JAVA 高级框架面试题

Spring

什么是 spring

Spring 是个 java 企业级应用的开源开发框架。目的是简化我们的开发流程以及更好适应三层架构(controller, service, dao)开发，实现对象间的解耦. 它还可以集成很多其他框架和插件比如: mybatis, hibernate, struts, jdbc, redis 等等.

在项目中是怎样使用 Spring，或者你是怎样理解 Spring

我们在项目当中会使用到 Spring 的 IOC 实现依赖注入或者控制反转， AOP 实现事物的控制，操作日志的收集，拦截器(写过一个登陆拦截器，过滤未登录的情况下请求登陆的接口数据)，异常处理, Spring test 以及注解的应用. Spring 其实像是一个幕后黑手，牢牢控制住整个项目. 连接着 SpringMVC 和 Mybatis

IOC

1. 什么叫 IOC

IOC 包括两方面: 一个叫依赖注入，另一个叫控制反转

依赖注入: 是说你不用创建对象，而只需要描述它如何被创建，简单讲就是帮助 new 对象并且管理这些对象。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

控制反转: 就是说我将 new 对象的这个权利由原来对象本身转移到 Spring 身上. 举个例子: Service A 调用 Dao B 的时候，如果不用 Spring 的话，那肯定要有 Service A 去 new 一个 Dao B 对象出来，再去调用. 但如果用了 Spring 后，我们 new Dao B 这个过程就已经由 Spring 处理好.

2. 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(容器启动时加载 bean)和懒加载(用到时加载 bean)。

3. 依赖注入的有几种方式

总共有四种: 常用的是构造器注入，setter 方式注入，还有静态工厂注入，动态工厂注入. 用构造器参数实现强制依赖，setter 方法实现可选依赖.

4. 什么是 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。

5. 饿汉式初始化 bean 和懒加载初始化 bean 有什么区别和怎样实现

这两者的区别就是一个是容器启动时加载所有的 bean，一个是使用时才去加载，默认是饿汉式初始化，如果要启用懒加载只需要在配置 bean 的时候 配置 lazy-init=" true "

Spring Bean

1. 什么是 Spring beans?

Spring beans 就是交由 Spring 管理的 java 对象。它们被 Spring IOC 容器初始化，装配，和管理。

2. 怎样定义一个 bean

一种通过 xml 配置，一种通过注解

3. 你怎样定义 bean 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。

4. Spring bean 的作用域有几种

Spring 框架支持以下五种 bean 的作用域：

a) singleton : bean 在每个 Spring ioc 容器中只有一个实例。

b) prototype：一个 bean 的定义可以有多个实例。

c) 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。

d) 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。

e) 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.

5. Spring 框架中的单例 bean 是线程安全的吗?

不一定，Spring 框架中的单例 bean 不是线程安全的。

6. Spring bean 的生命周期。

以 BeanFactory 为例，说明一个 Bean 的生命周期活动

第一步: bean 的建立

从配置文件中读取 bean 文件，生成各个实例

第二步: Setter 注入

执行 Bean 的属性依赖注入

第三步: BeanNameAware 的 setBeanName()

如果 Bean 类实现了 org.springframework.beans.factory.BeanNameAware 接口，则执行其 setBeanName()方法。

第四步: BeanFactoryAware 的 setBeanFactory()

如果 Bean 类实现了 org.springframework.beans.factory.BeanFactoryAware 接口，则执行其 setBeanFactory()方法

第五步: BeanPostProcessors 的 processBeforeInitialization()

容器中如果有实现 org.springframework.beans.factory.BeanPostProcessors 接口的实例，则任何 Bean 在初始化之前都会执行这个实例的 processBeforeInitialization()方法

第六步: InitializingBean 的 afterPropertiesSet()

如果 Bean 类实现了 org.springframework.beans.factory.InitializingBean 接口，则执行其 afterPropertiesSet()方法

Bean 定义文件中定义 init-method

在 Bean 定义文件中使用“init-method”属性设定方法名称

第七步: BeanPostProcessors 的 processAfterInitialization()

容器中如果有实现 org.springframework.beans.factory.BeanPostProcessors 接口的实例，则任何 Bean 在初始化之前都会执行这个实例的 processAfterInitialization()方法

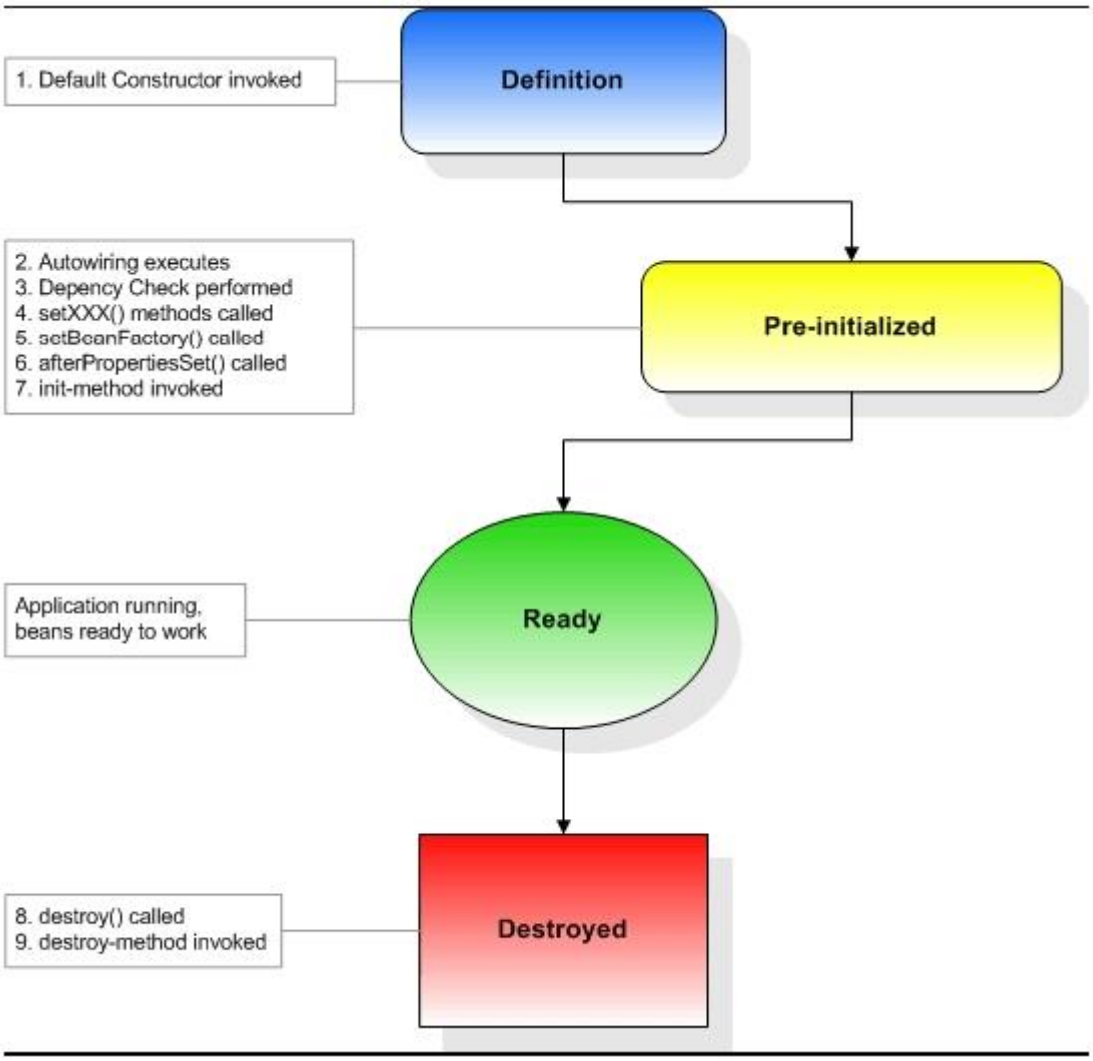
第八步: DisposableBean 的 destroy()

在容器关闭时，如果 Bean 类实现了

org.springframework.beans.factory.DisposableBean 接口，则执行它的 destroy()方法

Bean 定义文件中定义 destroy-method

在容器关闭时，可以在 Bean 定义文件中使用“destory-method”定义的方法



7. 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。

8. 在 Spring 中如何注入一个 java 集合？

<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。

9. 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。

10. 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。

11. 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。

12. 自动装配有哪些局限性

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。

13. 你可以在 Spring 中注入一个 null 和一个空字符串吗

可以。

AOP

1. 什么叫 AOP

AOP 字面意思叫面向切面编程，它是面向对象编程的一种补充，为什么这样讲，我们面向对象编程着重看点的是某个对象的某个方法，但如果我要关注一批方法的时候怎么办这个就是面向切面编程来处理.

2. AOP 有哪些好处和应用场景

好处

1) 降低模块的耦合度
2) 使系统容易扩展
3) 设计决定的迟绑定：使用 AOP,设计师可以推迟为将来的需求作决定，因为它
4) 可以把这种需求作为独立的方面很容易的实现。
5) 更好的代码复用性

场景

使用场景: AOP 能够将那些与业务无关，却为业务模块所共同调用的逻辑或责任，例如事务处理、日志管理、权限控制等，封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。

3. AOP 的实现方式

AOP 实现的方式是动态代理

4. 什么叫静态代理？什么叫动态代理？二者之间的区别

静态代理是: 由程序员创建或由特定工具自动生成源代码，再对其编译。在程序运行前，代理类的.class 文件就已经存在了。动态代理类：在程序运行时，运用反射机制动态创建而成

动态代理是: 与静态代理类对照的是动态代理类，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。java.lang.reflect 包中的 Proxy 类和 InvocationHandler 接口提供了生成动态代理类的

能力。

二者的区别是：静态代理是在运行前代理，而动态代理则是在运行时代理；另外一个最重要的是静态代理只能代理同一个接口下的实现类，而动态代理可以代理很多类．所以实现切面编程的时候采用了动态代理.因为只有动态代理才能扩展到面．这样说 AOP 编程是对 OOP 的一个补充．因为 AOP 关注的是一个切面，而 OOP 关注点还是在某个方法上.此外静态代理的话如果接口添加一个新的方法，所有的实现类和代理类都要做这个实现，这就增加了代码的复杂度，而动态代理只需要更改一下代理方法即可．

详情可以查看老师静态代理和动态代理的文档

5. 动态代理的两种实现方式

使用了 JDK 的动态代理和第三方框架 CGLIB(包括 Spring 的 cglib 还有其他的 cglib.jar)实现，实现例子可以参考老师文档

6. AOP 的几个名字解释

a) 连接点(JoinPoint)

表示需要在程序中插入横切关注点的扩展点，连接点可能是类初始化、方法执行、方法调用、字段调用或处理异常等等，Spring 只支持方法执行连接点，**在 AOP 中表示为“在哪里做”**

b) 切入点(PointCut)

选择一组相关连接点的模式，即可以认为连接点的集合，Spring 支持 perl5 正则表达式和 AspectJ 切入点模式，Spring 默认使用 AspectJ 语法，**在 AOP 中表示为“在哪里做的集合”**

c) 增强(Advice)

在连接点上执行的行为，增强提供了在 AOP 中需要在切入点所选择的连接点处进行扩展现有行为的手段；包括前置增强（before advice）、后置增强 (after advice)、环绕增强 （around advice），在 Spring 中通过代理模式实现 AOP，并通过拦截器模式以环绕连接点的拦截器链织入增强；**在 AOP 中表示为“做什么”**

d) 方面 / 切面(Aspect)

横切关注点的模块化，比如上边提到的日志组件。可以认为是增强、引入和切入点的组合；在 Spring 中可以使用 Schema 和@AspectJ 方式进行组织实现；**在 AOP 中表示为“在哪里做和做什么集合”**

e) 目标对象(Target Object)

需要被织入横切关注点的对象，即该对象是切入点选择的对象，需要被增强的对象，从而也可称为“被增强对象”；由于 Spring AOP 通过代理模式实现，从而这个对象永远是被代理对象，**在 AOP 中表示为“对谁做”**

f) AOP 代理(AOP Proxy)

AOP 框架使用代理模式创建的对象，从而实现在连接点处插入增强（即应用切面），就是**通过代理来对目标对象应用切面**。在 Spring 中，AOP 代理可以用 JDK 动态代理或 CGLIB 代理实现，而通过拦截器模型应用切面

g) 织入(Weaving)

织入是一个过程，是将切面应用到目标对象从而创建出 AOP 代理对象的过程，织入可以在编译期、类装载期、运行期进行

h) 引入(Inter-type Declaration)

也称为内部类型声明，为已有的类添加额外新的字段或方法，Spring 允许引入新的接口（必须对应一个实现）到所有被代理对象（目标对象）, **在 AOP 中表示为“做什么（新增什么）”**

i) 补充 AOP 的 Advice 类型

前置增强（Before advice）

在某连接点之前执行的增强，但这个增强不能阻止连接点前的执行（除非它抛出一个异常）。

后置返回增强（After returning advice）

在某连接点正常完成后执行的增强：例如，一个方法没有抛出任何异常，正常返回。

后置异常增强（After throwing advice）

在方法抛出异常退出时执行的增强。

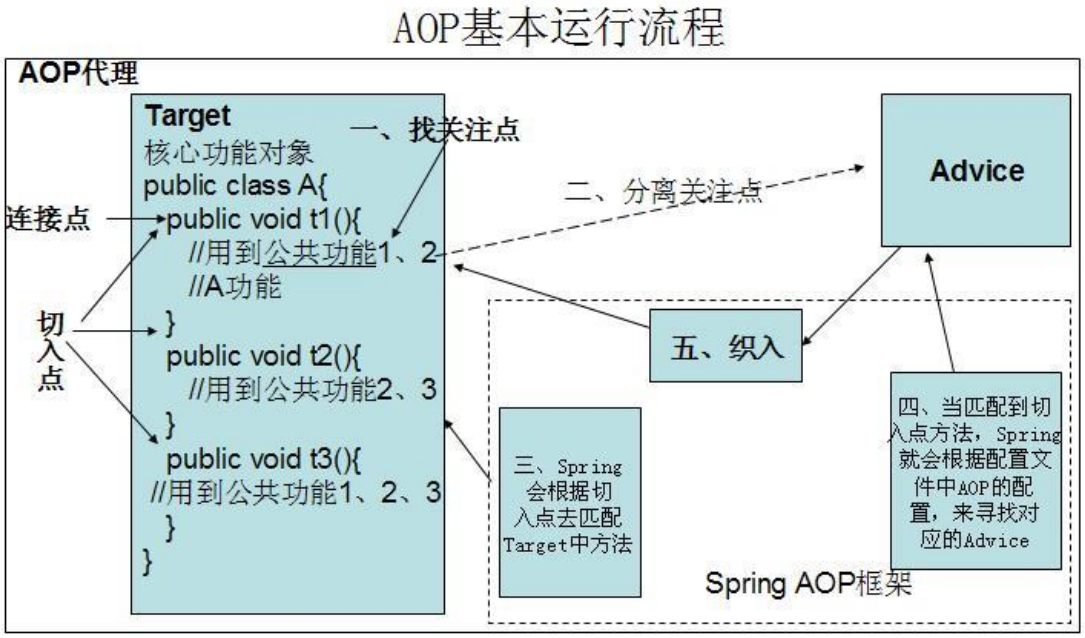
后置最终增强（After (finally) advice）：

当某连接点退出的时候执行的增强（不论是正常返回还是异常退出）。

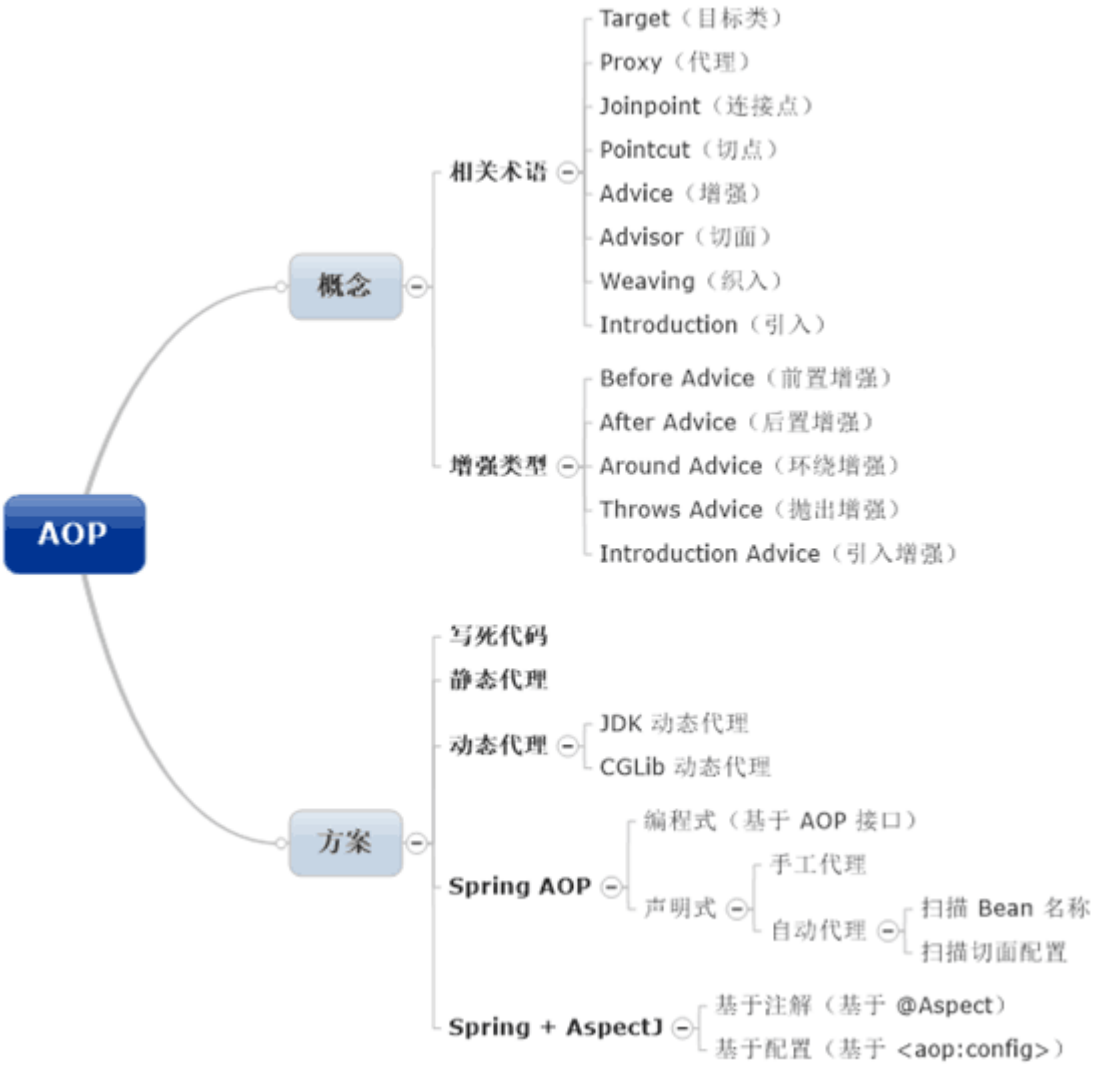
环绕增强（Around Advice）：

包围一个连接点的增强，如方法调用。这是最强大的一种增强类型。 环绕增强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。

7. AOP 的运行流程



8. AOP 总 结



事物管理

Spring 对事务的解决办法其实分为 2 种：编程式实现事务，AOP 配置声明式解决方案，编程时实现事物只做了解，Spring 中主要使用 AOP 声明式解决方法

Spring 提供了许多内置事务管理器实现，常用的有以下几种：

- **DataSourceTransactionManager**：位于 org.springframework.jdbc.datasource 包中，数据源事务管理器，提供对单个 javax.sql.DataSource 事务管理，用于 Spring JDBC 抽象框架、Mybatis 框架的事务管理；
- **HibernateTransactionManager**：位于 org.springframework.orm.hibernate3 或者 hibernate4 包中，提供对单个 org.hibernate.SessionFactory 事务支持，用于集成 Hibernate 框架时的事务管理；该事务管理器只支持 Hibernate3+版本，且 Spring3.0+版本只支持 Hibernate 3.2+版本；

- **JtaTransactionManager**：位于 org.springframework.transaction.jta 包中，提供对分布式事务管理的支持，并将事务管理委托给 Java EE 应用服务器事务管理器；

编程式实现事务 (了解)

Spring 提供两种编程式事务支持：直接使用 PlatformTransactionManager 实现和使用 TransactionTemplate 模板类，用于支持逻辑事务管理。 如果采用编程式事务推荐使用 TransactionTemplate 模板类。

Spring 声明式事务

就是我们项目当中所用的通过配置 SpringJdbc 的声明式来实现

1. 数据源配置
2. 指定事物管理器
3. 设定事物增强

大致如下：

```
<!-- 配置数据源 -->
    <bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource"
destroy-method="close">
        <property name="poolProperties">
            <bean class="org.apache.tomcat.jdbc.pool.PoolProperties">
                <property name="driverClassName" value="${JDBC.driver}"/>
                <property name="url" value="${JDBC.url}"/>
                <property name="username" value="${JDBC.username}"/>
                <property name="password" value="${JDBC.password}"/>
                <property name="jmxEnabled" value="true"/>
                <property name="testWhileIdle" value="true"/>
                <property name="testOnBorrow" value="true"/>
                <property name="testOnReturn" value="false"/>
                <property name="validationInterval" value="30000"/>
                <property name="validationQuery" value="SELECT 1"/>
                <property name="timeBetweenEvictionRunsMillis"
value="30000"/>
                <property name="maxActive" value="200"/>
                <property name="initialSize" value="10"/>
                <property name="maxWait" value="30000"/>
                <property name="minEvictableIdleTimeMillis"
value="30000"/>
                <property name="minIdle" value="10"/>
```

```
                <property name="logAbandoned" value="false"/>
                <property name="removeAbandoned" value="true"/>
                <property name="removeAbandonedTimeout" value="60"/>
                <property name="jdbcInterceptors"
value="org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;org.apac
he.tomcat.jdbc.pool.interceptor.StatementFinalizer"/>
            </bean>
        </property>
    </bean>

    <!-- ========================================事物配置
========================================= -->
    <!-- 指定事物管理器 -->
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <!-- 设置事物增强 -->
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
            <tx:method name="get*" read-only="true" />
            <tx:method name="find*" read-only="true" />
            <tx:method name="query*" read-only="true" />
            <tx:method name="load*" read-only="true" />
            <tx:method name="add*" rollback-for="Exception"/>
            <tx:method name="insert*" rollback-for="Exception" />
            <tx:method name="update*" rollback-for="Exception" />
            <tx:method name="delete*" rollback-for="Exception" />
        </tx:attributes>
    </tx:advice>

    <!-- 作用Shcema的方式配置事务，这里是把事务设置到了service层 这里就用到了
spring aop-->
    <aop:config>
        <aop:pointcut id="servicePointcut" expression="execution(*
com.shop.service.*.*(..))" />
        <aop:advisor advice-ref="txAdvice"
pointcut-ref="servicePointcut"/>
    </aop:config>

    <!-- 基于注解管理事物 就是在类名前加上@Transactional-->
```

```
<tx:annotation-driven transaction-manager="txManager"/>
```

事物增强的配置

```
<tx:advice id="……" transaction-manager="……">
<tx:attributes>
        <tx:method name="*"
                                propagation="REQUIRED"
                                isolation="DEFAULT"
                                timeout="-1"
                                read-only="true"
                                no-rollback-for=""
                                rollback-for="java.lang.Exception"/>
    </tx:attributes>
</tx:advice>

<!-- 最常用的配置 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
<tx:attributes>
            <tx:method name="save*" propagation="REQUIRED" />
            <tx:method name="add*" propagation="REQUIRED" />
            <tx:method name="create*" propagation="REQUIRED" />
            <tx:method name="insert*" propagation="REQUIRED" />
            <tx:method name="update*" propagation="REQUIRED" />
            <tx:method name="merge*" propagation="REQUIRED" />
            <tx:method name="del*" propagation="REQUIRED" />
            <tx:method name="remove*" propagation="REQUIRED" />
            <tx:method name="put*" propagation="REQUIRED" />
            <tx:method name="get*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="count*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="find*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="list*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="batchSaveOrUpdate" propagation="REQUIRES_NEW" />
        </tx:attributes>
</tx:advice>
<aop:config>
        <aop:pointcut id="txPointcut" expression="execution(* com.shop.service.*.*(..))" />
        <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />
</aop:config>
```

XML 形式的事务配置<tx:method >的属性详解

属性	类型	默认值	说明

propagation	Propagation 枚举	REQUIRED	事务传播属性
isolation	isolation 枚举	DEFAULT(所用数据库默认级别)	事务隔离级别
readOnly	boolean	false	是否才用优化的只读事务
timeout	int	-1	超时(秒)
rollbackFor	Class[]	{}	需要回滚的异常类
rollbackForClassName	String[]	{}	需要回滚的异常类名
noRollbackFor	Class[]	{}	不需要回滚的异常类
noRollbackForClassName	String[]	{}	不需要回滚的异常类

readOnly

事务属性中的 readOnly 标志表示对应的事务应该被最优化为只读事务。如果值为 true 就会告诉 Spring 我这个方法里面没有 insert 或者 update，你只需要提供只读的数据库 Connection 就行了，这种执行效率会比 read-write 的 Connection 高，所以这是一个最优化提示。在一些情况下，一些事务策略能够起到显著的最优化效果，例如在使用 Object/Relational 映射工具（如：Hibernate 或 TopLink）时避免 dirty checking（试图“刷新”）。

timeout

在属性中还有定义“timeout”值的选项，指定事务超时为几秒。一般不会使用这个属性。在 JTA 中，这将被简单地传递到 J2EE 服务器的事务协调程序，并据此得到相应的解释。

Isolation Level(事务隔离等级)的 5 个枚举值

为什么事务要有 Isolation Level 这个属性？先回顾下数据库事务的知识：

第一类丢失更新(lost update)：在完全未隔离事务的情况下，两个事物更新同一条数据资源，某一事物异常终止，回滚造成第一个完成的更新也同时丢失。

第二类丢失更新(second lost updates)：是不可重复读的特殊情况，如果两个事务都读取同一行，然后两个都进行写操作，并提交，第一个事务所做的改变就会丢失。

脏读(dirty read)：如果第二个事务查询到第一个事务还未提交的更新数据，形成脏读。因为

第一个事务你还不知道是否提交，所以数据不一定是正确的。

虚读(phantom read)：一个事务执行两次查询，第二次结果集包含第一次中没有或者某些行已被删除，造成两次结果不一致，只是另一个事务在这两次查询中间插入或者删除了数据造成的。

不可重复读(unrepeated read)：一个事务两次读取同一行数据，结果得到不同状态结果，如中间正好另一个事务更新了该数据，两次结果相异，不可信任。

当遇到以上这些情况时我们可以设置 isolation 下面这些枚举值：

DEFAULT：采用数据库默认隔离级别

SERIALIZABLE：最严格的级别，事务串行执行，资源消耗最大；

REPEATABLE_READ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。

READ_COMMITTED:大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。

READ_UNCOMMITTED：保证了读取过程中不会读取到非法数据。隔离级别在于处理多事务的并发问题。

关于 propagation 属性的 7 个传播行为

REQUIRED：指定当前方法必需在事务环境中运行，如果当前有事务环境就加入当前正在执行的事务环境，如果当前没有事务，就新建一个事务。这是默认值。

SUPPORTS：指定当前方法加入当前事务环境，如果当前没有事务，就以非事务方式执行。

MANDATORY：指定当前方法必须加入当前事务环境，如果当前没有事务，就抛出异常。

REQUIRES_NEW：指定当前方法总是会为自己发起一个新的事务，如果发现当前方法已运行在一个事务中,则原有事务被挂起,我自己创建一个属于自己的事务,直我自己这个方法 commit 结束,原先的事务才会恢复执行。

NOT_SUPPORTED：指定当前方法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，等我以非事务的状态运行完，再继续原来的事务。

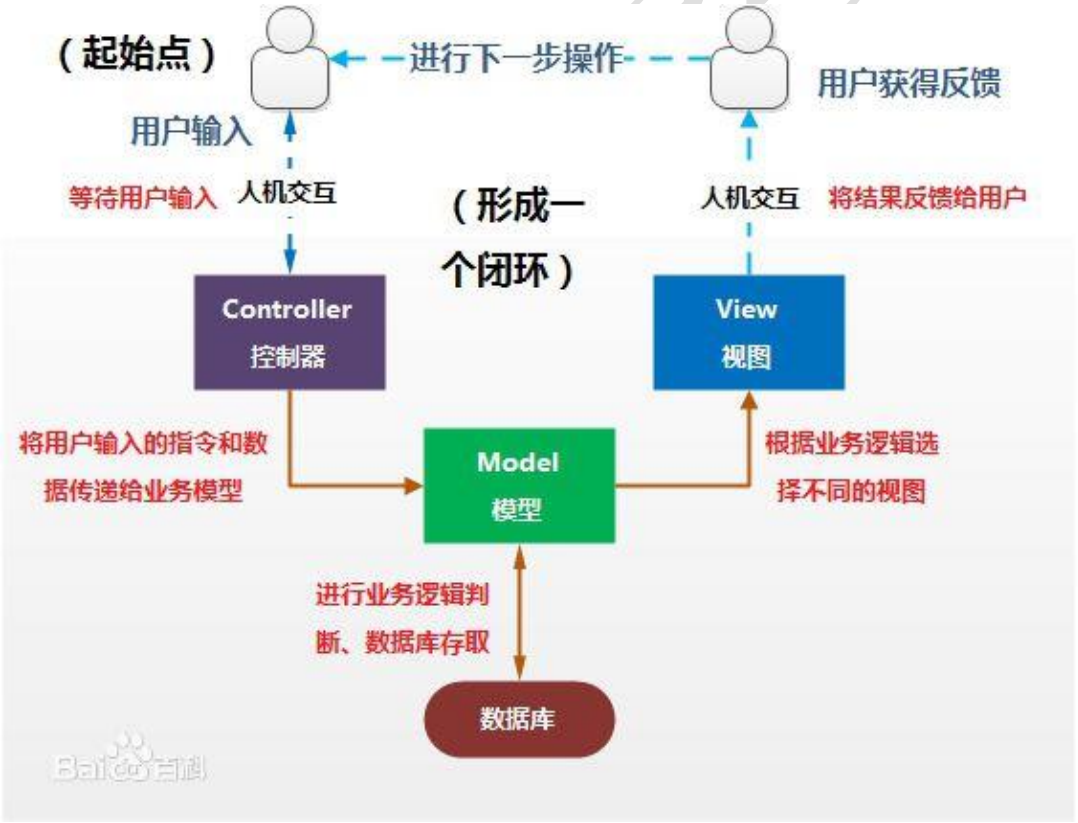
NEVER：指定当前方法绝对不能在事务范围内执行，如果方法在某个事务范围内执行，容器就抛异常，只有没关联到事务，才正常执行。

NESTED：指定当前方法执行时，如果已经有一个事务存在,则运行在这个嵌套的事务中.如果当前环境没有运行的事务，就新建一个事务，并与父事务相互独立，这个事务拥有多个可以回滚的保证点。就是指我自己内部事务回滚不会对外部事务造成影响，只对 DataSourceTransactionManager 事务管理器起效。

SpringMVC

什么叫 MVC

模型-视图-控制器（MVC）是一个众所周知的以设计界面应用程序为基础的设计模式。它主要通过分离模型、视图及控制器在应用程序中的角色将业务逻辑从界面中解耦。通常，模型负责封装应用程序数据在视图层展示。视图仅仅只是展示这些数据，不包含任何业务逻辑。控制器负责接收来自用户的请求，并调用后台服务（service 或者 dao）来处理业务逻辑。处理后，后台业务层可能会返回了一些数据在视图层展示。控制器收集这些数据及准备模型在视图层展示。**MVC 模式的核心思想是将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响**。



什么叫 SpringMVC

Spring MVC 是 Spring 家族中的一个 web 成员，它是一种基于 Java 的实现了 Web MVC 设计模式的请求驱动类型的轻量级 Web 框架，即使用了 MVC 架构模式的思想，将 web 层

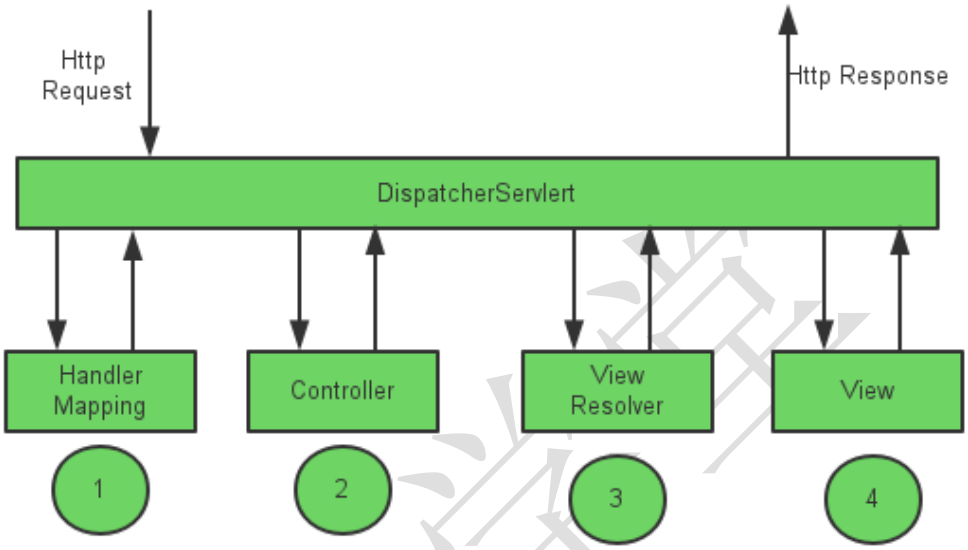
进行职责解耦，**基于请求驱动指的就是使用请求-响应模型**，框架的目的就是帮助我们简化开发，Spring MVC 也是要简化我们日常 Web 开发的。

Spring MVC 也是服务到工作者模式的实现，但进行可优化。**前端控制器是 `DispatcherServlet`**；应用控制器其实拆为**处理器映射器(Handler Mapping)**进行处理器管理和**视图解析器(View Resolver)**进行视图管理；页面控制器/动作/处理器为 Controller 接口（仅包含 `ModelAndView handleRequest(request, response)` 方法）的实现（也可以是任何的 POJO 类）；支持本地化（Locale）解析及文件上传等；提供了非常灵活的数据验证、格式化和数据绑定机制；提供了强大的约定大于配置（惯例优先原则）的契约式编程支持。

SpringMVC 的好处

1. 能简单的进行 Web 层的单元测试；
2. 支持灵活的 URL 到页面控制器的映射；
3. 多视图集成，如 Velocity、FreeMarker 等等，因为模型数据不放在特定的 API 里，而是放在一个 Model 里（Map 数据结构实现，因此很容易被其他框架使用）；
4. 非常灵活的数据验证、格式化和数据绑定机制，能使用任何对象进行数据绑定，不必实现特定框架的 API；
5. 更加简单的异常处理；
6. 对静态资源的支持；
7. 支持 Restful 风格.

SpringMVC 工作流程



具体流程步骤如下：

1、 首先用户发送请求——>DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制；

2、 DispatcherServlet——>HandlerMapping， HandlerMapping 将会把请求映射为 HandlerExecutionChain 对象（包含一个 Handler 处理器（页面控制器）对象、多个 HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；

3、 DispatcherServlet——>HandlerAdapter(Controller)，HandlerAdapter(Controller)将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；

4、 HandlerAdapter——>处理器功能处理方法的调用，HandlerAdapter 将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理；并返回一个 ModelAndView 对象（包含模型数据、逻辑视图名）；

5、 ModelAndView 的逻辑视图名——> ViewResolver， ViewResolver 将把逻辑视图名解析为具体的 View，通过这种策略模式，很容易更换其他视图技术；

6、 View——>渲染，View 会根据传进来的 Model 模型数据进行渲染，此处的 Model 实际是一个 Map 数据结构，因此很容易支持其他视图技术；

7、返回控制权给 DispatcherServlet，由 DispatcherServlet 返回响应给用户，到此一个流程结束。

总结

SpringMVC 需要掌握的东西主要是:

1. MVC 的理解
2. SpringMVC 的优点
3. SpringMVC 的几个名词: **前端控制器是 DispatcherServlet，处理器映射器(Handler Mapping)**进行处理器管理和**视图解析器(View Resolver)**.
4. SpringMVC 工作的流程图

SpringMVC 和 Struts2 的区别

SpringMVC 能够跟 Spring 无缝结合，所以配置会更少，甚至可以达到零配置

SpringMVC 的多视图应用，通过不同的扩展名可以返回不同的视图

SpringMVC 强大的注解应用，比如我要返回 JSON，只需要配置@ResponseBody，而 Struts2 则需要封装

SpringMVC 是针对于一个 Servlet 的封装，比如获取参数或者返回值，而 Struts2 则是对每一次请求有封装.

SpringMVC 设计更加的简单.

SpringMVC 支持 JSR303 更加方便验证

Mybatis

什么叫 ORM 框架

ORM 即对象关系映射，主要是把数据库中的关系数据映射称为程序中的对象，所以在我们的程序开发过程中我们针对于数据库的操作就转变成了针对于对象的操作，更加的符合我们面向对象编程.

Mybatis 的优点

1. 轻量级
2. 学习成本低
3. 执行性能高
4. 支持动态 sql 的编写

5. 方便 sql 文件的管理，因为都可以写在 mapper 文件里面

6. 解除 sql 与程序代码的耦合

Mybatis 的工作流程

(1) 加载配置并初始化

触发条件：加载配置文件 配置来源于两个地方，一处是配置文件，一处是 Java 代码的注解，将 SQL 的配置信息加载成为一个个 MappedStatement 对象（包括了传入参数映射配置、执行的 SQL 语句、结果映射配置），存储在内存中。

(2) 接收调用请求

触发条件：调用 Mybatis 提供的 API 传入参数：为 SQL 的 ID 和传入参数对象

处理过程：将请求传递给下层的请求处理层进行处理。

(3) 处理操作请求 触发条件：API 接口层传递请求过来

传入参数：为 SQL 的 ID 和传入参数对象

处理过程：

(A)根据 SQL 的 ID 查找对应的 MappedStatement 对象。

(B)根据传入参数对象解析 MappedStatement 对象，得到最终要执行的 SQL 和执行传入参数。

(C)获取数据库连接，根据得到的最终 SQL 语句和执行传入参数到数据库执行，并得到执行结果。

(D)根据 MappedStatement 对象中的结果映射配置对得到的执行结果进行转换处理，并得到最终的处理结果。

(E)释放连接资源。

(4)返回处理结果将最终的处理结果返回。

Mybatis 的缓存应用

MyBatis 的缓存分为一级缓存和二级缓存.

一级缓存放在 session 里面,默认就有,二级缓存放在它的命名空间里,默认是打开的;

二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态), 可在它的映射文件中配置<cache/>

Mybatis 的一对一，一对多的配置

一对一配置关键字 < association >< /association >

一对多配置关键字: <collection><collection>

都是在 resultMap 配置

Mybatis 动态语句的使用

1. if 条件判断
2. choose, when, otherwise 选择器使用
3. trim, where, set
4. foreach

Mybatis 其他知识点

1. 注解的使用, @Select, @Update, @Delete 使用

2. 传参的方式, 可以#{…}, ${…}. MyBatis 将 #{…} 解释为 JDBC prepared statement 的一个参数标记可以理解成占位符的方式。而将 ${…} 解释为字符串替换。${...}传参是不安全的, 可以 sql 注入

3. 批量新增, 批量删除和批量更新, 主要是我们要用到< foreach>集合的遍历