**第四部分：Spring**

**配置Spring:**

将spring-framework\libs\文件下的class类型的jar放入WEB-INF\lib下，common-logging\文件下的jar包放入WEB-INF\lib下。

src\下配置spring.xml文件：

<bean id=”” class=””> //class是类名称，可以是任何java类，包括java.util.\*，反射创建对象

<property name=”” ref=”” value=”” > //name是反射调用setter方法的名称，传入参数为类，则ref，基本类型及其包装类则value。还可以传入集合类型

<constructor-arg ref/value>//实现带参构造函数

ApplicationContext ctx=new ClassPathXmlApplicationContext(“spring.xml”)

Person p=ctx.getBean(“bean\_id”,Person.class)

**核心机制：依赖注入**

Spring推荐java类都应该面向接口定义出功能，然后写对应实现类，在调用者中添加接口，这样只需要改变xml文件中property对应ref就能引用不同被调用者类，强化多态作用。

设值注入：在调用者实现类中添加被调用者接口的成员变量，然后设置对应setter方法。

构造注入：上面的方法是默认反射调用无参构造函数，现在需要实现调用带参构造函数，通过这个参数来给成员变量赋值。

Spring中推荐类与类依赖使用设值注入ref，其他的value尽量不用，直接在类中赋值。

**Spring容器：**

BeanFactory是最基本容器接口，ApplicationContext是最常用的子接口，ClassSystemXmlApplicationContext/FileSystemXmlApplicationContext/WebApplicationContext等是常用实现类，参数为加载的xml文件路径。

**ApplicationContext的事件机制：**

编写Event继承ApplicationEvent或者使用自带的内部事件，即默认把该对象作为了Spring容器的事件。编写监听器实现ApplicationListener接口，函数中写明对各个Event的处理方法。在xml中添加监听器<bean class=””>. 这样当spring容器发布容器事件时，stx.publishEvent(Event),会立即调用监听器。

**Bean获取Spring容器：**

以往都是在spring容器中显性获取bean对象，现在实现bean中可以使用spring容器对象，这样借助spring对象可以实现如发布事件等更多功能。Bean类实现BeanFactoryAware\ApplicationContextAware接口，类中创建对应成员变量，还有接口中方法this.ctx=ctx，并在xml中添加; 这样当spring容器创建时，系统检测哪个bean实现了上述接口则会把对应spring容器引用传给此bean类。

**Bean作用范围：**

<bean scope=“”>可以指定作用范围，默认是singleton单例模式，还可以制定prototype普通模式，request模式，session模式，全局模式(后三种只有在web.xml中额外配置了才能实现)。

**注入集合类型：**

<property>

<list>

<value>\*\*</value>

</list>

<map>

<entry key=”” value=””> //spring会根据泛型自动转换类型

<entry key-ref=”” value-ref=””>

</map>

<set>

<value>

<bean> //嵌套bean，返回bean实例对象

<ref bean=”bean\_id”>

</set>

<props>

<prop key=””>\*\*</prop>

</props>

**注入组合属性：**

<property name=”person.name” value=””> // 在当前Bean类中有Person对象成员变量，相当于执行.getPerson().setName()，但要保证Person对象不为空

**Spring创建Bean类三种方式：使用时无区别，就创建方式不同**

构造器创建：默认无参构造器创建，使用<constructor-arg>标签则使用带参构建。

静态工厂创建：<bean class=”//工厂类” factory-method=”//调用工厂类中方法的名称”>

<constructor-arg value=”//方法需要的参数”>

<property name=”” ref/value=””> //创建出来的bean类注入属性

工厂类中静态方法通常依靠传入的一个参数，决定调用返回哪个构造器。

实例工厂创建：与上述基本相同，这里工厂类不再用静态方法，所以需要在xml中配置工厂类，然后<bean factory-bean=”//调用的工厂类id\_bean” factory-method=”//调用工厂类中 方法的名称”>

<constructor-arg value=”//方法需要的参数”>

<property name=”” ref/value=””> //创建出来的bean类注入属性

**建立Bean模板：避免xml文件累赘**

<bean id=”” abstract=”true”>

<property name=”” ref=””>

<property name=”” value=””>

</bean>

<bean id=”” class=”” parent=”bean\_id”> //这样这个bean就完全继承上面的模板bean中property属性

**容器中工厂Bean：与之前工厂创建Bean不同，之前spring作用只是调用工厂类，工厂产生什么bean还是由工厂类中new() 构造器代码决定，现在是完全交由xml决定，通过<property>传入参数确定构造出哪个bean实例**

工厂bean类实现FactoryBean接口，这样容器中返回的不是工厂类实例，而是getObject返回值，成员变量里有tragetClass targetField属性和对应的set方法，实现方法getObject中根据class 和 field 反射创建出所需要类的实例。系统自带的很多工厂Bean功能强大。

**获取Bean本身的ID：java应用通常由依赖注入管理Bean之前调用关系，不会显性调用getBean来获取实例，这样id号不知道**

Bean类需要实现BeanNameAware接口，和ApplicationContextAware接口一样有一个对应的setBeanName方法，容器会自动把此Bean的id传进去。

**容器中Bean声明周期：**

依赖注入之后行为：使用init-method属性标识bean中一方法，则此方法会在所有依赖注入完成之后自动执行。或者bean实现InitializingBean接口，里面一方法会自动执行。推荐前者方式，低入侵式设计。

销毁之前的行为：使用destory-method属性或者DisposableBean接口，在Bean销毁之前执行。Web应用中ApplicationContext会恰当关闭，但要让spring容器在JVM中关闭，需要添加一个关闭钩子。

**协调作用域不同的Bean依赖关系：**

singleton的bean中需要调用prototype的bean，则每次返回的后者都是同一个，为了解决这问题，引入lookup方法注入。调用者bean声明为抽象类，并定义一个抽象方法获取被调用者public abstract Axe getAxe()，在xml中调用者bean中添加<lookup-method name=”” bean=””>//系统会自动实现该抽象方法 return ctx.getBean(name)。这样就可以实现singleton的bean中调用getAxe()方法获取不同Axe()。

**获取其他Bean的属性值：**

**<bean>表示创造实例，<property>表示setter方法，下面的用来表示getter方法：**

<bean id="sheelAxe"//通过stoneAxe类的getSheelAxe()产生一个新的bean类class="org.springframework.beans.factory.config.PropertyPathFactoryBean"> //工厂bean

<property name="targetBeanName" value="stoneAxe"></property>

<property name="propertyPath" value="sheelAxe"></property>//value可以是复合属性sheelAxe.host==getSheelAxe().getHost()

</bean>

System.out.println(ctx.getBean("sheelAxe", String.class)); //.class是getSheelAxe的返回类型

也可以让此工厂类得到的getter返回值作为<property>属性

<property name=”age”> //代表需要setter，用嵌套bean

<bean id=”person.son.age” //此时id作为关键部分 person.getSon().getAge(). class="org.springframework.beans.factory.config.PropertyPathFactoryBean" >

</property>

**获取Field属性：**

唯一作用是获取静态变量，实例变量基本都是private所以没有意义。

和前面getter一样，利用FieldRetrievingFactoryBean,传入参数，返回Field属性

**调用类方法：方法可以有返回值，则可赋值给Bean，或者新的Bean，也可无返回值，就相当于调用一次该方法完成一些功能**

下面是静态方法

<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<property name="targetClass" value="beans.Chinese"></property> //类需要value

<property name="targetMethod" value="testMethod"></property>

<property name="arguments" value="xyh is a mesaa"></property>

</bean>

也可是实例方法

<property name="targetObject" ref="chinese"></property> //实例只需引用ref

**Bean后处理器：与之前Bean依赖注入之后，销毁之前不同(针对个别Bean)，这里是针对所有的Bean处理**

需要定义一个处理器实现Processor接口，重写里面两个方法 , 添加到xml中

public Object postProcessBeforeInitialization(Object bean， String bean\_id) throws BeansException

//在所有bean初始化之前执行，里面代码通常先判断bean类型instanceof，然后执行所需不同操作

public Object postProcessAfterInitialization(Object bean, String bean\_id) throws BeansException

//在所有bean初始化之后执行,同上可以用来修改初始化之后的bean中的属性等，这样呈现给用户的是修改之后的属性值。

整个处理流程： 注入依赖🡺开始初始化bean🡺调用自己的依赖注入之后行为init-method🡺初始化结束🡺调用自身销毁bean之前destory-method

**容器后处理器：执行在初始化容器之后，实例化Bean之前**

实现BeanFactoryPostProcessor接口，同上，可以实现预先读取properties文件的配置，与spring相辅相成。

**资源访问：Spring改进了原生Java访问资源文件的策略**

提供实现Resource接口的针对多种资源访问的不同类

UrlResource: http: file: ftp:资源都可以

ClassPathResource:访问WEB-INF/classes/下的文件

FileSystemResource：本地文件系统下文件，与File一样

ByteArrayResource：读取字节数据byte[]

ServletContextResource：访问WEB-INF/下文件，比如在jsp页面中是无法访问WEB-INF的资源的，但可以通过这个类访问

Spring提供两个标志性接口： 使得开发者不用自己选择使用上述哪个Resource，ResourceLoader默认跟着ApplicationContext类型(new ClassPathXmlApplicationContext() / FileSystemXmlApplictionContext()等创建方式和Resource一样)，也可自己确定使用哪个在getSource(“file:spring.xml classpath:spring.xml”)即可

ResourceLoader //ApplicationContext已经实现了此接口，所以ctx.getResource()可以直接用

ResourceLoaderAware和BeanFactoryAware用法一样，会自动传入一个ResourceLoader给bean使用, bean.getResourceLoader()就可以获取ResourceLoader，通常ApplicationContext已经实现了此接口，所以系统会传入ApplicationContext对象。

利用依赖注入引入资源：之前不管是直接用Resource类，还是通过接口ApplicationContext获得，总是会把资源耦合到java代码中

<bean id=”” class=””> //在对应的bean类中，添加成员变量Resource和对应的setter方法

<property name=”res” value=”classpath:spring.xml”>

**另一核心机制：AOP**

目的：程序员不用修改源代码的情况下，为java类添加新的功能，但其本质仍是修改源代码，只不过是由AOP框架来修改，编写的.java文件不变。

有两种实现机制：

静态AOP，AspectJ为代表，在编译.java文件时把Advice编译入.class文件。需要下载特殊编译器和核心jar

动态AOP , Spring AOP为代表，在运行阶段生成AOP代理(增强处理+目标对象方法)，Spring Ioc负责管理Spring AOP，所以不需要额外处理

工作流程： 定义普通业务组件🡺定义切入点（pointcut,符合切入点的连接点joinpoint处（目前spring只支持方法调用）会增加Advice）🡺定义增强处理(Advice，在Aspect切面中定义)

**基于注解的实现方式：Spring AOP利用@AspectJ一样的注解规范**

需要添加这三个类库支持

在xml中开启对切面支持：

<bean class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator">

注解语法：

@Aspect修饰Bean类，Bean后处理器会忽略，spring自动把该Bean作为切面类

@Before修饰方法:@Before("execution(\* beans.\*.\*(..))") //匹配beans下所有类的所有方法

@AfterReturning会在目标方法执行之后织入，若是目标方法返回值类型和Advice中一样，则在Advice方法中可以利用return形参访问目标方法返回值

@AfterReturning(returning="result",pointcut="execution(\* beans.\*.\*(..))")

public void authority(String result){

System.out.println("检查"+result);

}

@AfterThrowing //throwing=””若目标方法抛出的异常和Advice中一样，则Advice中可利用throwing形参访问目标抛出异常

@After 上面afterreturn只能在程序正确返回才会执行，但这不管正确与否都会执行，常用与资源回收，和finally类似

@Around //上面涉及的注解无法影响到目标方法（除非Advice方法参数个数不匹配报错！），@Around可以修改目标参数，控制目标执行时机等: jp.proceed(args) //执行之后目标方法才会执行，且参数用这里args代替原来的

Advice方法中获取目标方法参数：Advice方法第一个参数设置为JoinPoint类型，系统自动传入JoinPoint，然后通过Object[] getArgs(),Object getTarget()//返回目标对象 等方法访问

两个Aspect需要在同一个连接点织入时，系统默认随机，若是要排序则Aspect继承Order接口，或者用@Order修饰Aspect。

定义切入点pointcut：使得切入点表达式复用，由方法签名的public修饰符决定范围

@Pointcut("execution(\* beans.\*.\*(..))")

public void myPointcut(){}

@Around("AuthAspect.myPointcut()") //这里切入点表达式就用的上面定义的

语法：set\*later表示set开头later结尾， \* 单独表示也可

方法修饰符（可略）+“space”+方法返回值类型（\*可用）+”space“+方法所在类（可略 \* 可用）+方法名(\*可用)+ 参数列表（(..)/(\*,String,Integer)）//如pubic void java.util.play(\*\*)

**Spring缓存机制：基于注解实现，相比Hibernate二级缓存，其级别更高，可以在任意Bean类或者方法缓存，通常对Service类的方法缓存，缓存的是方法执行结果。**

spring.xml配置步骤：

1导入cache命名空间：

xmlns:cache=”<http://www.springframework.org/schema/cache>”

xsi:schemaLocation="http://www.springframework.org/schema/cache

http://www.springframework.org/schema/cache/spring-cache.xsd”

2通知spring需根据注解来启动缓存："cacheManager"是默认值，代表xml中缓存管理器id

<cache:annotation-driven cache-manager="cacheManager"></cache:annotation-driven>

3使用第三方EhCache缓存机制：将Hibernate\lib\optional\ehcache\下的jar包放到类加载路径，src\下添加ehcache.xml配置文件，里面需要配置和命名缓存区名称（重要）

Spring使用EhCacheCacheManager管理EhCache，只需在spring配置其即可自动使用EhCache，但其实现依赖于CacheManager（EhCacheManagerFactoryBean系统自带的工厂Bean可以返回此对象，属性中会用到encache.xml）。若hibernate也用ehcache.xml，则shared属性必须设置成true，表示共享缓存。

<bean id="ehCacheManager"//产生CacheManager class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">

<property name="shared" value="**true**"></property>

<property name="configLocation" value="classpath:ehcache.xml"></property>

</bean>

//id号必须和步骤2中指定的一样

<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">

<property name="cacheManager" ref="ehCacheManager"></property>//依赖注入

</bean>

4、或者使用spring自带的缓存机制：

<bean id="cacheManager" class="org.springframework.cache.support.SimpleCacheManager">

<property name="caches">

<set>

<bean class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean">

<property name="name" value="default"></property>

</bean>

<bean class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean">

<property name="name" value="Voronoi"></property>

</bean>

</set>

</property>

</bean>

**使用@Cacheable注解：**

1. 修饰类缓存: 当调用类的方法时参数相同即调用缓存中结果(即使调用不同方法，但参数一样也适用)，默认把所有参数作为key，key可以自定义修改,key相同则不会执行真正方法，直接用缓存区里的值。@Cacheable(value={”user”}//保存在哪个缓存区 key=”#name”//修改key为name， condition=”#age>50”//表示满足这个条件才会写入缓存)。 这样的特性使得更多的情况下使用修饰方法缓存，灵活性更高。
2. 修饰方法缓存：传入参数作为key，key相同就直接调用缓存区结果，其他功能同上。

**使用@CacheEvict注解：**

修饰方法：当调用这个方法时，主动清除对应的缓存区数据。

**Spring事务：之前的事务控制都是通过代码创建开启，事务耦合到代码中了，这里把事务作为Bean，事务所依赖的datasource利用注入配置**

Spring中通过PlatformTransactionManager接口（getTransation开始事务，commit提交事务，rollback回滚事务）操作事务，具体实现类则交给spring提供，这样使得操作事务代码不用变化，只需要注入不同事务对应的实现类

<bean id="transactionManager" //依赖注入hibernate操作事务所需SessionFactory-Bean

class="org.springframework.orm.hibernate4.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

2种事务管理方式：

编程式管理：在Dao中通过ApplicationContext获取上述transactionManager-bean，然后使用接口的三个函数

声明式管理：利用AOP代理方式为Dao方法执行之前织入事务，方法结束之后结束事务，这里通过xml实现，不是写一个Aspect类实现。效果使用Dao组件自动开始事务。

Spring2.0提供了简介的配置策略，利用命名空间管理

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx=”<http://www.springframework.org/schema/tx>”

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.0.xsd

http://www.springframework.org/schema/tx

<http://www.springframework.org/schema/tx/spring-tx-4.0.xsd>

<!-- 配置事务增强处理，指定事务管理器 -->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<tx:attributes>//这里有很多事务属性可以设置

<tx:method name="get\*" read-only="true"></tx:method>

<tx:method name="\*" isolation="DEFAULT" timeout="5"></tx:method>

</tx:attributes>

</tx:advice>

<!-- 配置AOP -->

<aop:config>

<!-- 配置切入点 -->

<aop:pointcut id="myPiontcut" expression="execution(\* daos.\*.(..))"></aop:pointcut>

<!-- 在切入点需要使用的增强处理 -->

<aop:advisor advice-ref="txAdvice" pointcut-ref="myPointcut"></aop:advisor>

</aop:config>

**Spring整合Struts2 : 最大好处利用spring为action依赖注入所需业务逻辑组件(Service)**

需要在web.xml中配置spring容器:

<listener> //spring提供该监听器，使得web创建时能够自动根据给定的spring.xml创建出WebApplicationContext对象，保存在web的ServletContext（代表当前web对象）中

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

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

<param-value>WEB-INF/classes/spring.xml</param-value>

</context-param>

让Spring负责产生Action（Spring作为Action的大工厂），即Struts2控制器拦截到用户请求，然后由Spring调用对应Action。

需要添加struts2的一个spring插件（struts2-spring-plugin-2.3.16.3.jar）支持，在struts.xml中<action class=“Bean\_id”>，制造伪Action，实际上由spring容器产生，这些系统自动完成。

这里注意，action代表一个请求，不能是单例模式，scope=”prototype/request”

**Spring整合Hibernate:最大好处Spring管理Dao组件操作持久化对象**

Hibernate实现持久层访问需要通过SessionFactory对象来开启事务，现在spring.xml使用声明式添加(通过hiernate.xml配置文件生成)

<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<property name="configLocations" value="classpath:hibernate.cfg.xml"></property>

</bean>

Dao组件设计模式：

先设计一个BaseDao<T>接口，抽取出通用方法，然后编写实现类BaseDaoImpl<T>，里面添加成员变量SessionFactory,和对应的setter方法(依赖注入需要)，具体实现的BookDao也先定义出接口继承BaseDao<Book>,添加独特的处理逻辑方法，然后具体实现类则再extends BaseDaoImpl<Book>，实现具体的方法。

Spring.xml编写方式：

在spring.xml文件中通常会把Dao、Action、DataSource等不同范围的类分开放到不同的spring.xml文件中，通过<import resource="\*\*\*.xml"/>互相调用。

下面这些都需要写入spring.xml当作Bean

Action组件：struts2拦截器拦截用户http请求，交给对应的action处理，依赖service注入，通过service完成控制操作，要是还要把得到的数据给jsp，那么action中要有相应的成员变量作为中介，相当于添加到request，而jsp🡺action🡺jsp(一次请求)这中间不需要中间变量，一直在request对象中，若再到action则需要在前jsp中把数据作为input提交一下(新的request)。

Service组件：逻辑业务组件(利用Dao方法+Domain方法，可使数据库相关操作分离)，依赖Dao注入，完成真正逻辑上的复杂功能(多个Dao)，面向Dao接口编程，系统有多少个业务方法，service组件功能函数就有多少个。

Dao组件：封装了针对Model整体在数据库增删改查等基本功能（如依靠hibernate下的session完成get(T)，update(T)，save(T)等操作），是数据库原子操作，依赖sessionfactory注入，一个model对应一个Dao，在再添加自己的特有的一些方法用来数据库操作(参数是对应model中的属性，返回值是List<Manager>类似)。

还需要的层次：

Model组件(domain层，或者PO对象)：持久化对象，需要用注解修饰映射到数据库中，在hibernate.cfg.xml中添加。

VO组件：值对象，通过new创建，gc销毁，专门帮助service层所需数据提供生存的地方(List<\*\*Bean>这样就可以让产生的数据寄存到一个对象中),vo对象属性都是在action中呈现给jsp的，PO对象用于数据库表关联不方便给service提供业务特定的数据。VO(PaymentBean)和PO(Payment)结构相同，意义不同，两者属性也自个根据需要不一定相同。