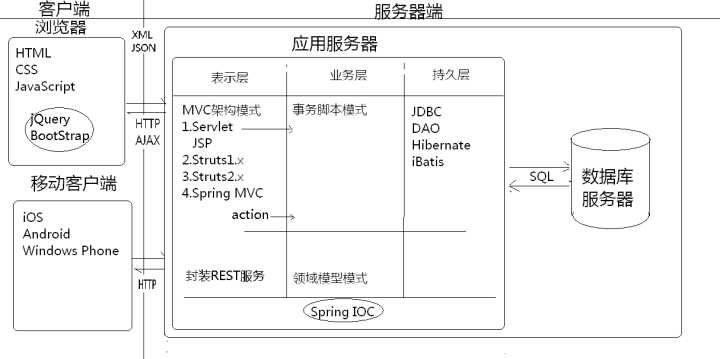
**java web 中dao 层和service层都使用接口，是否是为使用接口而使用接口？**

我想上面很多答者大概是没弄明白提问人的意思 -:)

JavaWeb 开发中，服务器端通常分为表示层、业务层、持久层，这就是所谓的三层架构。三层架构的每一层都有自己的开发模式，即架构模式，如下图。

其中，表示层一般是采用 MVC 架构模式，业务层有事务脚本模式、领域模型模式等，持久层有数据映射器模式（Hibernate即是典型的）、入口模式（iBatis、JDBC）。企业应用中最关键的显然是业务层。而对于初学者来说，事务脚本模式是最为简单，最容易掌握的。如果开发团队面向对象设计能力一般，而且业务逻辑相对简单，业务层一般都会采用事务脚本模式。为嘛？简单呀，是个人都能很快学会！（当然，如果业务逻辑复杂，用事务脚本模式就很不明智了。嗯，简单点讲，就是违背了单一职责设计原则，Service类成为万能的上帝，承担了太多职责。。。）



那么什么是事务脚本模式呢？

所谓事务，就是表示层的一个请求；所谓脚本就是一个方法或者一个函数；所谓事务脚本就是将一次请求封装为一个方法或者一个函数。所谓事务脚本模式，就是将业务层的对象分为三类，按照下图的方式组织起来（下图是用EA画的UML类图，一个简单的学生管理的业务层设计）。如上图所示，在事务脚本模式中，有三类对象。其中，Service类封装业务流程（或者说是界面上的业务流程），DAO类封装对持久层的访问，DTO类封装业务实体对象。各个对象之间的关系如上图所示，这就是所谓业务逻辑的组织方式。

OK，现在可以回答提问人的问题了。

为什么要用Service接口和DAO接口？我们还得回到最基本的面向对象设计原则上去。

面向对象设计原则中有三条与此相关：开闭原则、依赖倒转原则、理氏替换原则。还记得依赖倒转原则吧？高层不依赖于低层，二者都依赖于抽象，也就是面向接口编程。

为什么要用Service接口？是让表示层不依赖于业务层的具体实现。为什么要用DAO接口？是让业务层不依赖于持久层的具体实现。有了这两个接口，Spring IOC容器才能发挥作用。

举个例子，用DAO接口，那么持久层用Hibernate，还是用iBatis，还是 JDBC，随时可以替换，不用修改业务层Service类的代码。

OK，使用接口的意义就在此。

使用接口是为了调用与实现解耦，带来的好处是可以各干各的了，带来的坏处是从一个概念变成了两个概念，增加了系统的复杂度。衡量一下在具体场景中是弊大于利还是利大于弊，就可以做选择了。当然，在大部分场景下，还要考虑一个因素，就是你会不会写接口。没有良好接口设计能力的人，写出来的接口抽象不合理，等于没写，什么好处都得不到，只有坏处，这种情况下干脆别写。那怎么衡量你会不会写接口呢，我的经验是，至少见过一次写了接口后得到明确好处的例子。

什么情况下需要各干各的？

最简单的场景，写接口的是你，写实现的是你小弟。当然大多数类似情况没必要真的建一个interface然后再让人家去implements，把函数的第一行写好，注释写好，代码提交上，里面的内容让小弟去填就行了。

另一种情况，调用代码先于实现代码编写。比如你开发的是struts这种东西，那你指定得搞个Action接口。

再一种情况，多种业务的模式类似。此时这个接口类实际上相当于某一层的抽象。定义出一个层后，有多种实现，然后通过向调用端注入不同的实现类，实现不同的逻辑。如果这种注入不能在编译期完成的话，也就需要用接口抽象一下。

上面这几种情况写得有点绕，没办法，太难表述了并且好多事我自己也没想明白……

说到题目中的场景。

先说dao，这玩意儿是做数据库读写的。对应一下上面那几种情况：你作为项目架构师想写两行代码就让苦逼小弟加班干活然则自己去泡妹子的话，可能需要写个interface里面几个抽象的insert、delete之类的函数；项目在快速原型阶段如果客户满意就掏钱买oracle如果客户不满意就得免费MySQL的话，你可能需要定义个dao接口然后先用内存数据库写点能让原型跑起来的实现，等一切有定论了再说；每个类都有一个dao，每个dao都有crud基本方法的话你可能需要定义一个通用Dao接口然后搞点代码技巧不用一个个的去写体力代码从此登上人生巅峰。所以dao接口还是有用的。

再说service，这玩意儿更得具体问题具体分析。不去抠理论的话，什么是service，我的理解就是一段段实现了某个逻辑的代码的组合。所以service是个比dao更抽象的概念，严格来讲dao就是一种service。只不过在java web开发中，dao是个人人都得写的东西，所以都拿出来单说了。因此，后面说的service跟dao没有本质分别。

还是上面说的那几种情况：

先从工序上说，你在写上一层的时候，会用到下一层提供的逻辑，具体表现形式就是各种各样的service类和里面的方法。上一层开搞的时候，一定会知道的一个事是下一层会干什么事，比如“将传入编号对应的人员信息设置为离职”，但下一层的代码不一定已经一行一行写出来了。所以这会儿需要有个接口，让写上层代码的人先能把代码写下去。有各种理由可以支持这种工序的合理性，比如一般来说，上一层的一行代码会对应下一层的好多行代码，那先让写上层代码的人写一遍，解决高端层面的bug，会提高很多效率。

再从抽象角度说，不同业务模块之间的共用，不一定是共用某段代码，也可能是共用某段逻辑框架，这时候就需要抽象一个接口层出来，再通过不同的注入逻辑实现。比如模块1是登记学生信息，模块2是新闻发布，看上去风马牛不相及。但分析下来如果两个模块都有共同点，顺序都是1、验证是否有权限 2、验证输入参数是否合法 3、将输入参数转化为业务数据 4、数据库存取 5、写log，那就可以写一个service接口，里面有上述5个函数，再分别写两个service实现。具体执行的时候，通过各种注入方法，直接new也好，用spring注入也好，实现不同的效果。

当然上面说的这种情况很少有人这么干，因为已经普遍到这个程度，再精化精化就是struts了，java web的各种mvc框架都提供机制给你干这个事。但是每个项目或产品，都应该可以用类似的思路抽象出一些东西，如果抽象合理，会很大程度的提高项目架构的合理性。

这些要是能搞定，那什么写个接口然后实现个mock用于测试这种事，信手拈来。

想说的差不多都说了，发现写乱了……凑合看吧，我没什么理论基础，都是做各种小项目的野路子。

====强行装逼分割线===

1、项目架构师要不断找出各种层面的重复，然后消灭之。

2、《梦断代码》中写道，很多人把软件工程的各个要素视为开关，于是总想着是开还是关，但其实这些要素都是旋钮，你需要仔细的调整每个旋钮的位置，才能写出合格的软件作品。

3、软件工程的精髓就在两个字：**折衷**。

什么是面向接口编程？

在系统分析和架构中，分清层次和依赖关系，每个层次不是直接向其上层提供服务（即不是直接实例化在上层中），而是通过定义一组接口，仅向上层暴露其接口功能，上层对于下层仅仅是接口依赖，而不依赖具体类。

这样做的好处是显而易见的，首先对系统灵活性大有好处。当下层需要改变时，只要接口及接口功能不变，则上层不用做任何修改。甚至可以在不改动上层代码时将下层整个替换掉。

对系统的解耦有很大的作用。

在Spring当中的配置文件设置中，我们可以不修改代码，直接修改配置文件当中配置的对象，就可以修改相应的功能。