SSM

**spring部分**

**1.Spring定义:**

  spring是一个开源的轻量级的应用开发框架,其目的用于企业级应用程序开发,减少侵入(降低耦合)

  spring可以整合市面常见的开发框架

**2.耦合分为三种:**

零耦合          没有关系

抽象耦合        类里耦合的是接口或抽象类

集体耦合        类里耦合的是具体的实现类

**3.IOC**:Inversion of Control 控制反转,就是控制权的转移,即把创建(new)对象的权利,反转给第三方spring框架去创建(new)

Bean是Spring的核心、基础、根源。

**4.spring加载过程**

启动spring容器 ApplicationContext context =

new ClassPathXmlApplicationContext("hello/applicationContext.xml");

1). 读取hello文件下的applicationContext.xml,

2). Xml解析applicationContext.xml文件

3). 把解析xml的内容存储到内存的Map集合中

4). 从Map集合中读取集合内容,就是清单的内容

5). 遍历集合中的所有数据,并反射实例化对象 Class.forName(“hello.User”).newInstance()

从容器中取出对象

   User user=spring容器.getBean(“容器中对象的唯一id”);

使用对象

   user.sayHello();

**5.创建对象的4种方式:**

1)默认构造: 默认无参构造

2) 静态工厂设计模式 静态方法不需要创建类

有些类不能直接实例化，就不能直接通过new关键字来创建，类似这样的对象我们要通过静态工厂模式来创建对象。

3)实例工厂设计模式

静态工厂设计模式不会创建类对象，直接调用静态方法。而工厂方法设计模式会创建类，然后通过xml中配置去访问其方法。

4)spring工厂

这是spring框架自身提供的，它需要实现FactoryBean接口，实现代码就必须写在getObject()方法中。

public class SpringFactory implements FactoryBean<Calendar>{

public SpringFactory() {

System.out.println("我是一个spring工厂类"); }

public Calendar getObject() throws Exception {

return Calendar.getInstance(); } }

**6.spring创建对象是单例还是多例** scope=singleton 单例 prototype多例

我们从spring容器中获取对象，实例化成两个对象，两个对象地址相同说明对象是单例的。默认条件下，spring中的bean创建的对象是单例。单例对象有好有坏：

好处，spring创建后，那就不会再频繁创建，缓存在map中,省内存。

坏处, 对象里的数据不安全,即线程不安全

默认情况下，spring中创建的对象都是单例，并且维护其生命周期。单例对象的生命周期与spring容器共命运，同生共死。---spring容器中

但如果对象是多例的，那么Spring容器只负责对象的创建，不负责维护其生命周期，也就是说如果容器关闭，对象并未销毁，需要用户自行关闭。 ---内存中

**7.懒加载**  lazy-init="true" true为懒加载

单例有懒加载 非懒加载 单例默认都是直接创建对象,非懒加载

多例都是懒加载 懒加载加快了spring启动时的速度,没有用到时就不会创建对象，就不会占用内存

default-lazy-init="true"全局配置

**8.对象的初始化和销毁**

xml中配置 init-method='init' destroy-method='destroy'

单例情况下:spring容器关闭时触发销毁方法,单独调用销毁方法无效,单例是spring容器管理的

多例情况下:spring不管其销毁.

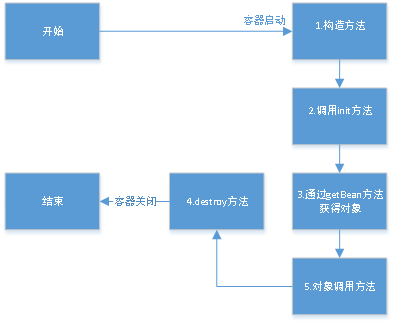
**9.pring 容器执行过程**

1).new Instance ----调用构造方法创建对象

2).Init-method ------执行初始化方法

3).对象调用方法。

4).容器关闭，执行销毁方法 ，如果scope=”property”时让其不负责对象的销毁



Spring对象生命周期

**10.DI依赖注入**

IoC反转控制只完成了单个对象的创建，需要配合DI依赖注入,在spring容器中实例化完的对象,注入到需要对象的地方,

**11依赖注入的方式**

**1)set方法注入**

**(1)单值注入**

<property name="name" value="wt"/> 注意：ref和value不能同时存在

(2)对象注入

<property name="dog" ref="dog"></property> <bean id="dog" class="hello.Dog"/>

(3)直接集合注入 支持List、Set、Map、Property

<property name="list">

<list>

<value>北京</value>

</list>

</property>

(4)间接集合注入

 <util:set id="uSet">

     <value>北京</value>

 </util:set>

(5)spring的表达方式注入 #

<util:properties id="jdbc"  location="classpath:mysql.properties"></util:properties>

<bean id="jdbcObj" class="hello.Jdbc">

     <property name="className" value="#{jdbc.driverClass}"></property>

     <property name="url" value="#{jdbc.url}"></property>

     <property name="userName" value="#{jdbc.userName}"></property>

     <property name="userPassword" value="#{jdbc.userPassword}"></property>

    </bean>

注意需要导入util的命名空间,建议此配置放在配置文件头上

(6)spring的表达方式注入 $

<context:property-placeholder  location=”classpath:mysql.properties”>

<bean id="jdbcObj" class="hello.Jdbc">

     <property name="className" value="${driverClass}"></property>

     <property name="url" value="${url}"></property>

     <property name="userName" value="${userName}"></property>

     <property name="userPassword" value="${userPassword}"></property>

    </bean>

(7)注入null或空字符串”” ""???

<property name="email">

<null />

</property>

**2)构造方法注入**

<constructor-arg index="0" value="tina"/>

在实际的开发中,用setter方式占有绝大多数,构造注入方式少

构造函数注入方式,在框架的源代码中应用居多,也有setter注入方式

**12.spring的三层架构**

controller+service+dao

13.**继承** <bean id="cartDao" class="parent.CartDao" parent="baseDao" />spring中的继承和java中的继承是不同的。

**抽象类**<bean id="baseDao" class="parent.BaseDao" abstract="true">

**自动装配:**

**1)byName**

<bean id="user" class="autowired.old.User" autowire="byName"/>

byName的处理过程是，先去找所有的set方法，setXyz去掉set，首字母变小写xyz，然后去找对应的属性，通过属性名称去配置文件中找id=xyz的，通过id再找到具体的类，然后进行set设置完成自动注入。byName是按bean的id名称对应，其实叫byId才更准确。

**2)byType**

byType按class装配

**@Autowired**

public class User {

@Autowired

private Cat cat; } 可以省略set方法,是按byType进行匹配。

**@Qualifier**

@Autowired不支持id匹配，如果按byName匹配怎么做？就要用到@Qualifier注解。

@Autowired @Qualifier(value="carA") //可以写在一行上也可以分开

**@Resource**

就想填入一个注解，byType、byName自由选择呢？spring没有给我们，jdk给了我们，jdk的@Resource注解。

@Resource(name="carA")

@Resource虽然更加规范不会依赖spring框架，但如果使用spring框架的话还是推荐使用@Autowired，实际开发中会有细小差异

**@Repository**   标记在dao持久层类上  也可以用@Component

**@Component**   标记在无法分类的类(util等工具类上)

**14 @Autowired与@Resource的区别**

1、@Autowired与@Resource都可以用来装配bean。都可以写在字段上，或写在setter方法上。

2、@Autowired默认按类型装配（属于spring），默认情况下必须要求依赖对象必须存在，如果要允许null 值，可以设置它的required属性为false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier注解进行使用。

3、@Resource（属于J2EE），默认按照名称进行装配，名称可以通过name属性进行指定。如果没有指定name属性，当注解写在字段上时，默认取字段名进行按照名称查找，如果注解写在setter方法上默认取属性名进行装配。 当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

它们的作用相同都是用注解方式注入对象，但执行顺序不同。@Autowired先byType，@Resource先byName。

**15包扫描**

在类上加上@Component 配置文件中<context:component-scan base-package="annotation"/>

执行过程：加载spirng核心配置文件，遍历到<context-component:scan，去指定的目录下找到所有含有@Component的类，按之前<bean声明的规则创建对象，保存在map中，key=类名首字母小写（实例名称），value=创建的对象。这样就无需大量的xml的配置，而且非常自动化，只要在规定的路径下添加新的类。

**16.BeanNameAware** spring提供了接口BeanNameAware可以查看类的id名称。

@Component

public class CIA implements BeanNameAware{

public void kill(String name){

System.out.println("Kill " + name); }

public void setBeanName(String name) {

System.out.println("当前的类的id：" + name); } }

<context:component-scan base-package="inspect"/>

Spring中的ID的生成策略看类中的第二个字母,如果第二个字母是大写的.则首字母不变.如果第二个字母小写,则首字母小写

**17.其他一些常见注解**

1).@Scope(value="prototype") 单例和多例

2).@Lazy //代表懒加载

3).@PostConstruct //表示初始化操作

4).@PreDestroy //表示销毁

**18.代理设计模式**

AOP 实现的关键就在于AOP框架自动创建的AOP代理，AOP代理则可分为静态代理和动态代理两大类，其中静态代理是指使用AOP框架提供的命令进行编译，从而在编译阶段就可生成AOP代理类，因此也称为编译时增强；而动态代理则在运行时借助于JDK动态代理、CGLIB等在内存中“临时”生成AOP动态代理类，因此也被称为运行时增强。

静态代理有：编译时织入（特殊编译器实现）、类加载时织入（特殊的类加载器实现）。

动态代理有：jdk动态代理（基于接口来实现）、CGlib（基于类实现）。

**静态代理**:和被代理类实现相同的接口,完成本职工作的同时,还要完成额外的操作.

优点:1.实现了业务层代码和事务层代码相分离

缺点:1.大量的代码重复,复用度低2.一个静态代理类只能代理一个类做法不通用。

动态代理三要素：proxy代理类，InvocationHandler接口，Invoke方法。

**jdk动态代理:**通过反射进行invoke回调,要求业务类必须有接口，没有接口无法生成代理对象

Proxy是JDK中用来创建代理对象的类，其中使用最多的是newProxyInstance方法来创建代理对象。代理类与目标类是兄弟关系.

public static Object newProxyInstance(

ClassLoader loader, //由哪个类加载器加载生成代理对象

Classs<?>[] interfaces, //接口对象数组，必须传入目标对象的接口

InvocationHandler h //表示一个invocationHandler对象，当代理对象执行方法时，会调用改````InvocatonHandler对象 ) throws IllegalArgumentException

Object proxy = Proxy.newProxyInstance(target.getClass().getClassLoader(), target.getClass().getInterfaces(),

// 采用匿名内部类方式，参数必须final

new InvocationHandler() {

// 代理对象，真实对象的方法，真实对象的参数

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

Object result = null;

Long beginTime = System.currentTimeMillis();

try {

tx.begin();

// 通过反射调用真实对象的方法

result = method.invoke(target, args);

tx.commit();

} .....return result; }});return proxy}

优点:

1.继承了静态代理的优点

2.能够解决代码重复问题

3.可以使用动态代理来处理一类业务

缺点:

1.要求必须实现接口,否则不能生成代理对象

2.如果指定方法较为麻烦.

**cglib动态代理:**不要求业务类必须有接口，有实现类即可,cglib使用继承，因此父类最好不要有final属性，final阻止继承和多态。代理类与目标类是继承关系,cglib创建的类都是目标对象的子类

(1)使用CGLib实现动态代理，CGLib底层采用ASM字节码生成框架，使用字节码技术生成代理类，比使用Java反射效率要高。唯一需要注意的是，CGLib不能对声明为final的方法进行代理，因为CGLib原理是动态生成被代理类的子类。

(2)在对JDK动态代理与CGlib动态代理的代码实验中看，1W次执行下，JDK7及8的动态代理性能比CGlib要好20%左右。

Spring提供了两种方式来生成代理对象: JDKProxy和Cglib，具体使用哪种方式生成由AopProxyFactory根据AdvisedSupport对象的配置来决定。默认的策略是如果目标类是接口，则使用JDK动态代理技术，如果目标对象没有实现接口，则默认会采用CGLIB代理。如果目标对象实现了接口，可以强制使用CGLIB实现代理。（添加CGLIB库，并在spring配置中加入<aop:aspectj-autoproxy proxy-target-class="true"/>）

**19动态代理缺点**

静态代理太死板，所以实际中使用不多。而动态代理非常灵活，有接口通常使用jdk动态代理方式，没有接口通常采用cglib方式。虽然动态代理已经很不错，代码分离，实现松耦合，比较通用，那还有没有改进的地方呢？

有没有发现比较霸道，如果一个类被它代理，则接口或者类的所有方法都被强迫执行，我们实际中组织类不是按事务需求，而是按业务划分，那必然有些方法需要事务，有些方法不需要。动态代理可不管这么多。再者我们还需要写一个动态代理类来完成，对开发者暴露了部分开发的内容，封装性不是很彻底。

**20面试：有了OOP为什么还要AOP？**

上面动态代理控制力度太粗的缺点只是从细节去说，下面跳出编程细节的圈圈，往更高层次看看。

AOP（Aspect-Oriented Programming，面向切面编程），可以说是OOP（Object-Oriented Programing，面向对象编程）**的补充和完善**。OOP引入封装、继承和多态性等概念来建立一种对象瀑布层次结构，用以模拟公共行为的一个集合。当我们需要为分散的对象引入公共行为的时候，OOP则显得无能为力。也就是说，OOP允许你定义从上到下的关系，但并不适合定义从左到右的关系。例如日志功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。对于其他类型的代码，如安全性、异常处理和透明的持续性也是如此。这种散布在各处的无关的代码被称为**横切（cross-cutting）代码。**

**在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。而AOP技术则恰恰相反，它利用一种称为“横切”的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其名为“Aspect”，即切面。**

所谓“切面”，简单地说，就是将那些与业务无关，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。AOP代表的是一个横向的关系，如果说“对象”是一个空心的圆柱体，其中封装的是对象的属性和行为；那么面向切面编程的方法，就仿佛一把利刃，将这些空心圆柱体剖开，以获得其内部的消息。而剖开的切面，也就是所谓的“切面”了。然后它又以巧夺天功的妙手将这些剖开的切面复原，不留痕迹。使用“横切”技术，AOP把软件系统分为两个部分：**核心关注点和横切关注点**。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处都基本相似。比如权限认证、日志、事务处理。Aop 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。简而言之，AOP的核心思想就是“将应用程序中的商业逻辑同对其提供支持的通用服务进行分离。”

**OOP实现了纵向复用，而AOP实现了横向复用，纵向+横向无敌！**

**21实现AOP的技术分为两大类:**

一是采用动态代理技术,利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；

二是采用静态织入的方式,引入特定的语法创建"方面",从而使得编译器可以在编译期间织入有关"方面"的代码;

AOP面向切面编程,AOP包括切面（Aspect）、连接点（JoinPoint）、通知（Advice）、切入点（PointCut）、目标对象（Target）实现方式就是通过对目标对象的代理在连接点前后加入通知，完成统一的切面操作。

**AOP(Aspect-Oriented Programming, 面向切面编程):** 是一种新的方法论, 是对传统 OOP(Object-Oriented Programming, 面向对象编程) 的补充.,AOP 的主要编程对象是**切面(aspect)**, 而切面模块化横切关注点.,在应用 AOP 编程时, 仍然需要定义公共功能, 但可以明确的定义这个功能在哪里, 以什么方式应用, 并且不必修改受影响的类. 这样一来横切关注点就被模块化到特殊的对象(切面)里.

**AOP 的好处:**

每个事物逻辑位于一个位置, 代码不分散, 便于维护和升级

业务模块更简洁, 只包含核心业务代码.

**AOP的调用过程**

当用户获取对象时,首先会与**切入点**做匹配,如果匹配成功则为其创建代理对象.当代理对象调用方法时即连接点.就会调用与切入点**绑定的通知**方法.

**切面(Aspect)**: 就是一个java类,可以理解为一个新功能类,这个新功能类中有若干通知方法,每个通知 方法代表一个功能

**通知(Advice):** 在切面的某个特定的连接点上执行的动作。切面中的**方法**(主要实现额外的操作)

**连接点（Joinpoint）**：用代理对象调用目标的方法的那句话的位置就叫做连接点

**切点（pointcut）(切入点)**根据指定表达式匹配哪些连接点,指定把什么新功能横切到什么位置上

切入到方法上,而不是类上

**目标对象**（Target Object）：就是原有业务的对象

前置通知（Before advice）：在目标方法执行之前执行

后置通知（After returning advice）：在目标方法执行之后 执行

异常通知（After throwing advice）：目标方法抛出异常后 执行

最终通知（After (finally) advice）：在目标方法执行之后都会执行

以上的四大通知都不能管理目标方法是否执行.

环绕通知（Around Advice）：在目标方法执行之前执行,执行之后执行.只有环绕通知能够控制目标方法的执行.环绕通知功能最为强大

<aop:after-throwing method="afterThrow" pointcut-ref="pc" throwing="throwable"/>

**ProceedingJoinPoint和JoinPoint的区别**

ProceedingJoinPoint对象是JoinPoint的子接口,该对象只用在@Around的切面方法中,

环绕通知 ProceedingJoinPoint 执行proceed方法的作用是让目标方法执行，这也是环绕通知和前置、后置通知方法的一个最大区别。

是在JoinPoint的基础上暴露出 proceed 这个方法。proceed很重要，这个是aop代理链执行的方法。

**22.切点表达式:**

1).within(包名.类名)

|  |  |
| --- | --- |
| **within(aop.service.UserServiceImpl))** | **指定类，只能指定一个类** |
| **within(aop.service.\*))** | 只包括当前目录下的类 |
| **within(aop.service.\*.\*))** | 只包括当前目录下的类 |
| **within(aop.service..\*))** | 指定当前目录包含所有子目录中的类 |

2).execution(返回值类型 包名.类名.方法名 (参数类型...))

建议使用 execution(\* service..\*.\*(..)) 切入点表达式*service 不要省略*

<aop:pointcut expression=

*"execution(int service.UserServiceImpl.add())"* id=*"txPointcut"*/>



|  |  |
| --- | --- |
| **execution(void aop.service.UserServiceImpl.addUser())** | **匹配方法** |
| **execution(void aop.service.PersonServiceImpl.addUser(String))** | 方法参数必须为字符串 |
| **execution(\* aop.service..\*.\*(..))** | 万能配置 |

23.织入Weaving--把原有业务和新功能业务织入在一起

**java织入提供3个织入点**：

编译期：在Java编译期，采用特殊的编译器，将切面织入到Java类中。

类加载期：通过特殊的类加载器，在类字节码加载到JVM时，织入切面。

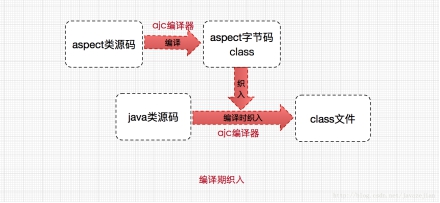
运行期：采用CGLIB或JDK动态代理进行切面的织入。

**AspectJ织入提供3个织入点：**

编译期：把aspect类（切面）和目标类放在一起用ajc（特殊编译器类似javac编译期）编译。

后编译期：目标类可能已经打成jar包，这时候也可以用ajc命令将jar在织入一次。

类加载期：Load-Time Weaving LTW，在jvm加载类的时候，做字节码的替换。



其中前2个时间点，可以理解为静态织入，因为在class文件生成后，就已经织入好了。类加载期织入，可以理解为动态织入，因为这个类替换是在jvm加载类的时候完成的。

类加载期织入又有3种方式：agent方式，启动时加参数替换默认的系统类加载器，自定义类加载器。

24.AOP注解方式

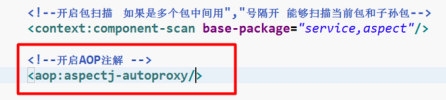
@Before 前置通知

@Around 环绕通知

@AfterThrowing 异常通知

@AfterReturning 后置通知

@After 最终通知





25.AOP应用场景

**1)异常处理 --xml方式**

异常处理类和业务逻辑类完全松耦合。时刻捕获生产生产环境中所有的错误，实时监控该系统，异常收集。

**2) 性能监控 --xml方式**

需求：计算service的各个类的各个方法的执行时间，也就是耗时

打印信息包括：类名、方法名、耗时

分析得出这个需求只能使用环绕通知的方式，要在业务执行前记录开始时间，在业务执行后记录完成时间，两个时间相减就是耗时，耗时过长的业务可能就存在性能问题。

**3) 权限控制 --注解方式,有权限输出,无权限提醒**

需求：对类的方法进行控制，允许的才能访问，不允许的就提示权限不足。分析得出先需要有权 限控制列表，然后在切面中获取执行的方法的注解，判断其是否在允许列表中，允许则业务继续执行，不允许则不能执行业务，同时提示权限不足。

这个需求也只能使用环绕通知，因为只有环绕通知才能改变执行的行为。

**4) 事务管理 --注解方式**

26spring整合jdbc

**springmvc部分**

**1.springmvc介绍**

作用是 浏览器 与  后台程序 进行交互的。

SpringMVC是一个前端控制框架,主要用来负责与页面的交互.SpringMVC是Spring家族的一大组件.Spring整合SpringMVC可以做到无缝集成

**2为什么要学学习springmvc**

**1)Servlet的缺点**

l 每个Servlet声明时需要在web.xml中配置8行代码，配置繁琐，内容多后，web.xml结构不清晰，不易维护。团队开发冲突不断。

l 每个Servlet只有一个执行方法，如果要实现数据库表的CRUD操作，需要写4个Servlet。

l 获取参数繁琐只能强制转类型，复杂的还需判断如日期类型转换，代码繁多重复手工劳动，非业务代码。

**2)Struts2的缺点**

l 学习成本高，学习难度大，很多复杂的原理

l 值栈对象结构复杂，每个request创建时会自动创建值栈对象，效率低

l action声明xml配置，result返回值配置，配置多，团队开发冲突大，没有校验，常会因为配置出错，开发效率低。

l OGNL表达式非常复杂，极其难以掌握，性能低，不安全，全球性安全事故频出。

**3.Struts2和springmvc的本质区别**

 1.springmvc入口是一个servlet前端控制器(DispatcherServlet)

struts2入口是一filter过滤器(StrutsPrepareAndExecuteFilter).

        2.struts2通过在action类中定义成员变量接收参数,(属性驱动和模型驱动),它只能使用多例模式管理action.

 springmvc通过在controller方法中定义形参接收参数,springmvc可以使用单例模式管理controller.

        3.springmvc是基于方法开发的,注解开发中使用requestMapping将url和方法进行 映射,如果根据url找到controller类的方法生成一个handler处理器对象(只包括一个method).

struts2是基于类开发的,每个请求过来创建一个action实例,实例对象中有若干个方法.

开发中建议使用springmvc,springmvc方法更类似service业务方法.

        4.struts2采用值栈存储请求和相应的数据,通过OGNL存取数据,

springmvc通过参数绑定期将request请求内容解析,并给方法形参赋值.

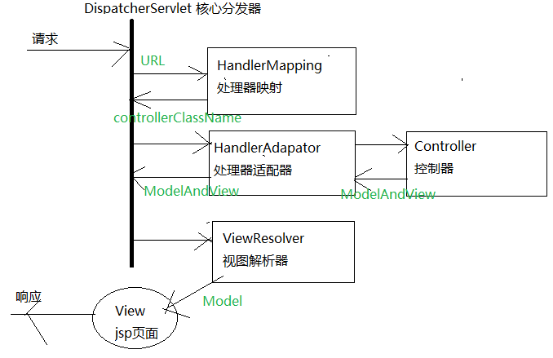
        5.struts2和springmvc的速度是相当的,由于struts2的漏洞较多,跟多企业使用springmvc

**springmvc和struts2比较**

|  |  |  |  |
| --- | --- | --- | --- |
| **技术** | **核心分发器** | **拦截级别** | **说明** |
| **Struts1** | DispatcherServlet | 类级别 | 基于Servlet实现  企业中很多旧项目采用的框架  action是单例模式，线程不安全的。  Struts1使用JSTL EL表达式，但是对集合和索引属性的支持很弱。 |
| **Struts2** | DispatcherFilter | 类级别  一个类对应一个request上下文 | 基于Filter实现  Struts2 action是原型模式 prototype，每次访问对象都会创建新的实例，保证线程安全性；  采用 OGNL解析页面标签。  Struts2是基于松耦合，和web容器脱钩ValueStack  复杂值栈、多例、OGNL导致性能低  安全漏洞频繁，不安全 |

|  |  |  |  |
| --- | --- | --- | --- |
| **SpringMVC** | DispatcherServlet | 方法级别  一个方法对应一个request上下文，而方法同时又跟一个url对应。 | 基于Servlet实现  Springmvc controller是单例模式，整个程序只有一个对象实例。Spring的安全性是通过绑定threadlocal实现  Spring3 mvc可以认为已经99.9%零配置了。  采用JSTL解析页面标签    基于web容器、单例、JSTL导致性能高 |

**4.springmvc的原理---springmvc的四大组件**



前端控制器DispatcherServlet--请求转发的核心控制器。

处理器映射器HandlerMapping--找到url中的请求，找到方法名。

处理器适配器HandlerAdaptor--主要用来处理请求对应的方法。

处理器Controller

视图解析器ViewResolver--将用户返回的页面名称为其拼接真实的路径 ,进行渲染

**5.步骤**

1)创建jsp页面

2).导jar包

3).web.xml中配置DispatcherServlet

4).配置springMVC的核心配置：spring\_mvc.xml

5)创建controller

**6.springmvc注解方式**

从springmvc2.5开始引入注解方式，特别到了**3.0**就全面引入注解方式，号称**xml零配置**。spring3.0配置注解引入后也就是这个点成为了它和struts2的分水岭。随着springmvc的成熟，struts2开始落幕，趋于被市场淘汰。

注解的工作原理:启动tomcat，加载项目的web.xml创建并初始化spring容器，同时创建springmvc容器。创建完springmvc容器后，进行包扫描，扫描所有的controller。找到配置的包路径下的所有含有@Controller的类，然后扫描这个类的所有@RequestMapping的映射，把这个注解形成一个map，这个map的key就是@RequestMapping中的字符串，val就是@RequestMapping所在类的所在方法。然后当用户在浏览器访问某个链接url时，先经过Servlet映射，如action结尾的，满足要求后进入DispatcherServlet。按映射规则去掉url中的协议、端口等，最终留下servlet映射后的部分，去map中找有无对应的key，如果有对应的key，就找到了对应类的对应的方法，就执行这个方法；如果没有找到，就抛出404错误。

**7.RESTFul架构支持--**Representational State Transfer 表现层状态转换 ,表现层就是资源

简单来说:RESTful是一种架构的规范与约束、原则，符合这种规范的架构就是RESTful架构。

RESTFul起因

REST这个词，是Roy Thomas Fielding在他2000年的博士论文中提出的。

Fielding是一个牛人，他是HTTP协议（1.0版和1.1版）的主要设计者、Apache服务器软件的作者之一、Apache基金会的第一任主席。所以，他的这篇论文一经发表，就引起了关注，并且立即对互联网开发产生了深远的影响.

get：获取资源 从没有到有资源

post：新建或更新资源

put：更新资源

delete：删除资源

------上面4中能体现资源的转换

（1）每一个URI代表一种资源；

（2）客户端和服务器之间，传递这种资源的某种表现层；

（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

要让一个资源可以被识别，需要有个唯一标识，在Web中这个唯一标识就是URI(Uniform Resource Identifier)。URI既可以看成是资源的地址，也可以看成是资源的名称。如果某些信息没有使用URI来表示，那它就不能算是一个资源， 只能算是资源的一些信息而已。URI的设计应该遵循可寻址性原则，具有自描述性，需要在形式上给人以直觉上的关联。

uri设计的技巧

* 使用\_或-来让URI可读性更好
* 使用/来表示资源的层级关系
* 使用?用来过滤资源
* ,或;可以用来表示同级资源的关系

RESTFul结构是一种参数的格式

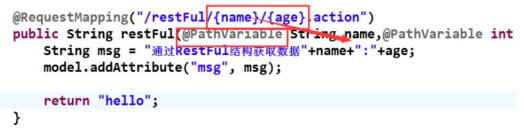
Url:localhost:XXXX/addUser.action?name=tom&age=18

get提交方式:

Url:localhost:XXXX/addUser/tom/18.action

RESTFul提交方式:

传值时值传递具体的值,多个值中间用"/"分割



restFul结构中的参数位置相对固定.

**8.redirect 重定向**

return "redirect:/user/userlist.action";

redirect方式相当于“response.sendRedirect()”,转发后浏览器的地址栏变为转发后的地址，因为转发即执行了一个新的request和response。

**forward转发**

forward方式相当于“request.getRequestDispatcher().forward(request,response)”，转发后浏览器地址 栏还是原来的地址。转发并没有执行新的request和response，而是和转发前的请求共用一个request和response。所以转发前请求的参数在转发后仍然可以读取到。

|  |  |
| --- | --- |
| 转发 | 重定向 |
| 服务器端完成 | 客户端完成 |
| 速度快 | 速度慢 |
| 同一次请求 | 两次不同请求 |
| 不会执行转发后的代码 | 会执行重定向之后的代码 |
| 地址栏没有变化 | 地址栏有变化 |
| 必须在同一台服务器中完成 | 可以在不懂服务器中完成 |
| request.getRequestDispatcher("new.jsp").forward(request, response); | response.sendRedirect("new.jsp"); |

**9.springmvc标签**

<%@ **taglib** uri="http://www.springframework.org/tags/form" prefix="sf" %>

**10.验证注解**

|  |  |  |
| --- | --- | --- |
| **验证注解** | **验证的数据类型** | **说明** |
| **@AssertFalse** | Boolean,Boolean | 验证注解的元素值是false |
| **@AssertTrue** | Boolean,Boolean | 验证注解的元素值是true |
| **@NotNull** | 任意类型 | 验证注解的元素值不是null |
| **@Null** | 任意类型 | 验证注解的元素值是null |
| **@Min(value=值)** | BigDecimal，BigInteger, byte,  short, int, long，等任何Number或CharSequence（存储的是数字）子类型 | 验证注解的元素值大于等于@Min指定的value值 |
| **@Max（value=值）** | 和@Min要求一样 | 验证注解的元素值小于等于@Max指定的value值 |
| **@DecimalMin(value=值)** | 和@Min要求一样 | 验证注解的元素值大于等于@ DecimalMin指定的value值 |
| **@DecimalMax(value=值)** | 和@Min要求一样 | 验证注解的元素值小于等于@ DecimalMax指定的value值 |
| **@Digits(integer=整数位数, fraction=小数位数)** | 和@Min要求一样 | 验证注解的元素值的整数位数和小数位数上限 |
| **@Size(min=下限, max=上限)** | 字符串、Collection、Map、数组等 | 验证注解的元素值的在min和max（包含）指定区间之内，如字符长度、集合大小 |
| **@Past** | java.util.Date,  java.util.Calendar;  Joda Time类库的日期类型 | 验证注解的元素值（日期类型）比当前时间早 |
| **@Future** | 与@Past要求一样 | 验证注解的元素值（日期类型）比当前时间晚 |
| **@NotBlank** | CharSequence子类型 | 验证注解的元素值不为空（不为null、去除首位空格后长度为0），不同于@NotEmpty，@NotBlank只应用于字符串且在比较时会去除字符串的首位空格 |
| **@Length(min=下限, max=上限)** | CharSequence子类型 | 验证注解的元素值长度在min和max区间内 |
| **@NotEmpty** | CharSequence子类型、Collection、Map、数组 | 验证注解的元素值不为null且不为空（字符串长度不为0、集合大小不为0） |
| **@Range(min=最小值, max=最大值)** | BigDecimal,BigInteger,CharSequence, byte, short, int, long等原子类型和包装类型 | 验证注解的元素值在最小值和最大值之间 |
| **@Email(regexp=正则表达式,**  **flag=标志的模式)** | CharSequence子类型（如String） | 验证注解的元素值是Email，也可以通过regexp和flag指定自定义的email格式 |
| **@Pattern(regexp=正则表达式,**  **flag=标志的模式)** | String，任何CharSequence的子类型 | 验证注解的元素值与指定的正则表达式匹配 |
| **@Valid** | 任何非原子类型 | 指定递归验证关联的对象；  如用户对象中有个地址对象属性，如果想在验证用户对象时一起验证地址对象的话，在地址对象上加@Valid注解即可级联验证 |

**11.拦截器**

1).创建拦截器类:implements HandlerInterceptor

2).配置文件

**12.Ajax+json**

@RequestBody接收的是一个json对象的字符串,而不是json对象,,该注解将返回的java对象自动转换为json字符串

**13.spring和springmvc容器**

Spring一种容器技术，这个容器技术是建立在WEB容器之上，WEB是一个宿主环境，而WEB容器是按照J2EE规范创建，也就是Servlet容器。

所谓容器就是拥有一个上下文对象，如Servlet容器就拥有自己的上下文对象ServletContext对象。这个对象就拥有Request对象和Response对象，这样它才能获取用户的请求，响应用户的请求。

有了这套结构，我们springmvc就直接getBean了，通过注解方式实现。其实getBean里面经历了很多过程。

所有程序的入口都在web.xml。spring通过监听器去读取spring的配置文件，然后创建spring的root根上下文WebApplicationContext。然后再加载servlet，springmvc的DispatcherServlet前端控制器。

spring和spirngmvc应该不是继承关系，而是引用关系，如果是继承在web.xml就无需两个入口spring和springmvc入口，一个不就够了。剖析源码也可以看到springmvc创建时，把WebApplicationContext，放入到自己的request对象中，可以作证其是引用关系。

Context-param----->listener------>filter----->servlet

**mybatis部分**

1.对比JDBC.Hibernate

1).原生的JDBC的缺点:

(1).原生的JDBC操作数据库时,需要频繁的开关链接.

(2).JDBC操作数据库时需要手写sql语句.学习需要成本

(3).查询数据库的结果集,需要人为的进行封装.

(4).JDBC中没有缓存处理

(5).JDBC的sql语句写到java文件

2).Hibernate框架（旧的框架）

面向对象的sql语句---Hql.好处:操作数据库时可以使用对象进行操作.能够将查询的结果集自动的封装为对象.(ORM)自动的对象关系映射.

缺点:产生了很多冗余的sql语句，单表数据量超过500万条时，执行速度变慢。

3).Mybatis框架

Mybatis是一个半自动的对象关系映射(ORM),实现结果集的自动封装,但是sql需要自己手写.

Mybatis中可以将sql写到配置文件中。

**2.#{}和${}**

在Mybatis 的Mapper.xml语句中parameterType向SQL语句传参有两种方式：#{}和${}

**#{}**:1)这种方式sql语句是经过预编译的,可以防止sql注入

2)#将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。

**${}:1)**无法防止sql注入,取值之后再去编译sql语句

2)一般用于传入数据库对象,例如传入表名.

3)一般能用#的就别用$.

4)MyBatis排序时使用order by 动态参数时需要注意，用$而不是#

5)如果sql语句中以列名为参数是,使用#不起作用,使用${map中的key},必须配合map一起联用;

**3.动态sql分类:**

• if

• choose (when, otherwise) --相当于switch -case

• trim (where, set)

1.where,<where>

2.set

3.trim 可以替换上面的的where和set

• foreach (where in())

**4.mybatis的关联映射:**

对一关联:<association> 多个用户归属一个组

对多关联:<collection> 一个组包含多个用户

**project部分**

**1.bootstrap前段框架简介:**

chartjs.bootcss.com

bootstrap是一个前段框架,主要用来做相应式的页面

bootstrap自适应浏览器,自适应分辨率(大小屏)

bootstrap适合做页面,不适合做大量的js交互

bootstrap提供大量的css样式,专门用来做各种页面

bootstraps使用的基础应用是jquery

使用Bootstrap中文网提供的CDN加速服务

[Bootstrap中文网](http://www.bootcss.com/)为Bootstrap构建了自己的CDN加速服务，并且采用国内云厂商的CDN服务，访问速度更快、加速效果更明显、没有速度和带宽限制、永久免费。Bootstrap中文网还对大量的开源工具库提供了CDN服务，请进入[Open CDN 主页](http://open.bootcss.com/)查看更多可用的工具库。

*<!-- 最新 Bootstrap 核心 CSS 文件 -->* <link rel="stylesheet" href="http://cdn.bootcss.com/twitter-bootstrap/3.0.3/css/bootstrap.min.css">

*<!-- 可选的Bootstrap主题文件（一般不用引入） -->* <link rel="stylesheet" href="http://cdn.bootcss.com/twitter-bootstrap/3.0.3/css/bootstrap-theme.min.css"> *<!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->* <script src="http://cdn.bootcss.com/jquery/1.10.2/jquery.min.js"></script> *<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->* <script src="http://cdn.bootcss.com/twitter-bootstrap/3.0.3/js/bootstrap.min.js"></script>

2.栅格系统

**Extjs**:extjs.org.cn 相比bootstrap更侧重js

全世界公认的非常牛的js框架,

基于bootstrap框架做出一个新的框架adminlte

补充内容:

js中escape方法unescape

escape方法,用来给一些特殊的字符串编码,网上的一种通用编码

escape("aa bb w3c!");//aa20%bb20%w3c21%

unescape("aa20%bb20%w3c21%");//aa bb w3c!