# Spring

Spring的核心是控制反转（[IoC](http://baike.baidu.com/view/146665.htm" \t "http://baike.baidu.com/_blank)）和面向切面（AOP）优点：使用IoC容器更加容易组合对象之间关系，面向接口编程，降低耦合，

IoC(Inversion of Control)控制反转，对象创建责任的反转，在spring中BeanFacotory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，具体实现类的具体对象的创建，不依赖于调用他的客户端代码，而将控制权交给spring（其实就是xml文件），这就是控制反转。

依赖注入： 侧重于过程， 把对象通过setter、contruct、args等方式 注入到另一个对象中作为这个对象的一个成员变量（也可能是其他）；DI是一种IoC  
控制反转： 侧重于结果，说的是对象的产生不是通过直接new 的，而是通过依赖注入的方式的。

1，谁依赖谁；2，谁注入，注入什么？

1.客户端代码中，具体对象的创建依赖于spring，即IOC容器；2.是IOC容器注入，在运行期，根据xml的配置信息，将具体的对象注入到相应的bean中。

**Aop面向切面编程：业务逻辑和系统服务隔离开来，把项目中需要在多处用到的功能，比如日志、安全和事物等集中到一个类中处理，而不用在每个需要用到该功能的地方显式调用。**

术语解释：

       横切关注点：分布应用于多处的功能

       切面：横切关注点可以被模块化为一个类，这个类被称为一个切面

       通知（advice）：切面要完成的工作。Spring的通知有5种类型：before、after、after-returning、after-throwing和around这五种类型。

       连接点（joinpoint）：连接点表示在何种操作发生时应用切面。比如方法调用时、修改字段时和抛出异常时等等

       切点（pointcut）：一般用某个包中的类的明确的方法来指定在何处应用切面，应用切面的这个点便称为切点，一般用切点来指定连接点。

* 切面（Aspect）： 一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是J2EE应用中一个关于横切关注点的很好的例子。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @Aspect 注解（@AspectJ风格）来实现。
* 连接点（Joinpoint）： 在程序执行过程中某个特定的点，比如某方法调用的时候或者处理异常的时候。 在Spring AOP中，一个连接点总是代表一个方法的执行。 通过声明一个org.aspectj.lang.JoinPoint类型的参数可以使通知（Advice）的主体部分获得连接点信息。
* 通知（Advice）： 在切面的某个特定的连接点（Joi
* npoint）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。 通知的类型将在后面部分进行讨论。许多AOP框架，包括Spring，都是以拦截器做通知模型， 并维护一个以连接点为中心的拦截器链。
* 切点（Pointcut）： 匹配连接点（Joinpoint）的断言。通知和一个切入点表达式关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。 切入点表达式如何和连接点匹配是AOP的核心：Spring缺省使用AspectJ切入点语法。
* 引入（Introduction）： （也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。 Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。 例如，你可以使用一个引入来使bean实现 IsModified 接口，以便简化缓存机制。
* 目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（advised） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个被代理（proxied）对象。
* **AOP代理（AOP Proxy）：** AOP框架创建的对象，用来实现切面契约（aspect contract）（包括通知方法执行等功能）。 **在Spring中，AOP代理可以是JDK动态代理或者CGLIB代理。** 注意：Spring 2.0最新引入的基于模式（schema-based）风格和@AspectJ注解风格的切面声明，对于使用这些风格的用户来说，代理的创建是透明的。
* **织入（Weaving）：** 把切面（aspect）连接到其它的应用程序类型或者对象上，并创建一个被通知（advised）的对象。 这些可以在编译时（例如使用AspectJ编译器），类加载时和运行时完成。 **Spring和其他纯Java AOP框架一样，在运行时完成织入。**

**二、通知的类型**

* 前置通知（Before advice）： 在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。
* 返回后通知（After returning advice）： 在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。
* 抛出异常后通知（After throwing advice）： 在方法抛出异常退出时执行的通知。
* 后通知（After (finally) advice）： 当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。
* 环绕通知（Around Advice）： 包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。

增加代码可重用性，利用一种称为"横切"的技术，能够剖解开封装的对象内部，并将那些影响了多个类并且与具体业务无关的公共行为 封装成一个独立的模块（称  
为切面）。更重要的是，它又能以巧夺天功的妙手将这些剖开的切面复原，不留痕迹的融入核心业务逻辑中。例如权限验证(validate),业务处理(Action),日志记录(log4)；通过动态代理技术或者是在程序编译期间进行静态的"织入"方式（要完成代码通过 AspectJ 编译时织入，通常需要两步：

编写 aspect 文件；

使用 ajc 编译器结合 aspect 文件对源代码进行编译。

）。

**AOP有三种织入切面的方法：其一是编译期织入，这要求使用特殊的Java编译器，AspectJ是其中的代表者；其二是类装载期织入，而这要求使用特殊的类装载器，AspectJ和AspectWerkz是其中的代表者；其三为动态代理织入，在运行期为目标类添加增强生成子类的方式，Spring AOP采用动态代理织入切面。**

**Spring AOP使用了两种代理机制，一种是基于JDK的动态代理，另一种是基于CGLib的动态代理**

Spring整合Structs

Spring是完整的一站式框架，而Struts仅是MVC框架，且着重于MVC中的C。Spring有三种方式整合Struts：使用 Spring 的 ActionSupport 类整合 Struts；使用 Spring 的 DelegatingRequestProcessor 覆盖 Struts 的 RequestProcessor；将 Struts Action 管理委托给 Spring 框架，动作委托最好。

JDK动态代理和cglib动态代理

动态代理类的字节码在程序运行时由Java反射机制动态生成，无需程序员手工编写它的源代码。动态代理类不仅简化了编程工作，而且提高了软件系统的可扩展性，因为Java 反射机制可以生成任意类型的动态代理类，缺点，代理的类需要实现接口。

1. public class BookFacadeProxy implements InvocationHandler {
2. private Object target;
3. /\*\*
4. \* 绑定委托对象并返回一个代理类
5. \* @param target
6. \* @return
7. \*/
8. public Object bind(Object target) {
9. this.target = target;
10. //取得代理对象
11. return Proxy.newProxyInstance(target.getClass().getClassLoader(),
12. target.getClass().getInterfaces(), this);   //要绑定接口(这是一个缺陷，cglib弥补了这一缺陷)
13. }
15. @Override
16. /\*\*
17. \* 调用方法
18. \*/
19. public Object invoke(Object proxy, Method method, Object[] args)
20. throws Throwable {
21. Object result=null;
22. System.out.println("事物开始");
23. //执行方法
24. result=method.invoke(target, args);
25. System.out.println("事物结束");
26. return result;
27. }
29. }

**cglib是针对类来实现代理的，他的原理是对指定的目标类生成一个子类，并覆盖其中方法实现增强，但因为采用的是继承，所以不能对final修饰的类进行代理**

**执行效率比jdk动态代理要高,但创建实例时没有前者快**

1. **public** **class** BookFacadeCglib **implements** MethodInterceptor {
2. **private** Object target;
4. /\*\*
5. \* 创建代理对象
6. \*
7. \* @param target
8. \* @return
9. \*/
10. **public** Object getInstance(Object target) {
11. **this**.target = target;
12. Enhancer enhancer = **new** Enhancer();
13. enhancer.setSuperclass(**this**.target.getClass());
14. // 回调方法
15. enhancer.setCallback(**this**);
16. // 创建代理对象
17. **return** enhancer.create();
18. }
20. @Override
21. // 回调方法
22. **public** Object intercept(Object obj, Method method, Object[] args,
23. MethodProxy proxy) **throws** Throwable {
24. System.out.println("事物开始");
25. proxy.invokeSuper(obj, args);
26. System.out.println("事物结束");
27. **return** **null**;

30. }
32. }

S**pring依赖注入（DI）的三种方式**，分别为：

1．  接口注入：接口注入需要额外声明一个接口，增加了类的数目，而且它的效果和属性注入并无本质区别，因此我们不提倡这种方式

2．  Setter方法注入：如果依赖关系较为复杂，那么构造子注入模式的构造函数也会相当庞大，而此时设值注入模式则更为简洁。如果用到了第三方类库，可能要求我们的组件提供一个默认的构造函数，此时构造子注入模式也不适用。

3．  构造方法注入：在构造期间完成一个完整的、合法的对象。 所有依赖关系在构造函数中集中呈现。 只有组件的创建者关心其内部依赖关系，对调用者而言，该依赖关系处于“黑盒”之中。

**最好的选择是使用构造器参数实现强制依赖注入，使用setter方法实现可选的依赖关系**

1、设值注入不会重写构造方法的值。如果我们对同一个变量同时使用了构造方法注入又使用了设置方法注入的话，那么构造方法将不能覆盖由设值方法注入的值。很明显，因为构造方法尽在对象被创建时调用。

2、在使用设值注入时有可能还不能保证某种依赖是否已经被注入，也就是说这时对象的依赖关系有可能是不完整的。而在另一种情况下，构造器注入则不允许生成依赖关系不完整的对象。

**传播行为分为六种，和事务隔离级别**PROPAGATION\_REQUIRED–支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。  
PROPAGATION\_SUPPORTS–支持当前事务，如果当前没有事务，就以非事务方式执行。  
PROPAGATION\_MANDATORY–支持当前事务，如果当前没有事务，就抛出异常。  
PROPAGATION\_REQUIRES\_NEW–新建事务，如果当前存在事务，把当前事务挂起。  
PROPAGATION\_NOT\_SUPPORTED–以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。  
PROPAGATION\_NEVER–以非事务方式执行，如果当前存在事务，则抛出异常。

隔离级别：  
ISOLATION\_DEFAULT 这是一个PlatfromTransactionManager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，它允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，不可重复读和幻读。  
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)。  
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。

锁定正在读取的行解决脏读，锁定所读取的所有行解决不可重复读，锁表解决幻象读。

spring提供的事务管理可以分为两类：编程式的和声明式的。

编程式的，比较灵活，但是代码量大，存在重复的代码比较多；自己写Try catch语句调用commit\rollback等函数来实现那段业务数据改变的事务性，需要自己写大量重复的事务控制代码。

声明式的比编程式的更灵活.通过框架和事务定义配置给自己的业务代码添加事务性，比如Spring里面的AOP，设置可以一次性给所有的业务方法添加上事务特性。

SpringSpring配置文件中关于事务配置总是由三个组成部分，分别是DataSource、TransactionManager和代理机制这三部分，无论哪种配置方式，一般变化的只是代理机制这部分。 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。Spring事务管理器的接口org.springframework.transaction.PlatformTransactionManager，通过这个接口，Spring为各个平台如JDBC、Hibernate等都提供了对应的事务管理器。还有DataSourceTransactionManager.

配置事务的三种方式：

1. 使用tx标签方式
2. 使用代理方式，然后将具体的Dao的parent属性设置为"transactionProxy"，
3. 使用拦截器

有三种方式向Spring 容器提供元数据:

1、XML配置文件

2、基于注解配置

3、基于Java的配置

**你如何定义bean的作用域？**

在Spring中创建一个bean的时候，我们可以声明它的作用域。只需要在bean定义的时候通过’scope’属性定义即可。例如，当Spring需要产生每次一个新的bean实例时，应该声明bean的scope属性为**prototype**（调用者管理该bean的生命周期）。如果每次你希望Spring返回一个实例，应该声明bean的scope属性为singleton（默认为**singleton**）（ 则将该bean放入springIOC的缓存池中，将触发spring对该bean的生命周期管理）。属性还有**‍request**表示该针对每一次HTTP请求都会产生一个新的bean，同时该bean仅在当前HTTP request内有效；**‍session**作用域表示该针对每一次HTTP请求都会产生一个新的bean，同时该bean仅在当前HTTP session内有效 ；**global session**作用域类似于标准的HTTP Session作用域，该作用域同样仅适用于WebApplicationContext环境

简要说明Spring什么时候实例化bean，首先要分2种情况  
  第一：如果你使用BeanFactory作为Spring Bean的工厂类，则所有的bean都是在第一次使用该Bean的时候实例化  
  第二：如果你使用ApplicationContext作为Spring Bean的工厂类，则又分为以下几种情况：  
       （1）：如果bean的scope是singleton的，并且lazy-init为false（默认是false，所以可以不用设置），则 /启动的时候就实例化该Bean，并且将实例化的Bean放在一个map结构的缓存中，下次再使用该Bean的时候， 直接从这个缓存中取  
       （2）：如果bean的scope是singleton的，并且lazy-init为true，则该Bean的实例化是在第一次使用该Bean的时候进行实例化  
       （3）：如果bean的scope是prototype的，则该Bean的实例化是在第一次使用该Bean的时候进行实例化

与BeanFactory通常以编程的方式被创建不同的是，ApplicationContext能以声明的方式创建，如使用ContextLoader

无状态的Bean适合用不变模式，技术就是单例模式，这样可以共享实例，提高性能。有状态的Bean，多线程环境下不安全，那么适合用Prototype原型模式。Prototype: 每次对bean的请求都会创建一个新的bean实例。

Struts2默认的实现是Prototype模式。也就是每个请求都新生成一个Action实例，所以不存在线程安全问题。需要注意的是，如果由Spring管理action的生命周期， scope要配成prototype作用域

解决单例不安全方法：1、需要的时候创建新实例；2、使用同步；3、使用ThreadLocal；

**Bean生命周期：**

**161、阐述Spring框架中Bean的生命周期？**

**答：**

**① Spring IoC容器找到关于Bean的定义并实例化该Bean。**

**② Spring IoC容器对Bean进行依赖注入。**

**③ 如果Bean实现了BeanNameAware接口，则将该Bean的id传给setBeanName方法。**

**④ 如果Bean实现了BeanFactoryAware接口，则将BeanFactory对象传给setBeanFactory方法。**

**⑤ 如果Bean实现了BeanPostProcessor接口，则调用其postProcessBeforeInitialization方法。**

**⑥ 如果Bean实现了InitializingBean接口，则调用其afterPropertySet方法。**

**⑦ 如果有和Bean关联的BeanPostProcessors对象，则这些对象的postProcessAfterInitialization方法被调用。**

**⑧ 当销毁Bean实例时，如果Bean实现了DisposableBean接口，则调用其destroy方法。**

Bean的完整生命周期从 spring 容器开始实例化 bean 开始，到销毁。可以从三点来理解

1、 bean自身的方法：包括构造方法、 set 方法、 init-method 指定的方法、 destroy-method 指定的方法

2、 Bean级生命周期接口方法：如 BeanNameAware 、 BeanFactoryAware 等这些接口方法由 bean类实现。

3、 容器级生命周期接口方法：上图中带星的。有InstantiationAwareBeanPostProcessor 、 BeanPostProcessor 等。一般称为后处理 器。他们一般不由bean 本身实现，独立存在，注册到 spring 容器中。 Spring 通过接口反射预先知道，当 spring 容器创建任何 bean 时，这些后处理器都会发生作用。所以他们是全局的，用户可以通过编码对只感兴趣的 bean 进行处理。

### Bean级生命周期接口解决bean的个性化处理问题，容器级生命周期接口解决容器中某些 bean 共性化处理问题

**Spring 提供了以下5中标准的事件：**

上下文更新事件（ContextRefreshedEvent）：该事件会在ApplicationContext被初始化或者更新时发布。也可以在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。

上下文开始事件（ContextStartedEvent）：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。

上下文停止事件（ContextStoppedEvent）：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。

上下文关闭事件（ContextClosedEvent）：当ApplicationContext被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。

请求处理事件（RequestHandledEvent）：在Web应用中，当一个http请求（request）结束触发该事件。

**Spring中注入Java集合类**

list元素用来注入一系列的值，允许有相同的值。

set元素用来注入一些列的值，不允许有相同的值。

map用来注入一组”键-值”对，键、值可以是任何类型的。

props也可以用来注入一组”键-值”对，这里的键、值都字符串类型

**自动装配提供五种不同的模式供Spring容器用来自动装配beans之间的依赖注入**

no：默认的方式是不进行自动装配，通过手工设置ref 属性来进行装配bean。

byName：通过参数名自动装配，Spring容器查找beans的属性

byType：通过参数数据类型自动自动装配，如果有多个bean符合条件，则抛出错误。

constructor：这个同byType类似,不过是应用于构造函数的参数。如果在BeanFactory中不是恰好有一个bean与构造函数参数相同类型，则抛出一个严重的错误。

autodetect：如果有默认的构造方法，通过 construct的方式自动装配，否则使用 byType的方式自动装配。

**Spring注解**

<context:annotation-config />表示开启注解

<context:component-scan base-package="cn.gacl.dao.impl,cn.gacl.service.impl,cn.gacl.action"/>

表明cn.gacl.dao.impl等多个包及其子包中，如果某个类的头上带有特定的注解【@Component/@Repository/@Service/@Controller】（代替Bean在 XML文件中通过<bean> 进行定义）注解bean的，不同于@Autowired注解bean里面的东西，就会将这个对象作为Bean注册进Spring容器。

@Component是所有受Spring 管理组件的通用形式，不推荐使用。

@Controller对应表现层的Bean【@Controller(value="UserAction")

@Service对应的是业务层Bean

@Repository对应数据访问层Bean

@Autowired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（在Bean类中使用自动注入功能），默认按类型装配，@Resource默认按名称装配，当找不到名称匹配的bean才会按类型装配（推荐使用）。

使用@Qualifier配合@Autowired来解决找不到对应bean或者多个对应bean问题

**152、Spring中如何使用注解来配置Bean？有哪些相关的注解？**

答：首先需要在Spring配置文件中增加如下配置：

<context:component-scan base-package="org.example"/>

然后可以用@Component、@Controller、@Service、@Repository注解来标注需要由Spring IoC容器进行对象托管的类。这几个注解没有本质区别，只不过@Controller通常用于控制器，@Service通常用于业务逻辑类，@Repository通常用于仓储类（例如我们的DAO实现类），普通的类用@Component来标注。

织入是将切面和其他应用类型或对象连接起来创建一个通知对象的过程。织入可以在编译、加载或运行时完成。

**Servlet何时被创建：**

　　1，默认情况下，当WEB客户第一次请求访问某个Servlet的时候，WEB容器将创建这个Servlet的实例。

　　2，当web.xml文件中如果<servlet>元素中指定了<load-on-startup>子元素时，Servlet容器在启动web服务器时，将按照顺序创建并初始化Servlet对象。

**Spring的ApplicationContext容器 最常被使用的 ApplicationContext 接口实现：**

\* FileSystemXmlApplicationContext：该容器从 XML 文件中加载已被定义的bean。在这里，你需要提供给构造器 XML 文件的完整路径.

\* ClassPathXmlApplicationContext：该容器从XML 文件中加载已被定义的 bean。在这里，你不需要提供 XML文件的完整路径， 只需正确配置 CLASSPATH 环境变量即可，因为，容器会从 CLASSPATH 中搜索 bean 配置文件。

\* WebXmlApplicationContext：该容器会在一个 web 应用程序的范围内加载在 XML 文件中已被定义的 bean。

**153、Spring支持的事务管理类型有哪些？你在项目中使用哪种方式？**答：**Spring支持编程式事务管理和声明式事务管理**。许多Spring框架的用户选择声明式事务管理，因为这种方式和应用程序的关联较少，因此更加符合轻量级容器的概念。声明式事务管理要优于编程式事务管理，尽管在灵活性方面它弱于编程式事务管理，因为编程式事务允许你通过代码控制业务。

**事务分为全局事务和局部事务。**全局事务由应用服务器管理，需要底层服务器JTA支持（如WebLogic、WildFly等）。局部事务和底层采用的持久化方案有关，例如使用JDBC进行持久化时，需要使用Connetion对象来操作事务；而采用Hibernate进行持久化时，需要使用Session对象来操作事务。

Spring提供了如下所示的事务管理器。

| 事务管理器实现类 | 目标对象 |
| --- | --- |
| DataSourceTransactionManager | 注入DataSource |
| HibernateTransactionManager | 注入SessionFactory |
| JdoTransactionManager | 管理JDO事务 |
| JtaTransactionManager | 使用JTA管理事务 |
| PersistenceBrokerTransactionManager | 管理Apache的OJB事务 |

**这些事务的父接口都是PlatformTransactionManager。**Spring的事务管理机制是一种典型的策略模式，PlatformTransactionManager代表事务管理接口，该接口定义了三个方法，该接口并不知道底层如何管理事务，但是它的实现类必须提供getTransaction()方法（开启事务）、commit()方法（提交事务）、rollback()方法（回滚事务）的多态实现，这样就可以用不同的实现类代表不同的事务管理策略。使用JTA全局事务策略时，需要底层应用服务器支持，而不同的应用服务器所提供的JTA全局事务可能存在细节上的差异，因此实际配置全局事务管理器是可能需要使用JtaTransactionManager的子类，如：WebLogicJtaTransactionManager（Oracle的WebLogic服务器提供）、UowJtaTransactionManager（IBM的WebSphere服务器提供）等。

**154、如何在Web项目中配置Spring的IoC容器？**

答：如果需要在Web项目中使用Spring的IoC容器，可以在Web项目配置文件web.xml中做出如下配置：

<context-param>

    <param-name>contextConfigLocation</param-name>

    <param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

    <listener-class>

        org.springframework.web.context.ContextLoaderListener

    </listener-class>

</listener>

**155、如何在Web项目中配置Spring MVC？**

答：要使用Spring MVC需要在Web项目配置文件中配置其前端控制器DispatcherServlet，如下所示：

<web-app>

    <servlet>

        <servlet-name>example</servlet-name>

        <servlet-class>

            org.springframework.web.servlet.DispatcherServlet

        </servlet-class>

        <load-on-startup>1</load-on-startup>

    </servlet>

    <servlet-mapping>

        <servlet-name>example</servlet-name>

        <url-pattern>\*.html</url-pattern>

    </servlet-mapping>

</web-app>

说明：上面的配置中使用了\*.html的后缀映射，这样做一方面不能够通过URL推断采用了何种服务器端的技术，另一方面可以欺骗搜索引擎，因为搜索引擎不会搜索动态页面，这种做法称为伪静态化。

**160、Spring IoC容器配置Bean的方式？**

答：

- 基于XML文件进行配置。

- 基于注解进行配置。

- 基于Java程序进行配置（Spring 3+）

**162、依赖注入时如何注入集合属性？**

答：可以在定义Bean属性时，通过<list> / <set> / <map> / <props>分别为其注入列表、集合、映射和键值都是字符串的映射属性。

**163、Spring中的自动装配有哪些限制？**

答：

- 如果使用了构造器注入或者setter注入，那么将覆盖自动装配的依赖关系。

- 基本数据类型的值、字符串字面量、类字面量无法使用自动装配来注入。

- 优先考虑使用显式的装配来进行更精确的依赖注入而不是使用自动装配。

**164、在Web项目中如何获得Spring的IoC容器？**

答：

WebApplicationContext ctx =

WebApplicationContextUtils.getWebApplicationContext(servletContext);

**165. 大型网站在架构上应当考虑哪些问题？**

答：

- **分层**：分层是处理任何复杂系统最常见的手段之一，将系统横向切分成若干个层面，每个层面只承担单一的职责，然后通过下层为上层提供的基础设施和服务以及上层对下层的调用来形成一个完整的复杂的系统。计算机网络的开放系统互联参考模型（OSI/RM）和Internet的TCP/IP模型都是分层结构，大型网站的软件系统也可以使用分层的理念将其分为持久层（提供数据存储和访问服务）、业务层（处理业务逻辑，系统中最核心的部分）和表示层（系统交互、视图展示）。需要指出的是：（1）分层是逻辑上的划分，在物理上可以位于同一设备上也可以在不同的设备上部署不同的功能模块，这样可以使用更多的计算资源来应对用户的并发访问；（2）层与层之间应当有清晰的边界，这样分层才有意义，才更利于软件的开发和维护。

- **分割**：分割是对软件的纵向切分。我们可以将大型网站的不同功能和服务分割开，形成高内聚低耦合的功能模块（单元）。在设计初期可以做一个粗粒度的分割，将网站分割为若干个功能模块，后期还可以进一步对每个模块进行细粒度的分割，这样一方面有助于软件的开发和维护，另一方面有助于分布式的部署，提供网站的并发处理能力和功能的扩展。

- **分布式：**除了上面提到的内容，网站的静态资源（JavaScript、CSS、图片等）也可以采用独立分布式部署并采用独立的域名，这样可以减轻应用服务器的负载压力，也使得浏览器对资源的加载更快。数据的存取也应该是分布式的，传统的商业级关系型数据库产品基本上都支持分布式部署，而新生的NoSQL产品几乎都是分布式的。当然，网站后台的业务处理也要使用分布式技术，例如查询索引的构建、数据分析等，这些业务计算规模庞大，可以使用Hadoop以及MapReduce分布式计算框架来处理。

- **集群**：集群使得有更多的服务器提供相同的服务，可以更好的提供对并发的支持。

- **缓存**：所谓缓存就是用空间换取时间的技术，将数据尽可能放在距离计算最近的位置。使用缓存是网站优化的第一定律。我们通常说的CDN、反向代理、热点数据都是对缓存技术的使用。

- **异步**：异步是实现软件实体之间解耦合的又一重要手段。异步架构是典型的生产者消费者模式，二者之间没有直接的调用关系，只要保持数据结构不变，彼此功能实现可以随意变化而不互相影响，这对网站的扩展非常有利。使用异步处理还可以提高系统可用性，加快网站的响应速度（用Ajax加载数据就是一种异步技术），同时还可以起到削峰作用（应对瞬时高并发）。&quot；能推迟处理的都要推迟处理”是网站优化的第二定律，而异步是践行网站优化第二定律的重要手段。

- **冗余**：各种服务器都要提供相应的冗余服务器以便在某台或某些服务器宕机时还能保证网站可以正常工作，同时也提供了灾难恢复的可能性。冗余是网站高可用性的重要保证。

**168、什么是XSS攻击？什么是SQL注入攻击？什么是CSRF攻击？**

答：

- **XSS（Cross Site Script，跨站脚本攻击）**是向网页中注入恶意脚本在用户浏览网页时在用户浏览器中执行恶意脚本的攻击方式。跨站脚本攻击分有两种形式：反射型攻击（诱使用户点击一个嵌入恶意脚本的链接以达到攻击的目标，目前有很多攻击者利用论坛、微博发布含有恶意脚本的URL就属于这种方式）和持久型攻击（将恶意脚本提交到被攻击网站的数据库中，用户浏览网页时，恶意脚本从数据库中被加载到页面执行，QQ邮箱的早期版本就曾经被利用作为持久型跨站脚本攻击的平台）。XSS虽然不是什么新鲜玩意，但是攻击的手法却不断翻新，防范XSS主要有两方面：消毒（对危险字符进行转义）和HttpOnly（防范XSS攻击者窃取Cookie数据）。

- **SQL注入攻击**是注入攻击最常见的形式（此外还有OS注入攻击（Struts 2的高危漏洞就是通过OGNL实施OS注入攻击导致的）），当服务器使用请求参数构造SQL语句时，恶意的SQL被嵌入到SQL中交给数据库执行。SQL注入攻击需要攻击者对数据库结构有所了解才能进行，攻击者想要获得表结构有多种方式：（1）如果使用开源系统搭建网站，数据库结构也是公开的（目前有很多现成的系统可以直接搭建论坛，电商网站，虽然方便快捷但是风险是必须要认真评估的）；（2）错误回显（如果将服务器的错误信息直接显示在页面上，攻击者可以通过非法参数引发页面错误从而通过错误信息了解数据库结构，Web应用应当设置友好的错误页，一方面符合最小惊讶原则，一方面屏蔽掉可能给系统带来危险的错误回显信息）；（3）盲注。防范SQL注入攻击也可以采用消毒的方式，通过正则表达式对请求参数进行验证，此外，参数绑定也是很好的手段，这样恶意的SQL会被当做SQL的参数而不是命令被执行，JDBC中的PreparedStatement就是支持参数绑定的语句对象，从性能和安全性上都明显优于Statement。

例子：

某个网站的登录验证的SQL查询代码为：

strSQL = "SELECT \* FROM users WHERE (name = '" + userName + "') and (pw = '"+ passWord +"');"

恶意填入

userName = "1' OR '1'='1";

与

passWord = "1' OR '1'='1";

时，将导致原本的SQL字符串被填为

strSQL = "SELECT \* FROM users WHERE (name = '1' OR '1'='1') and (pw = '1' OR '1'='1');"

也就是实际上运行的SQL命令会变成下面这样的

strSQL = "SELECT \* FROM users;"

因此达到无账号密码，亦可登录网站。所以SQL注入攻击被俗称为黑客的填空游戏。

- **CSRF攻击（Cross Site Request Forgery，跨站请求伪造）**是攻击者通过跨站请求，以合法的用户身份进行非法操作（如转账或发帖等）。CSRF的原理是利用浏览器的Cookie或服务器的Session，盗取用户身份，其原理如下图所示。防范CSRF的主要手段是识别请求者的身份，主要有以下几种方式：（1）在表单中添加令牌（token）；（2）验证码；（3）检查请求头中的Referer（前面提到防图片盗链接也是用的这种方式）。令牌和验证都具有一次消费性的特征，因此在原理上一致的，但是验证码是一种糟糕的用户体验，不是必要的情况下不要轻易使用验证码，目前很多网站的做法是如果在短时间内多次提交一个表单未获得成功后才要求提供验证码，这样会获得较好的用户体验。

**<context:component-scan>包含<context:annotation-config>**.通常在使用前者后，不再使用后者。而AutowiredAnnotationBeanPostProcessor和CommonAnnotationBeanPostPorcessor也会被包含进来

区别：前者可以扫描基于类的注解，而后者只能在bean注解后完成方法、成员变量的注解。

@Autowired是由Spring BeanPostProcessor处理的，所以不能在自己的BeanPostProcessor或BeanFactoryPostProcessor类型应用这些注解，这些类型必须通过XML或者Spring 的@Bean注解加载。

@Qualifier注解：

按类型自动装配可能多个bean实例的情况，可以使用Spring的@Qualifier注解缩小范围（或指定唯一），也可以用于指定单独的构造器参数或方法参数。

还可以用于注解集合类型变量。

基于java的容器注解：

@Bean标识一个用于配置和初始化一个由SpringIOC容器管理的新对象的方法，类似与XML配置文件的<bean/>

@Bean(name=””,initMethod=””,destoryMethod=””)

**可以在Spring的@Cpmponent注解的类中使用@Bean注解任何方法**（仅仅是可以）

**上一点中，通常使用的是@Configuraion**

@Configuration

Public class AppConfig{

@Bean

@Scope(value=”prototype”,proxyMode=ScopedProxyMode.TARGET\_CLASS)

Public MyService myService（）{

Return new MyServiceImpl();

}

}

上面和在XML中定义一个<bean/>是等效的

<context:property-placeholder location=”classpath:com/acme/jdbc.properties”/>加载文件以便后面引用其中的键值对

注解的方式：

在类中只用：

@Configuration

@ImportResource(“ ”)

在成员变量中使用@Value（“${key}”） 注解

@Resource

Spring还支持使用JSR-250@Resource注解的变量或setter方法，这是一种在JAVA EE 5或6的通用模式，Spring管理的对象也支持这种模式

@Resource有一个name属性，并且默认Spring解释该值作为被注入的bean的名称。

AOP实现方式：1.预编译（AspectJ）.2. 运行期动态代理（JDK动态代理、CGLib动态代理）SpringAOP、JbossAOP

## **Spring 概述**

### **1. 什么是spring?**

Spring 是个java企业级应用的开源开发框架。Spring主要用来开发Java应用，但是有些扩展是针对构建J2EE平台的web应用。**Spring 框架目标是简化Java企业级应用开发，并通过POJO为基础的编程模型促进良好的编程习惯。**

POJO 和JavaBean是我们常见的两个关键字，一般容易混淆，POJO全称是Plain Ordinary Java Object / Pure Old Java Object，中文可以翻译成：普通Java类，具有一部分getter/setter方法的那种类就可以称作POJO，但是JavaBean则比 POJO复杂很多， Java Bean 是可复用的组件，对 Java Bean 并没有严格的规范，理论上讲，任何一个 Java 类都可以是一个 Bean 。但通常情况下，由于 Java Bean 是被容器所创建（如 Tomcat) 的，所以 Java Bean 应具有一个无参的构造器，另外，通常 Java Bean 还要实现 Serializable 接口用于实现 Bean 的持久性。 Java Bean 是不能被跨进程访问的

JavaBean是一种组件技术，就好像你做了一个扳子，而这个扳子会在很多地方被拿去用，这个扳子也提供多种功能(你可以拿这个扳子扳、锤、撬等等)，而这个扳子就是一个组件。

一般在web应用程序中建立一个**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/qq635785620/article/details/_blank)**的映射对象时，我们只能称它为POJO。

POJO(Plain Old Java Object)这个名字用来强调它是一个普通java对象，而不是一个特殊的对象。

### **2. 使用Spring框架的好处是什么？**

* **轻量：**Spring 是轻量的，基本的版本大约2MB。
* **控制反转：**Spring通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。
* **面向切面的编程(AOP)：**Spring支持面向切面的编程，并且把**应用业务逻辑和系统服务分开。**
* **容器：**Spring 包含并管理应用中对象的生命周期和配置。
* **MVC框架**：Spring的WEB框架是个精心设计的框架，是Web框架的一个很好的替代品。
* **事务管理：**Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。
* **异常处理：**Spring 提供方便的API把具体技术相关的异常（比如由JDBC，Hibernate or JDO抛出的）转化为一致的unchecked 异常。

### **Spring由哪些模块组成?**



以下是Spring 框架的基本模块：

* Core module
* Bean module
* Context module
* Expression Language module
* JDBC module
* ORM module
* OXM module
* Java Messaging Service(JMS) module
* Transaction module
* Web module
* Web-Servlet module
* Web-Struts module
* Web-Portlet module

### **4. 核心容器（应用上下文) 模块。**

这是基本的Spring模块，提供spring 框架的基础功能，BeanFactory 是 任何以spring为基础的应用的核心。Spring 框架建立在此模块之上，它使Spring成为一个容器。

### **5. BeanFactory – BeanFactory 实现举例。**

Bean 工厂是工厂模式的一个实现，提供了控制反转功能，用来把应用的配置和依赖从正真的应用代码中分离。

最常用的BeanFactory 实现是XmlBeanFactory 类。

### **6. XMLBeanFactory**

最常用的就是org.springframework.beans.factory.xml.XmlBeanFactory ，它根据XML文件中的定义加载beans。该容器从XML 文件读取配置元数据并用它去创建一个完全配置的系统或应用。

### **7. 解释AOP模块**

AOP模块用于发给我们的Spring应用做面向切面的开发， 很多支持由AOP联盟提供，这样就确保了Spring和其他AOP框架的共通性。这个模块将元数据编程引入Spring。

### **8. 解释JDBC抽象和DAO模块。**

通过使用JDBC抽象和DAO模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用Spring的AOP 模块给Spring应用中的对象提供事务管理服务。

### **9. 解释对象/关系映射集成模块。**

Spring 通过提供ORM模块，支持我们在直接JDBC之上使用一个对象/关系映射映射(ORM)工具，Spring 支持集成主流的ORM框架，如Hiberate,JDO和 iBATIS SQL Maps。Spring的事务管理同样支持以上所有ORM框架及JDBC。

### **10.  解释WEB 模块。**

Spring的WEB模块是构建在application context 模块基础之上，提供一个适合web应用的上下文。这个模块也包括支持多种面向web的任务，如透明地处理多个文件上传请求和程序级请求参数的绑定到你的业务对象。它也有对Jakarta Struts的支持。

### **12.  Spring配置文件**

Spring配置文件是个XML 文件，这个文件包含了类信息，描述了如何配置它们，以及如何相互调用。

### **13.  什么是Spring IOC 容器？**

Spring IOC 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。

### **14.  IOC的优点是什么？**

IOC 或 依赖注入把应用的代码量降到最低。它使应用容易测试，单元测试不再需要单例和JNDI查找机制。最小的代价和最小的侵入性使松散耦合得以实现。IOC容器支持加载服务时的饿汉式初始化和懒加载。

### **15. ApplicationContext通常的实现是什么?**

* FileSystemXmlApplicationContext ：此容器从一个XML文件中加载beans的定义，XML Bean 配置文件的全路径名必须提供给它的构造函数。
* ClassPathXmlApplicationContext：此容器也从一个XML文件中加载beans的定义，这里，你需要正确设置classpath因为这个容器将在classpath里找bean配置。
* WebXmlApplicationContext：此容器加载一个XML文件，此文件定义了一个WEB应用的所有bean。

### **Bean 工厂和 Application contexts  有什么区别？**

* BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory 包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。
* BeanFactory还能在实例化对象的时生成协作类之间的关系。此举将bean自身与bean客户端的配置中解放出来。BeanFactory还包含了bean生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。
* 从表面上看，application context如同bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但application context在此基础上还提供了其他的功能。
* 提供了支持国际化的文本消息
* 统一的资源文件读取方式
* 已在监听器中注册的bean的事件
* 以下是三种较常见的 ApplicationContext 实现方式：
* 1、ClassPathXmlApplicationContext：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。
* ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);
* 2、FileSystemXmlApplicationContext ：由文件系统中的XML配置文件读取上下文。
* ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);
* 3、XmlWebApplicationContext：由Web应用的XML文件读取上下文。
* Application contexts提供一种方法处理文本消息，一个通常的做法是加载文件资源（比如镜像），它们可以向注册为监听器的bean发布事件。另外，在容器或容器内的对象上执行的那些不得不由bean工厂以程序化方式处理的操作，可以在Application contexts中以声明的方式处理。Application contexts实现了MessageSource接口，该接口的实现以可插拔的方式提供获取本地化消息的方法。

### **17. 一个Spring的应用看起来象什么？**

* 一个定义了一些功能的接口。
* 这实现包括属性，它的Setter ， getter 方法和函数等。
* Spring AOP。
* Spring 的XML 配置文件。
* 使用以上功能的客户端程序。

## **依赖注入**

### **18. 什么是Spring的依赖注入？**

依赖注入，是IOC的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC容器）负责把他们组装起来。

### **19.  有哪些不同类型的IOC（依赖注入）方式？**

* **构造器依赖注入：**构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。
* **Setter方法注入：**Setter方法注入是容器通过调用无参构造器或无参static工厂 方法实例化bean之后，调用该bean的setter方法，即实现了基于setter的依赖注入。

### **20. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入？**

你两种依赖方式都可以使用，构造器注入和Setter方法注入。最好的解决方案是用构造器参数实现强制依赖，setter方法实现可选依赖。

## **Spring Beans**

### **21.什么是Spring beans?**

Spring beans 是那些形成Spring应用的主干的java对象。它们被Spring IOC容器初始化，装配，和管理。这些beans通过容器中配置的元数据创建。比如，以XML文件中<bean/> 的形式定义。

Spring 框架定义的beans都是单件beans。在bean tag中有个属性”singleton”，如果它被赋为TRUE，bean 就是单件，否则就是一个 prototype bean。默认是TRUE，所以所有在Spring框架中的beans 缺省都是单件。

### **22. 一个 Spring Bean 定义 包含什么？**

一个Spring Bean 的定义包含容器必知的所有配置元数据，包括如何创建一个bean，它的生命周期详情及它的依赖。

### **23. 如何给Spring 容器提供配置元数据?**

这里有三种重要的方法给Spring 容器提供配置元数据。

XML配置文件。

基于注解的配置。

基于java的配置。

### **24. 你怎样定义类的作用域?**

当定义一个<bean> 在Spring里，我们还能给这个bean声明一个作用域。它可以通过bean 定义中的scope属性来定义。如，当Spring要在需要的时候每次生产一个新的bean实例，bean的scope属性被指定为prototype。另一方面，一个bean每次使用的时候必须返回同一个实例，这个bean的scope 属性 必须设为 singleton。

### **25. 解释Spring支持的几种bean的作用域。**

Spring框架支持以下五种bean的作用域：

* **singleton :**bean在每个Spring ioc 容器中只有一个实例。
* **prototype**：一个bean的定义可以有多个实例。
* **request**：每次http请求都会创建一个bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。
* **session**：在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。
* **global-session**：在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

缺省的Spring bean 的作用域是Singleton.

### **26. Spring框架中的单例bean是线程安全的吗?**

不，Spring框架中的单例bean不是线程安全的。

### **27. 解释Spring框架中bean的生命周期。**

* Spring容器 从XML 文件中读取bean的定义，并实例化bean。
* Spring根据bean的定义填充所有的属性。
* 如果bean实现了BeanNameAware 接口，Spring 传递bean 的ID 到 setBeanName方法。
* 如果Bean 实现了 BeanFactoryAware 接口， Spring传递beanfactory 给setBeanFactory 方法。

如果有任何与bean相关联的BeanPostProcessors，Spring会在postProcesserBeforeInitialization()方法内调用它们。

* 如果bean实现IntializingBean了，调用它的afterPropertySet方法，如果bean声明了初始化方法，调用此初始化方法。
* 如果有BeanPostProcessors 和bean 关联，这些bean的postProcessAfterInitialization() 方法将被调用。
* 如果bean实现了 DisposableBean，它将调用destroy()方法。

### **28.  哪些是重要的bean生命周期方法？ 你能重载它们吗？**

有两个重要的bean 生命周期方法，第一个是setup ， 它是在容器加载bean的时候被调用。第二个方法是 teardown  它是在容器卸载类的时候被调用。

The bean 标签有两个重要的属性（init-method和destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct和@PreDestroy）。

### **29. 什么是Spring的内部bean？**

当一个bean仅被用作另一个bean的属性时，它能被声明为一个内部bean，为了定义inner bean，在Spring 的 基于XML的 配置元数据中，可以在 <property/>或 <constructor-arg/> 元素内使用<bean/> 元素，内部bean通常是匿名的，它们的Scope一般是prototype。

### **30. 在 Spring中如何注入一个java集合？**

Spring提供以下几种集合的配置元素：

* <list>类型用于注入一列值，允许有相同的值。
* <set> 类型用于注入一组值，不允许有相同的值。
* <map> 类型用于注入一组键值对，键和值都可以为任意类型。
* <props>类型用于注入一组键值对，键和值都只能为String类型。

### **31. 什么是bean装配?**

装配，或bean 装配是指在Spring 容器中把bean组装到一起，前提是容器需要知道bean的依赖关系，如何通过依赖注入来把它们装配到一起。

### **32. 什么是bean的自动装配？**

Spring 容器能够自动装配相互合作的bean，这意味着容器不需要<constructor-arg>和<property>配置，能通过Bean工厂自动处理bean之间的协作。

### **33. 解释不同方式的自动装配 。**

有五种自动装配的方式，可以用来指导Spring容器用自动装配方式来进行依赖注入。

* **no**：默认的方式是不进行自动装配，通过显式设置ref 属性来进行装配。
* **byName：**通过参数名 自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byname，之后容器试图匹配、装配和该bean的属性具有相同名字的bean。
* **byType:：**通过参数类型自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byType，之后容器试图匹配、装配和该bean的属性具有相同类型的bean。如果有多个bean符合条件，则抛出错误。
* **constructor：这个方式类似于**byType， 但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
* **autodetect：**首先尝试使用constructor来自动装配，如果无法工作，则使用byType方式。

### **34.自动装配有哪些局限性 ?**

自动装配的局限性是：

* **重写**： 你仍需用 <constructor-arg>和 <property> 配置来定义依赖，意味着总要重写自动装配。
* **基本数据类型**：你不能自动装配简单的属性，如基本数据类型，String字符串，和类。
* **模糊特性：**自动装配不如显式装配精确，如果有可能，建议使用显式装配。

### **35. 你可以在Spring中注入一个null 和一个空字符串吗？**

**可以。**

## **Spring注解**

### **36. 什么是基于Java的Spring注解配置? 给一些注解的例子.**

基于Java的配置，允许你在少量的Java注解的帮助下，进行你的大部分Spring配置而非通过XML文件。

以@Configuration 注解为例，它用来标记类可以当做一个bean的定义，被Spring IOC容器使用。另一个例子是@Bean注解，它表示此方法将要返回一个对象，作为一个bean注册进Spring应用上下文。

### **37. 什么是基于注解的容器配置?**

相对于XML文件，注解型的配置依赖于通过字节码元数据装配组件，而非尖括号的声明。

开发者通过在相应的类，方法或属性上使用注解的方式，直接组件类中进行配置，而不是使用xml表述bean的装配关系。

### **38. 怎样开启注解装配？**

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在Spring配置文件中配置 <context:annotation-config/>元素。

### **39. @Required  注解**

这个注解表明bean的属性必须在配置的时候设置，通过一个bean定义的显式的属性值或通过自动装配，若@Required注解的bean属性未被设置，容器将抛出BeanInitializationException。

### **40. @Autowired 注解**

@Autowired 注解提供了更细粒度的控制，包括在何处以及如何完成自动装配。它的用法和@Required一样，修饰setter方法、构造器、属性或者具有任意名称和/或多个参数的PN方法。

### **41. @Qualifier 注解**

当有多个相同类型的bean却只有一个需要自动装配时，将@Qualifier 注解和@Autowire 注解结合使用以消除这种混淆，指定需要装配的确切的bean。

## **Spring数据访问**

### **42.在Spring框架中如何更有效地使用JDBC?**

使用SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写statements 和 queries从数据存取数据，JDBC也可以在Spring框架提供的模板类的帮助下更有效地被使用，这个模板叫JdbcTemplate （例子见这里[here](http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/)）

### **43. JdbcTemplate**

JdbcTemplate 类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

### **44. Spring对DAO的支持**

Spring对数据访问对象（DAO）的支持旨在简化它和数据访问技术如JDBC，Hibernate or JDO 结合使用。这使我们可以方便切换持久层。编码时也不用担心会捕获每种技术特有的异常。

### **45. 使用Spring通过什么方式访问Hibernate?**

在Spring中有两种方式访问Hibernate：

* 控制反转  Hibernate Template和 Callback。
* 继承 HibernateDAOSupport提供一个AOP 拦截器。

### **46. Spring支持的ORM**

Spring支持以下ORM：

* Hibernate
* iBatis
* JPA (Java Persistence API)
* TopLink
* JDO (Java Data Objects)
* OJB

### **47.如何通过HibernateDaoSupport将Spring和Hibernate结合起来？**

用Spring的 SessionFactory 调用 LocalSessionFactory。集成过程分三步：

* 配置the Hibernate SessionFactory。
* 继承HibernateDaoSupport实现一个DAO。
* 在AOP支持的事务中装配。

### **48. Spring支持的事务管理类型**

Spring支持两种类型的事务管理：

* **编程式事务管理**：这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
* **声明式事务管理：**这意味着你可以将业务代码和事务管理分离，你只需用注解和XML配置来管理事务。

### **49. Spring框架的事务管理有哪些优点？**

* 它为不同的事务API  如 JTA，JDBC，Hibernate，JPA 和JDO，提供一个不变的编程模式。
* 它为编程式事务管理提供了一套简单的API而不是一些复杂的事务API如
* 它支持声明式事务管理。
* 它和Spring各种数据访问抽象层很好得集成。

### **50. 你更倾向用那种事务管理类型？**

大多数Spring框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此更符合一个无侵入的轻量级容器的思想。声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

## **Spring面向切面编程（AOP）**

### **51.  解释AOP**

面向切面的编程，或AOP， 是一种编程技术，允许程序模块化横向切割关注点，或横切典型的责任划分，如日志和事务管理。

### **52. Aspect 切面**

AOP核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组API提供横切功能。比如，一个日志模块可以被称作日志的AOP切面。根据需求的不同，一个应用程序可以有若干切面。在Spring AOP中，切面通过带有@Aspect注解的类实现。

### **52. 在Spring AOP 中，关注点和横切关注的区别是什么？**

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。  
横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

### **54. 连接点**

连接点代表一个应用程序的某个位置，在这个位置我们可以插入一个AOP切面，它实际上是个应用程序执行Spring AOP的位置。

### **55. 通知**

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过SpringAOP框架触发的代码段。

Spring切面可以应用五种类型的通知：

* **before**：前置通知，在一个方法执行前被调用。
* **after:**在方法执行之后调用的通知，无论方法执行是否成功。
* **after-returning:**仅当方法成功完成后执行的通知。
* **after-throwing:**在方法抛出异常退出时执行的通知。
* **around:**在方法执行之前和之后调用的通知。

### **56. 切点**

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

### **57. 什么是引入?**

引入允许我们在已存在的类中增加新的方法和属性。

### **58. 什么是目标对象?**

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

### **59. 什么是代理?**

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

### **60. 有几种不同类型的自动代理？**

BeanNameAutoProxyCreator

DefaultAdvisorAutoProxyCreator

Metadata autoproxying

### **61. 什么是织入。什么是织入应用的不同点？**

织入是将切面和到其他应用类型或对象连接或创建一个被通知对象的过程。

织入可以在编译时，加载时，或运行时完成。

### **62. 解释基于XML Schema方式的切面实现。**

在这种情况下，切面由常规类以及基于XML的配置实现。

### **63. 解释基于注解的切面实现**

在这种情况下(基于@AspectJ的实现)，涉及到的切面声明的风格与带有java5标注的普通java类一致。

## **Spring 的MVC**

### **64. 什么是Spring的MVC框架？**

Spring 配备构建Web 应用的全功能MVC框架。Spring可以很便捷地和其他MVC框架集成，如Struts，Spring 的MVC框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

### **65. DispatcherServlet**

Spring的MVC框架是围绕DispatcherServlet来设计的，它用来处理所有的HTTP请求和响应。

### **66. WebApplicationContext**

WebApplicationContext 继承了ApplicationContext  并增加了一些WEB应用必备的特有功能，它不同于一般的ApplicationContext ，因为它能处理主题，并找到被关联的servlet。

### **67. 什么是Spring MVC框架的控制器？**

控制器提供一个访问应用程序的行为，此行为通常通过服务接口实现。控制器解析用户输入并将其转换为一个由视图呈现给用户的模型。Spring用一个非常抽象的方式实现了一个控制层，允许用户创建多种用途的控制器。

### **68. @Controller 注解**

该注解表明该类扮演控制器的角色，Spring不需要你继承任何其他控制器基类或引用Servlet API。

### **69. @RequestMapping 注解**

该注解是用来映射一个URL到一个类或一个特定的方处理法上。