="Mary";

与此同时，Mary正在读取自己的工资

Java代码

1. Connection con2 = getConnection();

2. select salary from employee where empId ="Mary";

3. con2.commit();

Mary发现自己的工资变为了8000，欢天喜地！

而财务发现操作有误，而回滚了事务,Mary的工资又变为了1000

Java代码

1. //con1

2. con1.rollback();

像这样,Mary记取的工资数8000是一个脏数据。

[复制代码](javascript:void(0);)

**ISOLATION\_READ\_COMMITTED**保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。**ISOLATION\_REPEATABLE\_READ**这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

[复制代码](javascript:void(0);)

在事务1中，Mary 读取了自己的工资为1000,操作并没有完成

Java代码

1. con1 = getConnection();

2. select salary from employee empId ="Mary";

在事务2中，这时财务人员修改了Mary的工资为2000,并提交了事务.

Java代码

1. con2 = getConnection();

2. update employee set salary = 2000;

3. con2.commit();

在事务1中，Mary 再次读取自己的工资时，工资变为了2000

Java代码

1. //con1

2. select salary from employee empId ="Mary";

在一个事务中前后两次读取的结果并不致，导致了不可重复读。

使用ISOLATION\_REPEATABLE\_READ可以避免这种情况发生。

[复制代码](javascript:void(0);)

**ISOLATION\_SERIALIZABLE**这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

[复制代码](javascript:void(0);)

@Transactional只能被应用到public方法上, 对于其它非public的方法,如果标记了@Transactional也不会报错,但方法没有事务功能。Spring使用声明式事务处理，默认情况下，如果被注解的数据库操作方法中发生了unchecked异常，所有的数据库操作将rollback；如果发生的异常是checked异常，默认情况下数据库操作还是会提交的。这种默认的行为是可以改变的。[事务详细讲解的文章](http://www.cnblogs.com/%E4%BA%8B%E5%8A%A1%E8%AF%A6%E7%BB%86%E8%AE%B2%E8%A7%A3%E7%9A%84%E6%96%87%E7%AB%A0)

第一种方式，每个bean都有一个代理类。

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans">

<!-- 数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://192.168.0.244:3306/test?useUnicode=true&amp;characterEncoding=UTF-8" />

<property name="username" value="root" />

<property name="password" value="root" />

<!-- 连接池启动时的初始值 -->

<property name="initialSize" value="10" />

<!-- 连接池的最大值 -->

<property name="maxActive" value="10" />

<!-- 最大空闲值.当经过一个高峰时间后，连接池可以慢慢将已经用不到的连接慢慢释放一部分，一直减少到maxIdle为止 -->

<property name="maxIdle" value="20" />

<!-- 最小空闲值.当空闲的连接数少于阀值时，连接池就会预申请去一些连接，以免洪峰来时来不及申请 -->

<property name="minIdle" value="10" />

<property name="defaultAutoCommit" value="true" />

</bean>

<!-- 会话工厂 -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="mappingLocations">

<list>

<value>classpath:/com/nms/entity/\*\*/\*.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect"> org.hibernate.dialect.MySQL5Dialect </prop>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.format\_sql">true</prop>

</props>

</property>

</bean>

<!-- 定义事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 配置服务层 -->

<bean id="userDaoAgency" class="com.dao.impl.UserDaoImpl">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<bean id="userDao" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<!-- 配置事务管理器 -->

<property name="transactionManager" ref="transactionManager" />

<property name="target" ref="userDaoAgency" />

<property name="proxyInterfaces" value="com.dao.UserDao" />

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

<prop key="\*">PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

</beans>

[复制代码](javascript:void(0);)

第二种方式，所有bean共享一个代理类。

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<beans>

<!-- 数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<!-- // 配置同上 -->

</bean>

<!-- 会话工厂 -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<!-- // 配置同上 -->

</bean>

<!-- 定义事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 定义事务 -->

<bean id="base" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean" lazy-init="true" abstract="true">

<!-- 配置事务管理器 -->

<property name="transactionManager" ref="transactionManager" />

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

<prop key="\*">PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

<!-- 配置服务层 -->

<bean id="userDao" class="com.dao.impl.UserDaoImpl">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 代理对象 -->

<bean id="userDaoAgency" parent="base">

<property name="target" ref="userDao" />

</bean>

</beans>

[复制代码](javascript:void(0);)

 第三种方式，使用拦截器。