**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 定义一个Controller，推荐使用这种方式

@RequestMapping 定义URI路径映射，可以放在类和方法上，支持占位符、Ant风格和正则表达式。不推荐使用Ant风格。带有几个参数，method：可以为post、get等；params：请求参数中有或者没有某参数，如params=”[!]create”表示有[没有]create参数，甚至可以指定参数值，如params=”submitFlag=create”；headers：请求头部，其他和params一样；生产者消费者：produces和consumes分别表示方法输出和接受什么类型的ContentType；除了生产者和消费者是覆盖类上的注解，其他都是继承。

@RequestParam 绑定单个请求参数，通过get和post都可以获取到

@ModelAttribute 绑定参数到命令对象，并且放到Model中

@SessionAttributes 绑定命令对象到session

@InitBind 被注解的Controller独享的PropertyEditor

@Value 绑定spEL表达式

**RESTful注解**

@CookieValue 绑定cookie数据

@RequestHeader 绑定请求头数据

@RequestBody 绑定请求内容数据。与HttpMessageConverter配合使用。

@RequestPart 绑定”multipart-formdata”数据，可以是从表单提交，也可以直接客户端

@ResponseBody 和@RequestBody一样。

@ResponseStatus 注解在处理方法上，忽略方法返回值，直接将HTTP状态码和信息返回给客户端

@ExceptionHandler Controller级别的异常处理

@PathVariable 绑定URI路径中的变量

HttpEntity<?>、ResponseEntity<?>和@RequestBody类似，但增加了对请求头、响应头的访问。

**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。

**可用的输入输出参数**：

输入：WebRequest/NativeRequest、ServletRequest/HttpServletRequest、MultipartRequest

HttpSession、Locale、Principal、InputStream/Reader、SessionStatus

输出：ServletResponse/HttpServletResponse、OutputStream/Writer、RedirectAttributes

UriComponentsBuilder、Map/ModelMap、命令/表单对象、Errors/BindingResult、HttpEntity

一些注意事项：

ModelAndView会覆盖Model中的同名参数；

不同于Struts2，SpringMVC无法绑定表单中的user.userName到user对象，即，一个页面只能绑定一个表单对象。

@RequestParam(“flag”) int[] flag，xxx?flag=1,2 会报错，将int[]改为String[]则不报错，这应该是bug。

Spring的表单标签库很难用，当前阶段应不予适用。

使用@Valid进行Bean Validate时，方法输入参数必须有BindingResult/Errors

ConversionService：

比PropertyEditor更通用

**Spring3.2 Reference**

001、<context:annotation-config/>自动注册AutowiredAnnotationBeanPostProcessor(支持@AutoWired注解，自动注入，找不到匹配的Bean时，啥都不干),

CommonAnnotationBeanPostProcessor(支持@Resource注解，作用和@AutoWired类似，但优先使用byName方式查找匹配；支持@PostConstruct和@PreDestroy注解，分别用于Bean初始化和销毁), PersistenceAnnotationBeanPostProcessor,

RequiredAnnotationBeanPostProcessor(支持@Required注解，自动注入，找不到匹配Bean时，抛出异常).注意：该配置只对它所在的ApplicationContext下的Bean起作用。

@Inject(JSR330定义)和@AutoWired作用一样。

@Qualifier，当有多个匹配的Bean时，用此注解唯一指定。

综上：<context:annotation-config/>的作用是支持Bean的属性注入。

002、<context:component-scan base-package="org.example"/>，扫描org.example包及其所有子包中的用 @Component、@Repository、@Service、@Controller注解的类并将其注册为Bean。它有子元素<context:exclude-filter/>和<context:include-filter/>用于排除和添加需要扫描的类。同时，该配置自动注册<context:annotation-config/>

综上：<context:component-scan/>的作用是自动扫描特定注解的类并将其注册成为Bean，同时支持注解形式的属性注入。

003、<context:property-placeholder location="classpath: jdbc.properties"/> 属性占位符，格式是${ }，从location指定的properties文件中获取资源。注意和messagesource(用于国际化)、SpEL的区别。

004、MessageSource，用于国际化。ApplicationContext实现了此接口，当ApplicationContext启动时，它会依次从自己、父容器中查找id/name为messagesource的Bean。我们一般使用ReloadableResourceBundleMessageSource这个实现类。

005、**校验/数据绑定/类型转换**

类型转换，spring3.0之前用的是PropertyEditor，注册自定义PropertyEditor如下：

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

<property name="customEditors">

<map>

<entry key="example.ExoticType" value="example.ExoticTypeEditor"/>

</map>

</property>

</bean>

3.0后，新增ConversionService，我们使用FormattingConversionServiceFactoryBean来注册，通过属性converters和formatters分别添加自定义类型转换。

**JSR303**校验启用，注册LocalValidatorFactoryBean，指定providerClass和validationMessageSource，分别表示Validator的实现类和messagesource。

@Controller级别的数据绑定 使用@InitBind注解一个方法，输入参数为WebDataBinder，为这个DataBinder指定validator和propertyEditor。

006、**SpEL**

ApplicationContext中可直接使用SpEL，如设置bean属性。使用的符号是#{ }，注意和PropertyPlaceHolderConfigurer的区别。systemProperties变量是预定义的，可以直接使用。如#{systemProperties[‘user.region’]}。@Value注解可以使用SpEL。

007、**AOP**

<aop:aspectj-autoproxy/>自动启用基于AspectJ的AOP。@Aspect注解不能被自动扫描到，还得需要@Component。

008、**事务**

事务策略定义在PlatformTransactionManager中，支持JDBC、JTA、Hibernate等。

数据库操作需要配置dataSource、txManager，txManager引用dataSource。如下：

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy- method="close">

<property name="driverClassName" value="${jdbc.driverClassName}" />

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

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

</bean>

<bean id="txManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

如果是JNDI和JTA事务，则：

<jee:jndi-lookup id="dataSource" jndi-name="jdbc/jpetstore"/>

<bean id="txManager" class="org.springframework.transaction.jta.JtaTransactionManager"/>

txManager中不需要配置dataSource，由容器负责。

基于注解的声明式事务：

<tx:annotation-driven transaction-manager="txManager"/>

@Transactional，默认配置：事务传播PROPAGATION\_REQUIRED，隔离级别ISOLATION\_DEFAULT，事务级别：Read/Write，超时：看底层，回滚：RuntimeException回滚，受检异常不回滚。

**009、OXM/Object Xml Mapping**

提供一致性接口Marshaller(Object to XML)和UnMarshaller(XML to Object)，具体实现可使用JAXB、JIBX、Castor、XmlBeans或其他类库。

**010、SpringMVC**

ViewResolver：forward和redirect，方法返回值类型为String，返回”forword:/web/reg”，第一个/有没有都是根目录。

ContentNegotiatingViewResolver：两种策略：一、根据后缀，如a.jsp，b.pdf，c.xml；二、根据Accept Header，如text/xml，application/pdf。

Locale：提供AcceptHeader、Cookie、Session等策略改变Locale。

文件上传：以commons-fileupload为例，需声明一个CommonsMultipartResolver。在处理方法上，使用MultipartFile类型来接收上传的文件。需要注意的是，当上传文件超过大小时，会抛出异常，需要自行处理。如想使用Controller上的@ExceptionHandler处理该异常，则需要把CommonsMultipartResolver的resolveLazily属性设为true，表示到解析时才抛出异常，这样才能进入Controller，从而被@ExceptionHandler捕获。

异常处理的三种方法：①、配置SimpleMappingExceptionResolver；②自行实现HandlerExceptionResolver接口或者继承其实现类，比如ExceptionHandlerExceptionResolver；③Controller中的@ExceptionHandler。捕获的时候先在①或②中查找匹配，找不到或处理不了再进入到③，再处理不了，抛出异常。对于返回状态码的异常，只能通过web.xml中的error-page元素来处理。

MVC命名空间：

< mvc:annotation-driven />，默认注册了RequestMappingHandlerMapping，RequestMapping -HandlerAdapter，ExceptionHandlerExceptionResolver，用以支持MVC下的各种注解。同时，开启了以下功能：

ConversionService，包括Converter和Formatter，如@Number、@DateTimeFormat

JSR303，如@Valid

HttpMessageConverter，注册了：ByteArray[HttpMessageConverter]，String\*，Resource\*(Spring的Resource接口)，Source\*( javax.xml.transform.Source，用于生成XML)，Form\*(将表单数据转换成一个MultiValueMap<String,String>)，Jaxb2RootElement\*(OXM，需要Jaxb2的支持)，MappingJackson2\*(JSON，需要Jackson支持)，AtomFeed\*(Atom Feed，需要Rome支持)，RssChannel\*(Rss Feed，也需要Rome支持)。

ContentNegotiation：后缀优先，再AcceptHeader。如果classpath下有相应jar包，如下类型会自动注册：.json，.xml，.atom，.rss，其他类型最好显式注册。

处理Excel和PDF：前者需要用到POI库，后者需要用到iText库。实现相应的View(可以简单继承AbstractExcelView和AbstractPdfView)就可以了，ViewResolver会使用InternalResourceViewResolver来解析。

**011、远程调用和Webservice**

RMI(Remoting Method Invocation)，Spring’s HTTP Invoker(Spring自己的实现)，Hessian(基于二进制协议)，Burlap(Hessian的XML变种)，JAX-RPC(JEE1.4的Webservice API)，JAX-WS(JEE5以上，JAX-RPC的继任者)，JMS(Java Message Service)。

**012、JMS**

JmsTemplate：和JdbcTemplate类似，线程安全，通过回调操作。自行释放资源。默认的knowledge是PTP。

标准的发送、接收消息流程，都要消费临时的ConnectionFactory、Connection、Session对象等，如果能将这些对象加以缓存，性能会好不少。

因此：SingleConnectionFactory和CachingConnectionFactory应运而生。前者每次都返回同一个Connection，忽略close()方法。CachingConnectionFactory缓存的对象更多。

Destination管理：和ConnectionFactory一样，可以作为JNDI资源来管理。也可以通过DestinationResolver来创建。

MessageListenerContainer，异步接收消息的Listener容器。注册消息消费者，参与事务管理，资源获取和释放，异常转换等等。SimpleMessageListenerContainer一般用于独立应用程序，DefaultMessageListenerContainer用于J2EE环境，支持XA事务和动态Destination。

事务处理：JmsTransactionManager，和Spring的其他事务管理模型保持一致。

MessageConverter：用于Java Object和JMS Message之间的转换。

接收消息：同步接收和异步接收。异步时，要保证MessageListener的线程安全性。Spring提供了更强大的类MessageListenerAdapter来将任意一个POJO适配成MessageListener。注意一个MessageListenerContainer只能注册一个MessageListener，因此当有多个Listener时，需要配置多个Container。

**013、JMX**

四大功能：

将任意spring bean自动注册为MBean；

为bean的管理接口提供灵活的管理机制；(窄化暴露的方法和属性)

声明式的服务暴露，支持远程和JSR160；

简化本地和远程MBean资源代理；

通过MBeanExporter将任意bean暴露为MBean，注意MBeanExporter不能延迟加载，即lazy-init必须为false。

**014、Task执行、计划**

两大接口：TaskExecutor和TaskScheduler，支持线程池，屏蔽了Java5、Java6和J2EE之间的差异。通过FactoryBean集成Quartz和JDK Timer。

SimpleAsyncTaskExecutor，没有线程池功能，每次都创建一个新线程；SyncTaskExecutor，仅用于测试；ConcurrentTaskExecutor，对java.util.concurrent.Executor的包装，一般也不使用，优先使用**ThreadPoolTaskExecutor**；SimpleThreadPoolTaskExecutor，是SimpleThreadPool的子类，使用场景，为Quartz和非Quartz组件提供线程池；TimerTaskExecutor，顾名思义，后台为TimerTask，对JDK Timer提供支持；WorkManagerTaskExecutor，对CommonJ提供支持。

注意：TaskExecutor是随着Spring容器关闭而关闭的，如果Spring容器关闭时前者没有一起关闭则有问题。

TaskScheduler在3.0后加入。它最重要的方法是：

ScheduledFuture schedule(Runnable task, Trigger trigger); Trigger接口本身非常简单：

Date nextExecutionTime(TriggerContext triggerContext);得到下一次执行时间即可。

TriggerContext记录了上一次执行的时间点信息，有最近计划执行时间，最近实际执行时间和最近执行完成时间。最有用的Trigger是Quartz提供的CronTrigger，可以参考其语法。

TaskScheduler最常用的实现是**ThreadPoolTaskScheduler**，它实现了TaskExecutor接口，同时代理了ScheduledExecutorService，因此它可以用来做异步的任务调度。

注解支持异步调用和任务调度：@Scheduled、@Async，在application中的命名空间配置为：<task:annotation-driven executor="x" scheduler="x"/>，executor和@Async对应，scheduler和@ Scheduled对应。

父子容器

问题：第11章测试，配合陈雄华那本书看看；集成MyBatis；

page 708

**一些问题**

Controller级别的Converter：这个没有实现方法，倒是有该级别的ConversionService，但相比PropertyEditor太过麻烦。因此还是用PropertyEditor比较好。

JSR303验证：先注册一个名为validator的LocalValidatorFactoryBean，将其注入到一个名为webBindingInitializer的ConfigurableWebBindingInitializer，再将webBindingInitializer注入到HandlerAdapter。

Formatter：全局配置在ConversionService中，参见005。Controller级别的最好不用。字段级别的用注解，比如@DateTimeFormat、@NumberFormat等。在页面上，需要使用bind标签展示，例如<sp:bind path="user.birthday">${status.value}</sp:bind>。注意当没有user这个变量时会报错。

forward、redirect共享变量：forward时，目前只能使用HttpServletRequest，不考虑Session；redirect时，可以使用RedirectAttributes，对于对象类型，添加到FlashAttribute，对于字符串变量，添加到Attribute，该变量会附加到URL参数中。

静态资源处理：spring提供了<mvc:resoures/>配置来定位静态资源，但该方法的执行效率肯定不及Web容器自身。因此还是将静态资源放在WebContent目录下，动态资源才放在WebContent/Web-INF下，此时需要开启<mvc:default-servlet-handler />来处理。