@Autowired 可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作。

@Autowired 是根据类型进行自动装配的。

@Qualifier指定注入Bean 的名称。@Qualifier只能和@Autowired 结合使用，是对@Autowired 有益的补充。一般来讲，@Qualifier 对方法签名中入参进行注释会降低代码的可读性，而对成员变量注释则相对好一些。

@Resource（JSR-250 标准注解，推荐使用它来代替Spring 专有的@Autowired注解）

@Resource 的作用相当于@Autowired，只不过@Autowired 按byType 自动注入，而@Resource 默认按byName 自动注入罢了。@Resource 有两个属性是比较重要的，分别是name 和type，Spring 将@Resource 注解的name 属性解析为bean 的名字，而type属性则解析为bean 的类型。所以如果使用name 属性，则使用byName 的自动注入策略，而使用type 属性时则使用byType 自动注入策略。如果既不指定name 也不指定type 属性，这时将通过反射机制使用byName 自动注入策略。

使用@Component 注解定义的Bean，默认的名称（id）是小写开头的非限定类名。也可以指定Bean 的名称：@Component("userDao")

@Component 是所有受Spring 管理组件的通用形式，Spring 还提供了更加细化的注解形式：@Repository、@Service、@Controller，它们分别对应存储层Bean，业务层Bean，和展示层Bean。

<context:component-scan />的base-package 属性指定了需要扫描的类包，类包及其递归子包中所有的类都会被处理。<context:component-scan />还允许定义过滤器将基包下的某些类纳入或排除。

默认情况下，Spring应用上下文中所有bean都是作为一单例(singleton)的形式创建的。即不管给定的一个bean被注入到其他bean多少次，每次所注入的都是同一个实例。

    当所使用的类是异变的(mutable)，他们会保持一些状态，此时重用是不安全的。

    Spring定义了多种作用域，可以基于这些作用域创建bean：

        -- 单例（Singleton）：在整个应用中，只创建bean的一个实例；

        -- 原型（Prototype）：每次注入或通过Spring应用上下文获取的时候，都会创建一个bean实例；

        -- 会话（Session）：在Web应用中，为每个会话创建一个bean实例；

        -- 请求（Request）：在Web应用中，为每个请求创建一个bean实例。

    单例是默认作用域，如果要选择其他作用域，要使用@Scope注解，与@Component或@Bean一起使用：

        -- 原型：@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)或@Scope("prototype")

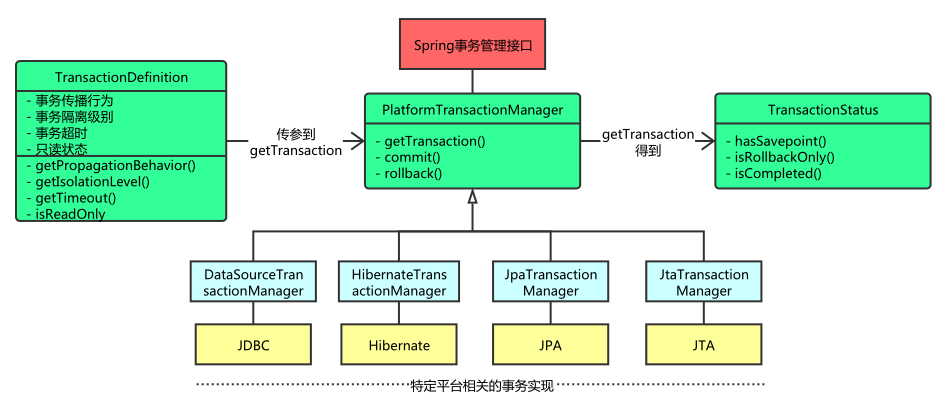
        -- 会话：@Scope(value=WebApplicationContext.SCOPE\_SESSION, proxyMode=ScopedProxyMode.INTERFACES)

        -- 请求：@Scope(value=WebApplicationContext.SCOPE\_REQUEST, proxyMode=ScopedProxyMode.INTERFACES)

BeanFactory和FactoryBean的区别

        BeanFactory是一个工厂类，用于管理Bean的一个工厂。BeanFactory是IOC容器的核心接口，负责实例化，定位，配置应用程序中的对象及建立这些对象间的依赖。

        FactoryBean是实现了FactoryBean<T>接口的Bean。根据该Bean的ID从BeanFactory中获取的实际上是FactoryBean的getObject()返回的对象，而不是FactoryBean本身，若要获取FactoryBean本身需要在id前面加一个&符号。



事务的第一个方面是传播行为（propagation behavior）。当事务方法被另一个事务方法调用时，必须指定事务应该如何传播。例如：方法可能继续在现有事务中运行，也可能开启一个新事务，并在自己的事务中运行。Spring定义了七种传播行为：

| **传播行为** | **含义** |
| --- | --- |
| PROPAGATION\_REQUIRED | 表示当前方法必须运行在事务中。如果当前事务存在，  方法将会在该事务中运行。否则，会启动一个新的事务 |
| PROPAGATION\_SUPPORTS | 表示当前方法不需要事务上下文，但是如果存在当前事务的话，  那么该方法会在这个事务中运行 |
| PROPAGATION\_MANDATORY | 表示该方法必须在事务中运行，如果当前事务不存在，  则会抛出一个异常 |
| PROPAGATION\_REQUIRED\_NEW | 表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。  如果存在当前事务，在该方法执行期间，当前事务会被挂起。  如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NOT\_SUPPORTED | 表示该方法不应该运行在事务中。如果存在当前事务，  在该方法运行期间，当前事务将被挂起。  如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NEVER | 表示当前方法不应该运行在事务上下文中。  如果当前正有一个事务在运行，则会抛出异常 |
| PROPAGATION\_NESTED | 表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。  嵌套的事务可以独立于当前事务进行单独地提交或回滚。  如果当前事务不存在，那么其行为与PROPAGATION\_REQUIRED  一样。注意各厂商对这种传播行为的支持是有所差异的。  可以参考资源管理器的文档来确认它们是否支持嵌套事务 |