**常用注解：**

@AutoWired 自动装配（DI），byName和byType最常见，需要在配置文件中注册AutoWiredAnnotationBeanPostProcessor。xml配置中有个对应的autowire属性，常见的有byType和byName两种，但是在找不到匹配Bean时不予理会，因此@Required应运而生，它需要注册RequiredAnnotationBeanPostProcessor。

@Inject，JSR330定义，可以作用在构造器、属性、方法上，可以设置required=false。

@Qualifier，当有多个符合条件的装配者时，使用该注解指定。相当于配置中的ref。

要使这些注解生效，得在配置文件中注册相应的BeanFactoryPostProcessor，也可以简化配置为：<context:annotation-config/>，注意引入相应的命名空间。

**Bean配置：**

静态工厂：

public class HelloApiStaticFactory {

//工厂方法

public static HelloApi newInstance(String message) {

//返回需要的Bean实例

return new HelloImpl2(message);

}

}

<!-- 使用静态工厂方法 -->

<bean id="bean3" class="cn.javass.spring.chapter2.HelloApiStaticFactory" factory-method="newInstance">

<constructor-arg index="0" value="Hello Spring!"/>

</bean>

通过factory-method方法创建实例，可以注入参数。问题：不知道创建出来的实例类型。

实例工厂：

package cn.javass.spring.chapter2;

public class HelloApiInstanceFactory {

public HelloApi newInstance(String message) {

return new HelloImpl2(message);

}

}

<!—1、定义实例工厂Bean -->

<bean id="beanInstanceFactory"

class="cn.javass.spring.chapter2.HelloApiInstanceFactory"/>

<!—2、使用实例工厂Bean创建Bean -->

<bean id="bean4"

factory-bean="beanInstanceFactory"

factory-method="newInstance">

<constructor-arg index="0" value="Hello Spring!"></constructor-arg>

</bean>

因为是实例方法，必须使用factory-bean单独指定工厂类。

**循环依赖：**

循环调用无解，死循环。

singleton的Bean可以循环依赖，因为Spring会将之缓存；prototype的Bean无法循环依赖，因为不缓存。对于singleton的循环依赖，Spring先将Bean实例化出来，不注入，缓存之，然后挨个注入。

**Resource和ResourceLoader**

Resource对各种资源提供统一接口，如ByteArrayResource，FileResource，URLResource等；而ResourceLoader则相当于Resource的一个类工厂，通过传入不同参数返回不同的Resource，而不需要我们自己判断。

**SpEL Spring Expression Language**

和OGNL、EL、JSTL等类似，不过天然与spring集成。需要注意的是，如果要修改模板开始结束符号，需要建立一个BeanFactoryPostProcessor，在它的postProcessBeanFactory方法中替换模板开始结束符号。如下：

StandardBeanExpressionResolver resolver = (StandardBeanExpressionResolver) beanFactory.getBeanExpressionResolver();

resolver.setExpressionPrefix("%{");

**AOP**

推荐使用自定义注解的方式来配置。类似于@Transactional，这样便于知道当前方法被哪些切面横切。注：@Transactional注解需要配置<tx:annotation-driven/>,需指定事务管理者

使用注解方式配置aop，需要配置<aop:aspectj-autoproxy />

**JDBC和ORM**

spring提供了大量的模板方法，关于jdbc的就不介绍了。

hibernate3：SessionFactory，用LocalSessionFactoryBean。另提供HibernateDaoSupport（可获取HibernateTemplate）。

iBatis：SqlMapClientFactoryBean。

**Web框架集成：**

在web.xml文件中

<context-param>

<param-name>contextClass</param-name>

<param-value>

org.springframework.web.context.support.XmlWebApplicationContext

</param-value>

</context-param> 这个是默认的ApplicationContext实现，如果不更改则可以不配置。

<context-param>

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

<param-value>

classpath:chapter10/applicationContext-message.xml

</param-value>

</context-param> 这个用来声明spring的配置文件位置

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener> 加载Spring Web

需要servlet2.4以上版本支持。

获取Spring Web容器：

WebApplicationContextUtils.getRequiredWebApplicationContext(servletContext) 该方法会抛出异常，也应该抛出异常。

集成Struts2

Struts2通过ObjectFactory接口创建和获取Action的实例，与Spring集成的关键点在于如何创建ObjectFactory实现来从Spring容器中获取对应的Action Bean。

Struts2提供了一个StrutsSpringObjectFactory类，我们使用它作为中介，连结Struts2和Spring。

1、在Spring配置文件中声明Action Bean

<bean id=”helloAction” class=”…”>…</bean>

2、在Struts2配置文件中声明Action

<action name=”hello” class=”helloAction”>…</action> 红字部分是关键

3、在Struts2配置文件或者properties文件中配置

struts.objectFactory = org.apache.struts2.spring.StrutsSpringObjectFactory

**零配置**

<context:component-scan>会自动引入<context:annotation-config/>

**SpringMVC**

前端控制器(Front Controller)：为表现层提供统一访问点，相当于一个门面。可以为多个请求提供共用逻辑(如上下文)，将选择视图与视图的具体处理功能分离。

应用控制器(Application Controller)：前端控制器将选择视图和视图处理分离后，需要应用控制器来管理如何选择视图、如何进行功能处理。

页面控制器(Page Controller)：功能处理代码，收集参数，封装数据到模型。

上下文(Context)：以前我们都把模型数据放到Request中，现在可以放到上下文中，从而与Servlet剥离。一般使用ThreadLocal来实现。

对应到SpringMVC中，前端控制器是DispatcherServlet，应用控制器拆分为处理器映射器(Handler Mapping)和视图解析器(View Resolver)，页面控制器是Controller(接口或注解)。

SpringMVC支持本地化(Locale)解析，主题(Theme)解析，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。提供了强大的约定优于配置的契约式编程支持。

处理流程，参见本目录下SpringMVC核心架构。

**支持的注解**

@Controller

@RequestMapping

@RequestParam

@ModelAttribute

@SessionAttributes

@InitBind

**RESTful注解**

@CookieValue

@RequestHeader

@RequestBody

@ResponseBody

@ResponseStatus

@ExceptionHandler

@PathVariable

一些配置

<mvc:annotation-driven>，自动注册基于注解所需要的DefaultAnnotationHandlerMapping，AnnotationMethodHandlerAdaptor；支持ConversionService自动注册；支持JSR-303验证框架的自动探测和注册；自动注册相应的HttpMessageConverter(用于支持@RequestBody和@ResponseBody，如xml输入输出转换[只需将JAXB实现放到classpath下]，JSON输入输出转换[只需将Jackson放到classpath下])

<mvc:interceptors>，注册自定义的处理器拦截器。

<mvc:view-controller>，和ParameterizableViewController 类似，收到相应请求后直接选择相应的视图。

<mvc:resources>，逻辑静态资源路径到物理静态资源路径的支持。

<mvc:default-servlet-handler>，当在 web.xml 中 DispatcherServlet 使用<url-pattern>/</url-pattern> 映射时，这个配置中的Servlet能映射静态资源（当 Spring Web MVC 框架没有处理请求对应的控制器时[如一些静态资源]，转交给这个Servlet来响应静态文件，否则报404）

**Spring容器上下文**

ContextLoaderListener的参数里指定的配置文件不包含springmvc，它们都是springmvc的父容器。

DispatcherServlet初始化：①初始化Spring Web MVC的上下文容器，并指定父容器(如果有)②初始化DispatcherServlet使用的一些策略，如HandlerMapping、HandlerAdaptor等，这些默认策略配置在org.springframework.web.servlet.DispatcherServlet.properties文件中。

**HandlerInterceptor**

三个方法，preHandle处理前，如果返回false则不继续往后传递；postHandle处理后页面渲染前；afterCompletion页面渲染完毕后。注意Interceptor是单例的。