# spring ：

## 是什么：

是一个轻量级的开源框架，核心是基于控制反转(Inversion of Control)(也被成为依赖注入)和面向切面编程(aspect oriented programming)

## 能干吗

设值注入：由spring把组件所依赖的具体对象注入给组件,使得组件以配置文件而不是硬编码的方式耦合在一起a new调用b，而b改了构造方法，就得在所有关联修改,并不方便扩展，所以用bean

分为6部分Core Container, DataAccess/Integration ,web ,AOP ,Instrumentation ,test

docs：包含spring的相关文档 api、开发手册等

libs:各个模块的jar包 每个模块均提供

开发所需的jar包

javadoc后缀表示的api

-sources后缀表示的源文件

schema：配置某些功能需要用到的schema文件 对于已经集成了spring的ide如eclipse等这些文件不需要专门导入

config怎么写 ：

bean标签来定义一个实例(一个对象),有两个常用属性 id表示实例名称 class里写全类名表示实例类型 里面有property标签

有name属性表示给哪个属性赋值 对应的是set方法名

name为普通类型用value属性为其赋值,对象用ref属性注入已经声明出来的bean

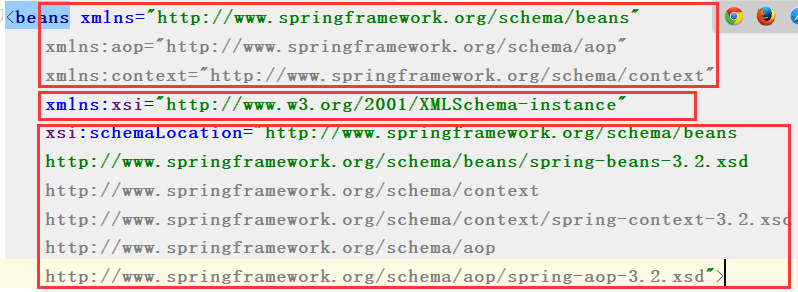
|  |
| --- |
| <**bean id="itripHotel" class="cn.travel.entity.ItripHotel"**>  <**property name="hotelName" value="如家"**></**property**> </**bean**> |

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml"); 实例化spring上下文

负责读取Spring配置文件，管理对象的加载生成(加入到容器里)，维护bean对象之间的依赖关系，负责bean的生命周期等

用于从classpath中读取spring配置文件

UserService service = context.getBean("service"); 根据id来获取对应实例



可以理解为先声明容器，在声明xsi用于使用容器，在注册容器。

Beans

Aop 实现切面用的

Context：

## AOP(aspect oriented programming)：

是什么：面向切面编程 是对面向对象编程有益的补充,一般适用于具有横切逻辑的场合，如访问控制，事物管理，性能监测等。

能干吗：在不改变原程序的基础上为代码增加新功能，对代码进行增强处理

为了在增强方法中获得当前连接点信息，可以在方法中声明一个JoinPoint类型的参数，Spring会自动注入该实例

getTarget 获取被代理的目标对象

getSignature 返回被代理的目标方法 jp.getSignature().getName()

getArgs 返回传递给目标方法的参数数组 Arrays.toString(jp.getArgs())

|  |
| --- |
| **private static** Logger *logger*= Logger.*getLogger*(WebLogAspect.**class**.getName()); *logger*.info(**"调用"**+jp.getTarget().getClass().getName()+**"的"**+jp.getSignature().getName() +**"方法，方法入参"**+ Arrays.*toString*(jp.getArgs())); |

对于后置增强方法(after-returning前置为before)还可以定义一个参数用于接收目标方法的返回值，配置文件里returning属性的值必须等于参数名并且目标方法返回值和参数类型是一致的。

### 1、aop相关的配置：

<aop:config>

//定义一个切入点表达式，并命名为 pointcut

<aop:pointcut id="pointcut" expression="execution(\* cn.travel.service.\*.\*(..))"/>

只要满足这个切入点的表达式都会为其增加增强方法

//引入包含增强方法的bean <bean id=”webLogger” class=”cn.aop.WebLogger”>

<aop:aspect ref="webLogger">

//将bfrore方法定义为前置增强并引入pointcut切入点

<aop:before method="before" pointcut-ref="pointcut"></aop:before>

//将after方法定义为后置并引用pointcut切入点 通过returning属性指定名为result的参数注入返回值

<aop:after-returning method="afterReturning" pointcut-ref="pointcut" returning="result"/>

</aop:aspect>

需要增强方法的service包里的类必须注入到容器里，例如现在的阶段必须在配置文件加上，然后getBean即可

<**bean class="cn.travel.service.UserServiceImpl" id="userServiceImpl"**></**bean**>

|  |
| --- |
| ApplicationContext context = **new** ClassPathXmlApplicationContext(**"applicationContext-mybatis.xml"**); UserServiceImpl userService = (UserServiceImpl) context.getBean(**"userServiceImpl"**); userService.asd(); |

### 2、execution()表达式支持模糊匹配

public \* addNewUser (entity User) \*表示匹配所有类型的返回值

public void \* (entity User) \*表示匹配所有方法名

public void addNewUser (..) ..表示匹配所有参数个数和类型

\* com.service.\*.\*(..) 这个表达式匹配com.service包下所有类的所有方法

\* com.service..\*.\*(..) 这个表达式匹配com.service包及其子包下所有类的所有方法

### 3、扩展增强：

异常抛出增强：在目标方法抛出异常时添加增强处理

标签为<aop:after-throwing 里面有thorwing属性对应Exception类型的参数名，Spring为其注入抛出异常的实例

前置增强before在逻辑代码之上执行

后置增强after-returning在逻辑代码之下执行 有returning属性，等于参数名并类型一致，方法出现异常则不执行

最终增强after相当于finally就算出现异常也正常执行

环绕增强：around可以在目标方法前后都增强处理

ProceedingJoinPoint为joinpoint子接口多封装了目标对象，通过proceed调用目标方法，用在环绕方法上

|  |
| --- |
| **public** Object aroundLogger(ProceedingJoinPoint jp){   *//随便输出点信息相当于前置增强(就是在目标方法之前输出)  logger*.info(**"调用"**+jp.getTarget().getClass().getName()+**"的"**+jp.getSignature().getName()  +**"方法，方法入参"**+ Arrays.*toString*(jp.getArgs()));  **try** {  Object result = jp.proceed();  *//随便输出点信息，相当于后置增强 出现异常就不会执行额  logger*.error(**"调用"**+jp.getTarget().getClass().getName()+**"的"**+jp.getSignature().getName()  +**"方法，方法返回值:"**+result);  **return** result;  } **catch** (Throwable e) {  *//异常增强  logger*.error(jp.getSignature().getName()+**"方法发生异常"**);  e.printStackTrace();  }**finally** {  *//最终增强  logger*.info(jp.getSignature().getName()+**"方法结束"**);  }  **return null**; } |

### 5、注解方式实现

1、使用注解定义切面： 类加上@Aspect注解

@Before("execution(\* test.\*.\*(..))")

@AfterReturning(pointcut = "execution(\* test.\*.\*(..))",returning = "re")

@AfterThrowing(pointcut = "pointcut()",throwing = "rx")

@After("execution(\* test.\*.\*(..))")

@Around("execution(\* test.\*.\*(..))") 方法记得用ProceedingJoinPoint参数

2、还定义一个公共切面 @Before("pointcut()") 这样使用即可

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

public void pointcut() {}

3、然后在配置文件完成织入即可

<bean class="cn.travel.aop.WebLogAspect"></bean>

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

### 4、注意：

每个增强类都得要无参构造方法

不写boot里aop的*半年数据迁移云备份那个东西*

需要增强方法的service包里的类必须注入到容器里，然后getbean即可

Server不用写名字，例如UserServiceImpl的默认容器名为userServiceImpl

<**context:component-scan base-package="cn.travel.service"**></**context:component-scan**>

|  |
| --- |
| ApplicationContext context = **new** ClassPathXmlApplicationContext(**"applicationContext-mybatis.xml"**); UserServiceImpl userService = (UserServiceImpl) context.getBean(**"userServiceImpl"**); userService.asd(); |

### 单词

Core 核心

Container 容器

Access 存取

Integration 集成

javadoc doc就是document文档

ApplicationContext 应用上下文

Signature 签名

pointcut 切入点 点，切入

expression 表达式

aspect 方面

inversion 倒置，反向

oriented 定向，导向

programming 编程

## 构造注入：

在bean(绑定了一个实例)内嵌套<constructor-arg>

参数为对象：<ref bean="st"/>

参数为String 给构造标签里写value

st是已经获取到的实例(对应相应bean的id)用来赋值到当前这个bean的构造方法

如果为多个参数就写多个constr 然后有index(从0开始)来指定位置

|  |
| --- |
| <**bean id="itripHotel" class="cn.travel.entity.ItripHotel"**>  <**constructor-arg**>  <**value**>拉拉</**value**>  </**constructor-arg**>  <**constructor-arg**>  <**ref bean="webLogAspect"**></**ref**>  </**constructor-arg**>  </**bean**> |

构造注入时效性好在对象实例化时就得到所依赖的对象，便于在对象的初始化方法中使用依赖对象；但受限于重载的形式灵活性没有设值注入强

### 扩展：

如果属性包含了特殊字符，在value标签里写<![CDATA[xxx]]> 最里面的方括号随便写内容

<ref bean="st"/>还有一个属性是local 用法是一样的，spring配置文件是可以拆分成多个的，local属性只能在同一个config中检索bean的id，bean可以在其他config检索id。

比如都是<bean> <property name=”dao”> dao属性 下面的各个都写到这个里面

内部bean <bean class=”test.daoImpl”>

有list属性时候 <list><value>1</value><value>1</value>

map属性 <map><entry> <key><value>a</value></key>

<value>凯凯</value> </entry>

properties属性 <props> <prop key=”a”>凯凯<prop>

注入空值value里不写东西 注入null 把value标签替换为<null/>

## 注解开发

### 使用注解定义bean

@Component(“userDao”) 定义了一个当前类实例的bean名字为userDao

@Repository 用于标注DAO类

@Service 用于标注业务类

@Controller 用于标注控制器类

@Autowired 在属性上写，自动装配合适的依赖对象，若有多个匹配时,则可以用

@Qualifier(“userDao”) 指定所需bean名称

### 注意：

然后可以在配置文件直接 <context:component-scan base-package="test"></context:component-scan>来扫描指定的包来注册bean。多个包名逗号隔开

如果对类中集合类型的变量使用@Autowired，会将所有和集合中元素类型匹配的bean组件都注入

### 单词

transaction 事务 穿sai ke 神

Manager 管理 mai nei zher

constructor 构造器 ken si chua ke ter

around 环绕 饿 ruang de

Component 组成的 ken po nen te

标注组件和组件 scan扫描

Repository 标注dao类 rui po si te rui

Autowired 自动匹配

Qualifier 限定符 k哦 里 fai er

base 基础 bei si

proxy 代理 pro ke sei

## 整合mybatis

spring对myBatis整合：需要导入整和用的三个jar包和dbcp两个jar包和mybatis和mysql包

mybatis映射文件照常，核心文件只写个别名和<mapper resource="UserMapper.xml"></mapper>，别的让Spring接管

但是用idea maven项目如果xml在mapper目录下那么需要

|  |
| --- |
| <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  <**include**>\*\*/\*.xml</**include**>  </**includes**>  </**resource**>  </**resources**> </**build**> |

|  |
| --- |
| *<!-- 获取数据库配置文件 -->* <**context:property-placeholder location="classpath:database.properties"**/>  *<!--配置数据源 (使用dbcp连接池) -->* <**bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" scope="singleton"**>  <**property name="driverClassName" value="${driver}"**/>  <**property name="url" value="${url}"**/>  <**property name="username" value="${user}"**/>  <**property name="password" value="${password}"**/>   <**property name="initialSize" value="${initialSize}"**/> *<!-- 初始化连接时,第一次就要创建的连接个数 -->* <**property name="maxActive" value="${maxActive}"**/> *<!-- 同时连接的最大连接数 -->* <**property name="maxIdle" value="${maxIdle}"**/> *<!-- 最大空闲连接数 -->* <**property name="minIdle" value="${minIdle}"**/> *<!-- 最小,低于该数值连接池将会创建新的连接 -->* <**property name="maxWait" value="${maxWait}"**/> *<!-- 定义最大等待事件 -->* <**property name="removeAbandoned" value="${removeAbandoned}"**/> *<!-- 是否开启无用连接回收的机制 -->* <**property name="removeAbandonedTimeout" value="${removeAbandonedTimeout}"**/>*<!-- 超过配置的时间后回收无用的连接 -->  <!--sql心跳 -->* <**property name= "testWhileIdle" value="true"**/> *<!-- 开启evict的定时效验 -->* <**property name= "timeBetweenEvictionRunsMillis" value="60000"**/> *<!-- 定义Evict的时间间隔,单位毫秒 -->* <**property name= "testOnBorrow" value="false"**/> *<!-- -->* <**property name= "testOnReturn" value="false"**/>  <**property name= "validationQuery" value="select 1"**/> *<!-- 定义效验sql语句,简单通信一下,测试效验是否有效 -->* <**property name= "numTestsPerEvictionRun" value="${maxActive}"**/>*<!-- 定义每次效验连接的数量 -->* </**bean**>  *<!--mybatis中用builder创建sqlSessionFactory，集成环境中用整合包中的SqlSessionFactoryBean代替sqlSessionFactory。 -->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  *<!-- 引用数据源组件 -->* <**property name="dataSource" ref="dataSource"**></**property**>  *<!-- 引用mybatis配置文件中的配置-->* <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**> </**bean**>  *<!-- basePackage定义了扫描的基准包，扫描包之下所有接口，如果他们在sql映射文件中定义过，则自动生成dao实现的代理对象  生成的对象名为接口名称首字母小写,多个包名用逗号隔开 配置了数据源才能自动生成代理对象 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="cn.travel.mapper"**></**property**> </**bean**> |

### 步骤

导入完jar和配置文件，那么直接写代码即可，记得扫描service包

### 重点

接口和xml的名字一定要一样

## 整合junti

|  |
| --- |
| <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>3.8.1</**version**> </**dependency**> <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>3.2.10.RELEASE</**version**> </**dependency**>  @ContextConfiguration(locations ={**"/applicationContext-mybatis.xml"**}) @RunWith(SpringJUnit4ClassRunner.**class**) **public class** test {  @Autowired  HotelMapper **hotelMapper**;  @Test  **public void** findList(){  System.***out***.println(**hotelMapper**.findList());  } } |

# Spring mvc

## mvc架构优点：

多视图共享一个模型大大提高代码重用性

mvc3个模块相互独立,松耦合框架

控制器提高了应用程序的灵活性和可配置性

有利于软件工程化管理

## mvc架构缺点：

原理复杂

增加了系统结构和实现的复杂性

视图对模型数据的低效率访问

并不适用于小型甚至中型项目,花费大量时间将mvc应用到规模不大的应用程序得不尝失

## 是什么：

Spring mvc 是spring框架中用于web开发的一个模块,是一个基于mvc设计模式的web开发框架。

采用了松耦合可插拔的组件结构,具有高度可配型，扩展性，灵活性，而且本身就是spring框架的一部分,性能方面有天生的优越性

## 能干吗：

本质上相当于servlet。

支持灵活的URL到页面控制器的映射；支持Restful风格。

非常容易与其他视图技术集成，如Velocity、FreeMarker等等

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；

## 环境搭建

1. 导入jar包
2. 在web.xml配置dispatcherServlet，映射到 / 即此servlet需要捕获并处理该项目所有url请求

3、创建springmvc配置文件，写<mvc:annotation-driven/>标签，配置该标签会自动注册：

1、DefaultAnnotationHandlerMapping(处理器映射,不采用注解时默认使BeanNameUrl处理器即在spring中查找与url同名的bean) 加注解为该请求映射到标注@requestMapping注解的控制器和方法上

BeanNameUrl用法：<bean id="/demo" class="main.java.controller.UserController"/>然后路径/demo就会进入这个控制器，控制器需要实现AbstractController实现方法后在里面写。

2、AnnotaionMethodHandlerAdapter(处理器适配器)，是spring对@controller分发请求的必要支持

处理器映射：对requemapping和conteoller注解的支持，找出请求url和处理器的关系从而找出进入相应的处理器

处理器适配器：找到对应的处理器之后 就用adapter进行封装，在用统一的适配器接口调用处理器 会将请求信息以一定的方式进行转换并绑定到请求的入参中

1. 然后在加scan扫描control，bean才能生效，才能进行映射

5、在cotroller类方法上加@RequestMapping("/index") 注解来映射请求的url 路径直接写/index就进入这个方法了

6、cotroller类方法返回值通过视图解析器解析为实际的物理视图,对于InternalResourceViewResolver 视图解析器

通过prefix + returnVal + suffix 得到实际物理视图转发 基于jsp视图的解析一般会使用它

Spring mvc 配置文件的命名，必须和web.xml里dispatcherServlet指定的名称一致 一般命名为 名称-servlet。

## 运行原理：

1.Tomcat在启动时加载解析web.xml,找到spring mvc的前端总控制器DispatcherServlet,并且通过DispatcherServlet来加载相关的配置文件。

2.DispatcherServlet接收到客户端请求，找到对应HandlerMapping(一般都是用DefaultAnnotationHandlerMapping它是对requemapping和conteoller注解的支持)，根据映射规则，找到对应的处理器，找到对应的处理器之后 就用adapter进行封装，在用统一的适配器接口调用处理器 会将请求信息以一定的方式进行转换并绑定到请求的入参中。

3.调用相应处理器中的处理方法，处理该请求后，会返回一个值。

4.DispatcherServlet根据得到的值，找到一个合适的ViewResolver（视图解析器），根据视图解析器的配置，DispatcherServlet将要显示的数据传给对应的视图，最后显示给用户。

# 重点：

Boot可以直接页面html suffix设置.html然后返回，但是ssm不可以，因为静态资源也会去访问dispatcherServlet会导致失败，解决：在web.xml加上这个，别的静态资源在用<mvc:resources location="/statics/" mapping="/statics/\*\*"/>处理

|  |
| --- |
| <**servlet-mapping**>  <**servlet-name**>default</**servlet-name**>  <**url-pattern**>\*.html</**url-pattern**> </**servlet-mapping**> |

如果出现个找不到springmvc-servlet错误可以在前面classpath后加一个\*

<**param-value**>classpath\*:springmvc-servlet.xml</**param-value**>