## 一、spring与springmvc及其配置

Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架 ,spring mvc是spring web层的框架，主要用来接受HTTP请求并转发给controller(需要配置dispatcherservlet) 重要组成部分有viewresolver handlermapping 以及dispatcherservlet等

springmvc配置 web.xml配置listener、filter、servletmapping以及hibernate/mybatis配置文件地址,dispatcherservlet(在servlet param中指定dispatcherservlet的配置文件地址 多为springmvc.xml)

Springmvc.xml中需要实现的基本功能配置: 配置component-scan指定扫描的bean包,将添加注解的bean自动注入容器;配置视图解析器viewresolver;配置<mvc:annotation-driven />相当于注册了DefaultAnnotationHandlerMapping(提供@requestmapping注解支持)和AnnotationMethodHandlerAdapter(处理继承HttpRequestHandler的处理器)两个bean，是spring MVC为@Controllers分发请求所必须的。并提供了：数据绑定支持，@NumberFormatannotation支持，@DateTimeFormat支持，@Valid支持，读写XML的支持（JAXB），读写JSON的支持（Jackson）。

<context:annotation-config> 是用于激活那些已经在spring容器里注册过的bean上面的注解。

<context:component-scan>除了具有<context:annotation-config>的功能之外，<context:component-scan>还可以在指定的package下扫描以及注册javabean 。

<context:annotation-config />和 <context:component-scan>同时存在的时候，前者会被忽略.

## 二、web.xml配置详解

启动WEB项目的时候，容器首先会去它的配置文件web.xml读取两个节点:

<listener></listener>和<context-param></context-param>(context-param中可以配置contextConfig

Location指定要读取的其他配置文件的地址)

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

**ContextLoaderListener**监听器的作用就是启动Web容器时，自动装配ApplicationContext的配置信息。

默认的路径是"/WEB-INF/applicationContext.xml，如果是要自定义文件名可以在web.xml里加入contextConfigLocation这个context参数;紧接着，容器创建一个ServletContext（application），这个WEB项目所有部分都将共享这个上下文。

容器以<context-param></context-param>的name作为键，value作为值，将其转化为键值对，存入ServletContext。

容器创建<listener></listener>中的类实例，根据配置的class类路径<listener-class>来创建监听，在监听中会有contextInitialized(ServletContextEvent args)初始化方法，启动Web应用时，系统调用Listener的该方法，在这个方法中获得：

ServletContext application =ServletContextEvent.getServletContext();

context-param的值= application.getInitParameter("context-param的键");

接着，容器会读取<filter></filter>，根据指定的类路径来实例化过滤器。

以上都是在WEB项目还没有完全启动起来的时候就已经完成了的工作。如果系统中有Servlet，则Servlet是在第一次发起请求的时候被实例化的，而且一般不会被容器销毁，它可以服务于多个用户的请求。所以，Servlet的初始化都要比上面提到的那几个要迟。

总的来说，web.xml的加载顺序是: <context-param>-> <listener> -> <filter> -> <servlet>。如果web.xml中出现了相同的元素，则按照在配置文件中出现的先后顺序来加载。

其他配置元素:

< session-config></session-config>用于设置容器的session参数，比如：<[session-timeout](http://wiki.metawerx.net/wiki/Web.xml.SessionTimeout)>用于指定http session的失效时间。

<listener></listener>为web应用程序定义监听器，监听器用来监听各种事件，比如：application和session事件，所有的监听器按照相同的方式定义，功能取决去它们各自实现的接口，常用的Web事件接口有如下几个：

* ServletContextListener：用于监听Web应用的启动和关闭；
* ServletRequestListener：用于监听用户的请求；
* ServletRequestAttributeListener：用于监听ServletRequest范围（request）内属性的改变；
* HttpSessionListener：用于监听用户session的开始和结束；
* HttpSessionAttributeListener：用于监听HttpSession范围（session）内属性的改变。

<listener>主要用于监听Web应用事件，其中有两个比较重要的WEB应用事件：应用的启动和停止（starting up or shutting down）和Session的创建和失效（created or destroyed）。

Web应用配置Listener的两种方式:使用@WebListener修饰Listener实现类和在web.xml中使用<listener>进行配置。

**<filter></filter>**Filter可认为是Servle的一种“加强版”，主要用于对用户请求request进行预处理，也可以对Response进行后处理，是个典型的处理链。使用Filter的完整流程是：Filter对用户请求进行预处理，接着将请求HttpServletRequest交给Servlet进行处理并生成响应，最后Filter再对服务器响应HttpServletResponse进行后处理。Filter与Servlet具有完全相同的生命周期，且Filter也可以通过<init-param>来配置初始化参数，获取Filter的初始化参数则使用FilterConfig的getInitParameter()。

## 三、spring AOP 面向切面

Spring aop基本概念

连接点 joinpoint:是一个应用执行过程中能插入切面的点

切面 aspect：通知advice和切点pointcut的集合，通知和切点定义了切面的功能以及何时何处完成其功能

通知 advice 切面的工作叫做通知 包括前置通知@befor 后置通知@after 环绕通知@around

切点 pointcut：定义通知被应用的位置(在哪些连接点) aspecj的@pointcut注解能够在一个切面中定义可重用的切点。

织入 ：是将切面应用到目标对象并创建新的代理对象的过程。切面在指定的连接点被织入到目标对象中。在目标对象的生命周期里有多个点可以进行织入：

编译期： 切面在目标类编译时被织入。 这种方式需要特殊的编译器。 AspectJ的织入编译器就是以这种方式织入切面的。

类加载期： 切面在目标类加载到JVM时被织入。 这种方式需要特殊的类加载器（ ClassLoader） ， 它可以在目标类被引入应用之前增强该目标类的字节码。 AspectJ 5的加载时织入（ load-time weaving， LTW） 就支持以这种方式织入切面。

运行期：切面在应用运行的某个时刻被织入。 一般在织入切面时，AOP容器会为目标对象动态地创建一个代理对象, Spring AOP就是以这种方式织入切面的。Spring使用aspectj来实现aop

## 四、过滤器filter与拦截器interceptor

**拦截器**：是在面向切面编程的;就是在你的service或者一个方法前调用一个方法，或者在方法后调用一个方法.比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。

拦截器是Spring容器内Spring框架支持的

**过滤器**：在HttpServletRequest 到达Servlet 之前，拦截客户的HttpServlet

Request根据需要检查HttpServletRequest,也可以修改HttpServletRequest 头和数据。

在HttpServletResponse 到达客户端之前，拦截HttpServletResponse 。

根据需要检查HttpServletResponse ，可以修改HttpServletResponse 头和数据。

创建一个Filter需要两个步骤:(1)创建Filter 处理类(2)在web.xml文件中配置Filter 。

过滤器是在Servlet规范中定义的，是Servlet容器支持的

**Filter常见应用场景：**

用户授权的Filter: Filter 负责检查用户请求，根据请求过滤用户非法请求。

日志Filter: 详细记录某些特殊的用户请求。

负责解码的Filter: 包括对非标准编码的请求解码.

能改变XML 内容的XSLTFilter 等。

两者的本质区别：拦截器是基于Java的反射机制的，而过滤器是基于函数回调。从灵活性上说拦截器功能更强大些，Filter能做的事情，他都能做，而且可以在请求前，请求后执行，比较灵活。Filter主要是针对URL地址做一个编码的事情、过滤掉没用的参数、安全校验（比较泛的，如是否登录），太细的话，还是建议用interceptor。

**Interceptor常见应用场景：**

1、日志记录：记录请求信息的日志，以便进行信息监控、信息统计、计算PV（Page View）等。

2、权限检查：如登录检测，进入处理器检测检测是否登录，如果没有直接返回到登录页面；

3、性能监控：有时候系统在某段时间莫名其妙的慢，可以通过拦截器在进入处理器之前记录开始时间，在处理完后记录结束时间，从而得到该请求的处理时间（如果有反向代理，如apache可以自动记录）；

4、通用行为：读取cookie得到用户信息并将用户对象放入请求，从而方便后续流程使用，还有如提取Locale、Theme信息等，只要是多个处理器都需要的即可使用拦截器实现。

5、OpenSessionInView：如Hibernate，在进入处理器打开Session，在完成后关闭Session。

本质也是AOP（面向切面编程），也就是说符合横切关注点的所有功能都可以放入拦截器实现。

**拦截器实现：**

需要实现HandlerInterceptor接口，**preHandle：**预处理回调方法，实现处理器的预处理（如登录检查），第三个参数为响应的处理器（如我们上一章的Controller实现）；返回值true表示继续流程（如调用下一个拦截器或处理器）；false表示流程中断（如登录检查失败），不会继续调用其他的拦截器或处理器，此时我们需要通过response来产生响应；

**postHandle：**后处理回调方法，实现处理器的后处理（但在渲染视图之前），此时我们可以通过modelAndView（模型和视图对象）对模型数据进行处理或对视图进行处理，modelAndView也可能为null。(ModelAndView返回之前调用这个方法)

**afterCompletion：**整个请求处理完毕回调方法，即在视图渲染完毕时回调，如性能监控中我们可以在此记录结束时间并输出消耗时间，还可以进行一些资源清理，类似于try-catch-finally中的finally，但仅调用处理器执行链中**preHandle返回true的拦截器的afterCompletion**。

**Filter与Interceptor的区别**：Filter是servlet规范规定的，只能用在web工程中，Interceptor是spring框架支持的。拦截器是spring组件，所以可以使用spring中的任何对象、资源，例如service对象、数据源、事务管理等，通过Ioc注入到拦截器即可

## 五、spring bean注入(四种方式)

**set方法注入** **setter**：

<bean name="springAction" class="com.bless.springdemo.action.SpringAction">

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

</bean>

**构造器注入**  **constructor**

<bean name="springAction" class="com.bless.springdemo.action.SpringAction">

<!--创建构造器注入,如果主类有带参的构造方法则需添加此配置-->

<constructor-arg ref="springDao"></constructor-arg>

<constructor-arg ref="user"></constructor-arg>

</bean>

**静态工厂注入、实例工厂注入**

Spring支持**5种自动装配模式(xml文件中bean的Autowire属性值)**，如下：

no ——默认情况下，不自动装配，通过“ref”attribute手动设定(这就是普通的set方法注入，需要在property标签中指定要注入的成员变量的ref)。

<bean id="iocBean" class="com.ect.IocBean ">

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

</bean>

byName ——根据Property的Name自动装配，如果一个bean的name，和另一个bean中的Property的name相同，则自动装配这个bean到Property中。

<bean id="iocBean" class="com.ect.IocBean " autowire = "byName">

byType ——根据Property的数据类型（Type）自动装配，如果一个bean的数据类型，兼容另一个bean中Property的数据类型，则自动装配。

Constructor ——根据构造函数参数的数据类型，进行byType模式的自动装配。

Autodetect ——如果发现默认的构造函数，用constructor模式，否则，用byType模式

**Spring2.5之后引入注解@Autowired以代替set方法**+xml文件的bean注入方式，@Autowired注解可以对类的变量、方法、构造函数标注，完成自动装配的工作。

@Autowired为spring提供的注解，注解的成员变量**默认按照类型byType注入**，容器注入时首先查找对应类型的bean，如果存在多个此类型bean，那么会查找bean name相同的。

Spring遇到一个在setter方法中使用的@Autowired 注解,它会**在方法中执行byType自动装配**.

@Autowired

ServiceBean serviceBeanByName;//当Autowired by name注入时bean对象的变量名即为bean名;此时实现ServiceBean接口的注解为@component(name=“serviceBeanByName”)的java类对象将被注入到serviceBeanByName变量中。

定义bean name的方式：@component(name=“beanname”)

@resource为J2EE提供，**默认按照名称注入**byname；@Resource有两个重要属性，分别是name和type，Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。

如果@Autowired 实现接口的类型有多个按照类型注入时会出错，这时可以配合@Qualifier(name)注解使@Autowired可以按照bean名称注入.

@Autowired标注构造函数，可以明确成员变量的加载顺序:

@Autowired

private User user;

private String school;

public UserAccountServiceImpl(){

this.school = user.getSchool();

}此段代码不能执行成功原因是java会先执行构造方法，然后再给@Autowired注解的user注入值，所以执行构造方法时会报错。此时将注解标注在构造方法上这样会先执行构造方法，那么bean初始化就会成功。

@Component注解**(会将类自动注册到Spring容器中)**及@Repository、@Service、@Controller将一个java类定义为bean，默认的bean名为类名的首字母小写。@Autowired和@component同时使用可以不必在xml文件中配置bean

## 六、session和cookie

Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；

浏览器发送请求时，会将一个sessionid放到request header中，服务器以sessionid为标志生成一个session，不同用户的请求sessionid一定不相同，这样服务器可以识别不同用户的请求 java程序可以用httpRequest.getsession方式得到请求的session

Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式。浏览器与服务器建立连接时，服务器自发向浏览器发送一个名为JSSIONID的cookie，值为浏览器请求时发送给服务器的sessionid的值。该cookie的maxAge属性一般为-1表示仅当前浏览器内有效，关闭浏览器失效。当浏览器发送一个http请求时会寻找本机上该网站设置的cookie文件，将cookie文件中的数据放到requestheader中发送到服务器

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗。考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以建议将登陆信息等重要信息存放为session,其他信息如果需要保,可以放在cookie中

cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。若不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就消失。这种生命期为浏览器会话期的cookie被称为会话cookie。

会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。若设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间

Java对cookie操作:

HttpServletRequest request;

HttpServletResponse response;

Cookie cookie = new Cookie("cookiename","cookievalue");

cookie.setMaxAge(3600);//设置生命周期

cookie.setPath("/");//设置路径，这个路径即该工程下都可以访问该cookie 如果不设置路径，那么只有设置该cookie路径及其子路径可以访问

response.addCookie(cookie);

Cookie[] cookies = request.getCookies();//这样便可以获取一个cookie数组

for(Cookie cookie : cookies){

cookie.getName();// get the cookie name

cookie.getValue(); // get the cookie value

}

删除一个name 为”username”的 cookie：

Cookie c = new Cookie(“username”,”“);

c.setMaxAge(0); //设置生命周期为0

cookie.setPath(request.getContextPath());//路径

response.addCookie(c);

浏览器在向服务器上的某个地址发送请求时，会先比较 cookie 的路径与访问的路径(地址)是否匹配, 叧有匹配的 cookie,才会发送。

cookie 的路径可以通过 cookie.setPath(String path)方法来设置。如果没有设置, 则有一个缺省的路径，缺省的路径是生成该 cookie 的组件的路径。

比如: /appname/addCookie 保存了一个 cookie,则该 cookie 的路径就是/appname/addCookie。

cookie 的路径必须是要访问的路径的上层目录或者是不要访问的路径相等， 浏览器才会将 cookie 发送给服务器。一般可以设置 setPath(“/appname”),表示访问该应用下的所有地址，均会发送。

要想在cookie中存储中文，那么必须使用URLEncoder类里面的encode(String s, String enc)方法进行中文转码:

Cookie cookie = new Cookie("userName", URLEncoder.encode("中文", "UTF-8"));

response.addCookie(cookie);

在获取cookie中的中文数据时，再使用URLDecoder类里面的decode(String s, String enc)进行解码:

URLDecoder.decode(cookies[i].getValue(), "UTF-8")

**客户端不支持cookie的解决方案：**

URL地址重写：将用户Session的id信息重写到URL地址中，服务器能够解析重写后的URL获取Session的id。HttpServletResponse类提供了encodeURL(Stringurl)实现URL地址重写。该方法会自动判断客户端是否支持Cookie。如果客户端支持Cookie，会将URL原封不动地输出来。如果客户端不支持Cookie，则会将用户Session的id重写到URL中。即在URL参数的前面添加了字符串";jsessionid=XXX"

设计模式

prototype原型模式

使用原型模式复制对象不会调用类的构造方法

因为对象的复制是通过调用Object类的clone方法来完成的，它直接在内存中复制数据，因此不会调用到类的构造方法。不但构造方法中的代码不会执行，甚至连访问权限都对原型模式无效

使用场景：

创建新对象成本较大（如初始化需要占用较长的时间，占用太多的CPU资源或网络资源），新的对象可以通过原型模式对已有对象进行复制来获得，如果是相似对象，则可以对其成员变量稍作修改。

如果系统要保存对象的状态，而对象的状态变化很小，或者对象本身占用内存较少时，可以使用原型模式配合备忘录模式来实现。

原型模式的主要思想是基于现有的对象克隆一个新的对象出来，一般是有对象的内部提供克隆的方法，通过该方法返回一个对象的副本，这种创建对象的方式，相比我们之前说的几类创建型模式还是有区别的，之前的讲述的工厂模式与抽象工厂都是通过工厂封装具体的new操作的过程，返回一个新的对象，有的时候我们通过这样的创建工厂创建对象不值得，特别是以下的几个场景的时候，可能使用原型模式更简单也效率更高。

当一个系统应该独立于它的产品创建、构成和表示时，要使用 Prototype模式

当要实例化的类是在运行时刻指定时，例如，通过动态装载；

为了避免创建一个与产品类层次平行的工厂类层次时

当一个类的实例只能有几个不同状态组合中的一种时。建立相应数目的原型并克隆它们可能比每次用合适的状态手工实例化该类更方便一些。（也就是当我们在处理一些对象比较简单，并且对象之间的区别很小，可能只是很固定的几个属性不同的时候，可能我们使用原型模式更合适）。

spring线程安全：

默认的Controller、Dao、service都是单例的，为避免出现线程安全问题应该尽量不声明成员变量

解决方案：在spring Controller配置文件中声明scope=prototype，每次都创建新的实例(默认为singleton) 或者使用THreadLocal

java克隆

浅克隆 对象中其他对象的成员变量会克隆引用；深克隆 会对对象成员变量也克隆(实现深克隆的方法：序列化与反序列化(需要实现Serializable接口))

public Object deepClone()

{

//将对象写到流里

ByteArrayOutoutStream bo=new ByteArrayOutputStream();

ObjectOutputStream oo=new ObjectOutputStream(bo);

oo.writeObject(this);

//从流里读出来

ByteArrayInputStream bi=new ByteArrayInputStream(bo.toByteArray());

ObjectInputStream oi=new ObjectInputStream(bi);

return(oi.readObject());

}

这样做的前提是对象以及对象内部所有引用到的对象都是可串行化的

hessian 配置

使用hessian服务的项目 配置hessian的bean 包括hessian的远程地址和接口类型ServiceInterface

提供hessian服务的项目配置hessian config

应用级协议Binary-RPC：一种和RMI类似的远程调用协议，和RMI的不同之处在于以标准二进制格式来定义请求的信息（请求的对象、方法、参数等），所以可以在跨语言通讯时可用

RPC协议的一次远程通信过程：

客户端发起请求，按照RPC协议将请求信息进行填充;填充完毕后将二进制格式文件转化为流，通过传输协议进行传输;接收到流后转换为二进制格式文件，按照rpc协议获取请求的信息并进行处理;处理完毕后将结果按照RPC协议写入二进制格式文件中并返回。

Hseeian是基于RPC实现的远程通信库，通信时通过Hessian本身提供的API来发起请求，Hessian基于http协议进行传输。

服务端接收请求并处理步骤：

服务端截获相应请求交给HessianServiceExporter—>HessianExporter将远程调用对象封装为HessianSkeleton框架 HessianSkeleton处理远程调用请求。

RPC协议使用c/s方式，采用http协议发送请求到服务器，等待服务器返回结果。这个请求包括一个参数集和一个文本集，通常形成classname.methodname形式。优点是跨语言夸平台，缺点是不支持对象，无法在编译期检查错误，只能在运行期检查。

webservice提供的服务是基于web容器的，底层使用http协议，类似一个远程服务提供者是一种请求应答的机制，是跨系统跨平台的。通过一个servlet提供服务出去。

RMI remote method invocation

只支持java语言，调用远程对象方法，允许方法返回java对象