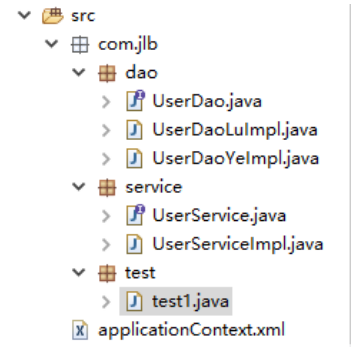
Spring的核心机制是依赖注入。依赖注入让bean与bean之间以配置文件组织在一起，而不是以硬编码的方式耦合在一起。依赖注入(Dependency Injection)和控制反转(Inversion of Control)是同一个概念。具体含义是:当某个角色(可能是一个Java实例，调用者)需要另一个角色(另一个Java实例，被调用者)的协助时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在Spring里，创建被调用者的工作不再由调用者来完成，因此称为控制反转;创建被调用者实例的工作通常由Spring容器来完成，然后注入调用者，因此也称为依赖注入。管是依赖注入，还是控制反转，都说明Spring采用动态、灵活的方式来管理各种对象。对象与对象之间的具体实现互相透明。（直接当的一段大道理，因为讲的很不错，很好理解，就直接用啦，手动俏皮）

当然说再多，都不如直观地使用一下便于理解，接下来我们就进入实战

完整的项目目录



UserDao

```java
public interface UserDao {

    void show();
}
```

两个实现类

```java
public class UserDaoLuImpl implements UserDao {

    @Override
    public void show() {
        // TODO Auto-generated method stub
        System.out.println("你好，鲁鲁");
    }

}
```

```java
public class UserDaoYeImpl implements UserDao{

    @Override
    public void show() {
        // TODO Auto-generated method stub
        System.out.println("你好，叶子");
    }

}
```

UserService

```java
public interface UserService {

    void show();
}
```

实现类

```
public class UserServiceImpl implements UserService {

    private UserDao userDao;

    @Override
    public void show() {
        // TODO Auto-generated method stub
        userDao.show();
    }

    public UserDao getUserDao() {
        return userDao;
    }

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

}
```

可以看到这边定义一个userDao对象，在show方法中调用了这个userDao对象的show方法，但实际上这只是个接口，没有实例化，show方法是空的啊，不急，这里就是要开始依赖注入的魅力了，我们来看看applicationContext.xml文件

```
<bean id="yeDao" class="com.jlb.dao.UserDaoYeImpl" />
<bean id="luDao" class="com.jlb.dao.UserDaoLuImpl"/>
<bean id="userService" class="com.jlb.service.UserServiceImpl">
    <!--  <property name="userDao" ref="yeDao"></property>  -->
    <property name="userDao" ref="luDao"></property>
</bean>
```

可以看到，这里给每个接口实现类都注册一个bean，尤其注意userService的bean，这就依赖注入的方式，子标签property中name对应实现类里定义的对象名，ref指向哪个bean，指向yeDao，相当于使用时实例化为ye，指向luDao，相当于使用时实例化为lu

我们来看看测试方法

```
@Test
public void testSpring2() throws Exception {
    // 读取配置文件
    ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
    UserService userService=(UserService)ctx.getBean("userService");
    userService.show();
}
```

没有什么特色，我们先指向ye

```
你好，叶子
```

再指向lu

```
你好，鲁鲁
```

这样的话，至少初步体会到了依赖注入的方便了吧

当然最主要的还是降低耦合度

像如下这样创建实例，耦合度太高，而且略显繁杂

```
UserDAO userDAO=new  UserDAOYeImpl();
UserDAO userDAO=new  UserDAOLuImpl();
```