一直疑问，这个接口的bean是怎么注入进去的？因为只看到使用@Service注入了实现类serviceImpl，使用时怎么能获取的接口，而且还能调用到实现类的方法，难道这个接口是在什么时候自动注入进去，且和实现类关联上了。



后来才知道，并没有注入接口的bean，只是注入了实现类serviceImpl的bean，接口只是用来接收的。

* @Autowired是Spring的注解，Autowired默认先按byType，如果发现找到多个bean，则又按照byName方式比对，如果还有多个，则报出异常；
* @Resource 是JDK1.6支持的注解，默认按照名称(Byname)进行装配, 如果没有指定name属性，当注解写在字段上时，默认取字段名，按照名称查找，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

再来说Controller获取实例的过程：使用@Autowired，程序在spring的容器中查找类型是TestService的bean，刚好找到有且只有一个此类型的bean，即testServiceImpl，所以就把testServiceImpl自动装配到了controller的实例testService中，testService其实就是TestServiceImpl实现类；

如果使用的是@Resource，则是先在容器中查找名字为testService的bean，但并没有找到，因为容器中的bean名字是TestServiceImpl(如果@Service没指定bean的value属性，则注入bean的名字就是类名，如果指定了则是指定的名字)，然后再通过类型查找TestService类型的bean，找到唯一的了个TestService类型bean（即TestServiceImpl），所以就自动装配实例成功了。

### 1、注意

byName通过参数名自动装配，如果一个bean的name 和另外一个bean的 property相同，就自动装配。

byType通过参数的数据类型自动自动装配，如果一个bean的数据类型和另外一个bean的property属性的数据类型兼容，就自动装配

效率上来说@Autowired/@Resource差不多，不过推荐使用@Resource一点，因为当接口有多个实现时@Resource直接就能通过name属性来指定实现类，而@Autowired还要结合@Qualifier注解来使用，且@Resource是jdk的注释，可与Spring解耦。

### 2、如果一个接口有多个实现类，怎么通过注解获取实例

如果一个接口有多个实现类时，通过注解获取实例时怎么知道应该获取的是哪一个实现类serviceImpl？

再增加一个TestServiceImple2



各个实现类可以通过以下2种方式来指定具体要使用哪一个实现：

#### 2.1 通过指定bean的名字来明确到底要实例哪一个类

@Autowired需要结合@Qualifier来使用，如下：

@Autowired

@Qualifier("testServiceImpl")

private TestService testService;

@Resource可直接通过指定name属性的值即可。

@Resource(name = "testServiceImpl")

private TestService testService;

@Resource如果不显示的指定name值，就会自动把实例变量的名称作为name的值的，所有也可以直接这样写：

@Resource

private TestService testServiceImpl;

#### 2.2 通过在实现类上添加@Primary注解来指定默认加载类

@Service

@Primary

public class TestServiceImpl2 implements TestService{

@Override

public String test() {

return "TestServiceImpl2";

}

}

这样如果在使用@Autowired/@Resource获取实例时如果不指定bean的名字，就会默认获取TestServiceImpl2的bean，如果指定了bean的名字则以指定的为准。

为什么非要调用接口来多次一举，而不直接调用实现类serviceImpl的bean来得简单明了呢？

* 直接获取实现类serviceImpl的bean也是可以的；
* 至于加一层接口的原因：一是AOP程序设置思想指导，给别人调用的接口，调用者只想知道方法和功能，而对于这个方法内部逻辑怎么实现的并不关心；二是可以降低各个模块间的关联，实现松耦合、程序分层、高扩展性，使程序更加灵活，他除了在规范上有卓越贡献外，最精髓的是在多态上的运用；继承只能单一继承，接口却可以多实现
* 当业务逻辑简单，变更较少，项目自用时，省略掉接口直接使用实现类更简单明了；反之则推荐使用接口;